MCUXpresso SDK Documentation

Release 26.09.00

NXP
Sep 29, 2026

Table of contents

1 MCXW72-LOC **3**
- 1.1 Overview . . . 3
- 1.2 SDK Introduction . . . 3
  - 1.2.1 MCUXpresso Software Development Kit (SDK) . . . 3
- 1.3 Getting Started with Repository-Layout SDK Package . . . 7
  - 1.3.1 Getting Started with Repository-Layout SDK Package . . . 7
- 1.4 Getting Started with MCUXpresso SDK GitHub . . . 26
  - 1.4.1 Getting Started with MCUXpresso SDK Repository . . . 26
- 1.5 Release Notes . . . 39
  - 1.5.1 MCUXpresso SDK Release Notes . . . 39
- 1.6 ChangeLog . . . 50
  - 1.6.1 MCUXpresso SDK Changelog . . . 50
- 1.7 Driver API Reference Manual . . . 153
- 1.8 Middleware Documentation . . . 153
  - 1.8.1 Wireless Bluetooth LE host stack and applications . . . 153
  - 1.8.2 Wireless Connectivity Framework . . . 153
  - 1.8.3 Multicore . . . 154
  - 1.8.4 FreeMASTER . . . 154
  - 1.8.5 FreeRTOS . . . 154

2 MCXW727C **155**
- 2.1 CACHE: LPCAC CACHE Memory Controller . . . 155
- 2.2 CCM32K: 32kHz Clock Control Module . . . 156
- 2.3 CE Basic Functions . . . 167
- 2.4 CE Command Functions . . . 168
- 2.5 CE CMSIS Functions . . . 169
- 2.6 CE Initialization Functions . . . 170
- 2.7 CE Matrix Functions . . . 170
- 2.8 CE Transform Functions . . . 175
- 2.9 Clock Driver . . . 177
- 2.10 CMC: Core Mode Controller Driver . . . 198
- 2.11 CRC: Cyclic Redundancy Check Driver . . . 211
- 2.12 EDMA: Enhanced Direct Memory Access (eDMA) Controller Driver . . . 214
- 2.13 ELEMU: Edgelock Messaging unit driver . . . 236
- 2.14 EWM: External Watchdog Monitor Driver . . . 240
- 2.15 FGPIO Driver . . . 244
- 2.16 C40ESP3 Flash Driver . . . 244
- 2.17 FlexCAN: Flex Controller Area Network Driver . . . 249
- 2.18 FlexCAN Driver . . . 249
- 2.19 FlexCAN eDMA Driver . . . 305
- 2.20 FlexIO: FlexIO Driver . . . 308
- 2.21 FlexIO Driver . . . 308
- 2.22 FlexIO eDMA SPI Driver . . . 327
- 2.23 FlexIO eDMA UART Driver . . . 331
- 2.24 FlexIO I2C Master Driver . . . 334
- 2.25 FlexIO I2S Driver . . . 343

2.26 FlexIO SPI Driver . . . 355
2.27 FlexIO UART Driver . . . 369
2.28 GPIO: General-Purpose Input/Output Driver . . . 381
2.29 GPIO Driver . . . 384
2.30 I3C: I3C Driver . . . 390
2.31 I3C Common Driver . . . 393
2.32 I3C Master Driver . . . 396
2.33 I3C Slave Driver . . . 427
2.34 IMU: Inter CPU Messaging Unit . . . 441
2.35 Common Driver . . . 448
2.36 LPADC: 12-bit SAR Analog-to-Digital Converter Driver . . . 464
2.37 LPCMP: Low Power Analog Comparator Driver . . . 486
2.38 LPI2C: Low Power Inter-Integrated Circuit Driver . . . 498
2.39 LPI2C Master Driver . . . 499
2.40 LPI2C Master DMA Driver . . . 515
2.41 LPI2C Slave Driver . . . 517
2.42 LPIT: Low-Power Interrupt Timer . . . 529
2.43 LPSPI: Low Power Serial Peripheral Interface . . . 536
2.44 LPSPI Peripheral driver . . . 536
2.45 LPSPI eDMA Driver . . . 560
2.46 LPTMR: Low-Power Timer . . . 568
2.47 LPUART: Low Power Universal Asynchronous Receiver/Transmitter Driver . . . 575
2.48 LPUART Driver . . . 575
2.49 LPUART eDMA Driver . . . 596
2.50 LTC: LP Trusted Cryptography . . . 599
2.51 LTC AES driver . . . 601
2.52 LTC DES driver . . . 606
2.53 LTC HASH driver . . . 614
2.54 LTC PKHA driver . . . 616
2.55 LTC Blocking APIs . . . 625
2.56 MCM: Miscellaneous Control Module . . . 625
2.57 MSCM: Miscellaneous System Control . . . 631
2.58 MU: Messaging Unit . . . 631
2.59 PORT: Port Control and Interrupts . . . 645
2.60 RTC: Real Time Clock . . . 654
2.61 SEMA42: Hardware Semaphores Driver . . . 662
2.62 SFA: Signal Frequency Analyser . . . 666
2.63 SMSCM: Secure Miscellaneous System Control Module . . . 676
2.64 SPC: System Power Control driver . . . 681
2.65 SYSPM: System Performance Monitor . . . 722
2.66 TPM: Timer PWM Module . . . 726
2.67 TRDC: Trusted Resource Domain Controller . . . 744
2.68 TRGMUX: Trigger Mux Driver . . . 768
2.69 VBAT: Smart Power Switch . . . 770
2.70 VREF: Voltage Reference Driver . . . 778
2.71 WDOG32: 32-bit Watchdog Timer . . . 781
2.72 WUU: Wakeup Unit driver . . . 788

3 Middleware **795**
3.1 Motor Control . . . 795
3.1.1 FreeMASTER . . . 795
3.2 Multicore SDK . . . 833
3.2.1 Quick links . . . 833
3.2.2 Components . . . 834
3.3 Wireless . . . 923
3.3.1 NXP Wireless Framework and Stacks . . . 923

4 RTOS **1803**

4.1 FreeRTOS . . . 1803
4.1.1 FreeRTOS kernel . . . 1803
4.1.2 FreeRTOS drivers . . . 1807
4.1.3 backoffalgorithm . . . 1807
4.1.4 corehttp . . . 1807
4.1.5 corejson . . . 1807
4.1.6 coremqtt . . . 1807
4.1.7 corepkcs11 . . . 1808
4.1.8 freertos-plus-tcp . . . 1808

5 Downloads **1809**
5.1 boards . . . 1809
5.2 develop . . . 1809

Index **1811**

This documentation contains information specific to the mcxw72loc board.

Chapter 1

MCXW72-LOC

1.1 Overview

The MCXW72 LOC is an IIOT evaluation kit development board for advanced development of the MCXW72 wireless MCU. It offers exhaustive evaluation of the MCX W72's multiprotocol wireless support for Bluetooth LE, Zigbee, Thread and Matter and CAN connectivity. The MCXW72's highly sensitive, optimized 2.4 GHz radio features is featured with dual antenna diversity via RF switch which can be bypassed to test via SMA RF connection. This localization platform is dedicated to Bluetooth Channel Sounding Ranging solution development with a dedicated on-chip Localization Compute Engine to reduce ranging latency. The board includes an advanced MCU-Link debug probe, Power and low power option, dual CAN transceiver, buttons, switches, LEDs and a MikroE Click connector.

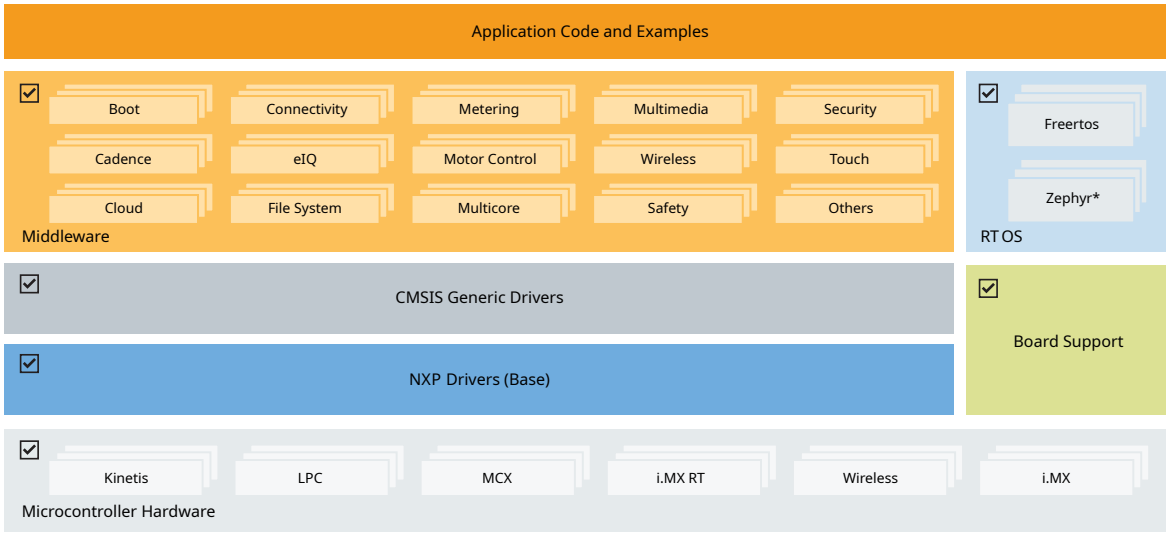
MCU device and part on board is shown below:

- Device: MCXW727C
- PartNumber: MCXW727CMFTA

1.2 SDK Introduction

1.2.1 MCUXpresso Software Development Kit (SDK)

The MCUXpresso SDK is a comprehensive software enablement package designed to simplify and accelerate application development with Arm® Cortex®-M based devices from NXP, including its general purpose, crossover and wireless-enabled MCUs. It includes production-grade software with integrated enabling software technologies (stacks and middleware), reference examples, and more.



*Zephyr RTOS leverages base SDK components via NXP Zepher HAL

Features

Comprehensive Driver Library

- Flexible peripheral drivers for rapid development
- Optimized for performance and code size

RTOS Integration

- FreeRTOS support with optimized configurations
- Real-time application frameworks
- Task management and scheduling examples

Middleware Components

- Connectivity stacks (USB, Ethernet, wireless, communication protocols)
- AI inference engines and neural network libraries
- Audio processing and voice recognition libraries
- Security frameworks and cryptographic libraries
- Graphics and user interface components
- Motor control and real-time control libraries
- File systems and storage solutions
- Partner middleware can be added using CMake and west manifests

Extensive Examples

- Basic peripheral use cases to full technology demonstrations
- Board-specific reference implementations

- Out-of-box Demo Applications
- Middleware component examples and integrations

Distribution Formats

MCUXpresso SDK is available in two main formats:

GitHub Repository SDK Git-based distribution hosted on GitHub for modular development, continuous updates, and collaborative workflows. The SDK uses multiple repositories managed by the Zephyr west tool to provide flexible, component-based development. See Getting Started for setup instructions.

SDK Package Downloadable packages from SDK Builder for offline development. Starting with release 25.12.00, two package versions are available:

- **Classic SDK Package**: Traditional board-specific packages with pre-configured IDE projects for MCUXpresso IDE, IAR, Keil, and other toolchains.
- **Repository-Layout SDK Package**: Board-specific packages that maintain the same structure and build system as the GitHub Repository SDK, providing offline access to the repository-based development experience. Available when selecting the ARMGCC toolchain.

Note: From version 25.12.00 onward, selecting ARMGCC downloads the Repository-Layout version; all other toolchains remain in the Classic version.

Delivery and Release

NXP integrates Coverity® Static Analysis and Black Duck Software Composition Analysis into the SDK development process to deliver secure, high-quality software.

- **Annual releases (June)** for popular products, with Long Term Support (LTS) on GitHub — back-ported security (SVE/PSIRT) and severe bug fixes for two years.
- **Quarterly releases** for NPI and recent products.
- SBOMs provided for all GitHub SDK repositories.

Release	Release Name	Maintenance
LTS	26.06.00-LTS	2 years until June 2028
Quarterly	26.09.00	None, rebase to mainline
LTS	27.06.00-LTS	2 years until June 2029

Supported platforms are listed at mcuxpresso.nxp.com/sdk-versions.

Toolchain Support

All driver examples and most application examples are provided for the following toolchains:

Toolchain	GitHub	Archive/Zip	Generated (west)	Getting Started
MCUXpresso IDE	□	□ eclipse	□	Link
MCUXpresso for VS Code	□	□ CMake	N.A.	*Link*
IAR Embedded Workbench	□	□ EWARM	□ EWARM	Link
Keil™ MDK-Arm	□	□ uVision	□ uVision	Link
CLI GNU toolchain	□	□ CMake	N.A.	*Link*

Archive/Zip packages can be downloaded from SDK Builder or imported directly in VS Code and MCUXpresso IDE using the Import Wizard.

IAR and Keil projects can be generated from the GitHub SDK CMake examples using a west build command.

Repository Architecture

The SDK uses a multi-repository structure on GitHub managed by the Zephyr West Tool, enabling:

- **Modular Organization**: Drivers, RTOS, middleware, and examples in separate repositories
- **Customizable Manifests**: Targeted setups for specific requirements
- **Selective Downloads**: Include only needed components
- **Independent Updates**: Component-level version control
- **Scalable Integration**: Easy addition of new repository components

Support and Resources

Community Support

- Open GitHub issues and discussions for questions and contributions
- MCUXpresso SDK Community Forum
- MCUXpresso Software and Tools Forum

Documentation

- Comprehensive guides and API documentation online

Technical Support

- NXP Support for commercial development assistance

SDK Product Page

- MCUXpresso SDK on NXP.com

Trainings

- MCUXpresso Training Hub
- MCUXpresso SDK Curriculum

1.3 Getting Started with Repository-Layout SDK Package

1.3.1 Getting Started with Repository-Layout SDK Package

Starting with version 25.09.00, MCUXpresso SDK introduced the Repository-Layout SDK Package for offline development:

- **Repository-Layout SDK Package**: A board-specific package that preserves the same structure and build system as the GitHub Repository SDK and provides offline access to the Repository SDK development experience. This package is available when you select the ARMGCC toolchain.

Starting with version 25.12.00, the SDK Builder downloads the Repository-Layout SDK Package by default when you select ARMGCC.

Note: MCUXpresso SDK first introduced the Repository-Layout SDK Package in version 25.09.00, but only for MCXW23x platforms.

Development Tools Installation

This guide explains how to install the essential tools for development with the MCUXpresso SDK.

Quick Start: Automated Installation (Recommended) The **MCUXpresso Installer** is the fastest way to get started. It automatically installs all the basic tools you need.

1. **Download the MCUXpresso Installer** from: MCUXpresso Installer
2. **Run the installer** and select **"MCUXpresso SDK Developer"** from the menu
3. **Click Install** and let it handle everything automatically

Manual Installation If you prefer to install tools manually or need specific versions, follow these steps:

Essential Tools

Git - Version Control

What it does: Manages code versions and downloads SDK repositories from GitHub.

Installation:

- Visit git-scm.com
- Download for your operating system
- Run installer with default settings
- **Important**: Make sure "Add Git to PATH" is selected during installation

Setup:

```
git config --global user.name "Your Name"
git config --global user.email "youremail@example.com"
```

Python - Scripting Environment

What it does: Runs build scripts and SDK tools.

Installation:

- Install Python **3.10 or newer** from python.org
- **Important**: Check "Add Python to PATH" during installation

West - SDK Management Tool

What it does: Manages SDK repositories and provides build commands. The west tool is developed by the Zephyr project for managing multiple repositories.

Installation:

```
pip install -U west
```

Minimum version: 1.2.0 or newer

Build System Tools

CMake - Build Configuration

What it does: Configures how your projects are built.

Recommended version: 3.30.0 or newer

Installation:

- **Windows**: Download `.msi` installer from cmake.org/download
- **Linux**: Use package manager or download from cmake.org
- **macOS**: Use Homebrew (`brew install cmake`) or download from cmake.org

Ninja - Fast Build System

What it does: Compiles your code quickly.

Minimum version: 1.12.1 or newer

Installation:

- **Windows**: Usually included, or download from ninja-build.org
- **Linux**: `sudo apt install ninja-build` or download binary
- **macOS**: `brew install ninja` or download binary

Ruby - IDE Project Generation (Optional)

What it does: Generates project files for IDEs like IAR and Keil.

When needed: Only if you want to use traditional IDEs instead of VS Code.

Installation: Follow the Ruby environment setup guide

Compiler Toolchains

Choose and install the compiler toolchain you want to use:

Toolchain	Best For	Download Link	Environment Variable
Arm GCC (Recommended)	Most users, free	Arm GNU Toolchain	ARMGCC_DIR
IAR EWARM	Professional development	IAR Systems	IAR_DIR
Keil MDK	Arm ecosystem	Arm Developer	MDK_DIR
Arm Compiler	Advanced optimization	Arm Developer	ARMCLANG_DIR
Code Warrior	DSC platforms	NXP CodeWarrior	CW_DIR
Xtensa	DSP core	Tensilica Tools	XCC_DIR

Setting Up Environment Variables

After toolchain installation, set an environment variable so the build system locates it:

Windows:

```
# Example for ARM GCC installed in C:\armgcc
setx ARMGCC_DIR "C:\armgcc"
```

Linux/macOS:

```
# Add to ~/.bashrc or ~/.zshrc
export ARMGCC_DIR="/usr"  # or your installation path
```

Verify Your Installation After installation, verify everything works by opening a terminal/command prompt and running these commands:

```
# Check each tool - you should see version numbers
git --version
python --version
west --version
cmake --version
ninja --version
arm-none-eabi-gcc --version  # (if using ARM GCC)
```

Troubleshooting Installation Issues

"Command not found" errors:

- The tool isn't in your system PATH
- **Solution**: Add the installation directory to your PATH environment variable

Python/pip issues:

- Try using `python3` and `pip3` instead of `python` and `pip`
- On Windows, run the Command Prompt as an Administrator

Slow downloads:

- Add timeout option: `pip install -U west --default-timeout=1000`
- Use alternative mirror: `pip install -U west -i https://pypi.tuna.tsinghua.edu.cn/simple`

Workspace Structure

After you initialize your SDK workspace, it creates a specific directory structure that organizes all SDK components. This structure is identical for both GitHub Repository SDK and Repository-Layout SDK Package.

Top-Level Organization

```
your-sdk-workspace/
  manifests/        # West manifest repository
  mcuxsdk/          # Main SDK content
```

The mcuxsdk/ directory serves as your primary working directory and contains all the SDK components.

SDK Component Layout Based on the actual SDK structure, the main directories include:

Directory	Contents	Purpose
arch/	Architecture-specific files	ARM CMSIS, build configurations
cmake	Build system modules	CMake configuration and build rules
compo	Software components	Reusable software libraries and utilities
devices	Device support packages	MCU-specific headers, startup code, linker scripts
drivers	Peripheral drivers	Hardware abstraction layer for MCU peripherals
examp	Sample applications	Demonstration code and reference implementations
middle	Optional software stacks	Networking, graphics, security, and other libraries
rtos/	Operating system support	FreeRTOS integration
scripts	Build and utility scripts	West extensions and development tools
svd	Svd files for devices, this is optional because of large size. Customers run west manifest config group.filter +optional and west update mcux-soc-svd to get this folder.	

Example Organization Examples follow a two-tier structure separating common code from board-specific implementations:

Common Example Files

```
examples/demo_apps/hello_world/
  CMakeLists.txt        # Build configuration
  example.yml           # Example metadata
```

(continues on next page)

(continued from previous page)

```
    hello_world.c          # Application source code
    Kconfig                # Configuration options
    readme.md              # General documentation
```

Board-Specific Files

```
examples/_boards/your_board/demo_apps/hello_world/
    app.h                      # Board specific application header
    example_board_readme.md    # Board specific documentation
    hardware_init.c            # Board specific hardware initialization
    pin_mux.c                  # Pin multiplexing configuration
    pin_mux.h                  # Pin multiplexing header definitions
    hello_world.bin            # Pre-built binary for quick testing
    hello_world.mex            # MCUXpresso Config Tools project file
    prj.conf                   # Board specific Kconfig configuration
    reconfig.cmake             # Board specific cmake configuration overrides
```

Device Support Structure Device support is organized hierarchically by MCU family:

```
devices/
    MCX/                   # MCU portfolio
        MCXW/              # MCU family
            MCXW235/       # Specific device
                MCXW235.h          # Device register definitions
                drivers/           # Device-specific drivers
                gcc/               # GNU toolchain files
                iar/               # IAR toolchain files
                mcuxpresso/        # MCUXpresso IDE files
                startup_MCXW235.c # Startup and vector table
                system_MCXW235.c  # System initialization
```

Middleware Organization Middleware components are categorized by functionality and maintained in separate repositories. Based on the manifest files, common middleware categories include:

- **Connectivity**: USB, TCP/IP, industrial protocols
- **Security**: Cryptographic libraries, secure boot
- **Wireless**: Bluetooth, IEEE 802.15.4, Wi-Fi
- **Graphics**: Display drivers, UI frameworks
- **Audio**: Processing libraries, voice recognition
- **Machine Learning**: Inference engines, neural networks
- **Safety**: IEC60730B safety libraries
- **Motor Control**: Motor control and real-time control libraries

Documentation Structure SDK documentation is distributed across multiple locations:

- `docs/` - Core SDK documentation and build infrastructure
- Component repositories - API documentation and integration guides
- Board directories - Hardware-specific setup instructions

For complete documentation, refer to the online documentation.

Understanding Example Structure Each example has **two README files**:

1. General README: examples/demo_apps/hello_world/readme.md

- What the example does
- General functionality description
- Common usage information

2. Board-Specific README: examples/_boards/{board_name}/demo_apps/hello_world/example_board_readme.md

- Board-specific setup instructions
- Hardware connections required
- Board-specific behavior notes

Tip: Always check both readme files - start with the general one, then read the board-specific one for detailed setup.

Building Your First Project

This guide explains how to build and run your first SDK example project using the west build system. This applies to both GitHub Repository SDK and Repository-Layout SDK Package.

Prerequisites

- GitHub Repository SDK workspace initialized OR Repository-Layout SDK Package extracted
- Development board connected via USB
- Build tools installed per *Installation Guide*

Understanding Board Support Before building, check whether your board is single-core or multicore, and which core IDs it supports:

```
west boards -b your_board
```

This prints the resolved device and, for multicore boards, the list of valid core_ids you'll need for the build command below. For a single-core board, no core ID is needed.

If you want to see every supported build configuration (toolchain/config combination) for a specific example instead, use:

```
west list_project -p examples/demo_apps/hello_world
```

This shows all supported build configurations. You can filter by toolchain:

```
west list_project -p examples/demo_apps/hello_world -t armgcc
```

west list_project is the most complete way to discover build commands across all boards and targets, but west boards -b your_board is the quicker way to look up just the board/core information you need once you already know which example you want to build.

Basic Build Process Simple Build

Build the hello_world example with default settings:

```
west build -b your_board examples/demo_apps/hello_world
```

The default toolchain is armgcc, and the build system will select the first debug target as default if no config is specified.

Specifying Configuration

```
# Release build
west build -b your_board examples/demo_apps/hello_world --config release

# Debug build (default)
west build -b your_board examples/demo_apps/hello_world --config debug
```

Alternative Toolchains

```
# IAR toolchain
west build -b your_board examples/demo_apps/hello_world --toolchain iar

# Other toolchains as supported by the example
```

Multicore Applications For multicore devices, west boards -b your_board (above) tells you which core IDs are valid. Specify the core either with -Dcore_id=:

```
west build -b evkbmimxrt1170 examples/demo_apps/hello_world --toolchain iar -Dcore_id=cm7 --config␣
↪flexspi_nor_debug
```

or more concisely by appending @<core_id> to the board name:

```
west build -b evkbmimxrt1170@cm7 examples/demo_apps/hello_world --toolchain iar --config flexspi_nor_
↪debug
```

Single-core boards don't need a core ID at all.

For multicore projects using sysbuild:

```
west build -b evkbmimxrt1170 --sysbuild ./examples/multicore_examples/hello_world/primary -Dcore_
↪id=cm7 --config flexspi_nor_debug --toolchain=armgcc -p always
```

Flash an Application Flash the built application to your board:

```
west flash -r linkserver
```

Debug Start a debug session:

```
west debug -r linkserver
```

Common Build Options Clean Build

Force a complete rebuild:

```
west build -b your_board examples/demo_apps/hello_world -p always
```

Dry Run

See the commands that get executed without running them:

```
west build -b your_board examples/demo_apps/hello_world --dry-run
```

Device Variants

For boards supporting multiple device variants:

```
west build -b your_board examples/demo_apps/hello_world --device DEVICE_PART_NUMBER --config␣
↪release
```

Project Configuration CMake Configuration Only

Run configuration without building:

```
west build -b your_board examples/demo_apps/hello_world -Dcore_id=cm7 --cmake-only -p
```

Interactive Configuration

Launch the configuration GUI:

```
west build -t guiconfig
```

Troubleshooting Build Failures

Use pristine builds to resolve dependency issues:

```
west build -b your_board examples/demo_apps/hello_world -p always
```

Getting Help

View the help information for west build:

```
west build -h
```

Check Supported Configurations

To see available configuration options and board targets for an example, refer to the below command:

```
west list_project -p examples/demo_apps/hello_world
```

Next Steps

- Explore other examples in the SDK
- Learn about *Command Line Development* for advanced options
- Try *VS Code Development* for integrated development
- Refer *Workspace Structure* to understand the SDK layout

Run a Demo Using IAR

The Repository-Layout SDK Package does not ship pre-generated IDE projects. Instead, you generate an IAR Embedded Workbench project on demand from the CMake build, the same way you would from a GitHub Repository SDK checkout.

Prerequisite: Ruby IDE project generation is implemented in Ruby and is only needed if you want to use IAR instead of the command line or VS Code. See *Ruby - IDE Project Generation (Optional)* for setup instructions before continuing.

1. Generate the IAR project If this is a pristine build, specify board, example, toolchain and core on the command line:

```
west build -b evkbmimxrt1170 examples/demo_apps/hello_world --toolchain iar -Dcore_id=cm7 --config␣
↪flexspi_nor_debug -p always -t guiproject
```

If you already built the example with `west build`, you can regenerate the project with the shorter form:

```
west build -t guiproject
```

The project files are generated into `mcuxsdk/build/iar`, using relative paths to refer to source files and include paths in the repository. For the full set of `guiproject` options, see IDE Project Generation.

Generate into a custom folder

By default the project is generated into `mcuxsdk/build/<toolchain>`. To generate into a different location instead — for example to keep projects for several examples or boards side by side — pass `-d` with the pristine build command:

```
west build -b evkbmimxrt1170 examples/demo_apps/hello_world --toolchain iar -Dcore_id=cm7 --config␣
↪flexspi_nor_debug -p always -t guiproject -d build_hello_world_iar
```

2. Open the project in IAR Embedded Workbench Open the generated workspace or project file from:

```
mcuxsdk/build/iar
```

3. Build the demo application

1. Select the desired build target from the drop-down menu, for example **hello_world - flexspi_nor_debug**.
2. Click **Make** to build the demo application.
3. The build completes without errors.

4. Run the demo application

1. Open a terminal application on the PC, such as PuTTY or TeraTerm, and connect to the debug COM port (see How to determine COM port).
2. Click **Download and Debug** to flash and start debugging.
3. Click **Go** to run to the entry point and resume execution.
4. The terminal displays the demo application output.

Next steps

- Run a demo using MDK — the same flow for Keil MDK
- *Building Your First Project* — build and run from the command line

MCUXpresso for VS Code Development

This guide covers using MCUXpresso for VS Code extension to build, debug, and develop SDK applications with an integrated development environment.

Prerequisites

- SDK workspace initialized (GitHub Repository SDK or Repository-Layout SDK Package)
- Development tools installed per *Installation Guide*
- Visual Studio Code installed
- MCUXpresso for VS Code extension installed

Extension Installation Install MCUXpresso for VS Code

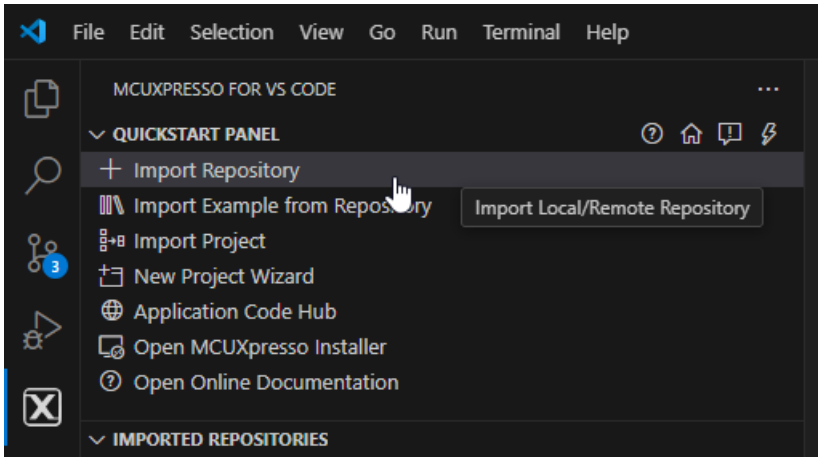
The MCUXpresso for VS Code extension provides integrated development capabilities for MCUXpresso SDK projects. Refer to the MCUXpresso for VS Code documentation for detailed installation and setup instructions.

SDK Import and Setup Import Methods

The SDK can be imported in several ways. The MCUXpresso for VS Code extension supports both GitHub Repository SDK and Repository-Layout SDK Package distributions.

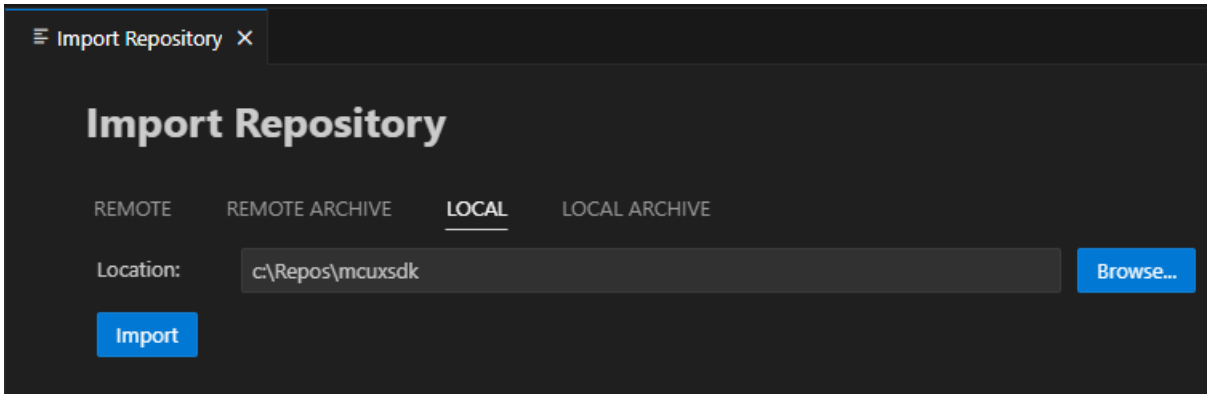
Import GitHub Repository SDK

Click **Import Repository** from the **QUICKSTART PANEL**

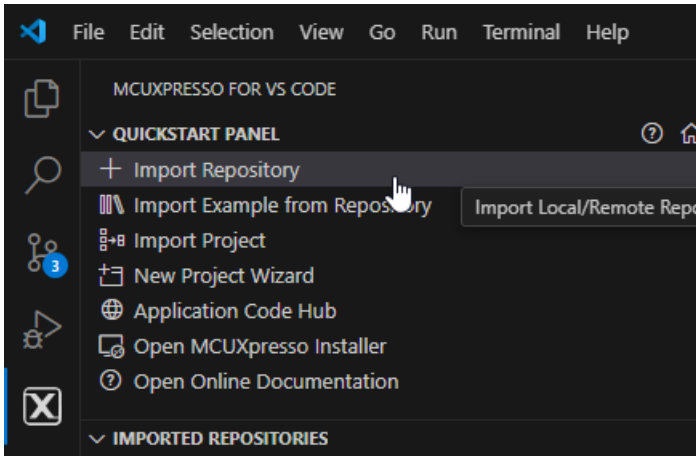


Note: You can import the SDK in several ways. Refer to the MCUXpresso for VS Code documentation for details.

Select **Local** if you've already obtained the SDK according to *setting up the repo*. Select your location and click **Import**.

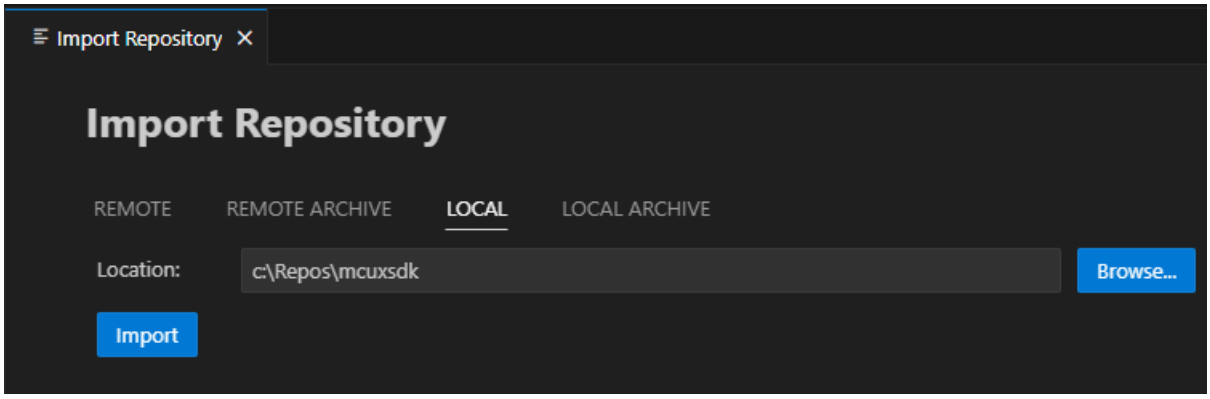


Import Repository-Layout SDK Package

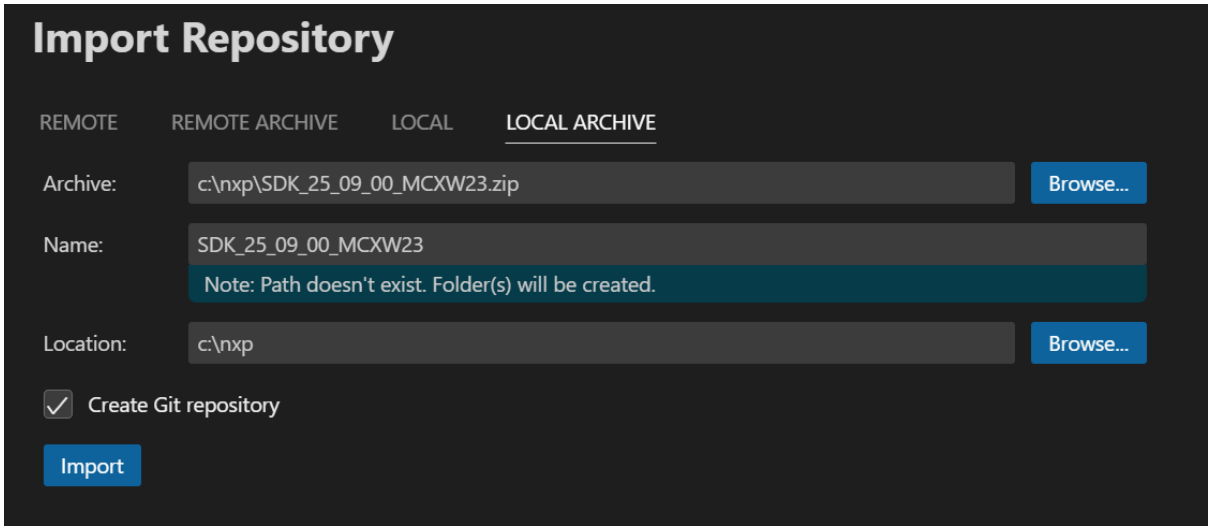


Click **Import Repository** from the **QUICKSTART PANEL**

Select **Local** if you've already unzipped the Repository-Layout SDK Package. Select your location and click **Import**.

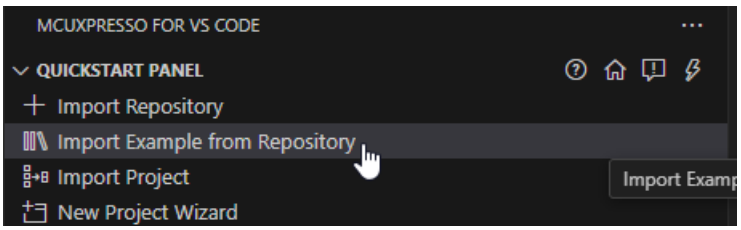


Else if the SDK is ZIP archive, select **Local Archive**, browse to the downloaded SDK ZIP file, fill the link of expect location, then click **Import**.

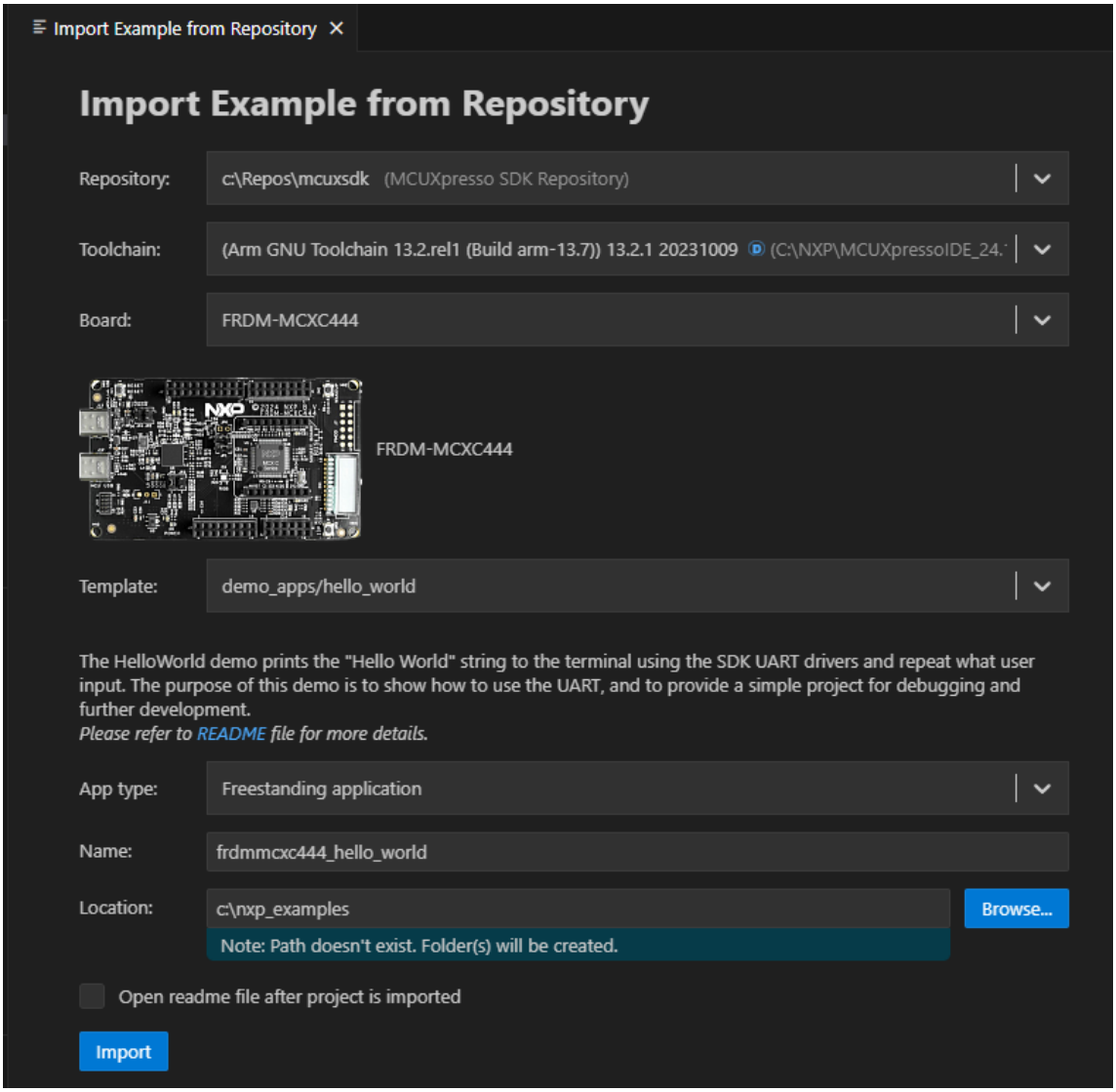


Building Example Applications Import Example Project

1. Click **Import Example from Repository** from the **QUICKSTART PANEL**



2. Configure project settings:
   - **MCUXpresso SDK**: Select your imported SDK
   - **Arm GNU Toolchain**: Choose toolchain
   - **Board**: Select your target development board
   - **Template**: Choose example category
   - **Application**: Select specific example (e.g., hello_world)
   - **App type**: Choose between Repository applications or Freestanding applications
3. Click **Import**



Application Types

Repository Applications:

- Located inside the MCUXpresso SDK
- Integrated with SDK workspace

Freestanding Applications:

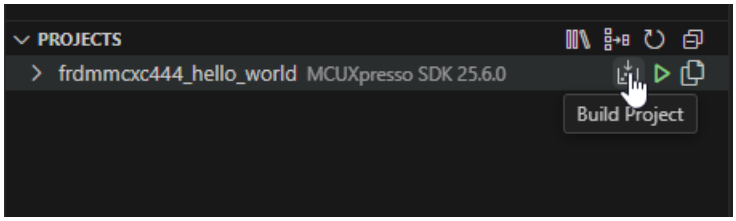
- Imported to user-defined location
- Independent of SDK location

Trust Confirmation

VS Code will prompt you to confirm if the imported files are trusted. Click **Yes** to proceed.

Building Projects Build Process

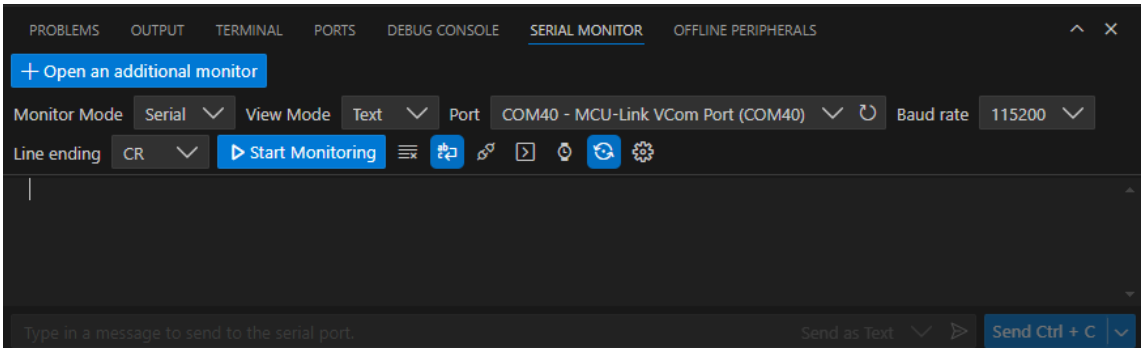
1. Navigate to **PROJECTS** view
2. Find your project
3. Click the **Build Project** icon



The integrated terminal will display build output at the bottom of the VS Code window.

Running and Debugging Serial Monitor Setup

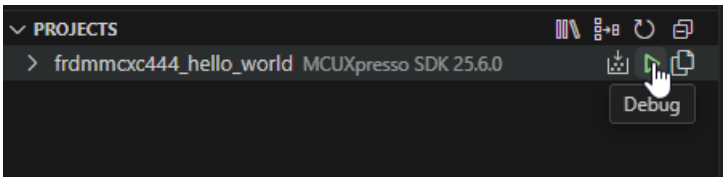
1. Open **Serial Monitor** from VS Code's integrated terminal



2. Configure serial settings:
   - **VCom Port**: Select port for your device
   - **Baud Rate**: Set to 115200

Debug Session

1. Navigate to **PROJECTS** view
2. Click the play button to initiate a debug session



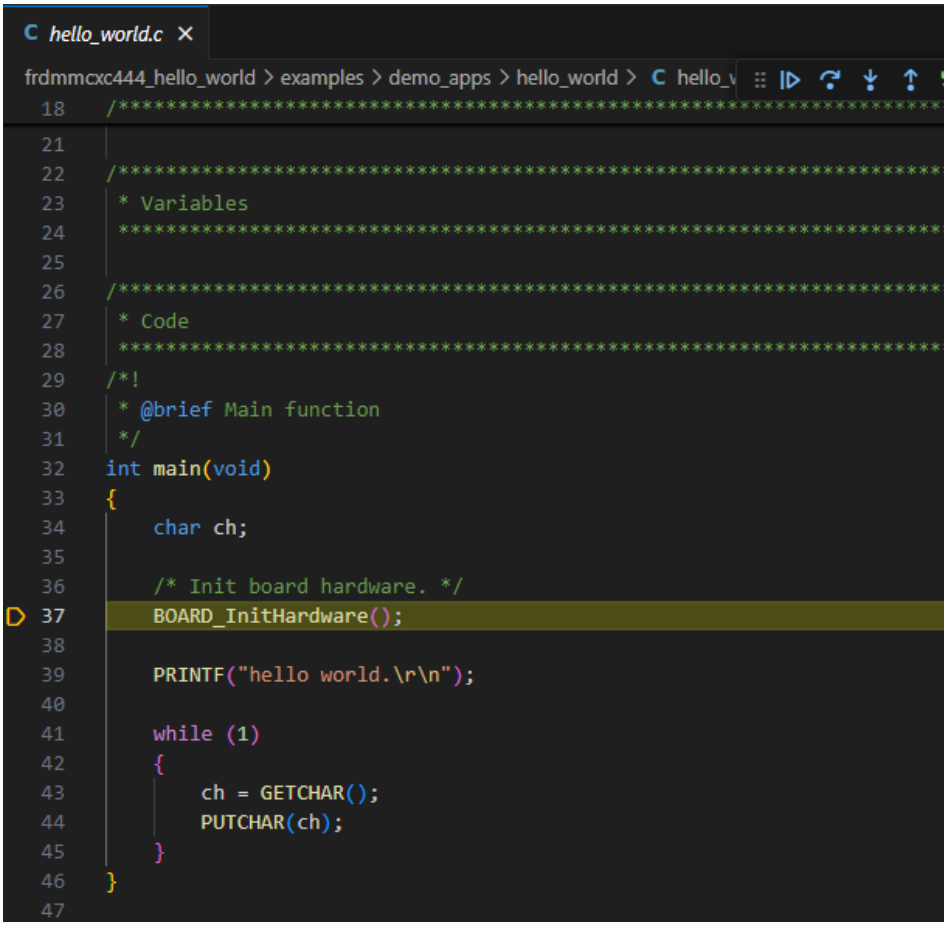
The debug session will begin with debug controls initially at the top of the interface.

Debug Controls

Use the debug controls to manage execution:

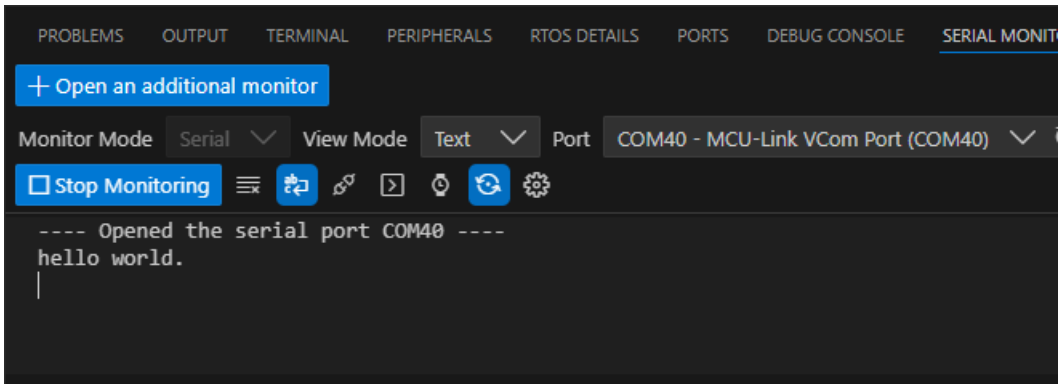
- **Continue**: Resume code execution
- **Step controls**: Navigate through code

- **Stop**: Terminate debug session



Monitor Output

Observe application output in the **Serial Monitor** to verify correct operation.



Debug Probe Support For comprehensive information on debug probe support and configuration, refer to the MCUXpresso for VS Code Debug documentation.

Project Configuration Workspace Management

The extension integrates with the MCUXpresso SDK workspace structure, providing access to:

- Example applications
- Board configurations

- Middleware components
- Build system integration

Multi-Project Support

The PROJECTS view allows management of multiple imported projects within the same workspace.

Troubleshooting Import Issues

SDK not detected:

- Verify SDK workspace is properly initialized
- Ensure all required repositories are updated
- Check SDK manifest files are present

Project import failures:

- Confirm board support exists for selected example
- Verify toolchain installation
- Check example compatibility with selected board

Build Problems

Build failures:

- Check integrated terminal for error messages
- Verify all dependencies are installed
- Ensure toolchain is properly configured

Debug Issues

Debug session fails:

- Verify board connection via USB
- Check debug probe drivers are installed
- Confirm build completed successfully

Serial monitor problems:

- Verify correct VCom port selection
- Check baud rate configuration (115200)
- Ensure board drivers are installed

Integration with Command Line MCUXpresso for VS Code integrates with the underlying west build system, allowing seamless integration with command line workflows described in *Command Line Development*.

Advanced Features Project Types

The extension supports both repository-based and freestanding project types, providing flexibility in project organization and SDK integration.

Build System Integration

The extension leverages the MCUXpresso SDK build system, providing access to all build configurations and options available through command line tools.

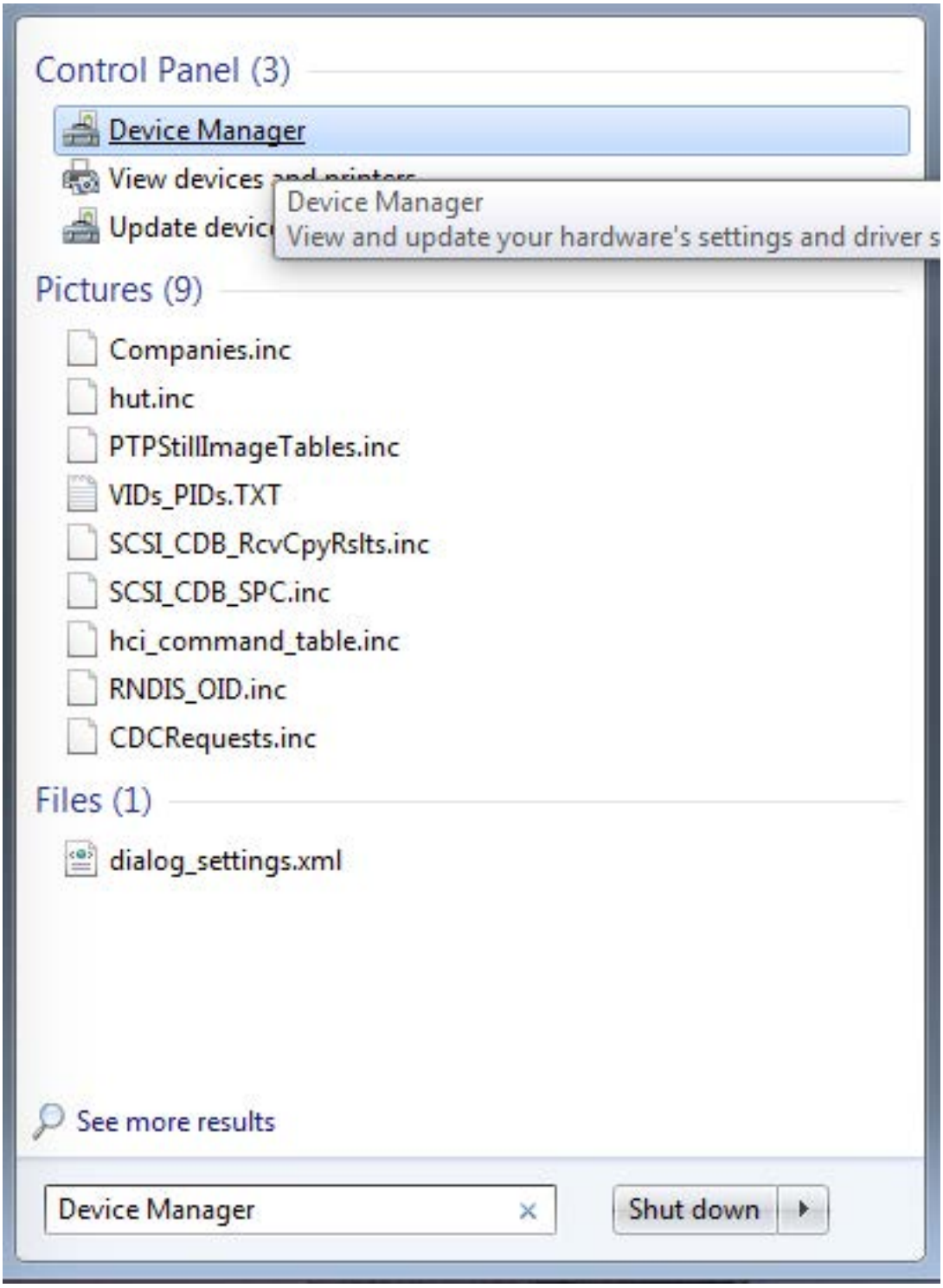
Next Steps

- Explore additional examples in the SDK
- Review *Command Line Development* for advanced build options
- Refer to the MCUXpresso for VS Code documentation for detailed documentation
- Learn about *SDK Architecture* for better understanding of the development environment

How to determine COM Port

This section describes the steps necessary to determine the debug COM port number of your NXP hardware development platform. All NXP boards ship with a factory programmed, on-board debug interface MCU-LINK.

1. To determine the COM port, open the Windows operating system **Device Manager**. This can be achieved by going to the Windows operating system **Start** menu and typing **Device Manager** in the search bar, as shown in *Figure 1*.



2. In the **Device Manager**, expand the **Ports (COM & LPT)** section to view the available ports.

How to set the board to the Bootloader ISP mode

On the MCXW72-LOC board:

1. Press and hold the BOOT CONFIG switch, **SW4**.
2. Connect the MCXW72-LOC board via the micro-USB connector to the PC.
3. Release the switch **SW4**.

4. Check the associated USB port number on the PC (such as **COM10**).
5. While the MCXW72 is in bootloader ISP mode, navigate to the folder where *blhost.exe* is located.
6. Type the command `.\blhost.exe -p COM10 -- get-property 1` to make sure that the MCXW72 is in Bootloader ISP mode, you should see:
   - Ping responded in 1 attempt(s)
   - Inject command 'get-property'
   - Response status = 0 (0x0) Success.
   - Response word 1 = 1258488064 (0x4b030100)
   - Current Version = K3.1.0

On a custom board:

1. Short the `BOOT_CFG` pin to VDD while reset.

Updating debugger firmware

The MCXW72-LOC board comes with a CMSIS-DAP-compatible debug interface (known as MCU-Link). This firmware in this debug interface may be updated using the host computer scripts. This firmware is typically used when switching between the default debugger protocol (CMSIS-DAP) to SEGGER J-Link, or for updating this firmware with new releases of these. This section contains the steps to re-program the debug probe firmware.

NXP provides the MCU-Link utility, which is the recommended tool for programming the latest versions of CMSIS-DAP and J-Link firmware onto MCU-Link. The utility can be downloaded from MCU-Link Debug Probe.

These steps show how to update the debugger firmware on your board for Windows operating system. For Linux OS, follow the instructions described in MCU-Link user guide, MCU-Link Debug Probe.

1. Install the MCU-Link utility.
2. Unplug the USB cable of the board.
3. Install the jumper on JP20.
4. Connect the probe to the host via USB (use Link USB connector).
5. Open a command shell and call the appropriate script located in the MCU-Link installation directory, *<MCU-Link install dir>*.
   1. To program CMSIS-DAP debug firmware: *<MCU-Link install dir>/scripts/program_CMSIS*.
   2. To program J-Link debug firmware: *<MCU-Link install dir>/scripts/program_JLINK*.
6. Remove the jumper on JP20.
7. Re-power the board by removing the USB cable and plugging it in again.

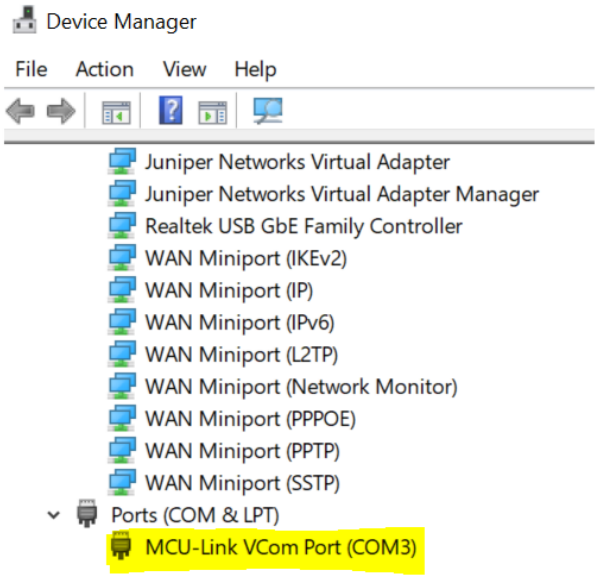
Updating NBU for Wireless Examples

The Narrow Band Unit (NBU) firmware is included in the SDK folder as a signed and encrypted FW.

middleware|wireless|ble_controller|bin|mcxw72_nbu_ble_hosted.bin

To program the NBU software for the MCXW72, perform the following steps:

1. While holding pressed the **SW4** on the MCXW72-LOC board, attach the USB connector J3 to your computer. Then, release the **SW4** after you plugged the USB cable on your computer.
2. Verify which COM port is assigned to your MCXW72-LOC board. To check the assigned COM port, in the Windows **Device Manager** program, search for Ports **(COM & LPT)** and save the COM port number. In this example, the assigned COM port is **COM3**.



3. Execute the following commands in order, replacing **COMX** with your actual COM port number (e.g., **COM3**):

   Make sure the device has entered ISP mode properly. If the device did not enter in ISP mode, it cannot be programmed.

```
blhost.exe -p COMX get-property 1
```

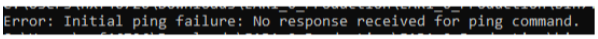
   Erase NBU flash

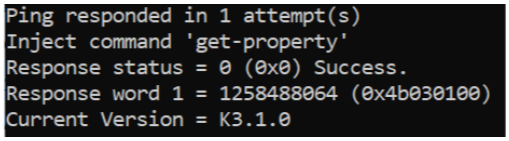
```
blhost.exe -p COMX flash-erase-all 2
```

   Program NBU binary into NBU flash

```
blhost.exe -p COMX write-memory 0x48800000 mcxw72_nbu_ble_hosted.bin
```

   The following examples show when the device did not enter in ISP (left) and when the device was programmed properly (right).





1.4 Getting Started with MCUXpresso SDK GitHub

1.4.1 Getting Started with MCUXpresso SDK Repository

Welcome to the **GitHub Repository SDK Guide**. This documentation provides instructions for setting up and working with the MCUXpresso SDK distributed in a **multi-repository model**. The SDK is distributed across multiple GitHub repositories and managed using the **Zephyr West** tool, enabling modular development and streamlined workflows.

Overview

The GitHub Repository SDK approach offers:

- **Modular Structure**: Multiple repositories for flexibility and scalability.
- **Zephyr West Integration**: Simplified repository management and synchronization.
- **Cross-Platform Support**: Designed for MCUXpresso SDK development environments.

Benefits of the Multi-Repository Approach

- **Scalability**: Easily add or update components without impacting the entire SDK.
- **Collaboration**: Enables distributed development across teams and repositories.
- **Version Control**: Independent versioning for components ensures better stability.
- **Automation**: Zephyr West simplifies dependency handling and repository synchronization.

Setup and Configuration

Follow these steps to prepare your development environment:

GitHub Repository Setup This guide explains how to initialize your MCUXpresso SDK workspace from GitHub repositories using the west tool. The GitHub Repository SDK uses multiple repositories hosted on GitHub to provide modular, flexible development.

Prerequisites

Verify the requirements:

System Requirements:

- Python 3.8 or later
- Git 2.25 or later
- CMake 3.20 or later
- Build tools for your target platform

Verification Commands:

```
python --version    # Should show 3.8+
git --version       # Should show 2.25+
cmake --version     # Should show 3.20+
west --version      # Should show west tool installation
```

Workspace Initialization

The GitHub Repository SDK uses the Zephyr west tool to manage multiple repositories containing different SDK components.

Step 1: Initialize Workspace

Create and initialize your SDK workspace from GitHub:

Get the latest SDK from main branch:

```
west init -m https://github.com/nxp-mcuxpresso/mcuxsdk-manifests.git mcuxpresso-sdk
```

Get SDK at specific revision:

```
west init -m https://github.com/nxp-mcuxpresso/mcuxsdk-manifests.git mcuxpresso-sdk --mr {revision}
```

Note: Replace {revision} *with the desired release tag, such as* v25.09.00

Step 2: Choose Your Repository Update Strategy

Navigate to the SDK workspace:

```
cd mcuxpresso-sdk
```

The west tool manages multiple GitHub repositories containing different SDK components. You have two options for downloading:

Option A: Download All Repositories (Complete SDK)

Download all SDK repositories for comprehensive development:

```
west update
```

This command downloads all the repositories defined in the manifest from GitHub. Initial download takes several minutes and requires ~7 GB of disk space.

Best for:

- Exploring the complete SDK
- Multi-board development projects
- Comprehensive middleware evaluation

Option B: Targeted Repository Download (Recommended)

Download only repositories needed for your specific board or device to save time and disk space:

```
# For specific board development
west update_board --set board your_board_name

# For specific device family development
west update_board --set device your_device_name

# List available repositories before downloading
west update_board --set board your_board_name --list-repo
```

Best for:

- Single board development
- Faster setup and reduced disk usage
- Focused development workflows

Examples:

```
# Update only repositories for FRDM-MCXW23 board
west update_board --set board frdmmcxw23

# Update only repositories for MCXW23 device family
west update_board --set device mcxw23
```

Step 3: Verify Installation

Confirm successful setup:

```
# Verify workspace structure
ls -la
# Should show: manifests/ and mcuxsdk/ directories

# Test build system
west list_project -p examples/demo_apps/hello_world
# Should display available build configurations
```

Advanced Repository Management

The west extension command `update_board` provides advanced repository management capabilities for optimized workspace setup with GitHub repositories.

Board-Specific Setup

Update only repositories required for a specific board:

```
# Update only repositories for specific board, e.g., frdmmcxw23
west update_board --set board frdmmcxw23

# List available repositories for the board before updating
west update_board --set board frdmmcxw23 --list-repo
```

Device-Specific Setup

Update only repositories required for a specific device family:

```
# Update only repositories for specific device, e.g., MCXW235
west update_board --set device mcxw23

# List available repositories for the device family
west update_board --set device mcxw23 --list-repo
```

Custom Configuration

For advanced users who want to create custom repository combinations:

```
# Use custom configuration file
west update_board --set custom path/to/custom-config.yml

# Generate custom configuration template
cp manifests/boards/custom.yml.template my-custom-config.yml
```

Benefits of Targeted Setup

Reduced Download Size

- Download only components needed for your target board or device
- Significantly faster initial setup for focused development
- Typical reduction from 7 GB to 2GB

Optimized Workspace

- Cleaner workspace with relevant components only
- Reduced disk space usage

- Faster repository operations

Flexible Development

- Switch between different board configurations easily
- Maintain separate workspaces for different projects
- Include optional components as needed

Repository Information

Before setting up your workspace, you can explore what repositories are available:

```
# Display repository information in console
west update_board --set board frdmmcxw23 --list-repo

# Export repository information to YAML file for reference
west update_board --set board frdmmcxw23 --list-repo -o board-repos.yml
```

This command lists all the available repositories with descriptions and outlines the included components in the workspace.

Package Generation (Optional)

The `update_board` command can also generate ZIP packages for offline distribution:

```
# Generate board-specific SDK package
west update_board --set board frdmmcxw23 -o frdmmcxw23-sdk.zip
```

Note: Package generation is primarily intended for creating custom SDK distributions. For regular development, use the workspace update commands without the `-o` option.

Workspace Management

Updating Your Workspace

Keep your SDK current with latest updates from GitHub:

For Complete SDK Workspace:

```
# Update manifest repository
cd manifests
git pull

# Update all component repositories
cd ..
west update
```

For Targeted Workspace:

```
# Update manifest repository
cd manifests
git pull

# Update board-specific repositories
cd ..
west update_board --set board your_board_name
```

Workspace Status

Check workspace synchronization status:

```
# Show status of all repositories
west status

# Show detailed information about repositories
west list
```

Troubleshooting

Network Issues:

- Use `west update --keep-descendants` for partial failures
- Configure Git credentials for private repositories
- Check firewall settings for Git protocol access

Permission Issues:

- Ensure write permissions in workspace directory
- Run commands without sudo/administrator privileges
- Verify Git SSH key configuration for authenticated access

Disk Space:

- Full SDK workspace requires approximately 7-8 GB
- Targeted workspace typically requires 1-2 GB
- Use board-specific setup to reduce workspace size

Repository Management Issues:

- Verify board/device names match available configurations
- Check that custom YAML files follow the correct template format
- Use `--list-repo` to verify available repositories before setup

Next Steps

With your workspace initialized:

1. Review *Workspace Structure* to understand the layout
2. Build your first project with *First Build Guide*
3. Explore *Development Workflows MCUXPresso VSCode* or *Development Workflows Command Line* for the details on project setup and execution

For advanced repository management, see the west tool documentation.

Prebuilt Binary Blobs Some SDK components ship with prebuilt binary files (`.a`, `.lib`, `.bin`, etc.) that are **not included in the git repository itself**. Instead they are fetched on demand from a dedicated blob repository using the `west blobs` command.

This keeps the main SDK checkout small and fast while still making the binaries available immediately when you need them.

When do you need to fetch blobs?

You typically need blobs when:

- You are building an example that links against a prebuilt library (e.g. a machine learning inference engine, a wireless stack library, or a graphics middleware).

- After a fresh west init + west update, the build fails with a linker error about a missing .a or .lib file.
- You run west blobs check and see MISSING entries.

Quick start

Run these commands from your SDK workspace root (the directory that contains manifests/ and mcuxsdk/):

```
# 1. See what blobs are available and whether they are already present
west blobs check

# 2. Fetch any missing blobs
west blobs fetch

# 3. Confirm everything is present
west blobs check
```

A green result looks like:

```
mcu-sdk-examples: 1078/1078 entries: OK
```

Fetch blobs for a specific project only

If you only need blobs for one component, pass the west project name:

```
west blobs fetch mcu-sdk-examples
```

Use west blobs list to see available project names and their blob files:

```
west blobs list
```

License

By fetching blobs you accept the license terms bundled with each blob set. The applicable license file is located alongside mcux_blobs.yml in the relevant SDK subdirectory (for example mcuxsdk/examples/LICENSE).

Some blob sets require explicit license acceptance. If a blob has click-through: True, west blobs fetch will prompt you to confirm before downloading.

Keeping blobs up to date

After running west update (which updates the SDK source), re-run:

```
west blobs fetch
```

Blobs that are already present and whose sha256 matches are skipped automatically; only new or changed blobs are downloaded.

Troubleshooting

Build error: cannot find -lsomething or missing .a / .bin

```
west blobs fetch        # fetch all missing blobs
west blobs check        # verify
```

Re-fetch a corrupted or incomplete blob

```
# Remove locally cached blobs for one project and re-download
west blobs clean mcu-sdk-examples
west blobs fetch mcu-sdk-examples
```

Network / proxy issues

`west blobs fetch` downloads files over HTTPS. If your network requires a proxy, set the standard environment variables before running the command:

```
export HTTPS_PROXY=http://your-proxy:port
west blobs fetch
```

Checking without downloading

```
west blobs check        # shows OK or MISSING per entry; does not download
west blobs list         # lists all declared blobs; does not download
```

Next steps

- *Building Your First Project* — build and run an SDK example
- *Exploring the SDK* — workspace layout and example organization

Explore SDK Structure and Content

Learn about the organization of the SDK and its components:

SDK Architecture Overview The MCUXpresso SDK uses a modular architecture where software components are distributed across multiple repositories hosted on GitHub and managed through the west tool. This approach provides flexibility, maintainability, and enables selective component inclusion.

Repository Organization

Based on the manifest structure, the SDK consists of four main repository categories:

Manifest Repository

The manifest repo (mcuxsdk-manifests) contains the west.yml manifest file that tracks all other repositories in the SDK.

Base Repositories

Recorded in `submanifests/base.yml` and loaded in the root west.yml manifest file. These are the foundational repositories that build the SDK:

- **Devices**: MCU-specific support packages
- **Examples**: Demonstration applications and code samples

- **Boards**: Board support packages

Middleware Repositories

Recorded in the `submanifests/middleware` subdirectory, categorized according to functionality:

- **Connectivity**: Networking stacks, USB, and communication protocols
- **Security**: Cryptographic libraries and secure boot components
- **Wireless**: Bluetooth, IEEE 802.15.4, and other wireless protocols
- **Graphics**: Display drivers and UI frameworks
- **Audio**: Audio processing and voice recognition libraries
- **Machine Learning**: AI inference engines and neural network libraries
- **Safety**: IEC60730B safety libraries
- **Motor Control**: Motor control and real-time control libraries

Internal Repositories

Recorded in `submanifests/internal.yml` and grouped into the "bifrost" group. These are only visible to NXP internal developers and hosted on NXP internal git servers.

Repository Hosting

Public repositories are hosted on GitHub under these organizations:

- nxp-mcuxpresso
- NXP
- nxp-zephyr

Internal repositories are hosted on NXP's private Git infrastructure.

Benefits of This Architecture

Selective Integration: Projects include only required components, reducing memory footprint and build complexity.

Independent Versioning: Each component maintains its own release cycle and version control.

Community Collaboration: Public repositories accept community contributions through standard Git workflows.

Scalable Maintenance: Component owners can update their repositories without affecting the entire SDK.

Workspace Management

The west tool manages repository synchronization, version tracking, and workspace updates. All repositories are checked out under the `mcuxsdk/` directory with their designated paths defined in the manifest files.

Development Workflows

Get started with building and running projects:

Command Line Development This guide covers developing with the MCUXpresso SDK using command line tools and the west build system. This workflow applies to both GitHub Repository SDK and Repository-Layout SDK Package distributions.

Prerequisites

- GitHub Repository SDK workspace initialized OR Repository-Layout SDK Package extracted
- Development tools installed per *Installation Guide*
- Target board connected via USB

Understanding Board Support

Use the west extension to discover available examples for your board:

```
west list_project -p examples/demo_apps/hello_world
```

This shows all supported build configurations. You can filter by toolchain:

```
west list_project -p examples/demo_apps/hello_world -t armgcc
```

Basic Build Commands

Standard Build Process

Build with default settings (armgcc toolchain, first debug config):

```
west build -b your_board examples/demo_apps/hello_world
```

Specifying Build Configuration

```
# Release build
west build -b your_board examples/demo_apps/hello_world --config release

# Debug build with specific toolchain
west build -b your_board examples/demo_apps/hello_world --toolchain iar --config debug
```

Multicore Applications

For multicore devices, specify the core ID:

```
west build -b evkbmimxrt1170 examples/demo_apps/hello_world --toolchain iar -Dcore_id=cm7 --config␣
↪flexspi_nor_debug
```

For multicore projects using sysbuild:

```
west build -b evkbmimxrt1170 --sysbuild ./examples/multicore_examples/hello_world/primary -Dcore_
↪id=cm7 --config flexspi_nor_debug --toolchain=armgcc -p always
```

Shield Support

For boards with shields:

```
west build -b mimxrt700evk --shield a8974 examples/issdk_examples/sensors/fxls8974cf/fxls8974cf_poll -
↪Dcore_id=cm33_core0
```

Advanced Build Options

Clean Builds

Force a complete rebuild:

```
west build -b your_board examples/demo_apps/hello_world -p always
```

Dry Run

See what commands would be executed:

```
west build -b your_board examples/demo_apps/hello_world --dry-run
```

Device Variants

For boards supporting multiple device variants:

```
west build -b your_board examples/demo_apps/hello_world --device MK22F12810 --config release
```

Project Configuration

CMake Configuration Only

Run configuration without building:

```
west build -b evkbmimxrt1170 examples/demo_apps/hello_world -Dcore_id=cm7 --cmake-only -p
```

Interactive Configuration

Launch the configuration GUI:

```
west build -t guiconfig
```

Flashing and Debugging

Flash Application

Flash the built application to your board:

```
west flash -r linkserver
```

Debug Session

Start a debugging session:

```
west debug -r linkserver
```

IDE Project Generation

To use a traditional IDE instead of the command line or VS Code, generate an IDE project from the CMake build with `-t guiproject`:

- *Run a demo using IAR*
- Run a demo using MDK

Both guides cover the Ruby dependency required for IDE project generation.

Troubleshooting

Build Failures

Use pristine builds to resolve dependency issues:

```
west build -b your_board examples/demo_apps/hello_world -p always
```

Toolchain Issues

Verify environment variables are set correctly:

```
# Check ARM GCC
echo $ARMGCC_DIR
arm-none-eabi-gcc --version

# Check IAR (if using)
echo $IAR_DIR
```

Getting Help

Display help information:

```
west build -h
west flash -h
west debug -h
```

Check Supported Configurations

If unsure about supported options for an example:

```
west list_project -p examples/demo_apps/hello_world
```

Best Practices

Project Organization

- Keep custom projects outside the SDK tree
- Use version control for your application code
- Document any SDK modifications

Build Efficiency

- Use `-p always` for clean builds when troubleshooting
- Leverage `--dry-run` to understand build processes
- Use specific configs and toolchains to reduce build time

Development Workflow

1. Start with existing examples closest to your requirements
2. Copy and modify rather than building from scratch
3. Test with hello_world before moving to complex examples
4. Use configuration tools for pin muxing and clock setup

Next Steps

- Explore *VS Code Development* for integrated development experience
- Review *Workspace Structure* to understand SDK organization
- Refer build system documentation for advanced configurations

Using MCUXpresso Config Tools MCUXpresso Config tools provide a user-friendly way to configure hardware initialization of your projects. This guide explains the basic workflow with the MCUXpresso SDK west build system and the Config Tools.

Prerequisites

- GitHub Repository SDK workspace initialized OR Repository-Layout SDK Package extracted
- MCUXpresso Config Tools standalone installed (version 25.09 or above)
- MCUXpresso SDK Project that can be successfully built

Board Files

MCUXpresso Config Tools generate source files for the board. These files include pin_mux.c/h and clock_config.c/h. The files contain initialization code functions that reflect the hardware configuration in the Config Tools. Within the SDK codebase, these files are specific for the board and either shared by multiple example projects or specific for one example. Open or import the configuration from the SDK project in the Config Tools and customize the settings to match the custom board or specific project use case and regenerate the code. See *User Guide for MCUXpresso Config Tools (Desktop)* (document GSMCUXCTUG) for details.

Note: When opening the configuration for SDK example projects, the board files may be shared across multiple examples. To ensure a separate copy of the board configuration files exists, create a freestanding project with copied board files.

Visual Studio Code

To open the configuration in Visual Studio Code, use the context menu for the project to access Config Tools. See MCUXpresso Extension Documentation for details.
Otherwise, use the manual workflow described in detail in the following section.

Manual Workflow

Use the following steps:

1. Before using Config Tools, run the west command to get the project information for Config Tools from the SDK project files, for example:

   ```
   west cfg_project_info -b lpcxpresso55s69 ...mcuxsdk/examples/demo_apps/hello_world/ -Dcore_
   ↪id=cm33_core0
   ```

   This results in the creation of the project information json file that is searched by the config tools when the configuration is created. The parameters of the command should match the build parameters that will be used for the project.

2. Launch the MCUXpresso Config Tools and in the **Start development** wizard, select **Create a new configuration based on the existing IDE/Toolchain project**. Select the created "cfg_tools" subfolder as a project folder (for example: ...mcuxsdk/examples/demo_apps/hello_world/cfg_tools/).

Updating the SDK West project

Note: Updating project is supported with Config Tools V25.12 or newer only.

Changes in the Config tools generated source code modules may require adjustments to the toolchain project to ensure a successful build. These changes may mean, for example, adding the newly generated files, adding include paths, required drivers, or other SDK components. This section describes how to manually resolve the changes needed in the project within the toolchain projects based on the SDK project managed by the West tool.

After the configuration in the Config Tools is finished, write updated files to the disk using the 'Update Code' command. The written files include a json file with the required changes for the toolchain project.

To resolve the changes in the project in the terminal, launch the west command that updates the project. For example:

```
west cfg_resolve -b lpcxpresso55s69 ...mcuxsdk/examples/demo_apps/hello_world/ -Dcore_id=cm33_core0
```

This command updates the appropriate cmake and kconfig files to address the changes. After this, the application can be built.

Note: The cfg_resolve command supports additional arguments. Launch the *west cfg_resolve -h* command to get the list and description.

1.5 Release Notes

1.5.1 MCUXpresso SDK Release Notes

Development tools

The MCUXpresso SDK was tested with following development tools. Same versions or above are recommended.

SDK 25.12.00 is the final release that includes MCUXpresso IDE support for MCX W7x devices. Beginning with SDK 26.x.x, MCUXpresso IDE will no longer be supported for MCX W7x development, and MCUXpresso for Visual Studio Code will become the recommended integrated development environment.

- IAR Embedded Workbench for Arm, version is 9.70.4
- MCUXpresso for VS Code v26.06
- GCC Arm Embedded Toolchain 14.2.x

Supported development systems

This release supports board and devices listed in following table. The board and devices in bold were tested in this release.

Development boards	MCU devices
MCXW72-LOC	**MCXW727CMFTA**

MCUXpresso SDK release package

The MCUXpresso SDK release package content is aligned with the silicon subfamily it supports. This includes the boards, CMSIS, devices, middleware, and RTOS support.

Device support The device folder contains the whole software enablement available for the specific System-on-Chip (SoC) subfamily. This folder includes clock-specific implementation, device register header files, device register feature header files, and the system configuration source files. Included with the standard SoC support are folders containing peripheral drivers, toolchain support, and a standard debug console. The device-specific header files provide a direct access to the microcontroller peripheral registers. The device header file provides an overall SoC memory mapped register definition. The folder also includes the feature header file for each peripheral on the microcontroller. The toolchain folder contains the startup code and linker files for each supported toolchain. The startup code efficiently transfers the code execution to the main() function.

Board support The boards folder provides the board-specific demo applications, driver examples, and middleware examples.

Demo application and other examples The demo applications demonstrate the usage of the peripheral drivers to achieve a system level solution. Each demo application contains a readme file that describes the operation of the demo and required setup steps. The driver examples demonstrate the capabilities of the peripheral drivers. Each example implements a common use case to help demonstrate the driver functionality.

RTOS

FreeRTOS Real-time operating system for microcontrollers from Amazon

Middleware

GenFSK link layer The Generic FSK protocol enables radio operation using a custom GFSK/GMSK or MSK modulation format.

Main Features supported:

- Highly configurable packet structure
- Optimized Sequence Command Set
- High-precision timebase to maintain network timing
- Two timer-compare mechanisms for Interrupt Generation and Sequence Launching
- Hardware automation for packet transmit and receive, CRC and Whitening
- Up to four network addresses to synchronize to, can be 8-bit, 16-bit or 32-bit
- Packet Lengths up to 2047 Bytes
- Support complex auto-sequence, like CCA before TX, Auto-ACK, TR.
- Many operating modes can support the sending and receiving of multiple protocol packets, such as Bluetooth LE.

Wireless XCVR The XCVR component provides a base Transceiver Driver for the 2.4 GHz narrowband radio.

Bluetooth LE Controller

- Main features supported:
  - Peripheral Role
  - Central Role
  - Multiple PHYs (1 Mbps, 2 Mbps, Coded PHY)
  - Asymmetric Connections
  - Public/Random/Static Addresses
  - Network/Device Privacy Modes
  - Extended Advertising
  - Extended Scanning
  - Passive/Active Scanning
  - LE Encryption
  - LE Ping Procedure
  - HCI Test Interface
  - UART Test Interface
  - Randomized Advertising Channel Indexing
  - Sleep Clock Accuracy Update - Mechanism
  - ADI Field in Scan Response Data
  - HCI Support for Debug Keys in LE - Secure Connections
- Main capabilities supported:
  - Simultaneous scanning 1 Mbps and Long Range
  - Scanning and advertising in parallel
  - 24 connections as a central role
  - 24 connections as a peripheral role
  - Any combination of central and peripheral roles (24 connections maximum)
  - 8 connections with a 7.5 ms connection interval
  - Five advertising sets in parallel.
  - 26 Accept List entries
  - 36 Resolvable Private Address (RPA) entries
  - Up to two Chain Packets per Extended Advertising set
  - Enhanced Notification on end of - Scanning/Advertising/Connection events
  - Connection event counters associated to Bluetooth LE packet reception
  - Timestamp associated to Bluetooth LE packet reception
  - RF channel info associated to Bluetooth LE packet reception
  - NXP proprietary Bluetooth LE Handover feature
  - Decision Based Advertising Filtering (DBAF)
  - Advertising Coding Selection (ACS)
  - Periodic Advertising with Responses (PAwR) Additional features supported for KW47 and MCX W72 devices:

- Channel Sounding Additional features supported as EAR (\Early Access Release) in the KW47 experimental build:
- Inline PCT Transfer (IPT)
- RTT_PHY capability
- Channel Sounding Enhancement 1
- Channel Sounding TX/SNR (18dB, 21 dB, 24 dB and 27dB)
- Channel Sounding PHY 2Mbps BT2.0
- Channel Sounding additional supported timings:
  * T_PM=10us
  * T_IP1/T_IP2=60, 50, 30
  * T_FCS=120, 100, 60
- LE Power Control (\LEPC).

Additional features supported as EAR (\Early Access Release) in the KW45 / KW47 experimental builds:

- LE Test Mode Enhancement (\UTP/OTA).
- LL Extended Feature Set
- Monitoring Advertisers
- Randomized Resolvable Private Address (\RPA)
- LE Enhanced Connection Update

  Note: Project configuration enabling Experimental features on KW45 and MCX W71 requires the Radio Subsystem (NBU) Firmware to be reprogrammed with the firmware provided in the SDK under |*middleware*|*wireless*|*ble_controller*|*bin*|*experimental*|. For NBU programming steps, see the *EVK Quick Start Guide* and Secure Provisioning SDK (SPSDK) documentation.

  Project configurations that require usage of the Bluetooth LE controller including all Bluetooth LE examples require the Radio Subsystem (NBU) Firmware to be re-programmed with the firmware provided in the SDK under middleware\wireless\ble_controller\bin.

Bluetooth LE Host Stack and Applications The Bluetooth LE Host Stack component provides an implementation for a Bluetooth 6.0 mandatory and some optional, proprietary, and experimental features. The Bluetooth LE Host Stack component provides application examples, services, and profiles.

Main features supported:

- Automotive Compliance
- MISRA Compliance
- HIS CCM <= 20
- Advanced Secure Mode
- Enhanced ATT
- GATT Caching
- GCC Libraries
- IAR Libraries
- Bluetooth LE Peripheral Libraries

- Bluetooth LE Central Libraries
- Bluetooth LE Full Host Features Libraries
- Bluetooth LE Host Optional Features Libraries
- Bluetooth LE Host Mandatory Features Libraries
- BareMetal and FreeRTOS Support
- Full Privacy Support
- NCP Mode - FSCI Application
- Enhanced Notifications
- Dynamic Database
- OTA Support - Sample Applications
- Decision based Advertising Filtering (DBAF)
- Advertising Coding Selection (ACS)
- Periodic Advertising with Responses (PAwR)
- Encrypted Advertising Data (EAD)
- Monitoring Advertisers
- Randomized RPA - Experimental feature
- Intrusion Detection System (IDS) - Experimental feature

  Note: The CDE(Complex-domain Distance Estimation) algorithm is an experimental feature intended only for testing purposes. It is not maintained and must not be included in production environments.

  Note: "RADE (Ranging Distance Estimation) is NXP's embedded Channel Sounding (CS) ranging algorithm. RADE is intended for non-U.S. markets and should not be used in U.S. market deployments. RADEX is the U.S.-market variant of the RADE algorithm."

Wireless Localization Localization

Connectivity framework The Connectivity Framework is a software component that provides hardware abstraction modules to the upper layer connectivity stacks and components. It also provides a list of services and APIs (see *Supported services*). The Connectivity Framework modules are located in the *middleware|wireless|framework SDK* folder.

Supported services

- **FSCI** - Framework Serial Communication Interface
- **FunctionLib** - Common function library utilities
- **HWParameter** - Hardware parameter management
- **LowPower** - Low power mode management
- **ModuleInfo** - Module information and versioning
- **NVS** - Non-Volatile Storage
- **NVM** - Non-Volatile Memory management
- **OtaSupport** - Over-The-Air update support
- **SecLib_RNG** - Security library and Random Number Generator
- **Sensors** - Sensor abstraction layer

- **SFC** - Smart Frequency Calibration
- **WorkQ** - Work queue management

Supported platform

- **KW45_MCXW71**
- **KW47_MCXW72**
- **MCXW23**
- **RW61X**
- **RT1060** and **RT1170**
- **i.MX RT595s**

Advanced features supported on platforms

KW45_MCXW71

- FRO32K with smart frequency calibration (see SFC)
- Power down mode support (for evaluation only)

KW47_MCXW72

- FRO32K with smart frequency calibration (see SFC)
- Power down mode support (for evaluation only)
- Crystal 32M trimming with temperature
- Debug module for NBU
- Extended NBU support with SecLib and pseudo RNG support

CMSIS DSP Library The MCUXpresso SDK is shipped with the standard CMSIS development pack, including the prebuilt libraries.

TF-M Trusted Firmware - M Library

PSA Test Suite Arm Platform Security Architecture Test Suite

NXP PSA CRYPTO DRIVER PSA crypto driver for crypto library integration via driver wrappers

secure_storage secure_storage

EdgeLock SE050 Plug and Trust Middleware Secure subsystem library - SSS APIs

Multicore Multicore Software Development Kit

NXP IoT Agent NXP IoT Agent

mbedTLS mbedtls SSL/TLS library v3.x

mbedTLS mbedtls SSL/TLS library v2.x

LittleFS LittleFS filesystem stack

FreeMASTER FreeMASTER communication driver for 32-bit platforms.

Release contents

Provides an overview of the MCUXpresso SDK release package contents and locations.

Deliverable	Location
Boards	INSTALL_DIR/boards
Demo Applications	INSTALL_DIR/boards/<board_name>/demo_apps
Driver Examples	INSTALL_DIR/boards/<board_name>/driver_examples
eIQ examples	INSTALL_DIR/boards/<board_name>/eiq_examples
Board Project Template for MCUXpresso IDE NPW	INSTALL_DIR/boards/<board_name>/project_template
Driver, SoC header files, extension header files and feature header files, utilities	INSTALL_DIR/devices/<device_name>
CMSIS drivers	INSTALL_DIR/devices/<device_name>/cmsis_drivers
Peripheral drivers	INSTALL_DIR/devices/<device_name>/drivers
Toolchain linker files and startup code	INSTALL_DIR/devices/<device_name>/<toolchain_nam
Utilities such as debug console	INSTALL_DIR/devices/<device_name>/utilities
Device Project Template for MCUXpresso IDE NPW	INSTALL_DIR/devices/<device_name>/project_templat
CMSIS Arm Cortex-M header files, DSP library source	INSTALL_DIR/CMSIS
Components and board device drivers	INSTALL_DIR/components
RTOS	INSTALL_DIR/rtos
Release Notes, Getting Started Document and other documents	INSTALL_DIR/docs
Tools such as shared cmake files	INSTALL_DIR/tools
Middleware	INSTALL_DIR/middleware

What is new

The following updates were implemented with respect to the previous SDK release version (26.09.00-pvw2).

- **Bluetooth LE Host Stack and Applications**

  Note: Version 1.10.23 is the first NXP Bluetooth LE Host release officially certified according to the Bluetooth® Core Specification Version 6.3.

  Improved

  - Optimized stack sizes for the PSA module (MCX W70/KW43).
  - Added MinimalHeapSize_c to all localization applications, scaled for RADE2 and number of connections.
  - Optimized memory buffer values in the loc_reader Channel Sounding application.

- Updated wireless ranging Python host application setup documentation (Python 3.12+ supported).

Fixed

- Fixed ExtendedFeatures mismatch between GAPInit and the HCILEReadAllRemoteFeaturesCompleteevent.
- Fixed EATT out-of-bounds read in IsEnhncdChanReconfInProgress().
- Fixed GattDbDynamic_AddCharServiceChanged using Notify instead of Indicate property.
- Fixed ShellGap_ConnectFromPawr setting the success status incorrectly.
- Miscellaneous Coverity fixes.
- Miscellaneous CERT-C fixes.
- Miscellaneous MISRA fixes.

Changed

- Bluetooth 6.3 Compliance: removed Data Signing (LE Security Mode 2).
- Details can be found in github repository **nxp-mcuxpresso/mcuxsdk-middleware-bluetooth-host/CHANGELOG.md**.

- **Bluetooth LE controller**
  - HADM, PAwR fixes, and stability improvements.
- **Transceiver drivers (XCVR)**
  - Added support for Bluetooth LE Channel Sounding.
  - Added API to control Power Amplifier (PA) ramp type and duration.
- **Connectivity framework**
  - **Major Changes**
    * [platform][DBG] Moved the NBU software watchdog (handshake based NBU stuck detection) into the common platform_dbg sources, making the feature available on both KW43/MCXW70 and KW47/MCXW72 without additional files.
    * [NVM] Refactored NVM handling to centralize address computation using offset-based flash access helpers (*_AtOffset) instead of stored flash addresses, reducing unsafe pointer arithmetic. NvUpdateSize() now returns uint16_t, added NV_PartitionBlankCheckAtOffset(), and deprecated NvIsMemoryAreaBlank(). Also fixed a data integrity issue in NvSaveAllDataSetEntry() and various Coverity/CERT-C findings.
    * [platform] IFR BLE BD address is now reversed to match the BLE Host stack expectation. The PLATFORM_IFR_BD_ADDR_IS_MSB_FIRST option was removed with the reverse loop reworked accordingly.
    * [OTA] Introduced gOtaEraseWholePartitionOnInit_d option (KW43 only, disabled by default) to erase the whole OTA partition at start of image transfer.
    * [wireless_mcu] Replaced the critical section in PLATFORM_RemoteActiveReq() and PLATFORM_RemoteActiveRel() with a shared mutex to reduce critical section duration. These APIs must not be called from ISR anymore.

    * [wireless_nbu] Updated the low power callback to check for pending RPMSG buffers so the NBU enters WFI only when a Tx message is pending, reducing main core latency.

- **Minor Changes**
    * [platform][zephyr] Zephyr feature flag overrides are now owned by the framework repo in a per platform configs/fwk_config_zephyr.h, included at the top of fwk_config.h under __ZEPHYR__, removing the dual maintenance with the Zephyr integration. Platforms: kw43_mcxw70, kw45_k32w1_mcxw71, kw47_mcxw72, mcxw23, rw61x.
    * [DBG] The NBU debug path now preserves the first fault/assert information: NBUDBG_StateCheck() no longer consumes the NBU debug state before a consumer callback is registered, the sys_debug_panic_triggered guard is hoisted in the hard fault handler so capture, host indication and coredump only run for the first fault, and only the first fault/assert is recorded through the new NBUDBG_RECORD_CLAIMED() macro.
    * [DBG] Added host-triggered NBU force fault for coredump capture through new PLATFORM_ForceNbuFault() and NBUDBG_ForceNbuFault() APIs, signaled to the NBU over MCMGR.
    * [DBG] Deferred the stall HCI event out of NBUDBG_StateCheck(), which now only sets the halted state and notifies the callback. Added new NBUDBG_SendStallEvent() API to emit the stall vendor event on demand.
    * [DBG] Integrated Zephyr coredump capture into the app-core fault handler under CONFIG_DEBUG_COREDUMP.
    * [NVS] Replaced memcpy() by HAL_FlashRead() for internal flash reads so that HAL checks and Async Flash mode synchronization are not bypassed.
    * [wireless_mcu] Added the missing fwk_config.h as first include in the kw43_mcxw70, kw45_k32w1_mcxw71 and kw47_mcxw72 platform files so that feature flag overrides are taken into account.
    * [settings] Added IAR compiler support for iterable sections and disabled the SETTINGS_NAME_END IAR warning.
    * [SecLib_RNG] Added to RNG_psa.c seed support including WorkQueue-based automatic reseeding gRngEnableAutoReseed_d, NBU seed forwarding via PLATFORM_SendRngSeed(), reseed counter logic gRngMaxRequests_d, and new APIs RNG_NotifyReseedNeeded() and RNG_IsReseedNeeded().
    * [OTA] Added a blank check of the OTA partition to avoid unnecessary erase operations and refactored OTA_MakeHeadRoom().
- **Bug Fixes**
    * [WorkQ] Increased the default FWK_SYSWORKQ_STACK_SIZE from 608 to 640 bytes to fix a stack overflow on the system work queue thread.
    * [platform][TSTMR] Fixed the 56 bit version of the timestamp, now reading the TSTMR instance base and testing only the base pointer validity.
    * [wireless_mcu] Fixed FRO6M calibration by replacing FWK_MRCC_TSTMR0_CC/FWK_MRCC_TSTMR0_MUX macros with MRCC_CC/MRCC_MUX from fsl_clock.h and preserving the MUX clock selection in PLATFORM_StartFro6MCalibration().
    * [DBG] Guarded FreeRTOS sys_dump_callstack_ext() with INCLUDE_xTaskGetHandle to fix -Werror=unused-function build failures.
    * [zephyr][lib][crc] Fixed build conflict with EdgeFast OPN CRC by guarding CRC sources with the Zephyr common framework component check to avoid symbol redefinition.

* [NVM] Fixed offset validation in NvGetEntryFromDataPtr() and corrected the bottom record address calculation in NvGetPageFreeSpace() when gUnmirroredFeatureSet_d is undefined.
* [docs] Fixed Sphinx/docutils warnings and errors in the framework documentation.
* [SFC] Prevent unnecessary SFA measurement retrieval when SFC is disabled by adding a check in the SFC ISR to verify that the SFC module is enabled before retrieving the SFA frequency measurement.
* [Coverity] Various Coverity compliance fixes in SecLib (DHKey handling in SecLib_GenerateBluetoothF5KeysSecure()) and OTA (replaced union with structure for callback/argument passing in message buffer).
* [MISRA][CERT-C] Various MISRA, CERT-C and Coverity compliance fixes gathered across platform, LowPower, ICS (wireless_mcu and wireless_nbu), OTA, FSCI, SFC, SecLib and NVM modules.

Known issues

This section lists the known issues, limitations, and/or workarounds.

FRO6M Clock Stability Issue

According to ERRATA ERR052742, the FRO6M clock is not stable on some parts. FRO6M outputs lower frequency signal instead of 6MHz when device is reset or wakes up from low power. It can impact peripherals using it as a clock source.

Impact on TSTMR Module The TSTMR (Time Stamp Timer) module exclusively uses the FRO6M clock source. Due to the aforementioned stability issues, **avoid using TSTMR-related APIs if your application requires high-precision timing**.

Recommendation For applications requiring precise timing, consider using alternative timer modules that support more stable clock sources.

Maximum file path length in Windows 7 operating system

The Windows 7 operating system imposes a 260-character maximum length for file paths. When installing the MCUXpresso SDK, place it in a directory close to the root to prevent file paths from exceeding the maximum character length specified by the Windows operating system. The recommended location is the C:\nxp folder.

Only FreeRTOS is tested for RTOS support

This release only supports the FreeRTOS kernel and a bare-metal non-preemptive task scheduler.

Bluetooth LE

Most sensor applications have pairing and bonding disabled to allow a faster interaction with mobile applications. These two security features can be enabled in the app_preinclude.h header file.

Bluetooth LE controller:

- The maximum Advertising data length is limited to:
  - 800 bytes for KW45/KW47
  - 31 bytes for KW43.
- The scanner may sporadically miss some chained packets.

Periodic Advertising with Responses (PAwR):

- Periodic Advertising with Response (PAwR) is not supported with the configuration "Subevent Interval = Number of Response Slots x Response Slot Spacing with Response Slot Spacing = 0x2".
- The feature is not functional with the Free-Running Oscillator (FRO32K); it requires a 32 KHz Crystal Oscillator with accuracy less than 50 ppm.

KW43:

- CS testmode does not work.
- CS works only over 1M ACL connection.
- CS performances are not yet guaranteed.

KW45 / MCXW 71: No specific issues.

KW47 / MCX W72: Channel Sounding (CS): Information:

- RF Bandwidth Occupancy and Connections/Channel Sounding Activities: The RF bandwidth occupancy is tightly coupled with configured activities, including the number of connections, connection event durations, the number of Channel Sounding (CS) procedures, and related CS parameters (e.g., subevent length, repeat mode). The Link Layer Scheduler is responsible for allocating RF bandwidth to these activities, optimizing bandwidth utilization. However, because each activity operates on its own timing constraints, some RF collisions are expected. This is inherent to the Bluetooth LE protocol, particularly when peripheral roles are configured, as anchor placement is controlled by the central peer devices. The application controls several timing parameters, such as connection interval and subevent length. To optimize user experience, the Link Layer expects the application to adjust these parameters to achieve the best RF bandwidth occupancy with the lowest occurrence of RF collisions.

Limitations:

- RTT with Sounding Sequence is not supported.
- Maximum 6 Channel Sounding procedures are supported in parallel.
- Phase measurement bias is within certification range (<1.7x2πns) with KW47 EVK board. However, if different PCB or antenna matching is used, some bias may appear due to increased delay.
- For experimental feature IPT(Inline PCT Transfer), only 1x1 configuration is supported.

Known issues:

- When CS Subevents are configured very close from each other (<700us), some Subevents may be aborted with reason 0x3.
- When CS offset is configured too close from ACL anchor point, the anchor point may not be served (TX on central or RX on peripheral will not happen). Ideally, CS Offset should be configured greater than 1ms.
- RTT bias compensation:
  - For parts not properly configured at production (IFR blank), RTT bias is not compensated properly. Consequently, an inaccuracy of +/-2m can be observed.

- Pairing procedure fails to complete after connection is established. Affects **digital_key** sample applications. Increase gHost_TaskStackSize_c to 1900U.
- Channel Sounding measurement cannot start after handover. Affects **digital_key** sample applications.
- For the digital_key_car_anchor_cs and digital_key_device_cs BM applications on KW47EVK and KW47LOC, and the loc_reader BM application on KW47EVK, MCXW72EVK, and FRDMMCXW72, increase **stack_size** in board_reconfig.cmake from 0x0CE4 to 0x1194 (4500 bytes). This change is required to avoid runtime stack overflow that can lead to a HardFault.

Other limitations

- The following Connectivity Framework configurations are Experimental and not recommended for mass production:
  - Power down on application power domain.
- GenFSK Connectivity_test application is not operational with Low Power enabled.
- Serial manager is only supported on UART (not I2C nor SPI).
- If the FRO32K is configured as the clock source of the CM33 Core then the debug session will block in IAR CMSIS-DAP while debugging. Use a lower debug wire speed, for example 1 MHz instead of the default one.

  In IAR, the option is in **Runtime Checking** -> **Debugger** -> **CMSIS DAP** -> **Interface** -> **Interface speed**.

1.6 ChangeLog

1.6.1 MCUXpresso SDK Changelog

Board Support Files

board [25.06.00]

- Initial version

clock_config [25.06.00]

- Initial version

pin_mux [25.06.00]

- Initial version

CACHE LPCAC

[2.1.2]

- Improvements
  - Add memory barrier when enabling/disabling cache.

[2.1.1]

- Bug Fixes
  - MISRA C-2012 issue fixed: rule 10.1.

[2.1.0]

- Bug Fixes
  - Deleted L1CACHE_EnableCodeCacheWriteBuffer function because of no enable bit in register CPCR2.

[2.0.0]

- Initial version.

CCM32K

[2.2.2]

- Bug Fixes
  - Fixed MISRA C-2012 Rule 10.1 violation in CCM32K_SetClockMonitorConfig: changed assert(config) to assert(config != NULL).

[2.2.1]

- Bug Fixes
  - Fixed doxygen warning: unbalanced grouping commands.

[2.2.0]

- Improvements
  - Removed wait loops in CCM32K's APIs.

[2.1.1]

- Improvements
  - Updated the judgment conditions in the CCM32K_Enable32kFro() API to avoid code stuck in the while loop.

[2.1.0]

- New Features
  - Added clock monitor control function group and clock gate control function group to support some devices that support clock monitor and clock gate control.

[2.0.0]

- Initial version.

CLOCK

[2.3.1]

- Improvements
  - The SIRC has been removed, but in some low-power wakeup use cases, the SIRC is still required. Updated the clock driver to provide the ability to configure the clock source to SIRC.

[2.3.0]

- Improvements
  - Remove SIRC-related enums, structures, and functions
  - Remove unused clock definitions: kCLOCK_Tstmr0, kCLOCK_Sirc_vsys_gating
  - Remove ERR052742 errata documentation for FRO6M clock stability

[2.2.7]

- Improvements
  - Fixed violations of the MISRA C-2012 Rule 10.1.

[2.2.6]

- Bug Fixes
  - Fixed violations of the CERT INT31-C.

[2.2.5]

- Improvements
  - Added comments for ERR052742: FRO6M clock is not stable.

[2.2.4]

- Improvements
  - Fixed the MSG issue that overrunning array fro192mFreq of 5 4-byte elements at element index 7.

[2.2.3]

- New Features
  - Provide the new API CLOCK_IsFIRCAutoTrimLocked to check whether FIRC auto trim locked to target frequency range.

[2.2.2]

- Bug fix
  - Updated the SDK_DEVICE_MAXIMUM_CPU_CLOCK_FREQUENCY for Core1.

[2.2.1]

- Bug fix
  - Fixed the GPIO_PinInit function may cause the hardfault in clock driver when pass the GPIOD as the fist parameter.

[2.2.0]

- Improvements
  - Added the support for TPM2.

[2.1.0]

- Improvements
  - Provide the FSL_SDK_FORCE_CLK_DRIVER_NS_ACCESS macro to force use the none secure address even the trustzone secure mode is enabled.

[2.0.0]

- Fixed the MISRA issues.

[1.1.0]

- Some minor fixes.

[1.0.0]

- Initial version.

CMC

[2.4.4]

- Bug Fixes
  - Fixed doxygen warning: unbalanced grouping commands.

[2.4.3]

- Improvements
  - Add timeout for while loop code.

[2.4.2]

- Improvements
  - Add support for specific KW4x device variants
    - Let kCMC_SecurityViolationReset depend on whether CMC_SRS_SECVIO_MASK definition exists or not.
    - CMC_LockWriteOperationToBootRomStatusReg and CMC_CheckBootRomStatusRegWriteLocked exist only if FSL_FEATURE_CMC_HAS_NO_BOOTROM_LOCK_REGISTER is not defined

[2.4.1]

- Improvements
  - Clear SCB SCR[SLEEPDEEP] bitfield after wakeup.

[2.4.0]

- New Features
  - Added new interface to support CMC[DIER] register.
- Improvements
  - Updated _cmc_system_sram_arrays enumeration to support some devices that provide more sram bank.

[2.3.1]

- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 10.7;

[2.3.0]

- Improvements
  - Added new functions to support controls of BLR register.
  - Updated CMC_GetBootRomStatus() to support some devices that equipped multiple BootROM status registers.
  - Added CMC_WriteBootRomStatusReg() to write BootRom Status register.

[2.2.1]

- Improvements
  - For some devices, SRS_JTAG bit is reserved. Added a feature macro to adapt to different devices.

[2.2.0]

- Improvements
  - Updated _cmc_system_sram_arrays enumeration, make it more universal.
  - Updated SRAM related APIs(CMC_PowerOffSRAMAllMode(), CMC_PowerOffSRAMLowPowerOnly(), CMC_PowerOnSRAMAllMode(), CMC_PowerOnSRAMLowPowerOnly()), due to updates of registers' names.
  - Renamed CMC_GetBootConfigPinLogic() to CMC_GetISPMODEPinLogic().
  - Renamed CMC_ClearBootConfig() to CMC_ClearISPModePinLogic().
  - Updated enumeration _cmc_power_mode_protection due to some macros are deleted in header file.

[2.1.0]

- Improvements
  - Added some macros to separate the scenes that some registers are reserved in some devices.

[2.0.0]

- Initial version.

COMMON

[2.6.7]

- Improvements
  - EnableIRQ/DisableIRQ: Use IRQSTEER_GetInstIdxByIRQ() to determine IRQSTEER instance index instead of simple division. Supports multiple IRQSTEER instances with different interrupt counts.

[2.6.6]

- Improvements
  - Support Cortex M55.

[2.6.5]

- Bug Fixes
  - Fixed the issue that the function SDK_DelayAtLeastUs may underflow and cause a long delay when the input parameter is small and the system clock is low.

[2.6.4]

- New Features
  - Add macros to tag deprecated features and experimental features.

[2.6.3]

- New Features
  - Added bit mask inversion macros to avoid type promotion.
  - Added register operation macros.
- Improvements
  - Make function MSDK_EnableCpuCycleCounter compatible with CMSIS-5 and CMSIS-6.
- Bug Fixes
  - Fixed build issue of CMSIS PACK BSP example caused by CMSIS 6.1 issue.

[2.6.2]

- Bug Fixes
  - Fixed violations of MISRA C-2012 rule for implicit conversions in boolean contexts

[2.6.1]

- Improvements
  - Support Cortex M23.

[2.6.0]

- Bug Fixes
  - Fix CERT-C violations.

[2.5.0]

- New Features
  - Added new APIs InitCriticalSectionMeasurementContext, DisableGlobalIRQEx and EnableGlobalIRQEx so that user can measure the execution time of the protected sections.

[2.4.3]

- Improvements
  - Enable irqs that mount under irqsteer interrupt extender.

[2.4.2]

- Improvements
  - Add the macros to convert peripheral address to secure address or non-secure address.

[2.4.1]

- Improvements
  - Improve for the macro redefinition error when integrated with zephyr.

[2.4.0]

- New Features
  - Added EnableIRQWithPriority, IRQ_SetPriority, and IRQ_ClearPendingIRQ for ARM.
  - Added MSDK_EnableCpuCycleCounter, MSDK_GetCpuCycleCount for ARM.

[2.3.3]

- New Features
  - Added NETC into status group.

[2.3.2]

- Improvements
  - Make driver aarch64 compatible

[2.3.1]

- Bug Fixes
  - Fixed MAKE_VERSION overflow on 16-bit platforms.

[2.3.0]

- Improvements
  - Split the driver to common part and CPU architecture related part.

[2.2.10]

- Bug Fixes
  - Fixed the ATOMIC macros build error in cpp files.

[2.2.9]

- Bug Fixes
  - Fixed MISRA C-2012 issue, 5.6, 5.8, 8.4, 8.5, 8.6, 10.1, 10.4, 17.7, 21.3.
  - Fixed SDK_Malloc issue that not allocate memory with required size.

[2.2.8]

- Improvements
  - Included stddef.h header file for MDK tool chain.
- New Features:
  - Added atomic modification macros.

[2.2.7]

- Other Change
  - Added MECC status group definition.

[2.2.6]

- Other Change
  - Added more status group definition.
- Bug Fixes
  - Undef __VECTOR_TABLE to avoid duplicate definition in cmsis_clang.h

[2.2.5]

- Bug Fixes
  - Fixed MISRA C-2012 rule-15.5.

[2.2.4]

- Bug Fixes
  - Fixed MISRA C-2012 rule-10.4.

[2.2.3]

- New Features
  - Provided better accuracy of SDK_DelayAtLeastUs with DWT, use macro SDK_DELAY_USE_DWT to enable this feature.
  - Modified the Cortex-M7 delay count divisor based on latest tests on RT series boards, this setting lets result be closer to actual delay time.

[2.2.2]

- New Features
  - Added include RTE_Components.h for CMSIS pack RTE.

[2.2.1]

- Bug Fixes
  - Fixed violation of MISRA C-2012 Rule 3.1, 10.1, 10.3, 10.4, 11.6, 11.9.

[2.2.0]

- New Features
  - Moved SDK_DelayAtLeastUs function from clock driver to common driver.

[2.1.4]

- New Features
  - Added OTFAD into status group.

[2.1.3]

- Bug Fixes
  - MISRA C-2012 issue fixed.
    - Fixed the rule: rule-10.3.

[2.1.2]

- Improvements
  - Add SUPPRESS_FALL_THROUGH_WARNING() macro for the usage of suppressing fallthrough warning.

[2.1.1]

- Bug Fixes
  - Deleted and optimized repeated macro.

[2.1.0]

- New Features
  - Added IRQ operation for XCC toolchain.
  - Added group IDs for newly supported drivers.

[2.0.2]

- Bug Fixes
  - MISRA C-2012 issue fixed.
    - Fixed the rule: rule-10.4.

[2.0.1]

- Improvements
  - Removed the implementation of LPC8XX Enable/DisableDeepSleepIRQ() function.
  - Added new feature macro switch "FSL_FEATURE_HAS_NO_NONCACHEABLE_SECTION" for specific SoCs which have no noncacheable sections, that helps avoid an unnecessary complex in link file and the startup file.
  - Updated the align(x) to **attribute**(aligned(x)) to support MDK v6 armclang compiler.

[2.0.0]

- Initial version.

CRC

[2.1.0]

- Bug fix:
  - Choosing CRC clocks from CRC clock array according to instance instead of hardcoded value.

[2.0.5]

- Bug fix:
  - Fix CERT-C issue with boolean-to-unsigned integer conversion.

[2.0.4]

- Improvements
  - Release peripheral from reset if necessary in init function.

[2.0.3]

- Bug fix:
  - Fix MISRA issues.

[2.0.2]

- Bug fix:
  - Fix MISRA issues.

[2.0.1]

- Bug fix:
  - DATA and DATALL macro definition moved from header file to source file.

[2.0.0]

- Initial version.

EDMA (DMA3)

[2.6.1]

- Bug Fixes
  - Fixed coverity issues with CERT INT31-C compliance.
  - Fixed violations of MISRA C-2012 rule 10.8.

[2.6.0]

- Improvements
  - Add API EDMA_EnableChannelMasterIDReplication, EDMA_SetChannelSecurityLevel and EDMA_SetChannelProtectionLevel to support edma CH_SBR register configuration.
  - Add FSL_FEATURE_EDMA_SOC_REQUIRES_SECURE_MASTER macro to separate SOC with secure master requirement.

[2.5.3]

- Bug Fixes
  - Fixed coverity issues with CERT INT30-C, CERT INT31-C compliance.
  - Fixed violations of MISRA C-2012 rule 10.4, 8.5.

[2.5.2]

- Bug Fixes
  - Fixed the EDMA header index retrieval error caused by done bit calculation mistake issue.

[2.5.1]

- Bug Fixes
  - enables the auto stop request feature in API EDMA_ResetChannel

[2.5.0]

- Improvements
  - Add API EDMA_GetTransferSize for EDMA_PrepareTransferConfig to reduce HIS CCM value.

[2.4.0]

- Improvements
  - Added peripheral to peripheral support in DMA3 driver.
- Bug Fixes
  - Fixed the ERQ bit reading error issue.

[2.3.2]

- Improvements
  - Conditionally compile interrupt handling code to solve the problem of using this driver on CPU cores that do not support interrupts.

[2.3.1]

- Bug Fixes
  - Added clear TCD_CITER_ELINKNO and TCD_BITER_ELINKNO registers in EDMA_AbortTransfer to make sure the TCD registers in a correct state for next calling of EDMA_SubmitTransfer.

[2.3.0]

- Improvements
  - Added feature FSL_FEATURE_EDMA_HAS_NO_SBR_ATTR_BIT to separate DMA without ATTR bitfield.

[2.2.7]

- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 10.8, 5.6.

[2.2.6]

- Bug Fixes
  - Fixed the TCD overwrite issue when submit transfer request in the callback if there is a active TCD in hardware.

[2.2.5]

- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 10.1, 10.4.

[2.2.4]

- Bug Fixes
  - Fix the issue that EDMA_AbortTransfer not reset edma handle(tcdUsed, tail, header) and fix EDMA_InstallTCDMemory(handle->header = 1)

[2.2.3]

- Improvements
  - Added feature FSL_FEATURE_EDMA_MODULE_CHANNEL_IRQ_ENTRY_SUPPORT_PARAMETER to support driver IRQ handler with parameters.
  - Added feature FSL_FEATURE_EDMA_HAS_COMMON_CLOCK_GATE to improve clock gate control in dma driver.

[2.2.2]

- Bug Fixes
  - Fixed the issue of EDMA_SubmitTransfer return busy when calling EDMA_EnableChannelInterrupts before submit transfer.
  - Fixed violations of MISRA C-2012 rule 10.4, 10.1, 9.2, 10.4, 10.6, 14.4, 10.7, 14.3, 11.6.

[2.2.1]

- Improvements
  - Removed channel MUX reset from EDMA_ResetChannel, since channel mux should be constant while channel is alive.

[2.2.0]

- Improvements
  - Added new API EDMA_SetChannelMux to support channel mux feature.
  - Added new API EDMA_PrepareTransferConfig to expose paramters source offset and destination offset.
  - Exposed EDMA_InstallTCD function to application.
  - Added source/destination address alignment check.

[2.1.1]

- Improvements
  - Added 8bytes transfer width feature support in driver.

[2.1.0]

- Bug Fixes
  - Added const type for parameter configuration in EDMA_SubmitTransfer and EDMA_HandleTransferConfig API.
  - Added configurations for srcAddr and destAddr in EDMA_PrepareTransfer API.

[2.0.2]

- Improvements
  - Updated eDMA driver to support MP_CR bit GMRC.
  - Updated eDMA instance name for i.MX 8QM.
  - Used instance number as factor to calculate channel number for different instance instead of hard code.

[2.0.1]

- New Features
  - Added control macro to enable/disable the CLOCK code in current driver.
  - Added s_EDMAEnabledChannel to record enabled channel to merge all the channel IRQ handler into driver IRQ handler.
  - Added feature macro for bits EMI and EBW in MP_CSR.
- Improvements
  - Removed all the separated channel IRQ handler in DMA driver.

[2.0.0]

- Initial version.

ELEMU

- 2.1.3 Fix CERT INT30-C and CERT INT31-C findings.
- 2.1.2 Fix macro BIT redefined issue when compiling with Zephyr.
- 2.1.1 Fix MISRA issues.
- 2.1.0 Cleanup of unused legacy FW. Updated ELEMU_loadFw function.
- 2.0.0 Renamed MU to ELEMU to avoid confusion with other Messaging units.
- 0.0.4 Updated register name according to latest device header file.
- 0.0.3 Fix MISRA issues.
- 0.0.2 The name of SNT has changed to ELEMU.
- 0.0.1 Initial version of SNT driver.

EWM

[2.0.4]

- Bug Fixes
  - Fixed CERT INT31-C violations.

[2.0.3]

- Bug Fixes
  - Fixed violation of MISRA C-2012 rules: 10.1, 10.3.

[2.0.2]

- Bug Fixes
  - Fixed violation of MISRA C-2012 rules: 10.3, 10.4.

[2.0.1]

- Bug Fixes
  - Fixed the hard fault in EWM_Deinit.

[2.0.0]

- Initial version.

FLASH

[2.4.2]

- Improvements
  - add non-blocking try-lock in async flash process

[2.4.1]

- Improvement
  - Fixed MISRA C-2012 rule 8.6, 21.15 and 11.6.
  - Fixed MSG CERT INT30-C.

[2.4.0]

- New Features
  - Add asynchronous flash mode support for non-blocking flash operations. Async mode allows applications to initiate flash operations and continue execution while the flash controller completes the operation in the background.

[2.3.5]

- Improvement
  - Add flash_cache_disable implementation for KW45, KW47 platforms. In the absence of flash cache disable functionality, the flash module will use software workaround to prevent stale data from being returned, by disabling the cache and re-enabling it.

[2.3.4]

- Improvement
  - Add the support for KW43/MCXW70 platforms.

[2.3.3]

- Improvements
  - Added timeout for while loop in flash_command_pre_sequence, flash_command_sequence, flash_erase_sequence, flash_pgm_sequence and flash_command_pre_sequence API.

[2.3.2]

- Improvement
- Enabled the kFLASH_PropertyPflash1SectorSize property support in FLASH_GetProperty API.

[2.3.1]

- Improvement
- Clear flash cache before every erase to prevent the possibility of returning stale data.

[2.3.0]

- New features
- Add support for MCX N series

[2.2.1]

- Improvement
  - To fix the potential unsigned integer overflow in the operation, added a check to ensure that the addition does not exceed the maximum value.

[2.2.0]

- New Features
  - Add the support for MW30.

[2.1.2]

- Improvement
  - Add the support for KW47/MCXW72 phantoms.

[2.1.1]

- Improvement
  - Add the conditional compiling flag '#if defined(RF_FMU)' to make the driver be compatible with the non-radio phantoms.

[2.1.0]

- Bug Fixes
  - Fix flash driver run error in flash memory

[2.0.0]

- Initial version.

FLEXCAN

[2.16.0]

- Improvements
  - Added four new members to flexcan_config_t so that the previously hard-coded register values can now be configured by the application:
    - enableThreeSamplingMode - controls CTRL1[SMP]; default true (backward compatible).
    - disableBusOffAutoRecovery - controls CTRL1[BOFFREC]; default false (backward compatible).
    - txArbitrationStartDelay - controls CTRL2[TASD] (0–31 Tq); default 0x16 (backward compatible).
    - enableEntireArbitrationCompare - controls CTRL2[EACEN]; default true (backward compatible).
  - Remove above unconditionally overwritten hard-coded registers value from FLEXCAN_Reset.

[2.15.2]

- Bug Fixes
  - Fixed violations of the CERT INT31-C.

[2.15.1]

- Improvements
  - Wait for RX Message Buffer busy state cleared before reading in following APIs.
    - FLEXCAN_ReadRxMb
    - FLEXCAN_ReadFDRxMb
  - Simplify RX Message Buffer handling sequence in interrupt handler FLEXCAN_SubHandlerForMB.
  - Aligned watermark comparison logic between FLEXCAN_EnhancedRxFifoHandleIRQ and FLEXCAN_TransferReceiveEnhancedFifoNonBlocking for consistency.
  - Fixed Ehanced typo issue.
  - Remove redundant feature macro for FlexCAN engine clock selection in FLEXCAN_Init.
- Bug Fixes
  - Fixed Enhanced Rx FIFO filter configuration assertion idFilterPairNum in FLEXCAN_SetEnhancedRxFifoConfig.
  - Fixed Bus Off Done interrupt and Data Phase of Fast CAN FD Frames Error interrupt enable mask shift for CTRL2 register.
  - Fixed Enhanced Rx FIFO interrupt enable and status flag comparing issue in FLEXCAN_EnhancedRxFifoHandleIRQ.

[2.15.0]

- Improvements
  - Add remote request and response non-blocking transfer APIs.
    - FLEXCAN_TransferRemoteRequestNonBlocking
    - FLEXCAN_TransferRemoteResponseNonBlocking
- Bug Fixes
  - Fixed improper CS code when configuring FlexCAN Remote Response Message Buffer in FLEXCAN_SetRemoteResponseMbConfig.

[2.14.7]

- Improvements
  - Add Coverage Justification for uncovered code.
- Bug Fixes
  - Fixed violations of the CERT INT30-C INT31-C.

[2.14.6]

- Bug Fixes
  - Fixed MCR register reconfiguration failure after FlexCAN soft reset on some platforms.

[2.14.5]

- Improvements
  - Make API FLEXCAN_GetFDMailboxOffset public.
  - Add API FLEXCAN_SetMbID and FLEXCAN_SetFDMbID to configure Message Buffer ID individually.
- Bug Fixes
  - Fixed violations of the CERT INT30-C INT31-C.
  - Fixed violations of the CERT ARR30-C.

[2.14.4]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 8.4, 10.1, 10.4, 18.1.

[2.14.3]

- Improvements
  - Add unhandled interrupt events check for following API:
    - FLEXCAN_MbHandleIRQ
    - FLEXCAN_EhancedRxFifoHandleIRQ
- Bug Fixes
  - Remove FLEXCAN_MemoryErrorHandleIRQ on some platform without memory error interrupt.

  - Add conditional compile for CTRL2[ISOCANFDEN] because some platform do not have this bit.

[2.14.2]

- Improvements
  - Add Coverage Justification for uncovered code.
  - Adjust API FLEXCAN_TransferAbortReceive order.
  - Update FLEXCAN_Enable to enter Freeze Mode first when enter Disable mode on some platform.
  - Added while loop timeout for following API:
    * FLEXCAN_EnterFreezeMode
    * FLEXCAN_ExitFreezeMode
    * FLEXCAN_Enable
    * FLEXCAN_Reset
    * FLEXCAN_TransferSendBlocking
    * FLEXCAN_TransferReceiveBlocking
    * FLEXCAN_TransferFDSendBlocking
    * FLEXCAN_TransferFDReceiveBlocking
    * FLEXCAN_TransferReceiveFifoBlocking
    * FLEXCAN_TransferReceiveEnhancedFifoBlocking
- Bug Fixes
  - Remove remote frame feature in CANFD mode because there is no remote frame in the CANFD format.
  - Remove legacy Rx FIFO disabled branch in FLEXCAN_SubHandlerForLegacyRxFIFO and FLEXCAN_SubHandlerForDataTransfered.

[2.14.1]

- Bug Fixes
  - Fixed register IMASK2-4 IFLAG2-4 HR_TIME_STAMPn access issue on FlexCAN instances with different number of MBs.
  - Fixed bit field MBDSR1-3 access issue on FlexCAN instances with different number of MBs.
- Improvements
  - Unified following API as same parameter and return value type:
    * FLEXCAN_GetMbStatusFlags
    * FLEXCAN_ClearMbStatusFlags
    * FLEXCAN_EnableMbInterrupts
    * FLEXCAN_DisableMbInterrupts
  - Add workaround for ERR050443 and ERR052403.
  - Update message buffer read process in API FLEXCAN_ReadRxMb and FLEXCAN_ReadFDRxMb to make critical section as short as possible.

  - Simplify API FLEXCAN_DriverDataIRQHandler implementation by remove parameter type.

[2.14.0]

- Improvements
  - Support external time tick feature.
  - Support high resolution timestamp feature.
  - Enter Freeze Mode first when enter Disable Mode on some platform.
  - Add feature macro for Pretended Networking because some FlexCAN instance do not have this feature.
  - Add feature macro for enhanced Rx FIFO because some FlexCAN instance do not have this feature.
  - Add new FlexCAN IRQ Handler FLEXCAN_DriverDataIRQHandler and FLEXCAN_DriverEventIRQHandler. Thses IRQ Handlers are used on soc which FlexCAN interrupts are grouped by specific function and assigned to different vector.
  - Update macro FLEXCAN_WAKE_UP_FLAG and FLEXCAN_PNWAKE_UP_FLAG to simplify code.
  - Replace macro FSL_FEATURE_FLEXCAN_HAS_NO_WAKMSK_SUPPORT with FSL_FEATURE_FLEXCAN_HAS_NO_SLFWAK_SUPPORT.
  - Replace macro FSL_FEATURE_FLEXCAN_HAS_NO_WAKSRC_SUPPORT with FSL_FEATURE_FLEXCAN_HAS_GLITCH_FILTER.
- Bug Fixes
  - Fixed wrong interrupt and status flag helper macro in enumeration _flexcan_flags and API FLEXCAN_DisableInterrupts.
  - Fixed interrupt flag helper macro typo issue.
  - Remove flags which will are unassociated with interrupt in macro FLEXCAN_MEMORY_ERROR_INT_FLAG.
  - Remove flags which will are unassociated with interrupt in macro FLEXCAN_ERROR_AND_STATUS_INT_FLAG.
  - Fixed array out-of-bounds access when read enhanced Rx FIFO.

[2.13.1]

- Improvements
  - Conditionally compile interrupt handling code to solve the problem of using this driver on CPU cores that do not support interrupts.

[2.13.0]

- Improvements
  - Support payload endianness selection feature.

[2.12.0]

- Improvements
  - Support automatic Remote Response feature.
  - Add API FLEXCAN_SetRemoteResponseMbConfig() to configure automatic Remote Response mailbox.

[2.11.8]

- Improvements
  - Synchronize flexcan driver update on s32z platform.

[2.11.7]

- Bug Fixes
  - Fixed FLEXCAN_TransferReceiveEnhancedFifoEDMA() compatibility with edma5.

[2.11.6]

- Bug Fixes
  - Fixed ERRATA_9595 FLEXCAN_EnterFreezeMode() may result to bus fault on some platform.

[2.11.5]

- Bug Fixes
  - Fixed flexcan_memset() crash under high optimization compilation.

[2.11.4]

- Improvements
  - Update CANFD max bitrate to 10Mbps on MCXNx3x and MCXNx4x.
  - Release peripheral from reset if necessary in init function.

[2.11.3]

- Bug Fixes
  - Fixed FLEXCAN_TransferReceiveEnhancedFifoEDMA() compile error with DMA3.

[2.11.2]

- Bug Fixes
  - Fixed bug that timestamp in flexcan_handle_t not updated when RX overflow happens.

[2.11.1]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 10.1.

[2.11.0]

- Bug Fixes
  - Fixed wrong base address argument in FLEXCAN2 IRQ Handler.
- Improvements
  - Add API to determine if the instance supports CAN FD mode at run time.

[2.10.1]

- Bug Fixes
  - Fixed HIS CCM issue.
  - Fixed RTOS issue by adding protection to read-modify-write operations on interrupt enable/disable API.

[2.10.0]

- Improvements
  - Update driver to make it able to support devices which has more than 64 8bytes MBs.
  - Update CAN FD transfer APIs to make them set/get edl bit according to frame content, which can make them compatible with classic CAN.

[2.9.2]

- Bug Fixes
  - Fixed the issue that FLEXCAN_CheckUnhandleInterruptEvents() can't detecting the exist enhanced RX FIFO interrupt status.
  - Fixed the issue that FLEXCAN_ReadPNWakeUpMB() does not return fail even no existing valid wake-up frame.
  - Fixed the issue that FLEXCAN_ReadEnhancedRxFifo() may clear bits other than the data available bit.
  - Fixed violations of the MISRA C-2012 rules 10.4, 10.8.
- Improvements
  - Return kStatus_FLEXCAN_RxFifoDisabled instead of kStatus_Fail when read FIFO fail during IRQ handler.
  - Remove unreachable code from timing calculates APIs.
  - Update Enhanced Rx FIFO handler to make it deal with underflow/overflow status first.

[2.9.1]

- Bug Fixes
  - Fixed the issue that FLEXCAN_TransferReceiveEnhancedFifoBlocking() API clearing Fifo data available flag more than once.
  - Fixed the issue that entering FLEXCAN_SubHandlerForEhancedRxFifo() even if Enhanced Rx fifo interrupts are not enabled.
  - Fixed the issue that FLEXCAN_TransferReceiveEnhancedFifoEDMA() update handle even if previous Rx FIFO receive not finished.

  - Fixed the issue that FLEXCAN_SetEnhancedRxFifoConfig() not configure the ERFCR[NFE] bits to the correct value.
  - Fixed the issue that FLEXCAN_ReceiveFifoEDMACallback() can't differentiate between Rx fifo and enhanced rx fifo.
  - Fixed the issue that FLEXCAN_TransferHandleIRQ() can't report Legacy Rx FIFO warning status.

[2.9.0]

- Improvements
- Add public set bit rate API to make driver easier to use.
- Update Legacy Rx FIFO transfer APIs to make it support received multiple frames during one API call.
- Optimized FLEXCAN_SubHandlerForDataTransfered() API in interrupt handling to reduce the probability of packet loss.

[2.8.7]

- Improvements
- Initialized the EDMA configuration structure in the FLEXCAN EDMA driver.

[2.8.6]

- Bug Fixes
- Fix Coverity overrun issues in fsl_flexcan_edma driver.

[2.8.5]

- Improvements
  - Make driver aarch64 compatible.

[2.8.4]

- Bug Fixes
  - Fixed FlexCan_Errata_6032 to disable all interrupts.

[2.8.3]

- Bug Fixes
  - Fixed an issue with the FLEXCAN_EnableInterrupts and FLEXCAN_DisableInterrupts interrupt enable bits in the CTRL1 register.

[2.8.2]

- Bug Fixes
  - Fixed errors in timing calculations and simplify the calculation process.
  - Fixed issue of CBT and FDCBT register may write failure.

[2.8.1]

- Bug Fixes
  - Fixed the issue of CAN FD three sampling points.
  - Added macro to support the devices that no MCR[SUPV] bit.
  - Remove unnecessary clear WMB operations.

[2.8.0]

- Improvements
  - Update config configuration.
    - Added enableSupervisorMode member to support enable/disable Supervisor mode.
  - Simplified the algorithm in CAN FD improved timing APIs.

[2.7.1]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 10.3, 10.7.

[2.7.0]

- Improvements
  - Update config configuration.
    - Added enablePretendedeNetworking member to support enable/disable Pretended Networking feature.
    - Added enableTransceiverDelayMeasure member to support enable/disable Transceiver Delay MeasurementPretended feature.
    - Added bitRate/bitRateFD member to work as baudRate/baudRateFD member union.
  - Rename all “baud” in code or comments to “bit” to align with the CAN spec.
  - Added Pretended Networking mode related APIs.
    - FLEXCAN_SetPNConfig
    - FLEXCAN_GetPNMatchCount
    - FLEXCAN_ReadPNWakeUpMB
  - Added support for Enhanced Rx FIFO.
  - Removed independent memory error interrupt/status APIs and put all interrupt/status control operation into FLEXCAN_EnableInterrupts/FLEXCAN_DisableInterrupts and FLEXCAN_GetStatusFlags/FLEXCAN_ClearStatusFlags APIs.
  - Update improved timing APIs to make it calculate improved timing according to CiA doc recommended.
    - FLEXCAN_CalculateImprovedTimingValues.
    - FLEXCAN_FDCalculateImprovedTimingValues.
  - Update FLEXCAN_SetBitRate/FLEXCAN_SetFDBitRate to added the use of enhanced timing registers.

[2.6.2]

- Improvements
  - Add CANFD frame data length enumeration.

[2.6.1]

- Bug Fixes
  - Fixed the issue of not fully initializing memory in FLEXCAN_Reset() API.

[2.6.0]

- Improvements
  - Enable CANFD ISO mode in FLEXCAN_FDInit API.
  - Enable the transceiver delay compensation feature when enable FD operation and set bitrate switch.
  - Implementation memory error control in FLEXCAN_Init API.
  - Improve FLEXCAN_FDCalculateImprovedTimingValues API to get same value for FPRESDIV and PRESDIV.
  - Added memory error configuration for user.
    * enableMemoryErrorControl
    * enableNonCorrectableErrorEnterFreeze
  - Added memory error related APIs.
    * FLEXCAN_GetMemoryErrorReportStatus
    * FLEXCAN_GetMemoryErrorStatusFlags
    * FLEXCAN_ClearMemoryErrorStatusFlags
    * FLEXCAN_EnableMemoryErrorInterrupts
    * FLEXCAN_DisableMemoryErrorInterrupts
- Bug Fixes
  - Fixed the issue of sent duff CAN frame after call FLEXCAN_FDInit() API.

[2.5.2]

- Bug Fixes
  - Fixed the code error issue and simplified the algorithm in improved timing APIs.
    * The bit field in CTRL1 register couldn't calculate higher ideal SP, we set it as the lowest one(75%)
      · FLEXCAN_CalculateImprovedTimingValues
      · FLEXCAN_FDCalculateImprovedTimingValues
  - Fixed MISRA-C 2012 Rule 17.7 and 14.4.
- Improvements
  - Pass EsrStatus to callback function when kStatus_FLEXCAN_ErrorStatus is comming.

[2.5.1]

- Bug Fixes
  - Fixed the non-divisible case in improved timing APIs.
    - FLEXCAN_CalculateImprovedTimingValues
    - FLEXCAN_FDCalculateImprovedTimingValues

[2.5.0]

- Bug Fixes
  - MISRA C-2012 issue check.
    - Fixed rules, containing: rule-10.1, rule-10.3, rule-10.4, rule-10.7, rule-10.8, rule-11.8, rule-12.2, rule-13.4, rule-14.4, rule-15.5, rule-15.6, rule-15.7, rule-16.4, rule-17.3, rule-5.8, rule-8.3, rule-8.5.
  - Fixed the issue that API FLEXCAN_SetFDRxMbConfig lacks inactive message buff.
  - Fixed the issue of Pa082 warning.
  - Fixed the issue of dead lock in the function of interruption handler.
  - Fixed the issue of Legacy Rx Fifo EDMA transfer data fail in evkmimxrt1060 and evkmimxrt1064.
  - Fixed the issue of setting CANFD Bit Rate Switch.
  - Fixed the issue of operating unknown pointer risk.
    - when used the pointer "handle->mbFrameBuf[mbIdx]" to update the timestamp in a short-live TX frame, the frame pointer became as unknown, the action of operating it would result in program stack destroyed.
  - Added assert to check current CAN clock source affected by other clock gates in current device.
    - In some chips, CAN clock sources could be selected by CCM. But for some clock sources affected by other clock gates, if user insisted on using that clock source, they had to open these gates at the same time. However, they should take into consideration the power consumption issue at system level. In RT10xx chips, CAN clock source 2 was affected by the clock gate of lpuart1. ERRATA ID: (ERR050235 in CCM).
- Improvements
  - Implementation for new FLEXCAN with ECC feature able to exit Freeze mode.
  - Optimized the function of interruption handler.
  - Added two APIs for FLEXCAN EDMA driver.
    - FLEXCAN_PrepareTransfConfiguration
    - FLEXCAN_StartTransferDatafromRxFIFO
  - Added new API for FLEXCAN driver.
    - FLEXCAN_GetTimeStamp
      - For TX non-blocking API, we wrote the frame into mailbox only, so no need to register TX frame address to the pointer, and the timestamp could be updated into the new global variable handle->timestamp[mbIdx], the FLEXCAN driver provided a new API for user to get it by handle and index number after TX DONE Success.
    - FLEXCAN_EnterFreezeMode

    * FLEXCAN_ExitFreezeMode
  - Added new configuration for user.
    * disableSelfReception
    * enableListenOnlyMode
  - Renamed the two clock source enum macros based on CLKSRC bit field value directly.
    * The CLKSRC bit value had no property about Oscillator or Peripheral type in lots of devices, it acted as two different clock input source only, but the legacy enum macros name contained such property, that misled user to select incorrect CAN clock source.
  - Created two new enum macros for the FLEXCAN driver.
    * kFLEXCAN_ClkSrc0
    * kFLEXCAN_ClkSrc1
  - Deprecated two legacy enum macros for the FLEXCAN driver.
    * kFLEXCAN_ClkSrcOsc
    * kFLEXCAN_ClkSrcPeri
  - Changed the process flow for Remote request frame response..
    * Created a new enum macro for the FLEXCAN driver.
      · kStatus_FLEXCAN_RxRemote
  - Changed the process flow for kFLEXCAN_StateRxRemote state in the interrupt handler.
    * Should the TX frame not register to the pointer of frame handle, interrupt handler would not be able to read the remote response frame from the mail box to ram, so user should read the frame by manual from mail box after a complete remote frame transfer.

[2.4.0]

- Bug Fixes
  - MISRA C-2012 issue check.
    * Fixed rules, containing: rule-12.1, rule-17.7, rule-16.4, rule-11.9, rule-8.4, rule-14.4, rule-10.8, rule-10.4, rule-10.3, rule-10.7, rule-10.1, rule-11.6, rule-13.5, rule-11.3, rule-8.3, rule-12.2 and rule-16.1.
  - Fixed the issue that CANFD transfer data fail when bus baudrate is 30Khz.
  - Fixed the issue that ERR009595 does not folllow the ERRATA document.
  - Fixed code error for ERR006032 work around solution.
  - Fixed the Coverity issue of BAD_SHIFT in FLEXCAN.
  - Fixed the Repo build warning issue for variable without initial.
- Improvements
  - Fixed the run fail issue of FlexCAN RemoteRequest UT Case.
  - Implementation all TX and RX transfering Timestamp used in FlexCAN demos.
  - Fixed the issue of UT Test Fail for CANFD payload size changed from 64BperMB to 8PerMB.
  - Implementation for improved timing API by baud rate.

[2.3.2]

- Improvements
  - Implementation for ERR005959.
  - Implementation for ERR005829.
  - Implementation for ERR006032.

[2.3.1]

- Bug Fixes
  - Added correct handle when kStatus_FLEXCAN_TxSwitchToRx is comming.

[2.3.0]

- Improvements
  - Added self-wakeup support for STOP mode in the interrupt handling.

[2.2.3]

- Bug Fixes
  - Fixed the issue of CANFD data phase's bit rate not set as expected.

[2.2.2]

- Improvements
  - Added a time stamp feature and enable it in the interrupt_transfer example.

[2.2.1]

- Improvements
  - Separated CANFD initialization API.
  - In the interrupt handling, fix the issue that the user cannot use the normal CAN API when with an FD.

[2.2.0]

- Improvements
  - Added FSL_FEATURE_FLEXCAN_HAS_SUPPORT_ENGINE_CLK_SEL_REMOVE feature to support SoCs without CAN Engine Clock selection in FlexCAN module.
  - Added FlexCAN Serial Clock Operation to support i.MX SoCs.

[2.1.0]

- Bug Fixes
  - Corrected the spelling error in the function name FLEXCAN_XXX().
  - Moved Freeze Enable/Disable setting from FLEXCAN_Enter/ExitFreezeMode() to FLEXCAN_Init().
  - Corrected wrong helper macro values.

- Improvements
  - Hid FLEXCAN_Reset() from user.
  - Used NDEBUG macro to wrap FLEXCAN_IsMbOccupied() function instead of DEBUG macro.

[2.0.0]

- Initial version.

FLEXCAN_EDMA

[2.12.1]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 18.1.

[2.12.0]

- Improvements
  - Support high resolution timestamp feature in enhanced Rx FIFO EDMA.
  - Add feature macro for enhanced Rx FIFO because some FlexCAN instance do not have this feature.
- Bug Fixes
  - Fixed array out-of-bounds access when read enhanced Rx FIFO in EDMA.

[2.11.7]

- Refer FLEXCAN driver change log 2.7.0 to 2.11.7

FLEXIO

[2.3.2]

- Bug Fixes
  - Fixed MISRA 10.4 issues.

[2.3.1]

- New Features
  - Added common IRQ handler entry FLEXIO_CommonDriverIRQHandler.

[2.3.0]

- Improvements
  - Supported platforms which don't have DOZE mode control.
  - Added more pin control functions.

[2.2.3]

- Improvements
  - Adapter the FLEXIO driver to platforms which don't have system level interrupt controller, such as NVIC.

[2.2.2]

- Improvements
  - Release peripheral from reset if necessary in init function.

[2.2.1]

- Improvements
  - Added doxygen index parameter comment in FLEXIO_SetClockMode.

[2.2.0]

- New Features
  - Added new APIs to support FlexIO pin register.

[2.1.0]

- Improvements
  - Added API FLEXIO_SetClockMode to set flexio channel counter and source clock.

[2.0.4]

- Bug Fixes
  - Fixed MISRA 8.4 issues.

[2.0.3]

- Bug Fixes
  - Fixed MISRA 10.4 issues.

[2.0.2]

- Improvements
  - Split FLEXIO component which combines all flexio/flexio_uart/flexio_i2c/flexio_i2s drivers into several components: FlexIO component, flexio_uart component, flexio_i2c_master component, and flexio_i2s component.
- Bug Fixes
  - Fixed MISRA issues
    - Fixed rules 10.1, 10.3, 10.4, 10.7, 11.6, 11.9, 14.4, 17.7.

[2.0.1]

- Bug Fixes
  - Fixed the dozen mode configuration error in FLEXIO_Init API. For enableInDoze = true, the configuration should be 0; for enableInDoze = false, the configuration should be 1.

FLEXIO_I2C

[2.6.3]

- Bug Fixes
  - Fixed MISRA 10.4, 10.7 and 14.4 issues.

[2.6.2]

- Improvements
  - Added timeout for while loop in FLEXIO_I2C_MasterTransferBlocking().
- Bug Fixes
  - Fixed build issues related to I2C_RETRY_TIMES.

[2.6.1]

- Bug Fixes
  - Fixed coverity issues

[2.6.0]

- Improvements
  - Supported platforms which don't have DOZE mode control.

[2.5.1]

- Improvements
  - Conditionally compile interrupt handling code to solve the problem of using this driver on CPU cores that do not support interrupts.

[2.5.0]

- Improvements
  - Split some functions, fixed CCM problem in file fsl_flexio_i2c_master.c.

[2.4.0]

- Improvements
  - Added delay of 1 clock cycle in FLEXIO_I2C_MasterTransferRunStateMachine to ensure that bus would be idle before next transfer if master is nacked.
  - Fixed issue that the restart setup time is less than the time in I2C spec by adding delay of 1 clock cycle before restart signal.

[2.3.0]

- Improvements
  - Used 3 timers instead of 2 to support transfer which is more than 14 bytes in single transfer.
  - Improved FLEXIO_I2C_MasterTransferGetCount so that the API can check whether the transfer is still in progress.
- Bug Fixes
  - Fixed MISRA 10.4 issues.

[2.2.0]

- New Features
  - Added timeout mechanism when waiting certain state in transfer API.
  - Added an API for checking bus pin status.
- Bug Fixes
  - Fixed COVERITY issue of useless call in FLEXIO_I2C_MasterTransferRunStateMachine.
  - Fixed MISRA issues
    - Fixed rules 10.1, 10.3, 10.4, 10.7, 11.6, 11.9, 14.4, 17.7.
  - Added codes in FLEXIO_I2C_MasterTransferCreateHandle to clear pending NVIC IRQ, disable internal IRQs before enabling NVIC IRQ.
  - Modified code so that during master's nonblocking transfer the start and slave address are sent after interrupts being enabled, in order to avoid potential issue of sending the start and slave address twice.

[2.1.7]

- Bug Fixes
  - Fixed the issue that FLEXIO_I2C_MasterTransferBlocking did not wait for STOP bit sent.
  - Fixed COVERITY issue of useless call in FLEXIO_I2C_MasterTransferRunStateMachine.
  - Fixed the issue that I2C master did not check whether bus was busy before transfer.

[2.1.6]

- Bug Fixes
  - Fixed the issue that I2C Master transfer APIs(blocking/non-blocking) did not support the situation of master transfer with subaddress and transfer data size being zero, which means no data followed the subaddress.

[2.1.5]

- Improvements
  - Unified component full name to FLEXIO I2C Driver.

[2.1.4]

- Bug Fixes
  - The following modifications support FlexIO using multiple instances:
    - Removed FLEXIO_Reset API in module Init APIs.
    - Updated module Deinit APIs to reset the shifter/timer config instead of disabling module/clock.
    - Updated module Enable APIs to only support enable operation.

[2.1.3]

- Improvements
  - Changed the prototype of FLEXIO_I2C_MasterInit to return kStatus_Success if initialized successfully or to return kStatus_InvalidArgument if "(srcClock_Hz / masterConfig->baudRate_Bps) / 2 - 1" exceeds 0xFFU.

[2.1.2]

- Bug Fixes
  - Fixed the FLEXIO I2C issue where the master could not receive data from I2C slave in high baudrate.
  - Fixed the FLEXIO I2C issue where the master could not receive NAK when master sent non-existent addr.
  - Fixed the FLEXIO I2C issue where the master could not get transfer count successfully.
  - Fixed the FLEXIO I2C issue where the master could not receive data successfully when sending data first.
  - Fixed the Dozen mode configuration error in FLEXIO_I2C_MasterInit API. For enableInDoze = true, the configuration should be 0; for enableInDoze = false, the configuration should be 1.
  - Fixed the issue that FLEXIO_I2C_MasterTransferBlocking API called FLEXIO_I2C_MasterTransferCreateHandle, which lead to the s_flexioHandle/s_flexioIsr/s_flexioType variable being written. Then, if calling FLEXIO_I2C_MasterTransferBlocking API multiple times, the s_flexioHandle/s_flexioIsr/s_flexioType variable would not be written any more due to it being out of range. This lead to the following situation: NonBlocking transfer APIs could not work due to the fail of register IRQ.

[2.1.1]

- Bug Fixes
  - Implemented the FLEXIO_I2C_MasterTransferBlocking API which is defined in header file but has no implementation in the C file.

[2.1.0]

- New Features
  - Added Transfer prefix in transactional APIs.
  - Added transferSize in handle structure to record the transfer size.

FLEXIO_I2S

[2.2.2]

- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 12.4.

[2.2.1]

- Improvements
  - Conditionally compile interrupt handling code to solve the problem of using this driver on CPU cores that do not support interrupts.

[2.2.0]

- New Features
  - Added timeout mechanism when waiting certain state in transfer API.
- Bug Fixes
  - Fixed IAR Pa082 warnings.
  - Fixed violations of the MISRA C-2012 rules 10.4, 14.4, 11.8, 11.9, 10.1, 17.7, 11.6, 10.3, 10.7.

[2.1.6]

- Bug Fixes
  - Added reset flexio before flexio i2s init to make sure flexio status is normal.

[2.1.5]

- Bug Fixes
  - Fixed the issue that I2S driver used hard code for bitwidth setting.

[2.1.4]

- Improvements
  - Unified component's full name to FLEXIO I2S (DMA/EDMA) driver.

[2.1.3]

- Bug Fixes
  - The following modifications support FLEXIO using multiple instances:
    * Removed FLEXIO_Reset API in module Init APIs.
    * Updated module Deinit APIs to reset the shifter/timer config instead of disabling module/clock.
    * Updated module Enable APIs to only support enable operation.

[2.1.2]

- New Features
  - Added configure items for all pin polarity and data valid polarity.
  - Added default configure for pin polarity and data valid polarity.

[2.1.1]

- Bug Fixes
  - Fixed FlexIO I2S RX data read error and eDMA address error.
  - Fixed FlexIO I2S slave timer compare setting error.

[2.1.0]

- New Features
  - Added Transfer prefix in transactional APIs.
  - Added transferSize in handle structure to record the transfer size.

FLEXIO_SPI

[2.4.4]

- Bug Fixes
  - Fixed MISRA 10.4 issues.

[2.4.3]

- Improvements
  - Make SPI_RETRY_TIMES configurable by CONFIG_SPI_RETRY_TIMES.

[2.4.2]

- Bug Fixes
  - Fixed FLEXIO_SPI_MasterTransferBlocking and FLEXIO_SPI_MasterTransferNonBlocking issue in CS continuous mode, the CS might not be continuous.

[2.4.1]

- Bug Fixes
  - Fixed coverity issues

[2.4.0]

- Improvements
  - Supported platforms which don't have DOZE mode control.

[2.3.5]

- Improvements
  - Conditionally compile interrupt handling code to solve the problem of using this driver on CPU cores that do not support interrupts.

[2.3.4]

- Bug Fixes
  - Fixed the txData from void * to const void * in transmit API

[2.3.3]

- Bugfixes
  - Fixed cs-continuous mode.

[2.3.2]

- Improvements
  - Changed FLEXIO_SPI_DUMMYDATA to 0x00.

[2.3.1]

- Bugfixes
  - Fixed IRQ SHIFTBUF overrun issue when one FLEXIO instance used as multiple SPIs.

[2.3.0]

- New Features
  - Supported FLEXIO_SPI slave transfer with continuous master CS signal and CPHA=0.
  - Supported FLEXIO_SPI master transfer with continuous CS signal.
  - Support 32 bit transfer width.
- Bug Fixes
  - Fixed wrong timer compare configuration for dma/edma transfer.
  - Fixed wrong byte order of rx data if transfer width is 16 bit, since the we use shifter buffer bit swapped/byte swapped register to read in received data, so the high byte should be read from the high bits of the register when MSB.

[2.2.1]

- Bug Fixes
  - Fixed bug in FLEXIO_SPI_MasterTransferAbortEDMA that when aborting EDMA transfer EDMA_AbortTransfer should be used rather than EDMA_StopTransfer.

[2.2.0]

- Improvements
  - Added timeout mechanism when waiting certain states in transfer driver.
- Bug Fixes
  - Fixed MISRA 10.4 issues.
  - Added codes in FLEXIO_SPI_MasterTransferCreateHandle and FLEXIO_SPI_SlaveTransferCreateHandle to clear pending NVIC IRQ before enabling NVIC IRQ, to fix issue of pending IRQ interfering the on-going process.

[2.1.3]

- Improvements
  - Unified component full name to FLEXIO SPI(DMA/EDMA) Driver.
- Bug Fixes
  - Fixed MISRA issues
    - Fixed rules 10.1, 10.3, 10.4, 10.7, 11.6, 11.9, 14.4, 17.7.

[2.1.2]

- Bug Fixes
  - The following modification support FlexIO using multiple instances:
    - Removed FLEXIO_Reset API in module Init APIs.
    - Updated module Deinit APIs to reset the shifter/timer config instead of disabling module/clock.
    - Updated module Enable APIs to only support enable operation.

[2.1.1]

- Bug Fixes
  - Fixed bug where FLEXIO SPI transfer data is in 16 bit per frame mode with eDMA.
  - Fixed bug when FLEXIO SPI works in eDMA and interrupt mode with 16-bit per frame and Lsbfirst.
  - Fixed the Dozen mode configuration error in FLEXIO_SPI_MasterInit/FLEXIO_SPI_SlaveInit API. For enableInDoze = true, the configuration should be 0; for enableInDoze = false, the configuration should be 1.
- Improvements
  - Added #ifndef/#endif to allow users to change the default TX value at compile time.

[2.1.0]

- New Features
  - Added Transfer prefix in transactional APIs.
  - Added transferSize in handle structure to record the transfer size.
- Bug Fixes

  - Fixed the error register address return for 16-bit data write in FLEXIO_SPI_GetTxDataRegisterAddress.
  - Provided independent IRQHandler/transfer APIs for Master and slave to fix the baudrate limit issue.

FLEXIO_UART

[2.6.5]

- Bug Fixes
  - Fixed MISRA 10.4 issues.

[2.6.4]

- Improvements
  - Make UART_RETRY_TIMES configurable by CONFIG_UART_RETRY_TIMES.

[2.6.3]

- Bug Fixes
  - Fixed coverity issues

[2.6.2]

- Bug Fixes
  - Fixed coverity issues

[2.6.1]

- Improvements
  - Improve baudrate calculation method, to support higher frequency FlexIO clock source.

[2.6.0]

- Improvements
  - Supported platforms which don't have DOZE mode control.

[2.5.1]

- Improvements
  - Conditionally compile interrupt handling code to solve the problem of using this driver on CPU cores that do not support interrupts.

[2.5.0]

- Improvements
  - Added API FLEXIO_UART_FlushShifters to flush UART fifo.

[2.4.0]

- Improvements
  - Use separate data for TX and RX in flexio_uart_transfer_t.
- Bug Fixes
  - Fixed bug that when ring buffer is used, if some data is received in ring buffer first before calling FLEXIO_UART_TransferReceiveNonBlocking, the received data count returned by FLEXIO_UART_TransferGetReceiveCount is wrong.

[2.3.0]

- Improvements
  - Added check for baud rate's accuracy that returns kStatus_FLEXIO_UART_BaudrateNotSupport when the best achieved baud rate is not within 3% error of configured baud rate.
- Bug Fixes
  - Added codes in FLEXIO_UART_TransferCreateHandle to clear pending NVIC IRQ before enabling NVIC IRQ, to fix issue of pending IRQ interfering the on-going process.

[2.2.0]

- Improvements
  - Added timeout mechanism when waiting for certain states in transfer driver.
- Bug Fixes
  - Fixed MISRA 10.4 issues.

[2.1.6]

- Bug Fixes
  - Fixed IAR Pa082 warnings.
  - Fixed MISRA issues
    - Fixed rules 10.1, 10.3, 10.4, 10.7, 11.6, 11.9, 14.4, 17.7.

[2.1.5]

- Improvements
  - Triggered user callback after all the data in ringbuffer were received in FLEXIO_UART_TransferReceiveNonBlocking.

[2.1.4]

- Improvements
  - Unified component full name to FLEXIO UART(DMA/EDMA) Driver.

[2.1.3]

- Bug Fixes
  - The following modifications support FLEXIO using multiple instances:
    - Removed FLEXIO_Reset API in module Init APIs.
    - Updated module Deinit APIs to reset the shifter/timer configuration instead of disabling module and clock.
    - Updated module Enable APIs to only support enable operation.

[2.1.2]

- Bug Fixes
  - Fixed the transfer count calculation issue in FLEXIO_UART_TransferGetReceiveCount, FLEXIO_UART_TransferGetSendCount, FLEXIO_UART_TransferGetReceiveCountDMA, FLEXIO_UART_TransferGetSendCountDMA, FLEXIO_UART_TransferGetReceiveCountEDMA and FLEXIO_UART_TransferGetSendCountEDMA.
  - Fixed the Dozen mode configuration error in FLEXIO_UART_Init API. For enableInDoze = true, the configuration should be 0; for enableInDoze = false, the configuration should be 1.
  - Added code to report errors if the user sets a too-low-baudrate which FLEXIO cannot reach.
  - Disabled FLEXIO_UART receive interrupt instead of all NVICs when reading data from ring buffer. If ring buffer is used, receive nonblocking will disable all NVIC interrupts to protect the ring buffer. This had negative effects on other IPs using interrupt.

[2.1.1]

- Bug Fixes
  - Changed the API name FLEXIO_UART_StopRingBuffer to FLEXIO_UART_TransferStopRingBuffer to align with the definition in C file.

[2.1.0]

- New Features
  - Added Transfer prefix in transactional APIs.
  - Added txSize/rxSize in handle structure to record the transfer size.
- Bug Fixes
  - Added an error handle to handle the situation that data count is zero or data buffer is NULL.

FLEXIO_UART_EDMA

[2.3.1]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules.

[2.3.0]

- Refer FLEXIO_UART driver change log to 2.3.0

GPIO

[2.8.6]

- Bug Fixes
  - Fixed CERT INT31-C violations in GPIO_FIT_REG macro for platforms with 8-bit register width.

[2.8.5]

- Improvements
  - Add AND logic in GPIO_GpioClearInterruptFlags to support clearing multiple interrupt flags simultaneously.

[2.8.4]

- Improvements
  - Make function GPIO_PortGetInterruptFlags and GPIO_PortClearInterruptFlags available for all variants, they are used for all pins flags in one GPIO port.

[2.8.3]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 Rule 10.1, 5.7.

[2.8.2]

- Bug Fixes
  - Fixed COVERITY issue that GPIO_GetInstance could return clock array overflow values due to GPIO base and clock being out of sync.

[2.8.1]

- Bug Fixes
  - Fixed CERT INT31-C issues.

[2.8.0]

- Improvements
  - Add API GPIO_PortInit/GPIO_PortDeinit to set GPIO clock enable and releasing GPIO reset.

[2.8.0]

- Improvements
  - Add API GPIO_PortInit/GPIO_PortDeinit to set GPIO clock enable and releasing GPIO reset.
  - Remove support for API GPIO_GetPinsDMARequestFlags with GPIO_ISFR_COUNT <= 1.

[2.7.3]

- Improvements
  - Release peripheral from reset if necessary in init function.

[2.7.2]

- New Features
  - Support devices without PORT module.

[2.7.1]

- Bug Fixes
  - Fixed MISRA C-2012 rule 10.4 issues in GPIO_GpioGetInterruptChannelFlags() function and GPIO_GpioClearInterruptChannelFlags() function.

[2.7.0]

- New Features
  - Added API to support Interrupt select (IRQS) bitfield.

[2.6.0]

- New Features
  - Added API to get GPIO version information.
  - Added API to control a pin for general purpose input.
  - Added some APIs to control pin in secure and previliege status.

[2.5.3]

- Bug Fixes
  - Correct the feature macro typo: FSL_FEATURE_GPIO_HAS_NO_INDEP_OUTPUT_CONTORL.

[2.5.2]

- Improvements
  - Improved GPIO_PortSet/GPIO_PortClear/GPIO_PortToggle functions to support devices without Set/Clear/Toggle registers.

[2.5.1]

- Bug Fixes
  - Fixed wrong macro definition.
  - Fixed MISRA C-2012 rule issues in the FGPIO_CheckAttributeBytes() function.
  - Defined the new macro to separate the scene when the width of registers is different.
  - Removed some redundant macros.
- New Features
  - Added some APIs to get/clear the interrupt status flag when the port doesn't control pins' interrupt.

[2.4.1]

- Improvements
  - Improved GPIO_CheckAttributeBytes() function to support 8 bits width GACR register.

[2.4.0]

- Improvements
  - API interface added:
    * New APIs were added to configure the GPIO interrupt clear settings.

[2.3.2]

- Bug Fixes
  - Fixed the issue for MISRA-2012 check.
    * Fixed rule 3.1, 10.1, 8.6, 10.6, and 10.3.

[2.3.1]

- Improvements
  - Removed deprecated APIs.

[2.3.0]

- New Features
  - Updated the driver code to adapt the case of interrupt configurations in GPIO module. New APIs were added to configure the GPIO interrupt settings if the module has this feature on it.

[2.2.1]

- Improvements
  - API interface changes:
    * Refined naming of APIs while keeping all original APIs by marking them as deprecated. The original APIs will be removed in next release. The main change is updating APIs with prefix of _PinXXX() and _PortXXX.

[2.1.1]

- Improvements
  - API interface changes:
    - Added an API for the check attribute bytes.

[2.1.0]

- Improvements
  - API interface changes:
    - Added "pins" or "pin" to some APIs' names.
    - Renamed "_PinConfigure" to "GPIO_PinInit".

I3C

[2.15.0]

- Improvements
  - Added new timing APIs (I3C_MasterCalcTiming, I3C_MasterSetTiming) and supporting types to provide bus-topology-aware baud rate calculation with ODHPP control and pre-computed timing switching.
- New Features
  - Added I3C_MasterGetI2CPresetTiming to get baud rate settings which meet I2C timing.
- Bug Fixes
  - Fixed Coverity issues.

[2.14.7]

- Bug Fixes
  - Fixed I3C master interrupt transfer not stopping the bus on errors other than Nak/IBIWon.

[2.14.6]

- New Features
  - Added common IRQ handler entry I3C_DriverIRQHandler.

[2.14.5]

- Bug Fixes
  - Fixed CERT INT31-C violation: cast bool field enableSlave to uint via ternary in I3C_Init.

[2.14.4]

- Improvements
  - Fixed MSG violations.

[2.14.3]

- Improvements
  - Fixed Coverity CERT-C violations.
  - Used I3C_RSTS instead of I3C special feature macro.
  - Adapted the driver to support new platform.

[2.14.2]

- Improvements
  - Added timeout for ENTDAA process API.
  - Added build system macro to control the timeout setting.

[2.14.1]

- Improvements
  - Split the function I3C_MasterTransferBlocking to meet the HIS-CCM requirement.

[2.14.0]

- Improvements
  - Added the choice to set fast start header with push-pull speed when all targets addresses have MSB 0 instead of forcing to set it.
  - Deleted duplicated busy check in I3C_MasterStart function.

[2.13.1]

- Bug Fixes
  - Disabled Rx auto-termination in repeated start interrupt event while transfer API doesn't enable it.
  - Waited the completion event after loading all Tx data in Tx FIFO.
- Improvements
  - Conditionally compile interrupt handling code to solve the problem of using this driver on CPU cores that do not support interrupts.

[2.13.0]

- New features
  - Added the hot-join support for I3C bus initialization API.
- Bug Fixes
  - Set read termination with START at the same time in case unknown issue.
  - Set MCTRL[TYPE] as 0 for DDR force exit.
- Improvements
  - Added the API to reset device count assigned by ENTDAA.
  - Provided the method to set global macro I3C_MAX_DEVCNT to determine how many device addresses ENTDAA can allocate at one time.

  - Initialized target management static array based on instance number for the case that multiple instances are used at the same time.

[2.12.0]

- Improvements
  - Added the slow clock parameter for Controller initialization function to calculate accurate timeout.
- Bug Fixes
  - Fixed the issue that BAMATCH field can't be 0. BAMATCH should be 1 for 1MHz slow clock.

[2.11.1]

- Bug Fixes
  - Fixed the issue that interrupt API transmits extra byte when subaddress and data size are null.
  - Fixed the slow clock calculation issue.

[2.11.0]

- New features
  - Added the START/ReSTART SCL delay setting for the Soc which supports this feature.
- Bug Fixes
  - Fixed the issue that ENTDAA process waits Rx pending flag which causes problem when Rx watermark isn't 0. Just check the Rx FIFO count.

[2.10.8]

- Improvements
  - Support more instances.

[2.10.7]

- Improvements
  - Fixed the potential compile warning.

[2.10.6]

- New features
  - Added the I3C private read/write with 0x7E address as start.

[2.10.5]

- New features
  - Added I3C HDR-DDR transfer support.

[2.10.4]

- Improvements
  - Added one more option for master to not set RDTERM when doing I3C Common Command Code transfer.

[2.10.3]

- Improvements
  - Masked the slave IBI/MR/HJ request functions with feature macro.

[2.10.2]

- Bug Fixes
  - Added workaround for errata ERR051617: I3C working with I2C mode creates the unintended Repeated START before actual STOP on some platforms.

[2.10.1]

- Bug Fixes
  - Fixed the issue that DAA function doesn't wait until all Rx data is read out from FIFO after master control done flag is set.
  - Fixed the issue that DAA function could return directly although the disabled interrupts are not enabled back.

[2.10.0]

- New features
  - Added I3C extended IBI data support.

[2.9.0]

- Improvements
  - Added adaptive termination for master blocking transfer. Set termination with start signal when receiving bytes less than 256.

[2.8.2]

- Improvements
  - Fixed the build warning due to armgcc strict check.

[2.8.1]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 17.7.

[2.8.0]

- Improvements
  - Added API I3C_MasterProcessDAASpecifiedBaudrate for temporary baud rate adjustment when I3C master assigns dynamic address.

[2.7.1]

- Bug Fixes
  - Fixed the issue that I3C slave handle STOP event before finishing data transmission.

[2.7.0]

- Fixed the CCM problem in file fsl_i3c.c.
- Fixed the FSL_FEATURE_I3C_HAS_NO_SCONFIG_IDRAND usage issue in I3C_GetDefaultConfig and I3C_Init.

[2.6.0]

- Fixed the FSL_FEATURE_I3C_HAS_NO_SCONFIG_IDRAND usage issue in fsl_i3c.h.
- Changed some static functions in fsl_i3c.c as non-static and define the functions in fsl_i3c.h to make I3C DMA driver reuse:
  - I3C_GetIBIType
  - I3C_GetIBIAddress
  - I3C_SlaveCheckAndClearError
- Changed the handle pointer parameter in IRQ related funtions to void * type to make it reuse in I3C DMA driver.
- Added new API I3C_SlaveRequestIBIWithSingleData for slave to request single data byte, this API could be used regardless slave is working in non-blocking interrupt or non-blocking dma.
- Added new API I3C_MasterGetDeviceListAfterDAA for master application to get the device information list built up in DAA process.

[2.5.4]

- Improved I3C driver to avoid setting state twice in the SendCommandState of I3C_RunTransferStateMachine.
- Fixed MISRA violation of rule 20.9.
- Fixed the issue that I3C_MasterEmitRequest did not use Type I3C SDR.

[2.5.3]

- Updated driver for new feature FSL_FEATURE_I3C_HAS_NO_SCONFIG_BAMATCH and FSL_FEATURE_I3C_HAS_NO_SCONFIG_IDRAND.

[2.5.2]

- Updated driver for new feature FSL_FEATURE_I3C_HAS_NO_MERRWARN_TERM.
- Fixed the issue that call to I3C_MasterTransferBlocking API did not generate STOP signal when NAK status was returned.

[2.5.1]

- Improved the receive terminate size setting for interrupt transfer read, now it's set at beginning of transfer if the receive size is less than 256 bytes.

[2.5.0]

- Added new API I3C_MasterRepeatedStartWithRxSize to send repeated start signal with receive terminate size specified.
- Fixed the status used in I3C_RunTransferStateMachine, changed to use pending interrupts as status to be handled in the state machine.
- Fixed MISRA 2012 violation of rule 10.3, 10.7.

[2.4.0]

- Bug Fixes
  - Fixed kI3C_SlaveMatchedFlag interrupt is not properly handled in I3C_SlaveTransferHandleIRQ when it comes together with interrupt kI3C_SlaveBusStartFlag.
  - Fixed the inaccurate I2C baudrate calculation in I3C_MasterSetBaudRate.
  - Added new API I3C_MasterGetIBIRules to get registered IBI rules.
  - Added new variable isReadTerm in struct _i3c_master_handle for transfer state routine to check if MCTRL.RDTERM is configured for read transfer.
  - Changed to emit Auto IBI in transfer state routine for slave start flag assertion.
  - Fixed the slave maxWriteLength and maxReadLength does not be configured into SMAXLIMITS register issue.
  - Fixed incorrect state for IBI in I3C master interrupt transfer IRQ handle routine.
  - Added isHotJoin in i3c_slave_config_t to request hot-join event during slave init.

[2.3.2]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 8.4, 17.7.
  - Fixed incorrect HotJoin event index in I3C_GetIBIType.

[2.3.1]

- Bug Fixes
  - Fixed the issue that call of I3C_MasterTransferBlocking/I3C_MasterTransferNonBlocking fails for the case which receive length 1 byte of data.
  - Fixed the issue that STOP signal is not sent when NAK status is detected during execution of I3C_MasterTransferBlocking function.

[2.3.0]

- Improvements
  - Added I3C common driver APIs to initialize I3C with both master and slave configuration.

  - Updated I3C master transfer callback to function set structure to include callback invoke for IBI event and slave2master event.
  - Updated I3C master non-blocking transfer model and always enable the interrupts to be able to re-act to the slave start event and handle slave IBI.

[2.2.0]

- Bug Fixes
  - Fixed the issue that I3C transfer size limit to 255 bytes.

[2.1.2]

- Bug Fixes
  - Reset default hkeep value to kI3C_MasterHighKeeperNone in I3C_MasterGetDefaultConfig

[2.1.1]

- Bug Fixes
  - Fixed incorrect FIFO reset operation in I3C Master Transfer APIs.
  - Fixed i3c slave IRQ handler issue, slave transmit could be underrun because tx FIFO is not filled in time right after start flag detected.

[2.1.0]

- Added definitions and APIs for I3C slave functionality, updated previous I3C APIs to support I3C functionality.

[2.0.0]

- Initial version.

IMU

[2.3.0]

- Bug Fixes
  - Fix MISRA C-2012 Rule 20.8 and 15.5 violations.

[2.2.1]

- Bug Fixes
  - Fix MISRA C-2012 Rule 14.3 violations.

[2.2.0]

- New Features
  - Added IMU_BUSY_POLL_COUNT parameter to prevent infinite polling loops in IMU operations.
  - Added timeout mechanism to all polling loops in IMU driver code.
- Improvements
  - Enhanced error handling in blocking functions to return timeout status.
  - Updated documentation to clarify timeout behavior and return values.
  - Added IMU_ERR_TIMEOUT error code for timeout conditions.

[2.1.1]

- Bug Fixes
  - Fix MISRA C-2012 violations.
  - Fixed IMU_GetStatusFlags bug that returns wrong RX FIFO status.

[2.1.0]

- Improvements:
  - Updated API prototype, remove CIU2_Type from parameters.

[2.0.0]

- Initial version.

LPADC

[2.11.0]

- New Features
  - Added the LPADC_USE_FIXED_POINT_GAIN_CALCULATION configuration macro. When set to 1, LPADC_FinishAutoCalibration() calculates the gain conversion result with integer-only arithmetic instead of the float/double software library, saving several kB of flash on code size sensitive devices. The fixed point result keeps the gain fraction in Q17.14, so the two least significant bits of the gain conversion result are always zero.

[2.10.3]

- Bug Fixes
  - Fixed MISRA C-2012 essential-type-model violations (Rules 10.1, 10.3, 10.4 and 10.8) in LPADC_FinishAutoCalibration and LPADC_GetOffsetValue.

[2.10.2]

- Improvements
  - Gate the trigger (TCTRL) enums and the trigger-id assert on FSL_FEATURE_ADC_TCTRL_COUNT for every trigger index, so devices with fewer triggers no longer expose unavailable trigger registers.

[2.10.1]

- Improvements
  - Reworked the gain-calibration ratio in LPADC_FinishAutoCalibration to use single-precision (float) arithmetic instead of double-precision, removing the dependency on the libgcc double soft-float routines (__floatsidf, __aeabi_dsub, __aeabi_ddiv, __aeabi_d2f) that were linked into every application using the LPADC driver, reducing ROM footprint.

[2.10.0]

- Improvements
  - Use #if !defined(ADC_TCTRL_COUNT) || (ADC_TCTRL_COUNT > nU) to wrap some configurations because these configurations are invalid/illegal on some SoCs.

[2.9.5]

- Improvements
  - Fix doxygen issue, grouping command should be balanced.

[2.9.4]

- Improvements
  - Update LPADC_GetDefaultConfig, change default conversionAverageMode value to: kLPADC_ConversionAverage128 for 3 bit width. kLPADC_ConversionAverage1024 for 4 bit width.

[2.9.3]

- Improvements
  - Add timeout for while loop code.

[2.9.2]

- Improvements
  - Fixed CERT-C issues.

[2.9.1]

- Bug Fixes
  - Fixed incorrect channel B FIFO selection logic.

[2.9.0]

- Bug Fixes
  - Add code to handle the case where GCC[GAIN_CAL] is a signed number.
  - Split LPADC_FinishAutoCalibration function into two functions.
  - Improved LPADC driver.

[2.8.4]

- Bug Fixes
  - Remove function 'LPADC_SetOffsetValue' assert statement, this statement may cause runtime errors in existing code.

[2.8.3]

- Bug Fixes
  - Fixed SDK lpadc driver examples compile issue, move condition 'commandId < ADC_CV_COUNT' to a more appropriate location.

[2.8.2]

- Bug Fixes
  - Fixed the violations of MISRA C-2012 rule 18.1, 10.3, 10.1 and 10.4.

[2.8.1]

- Bug Fixes
  - Fixed LPADC sample mode enum name mistake.

[2.8.0]

- Improvements
  - Release peripheral from reset if necessary in init function.
- Bug Fixes
  - Fixed function LPADC_GetConvResult() issue.
  - Fixed function LPADC_SetConvCommandConfig() bugs.

[2.7.2]

- Improvements
  - Use feature macros instead of header file macros.
- Bug Fixes
  - Fixed the violations of MISRA C-2012 rule 10.1, 10.3, 10.4 and 14.3.

[2.7.1]

- Improvements
  - Corrected descriptions of several functions.
  - Improved function LPADC_GetOffsetValue and LPADC_SetOffsetValue.
  - Revert changes of feature macros for lpadc.
  - Use feature macros instead of header file macros.
- Bug Fixes
  - Fixed the violations of MISRA C-2012 rule 10.8.
  - Fixed the violations of MISRA C-2012 rule 10.1, 10.3, 10.4 and 14.3.

[2.7.0]

- Improvements
  - Added supports of CFG2 register.
  - Removed some useless macros.

[2.6.2]

- Bug Fixes
  - Fixed the violations of MISRA C-2012 rules.
  - Fixed LPADC driver code compile error issue.

[2.6.1]

- Improvements
  - Updated the use of macros in the driver code.

[2.6.0]

- Improvements
  - Added the API LPADC_SetOffset12BitValue() to configure 12bit ADC conversion offset trim value manually.
  - Added the API LPADC_SetOffset16BitValue() to configure 16bit ADC conversion offset trim value manually.
  - Added API to set offset calibration mode.
  - Added configuration of alternate channel.
  - Updated auto calibration API and added calibration value conversion API.
- New feature
  - Added API LPADC_EnableHardwareTriggerCommandSelection() to enable trigger commands controlled by ADC_ETC.
  - Updated LPADC_DoAutoCalibration() to allow doing something else before the ADC initialization to be totally complete. Enhance initialization duration time of the ADC.
  - Added two new APIs to get/set calibration value.

[2.5.2]

- Improvements
  - Added while loop, LPADC_GetConvResult() will return only when the FIFO will not be empty.

[2.5.1]

- Bug Fixes
  - Fixed some typos in Lpadc driver comments.

[2.5.0]

- Improvements
  - Added missing items to enable trigger interrupts.

[2.4.0]

- New features
  - Added APIs to get/clear trigger status flags.

[2.3.0]

- Improvements
  - Removed LPADC_MeasureTemperature() function for the LPADC supports different temperature sensor calculation equations.

[2.2.1]

- Improvements
  - Optimized LPADC_MeasureTemperature() function to support the specific series with flash solidified calibration value.
  - Clean doxygen warnings.
- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 10.3, rule 10.8 and rule 17.7.

[2.2.0]

- New Feature
  - Added API LPADC_MeasureTemperature() to get correct temperature from the internal sensor.
- Improvements
  - Separated lpadc_conversion_resolution_mode_t with related feature macro.
- Bug Fixes
  - Fixed the violations of MISRA C-2012 rules:
    - Rule 10.3, 10.4, 10.6, 10.7 and 17.7.

[2.1.1]

- Improvements
  - Updated the gain calibration formula.
  - Used feature to segregate the new item kLPADC_TriggerPriorityPreemptSubsequently.

[2.1.0]

- New Features
  - Added the API LPADC_SetOffsetValue() to support configure offset trim value manually.
  - Added the API LPADC_DoOffsetCalibration() to do offset calibration independently.
- Improvements
  - Improved the usage of macros and removed invalid macros.

[2.0.2]

- Improvements
  - Added support for platforms with 2 FIFOs and different calibration measures.

[2.0.1]

- Bug Fixes
  - Ensured the API LPADC_SetConvCommandConfig configure related registers correctly.

[2.0.0]

- Initial version.

LPCMP

[2.3.2]

- Improvements
  - Fixed LPCMP CERT-C issues.

[2.3.1]

- Improvements
  - Update LPCMP driver to be compatible with platforms that do not support LPCMP nano power mode selection.

[2.3.0]

- New Feature
  - Added some new features for platforms which support
    - Plus input source selection.
    - Minus input source selection.
    - CMP to DAC link.
- Improvements
  - Removed some new features for platforms which doesn't support
    - Functional clock source selection.
    - DAC high power mode selection.
    - Round Robin clock source selection.

    * Round Robin trigger source selection.
    * Round Robin channel sample numbers setting.
    * Round Robin channel sample time threshold setting.
    * Round Robin internal trigger configuration.

[2.2.0]

- Improvements
  - Change FSL_FEATURE_LPCMP_HAS_NO_CCR0_CMP_STOP_EN to FSL_FEATURE_LPCMP_HAS_CCR0_CMP_STOP_EN.

[2.1.3]

- New Feature
  - Added new macro to handle the case where some instances do not have the CCR0 CMP_STOP_EN bit field.

[2.1.2]

- New Feature
  - Add macros to be compatible with some platforms that do not have the CCR0 CMP_STOP_EN bitfield.

[2.1.1]

- Improvements
  - Release peripheral from reset if necessary in init function.

[2.1.0]

- New Features:
  - Supported round robin mode and window mode feature.

[2.0.3]

- Bug Fixes:
  - Fixed the violation of MISRA-2012 rule 17.7.

[2.0.2]

- Bug Fixes:
  - The current API LPCMP_ClearStatusFlags has to check w1c bits.

[2.0.1]

- Added control macro to enable/disable the CLOCK code in current driver.

[2.0.0]

- Initial version.

LPI2C

[2.6.8]

- Bug Fixes
  - LPI2C_MasterTransferBlocking - added waiting while TX FIFO is not empty.
  - Added a check for an error from the previous transfer in the transfer API (when kLPI2C_TransferNoStopFlag was used).
  - Updated LPI2C_MasterReceive() to return kStatus_InvalidArgument when rxSize is 0.
  - Updated transfer API functions to return kStatus_InvalidArgument when transfer->dataSize is 0 and transfer->direction is kLPI2C_Read.
  - Fixed assert issue in LPI2C_MasterSetBaudRate().
  - Fixed CERT-C issues.
  - Fixed MISRA issues.
  - Improved HIS CCM for LPI2C_MasterTransferBlocking() and LPI2C_SlaveTransferHandleIRQ().

[2.6.7]

- Bug Fixes
  - Fixed MISRA 10.3 and 10.8 issues.
  - Fixed LPI2C_MasterStop() to send I2C STOP in case of error.

[2.6.6]

- Bug Fixes
  - Set handle->chunkSize to 0 in LPI2C_MasterTransferNonBlocking().

[2.6.5]

- Bug Fixes
  - Removed write to STAR register in LPI2C_SlaveTransferHandleIRQ handler in case of kLPI2C_SlaveRepeatedStartDetectFlag or kLPI2C_SlaveStopDetectFlag.
  - Removed configuration of FILTSDA from LPI2C_MasterSetBaudRate().
  - Set minimal possible value for FILTSDA and FILTSCL to 0.
  - Updated configuration of glitch filters in LPI2C_SlaveInit().
  - Set minimal possible value for SCFGR2[CLKHOLD] to 1.
  - Added error check to while loop in LPI2C_TransferStateMachineSendCommand().
- Improvements
  - Removed receive data size limitation in LPI2C_MasterTransferBlocking() and LPI2C_MasterTransferNonBlocking().

[2.6.4]

- Bug Fixes
  - Limited value of filtSda in LPI2C_MasterSetBaudRate().
  - Updated LPI2C_MasterStop() and LPI2C_MasterTransferBlocking() to send I2C STOP also in case of error.
  - Removed unused flag kLPI2C_TransferRepeatedStartFlag.

[2.6.3]

- Bug Fixes
  - Fixed static analysis identified issues.

[2.6.2]

- Improvements
  - Added timeout for while loop in LPI2C_TransferStateMachineSendCommand().

[2.6.1]

- Bug Fixes
  - Fixed coverity issues.

[2.6.0]

- New Feature
  - Added common IRQ handler entry LPI2C_DriverIRQHandler.

[2.5.7]

- Improvements
  - Added support for separated IRQ handlers.

[2.5.6]

- Improvements
  - Conditionally compile interrupt handling code to solve the problem of using this driver on CPU cores that do not support interrupts.

[2.5.5]

- Bug Fixes
  - Fixed LPI2C_SlaveInit() - allow to disable SDA/SCL glitch filter.

[2.5.4]

- Bug Fixes
  - Fixed LPI2C_MasterTransferBlocking() - the return value was sometime affected by call of LPI2C_MasterStop().

[2.5.3]

- Improvements
  - Added handler for LPI2C7 and LPI2C8.

[2.5.2]

- Bug Fixes
  - Fixed ERR051119 to ignore the nak flag when IGNACK=1 in LPI2C_MasterCheckAndClearError.

[2.5.1]

- Bug Fixes
  - Added bus stop incase of bus stall in LPI2C_MasterTransferBlocking.
- Improvements
  - Release peripheral from reset if necessary in init function.

[2.5.0]

- New Features
  - Added new function LPI2C_SlaveEnableAckStall to enable or disable ACKSTALL.

[2.4.1]

- Improvements
  - Before master transfer with transactional APIs, enable master function while disable slave function and vise versa for slave transfer to avoid the one affecting the other.

[2.4.0]

- Improvements
  - Split some functions, fixed CCM problem in file fsl_lpi2c.c.
- Bug Fixes
  - Fixed bug in LPI2C_MasterInit that the MCFGR2's value set in LPI2C_MasterSetBaudRate may be overwritten by mistake.

[2.3.2]

- Improvements
  - Initialized the EDMA configuration structure in the LPI2C EDMA driver.

[2.3.1]

- Improvements
  - Updated LPI2C_GetCyclesForWidth to add the parameter of minimum cycle, because for master SDA/SCL filter, master bus idle/pin low timeout and slave SDA/SCL filter configuration, 0 means disabling the feature and cannot be used.
- Bug Fixes

  - Fixed bug in LPI2C_SlaveTransferHandleIRQ that when restart detect event happens the transfer structure should not be cleared.
  - Fixed bug in LPI2C_RunTransferStateMachine, that when only slave address is transferred or there is still data remaining in tx FIFO the last byte's nack cannot be ignored.
  - Fixed bug in slave filter doze enable, that when FILTDZ is set it means disable rather than enable.
  - Fixed bug in the usage of LPI2C_GetCyclesForWidth. First its return value cannot be used directly to configure the slave FILTSDA, FILTSCL, DATAVD or CLKHOLD, because the real cycle width for them should be FILTSDA+3, FILTSCL+3, FILTSCL+DATAVD+3 and CLKHOLD+3. Second when cycle period is not affected by the prescaler value, prescaler value should be passed as 0 rather than 1.
  - Fixed wrong default setting for LPI2C slave. If enabling the slave tx SCL stall, then the default clock hold time should be set to 250ns according to I2C spec for 100kHz standard mode baudrate.
  - Fixed bug that before pushing command to the tx FIFO the FIFO occupation should be checked first in case FIFO overflow.

[2.3.0]

- New Features
  - Supported reading more than 256 bytes of data in one transfer as master.
  - Added API LPI2C_GetInstance.
- Bug Fixes
  - Fixed bug in LPI2C_MasterTransferAbortEDMA, LPI2C_MasterTransferAbort and LPI2C_MasterTransferHandleIRQ that before sending stop signal whether master is active and whether stop signal has been sent should be checked, to make sure no FIFO error or bus error will be caused.
  - Fixed bug in LPI2C master EDMA transactional layer that the bus error cannot be caught and returned by user callback, by monitoring bus error events in interrupt handler.
  - Fixed bug in LPI2C_GetCyclesForWidth that the parameter used to calculate clock cycle should be 2^prescaler rather than prescaler.
  - Fixed bug in LPI2C_MasterInit that timeout value should be configured after baudrate, since the timeout calculation needs prescaler as parameter which is changed during baudrate configuration.
  - Fixed bug in LPI2C_MasterTransferHandleIRQ and LPI2C_RunTransferStateMachine that when master writes with no stop signal, need to first make sure no data remains in the tx FIFO before finishes the transfer.

[2.2.0]

- Bug Fixes
  - Fixed issue that the SCL high time, start hold time and stop setup time do not meet I2C specification, by changing the configuration of data valid delay, setup hold delay, clock high and low parameters.
  - MISRA C-2012 issue fixed.
    - Fixed rule 8.4, 13.5, 17.7, 20.8.

[2.1.12]

- Bug Fixes
  - Fixed MISRA advisory 15.5 issues.

[2.1.11]

- Bug Fixes
  - Fixed the bug that, during master non-blocking transfer, after the last byte is sent/received, the kLPI2C_MasterNackDetectFlag is expected, so master should not check and clear kLPI2C_MasterNackDetectFlag when remainingBytes is zero, in case FIFO is emptied when stop command has not been sent yet.
  - Fixed the bug that, during non-blocking transfer slave may nack master while master is busy filling tx FIFO, and NDF may not be handled properly.

[2.1.10]

- Bug Fixes
  - MISRA C-2012 issue fixed.
    - Fixed rule 10.3, 14.4, 15.5.
  - Fixed unaligned access issue in LPI2C_RunTransferStateMachine.
  - Fixed uninitialized variable issue in LPI2C_MasterTransferHandleIRQ.
  - Used linked TCD to disable tx and enable rx in read operation to fix the issue that for platform sharing the same DMA request with tx and rx, during LPI2C read operation if interrupt with higher priority happened exactly after command was sent and before tx disabled, potentially both tx and rx could trigger dma and cause trouble.
  - Fixed MISRA issues.
    - Fixed rules 10.1, 10.3, 10.4, 11.6, 11.9, 14.4, 17.7.
  - Fixed the waitTimes variable not re-assignment issue for each byte read.
- New Features
  - Added the IRQHandler for LPI2C5 and LPI2C6 instances.
- Improvements
  - Updated the LPI2C_WAIT_TIMEOUT macro to unified name I2C_RETRY_TIMES.

[2.1.9]

- Bug Fixes
  - Fixed Coverity issue of unchecked return value in I2C_RTOS_Transfer.
  - Fixed Coverity issue of operands did not affect the result in LPI2C_SlaveReceive and LPI2C_SlaveSend.
  - Removed STOP signal wait when NAK detected.
  - Cleared slave repeat start flag before transmission started in LPI2C_SlaveSend/LPI2C_SlaveReceive. The issue was that LPI2C_SlaveSend/LPI2C_SlaveReceive did not handle with the reserved repeat start flag. This caused the next slave to send a break, and the master was always in the receive data status, but could not receive data.

[2.1.8]

- Bug Fixes
  - Fixed the transfer issue with LPI2C_MasterTransferNonBlocking, kLPI2C_TransferNoStopFlag, with the wait transfer done through callback in a way of not doing a blocking transfer.
  - Fixed the issue that STOP signal did not appear in the bus when NAK event occurred.

[2.1.7]

- Bug Fixes
  - Cleared the stopflag before transmission started in LPI2C_SlaveSend/LPI2C_SlaveReceive. The issue was that LPI2C_SlaveSend/LPI2C_SlaveReceive did not handle with the reserved stop flag and caused the next slave to send a break, and the master always stayed in the receive data status but could not receive data.

[2.1.6]

- Bug Fixes
  - Fixed driver MISRA build error and C++ build error in LPI2C_MasterSend and LPI2C_SlaveSend.
  - Reset FIFO in LPI2C Master Transfer functions to avoid any byte still remaining in FIFO during last transfer.
  - Fixed the issue that LPI2C_MasterStop did not return the correct NAK status in the bus for second transfer to the non-existing slave address.

[2.1.5]

- Bug Fixes
  - Extended the Driver IRQ handler to support LPI2C4.
  - Changed to use ARRAY_SIZE(kLpi2cBases) instead of FEATURE COUNT to decide the array size for handle pointer array.

[2.1.4]

- Bug Fixes
  - Fixed the LPI2C_MasterTransferEDMA receive issue when LPI2C shared same request source with TX/RX DMA request. Previously, the API used scatter-gather method, which handled the command transfer first, then the linked TCD which was pre-set with the receive data transfer. The issue was that the TX DMA request and the RX DMA request were both enabled, so when the DMA finished the first command TCD transfer and handled the receive data TCD, the TX DMA request still happened due to empty TX FIFO. The result was that the RX DMA transfer would start without waiting on the expected RX DMA request.
  - Fixed the issue by enabling IntMajor interrupt for the command TCD and checking if there was a linked TCD to disable the TX DMA request in LPI2C_MasterEDMACallback API.

[2.1.3]

- Improvements
  - Added LPI2C_WATI_TIMEOUT macro to allow the user to specify the timeout times for waiting flags in functional API and blocking transfer API.
  - Added LPI2C_MasterTransferBlocking API.

[2.1.2]

- Bug Fixes
  - In LPI2C_SlaveTransferHandleIRQ, reset the slave status to idle when stop flag was detected.

[2.1.1]

- Bug Fixes
  - Disabled the auto-stop feature in eDMA driver. Previously, the auto-stop feature was enabled at transfer when transferring with stop flag. Since transfer was without stop flag and the auto-stop feature was enabled, when starting a new transfer with stop flag, the stop flag would be sent before the new transfer started, causing unsuccesful sending of the start flag, so the transfer could not start.
  - Changed default slave configuration with address stall false.

[2.1.0]

- Improvements
  - API name changed:
    * LPI2C_MasterTransferCreateHandle -> LPI2C_MasterCreateHandle.
    * LPI2C_MasterTransferGetCount -> LPI2C_MasterGetTransferCount.
    * LPI2C_MasterTransferAbort -> LPI2C_MasterAbortTransfer.
    * LPI2C_MasterTransferHandleIRQ -> LPI2C_MasterHandleInterrupt.
    * LPI2C_SlaveTransferCreateHandle -> LPI2C_SlaveCreateHandle.
    * LPI2C_SlaveTransferGetCount -> LPI2C_SlaveGetTransferCount.
    * LPI2C_SlaveTransferAbort -> LPI2C_SlaveAbortTransfer.
    * LPI2C_SlaveTransferHandleIRQ -> LPI2C_SlaveHandleInterrupt.

[2.0.0]

- Initial version.

LPI2C_EDMA

[2.4.9]

- Bug Fixes
  - Fixed CERT-C issues.
  - Fixed MISRA issues.

[2.4.8]

- Bug Fixes
  - Fixed MISRA 10.6 and 14.3 issues.

[2.4.7]

- Bug Fixes
  - Fixed incorrect TX FIFO size (maxTxFifo) in LPI2C_MasterTransferEDMA() and LPI2C_MasterTransferEdmaHandleIRQ().

[2.4.6]

- Bug Fixes
  - Fixed static analysis identified issues.

[2.4.5]

- Improvements
  - Added condition to IRQ handler to check whether the interrupt is enabled - kLPI2C_MasterTxReadyFlag.

[2.4.4]

- Improvements
  - Added support for 2KB data transfer

[2.4.3]

- Improvements
  - Added support for separated IRQ handlers.

[2.4.2]

- Improvements
  - Add EDMA ext API to accommodate more types of EDMA.

[2.4.1]

- Refer LPI2C driver change log 2.0.0 to 2.4.1

LPIT

[2.1.3]

- Bug Fixes
  - Fixed doxygen generation warnings.

[2.1.2]

- Bug Fixes
  - Fix CERT INT31-C issues.

[2.1.1]

- Improvements
  - Release peripheral from reset if necessary in init function.

[2.1.0]

- Improvements
  - Add new function LPIT_SetTimerValue to set timeout period.

[2.0.2]

- Improvements
  - Improved LPIT_SetTimerPeriod implementation, configure timeout value with LPIT ticks minus 1 generate more correct interval.
  - Added timeout value configuration check for LPIT_SetTimerPeriod, at least input 3 ticks for calling LPIT_SetTimerPeriod.
- Bug Fixes
  - Fixed MISRA C-2012 rule 17.7 violations.

[2.0.1]

- Bug Fixes
  - MISRA C-2012 issue fixed.
    - Fixed rules, containing: rule-10.3, rule-14.4, rule-15.5.

[2.0.0]

- Initial version.

LPSPI

[2.7.8]

- Bug Fixes
  - Fixed CERT-C issues.
  - Fixed MISRA issues.
- Improvements
  - Improved HIS CCM.

[2.7.7]

- Bug Fixes
  - Fixed timing parameter setting sequence issue in LPSPI_MasterInit, the timing parameter should only be configured when LPSPI master is disabled.
- Improvements
  - Improved performance of LPSPI_MasterTransferBlocking().

[2.7.6]

- Bug Fixes
  - Fixed calculation of SCK-to-SCK delay.

[2.7.5]

- Bug Fixes
  - Fixed MISRA 10.3, 12,2 and 10.7 issues.

[2.7.4]

- Bug Fixes
  - Clear WIDTH bits from the TCR register before writing a new value in LPSPI_MasterTransferBlocking().

[2.7.3]

- Improvements
  - Added timeout for while loop in LPSPI_MasterTransferWriteAllTxData().
  - Make SPI_RETRY_TIMES configurable by CONFIG_SPI_RETRY_TIMES.

[2.7.2]

- Bug Fixes
  - Fixed coverity issues.

[2.7.1]

- Bug Fixes
  - Workaround for errata ERR050607
  - Workaround for errata ERR010655

[2.7.0]

- New Feature
  - Added common IRQ handler entry LPSPI_DriverIRQHandler.

[2.6.10]

- Improvements
  - Conditionally compile interrupt handling code to solve the problem of using this driver on CPU cores that do not support interrupts.

[2.6.9]

- Bug Fixes
  - Fixed reading of TCR register
  - Workaround for errata ERR050606

[2.6.8]

- Bug Fixes
  - Fixed build error when SPI_RETRY_TIMES is defined to non-zero value.

[2.6.7]

- Bug Fixes
  - Fixed the txData from void * to const void * in transmit API _lpspi_master_handle and _lpspi_slave_handle.

[2.6.6]

- Bug Fixes
  - Added LPSPI register init in LPSPI_MasterInit incase of LPSPI register exist.

[2.6.5]

- Improvements
  - Introduced FSL_FEATURE_LPSPI_HAS_NO_PCSCFG and FSL_FEATURE_LPSPI_HAS_NO_MULTI_WIDTH for conditional compile.
  - Release peripheral from reset if necessary in init function.

[2.6.4]

- Bug Fixes
  - Added LPSPI6_DriverIRQHandler for LPSPI6 instance.

[2.6.3]

- Hot Fixes
  - Added macro switch in function LPSPI_Enable about ERRATA051472.

[2.6.2]

- Bug Fixes
  - Disabled lpspi before LPSPI_MasterSetBaudRate incase of LPSPI opened.

[2.6.1]

- Bug Fixes
  - Fixed return value while calling LPSPI_WaitTxFifoEmpty in function LPSPI_MasterTransferNonBlocking.

[2.6.0]

- Feature
  - Added the new feature of multi-IO SPI .

[2.5.3]

- Bug Fixes
  - Fixed 3-wire txmask of handle vaule reentrant issue.

[2.5.2]

- Bug Fixes
  - Workaround for errata ERR051588 by clearing FIFO after transmit underrun occurs.

[2.5.1]

- Bug Fixes
  - Workaround for errata ERR050456 by resetting the entire module using LPSPIn_CR[RST] bit.

[2.5.0]

- Bug Fixes
  - Workaround for errata ERR011097 to wait the TX FIFO to go empty when writing TCR register and TCR[TXMSK] value is 1.
  - Added API LPSPI_WaitTxFifoEmpty for wait the txfifo to go empty.

[2.4.7]

- Bug Fixes
  - Fixed bug that the SR[REF] would assert if software disabled or enabled the LPSPI module in LPSPI_Enable.

[2.4.6]

- Improvements
  - Moved the configuration of registers for the 3-wire lpspi mode to the LPSPI_MasterInit and LPSPI_SlaveInit function.

[2.4.5]

- Improvements
  - Improved LPSPI_MasterTransferBlocking send performance when frame size is 1-byte.

[2.4.4]

- Bug Fixes
  - Fixed LPSPI_MasterGetDefaultConfig incorrect default inter-transfer delay calculation.

[2.4.3]

- Bug Fixes
  - Fixed bug that the ISR response speed is too slow on some platforms, resulting in the first transmission of overflow, Set proper RX watermarks to reduce the ISR response times.

[2.4.2]

- Bug Fixes
  - Fixed bug that LPSPI_MasterTransferBlocking will modify the parameter txbuff and rxbuff pointer.

[2.4.1]

- Bug Fixes
  - Fixed bug that LPSPI_SlaveTransferNonBlocking can't detect RX error.

[2.4.0]

- Improvements
  - Split some functions, fixed CCM problem in file fsl_lpspi.c.

[2.3.1]

- Improvements
  - Initialized the EDMA configuration structure in the LPSPI EDMA driver.
- Bug Fixes
  - Fixed bug that function LPSPI_MasterTransferBlocking should return after the transfer complete flag is set to make sure the PCS is re-asserted.

[2.3.0]

- New Features
  - Supported the master configuration of sampling the input data using a delayed clock to improve slave setup time.

[2.2.1]

- Bug Fixes
  - Fixed bug in LPSPI_SetPCSContinous when disabling PCS continous mode.

[2.2.0]

- Bug Fixes
  - Fixed bug in 3-wire polling and interrupt transfer that the received data is not correct and the PCS continous mode is not working.

[2.1.0]

- Improvements
  - Improved LPSPI_SlaveTransferHandleIRQ to fill up TX FIFO instead of write one data to TX register which improves the slave transmit performance.
  - Added new functional APIs LPSPI_SelectTransferPCS and LPSPI_SetPCSContinous to support changing PCS selection and PCS continous mode.
- Bug Fixes
  - Fixed bug in non-blocking and EDMA transfer APIs that kStatus_InvalidArgument is returned if user configures 3-wire mode and full-duplex transfer at the same time, but transfer state is already set to kLPSPI_Busy by mistake causing following transfer can not start.
  - Fixed bug when LPSPI slave using EDMA way to transfer, tx should be masked when tx data is null, otherwise in 3-wire mode which tx/rx use the same pin, the received data will be interfered.

[2.0.5]

- Improvements
  - Added timeout mechanism when waiting certain states in transfer driver.
- Bug Fixes
  - Fixed the bug that LPSPI can not transfer large data using EDMA.
  - Fixed MISRA 17.7 issues.
  - Fixed variable overflow issue introduced by MISRA fix.
  - Fixed issue that rxFifoMaxBytes should be calculated according to transfer width rather than FIFO width.
  - Fixed issue that completion flag was not cleared after transfer completed.

[2.0.4]

- Bug Fixes
  - Fixed in LPSPI_MasterTransferBlocking that master rxfifo may overflow in stall condition.
  - Eliminated IAR Pa082 warnings.
  - Fixed MISRA issues.
    - Fixed rules 10.1, 10.3, 10.4, 10.6, 11.9, 14.2, 14.4, 15.7, 17.7.

[2.0.3]

- Bug Fixes
  - Removed LPSPI_Reset from LPSPI_MasterInit and LPSPI_SlaveInit, because this API may glitch the slave select line. If needed, call this function manually.

[2.0.2]

- New Features
  - Added dummy data set up API to allow users to configure the dummy data to be transferred.
  - Enabled the 3-wire mode, SIN and SOUT pins can be configured as input/output pin.

[2.0.1]

- Bug Fixes
  - Fixed the bug that the clock source should be divided by the PRESCALE setting in LPSPI_MasterSetDelayTimes function.
  - Fixed the bug that LPSPI_MasterTransferBlocking function would hang in some corner cases.
- Optimization
  - Added #ifndef/#endif to allow user to change the default TX value at compile time.

[2.0.0]

- Initial version.

LPSPI_EDMA

[2.4.11]

- Bug Fixes
  - Fixed CERT-C issues.
  - Fixed MISRA issues.
- Improvements
  - Improved HIS CCM in LPSPI_MasterTransferEDMALite() and LPSPI_SlaveTransferEDMA().

[2.4.10]

- Bug Fixes
  - Fixed MISRA 10.6 issues.

[2.4.9]

- Improvements
  - Removed unused code from LPSPI_SeparateEdmaReadData().

[2.4.8]

- Improvements
  - Added timeout for while loop in EDMA_LpspiMasterCallback() and EDMA_LpspiSlaveCallback().

[2.4.7]

- Bug Fixes
  - Add macro LPSPI_ALIGN_TCD_SIZE_MASK to align an address to edma_tcd_t size.

[2.4.6]

- Improvements
  - Increased transmit FIFO watermark to ensure whole transmit FIFO will be used during data transfer.

[2.4.5]

- Bug Fixes
  - Fixed reading of TCR register
  - Workaround for errata ERR050606

[2.4.4]

- Improvements
  - Add EDMA ext API to accommodate more types of EDMA.

[2.4.3]

- Improvements
  - Supported 32K bytes transmit in DMA, improve the max datasize in LPSPI_MasterTransferEDMALite.

[2.4.2]

- Improvements
  - Added callback status in EDMA_LpspiMasterCallback and EDMA_LpspiSlaveCallback to check transferDone.

[2.4.1]

- Improvements
  - Add the TXMSK wait after TCR setting.

[2.4.0]

- Improvements
  - Separated LPSPI_MasterTransferEDMA functions to LPSPI_MasterTransferPrepareEDMA and LPSPI_MasterTransferEDMALite to optimize the process of transfer.

LPTMR

[2.2.1]

- Bug Fixes
  - Fix CERT INT31-C issues.

[2.2.0]

- Improvements
  - Updated lptmr_prescaler_clock_select_t, only define the valid options.

[2.1.1]

- Improvements
  - Updated the characters from "PTMR" to "LPTMR" in "FSL_FEATURE_PTMR_HAS_NO_PRESCALER_CLOCK_SOURCE_1_SUPPORT" feature definition.

[2.1.0]

- Improvements
  - Implement for some special devices' not supporting for all clock sources.
- Bug Fixes
  - Fixed issue when accessing CMR register.

[2.0.2]

- Bug Fixes
  - Fixed MISRA-2012 issues.
    - Rule 10.1.

[2.0.1]

- Improvements
  - Updated the LPTMR driver to support 32-bit CNR and CMR registers in some devices.

[2.0.0]

- Initial version.

LPUART

[2.11.0]

- New Feature
  - Added support to enable transmitter RTS and configure transmitter RTS polarity.
  - Added support to enable Transmit Data Inversion.
- Improvements

  - Updated to work on DSC platform.

[2.10.0]

- New Feature
  - Added support to configure RTS watermark.

[2.9.4]

- Improvements
  - Merged duplicate code.

[2.9.3]

- Improvements
  - Added timeout for while loops in LPUART_Deinit().

[2.9.2]

- Bug Fixes
  - Fixed coverity issues.

[2.9.1]

- Bug Fixes
  - Fixed coverity issues.

[2.9.0]

- New Feature
  - Added support for swap TXD and RXD pins.
  - Added common IRQ handler entry LPUART_DriverIRQHandler.

[2.8.3]

- Improvements
  - Conditionally compile interrupt handling code to solve the problem of using this driver on CPU cores that do not support interrupts.

[2.8.2]

- Bug Fix
  - Fixed the bug that LPUART_TransferEnable16Bit controled by wrong feature macro.

[2.8.1]

- Bug Fixes
  - Fixed issue for MISRA-2012 check.
    - Fixed rule-5.3, rule-5.8, rule-10.4, rule-11.3, rule-11.8.

[2.8.0]

- Improvements
  - Added support of DATA register for 9bit or 10bit data transmit in write and read API. Such as: LPUART_WriteBlocking16bit, LPUART_ReadBlocking16bit, LPUART_TransferEnable16Bit LPUART_WriteNonBlocking16bit, LPUART_ReadNonBlocking16bit.

[2.7.7]

- Bug Fixes
  - Fixed the bug that baud rate calculation overflow when srcClock_Hz is 528MHz.

[2.7.6]

- Bug Fixes
  - Fixed LPUART_EnableInterrupts and LPUART_DisableInterrupts bug that blocks if the LPUART address doesn't support exclusive access.

[2.7.5]

- Improvements
  - Release peripheral from reset if necessary in init function.

[2.7.4]

- Improvements
  - Added support for atomic register accessing in LPUART_EnableInterrupts and LPUART_DisableInterrupts.

[2.7.3]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 15.7.

[2.7.2]

- Bug Fix
  - Fixed the bug that the OSR calculation error when lupart init and lpuart set baud rate.

[2.7.1]

- Improvements
  - Added support for LPUART_BASE_PTRS_NS in security mode in file fsl_lpuart.c.

[2.7.0]

- Improvements
  - Split some functions, fixed CCM problem in file fsl_lpuart.c.

[2.6.0]

- Bug Fixes
  - Fixed bug that when there are multiple lpuart instance, unable to support different ISR.

[2.5.3]

- Bug Fixes
  - Fixed comments by replacing unused status flags kLPUART_NoiseErrorInRxDataRegFlag and kLPUART_ParityErrorInRxDataRegFlag with kLPUART_NoiseErrorFlag and kLPUART_ParityErrorFlag.

[2.5.2]

- Bug Fixes
  - Fixed bug that when setting watermark for TX or RX FIFO, the value may exceed the maximum limit.
- Improvements
  - Added check in LPUART_TransferDMAHandleIRQ and LPUART_TransferEdmaHandleIRQ to ensure if user enables any interrupts other than transfer complete interrupt, the dma transfer is not terminated by mistake.

[2.5.1]

- Improvements
  - Use separate data for TX and RX in lpuart_transfer_t.
- Bug Fixes
  - Fixed bug that when ring buffer is used, if some data is received in ring buffer first before calling LPUART_TransferReceiveNonBlocking, the received data count returned by LPUART_TransferGetReceiveCount is wrong.

[2.5.0]

- Bug Fixes
  - Added missing interrupt enable masks kLPUART_Match1InterruptEnable and kLPUART_Match2InterruptEnable.
  - Fixed bug in LPUART_EnableInterrupts, LPUART_DisableInterrupts and LPUART_GetEnabledInterrupts that the BAUD[LBKDIE] bit field should be soc specific.
  - Fixed bug in LPUART_TransferHandleIRQ that idle line interrupt should be disabled when rx data size is zero.
  - Deleted unused status flags kLPUART_NoiseErrorInRxDataRegFlag and kLPUART_ParityErrorInRxDataRegFlag, since firstly their function are the same as kLPUART_NoiseErrorFlag and kLPUART_ParityErrorFlag, secondly to obtain them one data word must be read out thus interfering with the receiving process.
  - Fixed bug in LPUART_GetStatusFlags that the STAT[LBKDIF], STAT[MA1F] and STAT[MA2F] should be soc specific.
  - Fixed bug in LPUART_ClearStatusFlags that tx/rx FIFO is reset by mistake when clearing flags.

  - Fixed bug in LPUART_TransferHandleIRQ that while clearing idle line flag the other bits should be masked in case other status bits be cleared by accident.
  - Fixed bug of race condition during LPUART transfer using transactional APIs, by disabling and re-enabling the global interrupt before and after critical operations on interrupt enable register.
  - Fixed DMA/eDMA transfer blocking issue by enabling tx idle interrupt after DMA/eDMA transmission finishes.
- New Features
  - Added APIs LPUART_GetRxFifoCount/LPUART_GetTxFifoCount to get rx/tx FIFO data count.
  - Added APIs LPUART_SetRxFifoWatermark/LPUART_SetTxFifoWatermark to set rx/tx FIFO water mark.

[2.4.1]

- Bug Fixes
  - Fixed MISRA advisory 17.7 issues.

[2.4.0]

- New Features
  - Added APIs to configure 9-bit data mode, set slave address and send address.

[2.3.1]

- Bug Fixes
  - Fixed MISRA advisory 15.5 issues.

[2.3.0]

- Improvements
  - Modified LPUART_TransferHandleIRQ so that txState will be set to idle only when all data has been sent out to bus.
  - Modified LPUART_TransferGetSendCount so that this API returns the real byte count that LPUART has sent out rather than the software buffer status.
  - Added timeout mechanism when waiting for certain states in transfer driver.

[2.2.8]

- Bug Fixes
  - Fixed issue for MISRA-2012 check.
    - Fixed rule-10.3, rule-14.4, rule-15.5.
  - Eliminated Pa082 warnings by assigning volatile variables to local variables and using local variables instead.
  - Fixed MISRA issues.
    - Fixed rules 10.1, 10.3, 10.4, 10.8, 14.4, 11.6, 17.7.
- Improvements

- Added check for kLPUART_TransmissionCompleteFlag in LPUART_WriteBlocking, LPUART_TransferHandleIRQ, LPUART_TransferSendDMACallback and LPUART_SendEDMACallback to ensure all the data would be sent out to bus.
- Rounded up the calculated sbr value in LPUART_SetBaudRate and LPUART_Init to achieve more acurate baudrate setting. Changed osr from uint32_t to uint8_t since osr's bigest value is 31.
- Modified LPUART_ReadBlocking so that if more than one receiver errors occur, all status flags will be cleared and the most severe error status will be returned.

[2.2.7]

- Bug Fixes
  - Fixed issue for MISRA-2012 check.
    - Fixed rule-12.1, rule-17.7, rule-14.4, rule-13.3, rule-14.4, rule-10.4, rule-10.8, rule-10.3, rule-10.7, rule-10.1, rule-11.6, rule-13.5, rule-11.3, rule-13.2, rule-8.3.

[2.2.6]

- Bug Fixes
  - Fixed the issue of register's being in repeated reading status while dealing with the IRQ routine.

[2.2.5]

- Bug Fixes
  - Do not set or clear the TIE/RIE bits when using LPUART_EnableTxDMA and LPUART_EnableRxDMA.

[2.2.4]

- Improvements
  - Added hardware flow control function support.
  - Added idle-line-detecting feature in LPUART_TransferNonBlocking function. If an idle line is detected, a callback is triggered with status kStatus_LPUART_IdleLineDetected returned. This feature may be useful when the received Bytes is less than the expected received data size. Before triggering the callback, data in the FIFO (if has FIFO) is read out, and no interrupt will be disabled, except for that the receive data size reaches 0.
  - Enabled the RX FIFO watermark function. With the idle-line-detecting feature enabled, users can set the watermark value to whatever you want (should be less than the RX FIFO size). Data is received and a callback will be triggered when data receive ends.

[2.2.3]

- Improvements
  - Changed parameter type in LPUART_RTOS_Init struct from rtos_lpuart_config to lpuart_rtos_config_t.
- Bug Fixes

  - Disabled LPUART receive interrupt instead of all NVICs when reading data from ring buffer. Otherwise when the ring buffer is used, receive nonblocking method will disable all NVICs to protect the ring buffer. This may has a negative effect on other IPs that are using the interrupt.

[2.2.2]

- Improvements
  - Added software reset feature support.
  - Added software reset API in LPUART_Init.

[2.2.1]

- Improvements
  - Added separate RX/TX IRQ number support.

[2.2.0]

- Improvements
  - Added support of 7 data bits and MSB.

[2.1.1]

- Improvements
  - Removed unnecessary check of event flags and assert in LPUART_RTOS_Receive.
  - Added code to always wait for RX event flag in LPUART_RTOS_Receive.

[2.1.0]

- Improvements
  - Update transactional APIs.

LPUART_EDMA

[2.4.0]

- Refer LPUART driver change log 2.1.0 to 2.4.0

LTC

[2.0.18]

- Bug fix:
- Fix CERT INT30-C and INT31-C compliance.

[2.0.17]

- Bug fix:
  - Fix CMAC for payloads over one block, and if BRIC is present on the device, remove XCBC and “decrypt key” functionality

[2.0.16]

- Bug fix:
  - Fix unitialized GCC warning in LTC_AES_GenerateDecryptKey().

[2.0.15]

- Bug fix:
  - Fix MISRA-2012 issues.

[2.0.14]

- Improvements:
  - Add feature macro FSL_FEATURE_LTC_HAS_NO_CLOCK_CONTROL_BIT into LTC_Deinit function.

[2.0.13]

- Improvements:
  - Add feature macro FSL_FEATURE_LTC_HAS_NO_CLOCK_CONTROL_BIT into LTC_init function.

[2.0.12]

- Bug fix:
  - Fix AES Decrypt in CBC modes fail when used kLTC_DecryptKey.

[2.0.11]

- Bug fix:
  - Fix MISRA-2012 issues.

[2.0.10]

- Bug fix:
  - Fix MISRA-2012 issues.

[2.0.9]

- Bug fix:
  - Fix sign-compare warning in ltc_set_context and in ltc_get_context.

[2.0.8]

- Bug fix:
  - Fixed coverity issues introduced in 2.0.7

[2.0.7]

- Bug fix:
  - Fixed MISRA-2012 issues
    - Fixed rule 15.7. 10.1. 8.6. 10.7. 10.3, 11.6, 11.3, 10.8, 16.1.

[2.0.6]

- Bug fix:
  - Fixed [KPSDK-23603][LTC] AES Decrypt in ECB and CBC modes fail when ciphertext size > 0xff0 bytes.

[2.0.5]

- Improvements:
  - Fixed MISRA issues
    - Fixed rule 15.7, rule 14.4, rule 10.4, rule 18.4, rule 17.7, 5.7, 12.2.

[2.0.4]

- Improvements:
  - Constant LTC_PKHA_CompareBigNum() processing time.

[2.0.3]

- Bug fix:
  - Fixed LTC_PKHA_CompareBigNum() in case an integer argument is an array of all zeros.

[2.0.2]

- Bug fix:
  - Fixed [KPSDK-10932][LTC][SHA] LTC_HASH() blocks indefinitely when message size exceeds 4080 Bytes.

[2.0.1]

- Bug fix:
  - Fixed c++ build warning in block_to_ififo() functions.

[2.0.0]

- Initial version.

MCM

[2.2.1]

- Improvements
  - Add feature to support platforms that do not have the MCM CPCR (Core Platform Control Register).

[2.2.0]

- Improvements
  - Support platforms with less features.

[2.1.0]

- Others
  - Remove byteID from mcm_lmem_fault_attribute_t for document update.

[2.0.0]

- Initial version.

MSCM

[2.0.0]

- Initial version.

MU

[2.8.5]

- Improvements
  - Guard CIER0 core-interrupt register access with FSL_FEATURE_MU_NO_CIER0_REG for devices whose MU instance has no CIER0 register (e.g. MCXE32B MU2).

[2.8.4]

- Improvements
  - Minor update to allow use in DSC projects.

[2.8.3]

- Improvements
  - Fix build failure when driver clock control disabled.
  - Make MU_GetInstance() static function

[2.8.2]

- Improvements
  - Fix MISRA C-2012 Rule 20.8 violations.

[2.8.1]

- Bug Fixes
  - Avoid incorrect MU_BUSY_POLL_COUNT macro use.

[2.8.0]

- New Features
  - Added MU1_BUSY_POLL_COUNT parameter to prevent infinite polling loops in MU operations.
  - Added timeout mechanism to all polling loops in MU driver code.
- Improvements
  - Updated function signatures to return status codes for better error handling:
    - Changed MU_ResetBothSides to return status_t instead of void
    - Updated MU_SendMsg to return status_t for timeout indication
    - Added new function MU_ReceiveMsgTimeout() to include timeout mechanism.
  - Enhanced documentation across all functions to clarify timeout behavior and return values.

[2.7.0]

- New Features
  - Added API MU_GetRxStatusFlags.

[2.6.0]

- New Features
  - Added API MU_GetInterruptsPending.

[2.5.1]

- Bug Fixes
  - Fixed the bug that MU_TriggerGeneralPurposeInterrupts and MU_TriggerInterrupts may trigger previous triggered general purpose interrupts again by mistake.

[2.5.0]

- New Features
  - Supported more than 4 general purpose interrupts.
  - Added seperate APIs for general purpose interrupts.

[2.4.0]

- Improvements
  - Supported the case that some features only avaiable with specific instances. These features include Hardware Reset, Boot Peer Core, Hold Reset. When using the features with instances which don't support them, driver will report error.

[2.3.3]

- Improvements
  - Release peripheral from reset if necessary in init function.

[2.3.2]

- Improvements
  - Supported platforms which don't have CCR0[RSTH], CCR0[CLKE], CCR0[HR], CCR0[HRM].

[2.3.1]

- Bug Fixes
  - Fixed build error for platforms which have CCR0[RSTH], but no CCR0[NMI].

[2.3.0]

- New features
  - Added support for i.MX RT7xx.

[2.2.1]

- Bug Fixes
  - Fixed issue that MU_GetInstance() is defined but never used.

[2.2.0]

- New features
  - Added support for i.MX RT118x.
- Bug Fixes
  - Fixed general purpose interrupt bug.
- Other Changes
  - Change _mu_interrupt_trigger item value.

[2.1.2]

- Bug Fixes
  - Fixed bug that general purpose interrupt can't be configured.

[2.1.1]

- Bug Fixes
  - Fixed MISRA C-2012 issues.

[2.1.0]

- Improvements
  - Added new enum mu_msg_reg_index_t.

[2.0.0]

- Initial version.

PORT

[2.5.1]

- Bug Fixes
  - Fix CERT INT31-C issues.
  - Fixed the violations of MISRA C-2012 rules: 10.1.

[2.5.0]

- Bug Fixes
  - Correct the kPORT_MuxAsGpio for some platforms.

[2.4.1]

- Bug Fixes
  - Fixed the violations of MISRA C-2012 rules: 10.1, 10.8 and 14.4.

[2.4.0]

- New Features
  - Updated port_pin_config_t to support input buffer and input invert.

[2.3.0]

- New Features
  - Added new APIs for Electrical Fast Transient(EFT) detect.
  - Added new API to configure port voltage range.

[2.2.0]

- New Features
  - Added new api PORT_EnablePinDoubleDriveStrength.

[2.1.1]

- Bug Fixes
  - Fixed the violations of MISRA C-2012 rules: 10.1, 10.4□11.3□11.8, 14.4.

[2.1.0]

- New Features
  - Updated the driver code to adapt the case of the interrupt configurations in GPIO module. Will move the pin configuration APIs to GPIO module.

[2.0.2]

- Other Changes
  - Added feature guard macros in the driver.

[2.0.1]

- Other Changes
  - Added “const” in function parameter.
  - Updated some enumeration variables’ names.

RTC

[2.4.0]

- New features
  - Add support for RTC clock output.
  - Add support for RTC time seconds interrupt configuration.

[2.3.3]

- Bug Fixes
  - Fix RTC_GetDatetime function validating datetime issue.

[2.3.2]

- Improvements
  - Handle errata 010716: Disable the counter before setting alarm register and then reenable the counter.

[2.3.1]

- Bug Fixes
  - Fixed CERT INT31-C violations.

[2.3.0]

- Improvements
  - Added API RTC_EnableLPOClock to set 1kHz LPO clock.
  - Added API RTC_EnableCrystalClock to replace API RTC_SetClockSource.

[2.2.2]

- Improvements
  - Refine _rtc_interrupt_enable order.

[2.2.1]

- Bug Fixes
  - Fixed the issue of Pa082 warning.
  - Fixed the issue of bit field mask checking.
  - Fixed the issue of hard code in RTC_Init.

[2.2.0]

- Bug Fixes
  - Fixed MISRA C-2012 issue.
    - Fixed rule contain: rule-17.7, rule-14.4, rule-10.4, rule-10.7, rule-10.1, rule-10.3.
  - Fixed central repository code formatting issue.
- Improvements
  - Added an API for enabling wakeup pin.

[2.1.0]

- Improvements
  - Added feature macro check for many features.

[2.0.0]

- Initial version.

SEMA42

[2.1.2]

- Improvements
  - Fix MISRA C-2012 Rule 20.8 violations.

[2.1.1]

- Improvements
  - Updated SEMA42_TryLock function to avoid unsigned integer operations wrap issue.

[2.1.0]

- New Features
  - Added SEMA42_BUSY_POLL_COUNT parameter to prevent infinite polling loops in SEMA42 operations.
  - Added timeout mechanism to all polling loops in SEMA42 driver code.
- Improvements
  - Updated SEMA42_Lock function to return status_t instead of void for better error handling.
  - Enhanced documentation to clarify timeout behavior and return values.

[2.0.4]

- Improvements
  - Release peripheral from reset if necessary in init function.

[2.0.3]

- Improvements
  - Changed to implement SEMA42_Lock base on SEMA42_TryLock.

[2.0.2]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 17.7.

[2.0.1]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 10.3, 10.4, 14.4, 18.1.

[2.0.0]

- Initial version.

SFA

[2.1.8]

- Bug Fixes
  - Fixed incorrect $(\text{void } *)\text{memset}()$ cast to $(\text{void})\text{memset}()$ to resolve unused-value compiler warning.

[2.1.7]

- New Features
  - Added common IRQ handler entry SFA_DriverIRQHandler.

[2.1.6]

- Bug Fixes
  - Updated timeout-related macro definitions and $\#\text{if}$ conditions to use explicit constant expressions for MISRA compliance.
  - Improved $\text{SFA_Mode0Calculate}()$ checks and rounding behavior for safer frequency calculation.

[2.1.5]

- Bug Fixes
  - Add assertions to prevent unsigned integer underflow in subtraction operations.
  - Add assertions to prevent unsigned integer overflow in multiplication calculations.
  - Add safe type conversion with temporary variables for uint32_t operations.
  - Add bounds checking for narrowing conversions from uint64_t to uint32_t.
  - Follow INT30-C and INT31-C CERT coding standards for safe integer operations.

[2.1.4]

- Bug Fixes
  - Fixed doxygen warnings: unbalanced grouping commands.

[2.1.3]

- Improvements
  - Add timeout for APIs with dfmea issues.

[2.1.2]

- Improvements
  - Updated SFA_ClearStatusFlag() function to support clearing all status flags.

[2.1.1]

- Improvements
  - Updated SFA driver code to reduce code redundancy.
  - Updated the use of macros in the driver code to enable the CUT pin function.

[2.1.0]

- Improvements
  - Updated how the clock frequency under test is calculated.
  - Supported configuration reference clock source.
  - Added clock count limit APIs.
  - Added macros to support fields that are configurable in different instances.

[2.0.1]

- Bug Fixes
  - Fixed the issue in SFA_Mode0Calculate function.
- Improvements
  - For the devices that support RF SFA, cast the peripheral structure to SFA_Type.

[2.0.0]

- Initial version.

SMSCM

[2.0.0]

- Initial version.

SPC

[2.9.0]

- New Features
  - Added VDD1P8 low-voltage detect APIs (config, Active/LP/HP enable, status flag) guarded by FSL_FEATURE_SPC_HAS_VDD1P8_LVD for SPC variants that monitor the VDD_FRO rail.
  - Added kSPC_PowerDomain3 and kSPC_INFRAPowerDomainRetain for SPC variants with a 4-domain topology.
  - Added VDD_GLITCH_DETECT_SC -> VDD_CORE_GLITCH_DETECT_SC alias block so both GLITCH_DETECT_SC and VDD_GLITCH_DETECT_SC register names resolve to a single unified set of driver macros.
- Improvements
  - Guarded DCDC_CFG FREQ_CNTRL access with FSL_FEATURE_SPC_HAS_NO_DCDC_FREQ_CNTRL for SPC variants that omit the frequency-control field.
  - Extended VD_STAT valid-flags mask to include VDD1P8 and composed it from per-domain sub-masks for cleaner compile-time selection.
- Bug Fixes
  - Fixed MISRA C-2012 Rule 14.3 violation in `SPC_SetHighPowerModeRegulatorsConfig`: moved `if (status == kStatus_Success)` guard for bandgap config inside the `FSL_FEATURE_SPC_HAS_SYS_LDO` conditional block so the condition is only present when `status` can actually change (i.e. when SYS LDO is configured).

[2.8.3]

- Bug Fixes
  - Fixed voltage detect status accumulation in Active, High Power, and Low Power mode helpers so VDD_SYS status bits are returned correctly when present.

[2.8.2]

- Bug Fixes
  - Mask voltage detect status bits to return only valid SPC VD status flags.
  - Mask external domain status bits before shifting to return only EVDSTAT field values.

[2.8.1]

- Bug Fixes
  - Add explicit boolean to integer conversions (? 1U : 0U) for enable flags in DCDC burst configuration.
  - Add explicit boolean to integer conversions (? 1U : 0U) for reset and interrupt enable flags in VDD core glitch detector.
  - Follow INT31-C CERT coding standard for safe boolean to unsigned long conversions.

[2.8.0]

- Improvements
  - Added feature macros to support some platforms which do not have system LDO, VDD_SYS, DCDC_BURST_CFG.
- Bug Fixes
  - Fixed doxygen warnings:
    1. unbalanced grouping commands.
    2. Argument 'base' of command @param is not found in the argument list of sfa_callback_t(status_t status)
    3. Explicit link request to 'kStatus_SPC_CORELDOVoltageSetFail.' could not be resolved

[2.7.1]

- Improvements
  - Make configuration structure pointers const in SPC driver API for Higher Power mode

[2.7.0]

- Improvements
  - Added new APIs to control VDD Core Glitch Detector.

[2.6.1]

- Improvements
  - Add timeout for while loop code.

[2.6.0]

- New Features
  - Added new feature FSL_FEATURE_SPC_HAS_SC_SPC_LP_REQ_BIT for compatibility with new platforms that do not support the SPC SC[SPC_LP_REQ] bitfield.

[2.5.0]

- New Features
  - Added new function group to support high power feature.

[2.4.1]

- Improvements
  - Added the new feature FSL_FEATURE_SPC_HAS_SC_REG_BUSY to support the device which has the SC[REG_BUSY] bitfield.

[2.4.0]

- Improvements
  - Set functions of VD_SYS_CFG[LVSEL] configuration as deprecated, since this bit filed is reserved for all devices.

[2.3.0]

- Improvements
  - Added SPC_SetSRAMOperateVoltage() to set SRAM operate voltage.
- Bug Fixes
  - Fixed the issue in SPC_SetDCDCBurstConfig() function.
  - Fixed a bug that is SPC has VDD_DS the voltage is not configured correctly.

[2.2.0]

- Bug Fixes
  - The default value of ACTIVE_CFG[CORELDO_VDD_DS] bit is 1, removed code of set that bit.
  - Fixed an issue of SPC_SetDCDCBurstConfig() to avoid program stuck in a while loop.
  - Removed some duplicate code that never reached.
- Improvements
  - Added two APIs to control Vdd core glitch detect feature in Active/lowpower modes.

[2.1.0]

- Improvements
  - Improved the spc_dcdc_voltage_level_t enumeration.
  - Added some low power request check APIs.
  - Added SPC_SoftwareGatePowerSwitch() and SPC_PowerModeControlPowerSwitch() function.
  - If APIs' parameters is in type of structure pointer, set it to const structure pointer.
- Bug Fixes
  - Fixed the issue in SPC_SetDCDCRefreshCount() function.
  - Fixed some issues in voltage detect enable APIs.
  - Fixed some issues in Bandgap control APIs.
  - Fixed some issues in regulator active/lowpower mode configuration APIs.

[2.0.2]

- Bug Fixes
  - Fixed the doxygen issue by adding missed comments.

[2.0.1]

- Bug Fixes
  - The enumerator spc_core_ldo_voltage_level_t may different for different devices, added the macro to separate different scenes.

[2.0.0]

- Initial version.

SYSPM

[2.3.1]

- Improvements
  - Add timeout for while loop.

[2.3.0]

- New Features
  - Supported instruction counter.

[2.2.0]

- New Features
  - Supported platforms which only have one monitor.
  - Supported platforms whose instrction counter can't be cleared.
  - Supported platforms whose counter can't be disabled.

[2.1.0]

- New Features
  - Added new APIs SYSPM_Init and SYSPM_Deinit.

[2.0.0]

- Initial version.

TPM

[2.5.0]

- Improvements
  - Added TPM_GetChannelControlBits API to replace the misspelled TPM_GetChannelContorlBits, which is now marked as deprecated.

[2.4.6]

- Improvements
  - Added feature macro FSL_FEATURE_TPM_HAS_NO_SC_CPWMS to support SoCs whose TPM SC register does not have the CPWMS field.

[2.4.5]

- Improvements
  - Added branch coverage justification comments and GCOVR_EXCL_BR_LINE markers for structurally unreachable branches in TPM_UpdatePwmDutycycle.

[2.4.4]

- Bug Fixes
  - Fixed CERT INT31-C: removed unsafe (uint8_t) cast on FSL_FEATURE_TPM_HAS_32BIT_COUNTERn() (which returns int with possible value -1) in TPM_MAX_COUNTER_VALUE macro and TPM_SetTimerPeriod; now uses direct int comparison to avoid truncation.

[2.4.3]

- Bug Fixes
  - Fixed CERT INT31-C Coverity issue: Casting from int to unsigned char without checking its value may result in lost or misinterpreted data.

[2.4.2]

- Bug Fixes
  - Fixed CERT INT30-C INT31-C issue for:
    - Unsigned integer operation may wrap.
    - Casting from “int” to “unsigned char” may result in lost or misinterpreted data.

[2.4.1]

- Improvements
  - Add Coverage Justification for uncovered code.

[2.4.0]

- New Feature
  - Added while loop timeout for MOD CnV CnSC and SC register write sequence.
  - Change the return type from void to status_t for following API:
    - TPM_DisableChannel
    - TPM_EnableChannel
    - TPM_SetupOutputCompare
    - TPM_SetTimerPeriod
    - TPM_StopTimer

[2.3.6]

- Bug Fixes
  - Fixed CERT INT30-C INT31-C issue for TPM_SetupDualEdgeCapture.

[2.3.5]

- New Feature
  - Added IRQ handler entry for TPM2.

[2.3.4]

- New Feature
  - Added common IRQ handler entry TPM_DriverIRQHandler.

[2.3.3]

- Improvements
  - Conditionally compile interrupt handling code to solve the problem of using this driver on CPU cores that do not support interrupts.

[2.3.2]

- Bug Fixes
  - Fixed ERR008085 TPM writing the TPMx_MOD or TPMx_CnV registers more than once may fail when the timer is disabled.

[2.3.1]

- Bug Fixes
  - Fixed compilation error when macro FSL_SDK_DISABLE_DRIVER_CLOCK_CONTROL is 1.

[2.3.0]

- Improvements
  - Create callback feature for TPM match and timer overflow interrupts.

[2.2.4]

- Improvements
  - Add feature macros(FSL_FEATURE_TPM_HAS_GLOBAL_TIME_BASE_EN, FSL_FEATURE_TPM_HAS_GLOBAL_TIME_BASE_SYNC).

[2.2.3]

- Improvements
  - Release peripheral from reset if necessary in init function.

[2.2.2]

- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 10.4.

[2.2.1]

- Bug Fixes
  - Fixed CCM issue by splitting function from TPM_SetupPwm() function to reduce function complexity.
  - Fixed violations of MISRA C-2012 rule 17.7.

[2.2.0]

- Improvements
  - Added TPM_SetChannelPolarity to support select channel input/output polarity.
  - Added TPM_EnableChannelExtTrigger to support enable external trigger input to be used by channel.
  - Added TPM_CalculateCounterClkDiv to help calculates the counter clock prescaler.
  - Added TPM_GetChannelValue to support get TPM channel value.
  - Added new TPM configuration.
    * syncGlobalTimeBase
    * extTriggerPolarity
    * chnlPolarity
  - Added new PWM signal configuration.
    * secPauseLevel
- Bug Fixes
  - Fixed TPM_SetupPwm can't configure 0% combined PWM issues.

[2.1.1]

- Improvements
  - Add feature macro for PWM pause level select feature.

[2.1.0]

- Improvements
  - Added TPM_EnableChannel and TPM_DisableChannel APIs.
  - Added new PWM signal configuration.
    * pauseLevel - Support select output level when counter first enabled or paused.
    * enableComplementary - Support enable/disable generate complementary PWM signal.
    * deadTimeValue - Support deadtime insertion for each pair of channels in combined PWM mode.
- Bug Fixes

  - Fixed issues about channel MSnB:MSnA and ELSnB:ELSnA bit fields and CnV register change request acknowledgement. Writes to these bits are ignored when the interval between successive writes is less than the TPM clock period.

[2.0.8]

- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 10.1, 10.4 ,10.7 and 14.4.

[2.0.7]

- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 10.4 and 17.7.

[2.0.6]

- Bug Fixes
  - Fixed Out-of-bounds issue.

[2.0.5]

- Bug Fixes
  - Fixed MISRA-2012 rules.
    - Rule 10.6, 10.7

[2.0.4]

- Bug Fixes
  - Fixed ERR050050 in functions TPM_SetupPwm/TPM_UpdatePwmDutycycle. When TPM was configured in EPWM mode as PS = 0, the compare event was missed on the first reload/overflow after writing 1 to the CnV register.

[2.0.3]

- Bug Fixes
  - MISRA-2012 issue fixed.
    - Fixed rules: rule-12.1, rule-17.7, rule-16.3, rule-14.4, rule-1.3, rule-10.4, rule-10.3, rule-10.7, rule-10.1, rule-10.6, and rule-18.1.

[2.0.2]

- Bug Fixes
  - Fixed issues in functions TPM_SetupPwm/TPM_UpdateChnlEdgeLevelSelect /TPM_SetupInputCapture/TPM_SetupOutputCompare/TPM_SetupDualEdgeCapture, wait acknowledgement when the channel is disabled.

[2.0.1]

- Bug Fixes
  - Fixed TPM_UpdateChnIEdgeLevelSelect ACK wait issue.
  - Fixed the issue that TPM_SetupdualEdgeCapture could not set FILTER register.
  - Fixed TPM_UpdateChnEdgeLevelSelect ACK wait issue.

[2.0.0]

- Initial version.

TRDC

[2.3.1]

- Improvements
  - Update APIs to check whether the memory access configuration can be updated.
  - Update APIs to mask the MRC address since only high 18 bits are valid.

[2.3.0]

- New Features
  - Added API TRDC_EnableProcessorDomainAssignment to disable/enable DAC.
- Bug Fixes
  - Fixed MISRA violation of missing function declaration.
  - Fixed wrong operation of domain mask in TRDC_MbcNseClearAll and TRDC_MrcDomainNseClear.

[2.2.1]

- Bug Fixes
  - Fixed MISRA violations of rule 10.3.

[2.2.0]

- New Features
  - Added new APIs to support SoC with more than one processor core.

[2.1.1]

- Bug Fixes
  - Fixed MISRA violations of rule 10.3, 10.6, 10.7 and 18.1.

[2.1.0]

- Improvements
  - Modified some of the addressing method according to the device header file's update.
  - Modified flash address configuration structure and default setting.
- New Features
  - Added API to set flash logic window array address.
- Bug Fixes
  - Fixed wrong return value of TRDC_GetFlashLogicalWindowPbase.
  - Fixed bug in TRDC_GetAndClearFirstSpecificDomainError that the error status is cleared by mistake before obtained by software.
  - Fixed MISRA violations of rule 10.3, 10.4, 10.6 and 10.8.

[2.0.0]

- Initial version.

TRGMUX

[2.0.1]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 10.1, 10.3, 10.8.

[2.0.0]

- Initial version.

VBAT

[2.1.1]

- Improvements
  - Added timeout for while loop.

[2.1.0]

- Improvements
  - Both FROCFGA and LDOCFGA registers are set as internal register, removed corresponding interfaces.

[2.0.0]

- Initial version.

VREF

[2.5.0]

- New features
  - Add VREF_SetAdcNegChannelRef() API for ADC negative channel control
  - Add VREF_SetAdcPosChannelRef() API for ADC positive channel control
  - Add conditional LPBGEN_MASK setting based on feature availability
  - Add IBIAS_EN_MASK setting for devices with IBIAS_EN feature

[2.4.1]

- Improvements
  - Improve doxygen documentation.

[2.4.0]

- Improvements
  - Support TEST_UNLOCK and TRIM0 registers read and write.

[2.3.0]

- Improvements
  - Add feature macros for compatibility with some platforms that don't have CSR[LPBG_BUF_EN] and UTRIM[TRIM2V1].

[2.2.2]

- Bug Fixes
  - Fixed typo in vref_buffer_mode_t enumerator.

[2.2.1]

- Improvements
  - Updated VREF driver code to reduce code redundancy.

[2.2.0]

- Improvements
  - Remove API VREF_SetTrimVal and VREF_GetTrimVal
  - Add new API VREF_SetVrefTrimVal,VREF_SetTrim21Val,VREF_GetVrefTrimVal and VREF_GetTrim21Val
  - Updated VREF driver code to conform to VREF IP architecture.

[2.1.0]

- Improvements
  - Remove API VREF_SetTrimVal and VREF_GetTrimVal
  - Add new API VREF_SetVrefTrimVal,VREF_SetTrim21Val,VREF_GetVrefTrimVal and VREF_GetTrim21Val
  - Updated VREF driver code to conform to VREF IP architecture.
- Bug Fixes
  - Fix issue which progeam stuck in VREF_Init

[2.0.0]

- Initial version.

WDOG32

[2.2.1]

- Bug Fixes
  - Fix CERT INT31-C that the bool value shall be converted to unsigned int 0 or 1 then passed to registers.
  - Fix MISRA 2012 20.3 vilation.

[2.2.0]

- Improvements
  - Added while loop timeout config value for WDOG32 reconfiguration and unlock sequence.
  - Change the return type of WDOG32_Init, WDOG32_Deinit and WDOG32_Unlock from void to status_t.

[2.1.0]

- Improvements
  - Release peripheral from reset if necessary in init function.

[2.0.4]

- Improvements
  - To ensure that the reconfiguration is inside 128 bus clocks unlock window, put all reconfiguration APIs in quick access code section.

[2.0.3]

- Bug Fixes
  - Fixed the noncompliance issue of the reference document.
    - Waited until for new configuration to take effect by checking the RCS bit field.
    - Waited until for registers to be unlocked by checking the ULK bit field.

- Improvements
  - Added 128 bus clocks delay ensures a smooth transition before restarting the counter with the new configuration when there is no RCS status bit.

[2.0.2]

- Bug Fixes
  - MISRA C-2012 issue fixed.
    - Fixed rules, containing: rule-10.3, rule-14.4, rule-15.5.
  - Fixed the issue of the inseparable process interrupted by other interrupt source.
    - WDOG32_Refresh

[2.0.1]

- Bug Fixes
  - WDOG must be configured within its configuration time period.
    - Added WDOG32_Init API to quick access section.
    - Defined register variable in WDOG32_Init API.

[2.0.0]

- Initial version.

WUU

[2.4.2]

- Bug Fixes
  - Fixed violations of CERT-C:
    - Check filterIndex >= 1U to prevent underflow.
    - Check multiplication results fit in uint8_t before assignment to shift variable.

[2.4.1]

- Improvements
  - The semantics of `kWUU_FilterActiveDSPD` and `kWUU_ExternalPinActiveDSPD` are not clear enough, because on some platforms it's not 'DSPD' (deep sleep, power down) but PDDPD (power down, deep power down). We deprecate them and will use the more generic `kWUU_FilterActiveLowLeakage` and `kWUU_ExternalPinActiveLowLeakage` in the future.

[2.4.0]

- New Features
  - Added WUU_ClearExternalWakeupPinsConfig() to clear settings of PDC and PE register.

[2.3.0]

- New Features
  - Added WUU_ClearInternalWakeUpModulesConfig() to clear settings of DM and ME register.

[2.2.1]

- Bug Fixes
  - Fixed WUU_SetPinFilterConfig() unable to set edge detection of pin filter config.
  - Fixed wrong macro used in WUU_GetPinFilterFlag() function.

[2.2.0]

- New Features
  - Added the WUU_GetExternalWakeupPinFlag() and WUU_ClearExternalWakeupPinFlag() function .

[2.1.0]

- New Features
  - Added the WUU_GetModuleInterruptFlag() function to support the devices that equipped MF register.

[2.0.0]

- Initial version.

1.7 Driver API Reference Manual

This section provides a link to the Driver API RM, detailing available drivers and their usage to help you integrate hardware efficiently.

MCXW727C

1.8 Middleware Documentation

Find links to detailed middleware documentation for key components. While not all onboard middleware is covered, this serves as a useful reference for configuration and development.

1.8.1 Wireless Bluetooth LE host stack and applications

NXP Bluetooth LE Host and Sample Applications

1.8.2 Wireless Connectivity Framework

framework

1.8.3 Multicore

Multicore SDK

1.8.4 FreeMASTER

freemaster

1.8.5 FreeRTOS

FreeRTOS

Chapter 2

MCXW727C

2.1 CACHE: LPCAC CACHE Memory Controller

static inline void L1CACHE_EnableCodeCache(void)

Enables the processor code bus cache.

static inline void L1CACHE_DisableCodeCache(void)

Disables the processor code bus cache.

static inline void L1CACHE_InvalidateCodeCache(void)

Invalidates the processor code bus cache.

void L1CACHE_InvalidateICacheByRange(uint32_t address, uint32_t size_byte)

Invalidates L1 instrument cache by range.

Parameters

- address – The start address of the memory to be invalidated.
- size_byte – The memory size.

static inline void L1CACHE_InvalidateDCacheByRange(uint32_t address, uint32_t size_byte)

Invalidates L1 data cache by range.

Parameters

- address – The start address of the memory to be invalidated.
- size_byte – The memory size.

static inline void L1CACHE_CleanDCacheByRange(uint32_t address, uint32_t size_byte)

Cleans L1 data cache by range.

The cache is write through mode, so there is nothing to do with the cache flush/clean operation.

Parameters

- address – The start address of the memory to be cleaned.
- size_byte – The memory size.

static inline void L1CACHE_CleanInvalidateDCacheByRange(uint32_t address, uint32_t size_byte)

Cleans and Invalidates L1 data cache by range.

Parameters

- address – The start address of the memory to be clean and invalidated.

- size_byte – The memory size.

static inline void ICACHE_InvalidateByRange(uint32_t address, uint32_t size_byte)

Invalidates instruction cache by range.

Parameters

- address – The physical address.
- size_byte – size of the memory to be invalidated.

static inline void DCACHE_InvalidateByRange(uint32_t address, uint32_t size_byte)

Invalidates data cache by range.

Parameters

- address – The physical address.
- size_byte – size of the memory to be invalidated.

static inline void DCACHE_CleanByRange(uint32_t address, uint32_t size_byte)

Clean data cache by range.

Parameters

- address – The physical address.
- size_byte – size of the memory to be cleaned.

static inline void DCACHE_CleanInvalidateByRange(uint32_t address, uint32_t size_byte)

Cleans and Invalidates data cache by range.

Parameters

- address – The physical address.
- size_byte – size of the memory to be Cleaned and Invalidated.

FSL_CACHE_DRIVER_VERSION

cache driver version 2.1.2.

2.2 CCM32K: 32kHz Clock Control Module

void CCM32K_Enable32kFro(CCM32K_Type *base, bool enable)

Enable/Disable 32kHz free-running oscillator.

Note: There is a start up time before clocks are output from the FRO.

Note: To enable FRO32k and set it as 32kHz clock source please follow steps:

```
CCM32K_Enable32kFro(base, true);        //Enable FRO analog oscillator.
CCM32K_DisableCLKOutToPeripherals(base, mask); //Disable clock out.
CCM32K_SelectClockSource(base, kCCM32K_ClockSourceSelectFro32k); //Select FRO32k as clock␣
↪source.
while(CCM32K_GetStatus(base) != kCCM32K_32kFroActiveStatusFlag); //Check FOR32k is active␣
↪and in used.
CCM32K_EnableCLKOutToPeripherals(base, mask); //Enable clock out if needed.
```

Parameters

- base – CCM32K peripheral base address.
- enable – Boolean value to enable or disable the 32kHz free-running oscillator. true — Enable 32kHz free-running oscillator. false — Disable 32kHz free-running oscillator.

static inline void CCM32K_Lock32kFroWriteAccess(CCM32K_Type *base)

Lock all further write accesses to the FRO32K_CTRL register until a POR occurs.

Parameters

- base – CCM32K peripheral base address.

static inline uint16_t CCM32K_Get32kFroTrimValue(CCM32K_Type *base)

Get frequency trim value of 32kHz free-running oscillator.

Parameters

- base – CCM32K peripheral base address.

Returns

The current trim value.

void CCM32K_Set32kFroTrimValue(CCM32K_Type *base, uint16_t trimValue)

Set the frequency trim value of 32kHz free-running oscillator by software.

Note: The frequency is decreased monotonically when the trimValue is changed progressively from 0x0U to 0x7FFU.

Note: If the FRO32 is enabled before invoking this function, then in this function the FRO32 will be disabled, after updating trim value the FRO32 will be re-enabled.

Parameters

- base – CCM32K peripheral base address.
- trimValue – The frequency trim value.

static inline void CCM32K_Disable32kFroIFRLoad(CCM32K_Type *base, bool disable)

Disable/Enable the function of setting 32kHz free-running oscillator trim value when IFR value gets loaded in the SOC.

Parameters

- base – CCM32K peripheral base address.
- disable – Boolean value to disable or enable IFR loading function. true — Disable IFR loading function. false — Enable IFR loading function.

static inline void CCM32K_Lock32kFroTrimWriteAccess(CCM32K_Type *base)

Lock all further write accesses to the FRO32K_TRIM register until a POR occurs.

Parameters

- base – CCM32K peripheral base address.

void CCM32K_Set32kOscConfig(CCM32K_Type *base, *ccm32k_osc_mode_t* mode, const *ccm32k_osc_config_t* *config)

Config 32k Crystal Oscillator.

Note: When the mode selected as *kCCM32K_Disable32kHzCrystalOsc* or *kCCM32K_Bypass32kHzCrystalOsc* the parameter config is useless, so it can be set as "NULL".

Note: To enable OSC32K and select it as clock source of 32kHz please follow steps:

```
CCM32K_Set32kOscConfig(base, kCCM32K_Enable32kHzCrystalOsc, config); //Enable OSC32k and␣
↪set config.
while((CCM32K_GetStatus(base) & kCCM32K_32kOscReadyStatusFlag) == 0UL); //Check if␣
↪OSC32K is stable.
CCM32K_DisableCLKOutToPeripherals(base, mask); //Disable clock out.
CCM32K_SelectClockSource(base, kCCM32K_ClockSourceSelectOsc32k); //Select OSC32k as clock␣
↪source.
while((CCM32K_GetStatus(base) & kCCM32K_32kOscActiveStatusFlag) == 0UL); //Check if␣
↪OSC32K is used as clock source.
CCM32K_EnableCLKOutToPeripherals(base, mask); //Enable clock out.
```

Parameters

- base – CCM32K peripheral base address.
- mode – The mode of 32k crystal oscillator.
- config – The pointer to the structure *ccm32k_osc_config_t*.

static inline void CCM32K_Lock32kOscWriteAccess(CCM32K_Type *base)

Lock all further write accesses to the OSC32K_CTRL register until a POR occurs.

Parameters

- base – CCM32K peripheral base address.

void CCM32K_EnableClockMonitor(CCM32K_Type *base, bool enable)

Enable/disable clock monitor.

Parameters

- base – CCM32K peripheral base address.
- enable – Used to enable/disable clock monitor.
  - **turn** Enable clock monitor.
  - **false** Disable clock monitor.

static inline void CCM32K_SetClockMonitorFreqTrimValue(CCM32K_Type *base, *ccm32k_clock_monitor_freq_trim_value_t* trimValue)

Set clock monitor frequency trim value.

Parameters

- base – CCM32K peripheral base address.
- trimValue – Clock minitor frequency trim value, please refer to *ccm32k_clock_monitor_freq_trim_value_t*.

static inline void CCM32K_SetClockMonitorDivideTrimValue(CCM32K_Type *base, *ccm32k_clock_monitor_divide_trim_value_t* trimValue)

Set clock monitor divide trim value.

Parameters

- base – CCM32K peripheral base address.
- trimValue – Clock minitor divide trim value, please refer to *ccm32k_clock_monitor_divide_trim_value_t*.

void CCM32K_SetClockMonitorConfig(CCM32K_Type *base, const *ccm32k_clock_monitor_config_t* *config)

Config clock monitor one time, including frequency trim value, divide trim value.

Parameters

- base – CCM32K peripheral base address.
- config – Pointer to *ccm32k_clock_monitor_config_t* structure.

static inline void CCM32K_LockClockMonitorWriteAccess(CCM32K_Type *base)

Lock all further write accesses to the CLKMON_CTRL register until a POR occurs.

Parameters

- base – CCM32K peripheral base address.

static inline void CCM32K_EnableCLKOutToPeripherals(CCM32K_Type *base, uint8_t peripheralMask)

Enable 32kHz clock output to selected peripherals.

Parameters

- base – CCM32K peripheral base address.
- peripheralMask – The mask of peripherals to enable 32kHz clock output, should be the OR'ed value of *ccm32k_clock_output_peripheral_t*.

static inline void CCM32K_DisableCLKOutToPeripherals(CCM32K_Type *base, uint8_t peripheralMask)

Disable 32kHz clock output to selected peripherals.

Parameters

- base – CCM32K peripheral base address.
- peripheralMask – The mask of peripherals to disable 32kHz clock output, should be the OR'ed value of *ccm32k_clock_output_peripheral_t*.

static inline void CCM32K_SelectClockSource(CCM32K_Type *base, *ccm32k_clock_source_select_t* clockSource)

Select CCM32K module's clock source which will be provide to the device.

Parameters

- base – CCM32K peripheral base address.
- clockSource – Used to select clock source, please refer to *ccm32k_clock_source_select_t* for details.

static inline void CCM32K_LockClockGateWriteAccess(CCM32K_Type *base)

Lock all further write access to the CGC32K register until a POR occurs.

Parameters

- base – CCM32K peripheral base address.

static inline uint32_t CCM32K_GetStatusFlag(CCM32K_Type *base)

Get the status flag.

Parameters

- base – CCM32K peripheral base address.

Returns
The status flag of the current node. The enumerator of status flags have been provided, please see the Enumerations title for details.

ccm32k_state_t CCM32K_GetCurrentState(CCM32K_Type *base)

Get current state.

Parameters

- base – CCM32K peripheral base address.

Returns
The CCM32K's current state, please refer to *ccm32k_state_t* for details.

ccm32k_clock_source_t CCM32K_GetClockSource(CCM32K_Type *base)

Return current clock source.

Parameters

- base – CCM32K peripheral base address.

Return values

- kCCM32K_ClockSourceNone – The none clock source is selected.
- kCCM32K_ClockSource32kFro – 32kHz free-running oscillator is selected as clock source.
- kCCM32K_ClockSource32kOsc – 32kHz crystal oscillator is selected as clock source..

FSL_CCM32K_DRIVER_VERSION

CCM32K driver version 2.2.2.

enum _ccm32k_osc_xtal_cap

The enumerator of internal capacitance of OSC's XTAL pin.

Values:

enumerator kCCM32K_OscXtal0pFCap

The internal capacitance for XTAL pin is 0pF.

enumerator kCCM32K_OscXtal2pFCap

The internal capacitance for XTAL pin is 2pF.

enumerator kCCM32K_OscXtal4pFCap

The internal capacitance for XTAL pin is 4pF.

enumerator kCCM32K_OscXtal6pFCap

The internal capacitance for XTAL pin is 6pF.

enumerator kCCM32K_OscXtal8pFCap

The internal capacitance for XTAL pin is 8pF.

enumerator kCCM32K_OscXtal10pFCap

The internal capacitance for XTAL pin is 10pF.

enumerator kCCM32K_OscXtal12pFCap

The internal capacitance for XTAL pin is 12pF.

enumerator kCCM32K_OscXtal14pFCap
The internal capacitance for XTAL pin is 14pF.

enumerator kCCM32K_OscXtal16pFCap
The internal capacitance for XTAL pin is 16pF.

enumerator kCCM32K_OscXtal18pFCap
The internal capacitance for XTAL pin is 18pF.

enumerator kCCM32K_OscXtal20pFCap
The internal capacitance for XTAL pin is 20pF.

enumerator kCCM32K_OscXtal22pFCap
The internal capacitance for XTAL pin is 22pF.

enumerator kCCM32K_OscXtal24pFCap
The internal capacitance for XTAL pin is 24pF.

enumerator kCCM32K_OscXtal26pFCap
The internal capacitance for XTAL pin is 26pF.

enumerator kCCM32K_OscXtal28pFCap
The internal capacitance for XTAL pin is 28pF.

enumerator kCCM32K_OscXtal30pFCap
The internal capacitance for XTAL pin is 30pF.

enum _ccm32k_osc_extal_cap
The enumerator of internal capacitance of OSC's EXTAL pin.

Values:

enumerator kCCM32K_OscExtal0pFCap
The internal capacitance for EXTAL pin is 0pF.

enumerator kCCM32K_OscExtal2pFCap
The internal capacitance for EXTAL pin is 2pF.

enumerator kCCM32K_OscExtal4pFCap
The internal capacitance for EXTAL pin is 4pF.

enumerator kCCM32K_OscExtal6pFCap
The internal capacitance for EXTAL pin is 6pF.

enumerator kCCM32K_OscExtal8pFCap
The internal capacitance for EXTAL pin is 8pF.

enumerator kCCM32K_OscExtal10pFCap
The internal capacitance for EXTAL pin is 10pF.

enumerator kCCM32K_OscExtal12pFCap
The internal capacitance for EXTAL pin is 12pF.

enumerator kCCM32K_OscExtal14pFCap
The internal capacitance for EXTAL pin is 14pF.

enumerator kCCM32K_OscExtal16pFCap
The internal capacitance for EXTAL pin is 16pF.

enumerator kCCM32K_OscExtal18pFCap
The internal capacitance for EXTAL pin is 18pF.

enumerator kCCM32K_OscExtal20pFCap
The internal capacitance for EXTAL pin is 20pF.

enumerator kCCM32K_OscExtal22pFCap
The internal capacitance for EXTAL pin is 22pF.

enumerator kCCM32K_OscExtal24pFCap
The internal capacitance for EXTAL pin is 24pF.

enumerator kCCM32K_OscExtal26pFCap
The internal capacitance for EXTAL pin is 26pF.

enumerator kCCM32K_OscExtal28pFCap
The internal capacitance for EXTAL pin is 28pF.

enumerator kCCM32K_OscExtal30pFCap
The internal capacitance for EXTAL pin is 30pF.

enum __ccm32k_osc_fine_adjustment_value
The enumerator of osc amplifier gain fine adjustment. Changes the oscillator amplitude by modifying the automatic gain control (AGC).

Values:

enumerator kCCM32K_OscFineAdjustmentRange0

enum __ccm32k_osc_coarse_adjustment_value
The enumerator of osc amplifier coarse fine adjustment. Tunes the internal transconductance (gm) by increasing the current.

Values:

enumerator kCCM32K_OscCoarseAdjustmentRange0

enumerator kCCM32K_OscCoarseAdjustmentRange1

enumerator kCCM32K_OscCoarseAdjustmentRange2

enumerator kCCM32K_OscCoarseAdjustmentRange3

enum _ccm32k_osc_mode

The enumerator of 32kHz oscillator.

Values:

enumerator kCCM32K_Disable32kHzCrystalOsc

Disable 32kHz Crystal Oscillator.

enumerator kCCM32K_Enable32kHzCrystalOsc

Enable 32kHz Crystal Oscillator.

enumerator kCCM32K_Bypass32kHzCrystalOsc

Bypass 32kHz Crystal Oscillator, use the 32kHz Oscillator or external 32kHz clock.

The enumerator of CCM32K status flag.

Values:

enumerator kCCM32K_32kOscReadyStatusFlag

Indicates the 32kHz crystal oscillator is stable.

enumerator kCCM32K_32kOscActiveStatusFlag

Indicates the 32kHz crystal oscillator is active and in use.

enumerator kCCM32K_32kFroActiveStatusFlag

Indicates the 32kHz free running oscillator is active and in use.

enumerator kCCM32K_ClockDetectStatusFlag

Indicates the clock monitor has detected an error.

enum _ccm32k_state

The enumerator of module state.

Values:

enumerator kCCM32K_Both32kFro32kOscDisabled

Indicates both 32kHz free running oscillator and 32kHz crystal oscillator are disabled.

enumerator kCCM32K_Only32kFroEnabled

Indicates only 32kHz free running oscillator is enabled.

enumerator kCCM32K_Only32kOscEnabled

Indicates only 32kHz crystal oscillator is enabled.

enumerator kCCM32K_Both32kFro32kOscEnabled

Indicates both 32kHz free running oscillator and 32kHz crystal oscillator are enabled.

enum _ccm32k_clock_source

The enumerator of clock source.

Values:

enumerator kCCM32K_ClockSourceNone

None clock source.

enumerator kCCM32K_ClockSource32kFro

32kHz free running oscillator is the clock source.

enumerator kCCM32K_ClockSource32kOsc

32kHz crystal oscillator is the clock source.

enum _ccm32k_clock_monitor_freq_trim_value

Clock monitor frequency trim values.

Values:

enumerator kCCM32K_ClockMonitor2CycleAssert

Clock monitor asserts 2 cycle after expected edge (assert after 10 cycles with no edge).

enumerator kCCM32K_ClockMonitor4CycleAssert

Clock monitor asserts 4 cycle after expected edge (assert after 12 cycles with no edge).

enumerator kCCM32K_ClockMonitor6CycleAssert

Clock monitor asserts 6 cycle after expected edge (assert after 14 cycles with no edge).

enumerator kCCM32K_ClockMonitor8CycleAssert

Clock monitor asserts 8 cycle after expected edge (assert after 16 cycles with no edge).

enum _ccm32k_clock_monitor_divide_trim_value

Clock monitor divide trim values.

Values:

enumerator kCCM32K_ClockMonitor_1kHzFro32k_1kHzOsc32k

Clock monitor operates at 1 kHz for both FRO32K and OSC32K.

enumerator kCCM32K_ClockMonitor_64HzFro32k_1kHzOsc32k

Clock monitor operates at 64 Hz for FRO32K and clock monitor operates at 1 kHz for OSC32K.

enumerator kCCM32K_ClockMonitor_1KHzFro32k_64HzOsc32k

Clock monitor operates at 1K Hz for FRO32K and clock monitor operates at 64 Hz for OSC32K.

enumerator kCCM32K_ClockMonitor_64HzFro32k_64HzOsc32k

Clock monitor operates at 64 Hz for FRO32K and clock monitor operates at 64 Hz for OSC32K.

enum _ccm32k_clock_source_select

CCM32K clock source enumeration.

Values:

enumerator kCCM32K_ClockSourceSelectFro32k

FRO32K clock output is selected as clock source.

enumerator kCCM32K_ClockSourceSelectOsc32k

OSC32K clock output is selected as clock source.

enum _ccm32k_clock_output_peripheral

32kHz clock output peripheral bit map.

Values:

enumerator kCCM32K_ClockOutToRtc

32kHz clock output to RTC.

enumerator kCCM32K_ClockOutToRfmc

32kHz clock output to Rfmc.

enumerator kCCM32K_ClockOutToNbu

32kHz clock output to NBU.

enumerator kCCM32K_ClockOutToWuuRmcPortD

32kHz clock output to WUU/RMC/PORTD.

enumerator kCCM32K_ClockOutToOtherModules

32kHz clock output to Other modules.

typedef enum *_ccm32k_osc_xtal_cap* ccm32k_osc_xtal_cap_t

The enumerator of internal capacitance of OSC's XTAL pin.

typedef enum *_ccm32k_osc_extal_cap* ccm32k_osc_extal_cap_t

The enumerator of internal capacitance of OSC's EXTAL pin.

typedef enum *_ccm32k_osc_fine_adjustment_value* ccm32k_osc_fine_adjustment_value_t

The enumerator of osc amplifier gain fine adjustment. Changes the oscillator amplitude by modifying the automatic gain control (AGC).

typedef enum *_ccm32k_osc_coarse_adjustment_value* ccm32k_osc_coarse_adjustment_value_t

The enumerator of osc amplifier coarse fine adjustment. Tunes the internal transconductance (gm) by increasing the current.

typedef enum *_ccm32k_osc_mode* ccm32k_osc_mode_t

The enumerator of 32kHz oscillator.

typedef enum *_ccm32k_state* ccm32k_state_t

The enumerator of module state.

typedef enum *_ccm32k_clock_source* ccm32k_clock_source_t

The enumerator of clock source.

typedef enum *_ccm32k_clock_monitor_freq_trim_value* ccm32k_clock_monitor_freq_trim_value_t

Clock monitor frequency trim values.

typedef enum *_ccm32k_clock_monitor_divide_trim_value* ccm32k_clock_monitor_divide_trim_value_t

Clock monitor divide trim values.

typedef struct *_ccm32k_clock_monitor_config* ccm32k_clock_monitor_config_t

Clock monitor configuration structure.

typedef enum *_ccm32k_clock_source_select* ccm32k_clock_source_select_t

CCM32K clock source enumeration.

typedef enum *_ccm32k_clock_output_peripheral* ccm32k_clock_output_peripheral_t

32kHz clock output peripheral bit map.

typedef struct *_ccm32k_osc_config* ccm32k_osc_config_t

The structure of oscillator configuration.

CCM32K_OSC32K_CTRL_OSC_MODE_MASK

CCM32K_OSC32K_CTRL_OSC_MODE_SHIFT

CCM32K_OSC32K_CTRL_OSC_MODE(x)

struct __ccm32k_clock_monitor_config

#include <fsl_ccm32k.h> Clock monitor configuration structure.

Public Members

bool enableClockMonitor

Used to enable/disable clock monitor.

ccm32k_clock_monitor_freq_trim_value_t freqTrimValue

Clock minitor frequency trim value.

ccm32k_clock_monitor_divide_trim_value_t divideTrimValue

Clock minitor divide trim value.

struct __ccm32k_osc_config

#include <fsl_ccm32k.h> The structure of oscillator configuration.

Public Members

bool enableInternalCapBank

enable/disable the internal capacitance bank.

ccm32k_osc_xtal_cap_t xtalCap

The internal capacitance for the OSC XTAL pin from the capacitor bank, only useful when the internal capacitance bank is enabled.

ccm32k_osc_extal_cap_t extalCap

The internal capacitance for the OSC EXTAL pin from the capacitor bank, only useful when the internal capacitance bank is enabled.

ccm32k_osc_fine_adjustment_value_t fineAdjustment

32kHz crystal oscillator amplifier fine adjustment value.

ccm32k_osc_coarse_adjustment_value_t coarseAdjustment

32kHz crystal oscillator amplifier coarse adjustment value.

2.3 CE Basic Functions

int32_t CE_ExecCmd(void)

Executes commands in the command queue.

Returns
Command execution status.

int32_t CE_NullCmd(void)

Sends a basic command to verify that ZV2117 is operational and that the command interface is functioning properly.

This function passes through the command processing interface and returns a success status.

Returns
Command execution status.

int32_t CE_Copy(int32_t *pDst, int32_t *pSrc, const int32_t N)

Copies data between buffers in the ZV2117 memory section.

Copies a specified number of 32-bit words from the source to the destination buffer. Both buffers must reside in the ZV2117 data memory section.

Parameters

- pDst – **[out]** Pointer to destination buffer
- pSrc – **[in]** Pointer to source buffer
- N – **[in]** Number of 32-bit words to copy

Returns
Command execution status.

2.4 CE Command Functions

int32_t CE_CmdInitBuffer(ce_cmdbuffer_t *psCmdBuffer, volatile uint32_t cmdbuffer[], volatile int32_t statusbuffer[], ce_cmd_mode_t cmdmode)

Initializes the CM33-ZV2117 command buffer.

This function must to be called once after power-up or reset, or when the command queue mode needs to be changed. Once configured, the command mode remains unchanged till reset or re-initialization.

Parameters

- psCmdBuffer – **[out]** Pointer to the command buffer structure. Must be allocated in ARM memory (not ZV2117).
- cmdbuffer – **[in]** Command buffer. Must be 256 words in ZV2117 data memory.
- statusbuffer – **[in]** Status buffer. Must be 134 words in ZV2117 data memory.
- cmdmode – **[in]** Command mode. One of:
  - kCE_CmdModeOneNonBlocking
  - kCE_CmdModeMultipleNonBlocking
  - kCE_CmdModeOneBlocking
  - kCE_CmdModeMultipleBlocking

Return values

0 – Initialization is successful.

int32_t CE_CmdReset(void)

Resets the CM33-ZV2117 command queue.

Any pending commands in the queue will be flushed.

Return values

0 – Reset is successful.

int32_t CE_CmdAdd(ce_cmd_t cmd, ce_cmdstruct_t *cmdargs)

Adds a command to the command queue.

Parameters

- cmd – **[in]** Command name. Choose from the enum description in fsl_ce_if.h. Not all of the cmd are implemented in the current release.
- cmdargs – **[in]** Arguments structure detailing the arguments for the function/command.

Return values

- 0 – Command added successfully.
- -1 – Command not added (queue is full).

int32_t CE_CmdLaunch(int32_t force_launch)

Launches the ZV2117 with the current command queue.

Parameters

- force_launch – **[in]**
  - 1: Launches the queue regardless of the command mode.
  - 0: Launches only if in single-command mode. Otherwise, does nothing.

Return values
0 – Launch is successful.

int32_t CE_CmdLaunchBlocking(void)

Launches the current command queue and waits for completion.

Return values
0 – Launch is successful.

int32_t CE_CmdLaunchNonBlocking(void)

Launches the current command queue and returns immediately.

ZV2117 will send an interrupt via MUA->GCR to ARM upon task completion. Alternatively, the user can poll for completion.

If using interrupt, the user must call *CE_CmdReset()* in the IRQ handler. IRQ::DSP_IRQn must be enabled.

The user can optionally also poll to figure out the command queue execution status.

Return values
0 – Launch is successful.

int32_t CE_CmdCheckStatus(void)

Checks the execution status of the current command queue. Only applicable in non-blocking mode.

Returns

- CE_STATUS_BUSY ZV2117 is still executing.
- CE_STATUS_IDLE Execution completed; ZV2117 is ready for new commands.

CE_COMPUTE_TIMEOUT

Maximum loop wait time for CE computation.

When CE is computing, driver will wait for the computation to complete. This parameter defines how many loops to check completion before return timeout. If defined as 0, driver will wait forever until completion.

2.5 CE CMSIS Functions

void ce_arm_cfft_f32(const *arm_cfft_instance_f32* *S, float *p1, uint8_t ifftFlag, uint8_t bitReverseFlag, float *pOut, float *pScratch)

CMSIS-compatible FFT implementation using ZV2117 backend.

This API is designed to be compatible with the ARM CM33 FFT interface. It delegates the FFT computation to the ZV2117 CE module. Please refer to CM33 documentation for details.

Note: This API only support float32 FFTs.

struct arm_cfft_instance_f32

#include <fsl_ce_cmsis.h> FFT/IFFT float32.

Public Members

uint16_t fftLen
Length of the FFT.

const float *pTwiddle
Points to the twiddle factor table.

const uint16_t *pBitRevTable
Points to the bit reversal table.

uint16_t bitRevLength
Length of the bit reversal table.

2.6 CE Initialization Functions

FSL_CE_DRIVER_VERSION
CE driver version.

typedef struct *_ce_copy_image* ce_copy_image_t
Structure for CE copy image to destination address.

Defines source and destination address for copying image with given size.

void CE_Init(*ce_copy_image_t* *ceCopyImage)
Initializes the CE.

Parameters

- ceCopyImage – **[in]** The information about the CE image to copy.

void CE_InstallFirmware(*ce_copy_image_t* *ceCopyImage)
Installs CE firmware by given image info.

Parameters

- ceCopyImage – **[in]** The information about the CE image to copy.

void CE_InitWithoutFirmware(void)
Initializes the CE.

This function is similar to CE_Init, but it does not install the firmware, the firmware can be installed using CE_InstallFirmware.

struct _ce_copy_image
#include <fsl_ce.h> Structure for CE copy image to destination address.

Defines source and destination address for copying image with given size.

2.7 CE Matrix Functions

int32_t CE_MatrixAdd_Q15(int16_t *pDst, int16_t *pA, int16_t *pB, int32_t M, int32_t N)
Calculates the sum of two real 16-bit integer (Q15) matrices.

Computes C = A + B, where A, B, and C are an M × N real int16_t matrices. All matrices are assumed to have the same format and dimensions.

Parameters

- pDst – **[out]** Pointer to the output matrix C (size M × N).
- pA – **[in]** Pointer to the input matrix A (size M × N).
- pB – **[in]** Pointer to the input matrix B (size M × N).
- M – **[in]** Number of rows of matrices A, B, and C.
- N – **[in]** Number of columns of matrices A, B, and C.

Returns
Command execution status.

int32_t CE_MatrixAdd_Q31(int32_t *pDst, int32_t *pA, int32_t *pB, int32_t M, int32_t N)

Calculates the sum of two real 32-bit integer (Q31) matrices.

Computes C = A + B, where A, B, and C are an M × N real int32_t matrices. All matrices are assumed to have the same format and dimensions.

Parameters

- pDst – **[out]** Pointer to the output matrix C (size M × N).
- pA – **[in]** Pointer to the input matrix A (size M × N).
- pB – **[in]** Pointer to the input matrix B (size M × N).
- M – **[in]** Number of rows of matrices A, B, and C.
- N – **[in]** Number of columns of matrices A, B, and C.

Returns
Command execution status.

int32_t CE_MatrixAdd_F32(float *pDst, float *pA, float *pB, int32_t M, int32_t N)

Calculates the sum of two real 32-bit floating-point matrices.

Computes C = A + B, where A, B, and C are an M × N real float32 matrices. All matrices are assumed to have the same format and dimensions.

Parameters

- pDst – **[out]** Pointer to the output matrix C (size M × N).
- pA – **[in]** Pointer to the input matrix A (size M × N).
- pB – **[in]** Pointer to the input matrix B (size M × N).
- M – **[in]** Number of rows of matrices A, B, and C.
- N – **[in]** Number of columns of matrices A, B, and C.

Returns
Command execution status.

int32_t CE_MatrixElemMul_F32(float *pDst, float *pA, float *pB, int32_t M, int32_t N)

Calculates the element-wise product of two real 32-bit floating-point matrices.

Computes C = A .× B, where A, B, and C are an M × N real float32 matrices. All matrices are assumed to have the same format and dimensions.

Parameters

- pDst – **[out]** Pointer to the output matrix C (size M × N).
- pA – **[in]** Pointer to the input matrix A (size M × N).
- pB – **[in]** Pointer to the input matrix B (size M × N).
- M – **[in]** Number of rows of matrices A, B, and C.

- N – **[in]** Number of columns of matrices A, B, and C.

Returns
Command execution status.

int32_t CE_MatrixMul_F32(float *pDst, float *pA, float *pB, int32_t M, int32_t N, int32_t P)

Calculates the matrix product of two real 32-bit floating-point matrices.

Computes C = A × B, where:

- A is an M × N real float32 matrix,
- B is an N × P real float32 matrix,
- C is the resulting M × P real float32 matrix.

All matrices are assumed to be in row-major format.

Note: Data precision and format is as defined by the argument type. Limits on max value of N:

- For F32: N < 128;
- For CF32: N < 64.

Parameters

- pDst – **[out]** Pointer to the output matrix C (size M × P)
- pA – **[in]** Pointer to the input matrix A (size M × N).
- pB – **[in]** Pointer to the input matrix B (size N × P).
- M – **[in]** Number of rows of matrix A.
- N – **[in]** Number of columns in matrix A (and rows in matrix B).
- P – **[in]** Number of columns of matrix B.

Returns
Command execution status.

int32_t CE_MatrixMul_CF32(float *pDst, float *pA, float *pB, int32_t M, int32_t N, int32_t P)

Calculates the matrix product of two complex 32-bit floating point matrices.

Computes C = A × B, where:

- A is an M × N complex float32 matrix,
- B is an N × P complex float32 matrix,
- C is the resulting M × P complex float32 matrix.

All matrices are assumed to be in row-major format.

Parameters

- pDst – **[out]** Pointer to the output matrix C (size M × P).
- pA – **[in]** Pointer to the input matrix A (size M × N).
- pB – **[in]** Pointer to the input matrix B (size N × P).
- M – **[in]** Number of rows of matrix A.
- N – **[in]** Number of columns in matrix A (and rows in matrix B).
- P – **[in]** Number of columns of matrix B.

Returns
Command execution status.

int32_t CE_MatrixInvHerm_CF32(float *pAinv, float *pA, float *pScratch, int32_t M, uint8_t flag_packedInput, uint8_t flag_cholInv)

Calculates the inverse or Cholesky inverse of a complex Hermitian matrix (in float32 precision).

Computes one of the following:

- Ainv = inv(A)
- Ainv = inv(chol(A))

where chol(A) is the lower triangular Cholesky decomposition of matrix A.

The input matrix must be in row-major format and can be either:

- A full M × M matrix, or
- A packed upper triangular matrix (containing only the upper triangle).

The output matrix is written in row-major format, and only the lower triangle is stored. The total number of output elements is:

- Mc = (M + 1) × M / 2

Note: The input, output and scratch buffers must be separately allocated and non-overlapping.

Parameters

- pAinv – **[out]** Pointer to the output matrix inverse. Mc elements are written (complex float32).
- pA – **[in]** Pointer to the input matrix A (complex float32, size M × M).
- pScratch – **[in]** Pointer to scratch buffer (minimum size Mc × 3 × 8 bytes).
- M – **[in]** Number of rows/columns in matrix A.
- flag_packedInput – **[in]** Format of input matrix:
  - 0: full matrix
  - 1: packed upper triangular matrix
- flag_cholInv – **[in]** Type of inverse to compute:
  - 0: inv(A)
  - 1: inv(chol(A))

Returns

Command execution status.

int32_t CE_MatrixEvdHerm_CF32(float *pLambdaOut, float *pUout, float *pUin, float *pScratch, int32_t M, float tol, int32_t max_iter, uint8_t flag_packedInput)

Calculates the Eigenvalue Decomposition (EVD) of a complex Hermitian matrix (float32 precision).

Performs EVD on an M×M complex Hermitian matrix A:

- [U, T] = eig(A)

The input matrix must be in column-major format and can be either:

- A full M × M matrix, or
- A packed upper triangular matrix.

The decomposition uses an iterative QR algorithm to compute:

- U: matrix of eigenvectors (M × M, column-major)
- T: vector of eigenvalues (M × 1)

Maximum supported matrix size: M 40.

Note: If input and output buffers overlap, the output U matrix will overwrite the input matrix A.

Parameters

- pLambdaOut – **[out]** Pointer to the output vector of eigenvalues (size M × 1, complex float32).
- pUout – **[out]** Pointer to the output matrix of eigenvectors (size M × M, column-major, complex float32).
- pUin – **[in]** Pointer to the input matrix A (size M × M, column-major, complex float32).
- pScratch – **[in]** Pointer to scratch buffer (minimum size: (M × M × 4 + 360) × 4 bytes).
- M – **[in]** Number of rows/columns in matrix A.
- tol – **[in]** Tolerance value for QR convergence. Smaller values yield better accuracy but require more iterations.
- max_iter – **[in]** Maximum number of QR iterations. If exceeded, the function terminates regardless of convergence.
- flag_packedInput – **[in]** Format of input matrix:
  - 0: full matrix
  - 1: packed upper triangular part only

Returns

Command execution status. The number of QR iterations is returned in status[3] register.

int32_t CE_MatrixChol_CF32(float *pL, float *pA, float *pScratch, int32_t M, uint8_t flag_packedInput)

Calculates the Cholesky Decomposition of a complex Hermitian matrix (float32 precision).

Calculates L = chol(A), where A is a complex Hermitian M × M matrix and L is a lower triangular matrix such that A = L × L ^ H.

The input matrix must be in row-major format and can be either:

- A full M × M matrix, or
- A packed format containing only the upper triangular elements.

The output matrix is also in row-major format, and only the lower triangular part is written. The total number of output elements is Mc, where:

- Mc = (M + 1) × M / 2

Note: The input, output and scratch buffers must be separately allocated and non-overlapping.

Parameters

- pL – **[out]** Pointer to the output lower triangular matrix L. Only Mc elements are written (complex-valued float32 data in row-major format).
- pA – **[in]** Pointer to the input matrix A (full or packed format). Must be complex-valued float32 data in row-major format.
- pScratch – **[in]** Pointer to a scratch buffer (at least Mc × 8 bytes).
- M – **[in]** Number of rows/columns of matrix A.
- flag_packedInput – **[in]** Format flag for input matrix:
  - 0: full matrix
  - 1: packed upper triangular matrix

Returns
Command execution status.

2.8 CE Transform Functions

int32_t CE_TransformCFFT_F16(float *pY, float *pX, float *pScratch, int32_t log2N)

Computes the FFT of complex-valued 16-bit floating point data.

Performs an N-point Fast Fourier Transform (FFT) on a complex-valued float16 input stream. The FFT size N must be a power of 2, with valid range: 16 □ N □ 16384.

Note:

- Data precision and format follow the argument types.
- float* is used to represent float16 pointers.
- The input, output and scratch buffers must be separately allocated and non-overlapping.

Parameters

- pY – **[out]** Pointer to the FFT output buffer (complex-valued float16 data).
- pX – **[in]** Pointer to the FFT input buffer (complex-valued float16 data).
- pScratch – **[in]** Pointer to a scratch buffer (at least N × 4 bytes).
- log2N – **[in]** Base-2 logarithm of the FFT size N.

Returns
Command execution status.

int32_t CE_TransformCFFT_F32(float *pY, float *pX, float *pScratch, int32_t log2N)

Computes the FFT of complex-valued 32-bit floating point data.

Performs an N-point Fast Fourier Transform (FFT) on a complex-valued float32 input stream. The FFT size N must be a power of 2, with valid range: 32 □ N □ 16384.

Note: The input, output and scratch buffers must be separately allocated and non-overlapping.

Parameters

- pY – **[out]** Pointer to the FFT output buffer (complex-valued float32).

- pX – **[in]** Pointer to the FFT input buffer (complex-valued float32).
- pScratch – **[in]** Pointer to a scratch buffer (at least N × 8 bytes).
- log2N – **[in]** Base-2 logarithm of the FFT size N.

Returns
Command execution status.

int32_t CE_TransformIFFT_F16(float *pY, float *pX, float *pScratch, int32_t log2N)

Computes the IFFT of complex-valued 16-bit floating point data.

Performs an N-point Inverse Fast Fourier Transform (IFFT) on a complex-valued float16 input stream. The IFFT size N must be a power of 2, with valid range: 16 □ N □ 16384.

Note:

- Data precision and format follow the argument types.
- float* is used to represent float16 pointers.
- The input, output and scratch buffers must be separately allocated and non-overlapping.

Parameters

- pY – **[out]** Pointer to the IFFT output buffer (complex-valued float16).
- pX – **[in]** Pointer to the IFFT input buffer (complex-valued float16).
- pScratch – **[in]** Pointer to a scratch buffer (at least N × 4 bytes).
- log2N – **[in]** Base-2 logarithm of the IFFT size N.

Returns
Command execution status.

int32_t CE_TransformIFFT_F32(float *pY, float *pX, float *pScratch, int32_t log2N)

Computes the IFFT of complex-valued 32-bit floating point data.

Performs an N-point Inverse Fast Fourier Transform (IFFT) on a complex-valued float32 input stream. The IFFT size N must be a power of 2, with valid range: 32 □ N □ 16384.

Note: The input, output and scratch buffers must be separately allocated and non-overlapping.

Parameters

- pY – **[out]** Pointer to the IFFT output buffer (complex-valued float32).
- pX – **[in]** Pointer to the IFFT input buffer (complex-valued float32).
- pScratch – **[in]** Pointer to a scratch buffer (at least N × 8 bytes).
- log2N – **[in]** Base-2 logarithm of the IFFT size N.

Returns
Command execution status.

2.9 Clock Driver

uint32_t CLOCK_GetSysClkFreq(*scg_sys_clk_t* type)

Gets the SCG system clock frequency.

This function gets the SCG system clock frequency. These clocks are used for core, platform, external, and bus clock domains.

Parameters

- type – Which type of clock to get, core clock or slow clock.

Returns

Clock frequency.

static inline void CLOCK_SetRunModeSysClkConfig(const *scg_sys_clk_config_t* *config)

Sets the system clock configuration for RUN mode.

This function sets the system clock configuration for RUN mode.

Parameters

- config – Pointer to the configuration.

static inline void CLOCK_GetCurSysClkConfig(*scg_sys_clk_config_t* *config)

Gets the system clock configuration in the current power mode.

This function gets the system configuration in the current power mode.

Parameters

- config – Pointer to the configuration.

static inline void CLOCK_SetClkOutSel(*clock_clkout_src_t* setting)

Sets the clock out selection.

This function sets the clock out selection (CLKOUTSEL).

Parameters

- setting – The selection to set.

status_t CLOCK_InitSysOsc(const *scg_sosc_config_t* *config)

Initializes the SCG system OSC.

This function enables the SCG system OSC clock according to the configuration.

Note: This function can't detect whether the system OSC has been enabled and used by an IP.

Parameters

- config – Pointer to the configuration structure.

Return values

- kStatus_Success – System OSC is initialized.
- kStatus_SCG_Busy – System OSC has been enabled and is used by the system clock.
- kStatus_ReadOnly – System OSC control register is locked.

status_t CLOCK_DeinitSysOsc(void)

De-initializes the SCG system OSC.

This function disables the SCG system OSC clock.

Note: This function can't detect whether the system OSC is used by an IP.

Return values

- kStatus_Success – System OSC is deinitialized.
- kStatus_SCG_Busy – System OSC is used by the system clock.
- kStatus_ReadOnly – System OSC control register is locked.

uint32_t CLOCK_GetSysOscFreq(void)

Gets the SCG system OSC clock frequency (SYSOSC).

Returns

Clock frequency; If the clock is invalid, returns 0.

static inline bool CLOCK_IsSysOscErr(void)

Checks whether the system OSC clock error occurs.

Returns

True if the error occurs, false if not.

static inline void CLOCK_ClearSysOscErr(void)

Clears the system OSC clock error.

static inline void CLOCK_SetSysOscMonitorMode(*scg_sosc_monitor_mode_t* mode)

Sets the system OSC monitor mode.

This function sets the system OSC monitor mode. The mode can be disabled, it can generate an interrupt when the error is disabled, or reset when the error is detected.

Parameters

- mode – Monitor mode to set.

static inline bool CLOCK_IsSysOscValid(void)

Checks whether the system OSC clock is valid.

Returns

True if clock is valid, false if not.

static inline void CLOCK_UnlockSysOscControlStatusReg(void)

Unlock the SOSCCSR control status register.

static inline void CLOCK_LockSysOscControlStatusReg(void)

Lock the SOSCCSR control status register.

status_t CLOCK_InitFirc(const *scg_firc_config_t* *config)

Initializes the SCG fast IRC clock.

This function enables the SCG fast IRC clock according to the configuration.

Note: This function can't detect whether the FIRC has been enabled and used by an IP.

Parameters

- config – Pointer to the configuration structure.

Return values

- kStatus_Success – FIRC is initialized.
- kStatus_SCG_Busy – FIRC has been enabled and is used by the system clock.
- kStatus_ReadOnly – FIRC control register is locked.

status_t CLOCK_DeinitFirc(void)

De-initializes the SCG fast IRC.

This function disables the SCG fast IRC.

Note: This function can't detect whether the FIRC is used by an IP.

Return values

- kStatus_Success – FIRC is deinitialized.
- kStatus_SCG_Busy – FIRC is used by the system clock.
- kStatus_ReadOnly – FIRC control register is locked.

uint32_t CLOCK_GetFircFreq(void)

Gets the SCG FIRC clock frequency.

Returns
Clock frequency; If the clock is invalid, returns 0.

static inline bool CLOCK_IsFircErr(void)

Checks whether the FIRC clock error occurs.

Returns
True if the error occurs, false if not.

static inline void CLOCK_ClearFircErr(void)

Clears the FIRC clock error.

static inline bool CLOCK_IsFircValid(void)

Checks whether the FIRC clock is valid.

Returns
True if clock is valid, false if not.

static inline void CLOCK_UnlockFircControlStatusReg(void)

Unlock the FIRCCSR control status register.

static inline void CLOCK_LockFircControlStatusReg(void)

Lock the FIRCCSR control status register.

static inline bool CLOCK_IsFIRCAutoTrimLocked(void)

Check whether FIRC auto trim locked to target frequency range.

When FIRCTREN and FIRCTRUP are enabled, TRIM_LOCK will indicate when auto trimming is complete and output FIRC frequency has locked to target FIRC range. TRIM_LOCK will automatically get cleared if FIRCTREN and FIRCTRUP are not set.

Returns
True if FIRC trim locked to target frequency range, false if not.

status_t CLOCK_InitRosc(const *scg_rosc_config_t* *config)

brief Initializes the SCG ROSC.

This function enables the SCG ROSC clock according to the configuration.

param config Pointer to the configuration structure. retval kStatus_Success ROSC is initialized. retval kStatus_SCG_Busy ROSC has been enabled and is used by the system clock. retval kStatus_ReadOnly ROSC control register is locked.

note This function can't detect whether the system OSC has been enabled and used by an IP.

status_t CLOCK_DeinitRosc(void)

brief De-initializes the SCG ROSC.

This function disables the SCG ROSC clock.

retval kStatus_Success System OSC is deinitialized. retval kStatus_SCG_Busy System OSC is used by the system clock. retval kStatus_ReadOnly System OSC control register is locked.

note This function can't detect whether the ROSC is used by an IP.

uint32_t CLOCK_GetRtcOscFreq(void)

Gets the SCG RTC OSC clock frequency.

Returns
Clock frequency; If the clock is invalid, returns 0.

status_t CLOCK_InitRfFro192M(const *fro192m_rf_clk_config_t* *config)

Initializes the FRO192M clock for the Radio Mode Controller.

This function configure the RF FRO192M clock according to the configuration.

Parameters

- config – Pointer to the configuration structure.

Return values
kStatus_Success – RF FRO192M is configured.

uint32_t CLOCK_GetRfFro192MFreq(void)

Gets the FRO192M clock frequency.

Returns
Clock frequency; If the clock is invalid, returns 0.

static inline bool CLOCK_IsRoscErr(void)

Checks whether the ROSC clock error occurs.

Returns
True if the error occurs, false if not.

static inline void CLOCK_ClearRoscErr(void)

Clears the ROSC clock error.

static inline void CLOCK_SetRoscMonitorMode(*scg_rosc_monitor_mode_t* mode)

Sets the ROSC monitor mode.

This function sets the ROSC monitor mode. The mode can be disabled, it can generate an interrupt when the error is disabled, or reset when the error is detected.

Parameters

- mode – Monitor mode to set.

static inline bool CLOCK_IsRoscValid(void)

Checks whether the ROSC clock is valid.

Returns

True if clock is valid, false if not.

static inline void CLOCK_UnlockRoscControlStatusReg(void)

Unlock the ROSCCSR control status register.

static inline void CLOCK_LockRoscControlStatusReg(void)

Lock the ROSCCSR control status register.

static inline void CLOCK_SetXtal0Freq(uint32_t freq)

Sets the XTAL0 frequency based on board settings.

Parameters

- freq – The XTAL0/EXTAL0 input clock frequency in Hz.

static inline void CLOCK_SetXtal32Freq(uint32_t freq)

Sets the XTAL32 frequency based on board settings.

Parameters

- freq – The XTAL32/EXTAL32 input clock frequency in Hz.

FSL_CLOCK_DRIVER_VERSION

CLOCK driver version 2.3.1.

enum _clock_name

Clock name used to get clock frequency.

These clocks source would be generated from SCG module.

Values:

enumerator kCLOCK_CoreSysClk

Cortex M33 clock.

enumerator kCLOCK_SlowClk

SLOW_CLK with DIVSLOW.

enumerator kCLOCK_PlatClk

PLAT_CLK.

enumerator kCLOCK_SysClk

SYS_CLK.

enumerator kCLOCK_BusClk

BUS_CLK with DIVBUS.

enumerator kCLOCK_ScgSysOscClk

SCG system OSC clock.

enumerator kCLOCK_ScgFircClk

SCG FIRC clock.

enumerator kCLOCK_RtcOscClk

RTC OSC clock.

enum _clock_ip_control

Clock source for peripherals that support various clock selections.

These options are for MRCC->XX[CC]

Values:

enumerator kCLOCK_IpClkControl_fun0

Peripheral clocks are disabled, module does not stall low power mode entry.

enumerator kCLOCK_IpClkControl_fun1

Peripheral clocks are enabled, module does not stall low power mode entry.

enumerator kCLOCK_IpClkControl_fun2

Peripherals clocks are enabled unless peripheral is idle, low power mode entry will stall until peripheral is idle.

enumerator kCLOCK_IpClkControl_fun3

Peripheral clocks are enabled unless in SLEEP mode (or lower), low power mode entry will stall until peripheral is idle Peripheral functional clocks that remain enabled in SLEEP mode are enabled and do not stall low power mode entry unless entering DEEPSLEEP mode (or lower)

enum _clock_ip_src

Clock source for peripherals that support various clock selections.

These options are for MRCC->XX[MUX].

Values:

enumerator kCLOCK_IpSrcFro6M

FRO 6M clock.

```
@note The SIRC has been removed, but in some
low-power wakeup use cases, the SIRC is still
required. This option provides the ability to
configure the clock source to SIRC.
```

enumerator kCLOCK_IpSrcFro192M

FRO 192M clock.

enumerator kCLOCK_IpSrcSoscClk

OSC RF clock.

enumerator kCLOCK_IpSrc32kClk

32k Clk clock.

enum _tpm2_ip_src

Clock source for TPM2.

These options are for RF_CMC1->TPM2_CFG[CLK_MUX_SEL].

Values:

enumerator kCLOCK_Tpm2SrcCoreClk
Core Clock.

enumerator kCLOCK_Tpm2SrcSoscClk
Radio Oscillator.

enum _clock_ip_name
Clock IP name.

Values:

enumerator kCLOCK_NOGATE
No clock gate for the IP in MRCC

enumerator kCLOCK_Ewm0
Clock ewm0

enumerator kCLOCK_Syspm0
Clock syspm0

enumerator kCLOCK_Wdog0
Clock wdog0

enumerator kCLOCK_Wdog1
Clock wdog1

enumerator kCLOCK_Sfa0
Clock sfa0

enumerator kCLOCK_Crc0
Clock crc0

enumerator kCLOCK_Secsubsys
Clock secsubsys

enumerator kCLOCK_Lpit0
Clock lpit0

enumerator kCLOCK_Tpm0
Clock tpm0

enumerator kCLOCK_Tpm1
Clock tpm1

enumerator kCLOCK_Lpi2c0
Clock lpi2c0

enumerator kCLOCK_Lpi2c1
Clock lpi2c1

enumerator kCLOCK_I3c0
Clock i3c

enumerator kCLOCK_Lpspi0
Clock lpspi0

enumerator kCLOCK_Lpspi1
Clock lpspi1

enumerator kCLOCK_Lpuart0
Clock lpuart0

enumerator kCLOCK_Lpuart1
Clock lpuart1

enumerator kCLOCK_Flexio0
Clock Flexio0

enumerator kCLOCK_Can0
Clock Can0

enumerator kCLOCK_Sema0
Clock Sema0

enumerator kCLOCK_Data_stream_2p4
Clock data_stream_2p4

enumerator kCLOCK_PortA
Clock portA

enumerator kCLOCK_PortB
Clock portB

enumerator kCLOCK_PortC
Clock portC

enumerator kCLOCK_Lpadc0
Clock lpadc0

enumerator kCLOCK_Lpcmp0
Clock lpcmp0

enumerator kCLOCK_Lpcmp1
Clock lpcmp1

enumerator kCLOCK_Vref0
Clock verf0

enumerator kCLOCK_Mtr_master
Clock mtr_master

enumerator kCLOCK_Can1
Clock Can1

enumerator kCLOCK_GpioA
Clock gpioA

enumerator kCLOCK_GpioB
Clock gpioB

enumerator kCLOCK_GpioC
Clock gpioC

enumerator kCLOCK_Dma0
Clock dma0

enumerator kCLOCK_Pflexnvm
Clock pflexnvm

enumerator kCLOCK_Sram0
Clock Sram0

enumerator kCLOCK_Sram1
Clock Sram1

enumerator kCLOCK_Sram2
Clock Sram2

enumerator kCLOCK_Sram3
Clock Sram3

enumerator kCLOCK_Sram0_NOECC
Clock Sram0 NOECC

enumerator kCLOCK_DSP0
Clock DSPV

enumerator kCLOCK_DSP0_MUA
Clock DSPV MUA

enumerator kCLOCK_Sram1_NOECC
Clock Sram1 NOECC

enumerator kCLOCK_Rf_2p4ghz_bist
Clock rf_2p4ghz_bist

enumerator kCLOCK_Lptmr0_wake2vsys
Clock LPTMR0 IPG clk erclk wake2vsys

enumerator kCLOCK_Lptmr1_wake2vsys
Clock LPTMR1 IPG clk erclk wake2vsys

enumerator kCLOCK_Lptmr2_wake2vsys
Clock LPTMR2 IPG clk erclk wake2vsys

enum _scg_status
SCG status return codes.
Values:

enumerator kStatus_SCG_Busy
Clock is busy.

enumerator kStatus_SCG_InvalidSrc
Invalid source.

enum _scg_sys_clk
SCG system clock type.
Values:

enumerator kSCG_SysClkSlow
System slow clock.

enumerator kSCG_SysClkBus
Bus clock.

enumerator kSCG_SysClkPlatform
Platform clock.

enumerator kSCG_SysClkCore
Core clock.

enum _scg_sys_clk_src
SCG system clock source.
Values:

enumerator kSCG_SysClkSrcSysOsc
System OSC.

enumerator kSCG_SysClkSrcSirc
Slow IRC.

enumerator kSCG_SysClkSrcFirc
Fast IRC.

enumerator kSCG_SysClkSrcRosc
RTC OSC.

enum _scg_sys_clk_div
SCG system clock divider value.

Values:

enumerator kSCG_SysClkDivBy1
Divided by 1.

enumerator kSCG_SysClkDivBy2
Divided by 2.

enumerator kSCG_SysClkDivBy3
Divided by 3.

enumerator kSCG_SysClkDivBy4
Divided by 4.

enumerator kSCG_SysClkDivBy5
Divided by 5.

enumerator kSCG_SysClkDivBy6
Divided by 6.

enumerator kSCG_SysClkDivBy7
Divided by 7.

enumerator kSCG_SysClkDivBy8
Divided by 8.

enumerator kSCG_SysClkDivBy9
Divided by 9.

enumerator kSCG_SysClkDivBy10
Divided by 10.

enumerator kSCG_SysClkDivBy11
Divided by 11.

enumerator kSCG_SysClkDivBy12
Divided by 12.

enumerator kSCG_SysClkDivBy13
Divided by 13.

enumerator kSCG_SysClkDivBy14
Divided by 14.

enumerator kSCG_SysClkDivBy15
Divided by 15.

enumerator kSCG_SysClkDivBy16
Divided by 16.

enum _clock_clkout_src
SCG clock out configuration (CLKOUTSEL).
Values:

enumerator kClockClkoutSelScgSlow
SCG Slow clock.

enumerator kClockClkoutSelSosc
System OSC.

enumerator kClockClkoutSelFirc
Fast IRC.

enumerator kClockClkoutSelScgRtcOsc
SCG RTC OSC clock.

enum _scg_sosc_monitor_mode
SCG system OSC monitor mode.
Values:

enumerator kSCG_SysOscMonitorDisable
Monitor disabled.

enumerator kSCG_SysOscMonitorInt
Interrupt when the SOSC error is detected.

enumerator kSCG_SysOscMonitorReset
Reset when the SOSC error is detected.

SOSC enable mode.
Values:

enumerator kSCG_SoscDisable
Disable SOSC clock.

enumerator kSCG_SoscEnable
Enable SOSC clock.

enumerator kSCG_SoscEnableInSleep
Enable SOSC in sleep mode.

enum _scg_rosc_monitor_mode
SCG ROSC monitor mode.

Values:

enumerator kSCG_RoscMonitorDisable
Monitor disabled.

enumerator kSCG_RoscMonitorInt
Interrupt when the RTC OSC error is detected.

enumerator kSCG_RoscMonitorReset
Reset when the RTC OSC error is detected.

enum _scg_firc_trim_mode
SCG fast IRC trim mode.

Values:

enumerator kSCG_FircTrimNonUpdate
FIRC trim enable but not enable trim value update. In this mode, the trim value is fixed to the initialized value which is defined by trimCoar and trimFine in configure structure *scg_firc_trim_config_t*.

enumerator kSCG_FircTrimUpdate
FIRC trim enable and trim value update enable. In this mode, the trim value is auto update.

enum _scg_firc_trim_src
SCG fast IRC trim source.

Values:

enumerator kSCG_FircTrimSrcSysOsc
System OSC.

enumerator kSCG_FircTrimSrcRtcOsc
RTC OSC (32.768 kHz).

FIRC enable mode.

Values:

enumerator kSCG_FircDisable
Disable FIRC clock.

enumerator kSCG_FircEnable
Enable FIRC clock.

enumerator kSCG_FircEnableInSleep
Enable FIRC in sleep mode.

enum _scg_firc_range
SCG fast IRC clock frequency range.
Values:

enumerator kSCG_FircRange48M
Fast IRC is trimmed to 48 MHz.

enumerator kSCG_FircRange64M
Fast IRC is trimmed to 64 MHz.

enumerator kSCG_FircRange96M
Fast IRC is trimmed to 96 MHz.

enumerator kSCG_FircRange192M
Fast IRC is trimmed to 192 MHz.

enum _fro192m_rf_range
FRO192M RF clock frequency range.
Values:

enumerator kFro192M_Range16M
FRO192M output frequenc 16 MHz.

enumerator kFro192M_Range24M
FRO192M output frequenc 24 MHz.

enumerator kFro192M_Range32M
FRO192M output frequenc 32 MHz.

enumerator kFro192M_Range48M
FRO192M output frequenc 48 MHz.

enumerator kFro192M_Range64M
FRO192M output frequenc 64 MHz.

enum _fro192m_rf_clk_div
RF Flash APB and RF_CMC clock divide.
Values:

enumerator kFro192M_ClkDivBy1
Divided by 1.

enumerator kFro192M_ClkDivBy2
Divided by 2.

enumerator kFro192M_ClkDivBy4
Divided by 4.

enumerator kFro192M_ClkDivBy8
Divided by 8.

typedef enum *_clock_name* clock_name_t
Clock name used to get clock frequency.

These clocks source would be generated from SCG module.

typedef enum *_clock_ip_control* clock_ip_control_t
Clock source for peripherals that support various clock selections.

These options are for MRCC->XX[CC]

typedef enum *_clock_ip_src* clock_ip_src_t
Clock source for peripherals that support various clock selections.

These options are for MRCC->XX[MUX].

typedef enum *_tpm2_ip_src* tpm2_src_t
Clock source for TPM2.

These options are for RF_CMC1->TPM2_CFG[CLK_MUX_SEL].

typedef enum *_clock_ip_name* clock_ip_name_t
Clock IP name.

typedef enum *_scg_sys_clk* scg_sys_clk_t
SCG system clock type.

typedef enum *_scg_sys_clk_src* scg_sys_clk_src_t
SCG system clock source.

typedef enum *_scg_sys_clk_div* scg_sys_clk_div_t
SCG system clock divider value.

typedef struct *_scg_sys_clk_config* scg_sys_clk_config_t
SCG system clock configuration.

typedef enum *_clock_clkout_src* clock_clkout_src_t
SCG clock out configuration (CLKOUTSEL).

typedef enum *_scg_sosc_monitor_mode* scg_sosc_monitor_mode_t
SCG system OSC monitor mode.

typedef struct *_scg_sosc_config* scg_sosc_config_t
SCG system OSC configuration.

typedef enum *_scg_rosc_monitor_mode* scg_rosc_monitor_mode_t
SCG ROSC monitor mode.

typedef struct *_scg_rosc_config* scg_rosc_config_t

SCG ROSC configuration.

typedef enum *_scg_firc_trim_mode* scg_firc_trim_mode_t

SCG fast IRC trim mode.

typedef enum *_scg_firc_trim_src* scg_firc_trim_src_t

SCG fast IRC trim source.

typedef struct *_scg_firc_trim_config* scg_firc_trim_config_t

SCG fast IRC clock trim configuration.

typedef enum *_scg_firc_range* scg_firc_range_t

SCG fast IRC clock frequency range.

typedef struct *_scg_firc_config_t* scg_firc_config_t

SCG fast IRC clock configuration.

typedef enum *_fro192m_rf_range* fro192m_rf_range_t

FRO192M RF clock frequency range.

typedef enum *_fro192m_rf_clk_div* fro192m_rf_clk_div_t

RF Flash APB and RF_CMC clock divide.

typedef struct *_fro192m_rf_clk_config* fro192m_rf_clk_config_t

FRO192M RF clock configuration.

volatile uint32_t g_xtal0Freq

External XTAL0 (OSC0/SYSOSC) clock frequency.

The XTAL0/EXTAL0 (OSC0/SYSOSC) clock frequency in Hz. When the clock is set up, use the function CLOCK_SetXtal0Freq to set the value in the clock driver. For example, if XTAL0 is 8 MHz:

```
CLOCK_InitSysOsc(...);
CLOCK_SetXtal0Freq(80000000);
```

This is important for the multicore platforms where only one core needs to set up the OSC0/SYSOSC using CLOCK_InitSysOsc. All other cores need to call the CLOCK_SetXtal0Freq to get a valid clock frequency.

volatile uint32_t g_xtal32Freq

External XTAL32/EXTAL32 clock frequency.

The XTAL32/EXTAL32 clock frequency in Hz. When the clock is set up, use the function CLOCK_SetXtal32Freq to set the value in the clock driver.

This is important for the multicore platforms where only one core needs to set up the clock. All other cores need to call the CLOCK_SetXtal32Freq to get a valid clock frequency.

static inline void CLOCK_EnableClock(*clock_ip_name_t* name)

Enable the clock for specific IP.

Parameters

- name – Which clock to enable, see *clock_ip_name_t*.

static inline void CLOCK_EnableTPM2(void)

Enable the TPM2 clock.

static inline void CLOCK_EnableClockLPMode(*clock_ip_name_t* name, *clock_ip_control_t* control)

Enable the clock for specific IP in low power mode.

Parameters

- name – Which clock to enable, see *clock_ip_name_t*.
- control – Clock Config, see *clock_ip_control_t*.

static inline void CLOCK_DisableClock(*clock_ip_name_t* name)

Disable the clock for specific IP.

Parameters

- name – Which clock to disable, see *clock_ip_name_t*.

static inline void CLOCK_DisableTPM2(void)

Disable the TPM2 clock.

static inline void CLOCK_SetIpSrc(*clock_ip_name_t* name, *clock_ip_src_t* src)

Set the clock source for specific IP module.

Set the clock source for specific IP, not all modules need to set the clock source, should only use this function for the modules need source setting.

Parameters

- name – Which peripheral to check, see *clock_ip_name_t*.
- src – Clock source to set.

static inline void CLOCK_SetTpm2Src(*tpm2_src_t* src)

Set the clock source for TPM2.

Parameters

- src – Clock source to set.

static inline void CLOCK_SetIpSrcDiv(*clock_ip_name_t* name, uint8_t divValue)

Set the clock source and divider for specific IP module.

Set the clock source and divider for specific IP, not all modules need to set the clock source and divider, should only use this function for the modules need source and divider setting.

Divider output clock = Divider input clock / (divValue+1)]).

Parameters

- name – Which peripheral to check, see *clock_ip_name_t*.
- divValue – The divider value.

uint32_t CLOCK_GetFreq(*clock_name_t* clockName)

Gets the clock frequency for a specific clock name.

This function checks the current clock configurations and then calculates the clock frequency for a specific clock name defined in clock_name_t.

Parameters

- clockName – Clock names defined in clock_name_t

Returns

Clock frequency value in hertz

uint32_t CLOCK_GetCoreSysClkFreq(void)

Get the core clock or system clock frequency.

Returns
Clock frequency in Hz.

uint32_t CLOCK_GetPlatClkFreq(void)

Get the platform clock frequency.

Returns
Clock frequency in Hz.

uint32_t CLOCK_GetBusClkFreq(void)

Get the bus clock frequency.

Returns
Clock frequency in Hz.

uint32_t CLOCK_GetFlashClkFreq(void)

Get the flash clock frequency.

Returns
Clock frequency in Hz.

uint32_t CLOCK_GetIpFreq(*clock_ip_name_t* name)

Gets the functional clock frequency for a specific IP module.

This function gets the IP module's functional clock frequency based on MRCC registers. It is only used for the IP modules which could select clock source by MRCC[PCS].

Parameters

- name – Which peripheral to get, see *clock_ip_name_t*.

Returns
Clock frequency value in Hz

FSL_SDK_DISABLE_DRIVER_CLOCK_CONTROL

Configure whether driver controls clock.

When set to 0, peripheral drivers will enable clock in initialize function and disable clock in de-initialize function. When set to 1, peripheral driver will not control the clock, application could control the clock out of the driver.

Note: All drivers share this feature switcher. If it is set to 1, application should handle clock enable and disable for all drivers.

SDK_DEVICE_MAXIMUM_CPU_CLOCK_FREQUENCY

EDMA_CLOCKS

Clock ip name array for EDMA.

SYSPM_CLOCKS

Clock ip name array for SYSPM.

SFA_CLOCKS

Clock ip name array for SFA.

CRC_CLOCKS
Clock ip name array for CRC.

TPM_CLOCKS
Clock ip name array for TPM.

LPI2C_CLOCKS
Clock ip name array for LPI2C.

I3C_CLOCKS
Clock ip name array for I3C.

LPSPI_CLOCKS
Clock ip name array for LPSPI.

LPUART_CLOCKS
Clock ip name array for LPUART.

PORT_CLOCKS
Clock ip name array for PORT.

LPADC_CLOCKS
Clock ip name array for LPADC.

LPCMP_CLOCKS
Clock ip name array for LPCMP.

VREF_CLOCKS
Clock ip name array for VREF.

GPIO_CLOCKS
Clock ip name array for GPIO.

LPIT_CLOCKS
Clock ip name array for LPIT.

RF_CLOCKS
Clock ip name array for RF.

WDOG_CLOCKS
Clock ip name array for WDOG.

FLEXCAN_CLOCKS
Clock ip name array for FLEXCAN.

FLEXIO_CLOCKS
Clock ip name array for FLEXIO.

TSTMR_CLOCKS
Clock ip name array for TSTMR.

EWM_CLOCKS
Clock ip name array for EWM.

SEMA42_CLOCKS
Clock ip name array for SEMA42.

MU_CLOCKS
Clock ip name array for MU.

MAKE_MRCC_REGADDR(base, offset)
"IP Connector name difinition used for clock gate, clock source and clock divider setting. It is defined as the corresponding register address.

CLOCK_REG(name)

struct __scg_sys_clk_config
#include <fsl_clock.h> SCG system clock configuration.

Public Members

uint32_t divSlow
Slow clock divider, see *scg_sys_clk_div_t*.

uint32_t divBus
Bus clock divider, see *scg_sys_clk_div_t*.

uint32_t ___pad0___
Reserved.

uint32_t divCore
Core clock divider, see *scg_sys_clk_div_t*.

uint32_t ___pad1___
Reserved.

uint32_t src
System clock source, see *scg_sys_clk_src_t*.

uint32_t ___pad2___
reserved.

struct __scg_sosc_config
#include <fsl_clock.h> SCG system OSC configuration.

Public Members

uint32_t freq
System OSC frequency.

uint32_t enableMode
Enable mode, OR'ed value of _scg_sosc_enable_mode.

scg_sosc_monitor_mode_t monitorMode
Clock monitor mode selected.

struct __scg__rosc__config
#include <fsl_clock.h> SCG ROSC configuration.

Public Members

scg_rosc_monitor_mode_t monitorMode
Clock monitor mode selected.

struct __scg__firc__trim__config
#include <fsl_clock.h> SCG fast IRC clock trim configuration.

Public Members

scg_firc_trim_mode_t trimMode
FIRC trim mode.

scg_firc_trim_src_t trimSrc
Trim source.

uint16_t trimDiv
Divider of SOSC for FIRC.

uint8_t trimCoar
Trim coarse value; Irrelevant if trimMode is kSCG_FircTrimUpdate.

uint8_t trimFine
Trim fine value; Irrelevant if trimMode is kSCG_FircTrimUpdate.

struct __scg__firc__config__t
#include <fsl_clock.h> SCG fast IRC clock configuration.

Public Members

uint32_t enableMode
Enable mode.

scg_firc_range_t range
Fast IRC frequency range.

const *scg_firc_trim_config_t* *trimConfig
Pointer to the FIRC trim configuration; set NULL to disable trim.

struct _fro192m_rf_clk_config
#include <fsl_clock.h> FRO192M RF clock configuration.

Public Members

fro192m_rf_range_t range
FRO192M RF clock frequency range.

fro192m_rf_clk_div_t apb_rfcmc_div
RF Flash APB and RF_CMC clock divide.

2.10 CMC: Core Mode Controller Driver

void CMC_SetClockMode(CMC_Type *base, *cmc_clock_mode_t* mode)
Sets clock mode.

This function configs the amount of clock gating when the core asserts Sleeping due to WFI, WFE or SLEEPONEXIT.

Parameters

- base – CMC peripheral base address.
- mode – System clock mode.

static inline void CMC_LockClockModeSetting(CMC_Type *base)
Locks the clock mode setting.

After invoking this function, any clock mode setting will be blocked.

Parameters

- base – CMC peripheral base address.

static inline *cmc_core_clock_gate_status_t* CMC_GetCoreClockGatedStatus(CMC_Type *base)
Gets the core clock gated status.

This function get the status to indicate whether the core clock is gated. The core clock gated status can be cleared by software.

Parameters

- base – CMC peripheral base address.

Returns
The status to indicate whether the core clock is gated.

static inline void CMC_ClearCoreClockGatedStatus(CMC_Type *base)
Clears the core clock gated status.

This function clear clock status flag by software.

Parameters

- base – CMC peripheral base address.

static inline uint8_t CMC_GetWakeupSource(CMC_Type *base)

Gets the Wakeup Source.

This function gets the Wakeup sources from the previous low power mode entry.

Parameters

- base – CMC peripheral base address.

Returns

The Wakeup sources from the previous low power mode entry. See *_cmc_wakeup_sources* for details.

static inline *cmc_clock_mode_t* CMC_GetClockMode(CMC_Type *base)

Gets the Clock mode.

This function gets the clock mode of the previous low power mode entry.

Parameters

- base – CMC peripheral base address.

Returns

The Low Power status.

static inline uint32_t CMC_GetSystemResetStatus(CMC_Type *base)

Gets the System reset status.

This function returns the system reset status. Those status updates on every MAIN Warm Reset to indicate the type/source of the most recent reset.

Parameters

- base – CMC peripheral base address.

Returns

The most recent system reset status. See *_cmc_system_reset_sources* for details.

static inline uint32_t CMC_GetStickySystemResetStatus(CMC_Type *base)

Gets the sticky system reset status since the last WAKE Cold Reset.

This function gets all source of system reset that have generated a system reset since the last WAKE Cold Reset, and that have not been cleared by software.

Parameters

- base – CMC peripheral base address.

Returns

System reset status that have not been cleared by software. See *_cmc_system_reset_sources* for details.

static inline void CMC_ClearStickySystemResetStatus(CMC_Type *base, uint32_t mask)

Clears the sticky system reset status flags.

Parameters

- base – CMC peripheral base address.
- mask – Bitmap of the sticky system reset status to be cleared.

static inline uint8_t CMC_GetResetCount(CMC_Type *base)

Gets the number of reset sequences completed since the last WAKE Cold Reset.

Parameters

- base – CMC peripheral base address.

Returns
The number of reset sequences.

static inline uint32_t CMC_GetResetInitStatusFlags(CMC_Type *base)

Gets status flags to indicate if any errors occurred during the reset initialization sequence.

Parameters

- base – CMC peripheral base address.

Returns
Status flags that indicate errors occurred during the reset initialization sequence.

void CMC_SetPowerModeProtection(CMC_Type *base, uint32_t allowedModes)

Configures all power mode protection settings.

This function configures the power mode protection settings for supported power modes. This should be done before set the lowPower mode for each power doamin.

The allowed lowpower modes are passed as bit map. For example, to allow Sleep and DeepSleep, use CMC_SetPowerModeProtection(CMC_base, kCMC_AllowSleepMode|kCMC_AllowDeepSleepMode). To allow all low power modes, use CMC_SetPowerModeProtection(CMC_base, kCMC_AllowAllLowPowerModes).

Parameters

- base – CMC peripheral base address.
- allowedModes – Bitmaps of the allowed power modes. See *_cmc_power_mode_protection* for details.

static inline void CMC_LockPowerModeProtectionSetting(CMC_Type *base)

Locks the power mode protection.

This function locks the power mode protection. After invoking this function, any power mode protection setting will be ignored.

Parameters

- base – CMC peripheral base address.

static inline void CMC_SetGlobalPowerMode(CMC_Type *base, *cmc_low_power_mode_t* lowPowerMode)

Config the same lowPower mode for all power domain.

This function configures the same low power mode for MAIN power domian and WAKE power domain.

Parameters

- base – CMC peripheral base address.
- lowPowerMode – The desired lowPower mode. See *cmc_low_power_mode_t* for details.

static inline void CMC_SetMAINPowerMode(CMC_Type *base, *cmc_low_power_mode_t* lowPowerMode)

Configures entry into low power mode for the MAIN Power domain.

This function configures the low power mode for the MAIN power domian, when the core executes WFI/WFE instruction. The available lowPower modes are defined in the *cmc_low_power_mode_t*.

Parameters

- base – CMC peripheral base address.

- lowPowerMode – The desired lowPower mode. See *cmc_low_power_mode_t* for details.

static inline *cmc_low_power_mode_t* CMC_GetMAINPowerMode(CMC_Type *base)

Gets the power mode of the MAIN Power domain.

Parameters

- base – CMC peripheral base address.

Returns

The power mode of MAIN Power domain. See *cmc_low_power_mode_t* for details.

static inline void CMC_SetWAKEPowerMode(CMC_Type *base, *cmc_low_power_mode_t* lowPowerMode)

Configure entry into low power mode for the WAKE Power domain.

This function configures the low power mode for the WAKE power domian, when the core executes WFI/WFE instruction. The available lowPower mode are defined in the *cmc_low_power_mode_t*.

Note: The lowPower Mode for the WAKE domain must not be configured to a lower power mode than any other power domain.

Parameters

- base – CMC peripheral base address.
- lowPowerMode – The desired lowPower mode. See *cmc_low_power_mode_t* for details.

static inline *cmc_low_power_mode_t* CMC_GetWAKEPowerMode(CMC_Type *base)

Gets the power mode of the WAKE Power domain.

Parameters

- base – CMC peripheral base address.

Returns

The power mode of WAKE Power domain. See *cmc_low_power_mode_t* for details.

void CMC_ConfigResetPin(CMC_Type *base, const *cmc_reset_pin_config_t* *config)

Configure reset pin.

This function configures reset pin. When enabled, the low power filter is enabled in both Active and Low power modes, the reset filter is only enabled in Active mode. When both filers are enabled, they operate in series.

Parameters

- base – CMC peripheral base address.
- config – Pointer to the reset pin config structure.

static inline void CMC_EnableSystemResetInterrupt(CMC_Type *base, uint32_t mask)

Enable system reset interrupts.

This function enables the system reset interrupts. The assertion of non-fatal warm reset can be delayed for 258 cycles of the 32K_CLK clock while an enabled interrupt is generated. Then Software can perform a graceful shutdown or abort the non-fatal warm reset provided the pending reset source is cleared by resetting the reset source and then clearing the pending flag.

Parameters

- base – CMC peripheral base address.
- mask – System reset interrupts. See *_cmc_system_reset_interrupt_enable* for details.

static inline void CMC_DisableSystemResetInterrupt(CMC_Type *base, uint32_t mask)

Disable system reset interrupts.

This function disables the system reset interrupts.

Parameters

- base – CMC peripheral base address.
- mask – System reset interrupts. See *_cmc_system_reset_interrupt_enable* for details.

static inline uint32_t CMC_GetSystemResetInterruptFlags(CMC_Type *base)

Gets System Reset interrupt flags.

This function returns the System reset interrupt flags.

Parameters

- base – CMC peripheral base address.

Returns

System reset interrupt flags. See *_cmc_system_reset_interrupt_flag* for details.

static inline void CMC_ClearSystemResetInterruptFlags(CMC_Type *base, uint32_t mask)

Clears System Reset interrupt flags.

This function clears system reset interrupt flags. The pending reset source can be cleared by resetting the source of the reset and then clearing the pending flags.

Parameters

- base – CMC peripheral base address.
- mask – System Reset interrupt flags. See *_cmc_system_reset_interrupt_flag* for details.

static inline void CMC_EnableNonMaskablePinInterrupt(CMC_Type *base, bool enable)

Enable/Disable Non maskable Pin interrupt.

Parameters

- base – CMC peripheral base address.
- enable – Enable or disable Non maskable pin interrupt. true - enable Non-maskable pin interrupt. false - disable Non-maskable pin interupt.

static inline uint8_t CMC_GetISPMODEPinLogic(CMC_Type *base)

Gets the logic state of the ISPMODE_n pin.

This function returns the logic state of the ISPMODE_n pin on the last negation of RESET_b pin.

Parameters

- base – CMC peripheral base address.

Returns

The logic state of the ISPMODE_n pin on the last negation of RESET_b pin.

static inline void CMC_ClearISPMODEPinLogic(CMC_Type *base)

Clears ISPMODE_n pin state.

Parameters

- base – CMC peripheral base address.

static inline void CMC_ForceBootConfiguration(CMC_Type *base, bool assert)

Set the logic state of the BOOT_CONFIGn pin.

This function force the logic state of the Boot_Confign pin to assert on next system reset.

Parameters

- base – CMC peripheral base address.
- assert – Assert the corresponding pin or not. true - Assert corresponding pin on next system reset. false - No effect.

static inline uint32_t CMC_GetBootRomStatus(CMC_Type *base, uint8_t index)

Gets the information written by the BootROM.

Parameters

- base – CMC peripheral base address.
- index – The index of BootROM status register, ranges from 0.

Returns

The status information written by the BootROM.

static inline void CMC_WriteBootRomStatusReg(CMC_Type *base, uint8_t index, uint32_t value)

Writes value to BootROM status register, in this way, BootROM status registers are used as general purpose register.

Note: Value in BootROM status registers are reset in cold reset.

Parameters

- base – CMC peripheral base address.
- index – The index of BootROM status register, ranges from 0.
- value – Value to write.

void CMC_PowerOffSRAMAllMode(CMC_Type *base, uint32_t mask)

Power off the selected system SRAM always.

This function power off the selected system SRAM always. The SRAM arrays should not be accessed while they are shut down. SRAM array contents are not retained if they are powered off.

Parameters

- base – CMC peripheral base address.
- mask – Bitmap of the SRAM arrays to be powered off all modes. See *_cmc_system_sram_arrays* for details. Check Reference Manual for the SRAM region and mask bit relationship.

static inline void CMC_PowerOnSRAMAllMode(CMC_Type *base, uint32_t mask)

Power on SRAM during all mode.

Parameters

- base – CMC peripheral base address.

- mask – Bitmap of the SRAM arrays to be powered on all modes. See *_cmc_system_sram_arrays* for details. Check Reference Manual for the SRAM region and mask bit relationship.

void CMC_PowerOffSRAMLowPowerOnly(CMC_Type *base, uint32_t mask)

Power off the selected system SRAM during low power mode only.

This function power off the selected system SRAM only during low power mode. SRAM array contents are not retained if they are power off.

Parameters

- base – CMC peripheral base address.
- mask – Bitmap of the SRAM arrays to be power off during low power mode only. See *_cmc_system_sram_arrays* for details. Check Reference Manual for the SRAM region and mask bit relationship.

static inline void CMC_PowerOnSRAMLowPowerOnly(CMC_Type *base, uint32_t mask)

Power on the selected system SRAM during low power mode only.

This function power on the selected system SRAM. The SRAM arrray contents are retained in low power modes.

Parameters

- base – CMC peripheral base address.
- mask – Bitmap of the SRAM arrays to be power on during low power mode only. See *_cmc_system_sram_arrays* for details. Check Reference Manual for the SRAM region and mask bit relationship.

void CMC_ConfigFlashMode(CMC_Type *base, bool wake, bool doze, bool disable)

Configs the low power mode of the on-chip flash memory.

This function configs the low power mode of the on-chip flash memory.

Parameters

- base – CMC peripheral base address.
- wake – true: Flash will exit low power state during the flash memory accesses. false: No effect.
- doze – true: Flash is disabled while core is sleeping false: No effect.
- disable – true: Flash memory is placed in low power state. false: No effect.

static inline void CMC_EnableDebugOperation(CMC_Type *base, bool enable)

Enables/Disables debug Operation when the core sleep.

This function configs what happens to debug when core sleeps.

Parameters

- base – CMC peripheral base address.
- enable – Enable or disable Debug when Core is sleeping. true - Debug remains enabled when the core is sleeping. false - Debug is disabled when the core is sleeping.

void CMC_PreEnterLowPowerMode(void)

Prepares to enter low power modes.

This function should be called before entering low power modes.

void CMC_PostExitLowPowerMode(void)

Recovers after wake up from low power modes.

This function should be called after wake up from low power modes. This function should be used with *CMC_PreEnterLowPowerMode()*

void CMC_GlobalEnterLowPowerMode(CMC_Type *base, *cmc_low_power_mode_t* lowPowerMode)

Configs the entry into the same low power mode for each power domains.

This function provides the feature to entry into the same low power mode for each power domains. Before invoking this function, please ensure the selected power mode have been allowed.

Parameters

- base – CMC peripheral base address.
- lowPowerMode – The low power mode to be entered. See *cmc_low_power_mode_t* for the details.

void CMC_EnterLowPowerMode(CMC_Type *base, const *cmc_power_domain_config_t* *config)

Configs the entry into different low power modes for each power domains.

This function provides the feature to entry into different low power modes for each power domains. Before invoking this function please ensure the selected modes are allowed.

Parameters

- base – CMC peripheral base address.
- config – Pointer to the cmc_power_domain_config_t structure.

FSL_CMC_DRIVER_VERSION

CMC driver version 2.4.4.

CMC_SRAM_BUSY_TIMEOUT

Max loops to wait for CMC SRAM operation complete.

When configuring the SRAM, driver will wait for the completion of new settings. This parameter defines how many loops to check completion before return timeout. If defined as 0, driver will wait forever until completion.

enum _cmc_power_mode_protection

CMC power mode Protection enumeration.

Values:

enumerator kCMC_AllowSleepMode

Allow Sleep mode.

enumerator kCMC_AllowDeepSleepMode

Allow Deep Sleep mode.

enumerator kCMC_AllowPowerDownMode

Allow Power Down mode.

enumerator kCMC_AllowDeepPowerDownMode

Allow Deep Power Down mode.

enumerator kCMC_AllowAllLowPowerModes
Allow all low power modes.

enum _cmc_wakeup_sources
Wake up sources from the previous low power mode entry.
Values:

enumerator kCMC_WakeupFromResetInterruptOrPowerDown
Wakeup source is reset interrupt, or wake up from [Deep] Power Down.

enumerator kCMC_WakeupFromDebugRequest
Wakeup source is debug request.

enumerator kCMC_WakeupFromInterrupt
Wakeup source is interrupt.

enumerator kCMC_WakeupFromDMAWakeup
Wakeup source is DMA Wakeup.

enumerator kCMC_WakeupFromWUURequest
Wakeup source is WUU request.

enumerator kCMC_WakeupFromBusMaster
Wakeup source is Bus master.

enum _cmc_system_reset_interrupt_enable
System Reset Interrupt enable enumeration.
Values:

enumerator kCMC_PinResetInterruptEnable
Pin Reset interrupt enable.

enumerator kCMC_DAPResetInterruptEnable
DAP Reset interrupt enable.

enumerator kCMC_LowPowerAcknowledgeTimeoutResetInterruptEnable
Low Power Acknowledge Timeout Reset interrupt enable.

enumerator kCMC_SystemClockGenerationResetInterruptEnable
System Clock Generation Reset interrupt enable.

enumerator kCMC_Watchdog0ResetInterruptEnable
Watchdog 0 Reset interrupt enable.

enumerator kCMC_SoftwareResetInterruptEnable
Software Reset interrupt enable.

enumerator kCMC_LockupResetInterruptEnable
Lockup Reset interrupt enable.

enumerator kCMC_Watchdog1ResetInterruptEnable
Watchdog 1 Reset interrupt enable

enum _cmc_system_reset_interrupt_flag
CMC System Reset Interrupt Status flag.
Values:

enumerator kCMC_PinResetInterruptFlag
Pin Reset interrupt flag.

enumerator kCMC_DAPResetInterruptFlag
DAP Reset interrupt flag.

enumerator kCMC_LowPowerAcknowledgeTimeoutResetFlag
Low Power Acknowledge Timeout Reset interrupt flag.

enumerator kCMC_Watchdog0ResetInterruptFlag
Watchdog 0 Reset interrupt flag.

enumerator kCMC_SoftwareResetInterruptFlag
Software Reset interrupt flag.

enumerator kCMC_LockupResetInterruptFlag
Lock up Reset interrupt flag.

enumerator kCMC_Watchdog1ResetInterruptFlag
Watchdog 1 Reset interrupt flag.

enum _cmc_system_sram_arrays
CMC System SRAM arrays low power mode enable enumeration.
Values:

enumerator kCMC_SRAMBank0
Power off SRAM Bank0, please refer to chip's RM for the corresponding SRAM array.

enumerator kCMC_SRAMBank1
Power off SRAM Bank1, please refer to chip's RM for the corresponding SRAM array.

enumerator kCMC_SRAMBank2
Power off SRAM Bank2, please refer to chip's RM for the corresponding SRAM array.

enumerator kCMC_SRAMBank3
Power off SRAM Bank3, please refer to chip's RM for the corresponding SRAM array.

enumerator kCMC_SRAMBank4
Power off SRAM Bank4, please refer to chip's RM for the corresponding SRAM array.

enumerator kCMC_SRAMBank5
Power off SRAM Bank5, please refer to chip's RM for the corresponding SRAM array.

enumerator kCMC_SRAMBank6
Power off SRAM Bank6, please refer to chip's RM for the corresponding SRAM array.

enumerator kCMC_SRAMBank7
Power off SRAM Bank7, please refer to chip's RM for the corresponding SRAM array.

enumerator kCMC_SRAMBank8
Power off SRAM Bank8, please refer to chip's RM for the corresponding SRAM array.

enumerator kCMC_SRAMBank9
Power off SRAM Bank9, please refer to chip's RM for the corresponding SRAM array.

enumerator kCMC_SRAMBank10
Power off SRAM Bank10, please refer to chip's RM for the corresponding SRAM array.

enumerator kCMC_AllSramArrays
Mask of all system SRAM arrays.

enum _cmc_system_reset_sources
System reset sources enumeration.
Values:

enumerator kCMC_WakeUpReset
The reset caused by a wakeup from Power Down or Deep Power Down mode.

enumerator kCMC_PORReset
The reset caused by power on reset detection logic.

enumerator kCMC_LVDReset
The reset caused by a Low Voltage Detect.

enumerator kCMC_HVDReset
The reset caused by a High voltage Detect.

enumerator kCMC_WarmReset
The last reset source is a warm reset source.

enumerator kCMC_FatalReset
The last reset source is a fatal reset source.

enumerator kCMC_PinReset
The reset caused by the RESET_b pin.

enumerator kCMC_DAPReset
The reset caused by a reset request from the Debug Access port.

enumerator kCMC_ResetTimeout
The reset caused by a timeout or other error condition in the system reset generation.

enumerator kCMC_LowPowerAcknowledgeTimeoutReset
The reset caused by a timeout in low power mode entry logic.

enumerator kCMC_SCGReset
The reset caused by a loss of clock or loss of lock event in the SCG.

enumerator kCMC_Watchdog0Reset
The reset caused by a WatchDog 0 timeout.

enumerator kCMC_SoftwareReset
The reset caused by a software reset request.

enumerator kCMC_LockUpReset
The reset caused by the ARM core indication of a LOCKUP event.

enumerator kCMC_Watchdog1Reset
The reset caused by a WatchDog 1 timeout.

enum _cmc_core_clock_gate_status
Indicate the core clock was gated.
Values:

enumerator kCMC_CoreClockNotGated
Core clock not gated.

enumerator kCMC_CoreClockGated
Core clock was gated due to low power mode entry.

enum _cmc_clock_mode
CMC clock mode enumeration.
Values:

enumerator kCMC_GateNoneClock
No clock gating.

enumerator kCMC_GateCoreClock
Gate Core clock.

enumerator kCMC_GateCorePlatformClock
Gate Core clock and platform clock.

enumerator kCMC_GateAllSystemClocks
Gate all System clocks, without getting core entering into low power mode.

enumerator kCMC_GateAllSystemClocksEnterLowPowerMode
Gate all System clocks, with core entering into low power mode.

enum _cmc_low_power_mode
CMC power mode enumeration.
Values:

enumerator kCMC_ActiveMode
Select Active mode.

enumerator kCMC_SleepMode
Select Sleep mode when a core executes WFI or WFE instruction.

enumerator kCMC_DeepSleepMode
Select Deep Sleep mode when a core executes WFI or WFE instruction.

enumerator kCMC_PowerDownMode
Select Power Down mode when a core executes WFI or WFE instruction.

enumerator kCMC_DeepPowerDown
Select Deep Power Down mode when a core executes WFI or WFE instruction.

typedef enum *_cmc_core_clock_gate_status* cmc_core_clock_gate_status_t
Indicate the core clock was gated.

typedef enum *_cmc_clock_mode* cmc_clock_mode_t
CMC clock mode enumeration.

typedef enum *_cmc_low_power_mode* cmc_low_power_mode_t
CMC power mode enumeration.

typedef struct *_cmc_reset_pin_config* cmc_reset_pin_config_t
CMC reset pin configuration.

typedef struct *_cmc_power_domain_config* cmc_power_domain_config_t
power mode configuration for each power domain.

CMC_BLR_LOCK_FIELD_WIDTH

CMC_BLR_LOCK_IDX_MASK(index)

CMC_BLR_LOCK_IDX_SHIFT(index)

CMC_BLR_LOCK_IDX(index, value)

struct _cmc_reset_pin_config
#include <fsl_cmc.h> CMC reset pin configuration.

Public Members

bool lowpowerFilterEnable
: Low Power Filter enable.

bool resetFilterEnable
: Reset Filter enable.

uint8_t resetFilterWidth
: Width of the Reset Filter.

struct __cmc_power_domain_config
: *#include <fsl_cmc.h>* power mode configuration for each power domain.

Public Members

cmc_clock_mode_t clock_mode
: Clock mode for each power domain.

cmc_low_power_mode_t main_domain
: The low power mode of the MAIN power domain.

cmc_low_power_mode_t wake_domain
: The low power mode of the WAKE power domain.

2.11 CRC: Cyclic Redundancy Check Driver

FSL_CRC_DRIVER_VERSION
: CRC driver version. Version 2.1.0.

Current version: 2.1.0

Change log:

- Version 2.1.0
  - Choosing CRC clocks from CRC clock array according to instance instead of hard-coded value.
- Version 2.0.5
  - Fix CERT-C issue with boolean-to-unsigned integer conversion.
- Version 2.0.4
  - Release peripheral from reset if necessary in init function.
- Version 2.0.3
  - Fix MISRA issues
- Version 2.0.2
  - Fix MISRA issues

- Version 2.0.1
  - move DATA and DATALL macro definition from header file to source file

enum _crc_bits

CRC bit width.

Values:

enumerator kCrcBits16

Generate 16-bit CRC code

enumerator kCrcBits32

Generate 32-bit CRC code

enum _crc_result

CRC result type.

Values:

enumerator kCrcFinalChecksum

CRC data register read value is the final checksum. Reflect out and final xor protocol features are applied.

enumerator kCrcIntermediateChecksum

CRC data register read value is intermediate checksum (raw value). Reflect out and final xor protocol feature are not applied. Intermediate checksum can be used as a seed for *CRC_Init()* to continue adding data to this checksum.

typedef enum *_crc_bits* crc_bits_t

CRC bit width.

typedef enum *_crc_result* crc_result_t

CRC result type.

typedef struct *_crc_config* crc_config_t

CRC protocol configuration.

This structure holds the configuration for the CRC protocol.

void CRC_Init(CRC_Type *base, const *crc_config_t* *config)

Enables and configures the CRC peripheral module.

This function enables the clock gate in the SIM module for the CRC peripheral. It also configures the CRC module and starts a checksum computation by writing the seed.

Parameters

- base – CRC peripheral address.
- config – CRC module configuration structure.

void CRC_Deinit(CRC_Type *base)

Disables the CRC peripheral module.

This function disables the clock gate in the SIM module for the CRC peripheral.

Parameters

- base – CRC peripheral address.

void CRC_GetDefaultConfig(*crc_config_t* *config)

Loads default values to the CRC protocol configuration structure.

Loads default values to the CRC protocol configuration structure. The default values are as follows.

```
config->polynomial = 0x1021;
config->seed = 0xFFFF;
config->reflectIn = false;
config->reflectOut = false;
config->complementChecksum = false;
config->crcBits = kCrcBits16;
config->crcResult = kCrcFinalChecksum;
```

Parameters

- config – CRC protocol configuration structure.

void CRC_WriteData(CRC_Type *base, const uint8_t *data, size_t dataSize)

Writes data to the CRC module.

Writes input data buffer bytes to the CRC data register. The configured type of transpose is applied.

Parameters

- base – CRC peripheral address.
- data – Input data stream, MSByte in data[0].
- dataSize – Size in bytes of the input data buffer.

uint32_t CRC_Get32bitResult(CRC_Type *base)

Reads the 32-bit checksum from the CRC module.

Reads the CRC data register (either an intermediate or the final checksum). The configured type of transpose and complement is applied.

Parameters

- base – CRC peripheral address.

Returns

An intermediate or the final 32-bit checksum, after configured transpose and complement operations.

uint16_t CRC_Get16bitResult(CRC_Type *base)

Reads a 16-bit checksum from the CRC module.

Reads the CRC data register (either an intermediate or the final checksum). The configured type of transpose and complement is applied.

Parameters

- base – CRC peripheral address.

Returns

An intermediate or the final 16-bit checksum, after configured transpose and complement operations.

CRC_DRIVER_USE_CRC16_CCIT_FALSE_AS_DEFAULT

Default configuration structure filled by *CRC_GetDefaultConfig()*. Use CRC16-CCIT-FALSE as defeault.

struct __crc__config

#include <fsl_crc.h> CRC protocol configuration.

This structure holds the configuration for the CRC protocol.

Public Members

uint32_t polynomial

CRC Polynomial, MSBit first. Example polynomial: 0x1021 = 1_0000_0010_0001 = x^12+x^5+1

uint32_t seed

Starting checksum value

bool reflectIn

Reflect bits on input.

bool reflectOut

Reflect bits on output.

bool complementChecksum

True if the result shall be complement of the actual checksum.

crc_bits_t crcBits

Selects 16- or 32- bit CRC protocol.

crc_result_t crcResult

Selects final or intermediate checksum return from *CRC_Get16bitResult()* or *CRC_Get32bitResult()*

2.12 EDMA: Enhanced Direct Memory Access (eDMA) Controller Driver

void EDMA_Init(DMA_Type *base, const *edma_config_t* *config)

Initializes the eDMA peripheral.

This function ungates the eDMA clock and configures the eDMA peripheral according to the configuration structure.

Note: This function enables the minor loop map feature.

Parameters

- base – eDMA peripheral base address.
- config – A pointer to the configuration structure, see "edma_config_t".

void EDMA_Deinit(DMA_Type *base)

Deinitializes the eDMA peripheral.

This function gates the eDMA clock.

Parameters

- base – eDMA peripheral base address.

void EDMA_InstallTCD(DMA_Type *base, uint32_t channel, *edma_tcd_t* *tcd)

Push content of TCD structure into hardware TCD register.

Parameters

- base – EDMA peripheral base address.
- channel – EDMA channel number.
- tcd – Point to TCD structure.

void EDMA_GetDefaultConfig(*edma_config_t* *config)

Gets the eDMA default configuration structure.

This function sets the configuration structure to default values. The default configuration is set to the following values:

```
config.enableMasterIdReplication = true;
config.enableHaltOnError = true;
config.enableRoundRobinArbitration = false;
config.enableDebugMode = false;
config.enableBufferedWrites = false;
```

Parameters

- config – A pointer to the eDMA configuration structure.

static inline void EDMA_EnableAllChannelLink(DMA_Type *base, bool enable)

Enables/disables all channel linking.

This function enables/disables all channel linking in the management page. For specific channel linking enablement & configuration, please refer to EDMA_SetChannelLink and EDMA_TcdSetChannelLink APIs.

For example, to disable all channel linking in the DMA0 management page:

```
EDMA_EnableAllChannelLink(DMA0, false);
```

Parameters

- base – eDMA peripheral base address.
- enable – Switcher of the channel linking feature for all channels. "true" means to enable. "false" means not.

void EDMA_ResetChannel(DMA_Type *base, uint32_t channel)

Sets all TCD registers to default values.

This function sets TCD registers for this channel to default values.

Note: This function must not be called while the channel transfer is ongoing or it causes unpredictable results.

Note: This function enables the auto stop request feature.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.

void EDMA_SetTransferConfig(DMA_Type *base, uint32_t channel, const *edma_transfer_config_t* *config, *edma_tcd_t* *nextTcd)

Configures the eDMA transfer attribute.

This function configures the transfer attribute, including source address, destination address, transfer size, address offset, and so on. It also configures the scatter gather feature if the user supplies the TCD address. Example:

```
edma_transfer_config_t config;
edma_tcd_t tcd;
config.srcAddr = ..;
config.destAddr = ..;
...
EDMA_SetTransferConfig(DMA0, channel, &config, &stcd);
```

Note: If nextTcd is not NULL, it means scatter gather feature is enabled and DREQ bit is cleared in the previous transfer configuration, which is set in the eDMA_ResetChannel.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- config – Pointer to eDMA transfer configuration structure.
- nextTcd – Point to TCD structure. It can be NULL if users do not want to enable scatter/gather feature.

void EDMA_SetMinorOffsetConfig(DMA_Type *base, uint32_t channel, const *edma_minor_offset_config_t* *config)

Configures the eDMA minor offset feature.

The minor offset means that the signed-extended value is added to the source address or destination address after each minor loop.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- config – A pointer to the minor offset configuration structure.

static inline void EDMA_SetChannelArbitrationGroup(DMA_Type *base, uint32_t channel, uint32_t group)

Configures the eDMA channel arbitration group.

This function configures the channel arbitration group. The arbitration group priorities are evaluated by numeric value from highest group number to lowest.

Parameters

- base – eDMA peripheral base address.

- channel – eDMA channel number
- group – Fixed-priority arbitration group number for the channel.

static inline void EDMA_SetChannelPreemptionConfig(DMA_Type *base, uint32_t channel, const *edma_channel_Preemption_config_t* *config)

Configures the eDMA channel preemption feature.

This function configures the channel preemption attribute and the priority of the channel.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number
- config – A pointer to the channel preemption configuration structure.

static inline uint32_t EDMA_GetChannelSystemBusInformation(DMA_Type *base, uint32_t channel)

Gets the eDMA channel identification and attribute information on the system bus interface.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.

Returns

The mask of the channel system bus information. Users need to use the _edma_channel_sys_bus_info type to decode the return variables.

static inline void EDMA_EnableChannelMasterIDReplication(DMA_Type *base, uint32_t channel, bool enable)

Set channel master ID replication.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- enable – true is enable, false is disable.

static inline void EDMA_SetChannelSecurityLevel(DMA_Type *base, uint32_t channel, *edma_channel_security_level_t* level)

Set channel security level.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- level – security level.

static inline void EDMA_SetChannelProtectionLevel(DMA_Type *base, uint32_t channel, *edma_channel_protection_level_t* level)

Set channel protection level.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- level – protection level.

void EDMA_SetChannelLink(DMA_Type *base, uint32_t channel, *edma_channel_link_type_t* type, uint32_t linkedChannel)

Sets the channel link for the eDMA transfer.

This function configures either the minor link or the major link mode. The minor link means that the channel link is triggered every time CITER decreases by 1. The major link means that the channel link is triggered when the CITER is exhausted.

Note: Users should ensure that DONE flag is cleared before calling this interface, or the configuration is invalid.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- type – A channel link type, which can be one of the following:
  - kEDMA_LinkNone
  - kEDMA_MinorLink
  - kEDMA_MajorLink
- linkedChannel – The linked channel number.

void EDMA_SetBandWidth(DMA_Type *base, uint32_t channel, *edma_bandwidth_t* bandWidth)

Sets the bandwidth for the eDMA transfer.

Because the eDMA processes the minor loop, it continuously generates read/write sequences until the minor count is exhausted. The bandwidth forces the eDMA to stall after the completion of each read/write access to control the bus request bandwidth seen by the crossbar switch.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- bandWidth – A bandwidth setting, which can be one of the following:
  - kEDMABandwidthStallNone
  - kEDMABandwidthStall4Cycle
  - kEDMABandwidthStall8Cycle

void EDMA_SetModulo(DMA_Type *base, uint32_t channel, *edma_modulo_t* srcModulo, *edma_modulo_t* destModulo)

Sets the source modulo and the destination modulo for the eDMA transfer.

This function defines a specific address range specified to be the value after (SADDR + SOFF)/(DADDR + DOFF) calculation is performed or the original register value. It provides the ability to implement a circular data queue easily.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- srcModulo – A source modulo value.
- destModulo – A destination modulo value.

static inline void EDMA_EnableAsyncRequest(DMA_Type *base, uint32_t channel, bool enable)

Enables an async request for the eDMA transfer.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- enable – The command to enable (true) or disable (false).

static inline void EDMA_EnableAutoStopRequest(DMA_Type *base, uint32_t channel, bool enable)

Enables an auto stop request for the eDMA transfer.

If enabling the auto stop request, the eDMA hardware automatically disables the hardware channel request.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- enable – The command to enable (true) or disable (false).

void EDMA_EnableChannelInterrupts(DMA_Type *base, uint32_t channel, uint32_t mask)

Enables the interrupt source for the eDMA transfer.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- mask – The mask of interrupt source to be set. Users need to use the defined edma_interrupt_enable_t type.

void EDMA_DisableChannelInterrupts(DMA_Type *base, uint32_t channel, uint32_t mask)

Disables the interrupt source for the eDMA transfer.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- mask – The mask of the interrupt source to be set. Use the defined edma_interrupt_enable_t type.

static inline void EDMA_SetChannelMux(DMA_Type *base, uint32_t channel, uint32_t mux)

Set channel mux source.

Note:When the peripheral is no longer needed, the mux configuration for that channel should be written to 0, thus releasing the resource.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- mux – the mux source value is SOC specific, please reference the SOC for detail.

void EDMA_TcdReset(*edma_tcd_t* *tcd)

Sets all fields to default values for the TCD structure.

This function sets all fields for this TCD structure to default value.

Note: This function enables the auto stop request feature.

Parameters

- tcd – Pointer to the TCD structure.

void EDMA_TcdSetTransferConfig(*edma_tcd_t* *tcd, const *edma_transfer_config_t* *config, *edma_tcd_t* *nextTcd)

Configures the eDMA TCD transfer attribute.

The TCD is a transfer control descriptor. The content of the TCD is the same as the hardware TCD registers. The STCD is used in the scatter-gather mode. This function configures the TCD transfer attribute, including source address, destination address, transfer size, address offset, and so on. It also configures the scatter gather feature if the user supplies the next TCD address. Example:

```
edma_transfer_config_t config = {
...
}
edma_tcd_t tcd __aligned(32);
edma_tcd_t nextTcd __aligned(32);
EDMA_TcdSetTransferConfig(&tcd, &config, &nextTcd);
```

Note: TCD address should be 32 bytes aligned or it causes an eDMA error.

Note: If the nextTcd is not NULL, the scatter gather feature is enabled and DREQ bit is cleared in the previous transfer configuration, which is set in the EDMA_TcdReset.

Parameters

- tcd – Pointer to the TCD structure.
- config – Pointer to eDMA transfer configuration structure.
- nextTcd – Pointer to the next TCD structure. It can be NULL if users do not want to enable scatter/gather feature.

void EDMA_TcdSetMinorOffsetConfig(*edma_tcd_t* *tcd, const *edma_minor_offset_config_t* *config)

Configures the eDMA TCD minor offset feature.

A minor offset is a signed-extended value added to the source address or a destination address after each minor loop.

Parameters

- tcd – A point to the TCD structure.
- config – A pointer to the minor offset configuration structure.

void EDMA_TcdSetChannelLink(*edma_tcd_t* *tcd, *edma_channel_link_type_t* type, uint32_t linkedChannel)

Sets the channel link for the eDMA TCD.

This function configures either a minor link or a major link. The minor link means the channel link is triggered every time CITER decreases by 1. The major link means that the channel link is triggered when the CITER is exhausted.

Note: Users should ensure that DONE flag is cleared before calling this interface, or the configuration is invalid.

Parameters

- tcd – Point to the TCD structure.
- type – Channel link type, it can be one of:
  - kEDMA_LinkNone
  - kEDMA_MinorLink
  - kEDMA_MajorLink
- linkedChannel – The linked channel number.

static inline void EDMA_TcdSetBandWidth(*edma_tcd_t* *tcd, *edma_bandwidth_t* bandWidth)

Sets the bandwidth for the eDMA TCD.

Because the eDMA processes the minor loop, it continuously generates read/write sequences until the minor count is exhausted. The bandwidth forces the eDMA to stall after the completion of each read/write access to control the bus request bandwidth seen by the crossbar switch.

Parameters

- tcd – A pointer to the TCD structure.
- bandWidth – A bandwidth setting, which can be one of the following:
  - kEDMABandwidthStallNone
  - kEDMABandwidthStall4Cycle
  - kEDMABandwidthStall8Cycle

void EDMA_TcdSetModulo(*edma_tcd_t* *tcd, *edma_modulo_t* srcModulo, *edma_modulo_t* destModulo)

Sets the source modulo and the destination modulo for the eDMA TCD.

This function defines a specific address range specified to be the value after (SADDR + SOFF)/(DADDR + DOFF) calculation is performed or the original register value. It provides the ability to implement a circular data queue easily.

Parameters

- tcd – A pointer to the TCD structure.
- srcModulo – A source modulo value.
- destModulo – A destination modulo value.

static inline void EDMA_TcdEnableAutoStopRequest(*edma_tcd_t* *tcd, bool enable)

Sets the auto stop request for the eDMA TCD.

If enabling the auto stop request, the eDMA hardware automatically disables the hardware channel request.

Parameters

- tcd – A pointer to the TCD structure.
- enable – The command to enable (true) or disable (false).

void EDMA_TcdEnableInterrupts(*edma_tcd_t* *tcd, uint32_t mask)

Enables the interrupt source for the eDMA TCD.

Parameters

- tcd – Point to the TCD structure.
- mask – The mask of interrupt source to be set. Users need to use the defined edma_interrupt_enable_t type.

void EDMA_TcdDisableInterrupts(*edma_tcd_t* *tcd, uint32_t mask)

Disables the interrupt source for the eDMA TCD.

Parameters

- tcd – Point to the TCD structure.
- mask – The mask of interrupt source to be set. Users need to use the defined edma_interrupt_enable_t type.

static inline void EDMA_EnableChannelRequest(DMA_Type *base, uint32_t channel)

Enables the eDMA hardware channel request.

This function enables the hardware channel request.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.

static inline void EDMA_DisableChannelRequest(DMA_Type *base, uint32_t channel)

Disables the eDMA hardware channel request.

This function disables the hardware channel request.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.

static inline void EDMA_TriggerChannelStart(DMA_Type *base, uint32_t channel)

Starts the eDMA transfer by using the software trigger.

This function starts a minor loop transfer.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.

uint32_t EDMA_GetRemainingMajorLoopCount(DMA_Type *base, uint32_t channel)

Gets the Remaining major loop count from the eDMA current channel TCD.

This function checks the TCD (Task Control Descriptor) status for a specified eDMA channel and returns the number of major loop count that has not finished.

Note: 1. This function can only be used to get unfinished major loop count of transfer without the next TCD, or it might be inaccuracy.

a. The unfinished/remaining transfer bytes cannot be obtained directly from registers while the channel is running. Because to calculate the remaining bytes, the initial NBYTES configured in DMA_TCDn_NBYTES_MLNO register is needed while the eDMA IP does not support getting it while a channel is active. In another word, the NBYTES value reading is always the actual (decrementing) NBYTES value the dma_engine is working with while a channel is running. Consequently, to get the remaining transfer bytes, a software-saved initial value of NBYTES (for example copied before enabling

the channel) is needed. The formula to calculate it is shown below: RemainingBytes = RemainingMajorLoopCount * NBYTES(initially configured)

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.

Returns
Major loop count which has not been transferred yet for the current TCD.

static inline uint32_t EDMA_GetErrorStatusFlags(DMA_Type *base)

Gets the eDMA channel error status flags.

Parameters

- base – eDMA peripheral base address.

Returns
The mask of error status flags. Users need to use the _edma_error_status_flags type to decode the return variables.

uint32_t EDMA_GetChannelStatusFlags(DMA_Type *base, uint32_t channel)

Gets the eDMA channel status flags.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.

Returns
The mask of channel status flags. Users need to use the _edma_channel_status_flags type to decode the return variables.

void EDMA_ClearChannelStatusFlags(DMA_Type *base, uint32_t channel, uint32_t mask)

Clears the eDMA channel status flags.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- mask – The mask of channel status to be cleared. Users need to use the defined _edma_channel_status_flags type.

void EDMA_CreateHandle(*edma_handle_t* *handle, DMA_Type *base, uint32_t channel)

Creates the eDMA handle.

This function is called if using the transactional API for eDMA. This function initializes the internal state of the eDMA handle.

Parameters

- handle – eDMA handle pointer. The eDMA handle stores callback function and parameters.
- base – eDMA peripheral base address.
- channel – eDMA channel number.

void EDMA_InstallTCDMemory(*edma_handle_t* *handle, *edma_tcd_t* *tcdPool, uint32_t tcdSize)

Installs the TCDs memory pool into the eDMA handle.

This function is called after the EDMA_CreateHandle to use scatter/gather feature.

Parameters

- handle – eDMA handle pointer.
- tcdPool – A memory pool to store TCDs. It must be 32 bytes aligned.
- tcdSize – The number of TCD slots.

void EDMA_SetCallback(*edma_handle_t* *handle, *edma_callback* callback, void *userData)

Installs a callback function for the eDMA transfer.

This callback is called in the eDMA IRQ handler. Use the callback to do something after the current major loop transfer completes.

Parameters

- handle – eDMA handle pointer.
- callback – eDMA callback function pointer.
- userData – A parameter for the callback function.

void EDMA_PrepareTransferConfig(*edma_transfer_config_t* *config, void *srcAddr, uint32_t srcWidth, int16_t srcOffset, void *destAddr, uint32_t destWidth, int16_t destOffset, uint32_t bytesEachRequest, uint32_t transferBytes)

Prepares the eDMA transfer structure configurations.

This function prepares the transfer configuration structure according to the user input.

Note: The data address and the data width must be consistent. For example, if the SRC is 4 bytes, the source address must be 4 bytes aligned, or it results in source address error (SAE).

Parameters

- config – The user configuration structure of type edma_transfer_config_t.
- srcAddr – eDMA transfer source address.
- srcWidth – eDMA transfer source address width(bytes).
- srcOffset – eDMA transfer source address offset
- destAddr – eDMA transfer destination address.
- destWidth – eDMA transfer destination address width(bytes).
- destOffset – eDMA transfer destination address offset
- bytesEachRequest – eDMA transfer bytes per channel request.
- transferBytes – eDMA transfer bytes to be transferred.

void EDMA_PrepareTransfer(*edma_transfer_config_t* *config, void *srcAddr, uint32_t srcWidth, void *destAddr, uint32_t destWidth, uint32_t bytesEachRequest, uint32_t transferBytes, *edma_transfer_type_t* transferType)

Prepares the eDMA transfer structure.

This function prepares the transfer configuration structure according to the user input.

Note: The data address and the data width must be consistent. For example, if the SRC is 4 bytes, the source address must be 4 bytes aligned, or it results in source address error (SAE).

Parameters

- config – The user configuration structure of type edma_transfer_config_t.
- srcAddr – eDMA transfer source address.
- srcWidth – eDMA transfer source address width(bytes).
- destAddr – eDMA transfer destination address.
- destWidth – eDMA transfer destination address width(bytes).
- bytesEachRequest – eDMA transfer bytes per channel request.
- transferBytes – eDMA transfer bytes to be transferred.
- transferType – eDMA transfer type.

status_t EDMA_SubmitTransfer(*edma_handle_t* *handle, const *edma_transfer_config_t* *config)

Submits the eDMA transfer request.

This function submits the eDMA transfer request according to the transfer configuration structure. If submitting the transfer request repeatedly, this function packs an unprocessed request as a TCD and enables scatter/gather feature to process it in the next time.

Parameters

- handle – eDMA handle pointer.
- config – Pointer to eDMA transfer configuration structure.

Return values

- kStatus_EDMA_Success – It means submit transfer request succeed.
- kStatus_EDMA_QueueFull – It means TCD queue is full. Submit transfer request is not allowed.
- kStatus_EDMA_Busy – It means the given channel is busy, need to submit request later.

void EDMA_StartTransfer(*edma_handle_t* *handle)

eDMA starts transfer.

This function enables the channel request. Users can call this function after submitting the transfer request or before submitting the transfer request.

Parameters

- handle – eDMA handle pointer.

void EDMA_StopTransfer(*edma_handle_t* *handle)

eDMA stops transfer.

This function disables the channel request to pause the transfer. Users can call *EDMA_StartTransfer()* again to resume the transfer.

Parameters

- handle – eDMA handle pointer.

void EDMA_AbortTransfer(*edma_handle_t* *handle)

eDMA aborts transfer.

This function disables the channel request and clear transfer status bits. Users can submit another transfer after calling this API.

Parameters

- handle – DMA handle pointer.

static inline uint32_t EDMA_GetUnusedTCDNumber(*edma_handle_t* *handle)

Get unused TCD slot number.

This function gets current tcd index which is run. If the TCD pool pointer is NULL, it will return 0.

Parameters

- handle – DMA handle pointer.

Returns
The unused tcd slot number.

static inline uint32_t EDMA_GetNextTCDAddress(*edma_handle_t* *handle)

Get the next tcd address.

This function gets the next tcd address. If this is last TCD, return 0.

Parameters

- handle – DMA handle pointer.

Returns
The next TCD address.

static inline *edma_transfer_size_t* EDMA_GetTransferSize(uint32_t width)

Get the transfer size.

This function gets the transfer size.

Parameters

- width – transfer width(bytes).

Returns
The transfer size.

void EDMA_HandleIRQ(*edma_handle_t* *handle)

eDMA IRQ handler for the current major loop transfer completion.

This function clears the channel major interrupt flag and calls the callback function if it is not NULL.

Parameters

- handle – eDMA handle pointer.

FSL_EDMA_DRIVER_VERSION

eDMA driver version

Version 2.6.1.

enum _edma_transfer_size

eDMA transfer configuration

Values:

enumerator kEDMA_TransferSize1Bytes

Source/Destination data transfer size is 1 byte every time

enumerator kEDMA_TransferSize2Bytes

Source/Destination data transfer size is 2 bytes every time

enumerator kEDMA_TransferSize4Bytes
Source/Destination data transfer size is 4 bytes every time

enumerator kEDMA_TransferSize8Bytes
Source/Destination data transfer size is 8 bytes every time

enumerator kEDMA_TransferSize16Bytes
Source/Destination data transfer size is 16 bytes every time

enumerator kEDMA_TransferSize32Bytes
Source/Destination data transfer size is 32 bytes every time

enumerator kEDMA_TransferSize64Bytes
Source/Destination data transfer size is 64 bytes every time

enum _edma_modulo
eDMA modulo configuration

Values:

enumerator kEDMA_ModuloDisable
Disable modulo

enumerator kEDMA_Modulo2bytes
Circular buffer size is 2 bytes.

enumerator kEDMA_Modulo4bytes
Circular buffer size is 4 bytes.

enumerator kEDMA_Modulo8bytes
Circular buffer size is 8 bytes.

enumerator kEDMA_Modulo16bytes
Circular buffer size is 16 bytes.

enumerator kEDMA_Modulo32bytes
Circular buffer size is 32 bytes.

enumerator kEDMA_Modulo64bytes
Circular buffer size is 64 bytes.

enumerator kEDMA_Modulo128bytes
Circular buffer size is 128 bytes.

enumerator kEDMA_Modulo256bytes
Circular buffer size is 256 bytes.

enumerator kEDMA_Modulo512bytes
Circular buffer size is 512 bytes.

enumerator kEDMA_Modulo1Kbytes
Circular buffer size is 1 K bytes.

enumerator kEDMA_Modulo2Kbytes
Circular buffer size is 2 K bytes.

enumerator kEDMA_Modulo4Kbytes
Circular buffer size is 4 K bytes.

enumerator kEDMA_Modulo8Kbytes
Circular buffer size is 8 K bytes.

enumerator kEDMA_Modulo16Kbytes
Circular buffer size is 16 K bytes.

enumerator kEDMA_Modulo32Kbytes
Circular buffer size is 32 K bytes.

enumerator kEDMA_Modulo64Kbytes
Circular buffer size is 64 K bytes.

enumerator kEDMA_Modulo128Kbytes
Circular buffer size is 128 K bytes.

enumerator kEDMA_Modulo256Kbytes
Circular buffer size is 256 K bytes.

enumerator kEDMA_Modulo512Kbytes
Circular buffer size is 512 K bytes.

enumerator kEDMA_Modulo1Mbytes
Circular buffer size is 1 M bytes.

enumerator kEDMA_Modulo2Mbytes
Circular buffer size is 2 M bytes.

enumerator kEDMA_Modulo4Mbytes
Circular buffer size is 4 M bytes.

enumerator kEDMA_Modulo8Mbytes
Circular buffer size is 8 M bytes.

enumerator kEDMA_Modulo16Mbytes
Circular buffer size is 16 M bytes.

enumerator kEDMA_Modulo32Mbytes
Circular buffer size is 32 M bytes.

enumerator kEDMA_Modulo64Mbytes
Circular buffer size is 64 M bytes.

enumerator kEDMA_Modulo128Mbytes
Circular buffer size is 128 M bytes.

enumerator kEDMA_Modulo256Mbytes
Circular buffer size is 256 M bytes.

enumerator kEDMA_Modulo512Mbytes
Circular buffer size is 512 M bytes.

enumerator kEDMA_Modulo1Gbytes
Circular buffer size is 1 G bytes.

enumerator kEDMA_Modulo2Gbytes
Circular buffer size is 2 G bytes.

enum _edma_bandwidth
Bandwidth control.
Values:

enumerator kEDMA_BandwidthStallNone
No eDMA engine stalls.

enumerator kEDMA_BandwidthStall4Cycle
eDMA engine stalls for 4 cycles after each read/write.

enumerator kEDMA_BandwidthStall8Cycle
eDMA engine stalls for 8 cycles after each read/write.

enum _edma_channel_link_type
Channel link type.
Values:

enumerator kEDMA_LinkNone
No channel link

enumerator kEDMA_MinorLink
Channel link after each minor loop

enumerator kEDMA_MajorLink
Channel link while major loop count exhausted

enum _edma_channel_security_level
eDMA channel security level
Values:

enumerator kEDMA_ChannelSecurityLevelNonSecure
Channel executes in non-secure mode

enumerator kEDMA_ChannelSecurityLevelSecure
Channel executes in secure mode

enum _edma_channel_protection_level
eDMA channel protection level
Values:

enumerator kEDMA_ChannelProtectionLevelUser
Channel executes at user privilege level

enumerator kEDMA_ChannelProtectionLevelPrivileged
Channel executes at privileged level

eDMA channel status flags, _edma_channel_status_flags
Values:

enumerator kEDMA_DoneFlag
DONE flag, set while transfer finished, CITER value exhausted

enumerator kEDMA_ErrorFlag
eDMA error flag, an error occurred in a transfer

enumerator kEDMA_InterruptFlag
eDMA interrupt flag, set while an interrupt occurred of this channel

eDMA channel error status flags, _edma_error_status_flags
Values:

enumerator kEDMA_DestinationBusErrorFlag
Bus error on destination address

enumerator kEDMA_SourceBusErrorFlag
Bus error on the source address

enumerator kEDMA_ScatterGatherErrorFlag
Error on the Scatter/Gather address, not 32byte aligned.

enumerator kEDMA_NbytesErrorFlag
NBYTES/CITER configuration error

enumerator kEDMA_DestinationOffsetErrorFlag
Destination offset not aligned with destination size

enumerator kEDMA_DestinationAddressErrorFlag
Destination address not aligned with destination size

enumerator kEDMA_SourceOffsetErrorFlag
Source offset not aligned with source size

enumerator kEDMA_SourceAddressErrorFlag
Source address not aligned with source size

enumerator kEDMA_TransferCanceledFlag
Transfer cancelled

enumerator kEDMA_ErrorChannelFlag
Error channel number of the cancelled channel number

enumerator kEDMA_ValidFlag
No error occurred, this bit is 0. Otherwise, it is 1.

eDMA channel system bus information, _edma_channel_sys_bus_info

Values:

enumerator kEDMA_AttributeOutput
DMA's AHB system bus attribute output value.

enumerator kEDMA_PrivilegedAccessLevel
Privileged Access Level for DMA transfers. 0b - User protection level; 1b - Privileged protection level.

enumerator kEDMA_MasterId
DMA's master ID when channel is active and master ID replication is enabled.

enum _edma_interrupt_enable
eDMA interrupt source

Values:

enumerator kEDMA_ErrorInterruptEnable
Enable interrupt while channel error occurs.

enumerator kEDMA_MajorInterruptEnable
Enable interrupt while major count exhausted.

enumerator kEDMA_HalfInterruptEnable
Enable interrupt while major count to half value.

enum _edma_transfer_type
eDMA transfer type

Values:

enumerator kEDMA_MemoryToMemory
Transfer from memory to memory

enumerator kEDMA_PeripheralToMemory
Transfer from peripheral to memory

enumerator kEDMA_MemoryToPeripheral
Transfer from memory to peripheral

enumerator kEDMA_PeripheralToPeripheral
Transfer from Peripheral to peripheral

eDMA transfer status, _edma_transfer_status

Values:

enumerator kStatus_EDMA_QueueFull
TCD queue is full.

enumerator kStatus_EDMA_Busy
Channel is busy and can't handle the transfer request.

typedef enum *_edma_transfer_size* edma_transfer_size_t
eDMA transfer configuration

typedef enum *_edma_modulo* edma_modulo_t
eDMA modulo configuration

typedef enum *_edma_bandwidth* edma_bandwidth_t
Bandwidth control.

typedef enum *_edma_channel_link_type* edma_channel_link_type_t
Channel link type.

typedef enum *_edma_channel_security_level* edma_channel_security_level_t
eDMA channel security level

typedef enum *_edma_channel_protection_level* edma_channel_protection_level_t
eDMA channel protection level

typedef enum *_edma_interrupt_enable* edma_interrupt_enable_t
eDMA interrupt source

typedef enum *_edma_transfer_type* edma_transfer_type_t
eDMA transfer type

typedef struct *_edma_config* edma_config_t
eDMA global configuration structure.

typedef struct *_edma_transfer_config* edma_transfer_config_t

eDMA transfer configuration

This structure configures the source/destination transfer attribute.

typedef struct *_edma_channel_Preemption_config* edma_channel_Preemption_config_t

eDMA channel priority configuration

typedef struct *_edma_minor_offset_config* edma_minor_offset_config_t

eDMA minor offset configuration

typedef struct *_edma_tcd* edma_tcd_t

eDMA TCD.

This structure is same as TCD register which is described in reference manual, and is used to configure the scatter/gather feature as a next hardware TCD.

typedef void (*edma_callback)(struct *_edma_handle* *handle, void *userData, bool transferDone, uint32_t tcds)

Define callback function for eDMA.

typedef uint32_t (*edma_memorymap_callback)(uint32_t addr)

Memroy map function callback for DMA.

typedef struct *_edma_handle* edma_handle_t

eDMA transfer handle structure

struct _edma_config

#include <fsl_edma.h> eDMA global configuration structure.

Public Members

bool enableMasterIdReplication

Enable (true) master ID replication. If Master ID replication is disabled, the privileged protection level (supervisor mode) for DMA transfers is used.

bool enableHaltOnError

Enable (true) transfer halt on error. Any error causes the HALT bit to set. Subsequently, all service requests are ignored until the HALT bit is cleared.

bool enableRoundRobinArbitration

Enable (true) round robin channel arbitration method or fixed priority arbitration is used for channel selection

bool enableDebugMode

Enable(true) eDMA debug mode. When in debug mode, the eDMA stalls the start of a new channel. Executing channels are allowed to complete.

struct _edma_transfer_config

#include <fsl_edma.h> eDMA transfer configuration

This structure configures the source/destination transfer attribute.

Public Members

uint32_t srcAddr
Source data address.

uint32_t destAddr
Destination data address.

edma_transfer_size_t srcTransferSize
Source data transfer size.

edma_transfer_size_t destTransferSize
Destination data transfer size.

int16_t srcOffset
Sign-extended offset applied to the current source address to form the next-state value as each source read is completed.

int16_t destOffset
Sign-extended offset applied to the current destination address to form the next-state value as each destination write is completed.

uint32_t minorLoopBytes
Bytes to transfer in a minor loop

uint32_t majorLoopCounts
Major loop iteration count.

struct __edma_channel_Preemption_config
#include <fsl_edma.h> eDMA channel priority configuration

Public Members

bool enableChannelPreemption
If true: a channel can be suspended by other channel with higher priority

bool enablePreemptAbility
If true: a channel can suspend other channel with low priority

uint8_t channelPriority
Channel priority

struct __edma_minor_offset_config
#include <fsl_edma.h> eDMA minor offset configuration

Public Members

bool enableSrcMinorOffset
Enable(true) or Disable(false) source minor loop offset.

bool enableDestMinorOffset
Enable(true) or Disable(false) destination minor loop offset.

uint32_t minorOffset
Offset for a minor loop mapping.

struct __edma_tcd
#include <fsl_edma.h> eDMA TCD.

This structure is same as TCD register which is described in reference manual, and is used to configure the scatter/gather feature as a next hardware TCD.

Public Members

__IO uint32_t SADDR
SADDR register, used to save source address

__IO uint16_t SOFF
SOFF register, save offset bytes every transfer

__IO uint16_t ATTR
ATTR register, source/destination transfer size and modulo

__IO uint32_t NBYTES
Nbytes register, minor loop length in bytes

__IO uint32_t SLAST
SLAST register

__IO uint32_t DADDR
DADDR register, used for destination address

__IO uint16_t DOFF
DOFF register, used for destination offset

__IO uint16_t CITER
CITER register, current minor loop numbers, for unfinished minor loop.

__IO uint32_t DLAST_SGA
DLASTSGA register, next stcd address used in scatter-gather mode

__IO uint16_t CSR
CSR register, for TCD control status

__IO uint16_t BITER
BITER register, begin minor loop count.

struct __edma__handle

#include <fsl_edma.h> eDMA transfer handle structure

Public Members

edma_callback callback

Callback function for major count exhausted.

void *userData

Callback function parameter.

DMA_Type *base

eDMA peripheral base address.

edma_tcd_t *tcdPool

Pointer to memory stored TCDs.

uint8_t channel

eDMA channel number.

volatile int8_t header

The first TCD index.

volatile int8_t tail

The last TCD index.

volatile int8_t tcdUsed

The number of used TCD slots.

volatile int8_t tcdSize

The total number of TCD slots in the queue.

uint8_t flags

The status of the current channel.

2.13 ELEMU: Edgelock Messaging unit driver

Values:

enumerator kStatus_ELEMU_AgumentOutOfRange

ELEMU status for out of range access.

enumerator kStatus_ELEMU_InvalidArgument

ELEMU status for invalid argument check.

enumerator kStatus_ELEMU_RequestTimeout
ELEMU status for timeout.

enumerator kStatus_ELEMU_Busy
ELEMU status for reservation by other core.

enum elemu_ownership_status_t
Values:

enumerator kStatus_ELEMU_Unknown
Unexpected ELEMU ownership.

enumerator kStatus_ELEMU_LockedByMe
ELEMU reserved for current core.

enumerator kStatus_ELEMU_LockedByOther
ELEMU reserved for other core.

enumerator kStatus_ELEMU_Free
ELEMU not reserved.

typedef struct *mu_hdr* mu_hdr_t

void ELEMU_mu_hal_send_data(ELEMU_Type *mu, uint8_t regid, uint32_t *data)
Send one word of data to specified transmit register.

This function waits until the transmit register is ready and writes one word of data.

Parameters

- mu – ELEMU peripheral base address
- regid – Transmit register index
- data – Pointer to data to be sent

void ELEMU_mu_hal_receive_data(ELEMU_Type *mu, uint8_t regid, uint32_t *data)
Receive one word of data from specified receive register.

This function waits until data is available in the receive register and reads it.

Parameters

- mu – ELEMU peripheral base address
- regid – Receive register index
- data – Pointer to store received data

status_t ELEMU_mu_hal_receive_data_wait(ELEMU_Type *mu, uint8_t regid, uint32_t *data, uint32_t wait)
Wait for data availability and receive one word from specified register.

Parameters

- mu – ELEMU peripheral base address
- regid – Receive register index
- data – Pointer to store received data
- wait – Maximum wait iterations

Returns
kStatus_Success if successful, kStatus_ELEMU_RequestTimeout if timeout

status_t ELEMU_mu_read_message(ELEMU_Type *mu, uint32_t *buf, uint8_t *size, uint8_t read_header)

Read a message from ELEMU receive registers.

This function reads a message from ELEMU including header and payload.

Parameters

- mu – ELEMU peripheral base address
- buf – Pointer to buffer to store message
- size – Pointer to size variable
- read_header – Flag to indicate if header should be read

Returns
kStatus_Success if successful, kStatus_ELEMU_InvalidArgument if buf or size is NULL

status_t ELEMU_mu_read_data_wait(ELEMU_Type *mu, uint32_t buf[], uint8_t *size, uint32_t wait)

Wait and read data from ELEMU receive registers.

This function waits for data availability and reads the specified number of words.

Parameters

- mu – ELEMU peripheral base address
- buf – Pointer to buffer to store data
- size – Pointer to size variable
- wait – Maximum wait iterations

Returns
kStatus_Success if successful, kStatus_ELEMU_InvalidArgument if buf or size is NULL, kStatus_ELEMU_RequestTimeout if timeout

status_t ELEMU_mu_send_message(ELEMU_Type *mu, uint32_t buf[], size_t wordCount)

Send a message to ELEMU using transmit registers.

This function sends a message buffer to ELEMU by writing words to transmit registers.

Parameters

- mu – ELEMU peripheral base address
- buf – Pointer to message buffer
- wordCount – Number of words to send

Returns
kStatus_Success if successful, kStatus_ELEMU_InvalidArgument if buf is NULL, kStatus_ELEMU_AgumentOutOfRange if wordCount exceeds register count

status_t ELEMU_mu_get_response(ELEMU_Type *mu, uint32_t *buf, size_t wordCount)

Get response message from ELEMU.

This function reads a response message from ELEMU using the read_message function.

Parameters

- mu – ELEMU peripheral base address
- buf – Pointer to buffer to store response

- wordCount – Expected number of words

Returns
kStatus_Success if successful, kStatus_ELEMU_InvalidArgument if buf is NULL, kStatus_ELEMU_AgumentOutOfRange if wordCount exceeds register count

status_t ELEMU_mu_wait_for_data(ELEMU_Type *mu, uint32_t *buf, size_t wordCount, uint32_t wait)

Wait for data availability and read from ELEMU.

This function waits for data and reads the specified number of words.

Parameters

- mu – ELEMU peripheral base address
- buf – Pointer to buffer to store data
- wordCount – Expected number of words
- wait – Maximum wait iterations

Returns
kStatus_Success if successful, kStatus_ELEMU_InvalidArgument if buf is NULL, kStatus_ELEMU_AgumentOutOfRange if wordCount exceeds register count

status_t ELEMU_mu_wait_for_ready(ELEMU_Type *mu, uint32_t wait)

Wait until ELEMU is ready for communication.

This function checks the status register and waits until ELEMU is ready.

Parameters

- mu – ELEMU peripheral base address
- wait – Maximum wait iterations

Returns
kStatus_Success if ready, kStatus_ELEMU_RequestTimeout if timeout

elemu_ownership_status_t ELEMU_mu_get_ownership_status(ELEMU_Type *mu)

Get current ownership status of ELEMU.

Parameters

- mu – ELEMU peripheral base address

Returns
Ownership status enum value

status_t ELEMU_mu_get_ownership(ELEMU_Type *mu)

Attempt to acquire ownership of ELEMU.

This function tries to acquire ownership of ELEMU using the semaphore register.

Parameters

- mu – ELEMU peripheral base address

Returns
kStatus_Success if acquired, kStatus_ELEMU_Busy if locked by other, kStatus_Fail if acquisition failed

status_t ELEMU_mu_release_ownership(ELEMU_Type *mu)

Release ownership of ELEMU.

This function releases ownership of ELEMU using the semaphore release register.

Parameters

- mu – ELEMU peripheral base address

Returns
kStatus_Success if released, kStatus_ELEMU_Busy if locked by other, kStatus_Fail if release failed

status_t ELEMU_mu_release_ownership_force(ELEMU_Type *mu)

Release ownership of ELEMU.

This function releases ownership of ELEMU using the semaphore release register.

Parameters

- mu – ELEMU peripheral base address

Returns
kStatus_Success if released, kStatus_ELEMU_Busy if locked by other, kStatus_Fail if release failed

void ELEMU_mu_init(ELEMU_Type *mu)

Initialize ELEMU module.

This function does nothing. ELEMU is initialized after leaving ROM.

Parameters

- mu – ELEMU peripheral base address

status_t ELEMU_LP_WakeupPathInit(ELEMU_Type *mu)

Initialize low-power wakeup path for ELEMU.

This function enables clocks and waits for ELEMU startup sequence.

Parameters

- mu – ELEMU peripheral base address

Returns
kStatus_Success if successful, kStatus_Fail otherwise

struct mu_hdr

#include <fsl_elemu.h>

2.14 EWM: External Watchdog Monitor Driver

void EWM_Init(EWM_Type *base, const *ewm_config_t* *config)

Initializes the EWM peripheral.

This function is used to initialize the EWM. After calling, the EWM runs immediately according to the configuration. Note that, except for the interrupt enable control bit, other control bits and registers are write once after a CPU reset. Modifying them more than once generates a bus transfer error.

This is an example.

```
ewm_config_t config;
EWM_GetDefaultConfig(&config);
config.compareHighValue = 0xAAU;
EWM_Init(ewm_base,&config);
```

Parameters

- base – EWM peripheral base address
- config – The configuration of the EWM

void EWM_Deinit(EWM_Type *base)

Deinitializes the EWM peripheral.

This function is used to shut down the EWM.

Parameters

- base – EWM peripheral base address

void EWM_GetDefaultConfig(*ewm_config_t* *config)

Initializes the EWM configuration structure.

This function initializes the EWM configuration structure to default values. The default values are as follows.

```
ewmConfig->enableEwm = true;
ewmConfig->enableEwmInput = false;
ewmConfig->setInputAssertLogic = false;
ewmConfig->enableInterrupt = false;
ewmConfig->ewm_lpo_clock_source_t = kEWM_LpoClockSource0;
ewmConfig->prescaler = 0;
ewmConfig->compareLowValue = 0;
ewmConfig->compareHighValue = 0xFEU;
```

See also:

ewm_config_t

Parameters

- config – Pointer to the EWM configuration structure.

static inline void EWM_EnableInterrupts(EWM_Type *base, uint32_t mask)

Enables the EWM interrupt.

This function enables the EWM interrupt.

Parameters

- base – EWM peripheral base address
- mask – The interrupts to enable The parameter can be combination of the following source if defined
  - kEWM_InterruptEnable

static inline void EWM_DisableInterrupts(EWM_Type *base, uint32_t mask)

Disables the EWM interrupt.

This function enables the EWM interrupt.

Parameters

- base – EWM peripheral base address
- mask – The interrupts to disable The parameter can be combination of the following source if defined
  - kEWM_InterruptEnable

static inline uint32_t EWM_GetStatusFlags(EWM_Type *base)

Gets all status flags.

This function gets all status flags.

This is an example for getting the running flag.

```
uint32_t status;
status = EWM_GetStatusFlags(ewm_base) & kEWM_RunningFlag;
```

See also:

_ewm_status_flags_t

- True: a related status flag has been set.
- False: a related status flag is not set.

Parameters

- base – EWM peripheral base address

Returns

State of the status flag: asserted (true) or not-asserted (false).

void EWM_Refresh(EWM_Type *base)

Services the EWM.

This function resets the EWM counter to zero.

Parameters

- base – EWM peripheral base address

FSL_EWM_DRIVER_VERSION

EWM driver version 2.0.4.

enum _ewm_lpo_clock_source

Describes EWM clock source.

Values:

enumerator kEWM_LpoClockSource0

EWM clock sourced from lpo_clk[0]

enumerator kEWM_LpoClockSource1

EWM clock sourced from lpo_clk[1]

enumerator kEWM_LpoClockSource2

EWM clock sourced from lpo_clk[2]

enumerator kEWM_LpoClockSource3

EWM clock sourced from lpo_clk[3]

enum _ewm_interrupt_enable_t

EWM interrupt configuration structure with default settings all disabled.

This structure contains the settings for all of EWM interrupt configurations.

Values:

enumerator kEWM_InterruptEnable
Enable the EWM to generate an interrupt

enum _ewm_status_flags_t
EWM status flags.
This structure contains the constants for the EWM status flags for use in the EWM functions.
Values:

enumerator kEWM_RunningFlag
Running flag, set when EWM is enabled

typedef enum *_ewm_lpo_clock_source* ewm_lpo_clock_source_t
Describes EWM clock source.

typedef struct *_ewm_config* ewm_config_t
Data structure for EWM configuration.
This structure is used to configure the EWM.

struct _ewm_config
#include <fsl_ewm.h> Data structure for EWM configuration.
This structure is used to configure the EWM.

Public Members

bool enableEwm
Enable EWM module

bool enableEwmInput
Enable EWM_in input

bool setInputAssertLogic
EWM_in signal assertion state

bool enableInterrupt
Enable EWM interrupt

ewm_lpo_clock_source_t clockSource
Clock source select

uint8_t prescaler
Clock prescaler value

uint8_t compareLowValue
Compare low-register value

uint8_t compareHighValue
Compare high-register value

2.15 FGPIO Driver

2.16 C40ESP3 Flash Driver

enum _flash_driver_api_keys

Enumeration for Flash driver API keys.

Note: The resulting value is built with a byte order such that the string being readable in expected order when viewed in a hex editor, if the value is treated as a 32-bit little endian value.

Values:

enumerator kFLASH_ApiEraseKey

Key value used to validate all flash erase APIs.

status_t FLASH_Init(*flash_config_t* *config)

Initializes the global flash properties structure members.

This function checks and initializes the Flash module for the other Flash APIs. When async mode is enabled, this also initializes the async context including the operation queue and buffer pool.

Parameters

- config – Pointer to the storage for the driver runtime state.

Return values

- kStatus_FLASH_Success – API was executed successfully.
- kStatus_FLASH_InvalidArgument – An invalid argument is provided.
- kStatus_FLASH_CommandFailure – Run-time error during the command execution.
- kStatus_FLASH_CommandNotSupported – Flash API is not supported.

status_t FLASH_Erase(*flash_config_t* *config, FMU_Type *base, uint32_t start, uint32_t lengthInBytes, uint32_t key)

Erases the flash sectors encompassed by parameters passed into function.

In synchronous mode, this function blocks until the erase completes. In asynchronous mode, this function queues the erase operation and returns immediately. The operation will be executed later by FLASH_Process() when sufficient idle time is available.

Parameters

- config – Pointer to the flash driver configuration.
- base – FMU peripheral base address.
- start – Start address of the region to erase.
- lengthInBytes – Size of the region to erase in bytes.
- key – Erase API key (kFLASH_ApiEraseKey).

Return values

- kStatus_FLASH_Success – Operation completed or queued successfully.
- kStatus_FLASH_InvalidArgument – Invalid parameters.

- #kStatus_FLASH_Busy – Queue or resources unavailable (async mode).

status_t FLASH_EraseAll(FMU_Type *base, uint32_t key)

Erases entire flash and ifr.

Note: This function always executes synchronously regardless of async mode setting.

Parameters

- base – FMU peripheral base address.
- key – Erase API key.

Return values

kStatus_FLASH_Success – Operation completed successfully.

status_t FLASH_Program(*flash_config_t* *config, FMU_Type *base, uint32_t start, uint8_t *src, uint32_t lengthInBytes)

Programs flash phrases with data at locations passed in through parameters.

In synchronous mode, this function blocks until programming completes. In asynchronous mode, this function copies the source data to an internal buffer, queues the operation, and returns immediately. The operation will be executed later by FLASH_Process() when sufficient idle time is available.

Parameters

- config – Pointer to the flash driver configuration.
- base – FMU peripheral base address.
- start – Target flash address for programming.
- src – Pointer to source data buffer.
- lengthInBytes – Number of bytes to program.

Return values

- kStatus_FLASH_Success – Operation completed or queued successfully.
- kStatus_FLASH_InvalidArgument – Invalid parameters.
- #kStatus_FLASH_Busy – Queue or buffer pool exhausted (async mode).

status_t FLASH_ProgramPage(*flash_config_t* *config, FMU_Type *base, uint32_t start, uint8_t *src, uint32_t lengthInBytes)

Programs flash pages with data at locations passed in through parameters.

Note: In async mode, this function behaves the same as *FLASH_Program()*.

Parameters

- config – Pointer to the flash driver configuration.
- base – FMU peripheral base address.
- start – Target flash address for programming.
- src – Pointer to source data buffer.
- lengthInBytes – Number of bytes to program.

Return values

kStatus_FLASH_Success – Operation completed or queued successfully.

status_t FLASH_Read(uint8_t *dst, uint32_t start, uint32_t lengthInBytes)

Reads data from flash memory.

This function always executes synchronously. In async mode, it checks the pending operation queue and applies any queued writes or erases to the returned data to ensure consistency.

Parameters

- config – Pointer to the flash driver configuration.
- start – Source flash address to read from.
- dst – Pointer to destination buffer.
- lengthInBytes – Number of bytes to read.

Return values

- kStatus_FLASH_Success – Read completed successfully.
- kStatus_FLASH_InvalidArgument – Invalid parameters.

status_t FLASH_VerifyErasePhrase(*flash_config_t* *config, FMU_Type *base, uint32_t start, uint32_t lengthInBytes)

Verify that the flash phrases are erased.

status_t FLASH_VerifyErasePage(*flash_config_t* *config, FMU_Type *base, uint32_t start, uint32_t lengthInBytes)

Verify that the flash pages are erased.

status_t FLASH_VerifyEraseSector(*flash_config_t* *config, FMU_Type *base, uint32_t start, uint32_t lengthInBytes)

Verify that the flash sectors are erased.

status_t FLASH_VerifyEraseAll(FMU_Type *base)

Verify that all flash and IFR space is erased.

status_t FLASH_VerifyEraseBlock(*flash_config_t* *config, FMU_Type *base, uint32_t blockaddr)

Verify that a flash block is erased.

status_t FLASH_VerifyEraseIFRPhrase(*flash_config_t* *config, FMU_Type *base, uint32_t start, uint32_t lengthInBytes)

Verify that the ifr phrases are erased.

status_t FLASH_VerifyEraseIFRPage(*flash_config_t* *config, FMU_Type *base, uint32_t start, uint32_t lengthInBytes)

Verify that the ifr pages are erased.

status_t FLASH_VerifyEraseIFRSector(*flash_config_t* *config, FMU_Type *base, uint32_t start, uint32_t lengthInBytes)

Verify that the ifr sectors are erased.

status_t FLASH_GetProperty(*flash_config_t* *config, *flash_property_tag_t* whichProperty, uint32_t *value)

Returns the desired flash property.

status_t Read_Into_MISR(*flash_config_t* *config, FMU_Type *base, uint32_t start, uint32_t ending, uint32_t *seed, uint32_t *signature)

Read into MISR.

The Read into MISR operation generates a signature based on the contents of the selected flash memory using an embedded MISR.

status_t Read_IFR_Into_MISR(*flash_config_t* *config, FMU_Type *base, uint32_t start, uint32_t ending, uint32_t *seed, uint32_t *signature)

Read IFR into MISR.

The Read IFR into MISR operation generates a signature based on the contents of the selected IFR space using an embedded MISR.

FSL_FLASH_DRIVER_VERSION

Flash driver version for SDK.

Version 2.4.2.

enum _flash_driver_version_constants

Flash driver version for ROM.

Values:

enumerator kFLASH_DriverVersionName

Flash driver version name.

enumerator kFLASH_DriverVersionMajor

Major flash driver version.

enumerator kFLASH_DriverVersionMinor

Minor flash driver version.

enumerator kFLASH_DriverVersionBugfix

Bugfix for flash driver version.

enum _flash_property_tag

Enumeration for various flash properties.

Values:

enumerator kFLASH_PropertyPflash0SectorSize

Pflash sector size property.

enumerator kFLASH_PropertyPflash0TotalSize

Pflash total size property.

enumerator kFLASH_PropertyPflash0BlockSize

Pflash block size property.

enumerator kFLASH_PropertyPflash0BlockCount

Pflash block count property.

enumerator kFLASH_PropertyPflash0BlockBaseAddr

Pflash block base address property.

enumerator kFLASH_PropertyPflash0FacSupport

Pflash fac support property.

enumerator kFLASH_PropertyPflash0AccessSegmentSize
Pflash access segment size property.

enumerator kFLASH_PropertyPflash0AccessSegmentCount
Pflash access segment count property.

enumerator kFLASH_PropertyPflash1SectorSize
Pflash sector size property.

enumerator kFLASH_PropertyPflash1TotalSize
Pflash total size property.

enumerator kFLASH_PropertyPflash1BlockSize
Pflash block size property.

enumerator kFLASH_PropertyPflash1BlockCount
Pflash block count property.

enumerator kFLASH_PropertyPflash1BlockBaseAddr
Pflash block base address property.

enumerator kFLASH_PropertyPflash1FacSupport
Pflash fac support property.

enumerator kFLASH_PropertyPflash1AccessSegmentSize
Pflash access segment size property.

enumerator kFLASH_PropertyPflash1AccessSegmentCount
Pflash access segment count property.

enumerator kFLASH_PropertyFlexRamBlockBaseAddr
FlexRam block base address property.

enumerator kFLASH_PropertyFlexRamTotalSize
FlexRam total size property.

typedef enum *_flash_property_tag* flash_property_tag_t
Enumeration for various flash properties.

typedef struct *_flash_mem_descriptor* flash_mem_desc_t
Flash memory descriptor.

typedef struct *_flash_ifr_desc* flash_ifr_desc_t

typedef struct *_msf1_config* msf1_config_t

typedef struct *_flash_config* flash_config_t

Flash driver state information.

An instance of this structure is allocated by the user of the flash driver and passed into each of the driver APIs.

FLASH_ADDR_MASK

struct _flash_mem_descriptor

#include <fsl_k4_flash.h> Flash memory descriptor.

Public Members

uint32_t blockBase

Base address of the flash block

uint32_t totalSize

The size of the flash block.

uint32_t blockCount

A number of flash blocks.

struct _flash_ifr_desc

#include <fsl_k4_flash.h>

struct _msf1_config

#include <fsl_k4_flash.h>

struct _flash_config

#include <fsl_k4_flash.h> Flash driver state information.

An instance of this structure is allocated by the user of the flash driver and passed into each of the driver APIs.

2.17 FlexCAN: Flex Controller Area Network Driver

2.18 FlexCAN Driver

bool FLEXCAN_IsInstanceHasFDMode(CAN_Type *base)

Determine whether the FlexCAN instance support CAN FD mode at run time.

Note: Use this API only if different soc parts share the SOC part name macro define. Otherwise, a different SOC part name can be used to determine at compile time whether the FlexCAN instance supports CAN FD mode or not. If need use this API to determine if CAN FD mode is supported, the FLEXCAN_Init function needs to be executed first, and then call this API and use the return to value determines whether to supports CAN FD mode, if return true, continue calling FLEXCAN_FDInit to enable CAN FD mode.

Parameters

- base – FlexCAN peripheral base address.

Returns
return TRUE if instance support CAN FD mode, FALSE if instance only support classic CAN (2.0) mode.

uint32_t FLEXCAN_GetFDMailboxOffset(CAN_Type *base, uint8_t mbIdx)

Get Mailbox offset number by dword.

This function gets the offset number of the specified mailbox. Mailbox is not consecutive between memory regions when payload is not 8 bytes so need to calculate the specified mailbox address. For example, in the first memory region, MB[0].CS address is 0x4002_4080. For 32 bytes payload frame, the second mailbox is ((1/12)*512 + 1%12*40)/4 = 10, meaning 10 dword after the 0x4002_4080, which is actually the address of mailbox MB[1].CS.

Parameters

- base – FlexCAN peripheral base address.
- mbIdx – Mailbox index.

Returns
Mailbox address offset in word.

status_t FLEXCAN_EnterFreezeMode(CAN_Type *base)

Enter FlexCAN Freeze Mode.

This function makes the FlexCAN work under Freeze Mode.

Parameters

- base – FlexCAN peripheral base address.

Returns
kStatus_Success Enter Freeze Mode successful kStatus_Timeout Timeout when wait for Freeze Mode Acknowledge

status_t FLEXCAN_ExitFreezeMode(CAN_Type *base)

Exit FlexCAN Freeze Mode.

This function makes the FlexCAN leave Freeze Mode.

Parameters

- base – FlexCAN peripheral base address.

Returns
kStatus_Success Enter Freeze Mode successful kStatus_Timeout Timeout when wait for Freeze Mode Acknowledge

uint32_t FLEXCAN_GetInstance(CAN_Type *base)

Get the FlexCAN instance from peripheral base address.

Parameters

- base – FlexCAN peripheral base address.

Returns
FlexCAN instance.

bool FLEXCAN_CalculateImprovedTimingValues(CAN_Type *base, uint32_t bitRate, uint32_t sourceClock_Hz, *flexcan_timing_config_t* *pTimingConfig)

Calculates the improved timing values by specific bit Rates for classical CAN.

This function use to calculates the Classical CAN timing values according to the given bit rate. The Calculated timing values will be set in CTRL1/CBT/ENCBT register. The calculation is based on the recommendation of the CiA 301 v4.2.0 and previous version document.

Parameters

- base – FlexCAN peripheral base address.
- bitRate – The classical CAN speed in bps defined by user, should be less than or equal to 1Mbps.
- sourceClock_Hz – The Source clock frequency in Hz.
- pTimingConfig – Pointer to the FlexCAN timing configuration structure.

Returns

TRUE if timing configuration found, FALSE if failed to find configuration.

void FLEXCAN_Init(CAN_Type *base, const *flexcan_config_t* *pConfig, uint32_t sourceClock_Hz)

Initializes a FlexCAN instance.

This function initializes the FlexCAN module with user-defined settings. This example shows how to set up the flexcan_config_t parameters and how to call the FLEXCAN_Init function by passing in these parameters.

```
flexcan_config_t flexcanConfig;
flexcanConfig.clkSrc                = kFLEXCAN_ClkSrc0;
flexcanConfig.bitRate               = 1000000U;
flexcanConfig.maxMbNum              = 16;
flexcanConfig.enableLoopBack        = false;
flexcanConfig.enableSelfWakeup      = false;
flexcanConfig.enableIndividMask     = false;
flexcanConfig.enableDoze            = false;
flexcanConfig.disableSelfReception  = false;
flexcanConfig.enableListenOnlyMode  = false;
flexcanConfig.timingConfig          = timingConfig;
FLEXCAN_Init(CAN0, &flexcanConfig, 40000000UL);
```

Parameters

- base – FlexCAN peripheral base address.
- pConfig – Pointer to the user-defined configuration structure.
- sourceClock_Hz – FlexCAN Protocol Engine clock source frequency in Hz.

bool FLEXCAN_FDCalculateImprovedTimingValues(CAN_Type *base, uint32_t bitRate, uint32_t bitRateFD, uint32_t sourceClock_Hz, *flexcan_timing_config_t* *pTimingConfig)

Calculates the improved timing values by specific bit rates for CANFD.

This function use to calculates the CANFD timing values according to the given nominal phase bit rate and data phase bit rate. The Calculated timing values will be set in CBT/ENCBT and FDCBT/EDCBT registers. The calculation is based on the recommendation of the CiA 1301 v1.0.0 document.

Parameters

- base – FlexCAN peripheral base address.
- bitRate – The CANFD bus control speed in bps defined by user.
- bitRateFD – The CAN FD data phase speed in bps defined by user. Equal to bitRate means disable bit rate switching.
- sourceClock_Hz – The Source clock frequency in Hz.

- pTimingConfig – Pointer to the FlexCAN timing configuration structure.

Returns

TRUE if timing configuration found, FALSE if failed to find configuration

void FLEXCAN_FDInit(CAN_Type *base, const *flexcan_config_t* *pConfig, uint32_t sourceClock_Hz, *flexcan_mb_size_t* dataSize, bool brs)

Initializes a FlexCAN instance.

This function initializes the FlexCAN module with user-defined settings. This example shows how to set up the flexcan_config_t parameters and how to call the FLEXCAN_FDInit function by passing in these parameters.

```
flexcan_config_t flexcanConfig;
flexcanConfig.clkSrc                = kFLEXCAN_ClkSrc0;
flexcanConfig.bitRate               = 1000000U;
flexcanConfig.bitRateFD             = 2000000U;
flexcanConfig.maxMbNum              = 16;
flexcanConfig.enableLoopBack        = false;
flexcanConfig.enableSelfWakeup      = false;
flexcanConfig.enableIndividMask     = false;
flexcanConfig.disableSelfReception = false;
flexcanConfig.enableListenOnlyMode = false;
flexcanConfig.enableDoze            = false;
flexcanConfig.timingConfig          = timingConfig;
FLEXCAN_FDInit(CAN0, &flexcanConfig, 80000000UL, kFLEXCAN_16BperMB, true);
```

Parameters

- base – FlexCAN peripheral base address.
- pConfig – Pointer to the user-defined configuration structure.
- sourceClock_Hz – FlexCAN Protocol Engine clock source frequency in Hz.
- dataSize – FlexCAN Message Buffer payload size. The actual transmitted or received CAN FD frame data size needs to be less than or equal to this value.
- brs – True if bit rate switch is enabled in FD mode.

void FLEXCAN_Deinit(CAN_Type *base)

De-initializes a FlexCAN instance.

This function disables the FlexCAN module clock and sets all register values to the reset value.

Parameters

- base – FlexCAN peripheral base address.

void FLEXCAN_GetDefaultConfig(*flexcan_config_t* *pConfig)

Gets the default configuration structure.

This function initializes the FlexCAN configuration structure to default values. The default values are as follows. flexcanConfig->clkSrc = kFLEXCAN_ClkSrc0; flexcanConfig->bitRate = 1000000U; flexcanConfig->bitRateFD = 2000000U; flexcanConfig->maxMbNum = 16; flexcanConfig->enableLoopBack = false; flexcanConfig->enableSelfWakeup = false; flexcanConfig->enableIndividMask = false; flexcanConfig->disableSelfReception = false; flexcanConfig->enableListenOnlyMode = false; flexcanConfig->enableDoze = false; flexcanConfig->enablePretendedeNetworking = false; flexcanConfig->enableMemoryErrorControl = true; flexcanConfig->enableNonCorrectableErrorEnterFreeze = true;

flexcanConfig->enableTransceiverDelayMeasure = true; flexcanConfig->enableRemoteRequestFrameStored = true; flexcanConfig->payloadEndianness = kFLEXCAN_bigEndian; flexcanConfig.timingConfig = timingConfig;

Parameters

- pConfig – Pointer to the FlexCAN configuration structure.

void FLEXCAN_SetTimingConfig(CAN_Type *base, const *flexcan_timing_config_t* *pConfig)

Sets the FlexCAN classical CAN protocol timing characteristic.

This function gives user settings to classical CAN or CAN FD nominal phase timing characteristic. The function is for an experienced user. For less experienced users, call the *FLEXCAN_SetBitRate()* instead.

Note: Calling *FLEXCAN_SetTimingConfig()* overrides the bit rate set in *FLEXCAN_Init()* or *FLEXCAN_SetBitRate()*.

Parameters

- base – FlexCAN peripheral base address.
- pConfig – Pointer to the timing configuration structure.

status_t FLEXCAN_SetBitRate(CAN_Type *base, uint32_t sourceClock_Hz, uint32_t bitRate_Bps)

Set bit rate of FlexCAN classical CAN frame or CAN FD frame nominal phase.

This function set the bit rate of classical CAN frame or CAN FD frame nominal phase base on *FLEXCAN_CalculateImprovedTimingValues()* API calculated timing values.

Note: Calling *FLEXCAN_SetBitRate()* overrides the bit rate set in *FLEXCAN_Init()*.

Parameters

- base – FlexCAN peripheral base address.
- sourceClock_Hz – Source Clock in Hz.
- bitRate_Bps – Bit rate in Bps.

Returns

kStatus_Success - Set CAN baud rate (only Nominal phase) successfully.

void FLEXCAN_SetFDTimingConfig(CAN_Type *base, const *flexcan_timing_config_t* *pConfig)

Sets the FlexCAN CANFD data phase timing characteristic.

This function gives user settings to CANFD data phase timing characteristic. The function is for an experienced user. For less experienced users, call the *FLEXCAN_SetFDBitRate()* to set both Nominal/Data bit Rate instead.

Note: Calling *FLEXCAN_SetFDTimingConfig()* overrides the data phase bit rate set in *FLEXCAN_FDInit()*/FLEXCAN_SetFDBitRate().

Parameters

- base – FlexCAN peripheral base address.
- pConfig – Pointer to the timing configuration structure.

status_t FLEXCAN_SetFDBitRate(CAN_Type *base, uint32_t sourceClock_Hz, uint32_t bitRateN_Bps, uint32_t bitRateD_Bps)

Set bit rate of FlexCAN FD frame.

This function set the baud rate of FLEXCAN FD base on *FLEXCAN_FDCalculateImprovedTimingValues()* API calculated timing values.

Parameters

- base – FlexCAN peripheral base address.
- sourceClock_Hz – Source Clock in Hz.
- bitRateN_Bps – Nominal bit Rate in Bps.
- bitRateD_Bps – Data bit Rate in Bps.

Returns

kStatus_Success - Set CAN FD bit rate (include Nominal and Data phase) successfully.

void FLEXCAN_SetRxMbGlobalMask(CAN_Type *base, uint32_t mask)

Sets the FlexCAN receive message buffer global mask.

This function sets the global mask for the FlexCAN message buffer in a matching process. The configuration is only effective when the Rx individual mask is disabled in the *FLEXCAN_Init()*.

Parameters

- base – FlexCAN peripheral base address.
- mask – Rx Message Buffer Global Mask value.

void FLEXCAN_SetRxFifoGlobalMask(CAN_Type *base, uint32_t mask)

Sets the FlexCAN receive FIFO global mask.

This function sets the global mask for FlexCAN FIFO in a matching process.

Parameters

- base – FlexCAN peripheral base address.
- mask – Rx Fifo Global Mask value.

void FLEXCAN_SetRxIndividualMask(CAN_Type *base, uint8_t maskIdx, uint32_t mask)

Sets the FlexCAN receive individual mask.

This function sets the individual mask for the FlexCAN matching process. The configuration is only effective when the Rx individual mask is enabled in the *FLEXCAN_Init()*. If the Rx FIFO is disabled, the individual mask is applied to the corresponding Message Buffer. If the Rx FIFO is enabled, the individual mask for Rx FIFO occupied Message Buffer is applied to the Rx Filter with the same index. Note that only the first 32 individual masks can be used as the Rx FIFO filter mask.

Parameters

- base – FlexCAN peripheral base address.
- maskIdx – The Index of individual Mask.
- mask – Rx Individual Mask value.

void FLEXCAN_SetTxMbConfig(CAN_Type *base, uint8_t mbIdx, bool enable)

Configures a FlexCAN transmit message buffer.

This function aborts the previous transmission, cleans the Message Buffer, and configures it as a Transmit Message Buffer.

Parameters

- base – FlexCAN peripheral base address.
- mbIdx – The Message Buffer index.
- enable – Enable/disable Tx Message Buffer.
  - true: Enable Tx Message Buffer.
  - false: Disable Tx Message Buffer.

void FLEXCAN_SetRxMbConfig(CAN_Type *base, uint8_t mbIdx, const *flexcan_rx_mb_config_t* *pRxMbConfig, bool enable)

Configures a FlexCAN Receive Message Buffer.

This function cleans a FlexCAN build-in Message Buffer and configures it as a Receive Message Buffer. User should invoke this API when CTRL2[RRS]=1. When CTRL2[RRS]=1, frame's ID is compared to the IDs of the receive mailboxes with the CODE field configured as kFLEXCAN_RxMbEmpty, kFLEXCAN_RxMbFull or kFLEXCAN_RxMbOverrun. Message buffer will store the remote frame in the same fashion of a data frame. No automatic remote response frame will be generated. User need to setup another message buffer to respond remote request.

Parameters

- base – FlexCAN peripheral base address.
- mbIdx – The Message Buffer index.
- pRxMbConfig – Pointer to the FlexCAN Message Buffer configuration structure.
- enable – Enable/disable Rx Message Buffer.
  - true: Enable Rx Message Buffer.
  - false: Disable Rx Message Buffer.

static inline void FLEXCAN_SetMbID(CAN_Type *base, uint8_t mbIdx, uint32_t id)

Configures a FlexCAN Message Buffer identifier.

Parameters

- base – FlexCAN peripheral base address.
- mbIdx – The Message Buffer index.
- id – CAN Message Buffer Identifier, should use *FLEXCAN_ID_EXT()* or *FLEXCAN_ID_STD()* macro.

void FLEXCAN_SetFDTxMbConfig(CAN_Type *base, uint8_t mbIdx, bool enable)

Configures a FlexCAN transmit message buffer.

This function aborts the previous transmission, cleans the Message Buffer, and configures it as a Transmit Message Buffer.

Parameters

- base – FlexCAN peripheral base address.
- mbIdx – The Message Buffer index.
- enable – Enable/disable Tx Message Buffer.
  - true: Enable Tx Message Buffer.
  - false: Disable Tx Message Buffer.

void FLEXCAN_SetFDRxMbConfig(CAN_Type *base, uint8_t mbIdx, const *flexcan_rx_mb_config_t* *pRxMbConfig, bool enable)

Configures a FlexCAN Receive Message Buffer.

This function cleans a FlexCAN build-in Message Buffer and configures it as a Receive Message Buffer.

Parameters

- base – FlexCAN peripheral base address.
- mbIdx – The Message Buffer index.
- pRxMbConfig – Pointer to the FlexCAN Message Buffer configuration structure.
- enable – Enable/disable Rx Message Buffer.
  - true: Enable Rx Message Buffer.
  - false: Disable Rx Message Buffer.

static inline void FLEXCAN_SetFDMbID(CAN_Type *base, uint8_t mbIdx, uint32_t id)

Configures a FlexCAN Message Buffer identifier.

Parameters

- base – FlexCAN peripheral base address.
- mbIdx – The Message Buffer index.
- id – CAN Message Buffer Identifier, should use *FLEXCAN_ID_EXT()* or *FLEXCAN_ID_STD()* macro.

void FLEXCAN_SetRemoteResponseMbConfig(CAN_Type *base, uint8_t mbIdx, const *flexcan_frame_t* *pFrame)

Configures a FlexCAN Remote Response Message Buffer.

User should invoke this API when CTRL2[RRS]=0. When CTRL2[RRS]=0, frame's ID is compared to the IDs of the receive mailboxes with the CODE field configured as kFLEXCAN_RxMbRanswer. If there is a matching ID, then this mailbox content will be transmitted as response. The received remote request frame is not stored in receive buffer. It is only used to trigger a transmission of a frame in response.

Parameters

- base – FlexCAN peripheral base address.
- mbIdx – The Message Buffer index.
- pFrame – Pointer to CAN message frame structure for response.

void FLEXCAN_SetRxFifoConfig(CAN_Type *base, const *flexcan_rx_fifo_config_t* *pRxFifoConfig, bool enable)

Configures the FlexCAN Legacy Rx FIFO.

This function configures the FlexCAN Rx FIFO with given configuration.

Note: Legacy Rx FIFO only can receive classic CAN message.

Parameters

- base – FlexCAN peripheral base address.
- pRxFifoConfig – Pointer to the FlexCAN Legacy Rx FIFO configuration structure. Can be NULL when enable parameter is false.

- enable – Enable/disable Legacy Rx FIFO.
  - true: Enable Legacy Rx FIFO.
  - false: Disable Legacy Rx FIFO.

void FLEXCAN_SetEnhancedRxFifoConfig(CAN_Type *base, const *flexcan_enhanced_rx_fifo_config_t* *pConfig, bool enable)

Configures the FlexCAN Enhanced Rx FIFO.

This function configures the Enhanced Rx FIFO with given configuration.

Note: Enhanced Rx FIFO support receive classic CAN or CAN FD messages, Legacy Rx FIFO and Enhanced Rx FIFO cannot be enabled at the same time.

Parameters

- base – FlexCAN peripheral base address.
- pConfig – Pointer to the FlexCAN Enhanced Rx FIFO configuration structure. Can be NULL when enable parameter is false.
- enable – Enable/disable Enhanced Rx FIFO.
  - true: Enable Enhanced Rx FIFO.
  - false: Disable Enhanced Rx FIFO.

void FLEXCAN_SetPNConfig(CAN_Type *base, const *flexcan_pn_config_t* *pConfig)

Configures the FlexCAN Pretended Networking mode.

This function configures the FlexCAN Pretended Networking mode with given configuration.

Parameters

- base – FlexCAN peripheral base address.
- pConfig – Pointer to the FlexCAN Rx FIFO configuration structure.

static inline uint64_t FLEXCAN_GetStatusFlags(CAN_Type *base)

Gets the FlexCAN module interrupt flags.

This function gets all FlexCAN status flags. The flags are returned as the logical OR value of the enumerators *_flexcan_flags*. To check the specific status, compare the return value with enumerators in *_flexcan_flags*.

Parameters

- base – FlexCAN peripheral base address.

Returns

FlexCAN status flags which are ORed by the enumerators in the _flexcan_flags.

static inline void FLEXCAN_ClearStatusFlags(CAN_Type *base, uint64_t mask)

Clears status flags with the provided mask.

This function clears the FlexCAN status flags with a provided mask. An automatically cleared flag can't be cleared by this function.

Parameters

- base – FlexCAN peripheral base address.
- mask – The status flags to be cleared, it is logical OR value of *_flexcan_flags*.

static inline void FLEXCAN_GetBusErrCount(CAN_Type *base, uint8_t *txErrBuf, uint8_t *rxErrBuf)

Gets the FlexCAN Bus Error Counter value.

This function gets the FlexCAN Bus Error Counter value for both Tx and Rx direction. These values may be needed in the upper layer error handling.

Parameters

- base – FlexCAN peripheral base address.
- txErrBuf – Buffer to store Tx Error Counter value.
- rxErrBuf – Buffer to store Rx Error Counter value.

static inline uint64_t FLEXCAN_GetMbStatusFlags(CAN_Type *base, uint64_t mask)

Gets the FlexCAN low 64 Message Buffer interrupt flags.

This function gets the interrupt flags of a given Message Buffers.

Parameters

- base – FlexCAN peripheral base address.
- mask – The ORed FlexCAN Message Buffer mask.

Returns

The status of given Message Buffers.

static inline uint64_t FLEXCAN_GetHigh64MbStatusFlags(CAN_Type *base, uint64_t mask)

Gets the FlexCAN High 64 Message Buffer interrupt flags.

Valid only if the number of available MBs exceeds 64.

Parameters

- base – FlexCAN peripheral base address.
- mask – The ORed FlexCAN Message Buffer mask.

Returns

The status of given Message Buffers.

static inline void FLEXCAN_ClearMbStatusFlags(CAN_Type *base, uint64_t mask)

Clears the FlexCAN low 64 Message Buffer interrupt flags.

This function clears the interrupt flags of a given Message Buffers.

Parameters

- base – FlexCAN peripheral base address.
- mask – The ORed FlexCAN Message Buffer mask.

static inline void FLEXCAN_ClearHigh64MbStatusFlags(CAN_Type *base, uint64_t mask)

Clears the FlexCAN High 64 Message Buffer interrupt flags.

Valid only if the number of available MBs exceeds 64.

Parameters

- base – FlexCAN peripheral base address.
- mask – The ORed FlexCAN Message Buffer mask.

void FLEXCAN_GetMemoryErrorReportStatus(CAN_Type *base, *flexcan_memory_error_report_status_t* *errorStatus)

Gets the FlexCAN Memory Error Report registers status.

This function gets the FlexCAN Memory Error Report registers status.

Parameters

- base – FlexCAN peripheral base address.
- errorStatus – Pointer to FlexCAN Memory Error Report registers status structure.

static inline uint8_t FLEXCAN_GetPNMatchCount(CAN_Type *base)

Gets the FlexCAN Number of Matches when in Pretended Networking.

This function gets the number of times a given message has matched the predefined filtering criteria for ID and/or PL before a wakeup event.

Parameters

- base – FlexCAN peripheral base address.

Returns

The number of received wake up msessages.

static inline uint32_t FLEXCAN_GetEnhancedFifoDataCount(CAN_Type *base)

Gets the number of FlexCAN Enhanced Rx FIFO available frames.

This function gets the number of CAN messages stored in the Enhanced Rx FIFO.

Parameters

- base – FlexCAN peripheral base address.

Returns

The number of available CAN messages stored in the Enhanced Rx FIFO.

static inline void FLEXCAN_EnableInterrupts(CAN_Type *base, uint64_t mask)

Enables FlexCAN interrupts according to the provided mask.

This function enables the FlexCAN interrupts according to the provided mask. The mask is a logical OR of enumeration members, see *_flexcan_interrupt_enable*.

Parameters

- base – FlexCAN peripheral base address.
- mask – The interrupts to enable. Logical OR of *_flexcan_interrupt_enable*.

static inline void FLEXCAN_DisableInterrupts(CAN_Type *base, uint64_t mask)

Disables FlexCAN interrupts according to the provided mask.

This function disables the FlexCAN interrupts according to the provided mask. The mask is a logical OR of enumeration members, see *_flexcan_interrupt_enable*.

Parameters

- base – FlexCAN peripheral base address.
- mask – The interrupts to disable. Logical OR of *_flexcan_interrupt_enable*.

static inline void FLEXCAN_EnableMbInterrupts(CAN_Type *base, uint64_t mask)

Enables FlexCAN low 64 Message Buffer interrupts.

This function enables the interrupts of given Message Buffers.

Parameters

- base – FlexCAN peripheral base address.
- mask – The ORed FlexCAN Message Buffer mask.

static inline void FLEXCAN_EnableHigh64MbInterrupts(CAN_Type *base, uint64_t mask)

Enables FlexCAN high 64 Message Buffer interrupts.

Valid only if the number of available MBs exceeds 64.

Parameters

- base – FlexCAN peripheral base address.
- mask – The ORed FlexCAN Message Buffer mask.

static inline void FLEXCAN_DisableMbInterrupts(CAN_Type *base, uint64_t mask)

Disables FlexCAN low 64 Message Buffer interrupts.

This function disables the interrupts of given Message Buffers.

Parameters

- base – FlexCAN peripheral base address.
- mask – The ORed FlexCAN Message Buffer mask.

static inline void FLEXCAN_DisableHigh64MbInterrupts(CAN_Type *base, uint64_t mask)

Disables FlexCAN high 64 Message Buffer interrupts.

Valid only if the number of available MBs exceeds 64.

Parameters

- base – FlexCAN peripheral base address.
- mask – The ORed FlexCAN Message Buffer mask.

void FLEXCAN_EnableRxFifoDMA(CAN_Type *base, bool enable)

Enables or disables the FlexCAN Rx FIFO DMA request.

This function enables or disables the DMA feature of FlexCAN build-in Rx FIFO.

Parameters

- base – FlexCAN peripheral base address.
- enable – true to enable, false to disable.

static inline uintptr_t FLEXCAN_GetRxFifoHeadAddr(CAN_Type *base)

Gets the Rx FIFO Head address.

This function returns the FlexCAN Rx FIFO Head address, which is mainly used for the DMA/eDMA use case.

Parameters

- base – FlexCAN peripheral base address.

Returns

FlexCAN Rx FIFO Head address.

static inline *status_t* FLEXCAN_Enable(CAN_Type *base, bool enable)

Enables or disables the FlexCAN module operation.

This function enables or disables the FlexCAN module.

Parameters

- base – FlexCAN base pointer.
- enable – true to enable, false to disable.

Returns

kStatus_Success Enable FlexCAN module successful kStatus_Timeout Timeout when wait for Low-Power Mode Acknowledge

status_t FLEXCAN_WriteTxMb(CAN_Type *base, uint8_t mbIdx, const *flexcan_frame_t* *pTxFrame)

Writes a FlexCAN Message to the Transmit Message Buffer.

This function writes a CAN Message to the specified Transmit Message Buffer and changes the Message Buffer state to start CAN Message transmit. After that the function returns immediately.

Parameters

- base – FlexCAN peripheral base address.
- mbIdx – The FlexCAN Message Buffer index.
- pTxFrame – Pointer to CAN message frame to be sent.

Return values

- kStatus_Success – - Write Tx Message Buffer Successfully.
- kStatus_Fail – - Tx Message Buffer is currently in use.

status_t FLEXCAN_ReadRxMb(CAN_Type *base, uint8_t mbIdx, *flexcan_frame_t* *pRxFrame)

Reads a FlexCAN Message from Receive Message Buffer.

This function reads a CAN message from a specified Receive Message Buffer. The function fills a receive CAN message frame structure with just received data and activates the Message Buffer again. The function returns immediately.

Parameters

- base – FlexCAN peripheral base address.
- mbIdx – The FlexCAN Message Buffer index.
- pRxFrame – Pointer to CAN message frame structure for reception.

Return values

- kStatus_Success – - Rx Message Buffer is full and has been read successfully.
- kStatus_FLEXCAN_RxOverflow – - Rx Message Buffer is already overflowed and has been read successfully.
- kStatus_Fail – - Rx Message Buffer is empty or inactive.
- kStatus_Timeout – - Timeout when wait for Rx Message Buffer busy.

status_t FLEXCAN_WriteFDTxMb(CAN_Type *base, uint8_t mbIdx, const *flexcan_fd_frame_t* *pTxFrame)

Writes a FlexCAN FD Message to the Transmit Message Buffer.

This function writes a CAN FD Message to the specified Transmit Message Buffer and changes the Message Buffer state to start CAN FD Message transmit. After that the function returns immediately.

Parameters

- base – FlexCAN peripheral base address.
- mbIdx – The FlexCAN FD Message Buffer index.
- pTxFrame – Pointer to CAN FD message frame to be sent.

Return values

- kStatus_Success – - Write Tx Message Buffer Successfully.
- kStatus_Fail – - Tx Message Buffer is currently in use.

status_t FLEXCAN_ReadFDRxMb(CAN_Type *base, uint8_t mbIdx, *flexcan_fd_frame_t* *pRxFrame)

Reads a FlexCAN FD Message from Receive Message Buffer.

This function reads a CAN FD message from a specified Receive Message Buffer. The function fills a receive CAN FD message frame structure with just received data and activates the Message Buffer again. The function returns immediately.

Parameters

- base – FlexCAN peripheral base address.
- mbIdx – The FlexCAN FD Message Buffer index.
- pRxFrame – Pointer to CAN FD message frame structure for reception.

Return values

- kStatus_Success – - Rx Message Buffer is full and has been read successfully.
- kStatus_FLEXCAN_RxOverflow – - Rx Message Buffer is already overflowed and has been read successfully.
- kStatus_Fail – - Rx Message Buffer is empty or inactive.
- kStatus_Timeout – - Timeout when wait for Rx Message Buffer busy.

status_t FLEXCAN_ReadRxFifo(CAN_Type *base, *flexcan_frame_t* *pRxFrame)

Reads a FlexCAN Message from Legacy Rx FIFO.

This function reads a CAN message from the FlexCAN Legacy Rx FIFO.

Parameters

- base – FlexCAN peripheral base address.
- pRxFrame – Pointer to CAN message frame structure for reception.

Return values

- kStatus_Success – - Read Message from Rx FIFO successfully.
- kStatus_Fail – - Rx FIFO is not enabled.

status_t FLEXCAN_ReadEnhancedRxFifo(CAN_Type *base, *flexcan_fd_frame_t* *pRxFrame)

Reads a FlexCAN Message from Enhanced Rx FIFO.

This function reads a CAN or CAN FD message from the FlexCAN Enhanced Rx FIFO.

Parameters

- base – FlexCAN peripheral base address.
- pRxFrame – Pointer to CAN FD message frame structure for reception.

Return values

- kStatus_Success – - Read Message from Rx FIFO successfully.
- kStatus_Fail – - Rx FIFO is not enabled.

status_t FLEXCAN_ReadPNWakeUpMB(CAN_Type *base, uint8_t mbIdx, *flexcan_frame_t* *pRxFrame)

Reads a FlexCAN Message from Wake Up MB.

This function reads a CAN message from the FlexCAN Wake up Message Buffers. There are four Wake up Message Buffers (WMBs) used to store incoming messages in Pretended Networking mode. The WMB index indicates the arrival order. The last message is stored in WMB3.

Parameters

- base – FlexCAN peripheral base address.
- pRxFrame – Pointer to CAN message frame structure for reception.
- mbIdx – The FlexCAN Wake up Message Buffer index. Range in 0x0 ~ 0x3.

Return values

- kStatus_Success – - Read Message from Wake up Message Buffer successfully.
- kStatus_Fail – - Wake up Message Buffer has no valid content.

status_t FLEXCAN_TransferFDSendBlocking(CAN_Type *base, uint8_t mbIdx, *flexcan_fd_frame_t* *pTxFrame)

Performs a polling send transaction on the CAN bus.

Note: A transfer handle does not need to be created before calling this API.

Parameters

- base – FlexCAN peripheral base pointer.
- mbIdx – The FlexCAN FD Message Buffer index.
- pTxFrame – Pointer to CAN FD message frame to be sent.

Return values

- kStatus_Success – - Write Tx Message Buffer Successfully.
- kStatus_Fail – - Tx Message Buffer is currently in use.
- kStatus_Timeout – - Failed to send frames within specific time.

status_t FLEXCAN_TransferFDReceiveBlocking(CAN_Type *base, uint8_t mbIdx, *flexcan_fd_frame_t* *pRxFrame)

Performs a polling receive transaction on the CAN bus.

Note: A transfer handle does not need to be created before calling this API.

Parameters

- base – FlexCAN peripheral base pointer.
- mbIdx – The FlexCAN FD Message Buffer index.
- pRxFrame – Pointer to CAN FD message frame structure for reception.

Return values

- kStatus_Success – - Rx Message Buffer is full and has been read successfully.
- kStatus_FLEXCAN_RxOverflow – - Rx Message Buffer is already overflowed and has been read successfully.
- kStatus_Fail – - Rx Message Buffer is empty.
- kStatus_Timeout – - Failed to receive frames within specific time.

status_t FLEXCAN_TransferFDSendNonBlocking(CAN_Type *base, *flexcan_handle_t* *handle, *flexcan_mb_transfer_t* *pMbXfer)

Sends a message using IRQ.

This function sends a message using IRQ. This is a non-blocking function, which returns right away. When messages have been sent out, the send callback function is called.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- pMbXfer – FlexCAN FD Message Buffer transfer structure. See the *flexcan_mb_transfer_t*.

Return values

- kStatus_Success – Start Tx Message Buffer sending process successfully.
- kStatus_Fail – Write Tx Message Buffer failed.
- kStatus_FLEXCAN_TxBusy – Tx Message Buffer is in use.

status_t FLEXCAN_TransferFDReceiveNonBlocking(CAN_Type *base, *flexcan_handle_t* *handle, *flexcan_mb_transfer_t* *pMbXfer)

Receives a message using IRQ.

This function receives a message using IRQ. This is non-blocking function, which returns right away. When the message has been received, the receive callback function is called.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- pMbXfer – FlexCAN FD Message Buffer transfer structure. See the *flexcan_mb_transfer_t*.

Return values

- kStatus_Success – - Start Rx Message Buffer receiving process successfully.
- kStatus_FLEXCAN_RxBusy – - Rx Message Buffer is in use.

void FLEXCAN_TransferFDAbortSend(CAN_Type *base, *flexcan_handle_t* *handle, uint8_t mbIdx)

Aborts the interrupt driven message send process.

This function aborts the interrupt driven message send process.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- mbIdx – The FlexCAN FD Message Buffer index.

void FLEXCAN_TransferFDAbortReceive(CAN_Type *base, *flexcan_handle_t* *handle, uint8_t mbIdx)

Aborts the interrupt driven message receive process.

This function aborts the interrupt driven message receive process.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.

- mbIdx – The FlexCAN FD Message Buffer index.

status_t FLEXCAN_TransferSendBlocking(CAN_Type *base, uint8_t mbIdx, *flexcan_frame_t* *pTxFrame)

Performs a polling send transaction on the CAN bus.

Note: A transfer handle does not need to be created before calling this API.

Parameters

- base – FlexCAN peripheral base pointer.
- mbIdx – The FlexCAN Message Buffer index.
- pTxFrame – Pointer to CAN message frame to be sent.

Return values

- kStatus_Success – - Write Tx Message Buffer Successfully.
- kStatus_Fail – - Tx Message Buffer is currently in use.
- kStatus_Timeout – - Failed to send frames within specific time.

status_t FLEXCAN_TransferReceiveBlocking(CAN_Type *base, uint8_t mbIdx, *flexcan_frame_t* *pRxFrame)

Performs a polling receive transaction on the CAN bus.

Note: A transfer handle does not need to be created before calling this API.

Parameters

- base – FlexCAN peripheral base pointer.
- mbIdx – The FlexCAN Message Buffer index.
- pRxFrame – Pointer to CAN message frame structure for reception.

Return values

- kStatus_Success – - Rx Message Buffer is full and has been read successfully.
- kStatus_FLEXCAN_RxOverflow – - Rx Message Buffer is already overflowed and has been read successfully.
- kStatus_Fail – - Rx Message Buffer is empty.
- kStatus_Timeout – - Failed to receive frames within specific time.

status_t FLEXCAN_TransferReceiveFifoBlocking(CAN_Type *base, *flexcan_frame_t* *pRxFrame)

Performs a polling receive transaction from Legacy Rx FIFO on the CAN bus.

Note: A transfer handle does not need to be created before calling this API.

Parameters

- base – FlexCAN peripheral base pointer.
- pRxFrame – Pointer to CAN message frame structure for reception.

Return values

- kStatus_Success – - Read Message from Rx FIFO successfully.

- kStatus_Fail – - Rx FIFO is not enabled.
- kStatus_Timeout – - Failed to receive frames within specific time.

status_t FLEXCAN_TransferReceiveEnhancedFifoBlocking(CAN_Type *base, *flexcan_fd_frame_t* *pRxFrame)

Performs a polling receive transaction from Enhanced Rx FIFO on the CAN bus.

Note: A transfer handle does not need to be created before calling this API.

Parameters

- base – FlexCAN peripheral base pointer.
- pRxFrame – Pointer to CAN FD message frame structure for reception.

Return values

- kStatus_Success – - Read Message from Rx FIFO successfully.
- kStatus_Fail – - Rx FIFO is not enabled.
- kStatus_Timeout – - Failed to receive frames within specific time.

void FLEXCAN_TransferCreateHandle(CAN_Type *base, *flexcan_handle_t* *handle, *flexcan_transfer_callback_t* callback, void *userData)

Initializes the FlexCAN handle.

This function initializes the FlexCAN handle, which can be used for other FlexCAN transactional APIs. Usually, for a specified FlexCAN instance, call this API once to get the initialized handle.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- callback – The callback function.
- userData – The parameter of the callback function.

status_t FLEXCAN_TransferSendNonBlocking(CAN_Type *base, *flexcan_handle_t* *handle, *flexcan_mb_transfer_t* *pMbXfer)

Sends a message using IRQ.

This function sends a message using IRQ. This is a non-blocking function, which returns right away. When messages have been sent out, the send callback function is called.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- pMbXfer – FlexCAN Message Buffer transfer structure. See the *flexcan_mb_transfer_t*.

Return values

- kStatus_Success – Start Tx Message Buffer sending process successfully.
- kStatus_Fail – Write Tx Message Buffer failed.
- kStatus_FLEXCAN_TxBusy – Tx Message Buffer is in use.

status_t FLEXCAN_TransferReceiveNonBlocking(CAN_Type *base, *flexcan_handle_t* *handle, *flexcan_mb_transfer_t* *pMbXfer)

Receives a message using IRQ.

This function receives a message using IRQ. This is non-blocking function, which returns right away. When the message has been received, the receive callback function is called.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- pMbXfer – FlexCAN Message Buffer transfer structure. See the *flexcan_mb_transfer_t*.

Return values

- kStatus_Success – - Start Rx Message Buffer receiving process successfully.
- kStatus_FLEXCAN_RxBusy – - Rx Message Buffer is in use.

status_t FLEXCAN_TransferRemoteRequestNonBlocking(CAN_Type *base, *flexcan_handle_t* *handle, *flexcan_mb_transfer_t* *pMbXfer)

Sends a remote request frame using IRQ.

This function sends a remote request frame using IRQ. This is a non-blocking function, which returns right away. When the remote request frame has been sent out, the send callback function is called. User should invoke API FLEXCAN_TransferReceiveNonBlocking to receive the response frame. Receive message buffer index should less than send message buffer index.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- pMbXfer – FlexCAN Message Buffer transfer structure. See the *flexcan_mb_transfer_t*.

Return values

- kStatus_Success – Start Tx remote request frame sending process successfully.
- kStatus_Fail – Write Tx Message Buffer failed.
- kStatus_FLEXCAN_TxBusy – Message Buffer is transmitting remote request frame.

status_t FLEXCAN_TransferRemoteResponseNonBlocking(CAN_Type *base, *flexcan_handle_t* *handle, *flexcan_mb_transfer_t* *pMbXfer)

Configures a FlexCAN Message Buffer for automatic remote response using IRQ.

This function configures a Message Buffer to automatically respond to remote request frames using IRQ. This is a non-blocking function, which returns right away. When a matching remote request frame is received, the configured response frame will be transmitted automatically, and the callback function will be called. User should invoke this API when CTRL2[RRS]=0. When CTRL2[RRS]=0, if a remote request frame is received and matches a mailbox configured with CODE=kFLEXCAN_RxMbRanswer, the mailbox content will be transmitted as a response frame automatically. The received remote request frame is not stored.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- pMbXfer – FlexCAN Message Buffer transfer structure. See the *flexcan_mb_transfer_t*.

Return values

- kStatus_Success – Configure remote response Message Buffer successfully.
- kStatus_Busy – Message Buffer is waiting for remote request frame or transmitting response frame.

status_t FLEXCAN_TransferReceiveFifoNonBlocking(CAN_Type *base, *flexcan_handle_t* *handle, *flexcan_fifo_transfer_t* *pFifoXfer)

Receives a message from Rx FIFO using IRQ.

This function receives a message using IRQ. This is a non-blocking function, which returns right away. When all messages have been received, the receive callback function is called.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- pFifoXfer – FlexCAN Rx FIFO transfer structure. See the *flexcan_fifo_transfer_t*.

Return values

- kStatus_Success – - Start Rx FIFO receiving process successfully.
- kStatus_FLEXCAN_RxFifoBusy – - Rx FIFO is currently in use.

status_t FLEXCAN_TransferGetReceiveFifoCount(CAN_Type *base, *flexcan_handle_t* *handle, size_t *count)

Gets the Rx Fifo transfer status during a interrupt non-blocking receive.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- count – Number of CAN messages receive so far by the non-blocking transaction.

Return values

- kStatus_InvalidArgument – count is Invalid.
- kStatus_Success – Successfully return the count.

status_t FLEXCAN_TransferReceiveEnhancedFifoNonBlocking(CAN_Type *base, *flexcan_handle_t* *handle, *flexcan_fifo_transfer_t* *pFifoXfer)

Receives a message from Enhanced Rx FIFO using IRQ.

This function receives a message using IRQ. This is a non-blocking function, which returns right away. When all messages have been received, the receive callback function is called.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- pFifoXfer – FlexCAN Rx FIFO transfer structure. See the ref flexcan_fifo_transfer_t.@

Return values

- kStatus_Success – - Start Rx FIFO receiving process successfully.
- kStatus_FLEXCAN_RxFifoBusy – - Rx FIFO is currently in use.

static inline *status_t* FLEXCAN_TransferGetReceiveEnhancedFifoCount(CAN_Type *base, *flexcan_handle_t* *handle, size_t *count)

Gets the Enhanced Rx Fifo transfer status during a interrupt non-blocking receive.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- count – Number of CAN messages receive so far by the non-blocking transaction.

Return values

- kStatus_InvalidArgument – count is Invalid.
- kStatus_Success – Successfully return the count.

uint32_t FLEXCAN_GetTimeStamp(*flexcan_handle_t* *handle, uint8_t mbIdx)

Gets the detail index of Mailbox's Timestamp by handle.

Then function can only be used when calling non-blocking Data transfer (TX/RX) API, After TX/RX data transfer done (User can get the status by handler's callback function), we can get the detail index of Mailbox's timestamp by handle, Detail non-blocking data transfer API (TX/RX) contain. -FLEXCAN_TransferSendNonBlocking -FLEXCAN_TransferFDSendNonBlocking -FLEXCAN_TransferReceiveNonBlocking -FLEXCAN_TransferFDReceiveNonBlocking -FLEXCAN_TransferReceiveFifoNonBlocking

Parameters

- handle – FlexCAN handle pointer.
- mbIdx – The FlexCAN Message Buffer index.

Return values

the – index of mailbox 's timestamp stored in the handle.

static inline uint32_t FLEXCAN_GetHighResolutionTimeStamp(CAN_Type *base, uint8_t mbIdx)

void FLEXCAN_TransferAbortSend(CAN_Type *base, *flexcan_handle_t* *handle, uint8_t mbIdx)

Aborts the interrupt driven message send process.

This function aborts the interrupt driven message send process.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- mbIdx – The FlexCAN Message Buffer index.

void FLEXCAN_TransferAbortReceive(CAN_Type *base, *flexcan_handle_t* *handle, uint8_t mbIdx)

Aborts the interrupt driven message receive process.

This function aborts the interrupt driven message receive process.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.

- mbIdx – The FlexCAN Message Buffer index.

void FLEXCAN_TransferAbortReceiveFifo(CAN_Type *base, *flexcan_handle_t* *handle)

Aborts the interrupt driven message receive from Rx FIFO process.

This function aborts the interrupt driven message receive from Rx FIFO process.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.

void FLEXCAN_TransferAbortReceiveEnhancedFifo(CAN_Type *base, *flexcan_handle_t* *handle)

Aborts the interrupt driven message receive from Enhanced Rx FIFO process.

This function aborts the interrupt driven message receive from Enhanced Rx FIFO process.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.

void FLEXCAN_TransferHandleIRQ(CAN_Type *base, *flexcan_handle_t* *handle)

FlexCAN IRQ handle function.

This function handles the FlexCAN Error, the Message Buffer, and the Rx FIFO IRQ request.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.

void FLEXCAN_MbHandleIRQ(CAN_Type *base, *flexcan_handle_t* *handle, uint32_t startMbIdx, uint32_t endMbIdx)

FlexCAN Message Buffer IRQ handle function.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- startMbIdx – First Message Buffer to handle.
- endMbIdx – Last Message Buffer to handle.

void FLEXCAN_EnhancedRxFifoHandleIRQ(CAN_Type *base, *flexcan_handle_t* *handle)

FlexCAN Enhanced Rx FIFO IRQ handle function.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.

void FLEXCAN_BusoffErrorHandleIRQ(CAN_Type *base, *flexcan_handle_t* *handle)

FlexCAN Bus Off, Error and Warning IRQ handle function.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.

void FLEXCAN_PNWakeUpHandleIRQ(CAN_Type *base, *flexcan_handle_t* *handle)

FlexCAN Pretended Networking Wake-up IRQ handle function.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.

void FLEXCAN_MemoryErrorHandleIRQ(CAN_Type *base, *flexcan_handle_t* *handle)

FlexCAN Memory Error IRQ handle function.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.

FSL_FLEXCAN_DRIVER_VERSION

FlexCAN driver version.

FlexCAN transfer status.

Values:

enumerator kStatus_FLEXCAN_TxBusy

Tx Message Buffer is Busy.

enumerator kStatus_FLEXCAN_TxIdle

Tx Message Buffer is Idle.

enumerator kStatus_FLEXCAN_TxSwitchToRx

Remote Message is send out and Message buffer changed to Receive one.

enumerator kStatus_FLEXCAN_RxBusy

Rx Message Buffer is Busy.

enumerator kStatus_FLEXCAN_RxIdle

Rx Message Buffer is Idle.

enumerator kStatus_FLEXCAN_RxOverflow

Rx Message Buffer is Overflowed.

enumerator kStatus_FLEXCAN_RxFifoBusy

Rx Message FIFO is Busy.

enumerator kStatus_FLEXCAN_RxFifoIdle

Rx Message FIFO is Idle.

enumerator kStatus_FLEXCAN_RxFifoOverflow

Rx Message FIFO is overflowed.

enumerator kStatus_FLEXCAN_RxFifoWarning

Rx Message FIFO is almost overflowed.

enumerator kStatus_FLEXCAN_RxFifoDisabled

Rx Message FIFO is disabled during reading.

enumerator kStatus_FLEXCAN_ErrorStatus
FlexCAN Module Error and Status.

enumerator kStatus_FLEXCAN_WakeUp
FlexCAN is waken up from STOP mode.

enumerator kStatus_FLEXCAN_UnHandled
UnHadled Interrupt asserted.

enumerator kStatus_FLEXCAN_RxRemote
Rx Remote Message Received in Mail box.

enumerator kStatus_FLEXCAN_RxFifoUnderflow
Enhanced Rx Message FIFO is underflow.

enumerator kStatus_FLEXCAN_MemoryError
FlexCAN Memory Error.

enum _flexcan_frame_format
FlexCAN frame format.
Values:

enumerator kFLEXCAN_FrameFormatStandard
Standard frame format attribute.

enumerator kFLEXCAN_FrameFormatExtend
Extend frame format attribute.

enum _flexcan_frame_type
FlexCAN frame type.
Values:

enumerator kFLEXCAN_FrameTypeData
Data frame type attribute.

enumerator kFLEXCAN_FrameTypeRemote
Remote frame type attribute.

enum _flexcan_clock_source
FlexCAN clock source.

Deprecated:
Do not use the kFLEXCAN_ClkSrcOs. It has been superceded kFLEXCAN_ClkSrc0
Do not use the kFLEXCAN_ClkSrcPeri. It has been superceded kFLEXCAN_ClkSrc1
Values:

enumerator kFLEXCAN_ClkSrcOsc
FlexCAN Protocol Engine clock from Oscillator.

enumerator kFLEXCAN_ClkSrcPeri
FlexCAN Protocol Engine clock from Peripheral Clock.

enumerator kFLEXCAN_ClkSrc0
FlexCAN Protocol Engine clock selected by user as SRC == 0.

enumerator kFLEXCAN_ClkSrc1
FlexCAN Protocol Engine clock selected by user as SRC == 1.

enum _flexcan_wake_up_source
FlexCAN wake up source.

Values:

enumerator kFLEXCAN_WakeupSrcUnfiltered
FlexCAN uses unfiltered Rx input to detect edge.

enumerator kFLEXCAN_WakeupSrcFiltered
FlexCAN uses filtered Rx input to detect edge.

enum _flexcan_endianness
FlexCAN payload endianness.

Values:

enumerator kFLEXCAN_bigEndian
Transmit frame with MSB first, receive frame with big-endian format.

enumerator kFLEXCAN_littleEndian
Transmit frame with LSB first, receive frame with little-endian format.

enum _flexcan_MB_timestamp_base
FlexCAN timebase used for capturing 16-bit TIME_STAMP field of message buffer.

Values:

enumerator kFLEXCAN_CANTimer
FlexCAN free-running timer.

enumerator kFLEXCAN_Lower16bitsHRTimer
Lower 16 bits of high-resolution on-chip timer.

enumerator kFLEXCAN_Upper16bitsHRTimer
Upper 16 bits of high-resolution on-chip timer.

enum _flexcan_capture_point
FlexCAN capture point of 32-bit high resolution timebase during a CAN frame.

Values:

enumerator kFLEXCAN_CANFrameID2ndBit
Second bit of identifier field of any frame is on the CAN bus. HR_TIME_STAMPn register will not capture 32-bit counter value.

enumerator kFLEXCAN_CANFrameEnd
End of the CAN frame.

enumerator kFLEXCAN_CANFrameStart
Start of the CAN frame.

enumerator kFLEXCAN_CANFDFrameRes
Start of frame for classical CAN frames; res bit for CAN FD frames.

enum _flexcan_rx_fifo_filter_type
FlexCAN Rx Fifo Filter type.

Values:

enumerator kFLEXCAN_RxFifoFilterTypeA
One full ID (standard and extended) per ID Filter element.

enumerator kFLEXCAN_RxFifoFilterTypeB
Two full standard IDs or two partial 14-bit ID slices per ID Filter Table element.

enumerator kFLEXCAN_RxFifoFilterTypeC
Four partial 8-bit Standard or extended ID slices per ID Filter Table element.

enumerator kFLEXCAN_RxFifoFilterTypeD
All frames rejected.

enum _flexcan_mb_size
FlexCAN Message Buffer Payload size.

Values:

enumerator kFLEXCAN_8BperMB
Selects 8 bytes per Message Buffer.

enumerator kFLEXCAN_16BperMB
Selects 16 bytes per Message Buffer.

enumerator kFLEXCAN_32BperMB
Selects 32 bytes per Message Buffer.

enumerator kFLEXCAN_64BperMB
Selects 64 bytes per Message Buffer.

enum _flexcan_fd_frame_length
FlexCAN CAN FD frame supporting data length (available DLC values).

For Tx, when the Data size corresponding to DLC value stored in the MB selected for transmission is larger than the MB Payload size, FlexCAN adds the necessary number of bytes

with constant 0xCC pattern to complete the expected DLC. For Rx, when the Data size corresponding to DLC value received from the CAN bus is larger than the MB Payload size, the high order bytes that do not fit the Payload size will lose.

Values:

enumerator kFLEXCAN_0BperFrame
Frame contains 0 valid data bytes.

enumerator kFLEXCAN_1BperFrame
Frame contains 1 valid data bytes.

enumerator kFLEXCAN_2BperFrame
Frame contains 2 valid data bytes.

enumerator kFLEXCAN_3BperFrame
Frame contains 3 valid data bytes.

enumerator kFLEXCAN_4BperFrame
Frame contains 4 valid data bytes.

enumerator kFLEXCAN_5BperFrame
Frame contains 5 valid data bytes.

enumerator kFLEXCAN_6BperFrame
Frame contains 6 valid data bytes.

enumerator kFLEXCAN_7BperFrame
Frame contains 7 valid data bytes.

enumerator kFLEXCAN_8BperFrame
Frame contains 8 valid data bytes.

enumerator kFLEXCAN_12BperFrame
Frame contains 12 valid data bytes.

enumerator kFLEXCAN_16BperFrame
Frame contains 16 valid data bytes.

enumerator kFLEXCAN_20BperFrame
Frame contains 20 valid data bytes.

enumerator kFLEXCAN_24BperFrame
Frame contains 24 valid data bytes.

enumerator kFLEXCAN_32BperFrame
Frame contains 32 valid data bytes.

enumerator kFLEXCAN_48BperFrame
Frame contains 48 valid data bytes.

enumerator kFLEXCAN_64BperFrame
Frame contains 64 valid data bytes.

enum _flexcan_efifo_dma_per_read_length
FlexCAN Enhanced Rx Fifo DMA transfer per read length enumerations.

Values:

enumerator kFLEXCAN_1WordPerRead
Transfer 1 32-bit words (CS).

enumerator kFLEXCAN_2WordPerRead
Transfer 2 32-bit words (CS + ID).

enumerator kFLEXCAN_3WordPerRead
Transfer 3 32-bit words (CS + ID + 1~4 bytes data).

enumerator kFLEXCAN_4WordPerRead
Transfer 4 32-bit words (CS + ID + 5~8 bytes data).

enumerator kFLEXCAN_5WordPerRead
Transfer 5 32-bit words (CS + ID + 9~12 bytes data).

enumerator kFLEXCAN_6WordPerRead
Transfer 6 32-bit words (CS + ID + 13~16 bytes data).

enumerator kFLEXCAN_7WordPerRead
Transfer 7 32-bit words (CS + ID + 17~20 bytes data).

enumerator kFLEXCAN_8WordPerRead
Transfer 8 32-bit words (CS + ID + 21~24 bytes data).

enumerator kFLEXCAN_9WordPerRead
Transfer 9 32-bit words (CS + ID + 25~28 bytes data).

enumerator kFLEXCAN_10WordPerRead
Transfer 10 32-bit words (CS + ID + 29~32 bytes data).

enumerator kFLEXCAN_11WordPerRead
Transfer 11 32-bit words (CS + ID + 33~36 bytes data).

enumerator kFLEXCAN_12WordPerRead
Transfer 12 32-bit words (CS + ID + 37~40 bytes data).

enumerator kFLEXCAN_13WordPerRead
Transfer 13 32-bit words (CS + ID + 41~44 bytes data).

enumerator kFLEXCAN_14WordPerRead
Transfer 14 32-bit words (CS + ID + 45~48 bytes data).

enumerator kFLEXCAN_15WordPerRead

Transfer 15 32-bit words (CS + ID + 49~52 bytes data).

enumerator kFLEXCAN_16WordPerRead

Transfer 16 32-bit words (CS + ID + 53~56 bytes data).

enumerator kFLEXCAN_17WordPerRead

Transfer 17 32-bit words (CS + ID + 57~60 bytes data).

enumerator kFLEXCAN_18WordPerRead

Transfer 18 32-bit words (CS + ID + 61~64 bytes data).

enumerator kFLEXCAN_19WordPerRead

Transfer 19 32-bit words (CS + ID + 64 bytes data + ID HIT).

enumerator kFLEXCAN_20WordPerRead

Transfer 20 32-bit words (CS + ID + 64 bytes data + ID HIT + HR timestamp).

enum _flexcan_rx_fifo_priority

FlexCAN Enhanced/Legacy Rx FIFO priority.

The matching process starts from the Rx MB(or Enhanced/Legacy Rx FIFO) with higher priority. If no MB(or Enhanced/Legacy Rx FIFO filter) is satisfied, the matching process goes on with the Enhanced/Legacy Rx FIFO(or Rx MB) with lower priority.

Values:

enumerator kFLEXCAN_RxFifoPrioLow

Matching process start from Rx Message Buffer first.

enumerator kFLEXCAN_RxFifoPrioHigh

Matching process start from Enhanced/Legacy Rx FIFO first.

enum _flexcan_interrupt_enable

FlexCAN interrupt enable enumerations.

This provides constants for the FlexCAN interrupt enable enumerations for use in the FlexCAN functions.

Note: FlexCAN Message Buffers and Legacy Rx FIFO interrupts not included in.

Values:

enumerator kFLEXCAN_BusOffInterruptEnable

Bus Off interrupt, use bit 15.

enumerator kFLEXCAN_ErrorInterruptEnable

CAN Error interrupt, use bit 14.

enumerator kFLEXCAN_TxWarningInterruptEnable

Tx Warning interrupt, use bit 11.

enumerator kFLEXCAN_RxWarningInterruptEnable

Rx Warning interrupt, use bit 10.

enumerator kFLEXCAN_FDErrorInterruptEnable

CAN FD Error interrupt, use bit 31.

enumerator kFLEXCAN_PNMatchWakeUpInterruptEnable

PN Match Wake Up interrupt, use high word bit 17.

enumerator kFLEXCAN_PNTimeoutWakeUpInterruptEnable

PN Timeout Wake Up interrupt, use high word bit 16. Enhanced Rx FIFO Underflow interrupt, use high word bit 31.

enumerator kFLEXCAN_ERxFifoUnderflowInterruptEnable

Enhanced Rx FIFO Overflow interrupt, use high word bit 30.

enumerator kFLEXCAN_ERxFifoOverflowInterruptEnable

Enhanced Rx FIFO Watermark interrupt, use high word bit 29.

enumerator kFLEXCAN_ERxFifoWatermarkInterruptEnable

Enhanced Rx FIFO Data Avilable interrupt, use high word bit 28.

enumerator kFLEXCAN_ERxFifoDataAvlInterruptEnable

enumerator kFLEXCAN_HostAccessNCErrorInterruptEnable

Host Access With Non-Correctable Errors interrupt, use high word bit 0.

enumerator kFLEXCAN_FlexCanAccessNCErrorInterruptEnable

FlexCAN Access With Non-Correctable Errors interrupt, use high word bit 2.

enumerator kFLEXCAN_HostOrFlexCanCErrorInterruptEnable

Host or FlexCAN Access With Correctable Errors interrupt, use high word bit 3.

enum _flexcan_flags

FlexCAN status flags.

This provides constants for the FlexCAN status flags for use in the FlexCAN functions.

Note: The CPU read action clears the bits corresponding to the FlEXCAN_ErrorFlag macro, therefore user need to read status flags and distinguish which error is occur using *_flexcan_error_flags* enumerations.

Values:

enumerator kFLEXCAN_ErrorOverrunFlag

Error Overrun Status.

enumerator kFLEXCAN_FDErrorIntFlag

CAN FD Error Interrupt Flag.

enumerator kFLEXCAN_BusoffDoneIntFlag
Bus Off process completed Interrupt Flag.

enumerator kFLEXCAN_SynchFlag
CAN Synchronization Status.

enumerator kFLEXCAN_TxWarningIntFlag
Tx Warning Interrupt Flag.

enumerator kFLEXCAN_RxWarningIntFlag
Rx Warning Interrupt Flag.

enumerator kFLEXCAN_IdleFlag
FlexCAN In IDLE Status.

enumerator kFLEXCAN_FaultConfinementFlag
FlexCAN Fault Confinement State.

enumerator kFLEXCAN_TransmittingFlag
FlexCAN In Transmission Status.

enumerator kFLEXCAN_ReceivingFlag
FlexCAN In Reception Status.

enumerator kFLEXCAN_BusOffIntFlag
Bus Off Interrupt Flag.

enumerator kFLEXCAN_ErrorIntFlag
CAN Error Interrupt Flag.

enumerator kFLEXCAN_ErrorFlag

enumerator kFLEXCAN_PNMatchIntFlag
PN Matching Event Interrupt Flag.

enumerator kFLEXCAN_PNTimeoutIntFlag
PN Timeout Event Interrupt Flag.

enumerator kFLEXCAN_ERxFifoUnderflowIntFlag
Enhanced Rx FIFO underflow Interrupt Flag.

enumerator kFLEXCAN_ERxFifoOverflowIntFlag
Enhanced Rx FIFO overflow Interrupt Flag.

enumerator kFLEXCAN_ERxFifoWatermarkIntFlag
Enhanced Rx FIFO watermark Interrupt Flag.

enumerator kFLEXCAN_ERxFifoDataAvlIntFlag
Enhanced Rx FIFO data available Interrupt Flag.

enumerator kFLEXCAN_ERxFifoEmptyFlag
Enhanced Rx FIFO empty status.

enumerator kFLEXCAN_ERxFifoFullFlag
Enhanced Rx FIFO full status.

enumerator kFLEXCAN_HostAccessNonCorrectableErrorIntFlag
Host Access With Non-Correctable Error Interrupt Flag.

enumerator kFLEXCAN_FlexCanAccessNonCorrectableErrorIntFlag
FlexCAN Access With Non-Correctable Error Interrupt Flag.

enumerator kFLEXCAN_CorrectableErrorIntFlag
Correctable Error Interrupt Flag.

enumerator kFLEXCAN_HostAccessNonCorrectableErrorOverrunFlag
Host Access With Non-Correctable Error Interrupt Overrun Flag.

enumerator kFLEXCAN_FlexCanAccessNonCorrectableErrorOverrunFlag
FlexCAN Access With Non-Correctable Error Interrupt Overrun Flag.

enumerator kFLEXCAN_CorrectableErrorOverrunFlag
Correctable Error Interrupt Overrun Flag.

enumerator kFLEXCAN_AllMemoryErrorIntFlag
All Memory Error Interrupt Flags.

enumerator kFLEXCAN_AllMemoryErrorFlag
All Memory Error Flags.

enum _flexcan_error_flags
FlexCAN error status flags.

The FlexCAN Error Status enumerations is used to report current error of the FlexCAN bus. This enumerations should be used with KFLEXCAN_ErrorFlag in *_flexcan_flags* enumerations to ditermine which error is generated.

Values:

enumerator kFLEXCAN_FDStuffingError
Stuffing Error.

enumerator kFLEXCAN_FDFormError
Form Error.

enumerator kFLEXCAN_FDCrcError
Cyclic Redundancy Check Error.

enumerator kFLEXCAN_FDBit0Error
Unable to send dominant bit.

enumerator kFLEXCAN_FDBit1Error
Unable to send recessive bit.

enumerator kFLEXCAN_TxErrorWarningFlag
Tx Error Warning Status.

enumerator kFLEXCAN_RxErrorWarningFlag
Rx Error Warning Status.

enumerator kFLEXCAN_StuffingError
Stuffing Error.

enumerator kFLEXCAN_FormError
Form Error.

enumerator kFLEXCAN_CrcError
Cyclic Redundancy Check Error.

enumerator kFLEXCAN_AckError
Received no ACK on transmission.

enumerator kFLEXCAN_Bit0Error
Unable to send dominant bit.

enumerator kFLEXCAN_Bit1Error
Unable to send recessive bit.

FlexCAN Legacy Rx FIFO status flags.

The FlexCAN Legacy Rx FIFO Status enumerations are used to determine the status of the Rx FIFO. Because Rx FIFO occupy the MB0 ~ MB7 (Rx Fifo filter also occupies more Message Buffer space), Rx FIFO status flags are mapped to the corresponding Message Buffer status flags.

Values:

enumerator kFLEXCAN_RxFifoOverflowFlag
Rx FIFO overflow flag.

enumerator kFLEXCAN_RxFifoWarningFlag
Rx FIFO almost full flag.

enumerator kFLEXCAN_RxFifoFrameAvlFlag
Frames available in Rx FIFO flag.

enum _flexcan_memory_error_type
FlexCAN Memory Error Type.

Values:

enumerator kFLEXCAN_CorrectableError
The memory error is correctable which means on bit error.

enumerator kFLEXCAN_NonCorrectableError
The memory error is non-correctable which means two bit errors.

enum _flexcan_memory_access_type
FlexCAN Memory Access Type.
Values:

enumerator kFLEXCAN_MoveOutFlexCanAccess
The memory error was detected during move-out FlexCAN access.

enumerator kFLEXCAN_MoveInAccess
The memory error was detected during move-in FlexCAN access.

enumerator kFLEXCAN_TxArbitrationAccess
The memory error was detected during Tx Arbitration FlexCAN access.

enumerator kFLEXCAN_RxMatchingAccess
The memory error was detected during Rx Matching FlexCAN access.

enumerator kFLEXCAN_MoveOutHostAccess
The memory error was detected during Rx Matching Host (CPU) access.

enum _flexcan_byte_error_syndrome
FlexCAN Memory Error Byte Syndrome.
Values:

enumerator kFLEXCAN_NoError
No bit error in this byte.

enumerator kFLEXCAN_ParityBits0Error
Parity bit 0 error in this byte.

enumerator kFLEXCAN_ParityBits1Error
Parity bit 1 error in this byte.

enumerator kFLEXCAN_ParityBits2Error
Parity bit 2 error in this byte.

enumerator kFLEXCAN_ParityBits3Error
Parity bit 3 error in this byte.

enumerator kFLEXCAN_ParityBits4Error
Parity bit 4 error in this byte.

enumerator kFLEXCAN_DataBits0Error
Data bit 0 error in this byte.

enumerator kFLEXCAN_DataBits1Error
Data bit 1 error in this byte.

enumerator kFLEXCAN_DataBits2Error
Data bit 2 error in this byte.

enumerator kFLEXCAN_DataBits3Error
Data bit 3 error in this byte.

enumerator kFLEXCAN_DataBits4Error
Data bit 4 error in this byte.

enumerator kFLEXCAN_DataBits5Error
Data bit 5 error in this byte.

enumerator kFLEXCAN_DataBits6Error
Data bit 6 error in this byte.

enumerator kFLEXCAN_DataBits7Error
Data bit 7 error in this byte.

enumerator kFLEXCAN_AllZeroError
All-zeros non-correctable error in this byte.

enumerator kFLEXCAN_AllOneError
All-ones non-correctable error in this byte.

enumerator kFLEXCAN_NonCorrectableErrors
Non-correctable error in this byte.

enum _flexcan_pn_match_source
FlexCAN Pretended Networking match source selection.

Values:

enumerator kFLEXCAN_PNMatSrcID
Message match with ID filtering.

enumerator kFLEXCAN_PNMatSrcIDAndData
Message match with ID filtering and payload filtering.

enum _flexcan_pn_match_mode
FlexCAN Pretended Networking mode match type.

Values:

enumerator kFLEXCAN_PNMatModeEqual
Match upon ID/Payload contents against an exact target value.

enumerator kFLEXCAN_PNMatModeGreater
Match upon an ID/Payload value greater than or equal to a specified target value.

enumerator kFLEXCAN_PNMatModeSmaller
Match upon an ID/Payload value smaller than or equal to a specified target value.

enumerator kFLEXCAN_PNMatModeRange
Match upon an ID/Payload value inside a range, greater than or equal to a specified lower limit, and smaller than or equal to a specified upper limit

typedef enum *_flexcan_frame_format* flexcan_frame_format_t
FlexCAN frame format.

typedef enum *_flexcan_frame_type* flexcan_frame_type_t
FlexCAN frame type.

typedef enum *_flexcan_clock_source* flexcan_clock_source_t
FlexCAN clock source.

Deprecated:
Do not use the kFLEXCAN_ClkSrcOs. It has been superceded kFLEXCAN_ClkSrc0

Do not use the kFLEXCAN_ClkSrcPeri. It has been superceded kFLEXCAN_ClkSrc1

typedef enum *_flexcan_wake_up_source* flexcan_wake_up_source_t
FlexCAN wake up source.

typedef enum *_flexcan_endianness* flexcan_endianness_t
FlexCAN payload endianness.

typedef enum *_flexcan_MB_timestamp_base* flexcan_MB_timestamp_base_t
FlexCAN timebase used for capturing 16-bit TIME_STAMP field of message buffer.

typedef enum *_flexcan_capture_point* flexcan_capture_point_t
FlexCAN capture point of 32-bit high resolution timebase during a CAN frame.

typedef enum *_flexcan_rx_fifo_filter_type* flexcan_rx_fifo_filter_type_t
FlexCAN Rx Fifo Filter type.

typedef enum *_flexcan_mb_size* flexcan_mb_size_t
FlexCAN Message Buffer Payload size.

typedef enum *_flexcan_efifo_dma_per_read_length* flexcan_efifo_dma_per_read_length_t
FlexCAN Enhanced Rx Fifo DMA transfer per read length enumerations.

typedef enum *_flexcan_rx_fifo_priority* flexcan_rx_fifo_priority_t

FlexCAN Enhanced/Legacy Rx FIFO priority.

The matching process starts from the Rx MB(or Enhanced/Legacy Rx FIFO) with higher priority. If no MB(or Enhanced/Legacy Rx FIFO filter) is satisfied, the matching process goes on with the Enhanced/Legacy Rx FIFO(or Rx MB) with lower priority.

typedef enum *_flexcan_memory_error_type* flexcan_memory_error_type_t

FlexCAN Memory Error Type.

typedef enum *_flexcan_memory_access_type* flexcan_memory_access_type_t

FlexCAN Memory Access Type.

typedef enum *_flexcan_byte_error_syndrome* flexcan_byte_error_syndrome_t

FlexCAN Memory Error Byte Syndrome.

typedef struct *_flexcan_memory_error_report_status* flexcan_memory_error_report_status_t

FlexCAN memory error register status structure.

This structure contains the memory access properties that caused a memory error access. It is used as the parameter of *FLEXCAN_GetMemoryErrorReportStatus()* function. And user can use FLEXCAN_GetMemoryErrorReportStatus to get the status of the last memory error access.

typedef struct *_flexcan_frame* flexcan_frame_t

FlexCAN message frame structure.

typedef struct *_flexcan_fd_frame* flexcan_fd_frame_t

CAN FD message frame structure.

The CAN FD message supporting up to sixty four bytes can be used for a data frame, depending on the length selected for the message buffers. The length should be a enumeration member, see *_flexcan_fd_frame_length*.

typedef struct *_flexcan_timing_config* flexcan_timing_config_t

FlexCAN protocol timing characteristic configuration structure.

typedef struct *_flexcan_config* flexcan_config_t

FlexCAN module configuration structure.

Deprecated:

Do not use the baudRate. It has been superceded bitRate

Do not use the baudRateFD. It has been superceded bitRateFD

typedef struct *_flexcan_rx_mb_config* flexcan_rx_mb_config_t

FlexCAN Receive Message Buffer configuration structure.

This structure is used as the parameter of *FLEXCAN_SetRxMbConfig()* function. The *FLEXCAN_SetRxMbConfig()* function is used to configure FlexCAN Receive Message Buffer. The function abort previous receiving process, clean the Message Buffer and activate the Rx Message Buffer using given Message Buffer setting.

typedef enum *_flexcan_pn_match_source* flexcan_pn_match_source_t

FlexCAN Pretended Networking match source selection.

typedef enum *_flexcan_pn_match_mode* flexcan_pn_match_mode_t

FlexCAN Pretended Networking mode match type.

typedef struct *_flexcan_pn_config* flexcan_pn_config_t

FlexCAN Pretended Networking configuration structure.

This structure is used as the parameter of *FLEXCAN_SetPNConfig()* function. The *FLEXCAN_SetPNConfig()* function is used to configure FlexCAN Networking work mode.

typedef struct *_flexcan_rx_fifo_config* flexcan_rx_fifo_config_t

FlexCAN Legacy Rx FIFO configuration structure.

typedef struct *_flexcan_enhanced_rx_fifo_std_id_filter* flexcan_enhanced_rx_fifo_std_id_filter_t

FlexCAN Enhanced Rx FIFO Standard ID filter element structure.

typedef struct *_flexcan_enhanced_rx_fifo_ext_id_filter* flexcan_enhanced_rx_fifo_ext_id_filter_t

FlexCAN Enhanced Rx FIFO Extended ID filter element structure.

typedef struct *_flexcan_enhanced_rx_fifo_config* flexcan_enhanced_rx_fifo_config_t

FlexCAN Enhanced Rx FIFO configuration structure.

typedef struct *_flexcan_mb_transfer* flexcan_mb_transfer_t

FlexCAN Message Buffer transfer.

typedef struct *_flexcan_fifo_transfer* flexcan_fifo_transfer_t

FlexCAN Rx FIFO transfer.

typedef struct *_flexcan_handle* flexcan_handle_t

FlexCAN handle structure definition.

typedef void (*flexcan_transfer_callback_t)(CAN_Type *base, *flexcan_handle_t* *handle, *status_t* status, uint64_t result, void *userData)

FLEXCAN_WAIT_TIMEOUT

FLEXCAN_POLLING_TIMEOUT

Max loops to wait for polling transfer.

FLEXCAN_MODULE_TIMEOUT

Max loops to wait for FlexCAN register access complete.

FLEXCAN_MB_BUSY_TIMEOUT

Max loops to wait for FlexCAN RX Message Buffer busy.

DLC_LENGTH_DECODE(dlc)

FlexCAN frame length helper macro.

FLEXCAN_ID_STD(id)

FlexCAN Frame ID helper macro.

Standard Frame ID helper macro.

FLEXCAN_ID_EXT(id)

Extend Frame ID helper macro.

FLEXCAN_RX_MB_STD_MASK(id, rtr, ide)

FlexCAN Rx Message Buffer Mask helper macro.

Standard Rx Message Buffer Mask helper macro.

FLEXCAN_RX_MB_EXT_MASK(id, rtr, ide)

Extend Rx Message Buffer Mask helper macro.

FLEXCAN_RX_FIFO_STD_MASK_TYPE_A(id, rtr, ide)

FlexCAN Legacy Rx FIFO Mask helper macro.

Standard Rx FIFO Mask helper macro Type A helper macro.

FLEXCAN_RX_FIFO_STD_MASK_TYPE_B_HIGH(id, rtr, ide)

Standard Rx FIFO Mask helper macro Type B upper part helper macro.

FLEXCAN_RX_FIFO_STD_MASK_TYPE_B_LOW(id, rtr, ide)

Standard Rx FIFO Mask helper macro Type B lower part helper macro.

FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_HIGH(id)

Standard Rx FIFO Mask helper macro Type C upper part helper macro.

FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_MID_HIGH(id)

Standard Rx FIFO Mask helper macro Type C mid-upper part helper macro.

FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_MID_LOW(id)

Standard Rx FIFO Mask helper macro Type C mid-lower part helper macro.

FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_LOW(id)

Standard Rx FIFO Mask helper macro Type C lower part helper macro.

FLEXCAN_RX_FIFO_EXT_MASK_TYPE_A(id, rtr, ide)

Extend Rx FIFO Mask helper macro Type A helper macro.

FLEXCAN_RX_FIFO_EXT_MASK_TYPE_B_HIGH(id, rtr, ide)

Extend Rx FIFO Mask helper macro Type B upper part helper macro.

FLEXCAN_RX_FIFO_EXT_MASK_TYPE_B_LOW(id, rtr, ide)

Extend Rx FIFO Mask helper macro Type B lower part helper macro.

FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_HIGH(id)

Extend Rx FIFO Mask helper macro Type C upper part helper macro.

FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_MID_HIGH(id)

Extend Rx FIFO Mask helper macro Type C mid-upper part helper macro.

FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_MID_LOW(id)

Extend Rx FIFO Mask helper macro Type C mid-lower part helper macro.

FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_LOW(id)

Extend Rx FIFO Mask helper macro Type C lower part helper macro.

FLEXCAN_RX_FIFO_STD_FILTER_TYPE_A(id, rtr, ide)

FlexCAN Rx FIFO Filter helper macro.

Standard Rx FIFO Filter helper macro Type A helper macro.

FLEXCAN_RX_FIFO_STD_FILTER_TYPE_B_HIGH(id, rtr, ide)
Standard Rx FIFO Filter helper macro Type B upper part helper macro.

FLEXCAN_RX_FIFO_STD_FILTER_TYPE_B_LOW(id, rtr, ide)
Standard Rx FIFO Filter helper macro Type B lower part helper macro.

FLEXCAN_RX_FIFO_STD_FILTER_TYPE_C_HIGH(id)
Standard Rx FIFO Filter helper macro Type C upper part helper macro.

FLEXCAN_RX_FIFO_STD_FILTER_TYPE_C_MID_HIGH(id)
Standard Rx FIFO Filter helper macro Type C mid-upper part helper macro.

FLEXCAN_RX_FIFO_STD_FILTER_TYPE_C_MID_LOW(id)
Standard Rx FIFO Filter helper macro Type C mid-lower part helper macro.

FLEXCAN_RX_FIFO_STD_FILTER_TYPE_C_LOW(id)
Standard Rx FIFO Filter helper macro Type C lower part helper macro.

FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_A(id, rtr, ide)
Extend Rx FIFO Filter helper macro Type A helper macro.

FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_B_HIGH(id, rtr, ide)
Extend Rx FIFO Filter helper macro Type B upper part helper macro.

FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_B_LOW(id, rtr, ide)
Extend Rx FIFO Filter helper macro Type B lower part helper macro.

FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_HIGH(id)
Extend Rx FIFO Filter helper macro Type C upper part helper macro.

FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_MID_HIGH(id)
Extend Rx FIFO Filter helper macro Type C mid-upper part helper macro.

FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_MID_LOW(id)
Extend Rx FIFO Filter helper macro Type C mid-lower part helper macro.

FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_LOW(id)
Extend Rx FIFO Filter helper macro Type C lower part helper macro.

ENHANCED_RX_FIFO_FSCH(x)
FlexCAN Enhanced Rx FIFO Filter and Mask helper macro.

RTR_STD_HIGH(x)

RTR_STD_LOW(x)

RTR_EXT(x)

ID_STD_LOW(id)

ID_STD_HIGH(id)

ID_EXT(id)

FLEXCAN_ENHANCED_RX_FIFO_STD_MASK_AND_FILTER(id, rtr, id_mask, rtr_mask)
Standard ID filter element with filter + mask scheme.

FLEXCAN_ENHANCED_RX_FIFO_STD_FILTER_WITH_RANGE(id_upper, rtr, id_lower, rtr_mask)
Standard ID filter element with filter range.

FLEXCAN_ENHANCED_RX_FIFO_STD_TWO_FILTERS(id1, rtr1, id2, rtr2)
Standard ID filter element with two filters without masks.

FLEXCAN_ENHANCED_RX_FIFO_EXT_MASK_AND_FILTER_LOW(id, rtr)
Extended ID filter element with filter + mask scheme low word.

FLEXCAN_ENHANCED_RX_FIFO_EXT_MASK_AND_FILTER_HIGH(id_mask, rtr_mask)
Extended ID filter element with filter + mask scheme high word.

FLEXCAN_ENHANCED_RX_FIFO_EXT_FILTER_WITH_RANGE_LOW(id_upper, rtr)
Extended ID filter element with range scheme low word.

FLEXCAN_ENHANCED_RX_FIFO_EXT_FILTER_WITH_RANGE_HIGH(id_lower, rtr_mask)
Extended ID filter element with range scheme high word.

FLEXCAN_ENHANCED_RX_FIFO_EXT_TWO_FILTERS_LOW(id2, rtr2)
Extended ID filter element with two filters without masks low word.

FLEXCAN_ENHANCED_RX_FIFO_EXT_TWO_FILTERS_HIGH(id1, rtr1)
Extended ID filter element with two filters without masks high word.

FLEXCAN_PN_STD_MASK(id, rtr)
FlexCAN Pretended Networking ID Mask helper macro.

Standard Rx Message Buffer Mask helper macro.

FLEXCAN_PN_EXT_MASK(id, rtr)
Extend Rx Message Buffer Mask helper macro.

FLEXCAN_PN_INT_MASK(x)
FlexCAN interrupt/status flag helper macro.

FLEXCAN_PN_INT_UNMASK(x)

FLEXCAN_PN_STATUS_MASK(x)

FLEXCAN_PN_STATUS_UNMASK(x)

FLEXCAN_EFIFO_INT_MASK(x)

FLEXCAN_EFIFO_INT_UNMASK(x)

FLEXCAN_EFIFO_STATUS_MASK(x)

FLEXCAN_EFIFO_STATUS_UNMASK(x)

FLEXCAN_MECR_INT_MASK(x)

FLEXCAN_MECR_INT_UNMASK(x)

FLEXCAN_MECR_STATUS_MASK(x)

FLEXCAN_MECR_STATUS_UNMASK(x)

FLEXCAN_ERROR_AND_STATUS_INT_FLAG

FLEXCAN_PNWAKE_UP_FLAG

FLEXCAN_WAKE_UP_FLAG

FLEXCAN_MEMORY_ERROR_INT_FLAG

FLEXCAN_ENHANCED_RX_FIFO_INT_FLAG
FlexCAN Enhanced Rx FIFO base address helper macro.

E_RX_FIFO(base)

FLEXCAN_CALLBACK(x)

FlexCAN transfer callback function.

The FlexCAN transfer callback returns a value from the underlying layer. If the status equals to kStatus_FLEXCAN_ErrorStatus, the result parameter is the Content of FlexCAN status register which can be used to get the working status(or error status) of FlexCAN module. If the status equals to other FlexCAN Message Buffer transfer status, the result is the index of Message Buffer that generate transfer event. If the status equals to other FlexCAN Message Buffer transfer status, the result is meaningless and should be Ignored.

struct __flexcan_memory_error_report_status

#include <fsl_flexcan.h> FlexCAN memory error register status structure.

This structure contains the memory access properties that caused a memory error access. It is used as the parameter of *FLEXCAN_GetMemoryErrorReportStatus()* function. And user can use FLEXCAN_GetMemoryErrorReportStatus to get the status of the last memory error access.

Public Members

flexcan_memory_error_type_t errorType

The type of memory error that giving rise to the report.

flexcan_memory_access_type_t accessType

The type of memory access that giving rise to the memory error.

uint16_t accessAddress

The address where memory error detected.

uint32_t errorData

The raw data word read from memory with error.

struct __flexcan_frame

#include <fsl_flexcan.h> FlexCAN message frame structure.

struct __flexcan_fd_frame

#include <fsl_flexcan.h> CAN FD message frame structure.

The CAN FD message supporting up to sixty four bytes can be used for a data frame, depending on the length selected for the message buffers. The length should be a enumeration member, see *_flexcan_fd_frame_length*.

Public Members

uint32_t idhit

Note: ID HIT offset is changed dynamically according to data length code (DLC), when DLC is 15, they will be located below. Using FLEXCAN_FixEnhancedRxFifoFrameIdHit

API is recommended to ensure this idhit value is correct. CAN Enhanced Rx FIFO filter hit id (This value is only used in Enhanced Rx FIFO receive mode).

uint32_t hrtimestamp

Note: HR timestamp offset is changed dynamically according to data length code (DLC). External 32-bit on-chip timer high-resolution timestamp.

struct __flexcan__timing__config

#include <fsl_flexcan.h> FlexCAN protocol timing characteristic configuration structure.

Public Members

uint32_t preDivider

Classic CAN or CAN FD nominal phase bit rate prescaler.

uint32_t rJumpwidth

Classic CAN or CAN FD nominal phase Re-sync Jump Width.

uint32_t phaseSeg1

Classic CAN or CAN FD nominal phase Segment 1.

uint32_t phaseSeg2

Classic CAN or CAN FD nominal phase Segment 2.

uint32_t propSeg

Classic CAN or CAN FD nominal phase Propagation Segment.

uint32_t fpreDivider

CAN FD data phase bit rate prescaler.

uint32_t frJumpwidth

CAN FD data phase Re-sync Jump Width.

uint32_t fphaseSeg1

CAN FD data phase Phase Segment 1.

uint32_t fphaseSeg2

CAN FD data phase Phase Segment 2.

uint32_t fpropSeg

CAN FD data phase Propagation Segment.

struct __flexcan__config

#include <fsl_flexcan.h> FlexCAN module configuration structure.

Deprecated:

Do not use the baudRate. It has been superceded bitRate

Do not use the baudRateFD. It has been superceded bitRateFD

Public Members

flexcan_clock_source_t clkSrc

Clock source for FlexCAN Protocol Engine.

flexcan_wake_up_source_t wakeupSrc

Wake up source selection.

uint8_t maxMbNum

The maximum number of Message Buffers used by user.

bool enableLoopBack

Enable or Disable Loop Back Self Test Mode.

bool enableTimerSync

Enable or Disable Timer Synchronization.

bool enableIndividMask

Enable or Disable Rx Individual Mask and Queue feature.

bool disableSelfReception

Enable or Disable Self Reflection.

bool enableListenOnlyMode

Enable or Disable Listen Only Mode.

bool enableDoze

Enable or Disable Doze Mode.

bool enablePretendedeNetworking

Enable or Disable the Pretended Networking mode.

bool enableMemoryErrorControl

Enable or Disable the memory errors detection and correction mechanism.

bool enableNonCorrectableErrorEnterFreeze

Enable or Disable Non-Correctable Errors In FlexCAN Access Put Device In Freeze Mode.

bool enableTransceiverDelayMeasure

Enable or Disable the transceiver delay measurement, when it is enabled, then the secondary sample point position is determined by the sum of the transceiver delay measurement plus the enhanced TDC offset.

bool enableRemoteRequestFrameStored

true: Store Remote Request Frame in the same fashion of data frame. false: Generate an automatic Remote Response Frame.

bool enableThreeSamplingMode

true: Three samples are used to determine the value of the received bit, providing noise immunity against fast transient changes. false: One sample is used to determine the value of the received bit.

bool disableBusOffAutoRecovery

true: Automatic recovering from Bus-Off state is disabled. false: Automatic recovering from Bus-Off state is enabled according to CAN Specification.

uint8_t txArbitrationStartDelay

Determines how many CAN bits the transmission arbitration process start point can be delayed from the first bit of CRC field on CAN bus, valid range 0-31.

bool enableEntireArbitrationCompare

true: Enable Entire Frame Arbitration Field Comparison for Rx MB. The IDE and RTR fields of the RX MBs are compared to their corresponding bits within the incoming frame (mask bits apply). false: The IDE field of the RX MB filter is always compared and RTR is never compared despite mask bits.

flexcan_endianness_t payloadEndianness

Selects the byte order for the payload of transmit and receive frames, see *flexcan_endianness_t*.

bool enableExternalTimeTick

true: External time tick clocks the free-running timer. false: FlexCAN bit clock clocks the free-running timer.

flexcan_MB_timestamp_base_t captureTimeBase

Timebase of message buffer 16-bit TIME_STAMP field.

flexcan_capture_point_t capturePoint

Point in time when 32-bit timebase is captured during CAN frame.

struct __flexcan_rx_mb_config

#include <fsl_flexcan.h> FlexCAN Receive Message Buffer configuration structure.

This structure is used as the parameter of *FLEXCAN_SetRxMbConfig()* function. The *FLEXCAN_SetRxMbConfig()* function is used to configure FlexCAN Receive Message Buffer. The function abort previous receiving process, clean the Message Buffer and activate the Rx Message Buffer using given Message Buffer setting.

Public Members

uint32_t id

CAN Message Buffer Frame Identifier, should be set using *FLEXCAN_ID_EXT()* or *FLEXCAN_ID_STD()* macro.

flexcan_frame_format_t format

CAN Frame Identifier *format(Standard of Extend)*.

flexcan_frame_type_t type

CAN Frame Type(Data or Remote for classical CAN only).

struct __flexcan__pn__config

#include <fsl_flexcan.h> FlexCAN Pretended Networking configuration structure.

This structure is used as the parameter of *FLEXCAN_SetPNConfig()* function. The *FLEXCAN_SetPNConfig()* function is used to configure FlexCAN Networking work mode.

Public Members

bool enableTimeout

Enable or Disable timeout event trigger wakeup.

uint16_t timeoutValue

The timeout value that generates a wakeup event, the counter timer is incremented based on 64 times the CAN Bit Time unit.

bool enableMatch

Enable or Disable match event trigger wakeup.

flexcan_pn_match_source_t matchSrc

Selects the match source (ID and/or data match) to trigger wakeup.

uint8_t matchNum

The number of times a given message must match the predefined ID and/or data before generating a wakeup event, range in 0x1 ~ 0xFF.

flexcan_pn_match_mode_t idMatchMode

The ID match type.

flexcan_pn_match_mode_t dataMatchMode

The data match type.

uint32_t idLower

The ID target values 1 which used either for ID match “equal to”, “smaller than”, “greater than” comparisons, or as the lower limit value in ID match “range detection”.

uint32_t idUpper

The ID target values 2 which used only as the upper limit value in ID match “range detection” or used to store the ID mask in “equal to”.

uint8_t lengthLower

The lower limit for length of data bytes which used only in data match “range detection”. Range in 0x0 ~ 0x8.

uint8_t lengthUpper

The upper limit for length of data bytes which used only in data match "range detection". Range in 0x0 ~ 0x8.

struct __flexcan_rx_fifo_config

#include <fsl_flexcan.h> FlexCAN Legacy Rx FIFO configuration structure.

Public Members

uint32_t *idFilterTable

Pointer to the FlexCAN Legacy Rx FIFO identifier filter table.

uint8_t idFilterNum

The FlexCAN Legacy Rx FIFO Filter elements quantity.

flexcan_rx_fifo_filter_type_t idFilterType

The FlexCAN Legacy Rx FIFO Filter type.

flexcan_rx_fifo_priority_t priority

The FlexCAN Legacy Rx FIFO receive priority.

struct __flexcan_enhanced_rx_fifo_std_id_filter

#include <fsl_flexcan.h> FlexCAN Enhanced Rx FIFO Standard ID filter element structure.

Public Members

uint32_t filterType

FlexCAN internal Free-Running Counter Time Stamp.

uint32_t rtr1

CAN FD frame data length code (DLC), range see *_flexcan_fd_frame_length*, When the length <= 8, it equal to the data length, otherwise the number of valid frame data is not equal to the length value. user can use *DLC_LENGTH_DECODE(length)* macro to get the number of valid data bytes.

uint32_t std1

CAN Frame Type(DATA or REMOTE).

uint32_t rtr2

CAN Frame Identifier(STD or EXT format).

uint32_t std2

Substitute Remote request.

struct __flexcan_enhanced_rx_fifo_ext_id_filter

#include <fsl_flexcan.h> FlexCAN Enhanced Rx FIFO Extended ID filter element structure.

Public Members

uint32_t filterType

FlexCAN internal Free-Running Counter Time Stamp.

uint32_t rtr1

CAN FD frame data length code (DLC), range see *_flexcan_fd_frame_length*, When the length <= 8, it equal to the data length, otherwise the number of valid frame data is not equal to the length value. user can use *DLC_LENGTH_DECODE(length)* macro to get the number of valid data bytes.

uint32_t std1

CAN Frame Type(DATA or REMOTE).

uint32_t rtr2

CAN Frame Identifier(STD or EXT format).

uint32_t std2

Substitute Remote request.

struct __flexcan__enhanced__rx__fifo__config

#include <fsl_flexcan.h> FlexCAN Enhanced Rx FIFO configuration structure.

Public Members

uint32_t *idFilterTable

Pointer to the FlexCAN Enhanced Rx FIFO identifier filter table, each table member occupies 32 bit word, table size should be equal to idFilterNum. There are two types of Enhanced Rx FIFO filter elements that can be stored in table : extended-ID filter element (1 word, occupie 1 table members) and standard-ID filter element (2 words, occupies 2 table members), the extended-ID filter element needs to be placed in front of the table.

uint8_t idFilterPairNum

idFilterPairNum is the Enhanced Rx FIFO identifier filter element pair numbers, each pair of filter elements occupies 2 words and can consist of one extended ID filter element or two standard ID filter elements.

uint8_t extendIdFilterNum

The number of extended ID filter element items in the FlexCAN enhanced Rx FIFO identifier filter table, each extended-ID filter element occupies 2 words, extendIdFilterNum need less than or equal to idFilterPairNum.

uint8_t fifoWatermark

(fifoWatermark + 1) is the minimum number of CAN messages stored in the Enhanced RX FIFO which can trigger FIFO watermark interrupt or a DMA request.

flexcan_efifo_dma_per_read_length_t dmaPerReadLength

Define the length of each read of the Enhanced RX FIFO element by the DAM, see *_flexcan_fd_frame_length*.

flexcan_rx_fifo_priority_t priority
The FlexCAN Enhanced Rx FIFO receive priority.

struct __flexcan_mb_transfer
#include <fsl_flexcan.h> FlexCAN Message Buffer transfer.

Public Members

flexcan_frame_t *frame
The buffer of CAN Message to be transfer.

uint8_t mbIdx
The index of Message buffer used to transfer Message.

struct __flexcan_fifo_transfer
#include <fsl_flexcan.h> FlexCAN Rx FIFO transfer.

Public Members

flexcan_fd_frame_t *framefd
The buffer of CAN Message to be received from Enhanced Rx FIFO.

flexcan_frame_t *frame
The buffer of CAN Message to be received from Legacy Rx FIFO.

size_t frameNum
Depth of CAN Message receive array of Legacy or Enhanced Rx FIFO.

struct __flexcan_handle
#include <fsl_flexcan.h> FlexCAN handle structure.

Public Members

flexcan_transfer_callback_t callback
Callback function.

void *userData
FlexCAN callback function parameter.

flexcan_frame_t *volatile mbFrameBuf[CAN_WORD1_COUNT]
The buffer for received CAN data from Message Buffers.

flexcan_fd_frame_t *volatile mbFDFrameBuf[CAN_WORD1_COUNT]
The buffer for received CAN FD data from Message Buffers.

flexcan_frame_t *volatile rxFifoFrameBuf

The buffer for received CAN data from Legacy Rx FIFO.

flexcan_fd_frame_t *volatile rxFifoFDFrameBuf

The buffer for received CAN FD data from Enhanced Rx FIFO.

size_t rxFifoFrameNum

The number of CAN messages remaining to be received from Legacy or Enhanced Rx FIFO.

size_t rxFifoTransferTotalNum

Total CAN Message number need to be received from Legacy or Enhanced Rx FIFO.

volatile uint8_t mbState[CAN_WORD1_COUNT]

Message Buffer transfer state.

volatile uint8_t rxFifoState

Rx FIFO transfer state.

volatile uint32_t timestamp[CAN_WORD1_COUNT]

Mailbox transfer timestamp.

struct byteStatus

Public Members

bool byteIsRead

The byte n (0~3) was read or not. The type of error and which bit in byte (n) is affected by the error.

struct ___unnamed14___

Public Members

uint32_t timestamp

FlexCAN internal Free-Running Counter Time Stamp.

uint32_t length

CAN frame data length in bytes (Range: 0~8).

uint32_t type

CAN Frame Type(DATA or REMOTE).

uint32_t format

CAN Frame Identifier(STD or EXT format).

uint32_t ___pad0___
Reserved.

uint32_t idhit
CAN Rx FIFO filter hit id(This value is only used in Rx FIFO receive mode).

struct ___unnamed16___

Public Members

uint32_t id
CAN Frame Identifier, should be set using *FLEXCAN_ID_EXT()* or *FLEXCAN_ID_STD()* macro.

uint32_t ___pad0___
Reserved.

union ___unnamed18___

Public Members

struct __flexcan_frame

struct __flexcan_frame

struct ___unnamed20___

Public Members

uint32_t dataWord0
CAN Frame payload word0.

uint32_t dataWord1
CAN Frame payload word1.

struct ___unnamed22___

Public Members

uint8_t dataByte3
CAN Frame payload byte3.

uint8_t dataByte2
CAN Frame payload byte2.

uint8_t dataByte1
CAN Frame payload byte1.

uint8_t dataByte0
CAN Frame payload byte0.

uint8_t dataByte7
CAN Frame payload byte7.

uint8_t dataByte6
CAN Frame payload byte6.

uint8_t dataByte5
CAN Frame payload byte5.

uint8_t dataByte4
CAN Frame payload byte4.

struct ___unnamed24___

Public Members

uint32_t timestamp
FlexCAN internal Free-Running Counter Time Stamp.

uint32_t length
CAN FD frame data length code (DLC), range see *_flexcan_fd_frame_length*, When the length <= 8, it equal to the data length, otherwise the number of valid frame data is not equal to the length value. user can use *DLC_LENGTH_DECODE(length)* macro to get the number of valid data bytes.

uint32_t type
CAN Frame Type(DATA only).

uint32_t format
CAN Frame Identifier(STD or EXT format).

uint32_t srr
Substitute Remote request.

uint32_t esi
Error State Indicator.

uint32_t brs
Bit Rate Switch.

uint32_t edl
Extended Data Length.

struct ___unnamed26___

Public Members

uint32_t id

CAN Frame Identifier, should be set using *FLEXCAN_ID_EXT()* or *FLEXCAN_ID_STD()* macro.

uint32_t ___pad0___

Reserved.

union ___unnamed28___

Public Members

struct __flexcan__fd__frame

struct __flexcan__fd__frame

struct ___unnamed30___

Public Members

uint32_t dataWord[16]

CAN FD Frame payload, 16 double word maximum.

struct ___unnamed32___

Public Members

uint8_t dataByte3

CAN Frame payload byte3.

uint8_t dataByte2

CAN Frame payload byte2.

uint8_t dataByte1

CAN Frame payload byte1.

uint8_t dataByte0

CAN Frame payload byte0.

uint8_t dataByte7

CAN Frame payload byte7.

uint8_t dataByte6
CAN Frame payload byte6.

uint8_t dataByte5
CAN Frame payload byte5.

uint8_t dataByte4
CAN Frame payload byte4.

union ___unnamed34___

Public Members

struct __flexcan__config

struct __flexcan__config

struct ___unnamed36___

Public Members

uint32_t baudRate
FlexCAN bit rate in bps, for classical CAN or CANFD nominal phase.

uint32_t baudRateFD
FlexCAN FD bit rate in bps, for CANFD data phase.

struct ___unnamed38___

Public Members

uint32_t bitRate
FlexCAN bit rate in bps, for classical CAN or CANFD nominal phase.

uint32_t bitRateFD
FlexCAN FD bit rate in bps, for CANFD data phase.

union ___unnamed40___

Public Members

struct __flexcan__pn__config

< The data target values 1 which used either for data match “equal to”, “smaller than”, “greater than” comparisons, or as the lower limit value in data match “range

detection”.

struct __flexcan_pn_config

struct ___unnamed44___

< The data target values 1 which used either for data match “equal to”, “smaller than”, “greater than” comparisons, or as the lower limit value in data match “range

detection”.

Public Members

uint32_t lowerWord0
CAN Frame payload word0.

uint32_t lowerWord1
CAN Frame payload word1.

struct ___unnamed46___

Public Members

uint8_t lowerByte3
CAN Frame payload byte3.

uint8_t lowerByte2
CAN Frame payload byte2.

uint8_t lowerByte1
CAN Frame payload byte1.

uint8_t lowerByte0
CAN Frame payload byte0.

uint8_t lowerByte7
CAN Frame payload byte7.

uint8_t lowerByte6
CAN Frame payload byte6.

uint8_t lowerByte5
CAN Frame payload byte5.

uint8_t lowerByte4
CAN Frame payload byte4.

union ___unnamed42___

Public Members

struct __flexcan__pn__config

< The data target values 2 which used only as the upper limit value in data match "range

detection" or used to store the data mask in "equal to".

struct __flexcan__pn__config

struct ___unnamed48___

< The data target values 2 which used only as the upper limit value in data match "range

detection" or used to store the data mask in "equal to".

Public Members

uint32_t upperWord0

CAN Frame payload word0.

uint32_t upperWord1

CAN Frame payload word1.

struct ___unnamed50___

Public Members

uint8_t upperByte3

CAN Frame payload byte3.

uint8_t upperByte2

CAN Frame payload byte2.

uint8_t upperByte1

CAN Frame payload byte1.

uint8_t upperByte0

CAN Frame payload byte0.

uint8_t upperByte7

CAN Frame payload byte7.

uint8_t upperByte6

CAN Frame payload byte6.

uint8_t upperByte5

CAN Frame payload byte5.

uint8_t upperByte4

CAN Frame payload byte4.

2.19 FlexCAN eDMA Driver

void FLEXCAN_TransferCreateHandleEDMA(CAN_Type *base, *flexcan_edma_handle_t* *handle, *flexcan_edma_transfer_callback_t* callback, void *userData, *edma_handle_t* *rxFifoEdmaHandle)

Initializes the FlexCAN handle, which is used in transactional functions.

Parameters

- base – FlexCAN peripheral base address.
- handle – Pointer to flexcan_edma_handle_t structure.
- callback – The callback function.
- userData – The parameter of the callback function.
- rxFifoEdmaHandle – User-requested DMA handle for Rx FIFO DMA transfer.

void FLEXCAN_PrepareTransfConfiguration(CAN_Type *base, *flexcan_fifo_transfer_t* *pFifoXfer, *edma_transfer_config_t* *pEdmaConfig)

Prepares the eDMA transfer configuration for FLEXCAN Legacy RX FIFO.

This function prepares the eDMA transfer configuration structure according to FLEXCAN Legacy RX FIFO.

Parameters

- base – FlexCAN peripheral base address.
- pFifoXfer – FlexCAN Rx FIFO EDMA transfer structure, see *flexcan_fifo_transfer_t*.
- pEdmaConfig – The user configuration structure of type edma_transfer_t.

status_t FLEXCAN_StartTransferDatafromRxFIFO(CAN_Type *base, *flexcan_edma_handle_t* *handle, *edma_transfer_config_t* *pEdmaConfig)

Start Transfer Data from the FLEXCAN Legacy Rx FIFO using eDMA.

This function to Update edma transfer confiugration and Start eDMA transfer

Parameters

- base – FlexCAN peripheral base address.
- handle – Pointer to flexcan_edma_handle_t structure.
- pEdmaConfig – The user configuration structure of type edma_transfer_t.

Return values

- kStatus_Success – if succeed, others failed.
- kStatus_FLEXCAN_RxFifoBusy – Previous transfer ongoing.

status_t FLEXCAN_TransferReceiveFifoEDMA(CAN_Type *base, *flexcan_edma_handle_t* *handle, *flexcan_fifo_transfer_t* *pFifoXfer)

Receives the CAN Message from the Legacy Rx FIFO using eDMA.

This function receives the CAN Message using eDMA. This is a non-blocking function, which returns right away. After the CAN Message is received, the receive callback function is called.

Parameters

- base – FlexCAN peripheral base address.
- handle – Pointer to flexcan_edma_handle_t structure.
- pFifoXfer – FlexCAN Rx FIFO EDMA transfer structure, see *flexcan_fifo_transfer_t*.

Return values

- kStatus_Success – if succeed, others failed.
- kStatus_FLEXCAN_RxFifoBusy – Previous transfer ongoing.

status_t FLEXCAN_TransferGetReceiveFifoCountEMDA(CAN_Type *base, *flexcan_edma_handle_t* *handle, size_t *count)

Gets the Legacy Rx Fifo transfer status during a interrupt non-blocking receive.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- count – Number of CAN messages receive so far by the non-blocking transaction.

Return values

- kStatus_InvalidArgument – count is Invalid.
- kStatus_Success – Successfully return the count.

void FLEXCAN_TransferAbortReceiveFifoEDMA(CAN_Type *base, *flexcan_edma_handle_t* *handle)

Aborts the receive Legacy/Enhanced Rx FIFO process which used eDMA.

This function aborts the receive Legacy/Enhanced Rx FIFO process which used eDMA.

Parameters

- base – FlexCAN peripheral base address.
- handle – Pointer to flexcan_edma_handle_t structure.

status_t FLEXCAN_TransferReceiveEnhancedFifoEDMA(CAN_Type *base, *flexcan_edma_handle_t* *handle, *flexcan_fifo_transfer_t* *pFifoXfer)

Receives the CAN FD Message from the Enhanced Rx FIFO using eDMA.

This function receives the CAN FD Message using eDMA. This is a non-blocking function, which returns right away. After the CAN Message is received, the receive callback function is called.

Parameters

- base – FlexCAN peripheral base address.
- handle – Pointer to flexcan_edma_handle_t structure.
- pFifoXfer – FlexCAN Rx FIFO EDMA transfer structure, see *flexcan_fifo_transfer_t*.

Return values

- kStatus_Success – if succeed, others failed.
- kStatus_FLEXCAN_RxFifoBusy – Previous transfer ongoing.

static inline *status_t* FLEXCAN_TransferGetReceiveEnhancedFifoCountEMDA(CAN_Type *base, *flexcan_edma_handle_t* *handle, size_t *count)

Gets the Enhanced Rx Fifo transfer status during a interrupt non-blocking receive.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- count – Number of CAN messages receive so far by the non-blocking transaction.

Return values

- kStatus_InvalidArgument – count is Invalid.
- kStatus_Success – Successfully return the count.

FSL_FLEXCAN_EDMA_DRIVER_VERSION

FlexCAN EDMA driver version.

typedef struct *_flexcan_edma_handle* flexcan_edma_handle_t

typedef void (*flexcan_edma_transfer_callback_t)(CAN_Type *base, *flexcan_edma_handle_t* *handle, *status_t* status, void *userData)

FlexCAN transfer callback function.

struct _flexcan_edma_handle

#include <fsl_flexcan_edma.h> FlexCAN eDMA handle.

Public Members

flexcan_edma_transfer_callback_t callback

Callback function.

void *userData

FlexCAN callback function parameter.

edma_handle_t *rxFifoEdmaHandle

The EDMA handler for Rx FIFO.

volatile uint8_t rxFifoState

Rx FIFO transfer state.

size_t frameNum

The number of messages that need to be received.

flexcan_fd_frame_t *framefd

Point to the buffer of CAN Message to be received from Enhanced Rx FIFO.

2.20 FlexIO: FlexIO Driver

2.21 FlexIO Driver

void FLEXIO_GetDefaultConfig(*flexio_config_t* *userConfig)

Gets the default configuration to configure the FlexIO module. The configuration can used directly to call the FLEXIO_Configure().

Example:

```
flexio_config_t config;
FLEXIO_GetDefaultConfig(&config);
```

Parameters

- userConfig – pointer to flexio_config_t structure

void FLEXIO_Init(FLEXIO_Type *base, const *flexio_config_t* *userConfig)

Configures the FlexIO with a FlexIO configuration. The configuration structure can be filled by the user or be set with default values by *FLEXIO_GetDefaultConfig()*.

Example

```
flexio_config_t config = {
.enableFlexio = true,
.enableInDoze = false,
.enableInDebug = true,
.enableFastAccess = false
};
FLEXIO_Configure(base, &config);
```

Parameters

- base – FlexIO peripheral base address
- userConfig – pointer to flexio_config_t structure

void FLEXIO_Deinit(FLEXIO_Type *base)

Gates the FlexIO clock. Call this API to stop the FlexIO clock.

Note: After calling this API, call the FLEXO_Init to use the FlexIO module.

Parameters

- base – FlexIO peripheral base address

uint32_t FLEXIO_GetInstance(FLEXIO_Type *base)

Get instance number for FLEXIO module.

Parameters

- base – FLEXIO peripheral base address.

void FLEXIO_Reset(FLEXIO_Type *base)

Resets the FlexIO module.

Parameters

- base – FlexIO peripheral base address

static inline void FLEXIO_Enable(FLEXIO_Type *base, bool enable)

Enables the FlexIO module operation.

Parameters

- base – FlexIO peripheral base address
- enable – true to enable, false to disable.

static inline uint32_t FLEXIO_ReadPinInput(FLEXIO_Type *base)

Reads the input data on each of the FlexIO pins.

Parameters

- base – FlexIO peripheral base address

Returns

FlexIO pin input data

static inline uint8_t FLEXIO_GetShifterState(FLEXIO_Type *base)

Gets the current state pointer for state mode use.

Parameters

- base – FlexIO peripheral base address

Returns

current State pointer

void FLEXIO_SetShifterConfig(FLEXIO_Type *base, uint8_t index, const *flexio_shifter_config_t* *shifterConfig)

Configures the shifter with the shifter configuration. The configuration structure covers both the SHIFTCTL and SHIFTCFG registers. To configure the shifter to the proper mode, select which timer controls the shifter to shift, whether to generate start bit/stop bit, and the polarity of start bit and stop bit.

Example

```
flexio_shifter_config_t config = {
.timerSelect = 0,
.timerPolarity = kFLEXIO_ShifterTimerPolarityOnPositive,
.pinConfig = kFLEXIO_PinConfigOpenDrainOrBidirection,
.pinPolarity = kFLEXIO_PinActiveLow,
.shifterMode = kFLEXIO_ShifterModeTransmit,
.inputSource = kFLEXIO_ShifterInputFromPin,
.shifterStop = kFLEXIO_ShifterStopBitHigh,
.shifterStart = kFLEXIO_ShifterStartBitLow
};
FLEXIO_SetShifterConfig(base, &config);
```

Parameters

- base – FlexIO peripheral base address
- index – Shifter index
- shifterConfig – Pointer to flexio_shifter_config_t structure

void FLEXIO_SetTimerConfig(FLEXIO_Type *base, uint8_t index, const *flexio_timer_config_t* *timerConfig)

Configures the timer with the timer configuration. The configuration structure covers both the TIMCTL and TIMCFG registers. To configure the timer to the proper mode, select trigger source for timer and the timer pin output and the timing for timer.

Example

```
flexio_timer_config_t config = {
.triggerSelect = FLEXIO_TIMER_TRIGGER_SEL_SHIFTnSTAT(0),
.triggerPolarity = kFLEXIO_TimerTriggerPolarityActiveLow,
.triggerSource = kFLEXIO_TimerTriggerSourceInternal,
.pinConfig = kFLEXIO_PinConfigOpenDrainOrBidirection,
.pinSelect = 0,
.pinPolarity = kFLEXIO_PinActiveHigh,
.timerMode = kFLEXIO_TimerModeDual8BitBaudBit,
.timerOutput = kFLEXIO_TimerOutputZeroNotAffectedByReset,
.timerDecrement = kFLEXIO_TimerDecSrcOnFlexIOClockShiftTimerOutput,
.timerReset = kFLEXIO_TimerResetOnTimerPinEqualToTimerOutput,
.timerDisable = kFLEXIO_TimerDisableOnTimerCompare,
.timerEnable = kFLEXIO_TimerEnableOnTriggerHigh,
.timerStop = kFLEXIO_TimerStopBitEnableOnTimerDisable,
.timerStart = kFLEXIO_TimerStartBitEnabled
};
FLEXIO_SetTimerConfig(base, &config);
```

Parameters

- base – FlexIO peripheral base address
- index – Timer index
- timerConfig – Pointer to the flexio_timer_config_t structure

static inline void FLEXIO_SetClockMode(FLEXIO_Type *base, uint8_t index, *flexio_timer_decrement_source_t* clocksource)

This function set the value of the prescaler on flexio channels.

Parameters

- base – Pointer to the FlexIO simulated peripheral type.
- index – Timer index
- clocksource – Set clock value

static inline void FLEXIO_EnableShifterStatusInterrupts(FLEXIO_Type *base, uint32_t mask)

Enables the shifter status interrupt. The interrupt generates when the corresponding SSF is set.

Note: For multiple shifter status interrupt enable, for example, two shifter status enable, can calculate the mask by using ((1 « shifter index0) | (1 « shifter index1))

Parameters

- base – FlexIO peripheral base address
- mask – The shifter status mask which can be calculated by (1 « shifter index)

static inline void FLEXIO_DisableShifterStatusInterrupts(FLEXIO_Type *base, uint32_t mask)

Disables the shifter status interrupt. The interrupt won't generate when the corresponding SSF is set.

Note: For multiple shifter status interrupt enable, for example, two shifter status enable, can calculate the mask by using ((1 « shifter index0) | (1 « shifter index1))

Parameters

- base – FlexIO peripheral base address
- mask – The shifter status mask which can be calculated by (1 « shifter index)

static inline void FLEXIO_EnableShifterErrorInterrupts(FLEXIO_Type *base, uint32_t mask)

Enables the shifter error interrupt. The interrupt generates when the corresponding SEF is set.

Note: For multiple shifter error interrupt enable, for example, two shifter error enable, can calculate the mask by using ((1 « shifter index0) | (1 « shifter index1))

Parameters

- base – FlexIO peripheral base address
- mask – The shifter error mask which can be calculated by (1 « shifter index)

static inline void FLEXIO_DisableShifterErrorInterrupts(FLEXIO_Type *base, uint32_t mask)

Disables the shifter error interrupt. The interrupt won't generate when the corresponding SEF is set.

Note: For multiple shifter error interrupt enable, for example, two shifter error enable, can calculate the mask by using ((1 « shifter index0) | (1 « shifter index1))

Parameters

- base – FlexIO peripheral base address
- mask – The shifter error mask which can be calculated by (1 « shifter index)

static inline void FLEXIO_EnableTimerStatusInterrupts(FLEXIO_Type *base, uint32_t mask)

Enables the timer status interrupt. The interrupt generates when the corresponding SSF is set.

Note: For multiple timer status interrupt enable, for example, two timer status enable, can calculate the mask by using ((1 « timer index0) | (1 « timer index1))

Parameters

- base – FlexIO peripheral base address
- mask – The timer status mask which can be calculated by (1 « timer index)

static inline void FLEXIO_DisableTimerStatusInterrupts(FLEXIO_Type *base, uint32_t mask)

Disables the timer status interrupt. The interrupt won't generate when the corresponding SSF is set.

Note: For multiple timer status interrupt enable, for example, two timer status enable, can calculate the mask by using ((1 « timer index0) | (1 « timer index1))

Parameters

- base – FlexIO peripheral base address
- mask – The timer status mask which can be calculated by (1 « timer index)

static inline uint32_t FLEXIO_GetShifterStatusFlags(FLEXIO_Type *base)

Gets the shifter status flags.

Parameters

- base – FlexIO peripheral base address

Returns

Shifter status flags

static inline void FLEXIO_ClearShifterStatusFlags(FLEXIO_Type *base, uint32_t mask)

Clears the shifter status flags.

Note: For clearing multiple shifter status flags, for example, two shifter status flags, can calculate the mask by using ((1 « shifter index0) | (1 « shifter index1))

Parameters

- base – FlexIO peripheral base address
- mask – The shifter status mask which can be calculated by (1 « shifter index)

static inline uint32_t FLEXIO_GetShifterErrorFlags(FLEXIO_Type *base)

Gets the shifter error flags.

Parameters

- base – FlexIO peripheral base address

Returns

Shifter error flags

static inline void FLEXIO_ClearShifterErrorFlags(FLEXIO_Type *base, uint32_t mask)

Clears the shifter error flags.

Note: For clearing multiple shifter error flags, for example, two shifter error flags, can calculate the mask by using ((1 « shifter index0) | (1 « shifter index1))

Parameters

- base – FlexIO peripheral base address
- mask – The shifter error mask which can be calculated by (1 « shifter index)

static inline uint32_t FLEXIO_GetTimerStatusFlags(FLEXIO_Type *base)

Gets the timer status flags.

Parameters

- base – FlexIO peripheral base address

Returns

Timer status flags

static inline void FLEXIO_ClearTimerStatusFlags(FLEXIO_Type *base, uint32_t mask)

Clears the timer status flags.

Note: For clearing multiple timer status flags, for example, two timer status flags, can calculate the mask by using ((1 « timer index0) | (1 « timer index1))

Parameters

- base – FlexIO peripheral base address
- mask – The timer status mask which can be calculated by (1 « timer index)

static inline void FLEXIO_EnableShifterStatusDMA(FLEXIO_Type *base, uint32_t mask, bool enable)

Enables/disables the shifter status DMA. The DMA request generates when the corresponding SSF is set.

Note: For multiple shifter status DMA enables, for example, calculate the mask by using ((1 « shifter index0) | (1 « shifter index1))

Parameters

- base – FlexIO peripheral base address
- mask – The shifter status mask which can be calculated by (1 « shifter index)
- enable – True to enable, false to disable.

uint32_t FLEXIO_GetShifterBufferAddress(FLEXIO_Type *base, *flexio_shifter_buffer_type_t* type, uint8_t index)

Gets the shifter buffer address for the DMA transfer usage.

Parameters

- base – FlexIO peripheral base address
- type – Shifter type of flexio_shifter_buffer_type_t
- index – Shifter index

Returns

Corresponding shifter buffer index

status_t FLEXIO_RegisterHandleIRQ(void *base, void *handle, *flexio_isr_t* isr)

Registers the handle and the interrupt handler for the FlexIO-simulated peripheral.

Parameters

- base – Pointer to the FlexIO simulated peripheral type.
- handle – Pointer to the handler for FlexIO simulated peripheral.
- isr – FlexIO simulated peripheral interrupt handler.

Return values

- kStatus_Success – Successfully create the handle.
- kStatus_OutOfRange – The FlexIO type/handle/ISR table out of range.

status_t FLEXIO_UnregisterHandleIRQ(void *base)

Unregisters the handle and the interrupt handler for the FlexIO-simulated peripheral.

Parameters

- base – Pointer to the FlexIO simulated peripheral type.

Return values

- kStatus_Success – Successfully create the handle.
- kStatus_OutOfRange – The FlexIO type/handle/ISR table out of range.

static inline void FLEXIO_ClearPortOutput(FLEXIO_Type *base, uint32_t mask)

Sets the output level of the multiple FLEXIO pins to the logic 0.

Parameters

- base – FlexIO peripheral base address
- mask – FLEXIO pin number mask

static inline void FLEXIO_SetPortOutput(FLEXIO_Type *base, uint32_t mask)

Sets the output level of the multiple FLEXIO pins to the logic 1.

Parameters

- base – FlexIO peripheral base address
- mask – FLEXIO pin number mask

static inline void FLEXIO_TogglePortOutput(FLEXIO_Type *base, uint32_t mask)

Reverses the current output logic of the multiple FLEXIO pins.

Parameters

- base – FlexIO peripheral base address
- mask – FLEXIO pin number mask

static inline void FLEXIO_PinWrite(FLEXIO_Type *base, uint32_t pin, uint8_t output)

Sets the output level of the FLEXIO pins to the logic 1 or 0.

Parameters

- base – FlexIO peripheral base address
- pin – FLEXIO pin number.
- output – FLEXIO pin output logic level.
  - 0: corresponding pin output low-logic level.
  - 1: corresponding pin output high-logic level.

static inline void FLEXIO_EnablePinOutput(FLEXIO_Type *base, uint32_t pin)

Enables the FLEXIO output pin function.

Parameters

- base – FlexIO peripheral base address
- pin – FLEXIO pin number.

static inline uint32_t FLEXIO_PinRead(FLEXIO_Type *base, uint32_t pin)

Reads the current input value of the FLEXIO pin.

Parameters

- base – FlexIO peripheral base address
- pin – FLEXIO pin number.

Return values

FLEXIO – port input value

- 0: corresponding pin input low-logic level.
- 1: corresponding pin input high-logic level.

static inline uint32_t FLEXIO_GetPinStatus(FLEXIO_Type *base, uint32_t pin)

Gets the FLEXIO input pin status.

Parameters

- base – FlexIO peripheral base address
- pin – FLEXIO pin number.

Return values

FLEXIO – port input status

- 0: corresponding pin input capture no status.
- 1: corresponding pin input capture rising or falling edge.

static inline void FLEXIO_SetPinLevel(FLEXIO_Type *base, uint8_t pin, bool level)

Sets the FLEXIO output pin level.

Parameters

- base – FlexIO peripheral base address
- pin – FlexIO pin number.
- level – FlexIO output pin level to set, can be either 0 or 1.

static inline bool FLEXIO_GetPinOverride(const FLEXIO_Type *const base, uint8_t pin)

Gets the enabled status of a FLEXIO output pin.

Parameters

- base – FlexIO peripheral base address
- pin – FlexIO pin number.

Return values

FlexIO – port enabled status

- 0: corresponding output pin is in disabled state.
- 1: corresponding output pin is in enabled state.

static inline void FLEXIO_ConfigPinOverride(FLEXIO_Type *base, uint8_t pin, bool enabled)

Enables or disables a FLEXIO output pin.

Parameters

- base – FlexIO peripheral base address
- pin – Flexio pin number.
- enabled – Enable or disable the FlexIO pin.

static inline void FLEXIO_ClearPortStatus(FLEXIO_Type *base, uint32_t mask)

Clears the multiple FLEXIO input pins status.

Parameters

- base – FlexIO peripheral base address
- mask – FLEXIO pin number mask

FSL_FLEXIO_DRIVER_VERSION

FlexIO driver version.

enum _flexio_timer_trigger_polarity

Define time of timer trigger polarity.

Values:

enumerator kFLEXIO_TimerTriggerPolarityActiveHigh

Active high.

enumerator kFLEXIO_TimerTriggerPolarityActiveLow
Active low.

enum _flexio_timer_trigger_source
Define type of timer trigger source.
Values:

enumerator kFLEXIO_TimerTriggerSourceExternal
External trigger selected.

enumerator kFLEXIO_TimerTriggerSourceInternal
Internal trigger selected.

enum _flexio_pin_config
Define type of timer/shifter pin configuration.
Values:

enumerator kFLEXIO_PinConfigOutputDisabled
Pin output disabled.

enumerator kFLEXIO_PinConfigOpenDrainOrBidirection
Pin open drain or bidirectional output enable.

enumerator kFLEXIO_PinConfigBidirectionOutputData
Pin bidirectional output data.

enumerator kFLEXIO_PinConfigOutput
Pin output.

enum _flexio_pin_polarity
Definition of pin polarity.
Values:

enumerator kFLEXIO_PinActiveHigh
Active high.

enumerator kFLEXIO_PinActiveLow
Active low.

enum _flexio_timer_mode
Define type of timer work mode.
Values:

enumerator kFLEXIO_TimerModeDisabled
Timer Disabled.

enumerator kFLEXIO_TimerModeDual8BitBaudBit
Dual 8-bit counters baud/bit mode.

enumerator kFLEXIO_TimerModeDual8BitPWM
Dual 8-bit counters PWM mode.

enumerator kFLEXIO_TimerModeSingle16Bit
Single 16-bit counter mode.

enumerator kFLEXIO_TimerModeDual8BitPWMLow
Dual 8-bit counters PWM Low mode.

enum _flexio_timer_output
Define type of timer initial output or timer reset condition.
Values:

enumerator kFLEXIO_TimerOutputOneNotAffectedByReset
Logic one when enabled and is not affected by timer reset.

enumerator kFLEXIO_TimerOutputZeroNotAffectedByReset
Logic zero when enabled and is not affected by timer reset.

enumerator kFLEXIO_TimerOutputOneAffectedByReset
Logic one when enabled and on timer reset.

enumerator kFLEXIO_TimerOutputZeroAffectedByReset
Logic zero when enabled and on timer reset.

enum _flexio_timer_decrement_source
Define type of timer decrement.
Values:

enumerator kFLEXIO_TimerDecSrcOnFlexIOClockShiftTimerOutput
Decrement counter on FlexIO clock, Shift clock equals Timer output.

enumerator kFLEXIO_TimerDecSrcOnTriggerInputShiftTimerOutput
Decrement counter on Trigger input (both edges), Shift clock equals Timer output.

enumerator kFLEXIO_TimerDecSrcOnPinInputShiftPinInput
Decrement counter on Pin input (both edges), Shift clock equals Pin input.

enumerator kFLEXIO_TimerDecSrcOnTriggerInputShiftTriggerInput
Decrement counter on Trigger input (both edges), Shift clock equals Trigger input.

enum _flexio_timer_reset_condition
Define type of timer reset condition.
Values:

enumerator kFLEXIO_TimerResetNever
Timer never reset.

enumerator kFLEXIO_TimerResetOnTimerPinEqualToTimerOutput
Timer reset on Timer Pin equal to Timer Output.

enumerator kFLEXIO_TimerResetOnTimerTriggerEqualToTimerOutput
Timer reset on Timer Trigger equal to Timer Output.

enumerator kFLEXIO_TimerResetOnTimerPinRisingEdge
Timer reset on Timer Pin rising edge.

enumerator kFLEXIO_TimerResetOnTimerTriggerRisingEdge
Timer reset on Trigger rising edge.

enumerator kFLEXIO_TimerResetOnTimerTriggerBothEdge
Timer reset on Trigger rising or falling edge.

enum _flexio_timer_disable_condition
Define type of timer disable condition.

Values:

enumerator kFLEXIO_TimerDisableNever
Timer never disabled.

enumerator kFLEXIO_TimerDisableOnPreTimerDisable
Timer disabled on Timer N-1 disable.

enumerator kFLEXIO_TimerDisableOnTimerCompare
Timer disabled on Timer compare.

enumerator kFLEXIO_TimerDisableOnTimerCompareTriggerLow
Timer disabled on Timer compare and Trigger Low.

enumerator kFLEXIO_TimerDisableOnPinBothEdge
Timer disabled on Pin rising or falling edge.

enumerator kFLEXIO_TimerDisableOnPinBothEdgeTriggerHigh
Timer disabled on Pin rising or falling edge provided Trigger is high.

enumerator kFLEXIO_TimerDisableOnTriggerFallingEdge
Timer disabled on Trigger falling edge.

enum _flexio_timer_enable_condition
Define type of timer enable condition.

Values:

enumerator kFLEXIO_TimerEnabledAlways
Timer always enabled.

enumerator kFLEXIO_TimerEnableOnPrevTimerEnable
Timer enabled on Timer N-1 enable.

enumerator kFLEXIO_TimerEnableOnTriggerHigh
Timer enabled on Trigger high.

enumerator kFLEXIO_TimerEnableOnTriggerHighPinHigh
Timer enabled on Trigger high and Pin high.

enumerator kFLEXIO_TimerEnableOnPinRisingEdge
Timer enabled on Pin rising edge.

enumerator kFLEXIO_TimerEnableOnPinRisingEdgeTriggerHigh
Timer enabled on Pin rising edge and Trigger high.

enumerator kFLEXIO_TimerEnableOnTriggerRisingEdge
Timer enabled on Trigger rising edge.

enumerator kFLEXIO_TimerEnableOnTriggerBothEdge
Timer enabled on Trigger rising or falling edge.

enum _flexio_timer_stop_bit_condition
Define type of timer stop bit generate condition.
Values:

enumerator kFLEXIO_TimerStopBitDisabled
Stop bit disabled.

enumerator kFLEXIO_TimerStopBitEnableOnTimerCompare
Stop bit is enabled on timer compare.

enumerator kFLEXIO_TimerStopBitEnableOnTimerDisable
Stop bit is enabled on timer disable.

enumerator kFLEXIO_TimerStopBitEnableOnTimerCompareDisable
Stop bit is enabled on timer compare and timer disable.

enum _flexio_timer_start_bit_condition
Define type of timer start bit generate condition.
Values:

enumerator kFLEXIO_TimerStartBitDisabled
Start bit disabled.

enumerator kFLEXIO_TimerStartBitEnabled
Start bit enabled.

enum _flexio_timer_output_state

FlexIO as PWM channel output state.

Values:

enumerator kFLEXIO_PwmLow

The output state of PWM channel is low

enumerator kFLEXIO_PwmHigh

The output state of PWM channel is high

enum _flexio_shifter_timer_polarity

Define type of timer polarity for shifter control.

Values:

enumerator kFLEXIO_ShifterTimerPolarityOnPositive

Shift on positive edge of shift clock.

enumerator kFLEXIO_ShifterTimerPolarityOnNegitive

Shift on negative edge of shift clock.

enum _flexio_shifter_mode

Define type of shifter working mode.

Values:

enumerator kFLEXIO_ShifterDisabled

Shifter is disabled.

enumerator kFLEXIO_ShifterModeReceive

Receive mode.

enumerator kFLEXIO_ShifterModeTransmit

Transmit mode.

enumerator kFLEXIO_ShifterModeMatchStore

Match store mode.

enumerator kFLEXIO_ShifterModeMatchContinuous

Match continuous mode.

enumerator kFLEXIO_ShifterModeState

SHIFTBUF contents are used for storing programmable state attributes.

enumerator kFLEXIO_ShifterModeLogic

SHIFTBUF contents are used for implementing programmable logic look up table.

enum _flexio_shifter_input_source

Define type of shifter input source.

Values:

enumerator kFLEXIO_ShifterInputFromPin
Shifter input from pin.

enumerator kFLEXIO_ShifterInputFromNextShifterOutput
Shifter input from Shifter N+1.

enum _flexio_shifter_stop_bit
Define of STOP bit configuration.
Values:

enumerator kFLEXIO_ShifterStopBitDisable
Disable shifter stop bit.

enumerator kFLEXIO_ShifterStopBitLow
Set shifter stop bit to logic low level.

enumerator kFLEXIO_ShifterStopBitHigh
Set shifter stop bit to logic high level.

enum _flexio_shifter_start_bit
Define type of START bit configuration.
Values:

enumerator kFLEXIO_ShifterStartBitDisabledLoadDataOnEnable
Disable shifter start bit, transmitter loads data on enable.

enumerator kFLEXIO_ShifterStartBitDisabledLoadDataOnShift
Disable shifter start bit, transmitter loads data on first shift.

enumerator kFLEXIO_ShifterStartBitLow
Set shifter start bit to logic low level.

enumerator kFLEXIO_ShifterStartBitHigh
Set shifter start bit to logic high level.

enum _flexio_shifter_buffer_type
Define FlexIO shifter buffer type.
Values:

enumerator kFLEXIO_ShifterBuffer
Shifter Buffer N Register.

enumerator kFLEXIO_ShifterBufferBitSwapped
Shifter Buffer N Bit Byte Swapped Register.

enumerator kFLEXIO_ShifterBufferByteSwapped
Shifter Buffer N Byte Swapped Register.

enumerator kFLEXIO_ShifterBufferBitByteSwapped
Shifter Buffer N Bit Swapped Register.

enumerator kFLEXIO_ShifterBufferNibbleByteSwapped
Shifter Buffer N Nibble Byte Swapped Register.

enumerator kFLEXIO_ShifterBufferHalfWordSwapped
Shifter Buffer N Half Word Swapped Register.

enumerator kFLEXIO_ShifterBufferNibbleSwapped
Shifter Buffer N Nibble Swapped Register.

enum _flexio_gpio_direction
FLEXIO gpio direction definition.
Values:

enumerator kFLEXIO_DigitalInput
Set current pin as digital input

enumerator kFLEXIO_DigitalOutput
Set current pin as digital output

enum _flexio_pin_input_config
FLEXIO gpio input config.
Values:

enumerator kFLEXIO_InputInterruptDisabled
Interrupt request is disabled.

enumerator kFLEXIO_InputInterruptEnable
Interrupt request is enable.

enumerator kFLEXIO_FlagRisingEdgeEnable
Input pin flag on rising edge.

enumerator kFLEXIO_FlagFallingEdgeEnable
Input pin flag on falling edge.

typedef enum *_flexio_timer_trigger_polarity* flexio_timer_trigger_polarity_t
Define time of timer trigger polarity.

typedef enum *_flexio_timer_trigger_source* flexio_timer_trigger_source_t
Define type of timer trigger source.

typedef enum *_flexio_pin_config* flexio_pin_config_t
Define type of timer/shifter pin configuration.

typedef enum *_flexio_pin_polarity* flexio_pin_polarity_t
Definition of pin polarity.

typedef enum *_flexio_timer_mode* flexio_timer_mode_t
Define type of timer work mode.

typedef enum *_flexio_timer_output* flexio_timer_output_t
Define type of timer initial output or timer reset condition.

typedef enum *_flexio_timer_decrement_source* flexio_timer_decrement_source_t
Define type of timer decrement.

typedef enum *_flexio_timer_reset_condition* flexio_timer_reset_condition_t
Define type of timer reset condition.

typedef enum *_flexio_timer_disable_condition* flexio_timer_disable_condition_t
Define type of timer disable condition.

typedef enum *_flexio_timer_enable_condition* flexio_timer_enable_condition_t
Define type of timer enable condition.

typedef enum *_flexio_timer_stop_bit_condition* flexio_timer_stop_bit_condition_t
Define type of timer stop bit generate condition.

typedef enum *_flexio_timer_start_bit_condition* flexio_timer_start_bit_condition_t
Define type of timer start bit generate condition.

typedef enum *_flexio_timer_output_state* flexio_timer_output_state_t
FlexIO as PWM channel output state.

typedef enum *_flexio_shifter_timer_polarity* flexio_shifter_timer_polarity_t
Define type of timer polarity for shifter control.

typedef enum *_flexio_shifter_mode* flexio_shifter_mode_t
Define type of shifter working mode.

typedef enum *_flexio_shifter_input_source* flexio_shifter_input_source_t
Define type of shifter input source.

typedef enum *_flexio_shifter_stop_bit* flexio_shifter_stop_bit_t
Define of STOP bit configuration.

typedef enum *_flexio_shifter_start_bit* flexio_shifter_start_bit_t
Define type of START bit configuration.

typedef enum *_flexio_shifter_buffer_type* flexio_shifter_buffer_type_t
Define FlexIO shifter buffer type.

typedef struct *_flexio_config_* flexio_config_t

Define FlexIO user configuration structure.

typedef struct *_flexio_timer_config* flexio_timer_config_t

Define FlexIO timer configuration structure.

typedef struct *_flexio_shifter_config* flexio_shifter_config_t

Define FlexIO shifter configuration structure.

typedef enum *_flexio_gpio_direction* flexio_gpio_direction_t

FLEXIO gpio direction definition.

typedef enum *_flexio_pin_input_config* flexio_pin_input_config_t

FLEXIO gpio input config.

typedef struct *_flexio_gpio_config* flexio_gpio_config_t

The FLEXIO pin configuration structure.

Each pin can only be configured as either an output pin or an input pin at a time. If configured as an input pin, use inputConfig param. If configured as an output pin, use outputLogic.

typedef void (*flexio_isr_t)(void *base, void *handle)

typedef for FlexIO simulated driver interrupt handler.

FLEXIO_Type *const s_flexioBases[]

Pointers to flexio bases for each instance.

const *clock_ip_name_t* s_flexioClocks[]

Pointers to flexio clocks for each instance.

void FLEXIO_SetPinConfig(FLEXIO_Type *base, uint32_t pin, *flexio_gpio_config_t* *config)

Configure a FLEXIO pin used by the board.

To Config the FLEXIO PIN, define a pin configuration, as either input or output, in the user file. Then, call the *FLEXIO_SetPinConfig()* function.

This is an example to define an input pin or an output pin configuration.

```
Define a digital input pin configuration,
flexio_gpio_config_t config =
{
  kFLEXIO_DigitalInput,
  0U,
  kFLEXIO_FlagRisingEdgeEnable | kFLEXIO_InputInterruptEnable,
}
Define a digital output pin configuration,
flexio_gpio_config_t config =
{
  kFLEXIO_DigitalOutput,
  0U,
  0U
}
```

Parameters

- base – FlexIO peripheral base address
- pin – FLEXIO pin number.
- config – FLEXIO pin configuration pointer.

FLEXIO_TIMER_TRIGGER_SEL_PININPUT(x)

Calculate FlexIO timer trigger.

FLEXIO_TIMER_TRIGGER_SEL_SHIFTnSTAT(x)

FLEXIO_TIMER_TRIGGER_SEL_TIMn(x)

struct _flexio_config_

#include <fsl_flexio.h> Define FlexIO user configuration structure.

Public Members

bool enableFlexio

Enable/disable FlexIO module

bool enableInDoze

Enable/disable FlexIO operation in doze mode

bool enableInDebug

Enable/disable FlexIO operation in debug mode

bool enableFastAccess

Enable/disable fast access to FlexIO registers, fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.

struct _flexio_timer_config

#include <fsl_flexio.h> Define FlexIO timer configuration structure.

Public Members

uint32_t triggerSelect

The internal trigger selection number using MACROs.

flexio_timer_trigger_polarity_t triggerPolarity

Trigger Polarity.

flexio_timer_trigger_source_t triggerSource

Trigger Source, internal (see 'trgsel') or external.

flexio_pin_config_t pinConfig

Timer Pin Configuration.

uint32_t pinSelect

Timer Pin number Select.

flexio_pin_polarity_t pinPolarity

Timer Pin Polarity.

flexio_timer_mode_t timerMode

Timer work Mode.

flexio_timer_output_t timerOutput

Configures the initial state of the Timer Output and whether it is affected by the Timer reset.

flexio_timer_decrement_source_t timerDecrement

Configures the source of the Timer decrement and the source of the Shift clock.

flexio_timer_reset_condition_t timerReset

Configures the condition that causes the timer counter (and optionally the timer output) to be reset.

flexio_timer_disable_condition_t timerDisable

Configures the condition that causes the Timer to be disabled and stop decrementing.

flexio_timer_enable_condition_t timerEnable

Configures the condition that causes the Timer to be enabled and start decrementing.

flexio_timer_stop_bit_condition_t timerStop

Timer STOP Bit generation.

flexio_timer_start_bit_condition_t timerStart

Timer STRAT Bit generation.

uint32_t timerCompare

Value for Timer Compare N Register.

struct __flexio__shifter__config

#include <fsl_flexio.h> Define FlexIO shifter configuration structure.

Public Members

uint32_t timerSelect

Selects which Timer is used for controlling the logic/shift register and generating the Shift clock.

flexio_shifter_timer_polarity_t timerPolarity

Timer Polarity.

flexio_pin_config_t pinConfig

Shifter Pin Configuration.

uint32_t pinSelect

Shifter Pin number Select.

flexio_pin_polarity_t pinPolarity

Shifter Pin Polarity.

flexio_shifter_mode_t shifterMode

Configures the mode of the Shifter.

uint32_t parallelWidth

Configures the parallel width when using parallel mode.

flexio_shifter_input_source_t inputSource

Selects the input source for the shifter.

flexio_shifter_stop_bit_t shifterStop

Shifter STOP bit.

flexio_shifter_start_bit_t shifterStart

Shifter START bit.

struct __flexio_gpio_config

#include <fsl_flexio.h> The FLEXIO pin configuration structure.

Each pin can only be configured as either an output pin or an input pin at a time. If configured as an input pin, use inputConfig param. If configured as an output pin, use outputLogic.

Public Members

flexio_gpio_direction_t pinDirection

FLEXIO pin direction, input or output

uint8_t outputLogic

Set a default output logic, which has no use in input

uint8_t inputConfig

Set an input config

2.22 FlexIO eDMA SPI Driver

status_t FLEXIO_SPI_MasterTransferCreateHandleEDMA(*FLEXIO_SPI_Type* *base, *flexio_spi_master_edma_handle_t* *handle, *flexio_spi_master_edma_transfer_callback_t* callback, void *userData, *edma_handle_t* *txHandle, *edma_handle_t* *rxHandle)

Initializes the FlexIO SPI master eDMA handle.

This function initializes the FlexIO SPI master eDMA handle which can be used for other FlexIO SPI master transactional APIs. For a specified FlexIO SPI instance, call this API once to get the initialized handle.

Parameters

- base – Pointer to FLEXIO_SPI_Type structure.
- handle – Pointer to flexio_spi_master_edma_handle_t structure to store the transfer state.
- callback – SPI callback, NULL means no callback.
- userData – callback function parameter.
- txHandle – User requested eDMA handle for FlexIO SPI RX eDMA transfer.
- rxHandle – User requested eDMA handle for FlexIO SPI TX eDMA transfer.

Return values

- kStatus_Success – Successfully create the handle.
- kStatus_OutOfRange – The FlexIO SPI eDMA type/handle table out of range.

status_t FLEXIO_SPI_MasterTransferEDMA(*FLEXIO_SPI_Type* *base, *flexio_spi_master_edma_handle_t* *handle, *flexio_spi_transfer_t* *xfer)

Performs a non-blocking FlexIO SPI transfer using eDMA.

Note: This interface returns immediately after transfer initiates. Call FLEXIO_SPI_MasterGetTransferCountEDMA to poll the transfer status and check whether the FlexIO SPI transfer is finished.

Parameters

- base – Pointer to FLEXIO_SPI_Type structure.
- handle – Pointer to flexio_spi_master_edma_handle_t structure to store the transfer state.
- xfer – Pointer to FlexIO SPI transfer structure.

Return values

- kStatus_Success – Successfully start a transfer.
- kStatus_InvalidArgument – Input argument is invalid.
- kStatus_FLEXIO_SPI_Busy – FlexIO SPI is not idle, is running another transfer.

void FLEXIO_SPI_MasterTransferAbortEDMA(*FLEXIO_SPI_Type* *base, *flexio_spi_master_edma_handle_t* *handle)

Aborts a FlexIO SPI transfer using eDMA.

Parameters

- base – Pointer to FLEXIO_SPI_Type structure.
- handle – FlexIO SPI eDMA handle pointer.

status_t FLEXIO_SPI_MasterTransferGetCountEDMA(*FLEXIO_SPI_Type* *base, *flexio_spi_master_edma_handle_t* *handle, size_t *count)

Gets the number of bytes transferred so far using FlexIO SPI master eDMA.

Parameters

- base – Pointer to FLEXIO_SPI_Type structure.
- handle – FlexIO SPI eDMA handle pointer.
- count – Number of bytes transferred so far by the non-blocking transaction.

static inline void FLEXIO_SPI_SlaveTransferCreateHandleEDMA(*FLEXIO_SPI_Type* *base, *flexio_spi_slave_edma_handle_t* *handle, *flexio_spi_slave_edma_transfer_callback_t* callback, void *userData, *edma_handle_t* *txHandle, *edma_handle_t* *rxHandle)

Initializes the FlexIO SPI slave eDMA handle.

This function initializes the FlexIO SPI slave eDMA handle.

Parameters

- base – Pointer to FLEXIO_SPI_Type structure.
- handle – Pointer to flexio_spi_slave_edma_handle_t structure to store the transfer state.
- callback – SPI callback, NULL means no callback.
- userData – callback function parameter.
- txHandle – User requested eDMA handle for FlexIO SPI TX eDMA transfer.
- rxHandle – User requested eDMA handle for FlexIO SPI RX eDMA transfer.

status_t FLEXIO_SPI_SlaveTransferEDMA(*FLEXIO_SPI_Type* *base, *flexio_spi_slave_edma_handle_t* *handle, *flexio_spi_transfer_t* *xfer)

Performs a non-blocking FlexIO SPI transfer using eDMA.

Note: This interface returns immediately after transfer initiates. Call FLEXIO_SPI_SlaveGetTransferCountEDMA to poll the transfer status and check whether the FlexIO SPI transfer is finished.

Parameters

- base – Pointer to FLEXIO_SPI_Type structure.
- handle – Pointer to flexio_spi_slave_edma_handle_t structure to store the transfer state.
- xfer – Pointer to FlexIO SPI transfer structure.

Return values

- kStatus_Success – Successfully start a transfer.
- kStatus_InvalidArgument – Input argument is invalid.
- kStatus_FLEXIO_SPI_Busy – FlexIO SPI is not idle, is running another transfer.

static inline void FLEXIO_SPI_SlaveTransferAbortEDMA(*FLEXIO_SPI_Type* *base, *flexio_spi_slave_edma_handle_t* *handle)

Aborts a FlexIO SPI transfer using eDMA.

Parameters

- base – Pointer to FLEXIO_SPI_Type structure.
- handle – Pointer to flexio_spi_slave_edma_handle_t structure to store the transfer state.

static inline *status_t* FLEXIO_SPI_SlaveTransferGetCountEDMA(*FLEXIO_SPI_Type* *base, *flexio_spi_slave_edma_handle_t* *handle, size_t *count)

Gets the number of bytes transferred so far using FlexIO SPI slave eDMA.

Parameters

- base – Pointer to FLEXIO_SPI_Type structure.
- handle – FlexIO SPI eDMA handle pointer.
- count – Number of bytes transferred so far by the non-blocking transaction.

FSL_FLEXIO_SPI_EDMA_DRIVER_VERSION

FlexIO SPI EDMA driver version.

typedef struct *_flexio_spi_master_edma_handle* flexio_spi_master_edma_handle_t

typedef for flexio_spi_master_edma_handle_t in advance.

typedef *flexio_spi_master_edma_handle_t* flexio_spi_slave_edma_handle_t

Slave handle is the same with master handle.

typedef void (*flexio_spi_master_edma_transfer_callback_t)(*FLEXIO_SPI_Type* *base, *flexio_spi_master_edma_handle_t* *handle, *status_t* status, void *userData)

FlexIO SPI master callback for finished transmit.

typedef void (*flexio_spi_slave_edma_transfer_callback_t)(*FLEXIO_SPI_Type* *base, *flexio_spi_slave_edma_handle_t* *handle, *status_t* status, void *userData)

FlexIO SPI slave callback for finished transmit.

struct _flexio_spi_master_edma_handle

#include <fsl_flexio_spi_edma.h> FlexIO SPI eDMA transfer handle, users should not touch the content of the handle.

Public Members

size_t transferSize

Total bytes to be transferred.

uint8_t nbytes

eDMA minor byte transfer count initially configured.

bool txInProgress
Send transfer in progress

bool rxInProgress
Receive transfer in progress

edma_handle_t *txHandle
DMA handler for SPI send

edma_handle_t *rxHandle
DMA handler for SPI receive

flexio_spi_master_edma_transfer_callback_t callback
Callback for SPI DMA transfer

void *userData
User Data for SPI DMA callback

2.23 FlexIO eDMA UART Driver

status_t FLEXIO_UART_TransferCreateHandleEDMA(*FLEXIO_UART_Type* *base, *flexio_uart_edma_handle_t* *handle, *flexio_uart_edma_transfer_callback_t* callback, void *userData, *edma_handle_t* *txEdmaHandle, *edma_handle_t* *rxEdmaHandle)

Initializes the UART handle which is used in transactional functions.

Parameters

- base – Pointer to FLEXIO_UART_Type.
- handle – Pointer to flexio_uart_edma_handle_t structure.
- callback – The callback function.
- userData – The parameter of the callback function.
- rxEdmaHandle – User requested DMA handle for RX DMA transfer.
- txEdmaHandle – User requested DMA handle for TX DMA transfer.

Return values

- kStatus_Success – Successfully create the handle.
- kStatus_OutOfRange – The FlexIO SPI eDMA type/handle table out of range.

status_t FLEXIO_UART_TransferSendEDMA(*FLEXIO_UART_Type* *base, *flexio_uart_edma_handle_t* *handle, *flexio_uart_transfer_t* *xfer)

Sends data using eDMA.

This function sends data using eDMA. This is a non-blocking function, which returns right away. When all data is sent out, the send callback function is called.

Parameters

- base – Pointer to FLEXIO_UART_Type

- handle – UART handle pointer.
- xfer – UART eDMA transfer structure, see *flexio_uart_transfer_t*.

Return values

- kStatus_Success – if succeed, others failed.
- kStatus_FLEXIO_UART_TxBusy – Previous transfer on going.

status_t FLEXIO_UART_TransferReceiveEDMA(*FLEXIO_UART_Type* *base, *flexio_uart_edma_handle_t* *handle, *flexio_uart_transfer_t* *xfer)

Receives data using eDMA.

This function receives data using eDMA. This is a non-blocking function, which returns right away. When all data is received, the receive callback function is called.

Parameters

- base – Pointer to FLEXIO_UART_Type
- handle – Pointer to flexio_uart_edma_handle_t structure
- xfer – UART eDMA transfer structure, see *flexio_uart_transfer_t*.

Return values

- kStatus_Success – if succeed, others failed.
- kStatus_UART_RxBusy – Previous transfer on going.

void FLEXIO_UART_TransferAbortSendEDMA(*FLEXIO_UART_Type* *base, *flexio_uart_edma_handle_t* *handle)

Aborts the sent data which using eDMA.

This function aborts sent data which using eDMA.

Parameters

- base – Pointer to FLEXIO_UART_Type
- handle – Pointer to flexio_uart_edma_handle_t structure

void FLEXIO_UART_TransferAbortReceiveEDMA(*FLEXIO_UART_Type* *base, *flexio_uart_edma_handle_t* *handle)

Aborts the receive data which using eDMA.

This function aborts the receive data which using eDMA.

Parameters

- base – Pointer to FLEXIO_UART_Type
- handle – Pointer to flexio_uart_edma_handle_t structure

status_t FLEXIO_UART_TransferGetSendCountEDMA(*FLEXIO_UART_Type* *base, *flexio_uart_edma_handle_t* *handle, size_t *count)

Gets the number of bytes sent out.

This function gets the number of bytes sent out.

Parameters

- base – Pointer to FLEXIO_UART_Type
- handle – Pointer to flexio_uart_edma_handle_t structure
- count – Number of bytes sent so far by the non-blocking transaction.

Return values

- kStatus_NoTransferInProgress – transfer has finished or no transfer in progress.
- kStatus_Success – Successfully return the count.

status_t FLEXIO_UART_TransferGetReceiveCountEDMA(*FLEXIO_UART_Type* *base, *flexio_uart_edma_handle_t* *handle, size_t *count)

Gets the number of bytes received.

This function gets the number of bytes received.

Parameters

- base – Pointer to FLEXIO_UART_Type
- handle – Pointer to flexio_uart_edma_handle_t structure
- count – Number of bytes received so far by the non-blocking transaction.

Return values

- kStatus_NoTransferInProgress – transfer has finished or no transfer in progress.
- kStatus_Success – Successfully return the count.

FSL_FLEXIO_UART_EDMA_DRIVER_VERSION

FlexIO UART EDMA driver version.

typedef struct *_flexio_uart_edma_handle* flexio_uart_edma_handle_t

typedef void (*flexio_uart_edma_transfer_callback_t)(*FLEXIO_UART_Type* *base, *flexio_uart_edma_handle_t* *handle, *status_t* status, void *userData)

UART transfer callback function.

struct _flexio_uart_edma_handle

#include <fsl_flexio_uart_edma.h> UART eDMA handle.

Public Members

flexio_uart_edma_transfer_callback_t callback

Callback function.

void *userData

UART callback function parameter.

size_t txDataSizeAll

Total bytes to be sent.

size_t rxDataSizeAll

Total bytes to be received.

edma_handle_t *txEdmaHandle

The eDMA TX channel used.

edma_handle_t *rxEdmaHandle
The eDMA RX channel used.

uint8_t nbytes
eDMA minor byte transfer count initially configured.

volatile uint8_t txState
TX transfer state.

volatile uint8_t rxState
RX transfer state

2.24 FlexIO I2C Master Driver

status_t FLEXIO_I2C_CheckForBusyBus(*FLEXIO_I2C_Type* *base)

Make sure the bus isn't already pulled down.

Check the FLEXIO pin status to see whether either of SDA and SCL pin is pulled down.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure..

Return values

- kStatus_Success –
- kStatus_FLEXIO_I2C_Busy –

status_t FLEXIO_I2C_MasterInit(*FLEXIO_I2C_Type* *base, *flexio_i2c_master_config_t* *masterConfig, uint32_t srcClock_Hz)

Ungates the FlexIO clock, resets the FlexIO module, and configures the FlexIO I2C hardware configuration.

Example

```
FLEXIO_I2C_Type base = {
.flexioBase = FLEXIO,
.SDAPinIndex = 0,
.SCLPinIndex = 1,
.shifterIndex = {0,1},
.timerIndex = {0,1}
};
flexio_i2c_master_config_t config = {
.enableInDoze = false,
.enableInDebug = true,
.enableFastAccess = false,
.baudRate_Bps = 100000
};
FLEXIO_I2C_MasterInit(base, &config, srcClock_Hz);
```

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- masterConfig – Pointer to flexio_i2c_master_config_t structure.
- srcClock_Hz – FlexIO source clock in Hz.

Return values

- kStatus_Success – Initialization successful
- kStatus_InvalidArgument – The source clock exceed upper range limitation

void FLEXIO_I2C_MasterDeinit(*FLEXIO_I2C_Type* *base)

De-initializes the FlexIO I2C master peripheral. Calling this API Resets the FlexIO I2C master shifer and timer config, module can't work unless the FLEXIO_I2C_MasterInit is called.

Parameters

- base – pointer to FLEXIO_I2C_Type structure.

void FLEXIO_I2C_MasterGetDefaultConfig(*flexio_i2c_master_config_t* *masterConfig)

Gets the default configuration to configure the FlexIO module. The configuration can be used directly for calling the *FLEXIO_I2C_MasterInit()*.

Example:

```
flexio_i2c_master_config_t config;
FLEXIO_I2C_MasterGetDefaultConfig(&config);
```

Parameters

- masterConfig – Pointer to flexio_i2c_master_config_t structure.

static inline void FLEXIO_I2C_MasterEnable(*FLEXIO_I2C_Type* *base, bool enable)

Enables/disables the FlexIO module operation.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- enable – Pass true to enable module, false does not have any effect.

uint32_t FLEXIO_I2C_MasterGetStatusFlags(*FLEXIO_I2C_Type* *base)

Gets the FlexIO I2C master status flags.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure

Returns

Status flag, use status flag to AND *_flexio_i2c_master_status_flags* can get the related status.

void FLEXIO_I2C_MasterClearStatusFlags(*FLEXIO_I2C_Type* *base, uint32_t mask)

Clears the FlexIO I2C master status flags.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- mask – Status flag. The parameter can be any combination of the following values:
  - kFLEXIO_I2C_RxFullFlag
  - kFLEXIO_I2C_ReceiveNakFlag

void FLEXIO_I2C_MasterEnableInterrupts(*FLEXIO_I2C_Type* *base, uint32_t mask)

Enables the FlexIO i2c master interrupt requests.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- mask – Interrupt source. Currently only one interrupt request source:
  - kFLEXIO_I2C_TransferCompleteInterruptEnable

void FLEXIO_I2C_MasterDisableInterrupts(*FLEXIO_I2C_Type* *base, uint32_t mask)

Disables the FlexIO I2C master interrupt requests.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- mask – Interrupt source.

void FLEXIO_I2C_MasterSetBaudRate(*FLEXIO_I2C_Type* *base, uint32_t baudRate_Bps, uint32_t srcClock_Hz)

Sets the FlexIO I2C master transfer baudrate.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure
- baudRate_Bps – the baud rate value in HZ
- srcClock_Hz – source clock in HZ

void FLEXIO_I2C_MasterStart(*FLEXIO_I2C_Type* *base, uint8_t address, *flexio_i2c_direction_t* direction)

Sends START + 7-bit address to the bus.

Note: This API should be called when the transfer configuration is ready to send a START signal and 7-bit address to the bus. This is a non-blocking API, which returns directly after the address is put into the data register but the address transfer is not finished on the bus. Ensure that the kFLEXIO_I2C_RxFullFlag status is asserted before calling this API.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- address – 7-bit address.
- direction – transfer direction. This parameter is one of the values in flexio_i2c_direction_t:
  - kFLEXIO_I2C_Write: Transmit
  - kFLEXIO_I2C_Read: Receive

void FLEXIO_I2C_MasterStop(*FLEXIO_I2C_Type* *base)

Sends the stop signal on the bus.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.

void FLEXIO_I2C_MasterRepeatedStart(*FLEXIO_I2C_Type* *base)

Sends the repeated start signal on the bus.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.

void FLEXIO_I2C_MasterAbortStop(*FLEXIO_I2C_Type* *base)

Sends the stop signal when transfer is still on-going.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.

void FLEXIO_I2C_MasterEnableAck(*FLEXIO_I2C_Type* *base, bool enable)

Configures the sent ACK/NAK for the following byte.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- enable – True to configure send ACK, false configure to send NAK.

status_t FLEXIO_I2C_MasterSetTransferCount(*FLEXIO_I2C_Type* *base, uint16_t count)

Sets the number of bytes to be transferred from a start signal to a stop signal.

Note: Call this API before a transfer begins because the timer generates a number of clocks according to the number of bytes that need to be transferred.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- count – Number of bytes need to be transferred from a start signal to a re-start/stop signal

Return values

- kStatus_Success – Successfully configured the count.
- kStatus_InvalidArgument – Input argument is invalid.

static inline void FLEXIO_I2C_MasterWriteByte(*FLEXIO_I2C_Type* *base, uint32_t data)

Writes one byte of data to the I2C bus.

Note: This is a non-blocking API, which returns directly after the data is put into the data register but the data transfer is not finished on the bus. Ensure that the TxEmptyFlag is asserted before calling this API.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- data – a byte of data.

static inline uint8_t FLEXIO_I2C_MasterReadByte(*FLEXIO_I2C_Type* *base)

Reads one byte of data from the I2C bus.

Note: This is a non-blocking API, which returns directly after the data is read from the data register. Ensure that the data is ready in the register.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.

Returns

data byte read.

status_t FLEXIO_I2C_MasterWriteBlocking(*FLEXIO_I2C_Type* *base, const uint8_t *txBuff, uint8_t txSize)

Sends a buffer of data in bytes.

Note: This function blocks via polling until all bytes have been sent.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- txBuff – The data bytes to send.
- txSize – The number of data bytes to send.

Return values

- kStatus_Success – Successfully write data.
- kStatus_FLEXIO_I2C_Nak – Receive NAK during writing data.
- kStatus_FLEXIO_I2C_Timeout – Timeout polling status flags.

status_t FLEXIO_I2C_MasterReadBlocking(*FLEXIO_I2C_Type* *base, uint8_t *rxBuff, uint8_t rxSize)

Receives a buffer of bytes.

Note: This function blocks via polling until all bytes have been received.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- rxBuff – The buffer to store the received bytes.
- rxSize – The number of data bytes to be received.

Return values

- kStatus_Success – Successfully read data.
- kStatus_FLEXIO_I2C_Timeout – Timeout polling status flags.

status_t FLEXIO_I2C_MasterTransferBlocking(*FLEXIO_I2C_Type* *base, *flexio_i2c_master_transfer_t* *xfer)

Performs a master polling transfer on the I2C bus.

Note: The API does not return until the transfer succeeds or fails due to receiving NAK.

Parameters

- base – pointer to FLEXIO_I2C_Type structure.
- xfer – pointer to flexio_i2c_master_transfer_t structure.

Returns

status of status_t.

status_t FLEXIO_I2C_MasterTransferCreateHandle(*FLEXIO_I2C_Type* *base, *flexio_i2c_master_handle_t* *handle, *flexio_i2c_master_transfer_callback_t* callback, void *userData)

Initializes the I2C handle which is used in transactional functions.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- handle – Pointer to flexio_i2c_master_handle_t structure to store the transfer state.
- callback – Pointer to user callback function.
- userData – User param passed to the callback function.

Return values

- kStatus_Success – Successfully create the handle.
- kStatus_OutOfRange – The FlexIO type/handle/isr table out of range.

status_t FLEXIO_I2C_MasterTransferNonBlocking(*FLEXIO_I2C_Type* *base, *flexio_i2c_master_handle_t* *handle, *flexio_i2c_master_transfer_t* *xfer)

Performs a master interrupt non-blocking transfer on the I2C bus.

Note: The API returns immediately after the transfer initiates. Call FLEXIO_I2C_MasterTransferGetCount to poll the transfer status to check whether the transfer is finished. If the return status is not kStatus_FLEXIO_I2C_Busy, the transfer is finished.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure
- handle – Pointer to flexio_i2c_master_handle_t structure which stores the transfer state
- xfer – pointer to flexio_i2c_master_transfer_t structure

Return values

- kStatus_Success – Successfully start a transfer.
- kStatus_FLEXIO_I2C_Busy – FlexIO I2C is not idle, is running another transfer.

status_t FLEXIO_I2C_MasterTransferGetCount(*FLEXIO_I2C_Type* *base, *flexio_i2c_master_handle_t* *handle, size_t *count)

Gets the master transfer status during a interrupt non-blocking transfer.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- handle – Pointer to flexio_i2c_master_handle_t structure which stores the transfer state.
- count – Number of bytes transferred so far by the non-blocking transaction.

Return values

- kStatus_InvalidArgument – count is Invalid.
- kStatus_NoTransferInProgress – There is not a non-blocking transaction currently in progress.
- kStatus_Success – Successfully return the count.

void FLEXIO_I2C_MasterTransferAbort(*FLEXIO_I2C_Type* *base, *flexio_i2c_master_handle_t* *handle)

Aborts an interrupt non-blocking transfer early.

Note: This API can be called at any time when an interrupt non-blocking transfer initiates to abort the transfer early.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure
- handle – Pointer to flexio_i2c_master_handle_t structure which stores the transfer state

void FLEXIO_I2C_MasterTransferHandleIRQ(void *i2cType, void *i2cHandle)

Master interrupt handler.

Parameters

- i2cType – Pointer to FLEXIO_I2C_Type structure
- i2cHandle – Pointer to flexio_i2c_master_transfer_t structure

FSL_FLEXIO_I2C_MASTER_DRIVER_VERSION

FlexIO I2C transfer status.

Values:

enumerator kStatus_FLEXIO_I2C_Busy

I2C is busy doing transfer.

enumerator kStatus_FLEXIO_I2C_Idle

I2C is busy doing transfer.

enumerator kStatus_FLEXIO_I2C_Nak

NAK received during transfer.

enumerator kStatus_FLEXIO_I2C_Timeout

Timeout polling status flags.

enum _flexio_i2c_master_interrupt

Define FlexIO I2C master interrupt mask.

Values:

enumerator kFLEXIO_I2C_TxEmptyInterruptEnable

Tx buffer empty interrupt enable.

enumerator kFLEXIO_I2C_RxFullInterruptEnable

Rx buffer full interrupt enable.

enum _flexio_i2c_master_status_flags

Define FlexIO I2C master status mask.

Values:

enumerator kFLEXIO_I2C_TxEmptyFlag

Tx shifter empty flag.

enumerator kFLEXIO_I2C_RxFullFlag

Rx shifter full/Transfer complete flag.

enumerator kFLEXIO_I2C_ReceiveNakFlag

Receive NAK flag.

enum _flexio_i2c_direction

Direction of master transfer.

Values:

enumerator kFLEXIO_I2C_Write

Master send to slave.

enumerator kFLEXIO_I2C_Read

Master receive from slave.

typedef enum *_flexio_i2c_direction* flexio_i2c_direction_t

Direction of master transfer.

typedef struct *_flexio_i2c_type* FLEXIO_I2C_Type

Define FlexIO I2C master access structure typedef.

typedef struct *_flexio_i2c_master_config* flexio_i2c_master_config_t

Define FlexIO I2C master user configuration structure.

typedef struct *_flexio_i2c_master_transfer* flexio_i2c_master_transfer_t

Define FlexIO I2C master transfer structure.

typedef struct *_flexio_i2c_master_handle* flexio_i2c_master_handle_t

FlexIO I2C master handle typedef.

typedef void (*flexio_i2c_master_transfer_callback_t)(*FLEXIO_I2C_Type* *base, *flexio_i2c_master_handle_t* *handle, *status_t* status, void *userData)

FlexIO I2C master transfer callback typedef.

I2C_RETRY_TIMES

Retry times for waiting flag.

struct _flexio_i2c_type

#include <fsl_flexio_i2c_master.h> Define FlexIO I2C master access structure typedef.

Public Members

FLEXIO_Type *flexioBase
FlexIO base pointer.

uint8_t SDAPinIndex
Pin select for I2C SDA.

uint8_t SCLPinIndex
Pin select for I2C SCL.

uint8_t shifterIndex[2]
Shifter index used in FlexIO I2C.

uint8_t timerIndex[3]
Timer index used in FlexIO I2C.

uint32_t baudrate
Master transfer baudrate, used to calculate delay time.

struct __flexio_i2c_master_config
#include <fsl_flexio_i2c_master.h> Define FlexIO I2C master user configuration structure.

Public Members

bool enableMaster
Enables the FlexIO I2C peripheral at initialization time.

bool enableInDoze
Enable/disable FlexIO operation in doze mode.

bool enableInDebug
Enable/disable FlexIO operation in debug mode.

bool enableFastAccess
Enable/disable fast access to FlexIO registers, fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.

uint32_t baudRate_Bps
Baud rate in Bps.

struct __flexio_i2c_master_transfer
#include <fsl_flexio_i2c_master.h> Define FlexIO I2C master transfer structure.

Public Members

uint32_t flags
Transfer flag which controls the transfer, reserved for FlexIO I2C.

uint8_t slaveAddress
7-bit slave address.

flexio_i2c_direction_t direction
Transfer direction, read or write.

uint32_t subaddress
Sub address. Transferred MSB first.

uint8_t subaddressSize
Size of sub address.

uint8_t volatile *data
Transfer buffer.

volatile size_t dataSize
Transfer size.

struct __flexio__i2c__master__handle
#include <fsl_flexio_i2c_master.h> Define FlexIO I2C master handle structure.

Public Members

flexio_i2c_master_transfer_t transfer
FlexIO I2C master transfer copy.

size_t transferSize
Total bytes to be transferred.

uint8_t state
Transfer state maintained during transfer.

flexio_i2c_master_transfer_callback_t completionCallback
Callback function called at transfer event. Callback function called at transfer event.

void *userData
Callback parameter passed to callback function.

bool needRestart
Whether master needs to send re-start signal.

2.25 FlexIO I2S Driver

void FLEXIO_I2S_Init(*FLEXIO_I2S_Type* *base, const *flexio_i2s_config_t* *config)

Initializes the FlexIO I2S.

This API configures FlexIO pins and shifter to I2S and configures the FlexIO I2S with a configuration structure. The configuration structure can be filled by the user, or be set with default values by *FLEXIO_I2S_GetDefaultConfig()*.

Note: This API should be called at the beginning of the application to use the FlexIO I2S driver. Otherwise, any access to the FlexIO I2S module can cause hard fault because the clock is not enabled.

Parameters

- base – FlexIO I2S base pointer
- config – FlexIO I2S configure structure.

void FLEXIO_I2S_GetDefaultConfig(*flexio_i2s_config_t* *config)

Sets the FlexIO I2S configuration structure to default values.

The purpose of this API is to get the configuration structure initialized for use in *FLEXIO_I2S_Init()*. Users may use the initialized structure unchanged in *FLEXIO_I2S_Init()* or modify some fields of the structure before calling *FLEXIO_I2S_Init()*.

Parameters

- config – pointer to master configuration structure

void FLEXIO_I2S_Deinit(*FLEXIO_I2S_Type* *base)

De-initializes the FlexIO I2S.

Calling this API resets the FlexIO I2S shifter and timer config. After calling this API, call the FLEXO_I2S_Init to use the FlexIO I2S module.

Parameters

- base – FlexIO I2S base pointer

static inline void FLEXIO_I2S_Enable(*FLEXIO_I2S_Type* *base, bool enable)

Enables/disables the FlexIO I2S module operation.

Parameters

- base – Pointer to FLEXIO_I2S_Type
- enable – True to enable, false dose not have any effect.

uint32_t FLEXIO_I2S_GetStatusFlags(*FLEXIO_I2S_Type* *base)

Gets the FlexIO I2S status flags.

Parameters

- base – Pointer to FLEXIO_I2S_Type structure

Returns

Status flag, which are ORed by the enumerators in the _flexio_i2s_status_flags.

void FLEXIO_I2S_EnableInterrupts(*FLEXIO_I2S_Type* *base, uint32_t mask)

Enables the FlexIO I2S interrupt.

This function enables the FlexIO UART interrupt.

Parameters

- base – Pointer to FLEXIO_I2S_Type structure
- mask – interrupt source

void FLEXIO_I2S_DisableInterrupts(*FLEXIO_I2S_Type* *base, uint32_t mask)

Disables the FlexIO I2S interrupt.

This function enables the FlexIO UART interrupt.

Parameters

- base – pointer to FLEXIO_I2S_Type structure
- mask – interrupt source

static inline void FLEXIO_I2S_TxEnableDMA(*FLEXIO_I2S_Type* *base, bool enable)

Enables/disables the FlexIO I2S Tx DMA requests.

Parameters

- base – FlexIO I2S base pointer
- enable – True means enable DMA, false means disable DMA.

static inline void FLEXIO_I2S_RxEnableDMA(*FLEXIO_I2S_Type* *base, bool enable)

Enables/disables the FlexIO I2S Rx DMA requests.

Parameters

- base – FlexIO I2S base pointer
- enable – True means enable DMA, false means disable DMA.

static inline uint32_t FLEXIO_I2S_TxGetDataRegisterAddress(*FLEXIO_I2S_Type* *base)

Gets the FlexIO I2S send data register address.

This function returns the I2S data register address, mainly used by DMA/eDMA.

Parameters

- base – Pointer to FLEXIO_I2S_Type structure

Returns

FlexIO i2s send data register address.

static inline uint32_t FLEXIO_I2S_RxGetDataRegisterAddress(*FLEXIO_I2S_Type* *base)

Gets the FlexIO I2S receive data register address.

This function returns the I2S data register address, mainly used by DMA/eDMA.

Parameters

- base – Pointer to FLEXIO_I2S_Type structure

Returns

FlexIO i2s receive data register address.

void FLEXIO_I2S_MasterSetFormat(*FLEXIO_I2S_Type* *base, *flexio_i2s_format_t* *format, uint32_t srcClock_Hz)

Configures the FlexIO I2S audio format in master mode.

Audio format can be changed in run-time of FlexIO I2S. This function configures the sample rate and audio data format to be transferred.

Parameters

- base – Pointer to FLEXIO_I2S_Type structure
- format – Pointer to FlexIO I2S audio data format structure.
- srcClock_Hz – I2S master clock source frequency in Hz.

void FLEXIO_I2S_SlaveSetFormat(*FLEXIO_I2S_Type* *base, *flexio_i2s_format_t* *format)

Configures the FlexIO I2S audio format in slave mode.

Audio format can be changed in run-time of FlexIO I2S. This function configures the sample rate and audio data format to be transferred.

Parameters

- base – Pointer to FLEXIO_I2S_Type structure
- format – Pointer to FlexIO I2S audio data format structure.

status_t FLEXIO_I2S_WriteBlocking(*FLEXIO_I2S_Type* *base, uint8_t bitWidth, uint8_t *txData, size_t size)

Sends data using a blocking method.

Note: This function blocks via polling until data is ready to be sent.

Parameters

- base – FlexIO I2S base pointer.
- bitWidth – How many bits in a audio word, usually 8/16/24/32 bits.
- txData – Pointer to the data to be written.
- size – Bytes to be written.

Return values

- kStatus_Success – Successfully write data.
- kStatus_FLEXIO_I2C_Timeout – Timeout polling status flags.

static inline void FLEXIO_I2S_WriteData(*FLEXIO_I2S_Type* *base, uint8_t bitWidth, uint32_t data)

Writes data into a data register.

Parameters

- base – FlexIO I2S base pointer.
- bitWidth – How many bits in a audio word, usually 8/16/24/32 bits.
- data – Data to be written.

status_t FLEXIO_I2S_ReadBlocking(*FLEXIO_I2S_Type* *base, uint8_t bitWidth, uint8_t *rxData, size_t size)

Receives a piece of data using a blocking method.

Note: This function blocks via polling until data is ready to be sent.

Parameters

- base – FlexIO I2S base pointer
- bitWidth – How many bits in a audio word, usually 8/16/24/32 bits.
- rxData – Pointer to the data to be read.
- size – Bytes to be read.

Return values

- kStatus_Success – Successfully read data.

- kStatus_FLEXIO_I2C_Timeout – Timeout polling status flags.

static inline uint32_t FLEXIO_I2S_ReadData(*FLEXIO_I2S_Type* *base)

Reads a data from the data register.

Parameters

- base – FlexIO I2S base pointer

Returns

Data read from data register.

void FLEXIO_I2S_TransferTxCreateHandle(*FLEXIO_I2S_Type* *base, *flexio_i2s_handle_t* *handle, *flexio_i2s_callback_t* callback, void *userData)

Initializes the FlexIO I2S handle.

This function initializes the FlexIO I2S handle which can be used for other FlexIO I2S transactional APIs. Call this API once to get the initialized handle.

Parameters

- base – Pointer to FLEXIO_I2S_Type structure
- handle – Pointer to flexio_i2s_handle_t structure to store the transfer state.
- callback – FlexIO I2S callback function, which is called while finished a block.
- userData – User parameter for the FlexIO I2S callback.

void FLEXIO_I2S_TransferSetFormat(*FLEXIO_I2S_Type* *base, *flexio_i2s_handle_t* *handle, *flexio_i2s_format_t* *format, uint32_t srcClock_Hz)

Configures the FlexIO I2S audio format.

Audio format can be changed at run-time of FlexIO I2S. This function configures the sample rate and audio data format to be transferred.

Parameters

- base – Pointer to FLEXIO_I2S_Type structure.
- handle – FlexIO I2S handle pointer.
- format – Pointer to audio data format structure.
- srcClock_Hz – FlexIO I2S bit clock source frequency in Hz. This parameter should be 0 while in slave mode.

void FLEXIO_I2S_TransferRxCreateHandle(*FLEXIO_I2S_Type* *base, *flexio_i2s_handle_t* *handle, *flexio_i2s_callback_t* callback, void *userData)

Initializes the FlexIO I2S receive handle.

This function initializes the FlexIO I2S handle which can be used for other FlexIO I2S transactional APIs. Call this API once to get the initialized handle.

Parameters

- base – Pointer to FLEXIO_I2S_Type structure.
- handle – Pointer to flexio_i2s_handle_t structure to store the transfer state.
- callback – FlexIO I2S callback function, which is called while finished a block.
- userData – User parameter for the FlexIO I2S callback.

status_t FLEXIO_I2S_TransferSendNonBlocking(*FLEXIO_I2S_Type* *base, *flexio_i2s_handle_t* *handle, *flexio_i2s_transfer_t* *xfer)

Performs an interrupt non-blocking send transfer on FlexIO I2S.

Note: The API returns immediately after transfer initiates. Call FLEXIO_I2S_GetRemainingBytes to poll the transfer status and check whether the transfer is finished. If the return status is 0, the transfer is finished.

Parameters

- base – Pointer to FLEXIO_I2S_Type structure.
- handle – Pointer to flexio_i2s_handle_t structure which stores the transfer state
- xfer – Pointer to flexio_i2s_transfer_t structure

Return values

- kStatus_Success – Successfully start the data transmission.
- kStatus_FLEXIO_I2S_TxBusy – Previous transmission still not finished, data not all written to TX register yet.
- kStatus_InvalidArgument – The input parameter is invalid.

status_t FLEXIO_I2S_TransferReceiveNonBlocking(*FLEXIO_I2S_Type* *base, *flexio_i2s_handle_t* *handle, *flexio_i2s_transfer_t* *xfer)

Performs an interrupt non-blocking receive transfer on FlexIO I2S.

Note: The API returns immediately after transfer initiates. Call FLEXIO_I2S_GetRemainingBytes to poll the transfer status to check whether the transfer is finished. If the return status is 0, the transfer is finished.

Parameters

- base – Pointer to FLEXIO_I2S_Type structure.
- handle – Pointer to flexio_i2s_handle_t structure which stores the transfer state
- xfer – Pointer to flexio_i2s_transfer_t structure

Return values

- kStatus_Success – Successfully start the data receive.
- kStatus_FLEXIO_I2S_RxBusy – Previous receive still not finished.
- kStatus_InvalidArgument – The input parameter is invalid.

void FLEXIO_I2S_TransferAbortSend(*FLEXIO_I2S_Type* *base, *flexio_i2s_handle_t* *handle)

Aborts the current send.

Note: This API can be called at any time when interrupt non-blocking transfer initiates to abort the transfer in a early time.

Parameters

- base – Pointer to FLEXIO_I2S_Type structure.

- handle – Pointer to flexio_i2s_handle_t structure which stores the transfer state

void FLEXIO_I2S_TransferAbortReceive(*FLEXIO_I2S_Type* *base, *flexio_i2s_handle_t* *handle)

Aborts the current receive.

Note: This API can be called at any time when interrupt non-blocking transfer initiates to abort the transfer in a early time.

Parameters

- base – Pointer to FLEXIO_I2S_Type structure.
- handle – Pointer to flexio_i2s_handle_t structure which stores the transfer state

status_t FLEXIO_I2S_TransferGetSendCount(*FLEXIO_I2S_Type* *base, *flexio_i2s_handle_t* *handle, size_t *count)

Gets the remaining bytes to be sent.

Parameters

- base – Pointer to FLEXIO_I2S_Type structure.
- handle – Pointer to flexio_i2s_handle_t structure which stores the transfer state
- count – Bytes sent.

Return values

- kStatus_Success – Succeed get the transfer count.
- kStatus_NoTransferInProgress – There is not a non-blocking transaction currently in progress.

status_t FLEXIO_I2S_TransferGetReceiveCount(*FLEXIO_I2S_Type* *base, *flexio_i2s_handle_t* *handle, size_t *count)

Gets the remaining bytes to be received.

Parameters

- base – Pointer to FLEXIO_I2S_Type structure.
- handle – Pointer to flexio_i2s_handle_t structure which stores the transfer state
- count – Bytes recieved.

Return values

- kStatus_Success – Succeed get the transfer count.
- kStatus_NoTransferInProgress – There is not a non-blocking transaction currently in progress.

Returns

count Bytes received.

void FLEXIO_I2S_TransferTxHandleIRQ(void *i2sBase, void *i2sHandle)

Tx interrupt handler.

Parameters

- i2sBase – Pointer to FLEXIO_I2S_Type structure.

- i2sHandle – Pointer to flexio_i2s_handle_t structure

void FLEXIO_I2S_TransferRxHandleIRQ(void *i2sBase, void *i2sHandle)

Rx interrupt handler.

Parameters

- i2sBase – Pointer to FLEXIO_I2S_Type structure.
- i2sHandle – Pointer to flexio_i2s_handle_t structure.

FSL_FLEXIO_I2S_DRIVER_VERSION

FlexIO I2S driver version 2.2.2.

FlexIO I2S transfer status.

Values:

enumerator kStatus_FLEXIO_I2S_Idle

FlexIO I2S is in idle state

enumerator kStatus_FLEXIO_I2S_TxBusy

FlexIO I2S Tx is busy

enumerator kStatus_FLEXIO_I2S_RxBusy

FlexIO I2S Tx is busy

enumerator kStatus_FLEXIO_I2S_Error

FlexIO I2S error occurred

enumerator kStatus_FLEXIO_I2S_QueueFull

FlexIO I2S transfer queue is full.

enumerator kStatus_FLEXIO_I2S_Timeout

FlexIO I2S timeout polling status flags.

enum _flexio_i2s_master_slave

Master or slave mode.

Values:

enumerator kFLEXIO_I2S_Master

Master mode

enumerator kFLEXIO_I2S_Slave

Slave mode

_flexio_i2s_interrupt_enable Define FlexIO FlexIO I2S interrupt mask.

Values:

enumerator kFLEXIO_I2S_TxDataRegEmptyInterruptEnable
Transmit buffer empty interrupt enable.

enumerator kFLEXIO_I2S_RxDataRegFullInterruptEnable
Receive buffer full interrupt enable.

_flexio_i2s_status_flags Define FlexIO FlexIO I2S status mask.

Values:

enumerator kFLEXIO_I2S_TxDataRegEmptyFlag
Transmit buffer empty flag.

enumerator kFLEXIO_I2S_RxDataRegFullFlag
Receive buffer full flag.

enum __flexio_i2s_sample_rate
Audio sample rate.

Values:

enumerator kFLEXIO_I2S_SampleRate8KHz
Sample rate 8000Hz

enumerator kFLEXIO_I2S_SampleRate11025Hz
Sample rate 11025Hz

enumerator kFLEXIO_I2S_SampleRate12KHz
Sample rate 12000Hz

enumerator kFLEXIO_I2S_SampleRate16KHz
Sample rate 16000Hz

enumerator kFLEXIO_I2S_SampleRate22050Hz
Sample rate 22050Hz

enumerator kFLEXIO_I2S_SampleRate24KHz
Sample rate 24000Hz

enumerator kFLEXIO_I2S_SampleRate32KHz
Sample rate 32000Hz

enumerator kFLEXIO_I2S_SampleRate44100Hz
Sample rate 44100Hz

enumerator kFLEXIO_I2S_SampleRate48KHz
Sample rate 48000Hz

enumerator kFLEXIO_I2S_SampleRate96KHz
Sample rate 96000Hz

enum _flexio_i2s_word_width
Audio word width.

Values:

enumerator kFLEXIO_I2S_WordWidth8bits
Audio data width 8 bits

enumerator kFLEXIO_I2S_WordWidth16bits
Audio data width 16 bits

enumerator kFLEXIO_I2S_WordWidth24bits
Audio data width 24 bits

enumerator kFLEXIO_I2S_WordWidth32bits
Audio data width 32 bits

typedef struct *_flexio_i2s_type* FLEXIO_I2S_Type
Define FlexIO I2S access structure typedef.

typedef enum *_flexio_i2s_master_slave* flexio_i2s_master_slave_t
Master or slave mode.

typedef struct *_flexio_i2s_config* flexio_i2s_config_t
FlexIO I2S configure structure.

typedef struct *_flexio_i2s_format* flexio_i2s_format_t
FlexIO I2S audio format, FlexIO I2S only support the same format in Tx and Rx.

typedef enum *_flexio_i2s_sample_rate* flexio_i2s_sample_rate_t
Audio sample rate.

typedef enum *_flexio_i2s_word_width* flexio_i2s_word_width_t
Audio word width.

typedef struct *_flexio_i2s_transfer* flexio_i2s_transfer_t
Define FlexIO I2S transfer structure.

typedef struct *_flexio_i2s_handle* flexio_i2s_handle_t

typedef void (*flexio_i2s_callback_t)(*FLEXIO_I2S_Type* *base, *flexio_i2s_handle_t* *handle, *status_t* status, void *userData)
FlexIO I2S xfer callback prototype.

I2S_RETRY_TIMES
Retry times for waiting flag.

FLEXIO_I2S_XFER_QUEUE_SIZE

FlexIO I2S transfer queue size, user can refine it according to use case.

struct _flexio_i2s_type

#include <fsl_flexio_i2s.h> Define FlexIO I2S access structure typedef.

Public Members

FLEXIO_Type *flexioBase

FlexIO base pointer

uint8_t txPinIndex

Tx data pin index in FlexIO pins

uint8_t rxPinIndex

Rx data pin index

uint8_t bclkPinIndex

Bit clock pin index

uint8_t fsPinIndex

Frame sync pin index

uint8_t txShifterIndex

Tx data shifter index

uint8_t rxShifterIndex

Rx data shifter index

uint8_t bclkTimerIndex

Bit clock timer index

uint8_t fsTimerIndex

Frame sync timer index

struct _flexio_i2s_config

#include <fsl_flexio_i2s.h> FlexIO I2S configure structure.

Public Members

bool enableI2S

Enable FlexIO I2S

flexio_i2s_master_slave_t masterSlave

Master or slave

flexio_pin_polarity_t txPinPolarity
Tx data pin polarity, active high or low

flexio_pin_polarity_t rxPinPolarity
Rx data pin polarity

flexio_pin_polarity_t bclkPinPolarity
Bit clock pin polarity

flexio_pin_polarity_t fsPinPolarity
Frame sync pin polarity

flexio_shifter_timer_polarity_t txTimerPolarity
Tx data valid on bclk rising or falling edge

flexio_shifter_timer_polarity_t rxTimerPolarity
Rx data valid on bclk rising or falling edge

struct __flexio_i2s_format
#include <fsl_flexio_i2s.h> FlexIO I2S audio format, FlexIO I2S only support the same format in Tx and Rx.

Public Members

uint8_t bitWidth
Bit width of audio data, always 8/16/24/32 bits

uint32_t sampleRate_Hz
Sample rate of the audio data

struct __flexio_i2s_transfer
#include <fsl_flexio_i2s.h> Define FlexIO I2S transfer structure.

Public Members

uint8_t *data
Data buffer start pointer

size_t dataSize
Bytes to be transferred.

struct __flexio_i2s_handle
#include <fsl_flexio_i2s.h> Define FlexIO I2S handle structure.

Public Members

uint32_t state
Internal state

flexio_i2s_callback_t callback
Callback function called at transfer event

void *userData
Callback parameter passed to callback function

uint8_t bitWidth
Bit width for transfer, 8/16/24/32bits

flexio_i2s_transfer_t queue[(4U)]
Transfer queue storing queued transfer

size_t transferSize[(4U)]
Data bytes need to transfer

volatile uint8_t queueUser
Index for user to queue transfer

volatile uint8_t queueDriver
Index for driver to get the transfer data and size

2.26 FlexIO SPI Driver

void FLEXIO_SPI_MasterInit(*FLEXIO_SPI_Type* *base, *flexio_spi_master_config_t* *masterConfig, uint32_t srcClock_Hz)

Ungates the FlexIO clock, resets the FlexIO module, configures the FlexIO SPI master hardware, and configures the FlexIO SPI with FlexIO SPI master configuration. The configuration structure can be filled by the user, or be set with default values by the *FLEXIO_SPI_MasterGetDefaultConfig()*.

Example

```
FLEXIO_SPI_Type spiDev = {
.flexioBase = FLEXIO,
.SDOPinIndex = 0,
.SDIPinIndex = 1,
.SCKPinIndex = 2,
.CSnPinIndex = 3,
.shifterIndex = {0,1},
.timerIndex = {0,1}
};
flexio_spi_master_config_t config = {
.enableMaster = true,
.enableInDoze = false,
.enableInDebug = true,
```

(continues on next page)

(continued from previous page)

```
.enableFastAccess = false,
.baudRate_Bps = 500000,
.phase = kFLEXIO_SPI_ClockPhaseFirstEdge,
.direction = kFLEXIO_SPI_MsbFirst,
.dataMode = kFLEXIO_SPI_8BitMode
};
FLEXIO_SPI_MasterInit(&spiDev, &config, srcClock_Hz);
```

Note: 1.FlexIO SPI master only support CPOL = 0, which means clock inactive low. 2.For FlexIO SPI master, the input valid time is 1.5 clock cycles, for slave the output valid time is 2.5 clock cycles. So if FlexIO SPI master communicates with other spi IPs, the maximum baud rate is FlexIO clock frequency divided by 2*2=4. If FlexIO SPI master communicates with FlexIO SPI slave, the maximum baud rate is FlexIO clock frequency divided by (1.5+2.5)*2=8.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- masterConfig – Pointer to the flexio_spi_master_config_t structure.
- srcClock_Hz – FlexIO source clock in Hz.

void FLEXIO_SPI_MasterDeinit(*FLEXIO_SPI_Type* *base)

Resets the FlexIO SPI timer and shifter config.

Parameters

- base – Pointer to the FLEXIO_SPI_Type.

void FLEXIO_SPI_MasterGetDefaultConfig(*flexio_spi_master_config_t* *masterConfig)

Gets the default configuration to configure the FlexIO SPI master. The configuration can be used directly by calling the FLEXIO_SPI_MasterConfigure(). Example:

```
flexio_spi_master_config_t masterConfig;
FLEXIO_SPI_MasterGetDefaultConfig(&masterConfig);
```

Parameters

- masterConfig – Pointer to the flexio_spi_master_config_t structure.

void FLEXIO_SPI_SlaveInit(*FLEXIO_SPI_Type* *base, *flexio_spi_slave_config_t* *slaveConfig)

Ungates the FlexIO clock, resets the FlexIO module, configures the FlexIO SPI slave hardware configuration, and configures the FlexIO SPI with FlexIO SPI slave configuration. The configuration structure can be filled by the user, or be set with default values by the *FLEXIO_SPI_SlaveGetDefaultConfig()*.

Note: 1.Only one timer is needed in the FlexIO SPI slave. As a result, the second timer index is ignored. 2.FlexIO SPI slave only support CPOL = 0, which means clock inactive low. 3.For FlexIO SPI master, the input valid time is 1.5 clock cycles, for slave the output valid time is 2.5 clock cycles. So if FlexIO SPI slave communicates with other spi IPs, the maximum baud rate is FlexIO clock frequency divided by 3*2=6. If FlexIO SPI slave communicates with FlexIO SPI master, the maximum baud rate is FlexIO clock frequency divided by (1.5+2.5)*2=8. Example

```
FLEXIO_SPI_Type spiDev = {
.flexioBase = FLEXIO,
.SDOPinIndex = 0,
.SDIPinIndex = 1,
.SCKPinIndex = 2,
.CSnPinIndex = 3,
.shifterIndex = {0,1},
.timerIndex = {0}
};
flexio_spi_slave_config_t config = {
.enableSlave = true,
.enableInDoze = false,
.enableInDebug = true,
.enableFastAccess = false,
.phase = kFLEXIO_SPI_ClockPhaseFirstEdge,
.direction = kFLEXIO_SPI_MsbFirst,
.dataMode = kFLEXIO_SPI_8BitMode
};
FLEXIO_SPI_SlaveInit(&spiDev, &config);
```

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- slaveConfig – Pointer to the flexio_spi_slave_config_t structure.

void FLEXIO_SPI_SlaveDeinit(*FLEXIO_SPI_Type* *base)

Gates the FlexIO clock.

Parameters

- base – Pointer to the FLEXIO_SPI_Type.

void FLEXIO_SPI_SlaveGetDefaultConfig(*flexio_spi_slave_config_t* *slaveConfig)

Gets the default configuration to configure the FlexIO SPI slave. The configuration can be used directly for calling the FLEXIO_SPI_SlaveConfigure(). Example:

```
flexio_spi_slave_config_t slaveConfig;
FLEXIO_SPI_SlaveGetDefaultConfig(&slaveConfig);
```

Parameters

- slaveConfig – Pointer to the flexio_spi_slave_config_t structure.

uint32_t FLEXIO_SPI_GetStatusFlags(*FLEXIO_SPI_Type* *base)

Gets FlexIO SPI status flags.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.

Returns

status flag; Use the status flag to AND the following flag mask and get the status.

- kFLEXIO_SPI_TxEmptyFlag
- kFLEXIO_SPI_RxEmptyFlag

void FLEXIO_SPI_ClearStatusFlags(*FLEXIO_SPI_Type* *base, uint32_t mask)

Clears FlexIO SPI status flags.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- mask – status flag The parameter can be any combination of the following values:
  - kFLEXIO_SPI_TxEmptyFlag
  - kFLEXIO_SPI_RxEmptyFlag

void FLEXIO_SPI_EnableInterrupts(*FLEXIO_SPI_Type* *base, uint32_t mask)

Enables the FlexIO SPI interrupt.

This function enables the FlexIO SPI interrupt.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- mask – interrupt source. The parameter can be any combination of the following values:
  - kFLEXIO_SPI_RxFullInterruptEnable
  - kFLEXIO_SPI_TxEmptyInterruptEnable

void FLEXIO_SPI_DisableInterrupts(*FLEXIO_SPI_Type* *base, uint32_t mask)

Disables the FlexIO SPI interrupt.

This function disables the FlexIO SPI interrupt.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- mask – interrupt source The parameter can be any combination of the following values:
  - kFLEXIO_SPI_RxFullInterruptEnable
  - kFLEXIO_SPI_TxEmptyInterruptEnable

void FLEXIO_SPI_EnableDMA(*FLEXIO_SPI_Type* *base, uint32_t mask, bool enable)

Enables/disables the FlexIO SPI transmit DMA. This function enables/disables the FlexIO SPI Tx DMA, which means that asserting the kFLEXIO_SPI_TxEmptyFlag does/doesn't trigger the DMA request.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- mask – SPI DMA source.
- enable – True means enable DMA, false means disable DMA.

static inline uint32_t FLEXIO_SPI_GetTxDataRegisterAddress(*FLEXIO_SPI_Type* *base, *flexio_spi_shift_direction_t* direction)

Gets the FlexIO SPI transmit data register address for MSB first transfer.

This function returns the SPI data register address, which is mainly used by DMA/eDMA.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- direction – Shift direction of MSB first or LSB first.

Returns
FlexIO SPI transmit data register address.

static inline uint32_t FLEXIO_SPI_GetRxDataRegisterAddress(*FLEXIO_SPI_Type* *base, *flexio_spi_shift_direction_t* direction)

Gets the FlexIO SPI receive data register address for the MSB first transfer.

This function returns the SPI data register address, which is mainly used by DMA/eDMA.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- direction – Shift direction of MSB first or LSB first.

Returns
FlexIO SPI receive data register address.

static inline void FLEXIO_SPI_Enable(*FLEXIO_SPI_Type* *base, bool enable)

Enables/disables the FlexIO SPI module operation.

Parameters

- base – Pointer to the FLEXIO_SPI_Type.
- enable – True to enable, false does not have any effect.

void FLEXIO_SPI_MasterSetBaudRate(*FLEXIO_SPI_Type* *base, uint32_t baudRate_Bps, uint32_t srcClockHz)

Sets baud rate for the FlexIO SPI transfer, which is only used for the master.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- baudRate_Bps – Baud Rate needed in Hz.
- srcClockHz – SPI source clock frequency in Hz.

static inline void FLEXIO_SPI_WriteData(*FLEXIO_SPI_Type* *base, *flexio_spi_shift_direction_t* direction, uint32_t data)

Writes one byte of data, which is sent using the MSB method.

Note: This is a non-blocking API, which returns directly after the data is put into the data register but the data transfer is not finished on the bus. Ensure that the TxEmptyFlag is asserted before calling this API.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- direction – Shift direction of MSB first or LSB first.
- data – 8/16/32 bit data.

static inline uint32_t FLEXIO_SPI_ReadData(*FLEXIO_SPI_Type* *base, *flexio_spi_shift_direction_t* direction)

Reads 8 bit/16 bit data.

Note: This is a non-blocking API, which returns directly after the data is read from the data register. Ensure that the RxFullFlag is asserted before calling this API.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- direction – Shift direction of MSB first or LSB first.

Returns
8 bit/16 bit data received.

status_t FLEXIO_SPI_WriteBlocking(*FLEXIO_SPI_Type* *base, *flexio_spi_shift_direction_t* direction, const uint8_t *buffer, size_t size)

Sends a buffer of data bytes.

Note: This function blocks using the polling method until all bytes have been sent.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- direction – Shift direction of MSB first or LSB first.
- buffer – The data bytes to send.
- size – The number of data bytes to send.

Return values

- kStatus_Success – Successfully create the handle.
- kStatus_FLEXIO_SPI_Timeout – The transfer timed out and was aborted.

status_t FLEXIO_SPI_ReadBlocking(*FLEXIO_SPI_Type* *base, *flexio_spi_shift_direction_t* direction, uint8_t *buffer, size_t size)

Receives a buffer of bytes.

Note: This function blocks using the polling method until all bytes have been received.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- direction – Shift direction of MSB first or LSB first.
- buffer – The buffer to store the received bytes.
- size – The number of data bytes to be received.

Return values

- kStatus_Success – Successfully create the handle.
- kStatus_FLEXIO_SPI_Timeout – The transfer timed out and was aborted.

status_t FLEXIO_SPI_MasterTransferBlocking(*FLEXIO_SPI_Type* *base, *flexio_spi_transfer_t* *xfer)

Receives a buffer of bytes.

Note: This function blocks via polling until all bytes have been received.

Parameters

- base – pointer to FLEXIO_SPI_Type structure

- xfer – FlexIO SPI transfer structure, see *flexio_spi_transfer_t*.

Return values

- kStatus_Success – Successfully create the handle.
- kStatus_FLEXIO_SPI_Timeout – The transfer timed out and was aborted.

void FLEXIO_SPI_FlushShifters(*FLEXIO_SPI_Type* *base)

Flush tx/rx shifters.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.

status_t FLEXIO_SPI_MasterTransferCreateHandle(*FLEXIO_SPI_Type* *base, *flexio_spi_master_handle_t* *handle, *flexio_spi_master_transfer_callback_t* callback, void *userData)

Initializes the FlexIO SPI Master handle, which is used in transactional functions.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- handle – Pointer to the flexio_spi_master_handle_t structure to store the transfer state.
- callback – The callback function.
- userData – The parameter of the callback function.

Return values

- kStatus_Success – Successfully create the handle.
- kStatus_OutOfRange – The FlexIO type/handle/ISR table out of range.

status_t FLEXIO_SPI_MasterTransferNonBlocking(*FLEXIO_SPI_Type* *base, *flexio_spi_master_handle_t* *handle, *flexio_spi_transfer_t* *xfer)

Master transfer data using IRQ.

This function sends data using IRQ. This is a non-blocking function, which returns right away. When all data is sent out/received, the callback function is called.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- handle – Pointer to the flexio_spi_master_handle_t structure to store the transfer state.
- xfer – FlexIO SPI transfer structure. See *flexio_spi_transfer_t*.

Return values

- kStatus_Success – Successfully start a transfer.
- kStatus_InvalidArgument – Input argument is invalid.
- kStatus_FLEXIO_SPI_Busy – SPI is not idle, is running another transfer.

void FLEXIO_SPI_MasterTransferAbort(*FLEXIO_SPI_Type* *base, *flexio_spi_master_handle_t* *handle)

Aborts the master data transfer, which used IRQ.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.

- handle – Pointer to the flexio_spi_master_handle_t structure to store the transfer state.

status_t FLEXIO_SPI_MasterTransferGetCount(*FLEXIO_SPI_Type* *base, *flexio_spi_master_handle_t* *handle, size_t *count)

Gets the data transfer status which used IRQ.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- handle – Pointer to the flexio_spi_master_handle_t structure to store the transfer state.
- count – Number of bytes transferred so far by the non-blocking transaction.

Return values

- kStatus_InvalidArgument – count is Invalid.
- kStatus_Success – Successfully return the count.

void FLEXIO_SPI_MasterTransferHandleIRQ(void *spiType, void *spiHandle)

FlexIO SPI master IRQ handler function.

Parameters

- spiType – Pointer to the FLEXIO_SPI_Type structure.
- spiHandle – Pointer to the flexio_spi_master_handle_t structure to store the transfer state.

status_t FLEXIO_SPI_SlaveTransferCreateHandle(*FLEXIO_SPI_Type* *base, *flexio_spi_slave_handle_t* *handle, *flexio_spi_slave_transfer_callback_t* callback, void *userData)

Initializes the FlexIO SPI Slave handle, which is used in transactional functions.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- handle – Pointer to the flexio_spi_slave_handle_t structure to store the transfer state.
- callback – The callback function.
- userData – The parameter of the callback function.

Return values

- kStatus_Success – Successfully create the handle.
- kStatus_OutOfRange – The FlexIO type/handle/ISR table out of range.

status_t FLEXIO_SPI_SlaveTransferNonBlocking(*FLEXIO_SPI_Type* *base, *flexio_spi_slave_handle_t* *handle, *flexio_spi_transfer_t* *xfer)

Slave transfer data using IRQ.

This function sends data using IRQ. This is a non-blocking function, which returns right away. When all data is sent out/received, the callback function is called.

Parameters

- handle – Pointer to the flexio_spi_slave_handle_t structure to store the transfer state.
- base – Pointer to the FLEXIO_SPI_Type structure.

- xfer – FlexIO SPI transfer structure. See *flexio_spi_transfer_t*.

Return values

- kStatus_Success – Successfully start a transfer.
- kStatus_InvalidArgument – Input argument is invalid.
- kStatus_FLEXIO_SPI_Busy – SPI is not idle; it is running another transfer.

static inline void FLEXIO_SPI_SlaveTransferAbort(*FLEXIO_SPI_Type* *base, *flexio_spi_slave_handle_t* *handle)

Aborts the slave data transfer which used IRQ, share same API with master.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- handle – Pointer to the flexio_spi_slave_handle_t structure to store the transfer state.

static inline *status_t* FLEXIO_SPI_SlaveTransferGetCount(*FLEXIO_SPI_Type* *base, *flexio_spi_slave_handle_t* *handle, size_t *count)

Gets the data transfer status which used IRQ, share same API with master.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- handle – Pointer to the flexio_spi_slave_handle_t structure to store the transfer state.
- count – Number of bytes transferred so far by the non-blocking transaction.

Return values

- kStatus_InvalidArgument – count is Invalid.
- kStatus_Success – Successfully return the count.

void FLEXIO_SPI_SlaveTransferHandleIRQ(void *spiType, void *spiHandle)

FlexIO SPI slave IRQ handler function.

Parameters

- spiType – Pointer to the FLEXIO_SPI_Type structure.
- spiHandle – Pointer to the flexio_spi_slave_handle_t structure to store the transfer state.

FSL_FLEXIO_SPI_DRIVER_VERSION

FlexIO SPI driver version.

Error codes for the FlexIO SPI driver.

Values:

enumerator kStatus_FLEXIO_SPI_Busy

FlexIO SPI is busy.

enumerator kStatus_FLEXIO_SPI_Idle

SPI is idle

enumerator kStatus_FLEXIO_SPI_Error
FlexIO SPI error.

enumerator kStatus_FLEXIO_SPI_Timeout
FlexIO SPI timeout polling status flags.

enum _flexio_spi_clock_phase
FlexIO SPI clock phase configuration.

Values:

enumerator kFLEXIO_SPI_ClockPhaseFirstEdge
First edge on SPSCK occurs at the middle of the first cycle of a data transfer.

enumerator kFLEXIO_SPI_ClockPhaseSecondEdge
First edge on SPSCK occurs at the start of the first cycle of a data transfer.

enum _flexio_spi_shift_direction
FlexIO SPI data shifter direction options.

Values:

enumerator kFLEXIO_SPI_MsbFirst
Data transfers start with most significant bit.

enumerator kFLEXIO_SPI_LsbFirst
Data transfers start with least significant bit.

enum _flexio_spi_data_bitcount_mode
FlexIO SPI data length mode options.

Values:

enumerator kFLEXIO_SPI_8BitMode
8-bit data transmission mode.

enumerator kFLEXIO_SPI_16BitMode
16-bit data transmission mode.

enumerator kFLEXIO_SPI_32BitMode
32-bit data transmission mode.

enum _flexio_spi_interrupt_enable
Define FlexIO SPI interrupt mask.

Values:

enumerator kFLEXIO_SPI_TxEmptyInterruptEnable
Transmit buffer empty interrupt enable.

enumerator kFLEXIO_SPI_RxFullInterruptEnable
Receive buffer full interrupt enable.

enum _flexio_spi_status_flags

Define FlexIO SPI status mask.

Values:

enumerator kFLEXIO_SPI_TxBufferEmptyFlag

Transmit buffer empty flag.

enumerator kFLEXIO_SPI_RxBufferFullFlag

Receive buffer full flag.

enum _flexio_spi_dma_enable

Define FlexIO SPI DMA mask.

Values:

enumerator kFLEXIO_SPI_TxDmaEnable

Tx DMA request source

enumerator kFLEXIO_SPI_RxDmaEnable

Rx DMA request source

enumerator kFLEXIO_SPI_DmaAllEnable

All DMA request source

enum _flexio_spi_transfer_flags

Define FlexIO SPI transfer flags.

Note: Use kFLEXIO_SPI_csContinuous and one of the other flags to OR together to form the transfer flag.

Values:

enumerator kFLEXIO_SPI_8bitMsb

FlexIO SPI 8-bit MSB first

enumerator kFLEXIO_SPI_8bitLsb

FlexIO SPI 8-bit LSB first

enumerator kFLEXIO_SPI_16bitMsb

FlexIO SPI 16-bit MSB first

enumerator kFLEXIO_SPI_16bitLsb

FlexIO SPI 16-bit LSB first

enumerator kFLEXIO_SPI_32bitMsb

FlexIO SPI 32-bit MSB first

enumerator kFLEXIO_SPI_32bitLsb

FlexIO SPI 32-bit LSB first

enumerator kFLEXIO_SPI_csContinuous
Enable the CS signal continuous mode

typedef enum *_flexio_spi_clock_phase* flexio_spi_clock_phase_t
FlexIO SPI clock phase configuration.

typedef enum *_flexio_spi_shift_direction* flexio_spi_shift_direction_t
FlexIO SPI data shifter direction options.

typedef enum *_flexio_spi_data_bitcount_mode* flexio_spi_data_bitcount_mode_t
FlexIO SPI data length mode options.

typedef struct *_flexio_spi_type* FLEXIO_SPI_Type
Define FlexIO SPI access structure typedef.

typedef struct *_flexio_spi_master_config* flexio_spi_master_config_t
Define FlexIO SPI master configuration structure.

typedef struct *_flexio_spi_slave_config* flexio_spi_slave_config_t
Define FlexIO SPI slave configuration structure.

typedef struct *_flexio_spi_transfer* flexio_spi_transfer_t
Define FlexIO SPI transfer structure.

typedef struct *_flexio_spi_master_handle* flexio_spi_master_handle_t
typedef for flexio_spi_master_handle_t in advance.

typedef *flexio_spi_master_handle_t* flexio_spi_slave_handle_t
Slave handle is the same with master handle.

typedef void (*flexio_spi_master_transfer_callback_t)(*FLEXIO_SPI_Type* *base, *flexio_spi_master_handle_t* *handle, *status_t* status, void *userData)
FlexIO SPI master callback for finished transmit.

typedef void (*flexio_spi_slave_transfer_callback_t)(*FLEXIO_SPI_Type* *base, *flexio_spi_slave_handle_t* *handle, *status_t* status, void *userData)
FlexIO SPI slave callback for finished transmit.

FLEXIO_SPI_DUMMYDATA
FlexIO SPI dummy transfer data, the data is sent while txData is NULL.

SPI_RETRY_TIMES
Retry times for waiting flag.

FLEXIO_SPI_XFER_DATA_FORMAT(flag)
Get the transfer data format of width and bit order.

struct _flexio_spi_type
#include <fsl_flexio_spi.h> Define FlexIO SPI access structure typedef.

Public Members

FLEXIO_Type *flexioBase
: FlexIO base pointer.

uint8_t SDOPinIndex
: Pin select for data output. To set SDO pin in Hi-Z state, user needs to mux the pin as GPIO input and disable all pull up/down in application.

uint8_t SDIPinIndex
: Pin select for data input.

uint8_t SCKPinIndex
: Pin select for clock.

uint8_t CSnPinIndex
: Pin select for enable.

uint8_t shifterIndex[2]
: Shifter index used in FlexIO SPI.

uint8_t timerIndex[2]
: Timer index used in FlexIO SPI.

struct __flexio_spi_master_config
: *#include <fsl_flexio_spi.h>* Define FlexIO SPI master configuration structure.

Public Members

bool enableMaster
: Enable/disable FlexIO SPI master after configuration.

bool enableInDoze
: Enable/disable FlexIO operation in doze mode.

bool enableInDebug
: Enable/disable FlexIO operation in debug mode.

bool enableFastAccess
: Enable/disable fast access to FlexIO registers, fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.

uint32_t baudRate_Bps
: Baud rate in Bps.

flexio_spi_clock_phase_t phase
: Clock phase.

flexio_spi_data_bitcount_mode_t dataMode
8bit or 16bit mode.

struct __flexio__spi__slave__config
#include <fsl_flexio_spi.h> Define FlexIO SPI slave configuration structure.

Public Members

bool enableSlave
Enable/disable FlexIO SPI slave after configuration.

bool enableInDoze
Enable/disable FlexIO operation in doze mode.

bool enableInDebug
Enable/disable FlexIO operation in debug mode.

bool enableFastAccess
Enable/disable fast access to FlexIO registers, fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.

flexio_spi_clock_phase_t phase
Clock phase.

flexio_spi_data_bitcount_mode_t dataMode
8bit or 16bit mode.

struct __flexio__spi__transfer
#include <fsl_flexio_spi.h> Define FlexIO SPI transfer structure.

Public Members

const uint8_t *txData
Send buffer.

uint8_t *rxData
Receive buffer.

size_t dataSize
Transfer bytes.

uint8_t flags
FlexIO SPI control flag, MSB first or LSB first.

struct __flexio__spi__master__handle
#include <fsl_flexio_spi.h> Define FlexIO SPI handle structure.

Public Members

const uint8_t *txData
Transfer buffer.

uint8_t *rxData
Receive buffer.

size_t transferSize
Total bytes to be transferred.

volatile size_t txRemainingBytes
Send data remaining in bytes.

volatile size_t rxRemainingBytes
Receive data remaining in bytes.

volatile uint32_t state
FlexIO SPI internal state.

uint8_t bytePerFrame
SPI mode, 2bytes or 1byte in a frame

flexio_spi_shift_direction_t direction
Shift direction.

flexio_spi_master_transfer_callback_t callback
FlexIO SPI callback.

void *userData
Callback parameter.

bool isCsContinuous
Is current transfer using CS continuous mode.

uint32_t timer1Cfg
TIMER1 TIMCFG regiser value backup.

2.27 FlexIO UART Driver

status_t FLEXIO_UART_Init(*FLEXIO_UART_Type* *base, const *flexio_uart_config_t* *userConfig, uint32_t srcClock_Hz)

Ungates the FlexIO clock, resets the FlexIO module, configures FlexIO UART hardware, and configures the FlexIO UART with FlexIO UART configuration. The configuration structure can be filled by the user or be set with default values by *FLEXIO_UART_GetDefaultConfig()*.

Example

```
FLEXIO_UART_Type base = {
.flexioBase = FLEXIO,
.TxPinIndex = 0,
.RxPinIndex = 1,
.shifterIndex = {0,1},
.timerIndex = {0,1}
};
flexio_uart_config_t config = {
.enableInDoze = false,
.enableInDebug = true,
.enableFastAccess = false,
.baudRate_Bps = 115200U,
.bitCountPerChar = 8
};
FLEXIO_UART_Init(base, &config, srcClock_Hz);
```

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- userConfig – Pointer to the flexio_uart_config_t structure.
- srcClock_Hz – FlexIO source clock in Hz.

Return values

- kStatus_Success – Configuration success.
- kStatus_FLEXIO_UART_BaudrateNotSupport – Baudrate is not supported for current clock source frequency.

void FLEXIO_UART_Deinit(*FLEXIO_UART_Type* *base)

Resets the FlexIO UART shifter and timer config.

Note: After calling this API, call the FLEXO_UART_Init to use the FlexIO UART module.

Parameters

- base – Pointer to FLEXIO_UART_Type structure

void FLEXIO_UART_GetDefaultConfig(*flexio_uart_config_t* *userConfig)

Gets the default configuration to configure the FlexIO UART. The configuration can be used directly for calling the *FLEXIO_UART_Init()*. Example:

```
flexio_uart_config_t config;
FLEXIO_UART_GetDefaultConfig(&userConfig);
```

Parameters

- userConfig – Pointer to the flexio_uart_config_t structure.

uint32_t FLEXIO_UART_GetStatusFlags(*FLEXIO_UART_Type* *base)

Gets the FlexIO UART status flags.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.

Returns

FlexIO UART status flags.

void FLEXIO_UART_ClearStatusFlags(*FLEXIO_UART_Type* *base, uint32_t mask)

Gets the FlexIO UART status flags.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- mask – Status flag. The parameter can be any combination of the following values:
  - kFLEXIO_UART_TxDataRegEmptyFlag
  - kFLEXIO_UART_RxEmptyFlag
  - kFLEXIO_UART_RxOverRunFlag

void FLEXIO_UART_EnableInterrupts(*FLEXIO_UART_Type* *base, uint32_t mask)

Enables the FlexIO UART interrupt.

This function enables the FlexIO UART interrupt.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- mask – Interrupt source.

void FLEXIO_UART_DisableInterrupts(*FLEXIO_UART_Type* *base, uint32_t mask)

Disables the FlexIO UART interrupt.

This function disables the FlexIO UART interrupt.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- mask – Interrupt source.

static inline uint32_t FLEXIO_UART_GetTxDataRegisterAddress(*FLEXIO_UART_Type* *base)

Gets the FlexIO UARt transmit data register address.

This function returns the UART data register address, which is mainly used by DMA/eDMA.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.

Returns

FlexIO UART transmit data register address.

static inline uint32_t FLEXIO_UART_GetRxDataRegisterAddress(*FLEXIO_UART_Type* *base)

Gets the FlexIO UART receive data register address.

This function returns the UART data register address, which is mainly used by DMA/eDMA.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.

Returns

FlexIO UART receive data register address.

static inline void FLEXIO_UART_EnableTxDMA(*FLEXIO_UART_Type* *base, bool enable)

Enables/disables the FlexIO UART transmit DMA. This function enables/disables the FlexIO UART Tx DMA, which means asserting the kFLEXIO_UART_TxDataRegEmptyFlag does/doesn't trigger the DMA request.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- enable – True to enable, false to disable.

static inline void FLEXIO_UART_EnableRxDMA(*FLEXIO_UART_Type* *base, bool enable)

Enables/disables the FlexIO UART receive DMA. This function enables/disables the FlexIO UART Rx DMA, which means asserting kFLEXIO_UART_RxDataRegFullFlag does/doesn't trigger the DMA request.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- enable – True to enable, false to disable.

static inline void FLEXIO_UART_Enable(*FLEXIO_UART_Type* *base, bool enable)

Enables/disables the FlexIO UART module operation.

Parameters

- base – Pointer to the FLEXIO_UART_Type.
- enable – True to enable, false does not have any effect.

static inline void FLEXIO_UART_WriteByte(*FLEXIO_UART_Type* *base, const uint8_t *buffer)

Writes one byte of data.

Note: This is a non-blocking API, which returns directly after the data is put into the data register. Ensure that the TxEmptyFlag is asserted before calling this API.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- buffer – The data bytes to send.

static inline void FLEXIO_UART_ReadByte(*FLEXIO_UART_Type* *base, uint8_t *buffer)

Reads one byte of data.

Note: This is a non-blocking API, which returns directly after the data is read from the data register. Ensure that the RxFullFlag is asserted before calling this API.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- buffer – The buffer to store the received bytes.

status_t FLEXIO_UART_WriteBlocking(*FLEXIO_UART_Type* *base, const uint8_t *txData, size_t txSize)

Sends a buffer of data bytes.

Note: This function blocks using the polling method until all bytes have been sent.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- txData – The data bytes to send.
- txSize – The number of data bytes to send.

Return values

- kStatus_FLEXIO_UART_Timeout – Transmission timed out and was aborted.
- kStatus_Success – Successfully wrote all data.

status_t FLEXIO_UART_ReadBlocking(*FLEXIO_UART_Type* *base, uint8_t *rxData, size_t rxSize)

Receives a buffer of bytes.

Note: This function blocks using the polling method until all bytes have been received.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- rxData – The buffer to store the received bytes.
- rxSize – The number of data bytes to be received.

Return values

- kStatus_FLEXIO_UART_Timeout – Transmission timed out and was aborted.
- kStatus_Success – Successfully received all data.

status_t FLEXIO_UART_TransferCreateHandle(*FLEXIO_UART_Type* *base, *flexio_uart_handle_t* *handle, *flexio_uart_transfer_callback_t* callback, void *userData)

Initializes the UART handle.

This function initializes the FlexIO UART handle, which can be used for other FlexIO UART transactional APIs. Call this API once to get the initialized handle.

The UART driver supports the "background" receiving, which means that users can set up a RX ring buffer optionally. Data received is stored into the ring buffer even when the user doesn't call the *FLEXIO_UART_TransferReceiveNonBlocking()* API. If there is already data received in the ring buffer, users can get the received data from the ring buffer directly. The ring buffer is disabled if passing NULL as ringBuffer.

Parameters

- base – to FLEXIO_UART_Type structure.
- handle – Pointer to the flexio_uart_handle_t structure to store the transfer state.
- callback – The callback function.
- userData – The parameter of the callback function.

Return values

- kStatus_Success – Successfully create the handle.
- kStatus_OutOfRange – The FlexIO type/handle/ISR table out of range.

void FLEXIO_UART_TransferStartRingBuffer(*FLEXIO_UART_Type* *base, *flexio_uart_handle_t* *handle, uint8_t *ringBuffer, size_t ringBufferSize)

Sets up the RX ring buffer.

This function sets up the RX ring buffer to a specific UART handle.

When the RX ring buffer is used, data received is stored into the ring buffer even when the user doesn't call the UART_ReceiveNonBlocking() API. If there is already data received in the ring buffer, users can get the received data from the ring buffer directly.

Note: When using the RX ring buffer, one byte is reserved for internal use. In other words, if ringBufferSize is 32, only 31 bytes are used for saving data.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- handle – Pointer to the flexio_uart_handle_t structure to store the transfer state.
- ringBuffer – Start address of ring buffer for background receiving. Pass NULL to disable the ring buffer.
- ringBufferSize – Size of the ring buffer.

void FLEXIO_UART_TransferStopRingBuffer(*FLEXIO_UART_Type* *base, *flexio_uart_handle_t* *handle)

Aborts the background transfer and uninstalls the ring buffer.

This function aborts the background transfer and uninstalls the ring buffer.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- handle – Pointer to the flexio_uart_handle_t structure to store the transfer state.

status_t FLEXIO_UART_TransferSendNonBlocking(*FLEXIO_UART_Type* *base, *flexio_uart_handle_t* *handle, *flexio_uart_transfer_t* *xfer)

Transmits a buffer of data using the interrupt method.

This function sends data using an interrupt method. This is a non-blocking function, which returns directly without waiting for all data to be written to the TX register. When all data is written to the TX register in ISR, the FlexIO UART driver calls the callback function and passes the *kStatus_FLEXIO_UART_TxIdle* as status parameter.

Note: The kStatus_FLEXIO_UART_TxIdle is passed to the upper layer when all data is written to the TX register. However, it does not ensure that all data is sent out.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- handle – Pointer to the flexio_uart_handle_t structure to store the transfer state.
- xfer – FlexIO UART transfer structure. See *flexio_uart_transfer_t*.

Return values

- kStatus_Success – Successfully starts the data transmission.
- kStatus_UART_TxBusy – Previous transmission still not finished, data not written to the TX register.

void FLEXIO_UART_TransferAbortSend(*FLEXIO_UART_Type* *base, *flexio_uart_handle_t* *handle)

Aborts the interrupt-driven data transmit.

This function aborts the interrupt-driven data sending. Get the remainBytes to find out how many bytes are still not sent out.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- handle – Pointer to the flexio_uart_handle_t structure to store the transfer state.

status_t FLEXIO_UART_TransferGetSendCount(*FLEXIO_UART_Type* *base, *flexio_uart_handle_t* *handle, size_t *count)

Gets the number of bytes sent.

This function gets the number of bytes sent driven by interrupt.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- handle – Pointer to the flexio_uart_handle_t structure to store the transfer state.
- count – Number of bytes sent so far by the non-blocking transaction.

Return values

- kStatus_NoTransferInProgress – transfer has finished or no transfer in progress.
- kStatus_Success – Successfully return the count.

status_t FLEXIO_UART_TransferReceiveNonBlocking(*FLEXIO_UART_Type* *base, *flexio_uart_handle_t* *handle, *flexio_uart_transfer_t* *xfer, size_t *receivedBytes)

Receives a buffer of data using the interrupt method.

This function receives data using the interrupt method. This is a non-blocking function, which returns without waiting for all data to be received. If the RX ring buffer is used and not empty, the data in ring buffer is copied and the parameter receivedBytes shows how many bytes are copied from the ring buffer. After copying, if the data in ring buffer is not enough to read, the receive request is saved by the UART driver. When new data arrives, the receive request is serviced first. When all data is received, the UART driver notifies the upper layer through a callback function and passes the status parameter kStatus_UART_RxIdle. For example, if the upper layer needs 10 bytes but there are only 5 bytes in the ring buffer, the 5 bytes are copied to xfer->data. This function returns with the parameter receivedBytes set to 5. For the last 5 bytes, newly arrived data is saved from the xfer->data[5]. When 5 bytes are received, the UART driver notifies upper layer. If the RX ring buffer is not enabled, this function enables the RX and RX interrupt to receive data to xfer->data. When all data is received, the upper layer is notified.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- handle – Pointer to the flexio_uart_handle_t structure to store the transfer state.
- xfer – UART transfer structure. See *flexio_uart_transfer_t*.
- receivedBytes – Bytes received from the ring buffer directly.

Return values

- kStatus_Success – Successfully queue the transfer into the transmit queue.
- kStatus_FLEXIO_UART_RxBusy – Previous receive request is not finished.

void FLEXIO_UART_TransferAbortReceive(*FLEXIO_UART_Type* *base, *flexio_uart_handle_t* *handle)

Aborts the receive data which was using IRQ.

This function aborts the receive data which was using IRQ.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- handle – Pointer to the flexio_uart_handle_t structure to store the transfer state.

status_t FLEXIO_UART_TransferGetReceiveCount(*FLEXIO_UART_Type* *base, *flexio_uart_handle_t* *handle, size_t *count)

Gets the number of bytes received.

This function gets the number of bytes received driven by interrupt.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- handle – Pointer to the flexio_uart_handle_t structure to store the transfer state.
- count – Number of bytes received so far by the non-blocking transaction.

Return values

- kStatus_NoTransferInProgress – transfer has finished or no transfer in progress.
- kStatus_Success – Successfully return the count.

void FLEXIO_UART_TransferHandleIRQ(void *uartType, void *uartHandle)

FlexIO UART IRQ handler function.

This function processes the FlexIO UART transmit and receives the IRQ request.

Parameters

- uartType – Pointer to the FLEXIO_UART_Type structure.
- uartHandle – Pointer to the flexio_uart_handle_t structure to store the transfer state.

void FLEXIO_UART_FlushShifters(*FLEXIO_UART_Type* *base)

Flush tx/rx shifters.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.

FSL_FLEXIO_UART_DRIVER_VERSION

FlexIO UART driver version.

Error codes for the UART driver.

Values:

enumerator kStatus_FLEXIO_UART_TxBusy
Transmitter is busy.

enumerator kStatus_FLEXIO_UART_RxBusy
Receiver is busy.

enumerator kStatus_FLEXIO_UART_TxIdle
UART transmitter is idle.

enumerator kStatus_FLEXIO_UART_RxIdle
UART receiver is idle.

enumerator kStatus_FLEXIO_UART_ERROR
ERROR happens on UART.

enumerator kStatus_FLEXIO_UART_RxRingBufferOverrun
UART RX software ring buffer overrun.

enumerator kStatus_FLEXIO_UART_RxHardwareOverrun
UART RX receiver overrun.

enumerator kStatus_FLEXIO_UART_Timeout
UART times out.

enumerator kStatus_FLEXIO_UART_BaudrateNotSupport
Baudrate is not supported in current clock source

enum _flexio_uart_bit_count_per_char
FlexIO UART bit count per char.

Values:

enumerator kFLEXIO_UART_7BitsPerChar
7-bit data characters

enumerator kFLEXIO_UART_8BitsPerChar
8-bit data characters

enumerator kFLEXIO_UART_9BitsPerChar
9-bit data characters

enum _flexio_uart_interrupt_enable
Define FlexIO UART interrupt mask.

Values:

enumerator kFLEXIO_UART_TxDataRegEmptyInterruptEnable
Transmit buffer empty interrupt enable.

enumerator kFLEXIO_UART_RxDataRegFullInterruptEnable
Receive buffer full interrupt enable.

enum _flexio_uart_status_flags
Define FlexIO UART status mask.
Values:

enumerator kFLEXIO_UART_TxDataRegEmptyFlag
Transmit buffer empty flag.

enumerator kFLEXIO_UART_RxDataRegFullFlag
Receive buffer full flag.

enumerator kFLEXIO_UART_RxOverRunFlag
Receive buffer over run flag.

typedef enum *_flexio_uart_bit_count_per_char* flexio_uart_bit_count_per_char_t
FlexIO UART bit count per char.

typedef struct *_flexio_uart_type* FLEXIO_UART_Type
Define FlexIO UART access structure typedef.

typedef struct *_flexio_uart_config* flexio_uart_config_t
Define FlexIO UART user configuration structure.

typedef struct *_flexio_uart_transfer* flexio_uart_transfer_t
Define FlexIO UART transfer structure.

typedef struct *_flexio_uart_handle* flexio_uart_handle_t

typedef void (*flexio_uart_transfer_callback_t)(*FLEXIO_UART_Type* *base, *flexio_uart_handle_t* *handle, *status_t* status, void *userData)
FlexIO UART transfer callback function.

UART_RETRY_TIMES
Retry times for waiting flag.

struct _flexio_uart_type
#include <fsl_flexio_uart.h> Define FlexIO UART access structure typedef.

Public Members

FLEXIO_Type *flexioBase
FlexIO base pointer.

uint8_t TxPinIndex
Pin select for UART_Tx.

uint8_t RxPinIndex
Pin select for UART_Rx.

uint8_t shifterIndex[2]
Shifter index used in FlexIO UART.

uint8_t timerIndex[2]
Timer index used in FlexIO UART.

struct __flexio_uart_config
#include <fsl_flexio_uart.h> Define FlexIO UART user configuration structure.

Public Members

bool enableUart
Enable/disable FlexIO UART TX & RX.

bool enableInDoze
Enable/disable FlexIO operation in doze mode

bool enableInDebug
Enable/disable FlexIO operation in debug mode

bool enableFastAccess
Enable/disable fast access to FlexIO registers, fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.

uint32_t baudRate_Bps
Baud rate in Bps.

flexio_uart_bit_count_per_char_t bitCountPerChar
number of bits, 7/8/9 -bit

struct __flexio_uart_transfer
#include <fsl_flexio_uart.h> Define FlexIO UART transfer structure.

Public Members

size_t dataSize
Transfer size

struct __flexio_uart_handle
#include <fsl_flexio_uart.h> Define FLEXIO UART handle structure.

Public Members

const uint8_t *volatile txData
Address of remaining data to send.

volatile size_t txDataSize
Size of the remaining data to send.

uint8_t *volatile rxData
Address of remaining data to receive.

volatile size_t rxDataSize
Size of the remaining data to receive.

size_t txDataSizeAll
Total bytes to be sent.

size_t rxDataSizeAll
Total bytes to be received.

uint8_t *rxRingBuffer
Start address of the receiver ring buffer.

size_t rxRingBufferSize
Size of the ring buffer.

volatile uint16_t rxRingBufferHead
Index for the driver to store received data into ring buffer.

volatile uint16_t rxRingBufferTail
Index for the user to get data from the ring buffer.

flexio_uart_transfer_callback_t callback
Callback function.

void *userData
UART callback function parameter.

volatile uint8_t txState
TX transfer state.

volatile uint8_t rxState
RX transfer state

union ___unnamed78___

Public Members

uint8_t *data
The buffer of data to be transfer.

uint8_t *rxData
The buffer to receive data.

const uint8_t *txData
The buffer of data to be sent.

2.28 GPIO: General-Purpose Input/Output Driver

FSL_GPIO_DRIVER_VERSION
GPIO driver version.

enum _gpio_pin_direction
GPIO direction definition.

Values:

enumerator kGPIO_DigitalInput
Set current pin as digital input

enumerator kGPIO_DigitalOutput
Set current pin as digital output

enum _gpio_checker_attribute
GPIO checker attribute.

Values:

enumerator kGPIO_UsernonsecureRWUsersecureRWPrivilegedsecureRW
User nonsecure:Read+Write; User Secure:Read+Write; Privileged Secure:Read+Write

enumerator kGPIO_UsernonsecureRUsersecureRWPrivilegedsecureRW
User nonsecure:Read; User Secure:Read+Write; Privileged Secure:Read+Write

enumerator kGPIO_UsernonsecureNUsersecureRWPrivilegedsecureRW
User nonsecure:None; User Secure:Read+Write; Privileged Secure:Read+Write

enumerator kGPIO_UsernonsecureRUsersecureRPrivilegedsecureRW
User nonsecure:Read; User Secure:Read; Privileged Secure:Read+Write

enumerator kGPIO_UsernonsecureNUsersecureRPrivilegedsecureRW
User nonsecure:None; User Secure:Read; Privileged Secure:Read+Write

enumerator kGPIO_UsernonsecureNUsersecureNPrivilegedsecureRW
User nonsecure:None; User Secure:None; Privileged Secure:Read+Write

enumerator kGPIO_UsernonsecureNUsersecureNPrivilegedsecureR
User nonsecure:None; User Secure:None; Privileged Secure:Read

enumerator kGPIO_UsernonsecureNUsersecureNPrivilegedsecureN
User nonsecure:None; User Secure:None; Privileged Secure:None

enumerator kGPIO_IgnoreAttributeCheck
Ignores the attribute check

enum _gpio_interrupt_config
Configures the interrupt generation condition.
Values:

enumerator kGPIO_InterruptStatusFlagDisabled
Interrupt status flag is disabled.

enumerator kGPIO_DMARisingEdge
ISF flag and DMA request on rising edge.

enumerator kGPIO_DMAFallingEdge
ISF flag and DMA request on falling edge.

enumerator kGPIO_DMAEitherEdge
ISF flag and DMA request on either edge.

enumerator kGPIO_FlagRisingEdge
Flag sets on rising edge.

enumerator kGPIO_FlagFallingEdge
Flag sets on falling edge.

enumerator kGPIO_FlagEitherEdge
Flag sets on either edge.

enumerator kGPIO_InterruptLogicZero
Interrupt when logic zero.

enumerator kGPIO_InterruptRisingEdge
Interrupt on rising edge.

enumerator kGPIO_InterruptFallingEdge
Interrupt on falling edge.

enumerator kGPIO_InterruptEitherEdge
Interrupt on either edge.

enumerator kGPIO_InterruptLogicOne
Interrupt when logic one.

enumerator kGPIO_ActiveHighTriggerOutputEnable
Enable active high-trigger output.

enumerator kGPIO_ActiveLowTriggerOutputEnable
Enable active low-trigger output.

enum _gpio_interrupt_selection
Configures the selection of interrupt/DMA request/trigger output.

Values:

enumerator kGPIO_InterruptOutput0
Interrupt/DMA request/trigger output 0.

enumerator kGPIO_InterruptOutput1
Interrupt/DMA request/trigger output 1.

enum gpio_pin_interrupt_control_t
GPIO pin and interrupt control.

Values:

enumerator kGPIO_PinControlNonSecure
Pin Control Non-Secure.

enumerator kGPIO_InterruptControlNonSecure
Interrupt Control Non-Secure.

enumerator kGPIO_PinControlNonPrivilege
Pin Control Non-Privilege.

enumerator kGPIO_InterruptControlNonPrivilege
Interrupt Control Non-Privilege.

typedef enum *_gpio_pin_direction* gpio_pin_direction_t
GPIO direction definition.

typedef enum *_gpio_checker_attribute* gpio_checker_attribute_t
GPIO checker attribute.

typedef struct *_gpio_pin_config* gpio_pin_config_t
The GPIO pin configuration structure.

Each pin can only be configured as either an output pin or an input pin at a time. If configured as an input pin, leave the outputConfig unused. Note that in some use cases, the corresponding port property should be configured in advance with the *PORT_SetPinConfig()*.

typedef enum *_gpio_interrupt_config* gpio_interrupt_config_t
Configures the interrupt generation condition.

typedef enum *_gpio_interrupt_selection* gpio_interrupt_selection_t

Configures the selection of interrupt/DMA request/trigger output.

typedef struct *_gpio_version_info* gpio_version_info_t

GPIO version information.

GPIO_FIT_REG(value)

struct _gpio_pin_config

#include <fsl_gpio.h> The GPIO pin configuration structure.

Each pin can only be configured as either an output pin or an input pin at a time. If configured as an input pin, leave the outputConfig unused. Note that in some use cases, the corresponding port property should be configured in advance with the *PORT_SetPinConfig()*.

Public Members

gpio_pin_direction_t pinDirection

GPIO direction, input or output

uint8_t outputLogic

Set a default output logic, which has no use in input

struct _gpio_version_info

#include <fsl_gpio.h> GPIO version information.

Public Members

uint16_t feature

Feature Specification Number.

uint8_t minor

Minor Version Number.

uint8_t major

Major Version Number.

2.29 GPIO Driver

void GPIO_PortInit(GPIO_Type *base)

Initializes the GPIO peripheral.

This function ungates the GPIO clock.

Parameters

- base – GPIO peripheral base pointer.

void GPIO_PortDenit(GPIO_Type *base)

Denitializes the GPIO peripheral.

Parameters

- base – GPIO peripheral base pointer.

void GPIO_PinInit(GPIO_Type *base, uint32_t pin, const *gpio_pin_config_t* *config)

Initializes a GPIO pin used by the board.

To initialize the GPIO, define a pin configuration, as either input or output, in the user file. Then, call the *GPIO_PinInit()* function.

This is an example to define an input pin or an output pin configuration.

```
Define a digital input pin configuration,
gpio_pin_config_t config =
{
  kGPIO_DigitalInput,
  0,
}
Define a digital output pin configuration,
gpio_pin_config_t config =
{
  kGPIO_DigitalOutput,
  0,
}
```

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- pin – GPIO port pin number
- config – GPIO pin configuration pointer

void GPIO_GetVersionInfo(GPIO_Type *base, *gpio_version_info_t* *verInfo)

Get GPIO version information.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- verInfo – GPIO version information

static inline void GPIO_SecurePrivilegeLock(GPIO_Type *base, *gpio_pin_interrupt_control_t* mask)

lock or unlock secure privilege.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – pin or interrupt macro

static inline void GPIO_EnablePinControlNonSecure(GPIO_Type *base, uint32_t mask)

Enable Pin Control Non-Secure.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

static inline void GPIO_DisablePinControlNonSecure(GPIO_Type *base, uint32_t mask)

Disable Pin Control Non-Secure.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

static inline void GPIO_EnablePinControlNonPrivilege(GPIO_Type *base, uint32_t mask)
Enable Pin Control Non-Privilege.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

static inline void GPIO_DisablePinControlNonPrivilege(GPIO_Type *base, uint32_t mask)
Disable Pin Control Non-Privilege.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

static inline void GPIO_EnableInterruptControlNonSecure(GPIO_Type *base, uint32_t mask)
Enable Interrupt Control Non-Secure.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

static inline void GPIO_DisableInterruptControlNonSecure(GPIO_Type *base, uint32_t mask)
Disable Interrupt Control Non-Secure.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

static inline void GPIO_EnableInterruptControlNonPrivilege(GPIO_Type *base, uint32_t mask)
Enable Interrupt Control Non-Privilege.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

static inline void GPIO_DisableInterruptControlNonPrivilege(GPIO_Type *base, uint32_t mask)
Disable Interrupt Control Non-Privilege.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

static inline void GPIO_PortInputEnable(GPIO_Type *base, uint32_t mask)
Enable port input.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

static inline void GPIO_PortInputDisable(GPIO_Type *base, uint32_t mask)
Disable port input.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

static inline void GPIO_PinWrite(GPIO_Type *base, uint32_t pin, uint8_t output)

Sets the output level of the multiple GPIO pins to the logic 1 or 0.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- pin – GPIO pin number
- output – GPIO pin output logic level.
  - 0: corresponding pin output low-logic level.
  - 1: corresponding pin output high-logic level.

static inline void GPIO_PortSet(GPIO_Type *base, uint32_t mask)

Sets the output level of the multiple GPIO pins to the logic 1.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

static inline void GPIO_PortClear(GPIO_Type *base, uint32_t mask)

Sets the output level of the multiple GPIO pins to the logic 0.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

static inline void GPIO_PortToggle(GPIO_Type *base, uint32_t mask)

Reverses the current output logic of the multiple GPIO pins.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

static inline uint32_t GPIO_PinRead(GPIO_Type *base, uint32_t pin)

Reads the current input value of the GPIO port.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- pin – GPIO pin number

Return values

GPIO – port input value

- 0: corresponding pin input low-logic level.
- 1: corresponding pin input high-logic level.

static inline void GPIO_SetPinInterruptConfig(GPIO_Type *base, uint32_t pin, *gpio_interrupt_config_t* config)

Configures the gpio pin interrupt/DMA request.

Parameters

- base – GPIO peripheral base pointer.
- pin – GPIO pin number.
- config – GPIO pin interrupt configuration.

- *kGPIO_InterruptStatusFlagDisabled*: Interrupt/DMA request disabled.
- *kGPIO_DMARisingEdge* : DMA request on rising edge(if the DMA requests exit).
- *kGPIO_DMAFallingEdge*: DMA request on falling edge(if the DMA requests exit).
- *kGPIO_DMAEitherEdge* : DMA request on either edge(if the DMA requests exit).
- *kGPIO_FlagRisingEdge* : Flag sets on rising edge(if the Flag states exit).
- *kGPIO_FlagFallingEdge* : Flag sets on falling edge(if the Flag states exit).
- *kGPIO_FlagEitherEdge* : Flag sets on either edge(if the Flag states exit).
- *kGPIO_InterruptLogicZero* : Interrupt when logic zero.
- *kGPIO_InterruptRisingEdge* : Interrupt on rising edge.
- *kGPIO_InterruptFallingEdge*: Interrupt on falling edge.
- *kGPIO_InterruptEitherEdge* : Interrupt on either edge.
- *kGPIO_InterruptLogicOne* : Interrupt when logic one.
- *kGPIO_ActiveHighTriggerOutputEnable* : Enable active high-trigger output (if the trigger states exit).
- *kGPIO_ActiveLowTriggerOutputEnable* : Enable active low-trigger output (if the trigger states exit).

static inline void GPIO_SetPinInterruptChannel(GPIO_Type *base, uint32_t pin, *gpio_interrupt_selection_t* selection)

Configures the gpio pin interrupt/DMA request/trigger output channel selection.

Parameters

- base – GPIO peripheral base pointer.
- pin – GPIO pin number.
- selection – GPIO pin interrupt output selection.
  - *kGPIO_InterruptOutput0*: Interrupt/DMA request/trigger output 0.
  - *kGPIO_InterruptOutput1* : Interrupt/DMA request/trigger output 1.

uint32_t GPIO_GpioGetInterruptFlags(GPIO_Type *base)

Read the GPIO interrupt status flags.

Parameters

- base – GPIO peripheral base pointer. (GPIOA, GPIOB, GPIOC, and so on.)

Returns

The current GPIO's interrupt status flag. '1' means the related pin's flag is set, '0' means the related pin's flag not set. For example, the return value 0x00010001 means the pin 0 and 17 have the interrupt pending.

uint32_t GPIO_GpioGetInterruptChannelFlags(GPIO_Type *base, uint32_t channel)

Read the GPIO interrupt status flags based on selected interrupt channel(IRQS).

Parameters

- base – GPIO peripheral base pointer. (GPIOA, GPIOB, GPIOC, and so on.)
- channel – '0' means selete interrupt channel 0, '1' means selete interrupt channel 1.

Returns
The current GPIO's interrupt status flag based on the selected interrupt channel. '1' means the related pin's flag is set, '0' means the related pin's flag not set. For example, the return value 0x00010001 means the pin 0 and 17 have the interrupt pending.

uint8_t GPIO_PinGetInterruptFlag(GPIO_Type *base, uint32_t pin)

Read individual pin's interrupt status flag.

Parameters

- base – GPIO peripheral base pointer. (GPIOA, GPIOB, GPIOC, and so on)
- pin – GPIO specific pin number.

Returns
The current selected pin's interrupt status flag.

void GPIO_GpioClearInterruptFlags(GPIO_Type *base, uint32_t mask)

Clears GPIO pin interrupt status flags.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

void GPIO_GpioClearInterruptChannelFlags(GPIO_Type *base, uint32_t mask, uint32_t channel)

Clears GPIO pin interrupt status flags based on selected interrupt channel(IRQS).

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro
- channel – '0' means selete interrupt channel 0, '1' means selete interrupt channel 1.

void GPIO_PinClearInterruptFlag(GPIO_Type *base, uint32_t pin)

Clear GPIO individual pin's interrupt status flag.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on).
- pin – GPIO specific pin number.

static inline void GPIO_SetMultipleInterruptPinsConfig(GPIO_Type *base, uint32_t mask, *gpio_interrupt_config_t* config)

Sets the GPIO interrupt configuration in PCR register for multiple pins.

Parameters

- base – GPIO peripheral base pointer.
- mask – GPIO pin number macro.
- config – GPIO pin interrupt configuration.
  - *kGPIO_InterruptStatusFlagDisabled*: Interrupt disabled.
  - *kGPIO_DMARisingEdge* : DMA request on rising edge(if the DMA requests exit).
  - *kGPIO_DMAFallingEdge*: DMA request on falling edge(if the DMA requests exit).
  - *kGPIO_DMAEitherEdge* : DMA request on either edge(if the DMA requests exit).

- *kGPIO_FlagRisingEdge* : Flag sets on rising edge(if the Flag states exit).
- *kGPIO_FlagFallingEdge* : Flag sets on falling edge(if the Flag states exit).
- *kGPIO_FlagEitherEdge* : Flag sets on either edge(if the Flag states exit).
- *kGPIO_InterruptLogicZero* : Interrupt when logic zero.
- *kGPIO_InterruptRisingEdge* : Interrupt on rising edge.
- *kGPIO_InterruptFallingEdge*: Interrupt on falling edge.
- *kGPIO_InterruptEitherEdge* : Interrupt on either edge.
- *kGPIO_InterruptLogicOne* : Interrupt when logic one.
- *kGPIO_ActiveHighTriggerOutputEnable* : Enable active high-trigger output (if the trigger states exit).
- *kGPIO_ActiveLowTriggerOutputEnable* : Enable active low-trigger output (if the trigger states exit)..

void GPIO_CheckAttributeBytes(GPIO_Type *base, *gpio_checker_attribute_t* attribute)

brief The GPIO module supports a device-specific number of data ports, organized as 32-bit words/8-bit Bytes. Each 32-bit/8-bit data port includes a GACR register, which defines the byte-level attributes required for a successful access to the GPIO programming model. If the GPIO module's GACR register organized as 32-bit words, the attribute controls for the 4 data bytes in the GACR follow a standard little endian data convention.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- attribute – GPIO checker attribute

GPIO_PortGetInterruptFlags(base)

GPIO_PortClearInterruptFlags(base, mask)

2.30 I3C: I3C Driver

FSL_I3C_DRIVER_VERSION

I3C driver version.

I3C status return codes.

Values:

enumerator kStatus_I3C_Busy

The master is already performing a transfer.

enumerator kStatus_I3C_Idle

The slave driver is idle.

enumerator kStatus_I3C_Nak

The slave device sent a NAK in response to an address.

enumerator kStatus_I3C_WriteAbort
The slave device sent a NAK in response to a write.

enumerator kStatus_I3C_Term
The master terminates slave read.

enumerator kStatus_I3C_HdrParityError
Parity error from DDR read.

enumerator kStatus_I3C_CrcError
CRC error from DDR read.

enumerator kStatus_I3C_ReadFifoError
Read from M/SRDATAB register when FIFO empty.

enumerator kStatus_I3C_WriteFifoError
Write to M/SWDATAB register when FIFO full.

enumerator kStatus_I3C_MsgError
Message SDR/DDR mismatch or read/write message in wrong state

enumerator kStatus_I3C_InvalidReq
Invalid use of request.

enumerator kStatus_I3C_Timeout
The module has stalled too long in a frame.

enumerator kStatus_I3C_SlaveCountExceed
The I3C slave count has exceed the definition in I3C_MAX_DEVCNT.

enumerator kStatus_I3C_IBIWon
The I3C slave event IBI or MR or HJ won the arbitration on a header address.

enumerator kStatus_I3C_OverrunError
Slave internal from-bus buffer/FIFO overrun.

enumerator kStatus_I3C_UnderrunError
Slave internal to-bus buffer/FIFO underrun

enumerator kStatus_I3C_UnderrunNak
Slave internal from-bus buffer/FIFO underrun and NACK error

enumerator kStatus_I3C_InvalidStart
Slave invalid start flag

enumerator kStatus_I3C_SdrParityError
SDR parity error

enumerator kStatus_I3C_S0S1Error
S0 or S1 error

enum _i3c_hdr_mode
I3C HDR modes.

Values:

enumerator kI3C_HDRModeNone

enumerator kI3C_HDRModeDDR

enumerator kI3C_HDRModeTSP

enumerator kI3C_HDRModeTSL

typedef enum *_i3c_hdr_mode* i3c_hdr_mode_t
I3C HDR modes.

typedef struct *_i3c_device_info* i3c_device_info_t
I3C device information.

I3C_RETRY_TIMES
Max loops to wait for I3C operation status complete.

This is the maximum number of loops to wait for I3C operation status complete. If set to 0, it will wait indefinitely.

I3C_MAX_DEVCNT

I3C_IBI_BUFF_SIZE

struct _i3c_device_info
#include <fsl_i3c.h> I3C device information.

Public Members

uint8_t dynamicAddr
Device dynamic address.

uint8_t staticAddr
Static address.

uint8_t dcr
Device characteristics register information.

uint8_t bcr
Bus characteristics register information.

uint16_t vendorID
Device vendor ID(manufacture ID).

uint32_t partNumber
Device part number info

uint16_t maxReadLength
Maximum read length.

uint16_t maxWriteLength
Maximum write length.

uint8_t hdrMode
Support hdr mode, could be OR logic in i3c_hdr_mode.

2.31 I3C Common Driver

typedef struct *_i3c_config* i3c_config_t
Structure with settings to initialize the I3C module, could both initialize master and slave functionality.

This structure holds configuration settings for the I3C peripheral. To initialize this structure to reasonable defaults, call the *I3C_GetDefaultConfig()* function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

uint32_t I3C_GetInstance(I3C_Type *base)
Get which instance current I3C is used.

Parameters

- base – The I3C peripheral base address.

void I3C_GetDefaultConfig(*i3c_config_t* *config)
Provides a default configuration for the I3C peripheral, the configuration covers both master functionality and slave functionality.

This function provides the following default configuration for I3C:

```
config->enableMaster                = kI3C_MasterCapable;
config->disableTimeout              = false;
config->hKeep                       = kI3C_MasterHighKeeperNone;
config->enableOpenDrainStop         = true;
config->enableOpenDrainHigh         = true;
config->baudRate_Hz.i2cBaud         = 400000U;
config->baudRate_Hz.i3cPushPullBaud = 12500000U;
config->baudRate_Hz.i3cOpenDrainBaud = 2500000U;
config->masterDynamicAddress        = 0x0AU;
config->slowClock_Hz                = 1000000U;
config->enableSlave                 = true;
config->vendorID                    = 0x11BU;
config->enableRandomPart            = false;
config->partNumber                  = 0;
config->dcr                         = 0;
config->bcr = 0;
```

(continues on next page)

(continued from previous page)

```
config->hdrMode            = (uint8_t)kI3C_HDRModeDDR;
config->nakAllRequest      = false;
config->ignoreS0S1Error    = false;
config->offline            = false;
config->matchSlaveStartStop = false;
```

After calling this function, you can override any settings in order to customize the configuration, prior to initializing the common I3C driver with *I3C_Init()*.

Parameters

- config – **[out]** User provided configuration structure for default values. Refer to *i3c_config_t*.

void I3C_Init(I3C_Type *base, const *i3c_config_t* *config, uint32_t sourceClock_Hz)

Initializes the I3C peripheral. This function enables the peripheral clock and initializes the I3C peripheral as described by the user provided configuration. This will initialize both the master peripheral and slave peripheral so that I3C module could work as pure master, pure slave or secondary master, etc. A software reset is performed prior to configuration.

Parameters

- base – The I3C peripheral base address.
- config – User provided peripheral configuration. Use *I3C_GetDefaultConfig()* to get a set of defaults that you can override.
- sourceClock_Hz – Frequency in Hertz of the I3C functional clock. Used to calculate the baud rate divisors, filter widths, and timeout periods.

struct _i3c_config

#include <fsl_i3c.h> Structure with settings to initialize the I3C module, could both initialize master and slave functionality.

This structure holds configuration settings for the I3C peripheral. To initialize this structure to reasonable defaults, call the *I3C_GetDefaultConfig()* function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

Public Members

i3c_master_enable_t enableMaster

Enable master mode.

bool disableTimeout

Whether to disable timeout to prevent the ERRWARN.

i3c_master_hkeep_t hKeep

High keeper mode setting.

bool enableOpenDrainStop

Whether to emit open-drain speed STOP.

bool enableOpenDrainHigh

Enable Open-Drain High to be 1 PPBAUD count for i3c messages, or 1 ODBAUD.

i3c_baudrate_hz_t baudRate_Hz
Desired baud rate settings.

i3c_start_scl_delay_t startSclDelay
I3C SCL delay after START.

i3c_start_scl_delay_t restartSclDelay
I3C SCL delay after Repeated START.

uint8_t masterDynamicAddress
Main master dynamic address configuration.

uint32_t maxWriteLength
Maximum write length.

uint32_t maxReadLength
Maximum read length.

bool enableSlave
Whether to enable slave.

uint8_t staticAddr
Static address.

uint16_t vendorID
Device vendor ID(manufacture ID).

uint32_t partNumber
Device part number info

uint8_t dcr
Device characteristics register information.

uint8_t bcr
Bus characteristics register information.

uint8_t hdrMode
Support hdr mode, could be OR logic in enumeration:i3c_hdr_mode_t.

bool nakAllRequest
Whether to reply NAK to all requests except broadcast CCC.

bool ignoreS0S1Error
Whether to ignore S0/S1 error in SDR mode.

bool offline
Whether to wait 60 us of bus quiet or HDR request to ensure slave track SDR mode safely.

bool matchSlaveStartStop

Whether to assert start/stop status only the time slave is addressed.

2.32 I3C Master Driver

void I3C_MasterGetDefaultConfig(*i3c_master_config_t* *masterConfig)

Provides a default configuration for the I3C master peripheral.

This function provides the following default configuration for the I3C master peripheral:

```
masterConfig->enableMaster          = kI3C_MasterOn;
masterConfig->disableTimeout        = false;
masterConfig->hKeep                 = kI3C_MasterHighKeeperNone;
masterConfig->enableOpenDrainStop   = true;
masterConfig->enableOpenDrainHigh   = true;
masterConfig->baudRate_Hz           = 100000U;
masterConfig->busType               = kI3C_TypeI2C;
```

After calling this function, you can override any settings in order to customize the configuration, prior to initializing the master driver with *I3C_MasterInit()*.

Parameters

- masterConfig – **[out]** User provided configuration structure for default values. Refer to *i3c_master_config_t*.

void I3C_MasterInit(I3C_Type *base, const *i3c_master_config_t* *masterConfig, uint32_t sourceClock_Hz)

Initializes the I3C master peripheral.

This function enables the peripheral clock and initializes the I3C master peripheral as described by the user provided configuration. A software reset is performed prior to configuration.

Parameters

- base – The I3C peripheral base address.
- masterConfig – User provided peripheral configuration. Use *I3C_MasterGetDefaultConfig()* to get a set of defaults that you can override.
- sourceClock_Hz – Frequency in Hertz of the I3C functional clock. Used to calculate the baud rate divisors, filter widths, and timeout periods.

void I3C_MasterDeinit(I3C_Type *base)

Deinitializes the I3C master peripheral.

This function disables the I3C master peripheral and gates the clock. It also performs a software reset to restore the peripheral to reset conditions.

Parameters

- base – The I3C peripheral base address.

static inline void I3C_MasterEnable(I3C_Type *base, *i3c_master_enable_t* enable)

Set I3C module master mode.

Parameters

- base – The I3C peripheral base address.
- enable – Enable master mode.

status_t I3C_MasterCheckAndClearError(I3C_Type *base, uint32_t status)

status_t I3C_MasterWaitForCtrlDone(I3C_Type *base, bool waitIdle)

status_t I3C_CheckForBusyBus(I3C_Type *base)

void I3C_SlaveGetDefaultConfig(*i3c_slave_config_t* *slaveConfig)

Provides a default configuration for the I3C slave peripheral.

This function provides the following default configuration for the I3C slave peripheral:

```
slaveConfig->enableslave          = true;
```

After calling this function, you can override any settings in order to customize the configuration, prior to initializing the slave driver with I3C_SlaveInit().

Parameters

- slaveConfig – **[out]** User provided configuration structure for default values. Refer to *i3c_slave_config_t*.

void I3C_SlaveInit(I3C_Type *base, const *i3c_slave_config_t* *slaveConfig, uint32_t slowClock_Hz)

Initializes the I3C slave peripheral.

This function enables the peripheral clock and initializes the I3C slave peripheral as described by the user provided configuration.

Parameters

- base – The I3C peripheral base address.
- slaveConfig – User provided peripheral configuration. Use I3C_SlaveGetDefaultConfig() to get a set of defaults that you can override.
- slowClock_Hz – Frequency in Hertz of the I3C slow clock. Used to calculate the bus match condition values. If FSL_FEATURE_I3C_HAS_NO_SCONFIG_BAMATCH defines as 1, this parameter is useless.

void I3C_SlaveDeinit(I3C_Type *base)

Deinitializes the I3C slave peripheral.

This function disables the I3C slave peripheral and gates the clock.

Parameters

- base – The I3C peripheral base address.

static inline void I3C_SlaveEnable(I3C_Type *base, bool isEnable)

Enable/Disable Slave.

Parameters

- base – The I3C peripheral base address.
- isEnable – Enable or disable.

static inline uint32_t I3C_MasterGetStatusFlags(I3C_Type *base)

Gets the I3C master status flags.

A bit mask with the state of all I3C master status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

See also:

_i3c_master_flags

Parameters

- base – The I3C peripheral base address.

Returns

State of the status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

static inline void I3C_MasterClearStatusFlags(I3C_Type *base, uint32_t statusMask)

Clears the I3C master status flag state.

The following status register flags can be cleared:

- *kI3C_MasterSlaveStartFlag*
- *kI3C_MasterControlDoneFlag*
- *kI3C_MasterCompleteFlag*
- *kI3C_MasterArbitrationWonFlag*
- *kI3C_MasterSlave2MasterFlag*

Attempts to clear other flags has no effect.

See also:

_i3c_master_flags.

Parameters

- base – The I3C peripheral base address.
- statusMask – A bitmask of status flags that are to be cleared. The mask is composed of *_i3c_master_flags* enumerators OR'd together. You may pass the result of a previous call to *I3C_MasterGetStatusFlags()*.

static inline uint32_t I3C_MasterGetErrorStatusFlags(I3C_Type *base)

Gets the I3C master error status flags.

A bit mask with the state of all I3C master error status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

See also:

_i3c_master_error_flags

Parameters

- base – The I3C peripheral base address.

Returns

State of the error status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

static inline void I3C_MasterClearErrorStatusFlags(I3C_Type *base, uint32_t statusMask)

Clears the I3C master error status flag state.

See also:

_i3c_master_error_flags.

Parameters

- base – The I3C peripheral base address.
- statusMask – A bitmask of error status flags that are to be cleared. The mask is composed of *_i3c_master_error_flags* enumerators OR'd together. You may pass the result of a previous call to *I3C_MasterGetStatusFlags()*.

i3c_master_state_t I3C_MasterGetState(I3C_Type *base)

Gets the I3C master state.

Parameters

- base – The I3C peripheral base address.

Returns

I3C master state.

static inline uint32_t I3C_SlaveGetStatusFlags(I3C_Type *base)

Gets the I3C slave status flags.

A bit mask with the state of all I3C slave status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

See also:

_i3c_slave_flags

Parameters

- base – The I3C peripheral base address.

Returns

State of the status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

static inline void I3C_SlaveClearStatusFlags(I3C_Type *base, uint32_t statusMask)

Clears the I3C slave status flag state.

The following status register flags can be cleared:

- *kI3C_SlaveBusStartFlag*
- *kI3C_SlaveMatchedFlag*
- *kI3C_SlaveBusStopFlag*

Attempts to clear other flags has no effect.

See also:

_i3c_slave_flags.

Parameters

- base – The I3C peripheral base address.
- statusMask – A bitmask of status flags that are to be cleared. The mask is composed of *_i3c_slave_flags* enumerators OR'd together. You may pass the result of a previous call to I3C_SlaveGetStatusFlags().

static inline uint32_t I3C_SlaveGetErrorStatusFlags(I3C_Type *base)

Gets the I3C slave error status flags.

A bit mask with the state of all I3C slave error status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

See also:

_i3c_slave_error_flags

Parameters

- base – The I3C peripheral base address.

Returns

State of the error status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

static inline void I3C_SlaveClearErrorStatusFlags(I3C_Type *base, uint32_t statusMask)

Clears the I3C slave error status flag state.

See also:

_i3c_slave_error_flags.

Parameters

- base – The I3C peripheral base address.
- statusMask – A bitmask of error status flags that are to be cleared. The mask is composed of *_i3c_slave_error_flags* enumerators OR'd together. You may pass the result of a previous call to I3C_SlaveGetErrorStatusFlags().

i3c_slave_activity_state_t I3C_SlaveGetActivityState(I3C_Type *base)

Gets the I3C slave state.

Parameters

- base – The I3C peripheral base address.

Returns

I3C slave activity state, refer *i3c_slave_activity_state_t*.

status_t I3C_SlaveCheckAndClearError(I3C_Type *base, uint32_t status)

static inline void I3C_MasterEnableInterrupts(I3C_Type *base, uint32_t interruptMask)

Enables the I3C master interrupt requests.

All flags except *kI3C_MasterBetweenFlag* and *kI3C_MasterNackDetectFlag* can be enabled as interrupts.

Parameters

- base – The I3C peripheral base address.

- interruptMask – Bit mask of interrupts to enable. See *_i3c_master_flags* for the set of constants that should be OR'd together to form the bit mask.

static inline void I3C_MasterDisableInterrupts(I3C_Type *base, uint32_t interruptMask)

Disables the I3C master interrupt requests.

All flags except *kI3C_MasterBetweenFlag* and *kI3C_MasterNackDetectFlag* can be enabled as interrupts.

Parameters

- base – The I3C peripheral base address.
- interruptMask – Bit mask of interrupts to disable. See *_i3c_master_flags* for the set of constants that should be OR'd together to form the bit mask.

static inline uint32_t I3C_MasterGetEnabledInterrupts(I3C_Type *base)

Returns the set of currently enabled I3C master interrupt requests.

Parameters

- base – The I3C peripheral base address.

Returns

A bitmask composed of *_i3c_master_flags* enumerators OR'd together to indicate the set of enabled interrupts.

static inline uint32_t I3C_MasterGetPendingInterrupts(I3C_Type *base)

Returns the set of pending I3C master interrupt requests.

Parameters

- base – The I3C peripheral base address.

Returns

A bitmask composed of *_i3c_master_flags* enumerators OR'd together to indicate the set of pending interrupts.

static inline void I3C_SlaveEnableInterrupts(I3C_Type *base, uint32_t interruptMask)

Enables the I3C slave interrupt requests.

Only below flags can be enabled as interrupts.

- *kI3C_SlaveBusStartFlag*
- *kI3C_SlaveMatchedFlag*
- *kI3C_SlaveBusStopFlag*
- *kI3C_SlaveRxReadyFlag*
- *kI3C_SlaveTxReadyFlag*
- *kI3C_SlaveDynamicAddrChangedFlag*
- *kI3C_SlaveReceivedCCCFlag*
- *kI3C_SlaveErrorFlag*
- *kI3C_SlaveHDRCommandMatchFlag*
- *kI3C_SlaveCCCHandledFlag*
- *kI3C_SlaveEventSentFlag*

Parameters

- base – The I3C peripheral base address.
- interruptMask – Bit mask of interrupts to enable. See *_i3c_slave_flags* for the set of constants that should be OR'd together to form the bit mask.

static inline void I3C_SlaveDisableInterrupts(I3C_Type *base, uint32_t interruptMask)

Disables the I3C slave interrupt requests.

Only below flags can be disabled as interrupts.

- *kI3C_SlaveBusStartFlag*
- *kI3C_SlaveMatchedFlag*
- *kI3C_SlaveBusStopFlag*
- *kI3C_SlaveRxReadyFlag*
- *kI3C_SlaveTxReadyFlag*
- *kI3C_SlaveDynamicAddrChangedFlag*
- *kI3C_SlaveReceivedCCCFlag*
- *kI3C_SlaveErrorFlag*
- *kI3C_SlaveHDRCommandMatchFlag*
- *kI3C_SlaveCCCHandledFlag*
- *kI3C_SlaveEventSentFlag*

Parameters

- base – The I3C peripheral base address.
- interruptMask – Bit mask of interrupts to disable. See *_i3c_slave_flags* for the set of constants that should be OR'd together to form the bit mask.

static inline uint32_t I3C_SlaveGetEnabledInterrupts(I3C_Type *base)

Returns the set of currently enabled I3C slave interrupt requests.

Parameters

- base – The I3C peripheral base address.

Returns

A bitmask composed of *_i3c_slave_flags* enumerators OR'd together to indicate the set of enabled interrupts.

static inline uint32_t I3C_SlaveGetPendingInterrupts(I3C_Type *base)

Returns the set of pending I3C slave interrupt requests.

Parameters

- base – The I3C peripheral base address.

Returns

A bitmask composed of *_i3c_slave_flags* enumerators OR'd together to indicate the set of pending interrupts.

static inline void I3C_MasterEnableDMA(I3C_Type *base, bool enableTx, bool enableRx, uint32_t width)

Enables or disables I3C master DMA requests.

Parameters

- base – The I3C peripheral base address.
- enableTx – Enable flag for transmit DMA request. Pass true for enable, false for disable.
- enableRx – Enable flag for receive DMA request. Pass true for enable, false for disable.
- width – DMA read/write unit in bytes.

static inline uint32_t I3C_MasterGetTxFifoAddress(I3C_Type *base, uint32_t width)

Gets I3C master transmit data register address for DMA transfer.

Parameters

- base – The I3C peripheral base address.
- width – DMA read/write unit in bytes.

Returns

The I3C Master Transmit Data Register address.

static inline uint32_t I3C_MasterGetRxFifoAddress(I3C_Type *base, uint32_t width)

Gets I3C master receive data register address for DMA transfer.

Parameters

- base – The I3C peripheral base address.
- width – DMA read/write unit in bytes.

Returns

The I3C Master Receive Data Register address.

static inline void I3C_SlaveEnableDMA(I3C_Type *base, bool enableTx, bool enableRx, uint32_t width)

Enables or disables I3C slave DMA requests.

Parameters

- base – The I3C peripheral base address.
- enableTx – Enable flag for transmit DMA request. Pass true for enable, false for disable.
- enableRx – Enable flag for receive DMA request. Pass true for enable, false for disable.
- width – DMA read/write unit in bytes.

static inline uint32_t I3C_SlaveGetTxFifoAddress(I3C_Type *base, uint32_t width)

Gets I3C slave transmit data register address for DMA transfer.

Parameters

- base – The I3C peripheral base address.
- width – DMA read/write unit in bytes.

Returns

The I3C Slave Transmit Data Register address.

static inline uint32_t I3C_SlaveGetRxFifoAddress(I3C_Type *base, uint32_t width)

Gets I3C slave receive data register address for DMA transfer.

Parameters

- base – The I3C peripheral base address.
- width – DMA read/write unit in bytes.

Returns

The I3C Slave Receive Data Register address.

static inline void I3C_MasterSetWatermarks(I3C_Type *base, *i3c_tx_trigger_level_t* txLvl, *i3c_rx_trigger_level_t* rxLvl, bool flushTx, bool flushRx)

Sets the watermarks for I3C master FIFOs.

Parameters

- base – The I3C peripheral base address.
- txLvl – Transmit FIFO watermark level. The *kI3C_MasterTxReadyFlag* flag is set whenever the number of words in the transmit FIFO reaches *txLvl*.
- rxLvl – Receive FIFO watermark level. The *kI3C_MasterRxReadyFlag* flag is set whenever the number of words in the receive FIFO reaches *rxLvl*.
- flushTx – true if TX FIFO is to be cleared, otherwise TX FIFO remains unchanged.
- flushRx – true if RX FIFO is to be cleared, otherwise RX FIFO remains unchanged.

static inline void I3C_MasterGetFifoCounts(I3C_Type *base, size_t *rxCount, size_t *txCount)

Gets the current number of bytes in the I3C master FIFOs.

Parameters

- base – The I3C peripheral base address.
- txCount – **[out]** Pointer through which the current number of bytes in the transmit FIFO is returned. Pass NULL if this value is not required.
- rxCount – **[out]** Pointer through which the current number of bytes in the receive FIFO is returned. Pass NULL if this value is not required.

static inline void I3C_SlaveSetWatermarks(I3C_Type *base, *i3c_tx_trigger_level_t* txLvl, *i3c_rx_trigger_level_t* rxLvl, bool flushTx, bool flushRx)

Sets the watermarks for I3C slave FIFOs.

Parameters

- base – The I3C peripheral base address.
- txLvl – Transmit FIFO watermark level. The *kI3C_SlaveTxReadyFlag* flag is set whenever the number of words in the transmit FIFO reaches *txLvl*.
- rxLvl – Receive FIFO watermark level. The *kI3C_SlaveRxReadyFlag* flag is set whenever the number of words in the receive FIFO reaches *rxLvl*.
- flushTx – true if TX FIFO is to be cleared, otherwise TX FIFO remains unchanged.
- flushRx – true if RX FIFO is to be cleared, otherwise RX FIFO remains unchanged.

static inline void I3C_SlaveGetFifoCounts(I3C_Type *base, size_t *rxCount, size_t *txCount)

Gets the current number of bytes in the I3C slave FIFOs.

Parameters

- base – The I3C peripheral base address.
- txCount – **[out]** Pointer through which the current number of bytes in the transmit FIFO is returned. Pass NULL if this value is not required.
- rxCount – **[out]** Pointer through which the current number of bytes in the receive FIFO is returned. Pass NULL if this value is not required.

void I3C_MasterSetBaudRate(I3C_Type *base, const *i3c_baudrate_hz_t* *baudRate_Hz, uint32_t sourceClock_Hz)

Sets the I3C bus frequency for master transactions.

The I3C master is automatically disabled and re-enabled as necessary to configure the baud rate. Do not call this function during a transfer, or the transfer is aborted.

Note: On devices affected by I3C errata ERR053429, an SDR read with PPBAUD < 2 can lose the last received byte when the RxFIFO fills before the CPU/DMA services it. For safe SDR reads, request a push-pull baud that yields PPBAUD > 1, or guarantee the CPU/DMA drains the RxFIFO fast enough to avoid a Full event.

Parameters

- base – The I3C peripheral base address.
- baudRate_Hz – Pointer to structure of requested bus frequency in Hertz.
- sourceClock_Hz – I3C functional clock frequency in Hertz.

status_t I3C_MasterGetI2CPresetTiming(*i3c_legacy_i2c_speed_mode_t* speed, *i3c_master_src_clk_t* srcClk, *i3c_timing_t* *timing)

Get prevalidated timing values for supported I2C speed.

Parameters

- speed – **[in]** I2C Speed mode.
- srcClk – **[in]** FCLK source clock.
- timing – **[out]** Prevalidated timing values. Valid when kStatus_Success is returned.

Returns

kStatus_Success A supported preset timing is found. kStatus_Fail The requested timing is not supported.

status_t I3C_MasterCalcTiming(const *i3c_master_baudrate_config_t* *baudRate, uint32_t sourceClock_Hz, *i3c_timing_t* *timing)

Calculate timing from desired baud rates.

Parameters

- baudRate – **[in]** Desired baud rates and bus configuration.
- sourceClock_Hz – **[in]** Master peripheral clock (FCLK) frequency in Hz.
- timing – **[out]** I3C timing register field values. Valid only when kStatus_Success is returned.

Returns

kStatus_Success Target rate is achieved. kStatus_Fail Target rate isn't achieved.

void I3C_MasterSetTiming(I3C_Type *base, const *i3c_timing_t* *timing)

Set I3C timing parameters.

Parameters

- base – I3C peripheral base address.
- timing – Pre-calculated timing values.

static inline bool I3C_MasterGetBusIdleState(I3C_Type *base)

Returns whether the bus is idle.

Requires the master mode to be enabled.

Parameters

- base – The I3C peripheral base address.

Return values

- true – Bus is busy.
- false – Bus is idle.

status_t I3C_MasterStartWithRxSize(I3C_Type *base, *i3c_bus_type_t* type, uint8_t address, *i3c_direction_t* dir, uint8_t rxSize)

Sends a START signal and slave address on the I2C/I3C bus, receive size is also specified in the call.

This function is used to initiate a new master mode transfer. First, the bus state is checked to ensure that another master is not occupying the bus. Then a START signal is transmitted, followed by the 7-bit address specified in the a address parameter. Note that this function does not actually wait until the START and address are successfully sent on the bus before returning.

Parameters

- base – The I3C peripheral base address.
- type – The bus type to use in this transaction.
- address – 7-bit slave device address, in bits [6:0].
- dir – Master transfer direction, either *kI3C_Read* or *kI3C_Write*. This parameter is used to set the R/w bit (bit 0) in the transmitted slave address.
- rxSize – Read terminate size for the followed read transfer, limit to 255 bytes.

Return values

- kStatus_Success – START signal and address were successfully enqueued in the transmit FIFO.
- kStatus_I3C_Busy – Another master is currently utilizing the bus.

status_t I3C_MasterStart(I3C_Type *base, *i3c_bus_type_t* type, uint8_t address, *i3c_direction_t* dir)

Sends a START signal and slave address on the I2C/I3C bus.

This function is used to initiate a new master mode transfer. First, the bus state is checked to ensure that another master is not occupying the bus. Then a START signal is transmitted, followed by the 7-bit address specified in the *address* parameter. Note that this function does not actually wait until the START and address are successfully sent on the bus before returning.

Parameters

- base – The I3C peripheral base address.
- type – The bus type to use in this transaction.
- address – 7-bit slave device address, in bits [6:0].
- dir – Master transfer direction, either *kI3C_Read* or *kI3C_Write*. This parameter is used to set the R/w bit (bit 0) in the transmitted slave address.

Return values

- kStatus_Success – START signal and address were successfully enqueued in the transmit FIFO.
- kStatus_I3C_Busy – Another master is currently utilizing the bus.

status_t I3C_MasterRepeatedStartWithRxSize(I3C_Type *base, *i3c_bus_type_t* type, uint8_t address, *i3c_direction_t* dir, uint8_t rxSize)

Sends a repeated START signal and slave address on the I2C/I3C bus, receive size is also specified in the call.

This function is used to send a Repeated START signal when a transfer is already in progress. Like *I3C_MasterStart()*, it also sends the specified 7-bit address. Call this API also configures the read terminate size for the following read transfer. For example, set the rxSize = 2, the following read transfer will be terminated after two bytes of data received. Write transfer will not be affected by the rxSize configuration.

Note: This function exists primarily to maintain compatible APIs between I3C and I2C drivers, as well as to better document the intent of code that uses these APIs.

Parameters

- base – The I3C peripheral base address.
- type – The bus type to use in this transaction.
- address – 7-bit slave device address, in bits [6:0].
- dir – Master transfer direction, either *kI3C_Read* or *kI3C_Write*. This parameter is used to set the R/w bit (bit 0) in the transmitted slave address.
- rxSize – Read terminate size for the followed read transfer, limit to 255 bytes.

Return values

kStatus_Success – Repeated START signal and address were successfully enqueued in the transmit FIFO.

static inline *status_t* I3C_MasterRepeatedStart(I3C_Type *base, *i3c_bus_type_t* type, uint8_t address, *i3c_direction_t* dir)

Sends a repeated START signal and slave address on the I2C/I3C bus.

This function is used to send a Repeated START signal when a transfer is already in progress. Like *I3C_MasterStart()*, it also sends the specified 7-bit address.

Note: This function exists primarily to maintain compatible APIs between I3C and I2C drivers, as well as to better document the intent of code that uses these APIs.

Parameters

- base – The I3C peripheral base address.
- type – The bus type to use in this transaction.
- address – 7-bit slave device address, in bits [6:0].
- dir – Master transfer direction, either *kI3C_Read* or *kI3C_Write*. This parameter is used to set the R/w bit (bit 0) in the transmitted slave address.

Return values

kStatus_Success – Repeated START signal and address were successfully enqueued in the transmit FIFO.

status_t I3C_MasterSend(I3C_Type *base, const void *txBuff, size_t txSize, uint32_t flags)

Performs a polling send transfer on the I2C/I3C bus.

Sends up to *txSize* number of bytes to the previously addressed slave device. The slave may reply with a NAK to any byte in order to terminate the transfer early. If this happens, this function returns *kStatus_I3C_Nak*.

Parameters

- base – The I3C peripheral base address.

- txBuff – The pointer to the data to be transferred.
- txSize – The length in bytes of the data to be transferred.
- flags – Bit mask of options for the transfer. See enumeration *_i3c_master_transfer_flags* for available options.

Return values

- kStatus_Success – Data was sent successfully.
- kStatus_I3C_Busy – Another master is currently utilizing the bus.
- kStatus_I3C_Timeout – The module has stalled too long in a frame.
- kStatus_I3C_Nak – The slave device sent a NAK in response to an address.
- kStatus_I3C_WriteAbort – The slave device sent a NAK in response to a write.
- kStatus_I3C_MsgError – Message SDR/DDR mismatch or read/write message in wrong state.
- kStatus_I3C_WriteFifoError – Write to M/SWDATAB register when FIFO full.
- kStatus_I3C_InvalidReq – Invalid use of request.

status_t I3C_MasterReceive(I3C_Type *base, void *rxBuff, size_t rxSize, uint32_t flags)

Performs a polling receive transfer on the I2C/I3C bus.

Parameters

- base – The I3C peripheral base address.
- rxBuff – The pointer to the data to be transferred.
- rxSize – The length in bytes of the data to be transferred.
- flags – Bit mask of options for the transfer. See enumeration *_i3c_master_transfer_flags* for available options.

Return values

- kStatus_Success – Data was received successfully.
- kStatus_I3C_Busy – Another master is currently utilizing the bus.
- kStatus_I3C_Timeout – The module has stalled too long in a frame.
- kStatus_I3C_Term – The master terminates slave read.
- kStatus_I3C_HdrParityError – Parity error from DDR read.
- kStatus_I3C_CrcError – CRC error from DDR read.
- kStatus_I3C_MsgError – Message SDR/DDR mismatch or read/write message in wrong state.
- kStatus_I3C_ReadFifoError – Read from M/SRDATAB register when FIFO empty.
- kStatus_I3C_InvalidReq – Invalid use of request.

status_t I3C_MasterStop(I3C_Type *base)

Sends a STOP signal on the I2C/I3C bus.

This function does not return until the STOP signal is seen on the bus, or an error occurs.

Parameters

- base – The I3C peripheral base address.

Return values

- kStatus_Success – The STOP signal was successfully sent on the bus and the transaction terminated.
- kStatus_I3C_Busy – Another master is currently utilizing the bus.
- kStatus_I3C_Timeout – The module has stalled too long in a frame.
- kStatus_I3C_InvalidReq – Invalid use of request.

void I3C_MasterEmitRequest(I3C_Type *base, *i3c_bus_request_t* masterReq)

I3C master emit request.

Parameters

- base – The I3C peripheral base address.
- masterReq – I3C master request of type *i3c_bus_request_t*

static inline void I3C_MasterEmitIBIResponse(I3C_Type *base, *i3c_ibi_response_t* ibiResponse)

I3C master emit request.

Parameters

- base – The I3C peripheral base address.
- ibiResponse – I3C master emit IBI response of type *i3c_ibi_response_t*

void I3C_MasterRegisterIBI(I3C_Type *base, *i3c_register_ibi_addr_t* *ibiRule)

I3C master register IBI rule.

Parameters

- base – The I3C peripheral base address.
- ibiRule – Pointer to ibi rule description of type *i3c_register_ibi_addr_t*

void I3C_MasterGetIBIRules(I3C_Type *base, *i3c_register_ibi_addr_t* *ibiRule)

I3C master get IBI rule.

Parameters

- base – The I3C peripheral base address.
- ibiRule – Pointer to store the read out ibi rule description.

i3c_ibi_type_t I3C_GetIBIType(I3C_Type *base)

I3C master get IBI Type.

Parameters

- base – The I3C peripheral base address.

Return values

i3c_ibi_type_t – Type of *i3c_ibi_type_t*.

static inline uint8_t I3C_GetIBIAddress(I3C_Type *base)

I3C master get IBI Address.

Parameters

- base – The I3C peripheral base address.

Return values

The – 8-bit IBI address.

status_t I3C_MasterProcessDAASpecifiedBaudrate(I3C_Type *base, uint8_t *addressList, uint32_t count, *i3c_master_daa_baudrate_t* *daaBaudRate)

Performs a DAA in the i3c bus with specified temporary baud rate.

Parameters

- base – The I3C peripheral base address.
- addressList – The pointer for address list which is used to do DAA.
- count – The address count in the address list.
- daaBaudRate – The temporary baud rate in DAA process, NULL for using initial setting. The initial setting is set back between the completion of the DAA and the return of this function.

Return values

- kStatus_Success – The transaction was started successfully.
- kStatus_I3C_Busy – Either another master is currently utilizing the bus, or a non-blocking transaction is already in progress.
- kStatus_I3C_SlaveCountExceed – The I3C slave count has exceed the definition in I3C_MAX_DEVCNT.

static inline *status_t* I3C_MasterProcessDAA(I3C_Type *base, uint8_t *addressList, uint32_t count)

Performs a DAA in the i3c bus.

Parameters

- base – The I3C peripheral base address.
- addressList – The pointer for address list which is used to do DAA.
- count – The address count in the address list. The initial setting is set back between the completion of the DAA and the return of this function.

Return values

- kStatus_Success – The transaction was started successfully.
- kStatus_I3C_Busy – Either another master is currently utilizing the bus, or a non-blocking transaction is already in progress.
- kStatus_I3C_SlaveCountExceed – The I3C slave count has exceed the definition in I3C_MAX_DEVCNT.

i3c_device_info_t *I3C_MasterGetDeviceListAfterDAA(I3C_Type *base, uint8_t *count)

Get device information list after DAA process is done.

Parameters

- base – The I3C peripheral base address.
- count – **[out]** The pointer to store the available device count.

Returns

Pointer to the i3c_device_info_t array.

void I3C_MasterClearDeviceCount(I3C_Type *base)

Clear the global device count which represents current devices number on the bus. When user resets all dynamic addresses on the bus, should call this API.

Parameters

- base – The I3C peripheral base address.

status_t I3C_MasterTransferBlocking(I3C_Type *base, *i3c_master_transfer_t* *transfer)

Performs a master polling transfer on the I2C/I3C bus.

Note: The API does not return until the transfer succeeds or fails due to error happens during transfer.

Parameters

- base – The I3C peripheral base address.
- transfer – Pointer to the transfer structure.

Return values

- kStatus_Success – Data was received successfully.
- kStatus_I3C_Busy – Another master is currently utilizing the bus.
- kStatus_I3C_IBIWon – The I3C slave event IBI or MR or HJ won the arbitration on a header address.
- kStatus_I3C_Timeout – The module has stalled too long in a frame.
- kStatus_I3C_Nak – The slave device sent a NAK in response to an address.
- kStatus_I3C_WriteAbort – The slave device sent a NAK in response to a write.
- kStatus_I3C_Term – The master terminates slave read.
- kStatus_I3C_HdrParityError – Parity error from DDR read.
- kStatus_I3C_CrcError – CRC error from DDR read.
- kStatus_I3C_MsgError – Message SDR/DDR mismatch or read/write message in wrong state.
- kStatus_I3C_ReadFifoError – Read from M/SRDATAB register when FIFO empty.
- kStatus_I3C_WriteFifoError – Write to M/SWDATAB register when FIFO full.
- kStatus_I3C_InvalidReq – Invalid use of request.

status_t I3C_SlaveSend(I3C_Type *base, const void *txBuff, size_t txSize)

Performs a polling send transfer on the I3C bus.

Parameters

- base – The I3C peripheral base address.
- txBuff – The pointer to the data to be transferred.
- txSize – The length in bytes of the data to be transferred.

Returns

Error or success status returned by API.

status_t I3C_SlaveReceive(I3C_Type *base, void *rxBuff, size_t rxSize)

Performs a polling receive transfer on the I3C bus.

Parameters

- base – The I3C peripheral base address.
- rxBuff – The pointer to the data to be transferred.
- rxSize – The length in bytes of the data to be transferred.

Returns

Error or success status returned by API.

void I3C_MasterTransferCreateHandle(I3C_Type *base, *i3c_master_handle_t* *handle, const *i3c_master_transfer_callback_t* *callback, void *userData)

Creates a new handle for the I3C master non-blocking APIs.

The creation of a handle is for use with the non-blocking APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the *I3C_MasterTransferAbort()* API shall be called.

Note: The function also enables the NVIC IRQ for the input I3C. Need to notice that on some SoCs the I3C IRQ is connected to INTMUX, in this case user needs to enable the associated INTMUX IRQ in application.

Parameters

- base – The I3C peripheral base address.
- handle – **[out]** Pointer to the I3C master driver handle.
- callback – User provided pointer to the asynchronous callback function.
- userData – User provided pointer to the application callback data.

status_t I3C_MasterTransferNonBlocking(I3C_Type *base, *i3c_master_handle_t* *handle, *i3c_master_transfer_t* *transfer)

Performs a non-blocking transaction on the I2C/I3C bus.

Parameters

- base – The I3C peripheral base address.
- handle – Pointer to the I3C master driver handle.
- transfer – The pointer to the transfer descriptor.

Return values

- kStatus_Success – The transaction was started successfully.
- kStatus_I3C_Busy – Either another master is currently utilizing the bus, or a non-blocking transaction is already in progress.

status_t I3C_MasterTransferGetCount(I3C_Type *base, *i3c_master_handle_t* *handle, size_t *count)

Returns number of bytes transferred so far.

Parameters

- base – The I3C peripheral base address.
- handle – Pointer to the I3C master driver handle.
- count – **[out]** Number of bytes transferred so far by the non-blocking transaction.

Return values

- kStatus_Success –
- kStatus_NoTransferInProgress – There is not a non-blocking transaction currently in progress.

void I3C_MasterTransferAbort(I3C_Type *base, *i3c_master_handle_t* *handle)

Terminates a non-blocking I3C master transmission early.

Note: It is not safe to call this function from an IRQ handler that has a higher priority than the I3C peripheral's IRQ priority.

Parameters

- base – The I3C peripheral base address.
- handle – Pointer to the I3C master driver handle.

void I3C_MasterTransferHandleIRQ(I3C_Type *base, void *intHandle)

Reusable routine to handle master interrupts.

Note: This function does not need to be called unless you are reimplementing the non-blocking API's interrupt handler routines to add special functionality.

Parameters

- base – The I3C peripheral base address.
- intHandle – Pointer to the I3C master driver handle.

void I3C_DriverIRQHandler(uint32_t instance)

Common IRQ handler entry for all I3C instances.

This function dispatches the IRQ to the correct instance handler using the instance index.

Parameters

- instance – I3C instance number.

enum _i3c_master_flags

I3C master peripheral flags.

The following status register flags can be cleared:

- *kI3C_MasterSlaveStartFlag*
- *kI3C_MasterControlDoneFlag*
- *kI3C_MasterCompleteFlag*
- *kI3C_MasterArbitrationWonFlag*
- *kI3C_MasterSlave2MasterFlag*

All flags except *kI3C_MasterBetweenFlag* and *kI3C_MasterNackDetectFlag* can be enabled as interrupts.

Note: These enums are meant to be OR'd together to form a bit mask.

Values:

enumerator kI3C_MasterBetweenFlag

Between messages/DAAs flag

enumerator kI3C_MasterNackDetectFlag

NACK detected flag

enumerator kI3C_MasterSlaveStartFlag
Slave request start flag

enumerator kI3C_MasterControlDoneFlag
Master request complete flag

enumerator kI3C_MasterCompleteFlag
Transfer complete flag

enumerator kI3C_MasterRxReadyFlag
Rx data ready in Rx buffer flag

enumerator kI3C_MasterTxReadyFlag
Tx buffer ready for Tx data flag

enumerator kI3C_MasterArbitrationWonFlag
Header address won arbitration flag

enumerator kI3C_MasterErrorFlag
Error occurred flag

enumerator kI3C_MasterSlave2MasterFlag
Switch from slave to master flag

enumerator kI3C_MasterClearFlags

enum _i3c_master_error_flags
I3C master error flags to indicate the causes.

Note: These enums are meant to be OR'd together to form a bit mask.

Values:

enumerator kI3C_MasterErrorNackFlag
Slave NACKed the last address

enumerator kI3C_MasterErrorWriteAbortFlag
Slave NACKed the write data

enumerator kI3C_MasterErrorParityFlag
Parity error from DDR read

enumerator kI3C_MasterErrorCrcFlag
CRC error from DDR read

enumerator kI3C_MasterErrorReadFlag
Read from MRDATAB register when FIFO empty

enumerator kI3C_MasterErrorWriteFlag
Write to MWDATAB register when FIFO full

enumerator kI3C_MasterErrorMsgFlag
Message SDR/DDR mismatch or read/write message in wrong state

enumerator kI3C_MasterErrorInvalidReqFlag
Invalid use of request

enumerator kI3C_MasterErrorTimeoutFlag
The module has stalled too long in a frame

enumerator kI3C_MasterAllErrorFlags
All error flags

enum _i3c_master_state
I3C working master state.
Values:

enumerator kI3C_MasterStateIdle
Bus stopped.

enumerator kI3C_MasterStateSlvReq
Bus stopped but slave holding SDA low.

enumerator kI3C_MasterStateMsgSdr
In SDR Message mode from using MWMSG_SDR.

enumerator kI3C_MasterStateNormAct
In normal active SDR mode.

enumerator kI3C_MasterStateDdr
In DDR Message mode.

enumerator kI3C_MasterStateDaa
In ENTDAA mode.

enumerator kI3C_MasterStateIbiAck
Waiting on IBI ACK/NACK decision.

enumerator kI3C_MasterStateIbiRcv
Receiving IBI.

enum _i3c_master_enable
I3C master enable configuration.
Values:

enumerator kI3C_MasterOff
Master off.

enumerator kI3C_MasterOn
Master on.

enumerator kI3C_MasterCapable
Master capable.

enum _i3c_master_hkeep
I3C high keeper configuration.
Values:

enumerator kI3C_MasterHighKeeperNone
Use PUR to hold SCL high.

enumerator kI3C_MasterHighKeeperWiredIn
Use pin_HK controls.

enumerator kI3C_MasterPassiveSDA
Hi-Z for Bus Free and hold SDA.

enumerator kI3C_MasterPassiveSDASCL
Hi-Z both for Bus Free, and can Hi-Z SDA for hold.

enum _i3c_bus_request
Emits the requested operation when doing in pieces vs. by message.
Values:

enumerator kI3C_RequestNone
No request.

enumerator kI3C_RequestEmitStartAddr
Request to emit start and address on bus.

enumerator kI3C_RequestEmitStop
Request to emit stop on bus.

enumerator kI3C_RequestIbiAckNack
Manual IBI ACK or NACK.

enumerator kI3C_RequestProcessDAA
Process DAA.

enumerator kI3C_RequestForceExit
Request to force exit.

enumerator kI3C_RequestAutoIbi
Hold in stopped state, but Auto-emit START,7E.

enum _i3c_bus_type
Bus type with EmitStartAddr.

Values:

enumerator kI3C_TypeI3CSdr
SDR mode of I3C.

enumerator kI3C_TypeI2C
Standard i2c protocol.

enumerator kI3C_TypeI3CDdr
HDR-DDR mode of I3C.

enum _i3c_ibi_response
IBI response.

Values:

enumerator kI3C_IbiRespAck
ACK with no mandatory byte.

enumerator kI3C_IbiRespNack
NACK.

enumerator kI3C_IbiRespAckMandatory
ACK with mandatory byte.

enumerator kI3C_IbiRespManual
Reserved.

enum _i3c_ibi_type
IBI type.

Values:

enumerator kI3C_IbiNormal
In-band interrupt.

enumerator kI3C_IbiHotJoin
slave hot join.

enumerator kI3C_IbiMasterRequest
slave master ship request.

enum _i3c_ibi_state
IBI state.

Values:

enumerator kI3C_IbiReady
In-band interrupt ready state, ready for user to handle.

enumerator kI3C_IbiDataBuffNeed
In-band interrupt need data buffer for data receive.

enumerator kI3C_IbiAckNackPending
In-band interrupt Ack/Nack pending for decision.

enum _i3c_direction
Direction of master and slave transfers.
Values:

enumerator kI3C_Write
Master transmit.

enumerator kI3C_Read
Master receive.

enum _i3c_tx_trigger_level
Watermark of TX int/dma trigger level.
Values:

enumerator kI3C_TxTriggerOnEmpty
Trigger on empty.

enumerator kI3C_TxTriggerUntilOneQuarterOrLess
Trigger on 1/4 full or less.

enumerator kI3C_TxTriggerUntilOneHalfOrLess
Trigger on 1/2 full or less.

enumerator kI3C_TxTriggerUntilOneLessThanFull
Trigger on 1 less than full or less.

enum _i3c_rx_trigger_level
Watermark of RX int/dma trigger level.
Values:

enumerator kI3C_RxTriggerOnNotEmpty
Trigger on not empty.

enumerator kI3C_RxTriggerUntilOneQuarterOrMore
Trigger on 1/4 full or more.

enumerator kI3C_RxTriggerUntilOneHalfOrMore
Trigger on 1/2 full or more.

enumerator kI3C_RxTriggerUntilThreeQuarterOrMore
Trigger on 3/4 full or more.

enum _i3c_rx_term_ops
I3C master read termination operations.
Values:

enumerator kI3C_RxTermDisable
Master doesn't terminate read, used for CCC transfer.

enumerator kI3C_RxAutoTerm
Master auto terminate read after receiving specified bytes(<=255).

enumerator kI3C_RxTermLastByte
Master terminates read at any time after START, no length limitation.

enum _i3c_start_scl_delay
I3C start SCL delay options.
Values:

enumerator kI3C_NoDelay
No delay.

enumerator kI3C_IncreaseSclHalfPeriod
Increases SCL clock period by 1/2.

enumerator kI3C_IncreaseSclOnePeriod
Increases SCL clock period by 1.

enumerator kI3C_IncreaseSclOneAndHalfPeriod
Increases SCL clock period by 1 1/2

enum _i3c_master_bus_mode
I3C bus topology. Selects how the baud rate is calculated for the bus.
Values:

enumerator kI3C_PureBus
Only I3C devices on the bus.

enumerator kI3C_MixedFastBus
I3C and legacy I2C Fm/Fm+ devices share the bus.

enumerator kI3C_LimitedBus
I3C controller on a legacy I2C bus.

enum _i3c_legacy_i2c_speed_mode
Legacy I2C preset selector.
Values:

enumerator kI3C_LegacyI2CStandardMode
Reserved for future Standard-mode support.

enumerator kI3C_LegacyI2CFastMode
Legacy I2C Fast-mode preset.

enumerator kI3C_LegacyI2CFastModePlus
Legacy I2C Fast-mode Plus preset.

enum _i3c_master_src_clk
Validated source clock selector for I2C timing presets.

Values:

enumerator kI3C_MasterSrcClk24M
Use the validated 24 MHz timing table.

enumerator kI3C_MasterSrcClk25M
Use the validated 25 MHz timing table.

enum _i3c_master_transfer_flags
Transfer option flags.

Note: These enumerations are intended to be OR'd together to form a bit mask of options for the *_i3c_master_transfer::flags* field.

Values:

enumerator kI3C_TransferDefaultFlag
Transfer starts with a start signal, stops with a stop signal.

enumerator kI3C_TransferNoStartFlag
Don't send a start condition, address, and sub address

enumerator kI3C_TransferRepeatedStartFlag
Send a repeated start condition

enumerator kI3C_TransferNoStopFlag
Don't send a stop condition.

enumerator kI3C_TransferWordsFlag
Transfer in words, else transfer in bytes.

enumerator kI3C_TransferDisableRxTermFlag
Disable Rx termination. Note: It's for I3C CCC transfer.

enumerator kI3C_TransferRxAutoTermFlag
Set Rx auto-termination. Note: It's adaptive based on Rx size(<=255 bytes) except in I3C_MasterReceive.

enumerator kI3C_TransferStartWithBroadcastAddr
Start transfer with 0x7E, then read/write data with device address.

typedef enum *_i3c_master_state* i3c_master_state_t
I3C working master state.

typedef enum *_i3c_master_enable* i3c_master_enable_t
I3C master enable configuration.

typedef enum *_i3c_master_hkeep* i3c_master_hkeep_t
I3C high keeper configuration.

typedef enum *_i3c_bus_request* i3c_bus_request_t
Emits the requested operation when doing in pieces vs. by message.

typedef enum *_i3c_bus_type* i3c_bus_type_t
Bus type with EmitStartAddr.

typedef enum *_i3c_ibi_response* i3c_ibi_response_t
IBI response.

typedef enum *_i3c_ibi_type* i3c_ibi_type_t
IBI type.

typedef enum *_i3c_ibi_state* i3c_ibi_state_t
IBI state.

typedef enum *_i3c_direction* i3c_direction_t
Direction of master and slave transfers.

typedef enum *_i3c_tx_trigger_level* i3c_tx_trigger_level_t
Watermark of TX int/dma trigger level.

typedef enum *_i3c_rx_trigger_level* i3c_rx_trigger_level_t
Watermark of RX int/dma trigger level.

typedef enum *_i3c_rx_term_ops* i3c_rx_term_ops_t
I3C master read termination operations.

typedef enum *_i3c_start_scl_delay* i3c_start_scl_delay_t
I3C start SCL delay options.

typedef struct *_i3c_register_ibi_addr* i3c_register_ibi_addr_t
Structure with setting master IBI rules and slave registry.

typedef struct *_i3c_baudrate* i3c_baudrate_hz_t
Structure with I3C baudrate settings.

typedef struct *_i3c_master_daa_baudrate* i3c_master_daa_baudrate_t

I3C DAA baud rate configuration.

typedef enum *_i3c_master_bus_mode* i3c_master_bus_mode_t

I3C bus topology. Selects how the baud rate is calculated for the bus.

typedef enum *_i3c_legacy_i2c_speed_mode* i3c_legacy_i2c_speed_mode_t

Legacy I2C preset selector.

typedef enum *_i3c_master_src_clk* i3c_master_src_clk_t

Validated source clock selector for I2C timing presets.

typedef struct *_i3c_master_baudrate_config* i3c_master_baudrate_config_t

I3C master baud rate configuration.

typedef struct *_i3c_timing* i3c_timing_t

Pre-calculated I3C MCONFIG timing fields.

Pre-compute the timing at init, then switch with *I3C_MasterSetTiming()*. The actualXxxBaud fields report the rates the hardware achieves.

typedef struct *_i3c_master_config* i3c_master_config_t

Structure with settings to initialize the I3C master module.

This structure holds configuration settings for the I3C peripheral. To initialize this structure to reasonable defaults, call the *I3C_MasterGetDefaultConfig()* function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

typedef struct *_i3c_master_transfer* i3c_master_transfer_t

typedef struct *_i3c_master_handle* i3c_master_handle_t

typedef struct *_i3c_master_transfer_callback* i3c_master_transfer_callback_t

i3c master callback functions.

typedef void (*i3c_master_isr_t)(I3C_Type *base, void *handle)

Typedef for master interrupt handler.

struct __i3c_register_ibi_addr

#include <fsl_i3c.h> Structure with setting master IBI rules and slave registry.

Public Members

uint8_t address[5]

Address array for registry.

bool i3cFastStart

Allow the START header to run as push-pull speed if all dynamic addresses take MSB 0.

bool ibiHasPayload

Whether the address array has mandatory IBI byte.

struct __i3c__baudrate

#include <fsl_i3c.h> Structure with I3C baudrate settings.

Public Members

uint32_t i2cBaud

Desired I2C baud rate in Hertz.

uint32_t i3cPushPullBaud

Desired I3C push-pull baud rate in Hertz.

uint32_t i3cOpenDrainBaud

Desired I3C open-drain baud rate in Hertz.

struct __i3c__master__daa__baudrate

#include <fsl_i3c.h> I3C DAA baud rate configuration.

Public Members

uint32_t sourceClock__Hz

FCLK, function clock in Hertz.

uint32_t i3cPushPullBaud

Desired I3C push-pull baud rate in Hertz.

uint32_t i3cOpenDrainBaud

Desired I3C open-drain baud rate in Hertz.

struct __i3c__master__baudrate__config

#include <fsl_i3c.h> I3C master baud rate configuration.

Public Members

i3c_master_bus_mode_t busMode

Bus topology.

uint32_t i2cBaud

Desired legacy I2C baud rate in Hz; 0 = do not program I2CBAUD.

uint32_t i3cPushPullBaud

Desired I3C push-pull SCL frequency in Hz.

uint32_t i3cOpenDrainBaud

Desired I3C open-drain SCL frequency in Hz.

bool enableODHPP

Open-drain HIGH shape: true = narrow (normal I3C frames), false = symmetric (required for the first-7Eh broadcast).

struct __i3c__timing

#include <fsl_i3c.h> Pre-calculated I3C MCONFIG timing fields.

Pre-compute the timing at init, then switch with *I3C_MasterSetTiming()*. The actualXxxBaud fields report the rates the hardware achieves.

Public Members

uint8_t ppBaud

MCONFIG[PPBAUD] field.

uint8_t ppLow

MCONFIG[PPLOW] field.

uint8_t odBaud

MCONFIG[ODBAUD] field (0 is forbidden).

bool odHighPP

MCONFIG[ODHPP] field.

uint8_t i2cBaud

MCONFIG[I2CBAUD] field.

uint32_t actualPpBaud

Achieved PP SCL frequency in Hz.

uint32_t actualOdBaud

Achieved OD SCL frequency in Hz.

uint32_t actualI2cBaud

Achieved I2C SCL frequency in Hz (0 if no I2C is configured).

struct __i3c__master__config

#include <fsl_i3c.h> Structure with settings to initialize the I3C master module.

This structure holds configuration settings for the I3C peripheral. To initialize this structure to reasonable defaults, call the *I3C_MasterGetDefaultConfig()* function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

Public Members

i3c_master_enable_t enableMaster
Enable master mode.

bool disableTimeout
Whether to disable timeout to prevent the ERRWARN.

i3c_master_hkeep_t hKeep
High keeper mode setting.

bool enableOpenDrainStop
Whether to emit open-drain speed STOP.

bool enableOpenDrainHigh
Enable Open-Drain High to be 1 PPBAUD count for i3c messages, or 1 ODBAUD.

i3c_baudrate_hz_t baudRate_Hz
Desired baud rate settings.

i3c_start_scl_delay_t startSclDelay
I3C SCL delay after START.

i3c_start_scl_delay_t restartSclDelay
I3C SCL delay after Repeated START.

struct _i3c_master_transfer_callback
#include <fsl_i3c.h> i3c master callback functions.

Public Members

void (*slave2Master)(I3C_Type *base, void *userData)
Transfer complete callback

void (*ibiCallback)(I3C_Type *base, *i3c_master_handle_t* *handle, *i3c_ibi_type_t* ibiType, *i3c_ibi_state_t* ibiState)
IBI event callback

void (*transferComplete)(I3C_Type *base, *i3c_master_handle_t* *handle, *status_t* completionStatus, void *userData)
Transfer complete callback

struct _i3c_master_transfer
#include <fsl_i3c.h> Non-blocking transfer descriptor structure.

This structure is used to pass transaction parameters to the *I3C_MasterTransferNonBlocking()* API.

Public Members

uint32_t flags

Bit mask of options for the transfer. See enumeration *_i3c_master_transfer_flags* for available options. Set to 0 or *kI3C_TransferDefaultFlag* for normal transfers.

uint8_t slaveAddress

The 7-bit slave address.

i3c_direction_t direction

Either *kI3C_Read* or *kI3C_Write*.

uint32_t subaddress

Sub address. Transferred MSB first.

size_t subaddressSize

Length of sub address to send in bytes. Maximum size is 4 bytes.

void *data

Pointer to data to transfer.

size_t dataSize

Number of bytes to transfer.

i3c_bus_type_t busType

bus type.

i3c_ibi_response_t ibiResponse

ibi response during transfer.

struct __i3c__master__handle

#include <fsl_i3c.h> Driver handle for master non-blocking APIs.

Note: The contents of this structure are private and subject to change.

Public Members

uint8_t state

Transfer state machine current state.

uint32_t remainingBytes

Remaining byte count in current state.

i3c_rx_term_ops_t rxTermOps

Read termination operation.

i3c_master_transfer_t transfer
Copy of the current transfer info.

uint8_t ibiAddress
Slave address which request IBI.

uint8_t *ibiBuff
Pointer to IBI buffer to keep ibi bytes.

size_t ibiPayloadSize
IBI payload size.

i3c_ibi_type_t ibiType
IBI type.

i3c_master_transfer_callback_t callback
Callback functions pointer.

void *userData
Application data passed to callback.

2.33 I3C Slave Driver

void I3C_SlaveGetDefaultConfig(*i3c_slave_config_t* *slaveConfig)

Provides a default configuration for the I3C slave peripheral.

This function provides the following default configuration for the I3C slave peripheral:

```
slaveConfig->enableslave          = true;
```

After calling this function, you can override any settings in order to customize the configuration, prior to initializing the slave driver with I3C_SlaveInit().

Parameters

- slaveConfig – **[out]** User provided configuration structure for default values. Refer to *i3c_slave_config_t*.

void I3C_SlaveInit(I3C_Type *base, const *i3c_slave_config_t* *slaveConfig, uint32_t slowClock_Hz)

Initializes the I3C slave peripheral.

This function enables the peripheral clock and initializes the I3C slave peripheral as described by the user provided configuration.

Parameters

- base – The I3C peripheral base address.
- slaveConfig – User provided peripheral configuration. Use I3C_SlaveGetDefaultConfig() to get a set of defaults that you can override.
- slowClock_Hz – Frequency in Hertz of the I3C slow clock. Used to calculate the bus match condition values. If FSL_FEATURE_I3C_HAS_NO_SCONFIG_BAMATCH defines as 1, this parameter is useless.

void I3C_SlaveDeinit(I3C_Type *base)

Deinitializes the I3C slave peripheral.

This function disables the I3C slave peripheral and gates the clock.

Parameters

- base – The I3C peripheral base address.

static inline void I3C_SlaveEnable(I3C_Type *base, bool isEnable)

Enable/Disable Slave.

Parameters

- base – The I3C peripheral base address.
- isEnable – Enable or disable.

static inline uint32_t I3C_SlaveGetStatusFlags(I3C_Type *base)

Gets the I3C slave status flags.

A bit mask with the state of all I3C slave status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

See also:

_i3c_slave_flags

Parameters

- base – The I3C peripheral base address.

Returns

State of the status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

static inline void I3C_SlaveClearStatusFlags(I3C_Type *base, uint32_t statusMask)

Clears the I3C slave status flag state.

The following status register flags can be cleared:

- *kI3C_SlaveBusStartFlag*
- *kI3C_SlaveMatchedFlag*
- *kI3C_SlaveBusStopFlag*

Attempts to clear other flags has no effect.

See also:

_i3c_slave_flags.

Parameters

- base – The I3C peripheral base address.
- statusMask – A bitmask of status flags that are to be cleared. The mask is composed of *_i3c_slave_flags* enumerators OR'd together. You may pass the result of a previous call to I3C_SlaveGetStatusFlags().

static inline uint32_t I3C_SlaveGetErrorStatusFlags(I3C_Type *base)

Gets the I3C slave error status flags.

A bit mask with the state of all I3C slave error status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

See also:

_i3c_slave_error_flags

Parameters

- base – The I3C peripheral base address.

Returns

State of the error status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

static inline void I3C_SlaveClearErrorStatusFlags(I3C_Type *base, uint32_t statusMask)

Clears the I3C slave error status flag state.

See also:

_i3c_slave_error_flags.

Parameters

- base – The I3C peripheral base address.
- statusMask – A bitmask of error status flags that are to be cleared. The mask is composed of *_i3c_slave_error_flags* enumerators OR'd together. You may pass the result of a previous call to I3C_SlaveGetErrorStatusFlags().

i3c_slave_activity_state_t I3C_SlaveGetActivityState(I3C_Type *base)

Gets the I3C slave state.

Parameters

- base – The I3C peripheral base address.

Returns

I3C slave activity state, refer *i3c_slave_activity_state_t*.

static inline void I3C_SlaveEnableInterrupts(I3C_Type *base, uint32_t interruptMask)

Enables the I3C slave interrupt requests.

Only below flags can be enabled as interrupts.

- *kI3C_SlaveBusStartFlag*
- *kI3C_SlaveMatchedFlag*
- *kI3C_SlaveBusStopFlag*
- *kI3C_SlaveRxReadyFlag*
- *kI3C_SlaveTxReadyFlag*
- *kI3C_SlaveDynamicAddrChangedFlag*
- *kI3C_SlaveReceivedCCCFlag*
- *kI3C_SlaveErrorFlag*

- *kI3C_SlaveHDRCommandMatchFlag*
- *kI3C_SlaveCCCHandledFlag*
- *kI3C_SlaveEventSentFlag*

Parameters

- base – The I3C peripheral base address.
- interruptMask – Bit mask of interrupts to enable. See *_i3c_slave_flags* for the set of constants that should be OR'd together to form the bit mask.

static inline void I3C_SlaveDisableInterrupts(I3C_Type *base, uint32_t interruptMask)

Disables the I3C slave interrupt requests.

Only below flags can be disabled as interrupts.

- *kI3C_SlaveBusStartFlag*
- *kI3C_SlaveMatchedFlag*
- *kI3C_SlaveBusStopFlag*
- *kI3C_SlaveRxReadyFlag*
- *kI3C_SlaveTxReadyFlag*
- *kI3C_SlaveDynamicAddrChangedFlag*
- *kI3C_SlaveReceivedCCCFlag*
- *kI3C_SlaveErrorFlag*
- *kI3C_SlaveHDRCommandMatchFlag*
- *kI3C_SlaveCCCHandledFlag*
- *kI3C_SlaveEventSentFlag*

Parameters

- base – The I3C peripheral base address.
- interruptMask – Bit mask of interrupts to disable. See *_i3c_slave_flags* for the set of constants that should be OR'd together to form the bit mask.

static inline uint32_t I3C_SlaveGetEnabledInterrupts(I3C_Type *base)

Returns the set of currently enabled I3C slave interrupt requests.

Parameters

- base – The I3C peripheral base address.

Returns

A bitmask composed of *_i3c_slave_flags* enumerators OR'd together to indicate the set of enabled interrupts.

static inline uint32_t I3C_SlaveGetPendingInterrupts(I3C_Type *base)

Returns the set of pending I3C slave interrupt requests.

Parameters

- base – The I3C peripheral base address.

Returns

A bitmask composed of *_i3c_slave_flags* enumerators OR'd together to indicate the set of pending interrupts.

static inline void I3C_SlaveEnableDMA(I3C_Type *base, bool enableTx, bool enableRx, uint32_t width)

Enables or disables I3C slave DMA requests.

Parameters

- base – The I3C peripheral base address.
- enableTx – Enable flag for transmit DMA request. Pass true for enable, false for disable.
- enableRx – Enable flag for receive DMA request. Pass true for enable, false for disable.
- width – DMA read/write unit in bytes.

static inline uint32_t I3C_SlaveGetTxFifoAddress(I3C_Type *base, uint32_t width)

Gets I3C slave transmit data register address for DMA transfer.

Parameters

- base – The I3C peripheral base address.
- width – DMA read/write unit in bytes.

Returns

The I3C Slave Transmit Data Register address.

static inline uint32_t I3C_SlaveGetRxFifoAddress(I3C_Type *base, uint32_t width)

Gets I3C slave receive data register address for DMA transfer.

Parameters

- base – The I3C peripheral base address.
- width – DMA read/write unit in bytes.

Returns

The I3C Slave Receive Data Register address.

static inline void I3C_SlaveSetWatermarks(I3C_Type *base, *i3c_tx_trigger_level_t* txLvl, *i3c_rx_trigger_level_t* rxLvl, bool flushTx, bool flushRx)

Sets the watermarks for I3C slave FIFOs.

Parameters

- base – The I3C peripheral base address.
- txLvl – Transmit FIFO watermark level. The *kI3C_SlaveTxReadyFlag* flag is set whenever the number of words in the transmit FIFO reaches *txLvl*.
- rxLvl – Receive FIFO watermark level. The *kI3C_SlaveRxReadyFlag* flag is set whenever the number of words in the receive FIFO reaches *rxLvl*.
- flushTx – true if TX FIFO is to be cleared, otherwise TX FIFO remains unchanged.
- flushRx – true if RX FIFO is to be cleared, otherwise RX FIFO remains unchanged.

static inline void I3C_SlaveGetFifoCounts(I3C_Type *base, size_t *rxCount, size_t *txCount)

Gets the current number of bytes in the I3C slave FIFOs.

Parameters

- base – The I3C peripheral base address.
- txCount – **[out]** Pointer through which the current number of bytes in the transmit FIFO is returned. Pass NULL if this value is not required.

- rxCount – **[out]** Pointer through which the current number of bytes in the receive FIFO is returned. Pass NULL if this value is not required.

status_t I3C_SlaveSend(I3C_Type *base, const void *txBuff, size_t txSize)

Performs a polling send transfer on the I3C bus.

Parameters

- base – The I3C peripheral base address.
- txBuff – The pointer to the data to be transferred.
- txSize – The length in bytes of the data to be transferred.

Returns

Error or success status returned by API.

status_t I3C_SlaveReceive(I3C_Type *base, void *rxBuff, size_t rxSize)

Performs a polling receive transfer on the I3C bus.

Parameters

- base – The I3C peripheral base address.
- rxBuff – The pointer to the data to be transferred.
- rxSize – The length in bytes of the data to be transferred.

Returns

Error or success status returned by API.

void I3C_SlaveTransferCreateHandle(I3C_Type *base, *i3c_slave_handle_t* *handle, *i3c_slave_transfer_callback_t* callback, void *userData)

Creates a new handle for the I3C slave non-blocking APIs.

The creation of a handle is for use with the non-blocking APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the *I3C_SlaveTransferAbort()* API shall be called.

Note: The function also enables the NVIC IRQ for the input I3C. Need to notice that on some SoCs the I3C IRQ is connected to INTMUX, in this case user needs to enable the associated INTMUX IRQ in application.

Parameters

- base – The I3C peripheral base address.
- handle – **[out]** Pointer to the I3C slave driver handle.
- callback – User provided pointer to the asynchronous callback function.
- userData – User provided pointer to the application callback data.

status_t I3C_SlaveTransferNonBlocking(I3C_Type *base, *i3c_slave_handle_t* *handle, uint32_t eventMask)

Starts accepting slave transfers.

Call this API after calling I2C_SlaveInit() and *I3C_SlaveTransferCreateHandle()* to start processing transactions driven by an I2C master. The slave monitors the I2C bus and pass events to the callback that was passed into the call to *I3C_SlaveTransferCreateHandle()*. The callback is always invoked from the interrupt context.

The set of events received by the callback is customizable. To do so, set the *eventMask* parameter to the OR'd combination of *i3c_slave_transfer_event_t* enumerators for the events you wish to receive. The *kI3C_SlaveTransmitEvent* and *kI3C_SlaveReceiveEvent* events are

always enabled and do not need to be included in the mask. Alternatively, you can pass 0 to get a default set of only the transmit and receive events that are always enabled. In addition, the *kI3C_SlaveAllEvents* constant is provided as a convenient way to enable all events.

Parameters

- base – The I3C peripheral base address.
- handle – Pointer to struct: *_i3c_slave_handle* structure which stores the transfer state.
- eventMask – Bit mask formed by OR'ing together *i3c_slave_transfer_event_t* enumerators to specify which events to send to the callback. Other accepted values are 0 to get a default set of only the transmit and receive events, and *kI3C_SlaveAllEvents* to enable all events.

Return values

- kStatus_Success – Slave transfers were successfully started.
- kStatus_I3C_Busy – Slave transfers have already been started on this handle.

status_t I3C_SlaveTransferGetCount(I3C_Type *base, *i3c_slave_handle_t* *handle, size_t *count)

Gets the slave transfer status during a non-blocking transfer.

Parameters

- base – The I3C peripheral base address.
- handle – Pointer to i2c_slave_handle_t structure.
- count – **[out]** Pointer to a value to hold the number of bytes transferred. May be NULL if the count is not required.

Return values

- kStatus_Success –
- kStatus_NoTransferInProgress –

void I3C_SlaveTransferAbort(I3C_Type *base, *i3c_slave_handle_t* *handle)

Aborts the slave non-blocking transfers.

Note: This API could be called at any time to stop slave for handling the bus events.

Parameters

- base – The I3C peripheral base address.
- handle – Pointer to struct: *_i3c_slave_handle* structure which stores the transfer state.

void I3C_SlaveTransferHandleIRQ(I3C_Type *base, void *intHandle)

Reusable routine to handle slave interrupts.

Note: This function does not need to be called unless you are reimplementing the non blocking API's interrupt handler routines to add special functionality.

Parameters

- base – The I3C peripheral base address.

- intHandle – Pointer to struct: *_i3c_slave_handle* structure which stores the transfer state.

enum _i3c_slave_flags

I3C slave peripheral flags.

The following status register flags can be cleared:

- *kI3C_SlaveBusStartFlag*
- *kI3C_SlaveMatchedFlag*
- *kI3C_SlaveBusStopFlag*

Only below flags can be enabled as interrupts.

- *kI3C_SlaveBusStartFlag*
- *kI3C_SlaveMatchedFlag*
- *kI3C_SlaveBusStopFlag*
- *kI3C_SlaveRxReadyFlag*
- *kI3C_SlaveTxReadyFlag*
- *kI3C_SlaveDynamicAddrChangedFlag*
- *kI3C_SlaveReceivedCCCFlag*
- *kI3C_SlaveErrorFlag*
- *kI3C_SlaveHDRCommandMatchFlag*
- *kI3C_SlaveCCCHandledFlag*
- *kI3C_SlaveEventSentFlag*

Note: These enums are meant to be OR'd together to form a bit mask.

Values:

enumerator kI3C_SlaveNotStopFlag

Slave status not stop flag

enumerator kI3C_SlaveMessageFlag

Slave status message, indicating slave is listening to the bus traffic or responding

enumerator kI3C_SlaveRequiredReadFlag

Slave status required, either is master doing SDR read from slave, or is IBI pushing out.

enumerator kI3C_SlaveRequiredWriteFlag

Slave status request write, master is doing SDR write to slave, except slave in ENTDAA mode

enumerator kI3C_SlaveBusDAAFlag

I3C bus is in ENTDAA mode

enumerator kI3C_SlaveBusHDRModeFlag

I3C bus is in HDR mode

enumerator kI3C_SlaveBusStartFlag
Start/Re-start event is seen since the bus was last cleared

enumerator kI3C_SlaveMatchedFlag
Slave address(dynamic/static) matched since last cleared

enumerator kI3C_SlaveBusStopFlag
Stop event is seen since the bus was last cleared

enumerator kI3C_SlaveRxReadyFlag
Rx data ready in rx buffer flag

enumerator kI3C_SlaveTxReadyFlag
Tx buffer ready for Tx data flag

enumerator kI3C_SlaveDynamicAddrChangedFlag
Slave dynamic address has been assigned, re-assigned, or lost

enumerator kI3C_SlaveReceivedCCCFlag
Slave received Common command code

enumerator kI3C_SlaveErrorFlag
Error occurred flag

enumerator kI3C_SlaveHDRCommandMatchFlag
High data rate command match

enumerator kI3C_SlaveCCCHandledFlag
Slave received Common command code is handled by I3C module

enumerator kI3C_SlaveEventSentFlag
Slave IBI/P2P/MR/HJ event has been sent

enumerator kI3C_SlaveIbiDisableFlag
Slave in band interrupt is disabled.

enumerator kI3C_SlaveMasterRequestDisabledFlag
Slave master request is disabled.

enumerator kI3C_SlaveHotJoinDisabledFlag
Slave Hot-Join is disabled.

enumerator kI3C_SlaveClearFlags
All flags which are cleared by the driver upon starting a transfer.

enumerator kI3C_SlaveAllIrqFlags

enum _i3c_slave_error_flags

I3C slave error flags to indicate the causes.

Note: These enums are meant to be OR'd together to form a bit mask.

Values:

enumerator kI3C_SlaveErrorOverrunFlag

Slave internal from-bus buffer/FIFO overrun.

enumerator kI3C_SlaveErrorUnderrunFlag

Slave internal to-bus buffer/FIFO underrun

enumerator kI3C_SlaveErrorUnderrunNakFlag

Slave internal from-bus buffer/FIFO underrun and NACK error

enumerator kI3C_SlaveErrorTermFlag

Terminate error from master

enumerator kI3C_SlaveErrorInvalidStartFlag

Slave invalid start flag

enumerator kI3C_SlaveErrorSdrParityFlag

SDR parity error

enumerator kI3C_SlaveErrorHdrParityFlag

HDR parity error

enumerator kI3C_SlaveErrorHdrCRCFlag

HDR-DDR CRC error

enumerator kI3C_SlaveErrorS0S1Flag

S0 or S1 error

enumerator kI3C_SlaveErrorOverreadFlag

Over-read error

enumerator kI3C_SlaveErrorOverwriteFlag

Over-write error

enum _i3c_slave_event

I3C slave.event.

Values:

enumerator kI3C_SlaveEventNormal

Normal mode.

enumerator kI3C_SlaveEventIBI
In band interrupt event.

enumerator kI3C_SlaveEventMasterReq
Master request event.

enumerator kI3C_SlaveEventHotJoinReq
Hot-join event.

enum _i3c_slave_activity_state
I3C slave.activity state.

Values:

enumerator kI3C_SlaveNoLatency
Normal bus operation

enumerator kI3C_SlaveLatency1Ms
1ms of latency.

enumerator kI3C_SlaveLatency100Ms
100ms of latency.

enumerator kI3C_SlaveLatency10S
10s latency.

enum _i3c_slave_transfer_event
Set of events sent to the callback for non blocking slave transfers.

These event enumerations are used for two related purposes. First, a bit mask created by OR'ing together events is passed to *I3C_SlaveTransferNonBlocking()* in order to specify which events to enable. Then, when the slave callback is invoked, it is passed the current event through its *transfer* parameter.

Note: These enumerations are meant to be OR'd together to form a bit mask of events.

Values:

enumerator kI3C_SlaveAddressMatchEvent
Received the slave address after a start or repeated start.

enumerator kI3C_SlaveTransmitEvent
Callback is requested to provide data to transmit (slave-transmitter role).

enumerator kI3C_SlaveReceiveEvent
Callback is requested to provide a buffer in which to place received data (slave-receiver role).

enumerator kI3C_SlaveRequiredTransmitEvent
Callback is requested to provide a buffer in which to place received data (slave-receiver role).

enumerator kI3C_SlaveStartEvent
A start/repeated start was detected.

enumerator kI3C_SlaveHDRCommandMatchEvent
Slave Match HDR Command.

enumerator kI3C_SlaveCompletionEvent
A stop was detected, completing the transfer.

enumerator kI3C_SlaveRequestSentEvent
Slave request event sent.

enumerator kI3C_SlaveReceivedCCCEvent
Slave received CCC event, need to handle by application.

enumerator kI3C_SlaveAllEvents
Bit mask of all available events.

typedef enum *_i3c_slave_event* i3c_slave_event_t
I3C slave.event.

typedef enum *_i3c_slave_activity_state* i3c_slave_activity_state_t
I3C slave.activity state.

typedef struct *_i3c_slave_config* i3c_slave_config_t
Structure with settings to initialize the I3C slave module.

This structure holds configuration settings for the I3C peripheral. To initialize this structure to reasonable defaults, call the I3C_SlaveGetDefaultConfig() function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

typedef enum *_i3c_slave_transfer_event* i3c_slave_transfer_event_t
Set of events sent to the callback for non blocking slave transfers.

These event enumerations are used for two related purposes. First, a bit mask created by OR'ing together events is passed to *I3C_SlaveTransferNonBlocking()* in order to specify which events to enable. Then, when the slave callback is invoked, it is passed the current event through its *transfer* parameter.

Note: These enumerations are meant to be OR'd together to form a bit mask of events.

typedef struct *_i3c_slave_transfer* i3c_slave_transfer_t
I3C slave transfer structure.

typedef struct *_i3c_slave_handle* i3c_slave_handle_t

typedef void (*i3c_slave_transfer_callback_t)(I3C_Type *base, *i3c_slave_transfer_t* *transfer, void *userData)

Slave event callback function pointer type.

This callback is used only for the slave non-blocking transfer API. To install a callback, use the I3C_SlaveSetCallback() function after you have created a handle.

Param base
Base address for the I3C instance on which the event occurred.

Param transfer
Pointer to transfer descriptor containing values passed to and/or from the callback.

Param userData
Arbitrary pointer-sized value passed from the application.

typedef void (*i3c_slave_isr_t)(I3C_Type *base, void *handle)

Typedef for slave interrupt handler.

struct i3c_slave_config

#include <fsl_i3c.h> Structure with settings to initialize the I3C slave module.

This structure holds configuration settings for the I3C peripheral. To initialize this structure to reasonable defaults, call the I3C_SlaveGetDefaultConfig() function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

Public Members

bool enableSlave

Whether to enable slave.

uint8_t staticAddr

Static address.

uint16_t vendorID

Device vendor ID(manufacture ID).

uint32_t partNumber

Device part number info

uint8_t dcr

Device characteristics register information.

uint8_t bcr

Bus characteristics register information.

uint8_t hdrMode

Support hdr mode, could be OR logic in enumeration:i3c_hdr_mode_t.

bool nakAllRequest

Whether to reply NAK to all requests except broadcast CCC.

bool ignoreS0S1Error

Whether to ignore S0/S1 error in SDR mode.

bool offline

Whether to wait 60 us of bus quiet or HDR request to ensure slave track SDR mode safely.

bool matchSlaveStartStop

Whether to assert start/stop status only the time slave is addressed.

uint32_t maxWriteLength

Maximum write length.

uint32_t maxReadLength

Maximum read length.

struct __i3c_slave_transfer

#include <fsl_i3c.h> I3C slave transfer structure.

Public Members

uint32_t event

Reason the callback is being invoked.

uint8_t *txData

Transfer buffer

size_t txDataSize

Transfer size

uint8_t *rxData

Transfer buffer

size_t rxDataSize

Transfer size

status_t completionStatus

Success or error code describing how the transfer completed. Only applies for *kI3C_SlaveCompletionEvent*.

size_t transferredCount

Number of bytes actually transferred since start or last repeated start.

struct __i3c_slave_handle

#include <fsl_i3c.h> I3C slave handle structure.

Note: The contents of this structure are private and subject to change.

Public Members

i3c_slave_transfer_t transfer
I3C slave transfer copy.

bool isBusy
Whether transfer is busy.

bool wasTransmit
Whether the last transfer was a transmit.

uint32_t eventMask
Mask of enabled events.

uint32_t transferredCount
Count of bytes transferred.

i3c_slave_transfer_callback_t callback
Callback function called at transfer event.

void *userData
Callback parameter passed to callback.

size_t txFifoSize
Tx Fifo size

2.34 IMU: Inter CPU Messaging Unit

status_t IMU_Init(imu_link_t link)
Initializes the IMU module.

This function sets IMU to initialized state, including:

- Flush the send FIFO.
- Unlock the send FIFO.
- Set the water mark to (IMU_MAX_MSG_FIFO_WATER_MARK)
- Flush the receive FIFO.

If IMU_BUSY_POLL_COUNT is defined and non-zero, the function will timeout after the specified number of polling iterations and return kStatus_Timeout if flushing the receive FIFO takes too long.

Parameters

- link – IMU link.

Return values

- kStatus_Success – The IMU was initialized successfully.
- kStatus_InvalidArgument – Invalid link parameter.
- kStatus_Timeout – Timeout occurred while flushing the receive FIFO.

Returns
status_t

void IMU_Deinit(imu_link_t link)

De-initializes the IMU module.

Parameters

- link – IMU link.

static inline void IMU_WriteMsg(imu_link_t link, uint32_t msg)

Write one message to TX FIFO.

This function writes message to the TX FIFO, user need to make sure there is empty space in the TX FIFO, and TX FIFO not locked before calling this function.

Parameters

- link – IMU link.
- msg – The message to send.

static inline uint32_t IMU_ReadMsg(imu_link_t link)

Read one message from RX FIFO.

User need to make sure there is available message in the RX FIFO.

Parameters

- link – IMU link.

Returns
The message.

int32_t IMU_SendMsgsBlocking(imu_link_t link, const uint32_t *msgs, int32_t msgCount, bool lockSendFifo)

Blocking to send messages.

This function blocks until all messages have been filled to TX FIFO.

- If the TX FIFO is locked, this function returns IMU_ERR_TX_FIFO_LOCKED.
- If TX FIFO not locked, this function waits the available empty slot in TX FIFO, and fills the message to TX FIFO.
- To lock TX FIFO after filling all messages, set lockSendFifo to true.

If IMU_BUSY_POLL_COUNT is defined and non-zero, the function will timeout after the specified number of polling iterations and return IMU_ERR_TIMEOUT if waiting for FIFO space takes too long.

Parameters

- link – IMU link.
- msgs – The messages to send.
- msgCount – Message count, one message is a 32-bit word.
- lockSendFifo – If set to true, the TX FIFO is locked after all messages filled to TX FIFO.

Returns
If TX FIFO is locked, this function returns IMU_ERR_TX_FIFO_LOCKED. If a timeout occurs while waiting for FIFO space, it returns IMU_ERR_TIMEOUT. Otherwise, this function returns the actual message count sent out, which equals msgCount because this function is blocking until all messages have been filled into TX FIFO or a timeout occurs.

int32_t IMU_TrySendMsgs(imu_link_t link, const uint32_t *msgs, int32_t msgCount, bool lockSendFifo)

Try to send messages.

This function is similar with *IMU_SendMsgsBlocking*, the difference is, this function tries to send as many as possible, if there is not enough empty slot in TX FIFO, this function fills messages to available empty slots and returns how many messages have been filled.

- If the TX FIFO is locked, this function returns IMU_ERR_TX_FIFO_LOCKED.
- If TX FIFO not locked, this function fills messages to TX FIFO empty slot, and returns how many messages have been filled.
- If lockSendFifo is set to true, TX FIFO is locked after all messages have been filled to TX FIFO. In other word, TX FIFO is locked if the function return value equals msgCount, when lockSendFifo set to true.

Parameters

- link – IMU link.
- msgs – The messages to send.
- msgCount – Message count, one message is a 32-bit word.
- lockSendFifo – If set to true, the TX FIFO is locked after all messages filled to TX FIFO.

Returns

If TX FIFO is locked, this function returns IMU_ERR_TX_FIFO_LOCKED, otherwise, this function returns the actual message count sent out.

int32_t IMU_TryReceiveMsgs(imu_link_t link, uint32_t *msgs, int32_t desiredMsgCount, bool *needAckLock)

Try to receive messages.

This function tries to read messages from RX FIFO. It reads the messages already exists in RX FIFO and returns the actual read count.

- If the RX FIFO has enough messages, this function reads the messages and returns.
- If the RX FIFO does not have enough messages, this function the messages in RX FIFO and returns the actual read count.
- During message reading, if RX FIFO is empty and locked, in this case peer CPU will not send message until current CPU send lock ack message. Then this function returns the message count actually received, and sets needAckLock to true to inform upper layer.

Parameters

- link – IMU link.
- msgs – The buffer to read messages.
- desiredMsgCount – Desired read count, one message is a 32-bit word.
- needAckLock – Upper layer should always check this value. When this is set to true by this function, upper layer should send lock ack message to peer CPU.

Returns

Count of messages actually received.

int32_t IMU_ReceiveMsgsBlocking(imu_link_t link, uint32_t *msgs, int32_t desiredMsgCount, bool *needAckLock)

Blocking to receive messages.

This function blocks until all desired messages have been received or the RX FIFO is locked.

- If the RX FIFO has enough messages, this function reads the messages and returns.
- If the RX FIFO does not have enough messages, this function waits for the new messages.
- During message reading, if RX FIFO is empty and locked, in this case peer CPU will not send message until current CPU send lock ack message. Then this function returns the message count actually received, and sets needAckLock to true to inform upper layer.

Parameters

- link – IMU link.
- msgs – The buffer to read messages.
- desiredMsgCount – Desired read count, one message is a 32-bit word.
- needAckLock – Upper layer should always check this value. When this is set to true by this function, upper layer should send lock ack message to peer CPU.

Returns

Count of messages actually received.

int32_t IMU_SendMsgPtrBlocking(imu_link_t link, uint32_t msgPtr, bool lockSendFifo)

Blocking to send messages pointer.

Compared with *IMU_SendMsgsBlocking*, this function fills message pointer to TX FIFO, but not the message content.

This function blocks until the message pointer is filled to TX FIFO.

- If the TX FIFO is locked, this function returns IMU_ERR_TX_FIFO_LOCKED.
- If TX FIFO not locked, this function waits the available empty slot in TX FIFO, and fills the message pointer to TX FIFO.
- To lock TX FIFO after filling the message pointer, set lockSendFifo to true.

If IMU_BUSY_POLL_COUNT is defined and non-zero, the function will timeout after the specified number of polling iterations and return IMU_ERR_TIMEOUT if waiting for FIFO space takes too long.

Parameters

- link – IMU link.
- msgPtr – The buffer pointer to message to send.
- lockSendFifo – If set to true, the TX FIFO is locked after message pointer filled to TX FIFO.

Returns

If TX FIFO is locked, this function returns IMU_ERR_TX_FIFO_LOCKED. If a timeout occurs while waiting for FIFO space, it returns IMU_ERR_TIMEOUT. Otherwise, this function returns 0 to indicate success.

static inline void IMU_LockSendFifo(imu_link_t link, bool lock)

Lock or unlock the TX FIFO.

Parameters

- link – IMU link.
- lock – Use true to lock the FIFO, use false to unlock.

void IMU_FlushSendFifo(imu_link_t link)

Flush the send FIFO.

Flush all messages in send FIFO.

Parameters

- link – IMU link.

static inline void IMU_SetSendFifoWaterMark(imu_link_t link, uint8_t waterMark)

Set send FIFO warter mark.

The warter mark must be less than IMU_MAX_MSG_FIFO_WATER_MARK, i.e. 0 < waterMark <= IMU_MAX_MSG_FIFO_WATER_MARK.

Parameters

- link – IMU link.
- waterMark – Send FIFO warter mark.

static inline uint32_t IMU_GetReceivedMsgCount(imu_link_t link)

Get the message count in receive FIFO.

Parameters

- link – IMU link.

Returns

The message count in receive FIFO.

static inline uint32_t IMU_GetSendFifoEmptySpace(imu_link_t link)

Get the empty slot in send FIFO.

Parameters

- link – IMU link.

Returns

The empty slot count in send FIFO.

uint32_t IMU_GetStatusFlags(imu_link_t link)

Gets the IMU status flags.

Parameters

- link – IMU link.

Returns

Bit mask of the IMU status flags, see *_imu_status_flags*.

static inline void IMU_ClearPendingInterrupts(imu_link_t link, uint32_t mask)

Clear the IMU IRQ.

Parameters

- link – IMU link.
- mask – Bit mask of the interrupts to clear, see *_imu_interrupts*.

FSL_IMU_DRIVER_VERSION

IMU driver version.

enum _imu_status_flags

IMU status flags. .

Values:

enumerator kIMU_TxFifoEmpty

enumerator kIMU_TxFifoFull

enumerator kIMU_TxFifoAlmostFull

enumerator kIMU_TxFifoLocked

enumerator kIMU_RxFifoEmpty

enumerator kIMU_RxFifoFull

enumerator kIMU_RxFifoAlmostFull

enumerator kIMU_RxFifoLocked

enum _imu_interrupts

IMU interrupt. .

Values:

enumerator kIMU_RxMsgReadyInterrupt

enumerator kIMU_TxFifoSpaceAvailableInterrupt

IMU_MSG_FIFO_STATUS_MSG_FIFO_LOCKED_MASK

IMU_MSG_FIFO_STATUS_MSG_FIFO_ALMOST_FULL_MASK

IMU_MSG_FIFO_STATUS_MSG_FIFO_FULL_MASK

IMU_MSG_FIFO_STATUS_MSG_FIFO_EMPTY_MASK

IMU_MSG_FIFO_STATUS_MSG_COUNT_MASK

IMU_MSG_FIFO_STATUS_MSG_COUNT_SHIFT

IMU_MSG_FIFO_STATUS_WR_PTR_MASK

IMU_MSG_FIFO_STATUS_WR_PTR_SHIFT

IMU_MSG_FIFO_STATUS_RD_PTR_MASK

IMU_MSG_FIFO_STATUS_RD_PTR_SHIFT

IMU_MSG_FIFO_CNTL_MSG_RDY_INT_CLR_MASK

IMU_MSG_FIFO_CNTL_SP_AV_INT_CLR_MASK

IMU_MSG_FIFO_CNTL_FIFO_FLUSH_MASK

IMU_MSG_FIFO_CNTL_WAIT_FOR_ACK_MASK

IMU_MSG_FIFO_CNTL_FIFO_FULL_WATERMARK_MASK

IMU_MSG_FIFO_CNTL_FIFO_FULL_WATERMARK_SHIFT

IMU_MSG_FIFO_CNTL_FIFO_FULL_WATERMARK(x)

IMU_WR_MSG(link, msg)

IMU_RD_MSG(link)

IMU_RX_FIFO_LOCKED(link)

IMU_TX_FIFO_LOCKED(link)

IMU_TX_FIFO_ALMOST_FULL(link)

IMU_RX_FIFO_EMPTY(link)

Get Rx FIFO empty status.

IMU_LOCK_TX_FIFO(link)

IMU_UNLOCK_TX_FIFO(link)

IMU_RX_FIFO_MSG_COUNT(link)

IMU_TX_FIFO_MSG_COUNT(link)

IMU_RX_FIFO_MSG_COUNT_FROM_STATUS(rxFifoStatus)

IMU_RX_FIFO_LOCKED_FROM_STATUS(rxFifoStatus)

IMU_TX_FIFO_STATUS(link)

IMU_RX_FIFO_STATUS(link)

IMU_TX_FIFO_CNTL(link)

IMU_ERR_TX_FIFO_LOCKED

IMU driver returned error value.

IMU_ERR_TIMEOUT

IMU driver returned error value timeout.

IMU_MSG_FIFO_MAX_COUNT

Maximum message numbers in FIFO.

IMU_MAX_MSG_FIFO_WATER_MARK

Maximum message FIFO warter mark.

IMU_FIFO_SW_WRAPAROUND(ptr)

IMU_WR_PTR(link)

IMU_RD_PTR(link)

IMU_BUSY_POLL_COUNT

Maximum polling iterations for IMU waiting loops.

This parameter defines the maximum number of iterations for any polling loop in the IMU driver code before timing out and returning an error.

It applies to all waiting loops in IMU driver.

This is a count of loop iterations, not a time-based value.

If defined as 0, polling loops will continue indefinitely until their exit condition is met, which could potentially cause the system to hang if sensors don't respond or if communication interfaces fail.

struct IMU_Type

#include <fsl_imu.h> IMU register structure.

2.35 Common Driver

FSL_COMMON_DRIVER_VERSION

common driver version.

DEBUG_CONSOLE_DEVICE_TYPE_NONE

No debug console.

DEBUG_CONSOLE_DEVICE_TYPE_UART

Debug console based on UART.

DEBUG_CONSOLE_DEVICE_TYPE_LPUART

Debug console based on LPUART.

DEBUG_CONSOLE_DEVICE_TYPE_LPSCI

Debug console based on LPSCI.

DEBUG_CONSOLE_DEVICE_TYPE_USBCDC

Debug console based on USBCDC.

DEBUG_CONSOLE_DEVICE_TYPE_FLEXCOMM

Debug console based on FLEXCOMM.

DEBUG_CONSOLE_DEVICE_TYPE_IUART
Debug console based on i.MX UART.

DEBUG_CONSOLE_DEVICE_TYPE_VUSART
Debug console based on LPC_VUSART.

DEBUG_CONSOLE_DEVICE_TYPE_MINI_USART
Debug console based on LPC_USART.

DEBUG_CONSOLE_DEVICE_TYPE_SWO
Debug console based on SWO.

DEBUG_CONSOLE_DEVICE_TYPE_QSCI
Debug console based on QSCI.

MIN(a, b)
Computes the minimum of *a* and *b*.

MAX(a, b)
Computes the maximum of *a* and *b*.

UINT16_MAX
Max value of uint16_t type.

UINT32_MAX
Max value of uint32_t type.

UINTPTR_SIZE

UINT64_H(X)
Macro to get upper 32 bits of a 64-bit value

UINT64_L(X)
Macro to get lower 32 bits of a 64-bit value

MCUX_MASK_INVERT_8(mask)
8-bit mask inversion.

MCUX_MASK_INVERT_16(mask)
16-bit mask inversion.

MCUX_MASK_INVERT_32(mask)
32-bit mask inversion for completeness.

MCUX_REG_WRITE8(reg, value)
8-bit register write macro

MCUX_REG_WRITE16(reg, value)
16-bit register write macro

MCUX_REG_WRITE32(reg, value)
32-bit register write macro

MCUX_REG_READ8(reg)
8-bit register read macro

MCUX_REG_READ16(reg)
16-bit register read macro

MCUX_REG_READ32(reg)
32-bit register read macro

MCUX_REG_BIT_SET8(reg, mask)
8-bit register bit set macro

MCUX_REG_BIT_SET16(reg, mask)
16-bit register bit set macro

MCUX_REG_BIT_SET32(reg, mask)
32-bit register bit set macro

MCUX_REG_BIT_CLEAR8(reg, mask)
8-bit register bit clear macro

MCUX_REG_BIT_CLEAR16(reg, mask)
16-bit register bit clear macro

MCUX_REG_BIT_CLEAR32(reg, mask)
32-bit register bit clear macro

MCUX_REG_BIT_GET8(reg, mask)
8-bit register bit get macro

MCUX_REG_BIT_GET16(reg, mask)
16-bit register bit get macro

MCUX_REG_BIT_GET32(reg, mask)
32-bit register bit get macro

MCUX_REG_MODIFY8(reg, mask, value)
32-bit register read-modify-write macro

MCUX_REG_MODIFY16(reg, mask, value)
16-bit register read-modify-write macro

MCUX_REG_MODIFY32(reg, mask, value)
32-bit register read-modify-write macro

SDK_ATOMIC_LOCAL_ADD(addr, val)
Add value *val* from the variable at address *address*.

SDK_ATOMIC_LOCAL_SUB(addr, val)
Subtract value *val* to the variable at address *address*.

SDK_ATOMIC_LOCAL_SET(addr, bits)
Set the bits specifiled by *bits* to the variable at address *address*.

SDK_ATOMIC_LOCAL_CLEAR(addr, bits)
Clear the bits specifiled by *bits* to the variable at address *address*.

SDK_ATOMIC_LOCAL_TOGGLE(addr, bits)
Toggle the bits specifiled by *bits* to the variable at address *address*.

SDK_ATOMIC_LOCAL_CLEAR_AND_SET(addr, clearBits, setBits)
For the variable at address *address*, clear the bits specifiled by *clearBits* and set the bits specifiled by *setBits*.

SDK_ATOMIC_LOCAL_COMPARE_AND_SET(addr, expected, newValue)

For the variable at address *address*, check whether the value equal to *expected*. If value same as *expected* then update *newValue* to address and return **true** , else return **false** .

SDK_ATOMIC_LOCAL_TEST_AND_SET(addr, newValue)

For the variable at address *address*, set as *newValue* value and return old value.

USEC_TO_COUNT(us, clockFreqInHz)

Macro to convert a microsecond period to raw count value

COUNT_TO_USEC(count, clockFreqInHz)

Macro to convert a raw count value to microsecond

MSEC_TO_COUNT(ms, clockFreqInHz)

Macro to convert a millisecond period to raw count value

COUNT_TO_MSEC(count, clockFreqInHz)

Macro to convert a raw count value to millisecond

SDK_ISR_EXIT_BARRIER

SDK_ALIGN(var, alignbytes)

Macro to define a variable with alignbytes alignment

SDK_SIZEALIGN(var, alignbytes)

Macro to define a variable with L1 d-cache line size alignment

Macro to define a variable with L2 cache line size alignment

Macro to change a value to a given size aligned value (rounded up)

SDK_SIZEALIGN_UP(var, alignbytes)

Macro to change a value to a given size aligned value (rounded up), the wrapper of SDK_SIZEALIGN

SDK_SIZEALIGN_DOWN(var, alignbytes)

Macro to change a value to a given size aligned value (rounded down)

SDK_IS_ALIGNED(var, alignbytes)

Macro to check if a value is aligned to a given size

AT_NONCACHEABLE_SECTION(var)

Define a variable *var*, and place it in non-cacheable section.

AT_NONCACHEABLE_SECTION_ALIGN(var, alignbytes)

Define a variable *var*, and place it in non-cacheable section, the start address of the variable is aligned to *alignbytes*.

AT_NONCACHEABLE_SECTION_INIT(var)

Define a variable *var* with initial value, and place it in non-cacheable section.

AT_NONCACHEABLE_SECTION_ALIGN_INIT(var, alignbytes)

Define a variable *var* with initial value, and place it in non-cacheable section, the start address of the variable is aligned to *alignbytes*.

AT_CACHE_LINE_SECTION(var)

Define a variable *var*, which is cache line size aligned and be placed in CacheLineData section.

AT_CACHE_LINE_SECTION_INIT(var)

Define a variable *var* with initial value, which is cache line size aligned and be placed in CacheLineData.init section.

AT_QUICKACCESS_SECTION_CODE(func)
Place function in a section which can be accessed quickly by core.

AT_QUICKACCESS_SECTION_DATA(var)
Place data in a section which can be accessed quickly by core.

AT_QUICKACCESS_SECTION_DATA_ALIGN(var, alignbytes)
Place data in a section which can be accessed quickly by core, and the variable address is set to align with *alignbytes*.

MCUX_RAMFUNC
Function attribute to place function in RAM. For example, to place function my_func in ram, use like:

```
MCUX_RAMFUNC my_func
```

RAMFUNCTION_SECTION_CODE(func)
Place function in ram.

MCUX_DEPRECATED
Deprecated APIs.

MCUX_DEPRECATED_MACRO
Deprecated macros.

MCUX_EXPERIMENTAL
Experimental APIs.

MCUX_EXPERIMENTAL_MACRO
Experimental macros.

enum _status_groups
Status group numbers.

Values:

enumerator kStatusGroup_Generic
Group number for generic status codes.

enumerator kStatusGroup_FLASH
Group number for FLASH status codes.

enumerator kStatusGroup_LPSPI
Group number for LPSPI status codes.

enumerator kStatusGroup_FLEXIO_SPI
Group number for FLEXIO SPI status codes.

enumerator kStatusGroup_DSPI
Group number for DSPI status codes.

enumerator kStatusGroup_FLEXIO_UART
Group number for FLEXIO UART status codes.

enumerator kStatusGroup_FLEXIO_I2C
Group number for FLEXIO I2C status codes.

enumerator kStatusGroup_LPI2C
Group number for LPI2C status codes.

enumerator kStatusGroup_UART
Group number for UART status codes.

enumerator kStatusGroup_I2C
Group number for UART status codes.

enumerator kStatusGroup_LPSCI
Group number for LPSCI status codes.

enumerator kStatusGroup_LPUART
Group number for LPUART status codes.

enumerator kStatusGroup_SPI
Group number for SPI status code.

enumerator kStatusGroup_XRDC
Group number for XRDC status code.

enumerator kStatusGroup_SEMA42
Group number for SEMA42 status code.

enumerator kStatusGroup_SDHC
Group number for SDHC status code

enumerator kStatusGroup_SDMMC
Group number for SDMMC status code

enumerator kStatusGroup_SAI
Group number for SAI status code

enumerator kStatusGroup_MCG
Group number for MCG status codes.

enumerator kStatusGroup_SCG
Group number for SCG status codes.

enumerator kStatusGroup_SDSPI
Group number for SDSPI status codes.

enumerator kStatusGroup_FLEXIO_I2S
Group number for FLEXIO I2S status codes

enumerator kStatusGroup_FLEXIO_MCULCD
Group number for FLEXIO LCD status codes

enumerator kStatusGroup_FLASHIAP
Group number for FLASHIAP status codes

enumerator kStatusGroup_FLEXCOMM_I2C
Group number for FLEXCOMM I2C status codes

enumerator kStatusGroup_I2S
Group number for I2S status codes

enumerator kStatusGroup_IUART
Group number for IUART status codes

enumerator kStatusGroup_CSI
Group number for CSI status codes

enumerator kStatusGroup_MIPI_DSI
Group number for MIPI DSI status codes

enumerator kStatusGroup_SDRAMC
Group number for SDRAMC status codes.

enumerator kStatusGroup_POWER
Group number for POWER status codes.

enumerator kStatusGroup_ENET
Group number for ENET status codes.

enumerator kStatusGroup_PHY
Group number for PHY status codes.

enumerator kStatusGroup_TRGMUX
Group number for TRGMUX status codes.

enumerator kStatusGroup_SMARTCARD
Group number for SMARTCARD status codes.

enumerator kStatusGroup_LMEM
Group number for LMEM status codes.

enumerator kStatusGroup_QSPI
Group number for QSPI status codes.

enumerator kStatusGroup_DMA
Group number for DMA status codes.

enumerator kStatusGroup_EDMA
Group number for EDMA status codes.

enumerator kStatusGroup_DMAMGR
Group number for DMAMGR status codes.

enumerator kStatusGroup_FLEXCAN
Group number for FlexCAN status codes.

enumerator kStatusGroup_LTC
Group number for LTC status codes.

enumerator kStatusGroup_FLEXIO_CAMERA
Group number for FLEXIO CAMERA status codes.

enumerator kStatusGroup_LPC_SPI
Group number for LPC_SPI status codes.

enumerator kStatusGroup_LPC_USART
Group number for LPC_USART status codes.

enumerator kStatusGroup_DMIC
Group number for DMIC status codes.

enumerator kStatusGroup_SDIF
Group number for SDIF status codes.

enumerator kStatusGroup_SPIFI
Group number for SPIFI status codes.

enumerator kStatusGroup_OTP
Group number for OTP status codes.

enumerator kStatusGroup_MCAN
Group number for MCAN status codes.

enumerator kStatusGroup_CAAM
Group number for CAAM status codes.

enumerator kStatusGroup_ECSPI
Group number for ECSPI status codes.

enumerator kStatusGroup_USDHC
Group number for USDHC status codes.

enumerator kStatusGroup_LPC_I2C
Group number for LPC_I2C status codes.

enumerator kStatusGroup_DCP
Group number for DCP status codes.

enumerator kStatusGroup_MSCAN
Group number for MSCAN status codes.

enumerator kStatusGroup_ESAI
Group number for ESAI status codes.

enumerator kStatusGroup_FLEXSPI
Group number for FLEXSPI status codes.

enumerator kStatusGroup_MMDC
Group number for MMDC status codes.

enumerator kStatusGroup_PDM
Group number for MIC status codes.

enumerator kStatusGroup_SDMA
Group number for SDMA status codes.

enumerator kStatusGroup_ICS
Group number for ICS status codes.

enumerator kStatusGroup_SPDIF
Group number for SPDIF status codes.

enumerator kStatusGroup_LPC_MINISPI
Group number for LPC_MINISPI status codes.

enumerator kStatusGroup_HASHCRYPT
Group number for Hashcrypt status codes

enumerator kStatusGroup_LPC_SPI_SSP
Group number for LPC_SPI_SSP status codes.

enumerator kStatusGroup_I3C
Group number for I3C status codes

enumerator kStatusGroup_LPC_I2C_1
Group number for LPC_I2C_1 status codes.

enumerator kStatusGroup_NOTIFIER
Group number for NOTIFIER status codes.

enumerator kStatusGroup_DebugConsole
Group number for debug console status codes.

enumerator kStatusGroup_SEMC
Group number for SEMC status codes.

enumerator kStatusGroup_ApplicationRangeStart
Starting number for application groups.

enumerator kStatusGroup_IAP
Group number for IAP status codes

enumerator kStatusGroup_SFA
Group number for SFA status codes

enumerator kStatusGroup_SPC
Group number for SPC status codes.

enumerator kStatusGroup_PUF
Group number for PUF status codes.

enumerator kStatusGroup_TOUCH_PANEL
Group number for touch panel status codes

enumerator kStatusGroup_VBAT
Group number for VBAT status codes

enumerator kStatusGroup_XSPI
Group number for XSPI status codes

enumerator kStatusGroup_PNGDEC
Group number for PNGDEC status codes

enumerator kStatusGroup_JPEGDEC
Group number for JPEGDEC status codes

enumerator kStatusGroup_AUDMIX
Group number for AUDMIX status codes

enumerator kStatusGroup_HAL_GPIO
Group number for HAL GPIO status codes.

enumerator kStatusGroup_HAL_UART
Group number for HAL UART status codes.

enumerator kStatusGroup_HAL_TIMER
Group number for HAL TIMER status codes.

enumerator kStatusGroup_HAL_SPI
Group number for HAL SPI status codes.

enumerator kStatusGroup_HAL_I2C
Group number for HAL I2C status codes.

enumerator kStatusGroup_HAL_FLASH
Group number for HAL FLASH status codes.

enumerator kStatusGroup_HAL_PWM
Group number for HAL PWM status codes.

enumerator kStatusGroup_HAL_RNG
Group number for HAL RNG status codes.

enumerator kStatusGroup_HAL_I2S
Group number for HAL I2S status codes.

enumerator kStatusGroup_HAL_ADC_SENSOR
Group number for HAL ADC SENSOR status codes.

enumerator kStatusGroup_TIMERMANAGER
Group number for TiMER MANAGER status codes.

enumerator kStatusGroup_SERIALMANAGER
Group number for SERIAL MANAGER status codes.

enumerator kStatusGroup_LED
Group number for LED status codes.

enumerator kStatusGroup_BUTTON
Group number for BUTTON status codes.

enumerator kStatusGroup_EXTERN_EEPROM
Group number for EXTERN EEPROM status codes.

enumerator kStatusGroup_SHELL
Group number for SHELL status codes.

enumerator kStatusGroup_MEM_MANAGER
Group number for MEM MANAGER status codes.

enumerator kStatusGroup_LIST
Group number for List status codes.

enumerator kStatusGroup_OSA
Group number for OSA status codes.

enumerator kStatusGroup_COMMON_TASK
Group number for Common task status codes.

enumerator kStatusGroup_MSG
Group number for messaging status codes.

enumerator kStatusGroup_SDK_OCOTP
Group number for OCOTP status codes.

enumerator kStatusGroup_SDK_FLEXSPINOR
Group number for FLEXSPINOR status codes.

enumerator kStatusGroup_CODEC
Group number for codec status codes.

enumerator kStatusGroup_ASRC
Group number for codec status ASRC.

enumerator kStatusGroup_OTFAD
Group number for codec status codes.

enumerator kStatusGroup_SDIOSLV
Group number for SDIOSLV status codes.

enumerator kStatusGroup_MECC
Group number for MECC status codes.

enumerator kStatusGroup_ENET_QOS
Group number for ENET_QOS status codes.

enumerator kStatusGroup_LOG
Group number for LOG status codes.

enumerator kStatusGroup_I3CBUS
Group number for I3CBUS status codes.

enumerator kStatusGroup_QSCI
Group number for QSCI status codes.

enumerator kStatusGroup_ELEMU
Group number for ELEMU status codes.

enumerator kStatusGroup_QUEUEDSPI
Group number for QSPI status codes.

enumerator kStatusGroup_POWER_MANAGER
Group number for POWER_MANAGER status codes.

enumerator kStatusGroup_IPED
Group number for IPED status codes.

enumerator kStatusGroup_ELS_PKC
Group number for ELS PKC status codes.

enumerator kStatusGroup_CSS_PKC
Group number for CSS PKC status codes.

enumerator kStatusGroup_HOSTIF
Group number for HOSTIF status codes.

enumerator kStatusGroup_CLIF
Group number for CLIF status codes.

enumerator kStatusGroup_BMA
Group number for BMA status codes.

enumerator kStatusGroup_NETC
Group number for NETC status codes.

enumerator kStatusGroup_ELE
Group number for ELE status codes.

enumerator kStatusGroup_GLIKEY
Group number for GLIKEY status codes.

enumerator kStatusGroup_AON_POWER
Group number for AON_POWER status codes.

enumerator kStatusGroup_AON_COMMON
Group number for AON_COMMON status codes.

enumerator kStatusGroup_ENDAT3
Group number for ENDAT3 status codes.

enumerator kStatusGroup_HIPERFACE
Group number for HIPERFACE status codes.

enumerator kStatusGroup_NPX
Group number for NPX status codes.

enumerator kStatusGroup_ELA_CSEC
Group number for ELA_CSEC status codes.

enumerator kStatusGroup_FLEXIO_T_FORMAT
Group number for T-format status codes.

enumerator kStatusGroup_FLEXIO_A_FORMAT
Group number for A-format status codes.

enumerator kStatusGroup_LPC_QSPI
Group number for LPC QSPI status codes.

enumerator kStatusGroup_EVENT_CTRL
Group number for Event controller status codes.

Generic status return codes.

Values:

enumerator kStatus_Success
Generic status for Success.

enumerator kStatus_Fail
Generic status for Fail.

enumerator kStatus_ReadOnly
Generic status for read only failure.

enumerator kStatus_OutOfRange
Generic status for out of range access.

enumerator kStatus_InvalidArgument
Generic status for invalid argument check.

enumerator kStatus_Timeout
Generic status for timeout.

enumerator kStatus_NoTransferInProgress
Generic status for no transfer in progress.

enumerator kStatus_Busy
Generic status for module is busy.

enumerator kStatus_NoData
Generic status for no data is found for the operation.

typedef int32_t status_t
Type used for all status and error return values.

void *SDK_Malloc(size_t size, size_t alignbytes)
Allocate memory with given alignment and aligned size.

This is provided to support the dynamically allocated memory used in cache-able region.

Parameters

- size – The length required to malloc.
- alignbytes – The alignment size.

Return values

The – allocated memory.

void SDK_Free(void *ptr)

Free memory.

Parameters

- ptr – The memory to be release.

void SDK_DelayAtLeastUs(uint32_t delayTime_us, uint32_t coreClock_Hz)

Delay at least for some time. Please note that, this API uses while loop for delay, different run-time environments make the time not precise, if precise delay count was needed, please implement a new delay function with hardware timer.

Parameters

- delayTime_us – Delay time in unit of microsecond.
- coreClock_Hz – Core clock frequency with Hz.

static inline *status_t* EnableIRQ(IRQn_Type interrupt)

Enable specific interrupt.

Enable LEVEL1 interrupt. For some devices, there might be multiple interrupt levels. For example, there are NVIC and intmux. Here the interrupts connected to NVIC are the LEVEL1 interrupts, because they are routed to the core directly. The interrupts connected to intmux are the LEVEL2 interrupts, they are routed to NVIC first then routed to core.

This function only enables the LEVEL1 interrupts. The number of LEVEL1 interrupts is indicated by the feature macro FSL_FEATURE_NUMBER_OF_LEVEL1_INT_VECTORS.

Parameters

- interrupt – The IRQ number.

Return values

- kStatus_Success – Interrupt enabled successfully
- kStatus_Fail – Failed to enable the interrupt

static inline *status_t* DisableIRQ(IRQn_Type interrupt)

Disable specific interrupt.

Disable LEVEL1 interrupt. For some devices, there might be multiple interrupt levels. For example, there are NVIC and intmux. Here the interrupts connected to NVIC are the LEVEL1 interrupts, because they are routed to the core directly. The interrupts connected to intmux are the LEVEL2 interrupts, they are routed to NVIC first then routed to core.

This function only disables the LEVEL1 interrupts. The number of LEVEL1 interrupts is indicated by the feature macro FSL_FEATURE_NUMBER_OF_LEVEL1_INT_VECTORS.

Parameters

- interrupt – The IRQ number.

Return values

- kStatus_Success – Interrupt disabled successfully
- kStatus_Fail – Failed to disable the interrupt

static inline *status_t* EnableIRQWithPriority(IRQn_Type interrupt, uint8_t priNum)

Enable the IRQ, and also set the interrupt priority.

Only handle LEVEL1 interrupt. For some devices, there might be multiple interrupt levels. For example, there are NVIC and intmux. Here the interrupts connected to NVIC are the LEVEL1 interrupts, because they are routed to the core directly. The interrupts connected to intmux are the LEVEL2 interrupts, they are routed to NVIC first then routed to core.

This function only handles the LEVEL1 interrupts. The number of LEVEL1 interrupts is indicated by the feature macro FSL_FEATURE_NUMBER_OF_LEVEL1_INT_VECTORS.

Parameters

- interrupt – The IRQ to Enable.
- priNum – Priority number set to interrupt controller register.

Return values

- kStatus_Success – Interrupt priority set successfully
- kStatus_Fail – Failed to set the interrupt priority.

static inline *status_t* IRQ_SetPriority(IRQn_Type interrupt, uint8_t priNum)

Set the IRQ priority.

Only handle LEVEL1 interrupt. For some devices, there might be multiple interrupt levels. For example, there are NVIC and intmux. Here the interrupts connected to NVIC are the LEVEL1 interrupts, because they are routed to the core directly. The interrupts connected to intmux are the LEVEL2 interrupts, they are routed to NVIC first then routed to core.

This function only handles the LEVEL1 interrupts. The number of LEVEL1 interrupts is indicated by the feature macro FSL_FEATURE_NUMBER_OF_LEVEL1_INT_VECTORS.

Parameters

- interrupt – The IRQ to set.
- priNum – Priority number set to interrupt controller register.

Return values

- kStatus_Success – Interrupt priority set successfully
- kStatus_Fail – Failed to set the interrupt priority.

static inline *status_t* IRQ_ClearPendingIRQ(IRQn_Type interrupt)

Clear the pending IRQ flag.

Only handle LEVEL1 interrupt. For some devices, there might be multiple interrupt levels. For example, there are NVIC and intmux. Here the interrupts connected to NVIC are the LEVEL1 interrupts, because they are routed to the core directly. The interrupts connected to intmux are the LEVEL2 interrupts, they are routed to NVIC first then routed to core.

This function only handles the LEVEL1 interrupts. The number of LEVEL1 interrupts is indicated by the feature macro FSL_FEATURE_NUMBER_OF_LEVEL1_INT_VECTORS.

Parameters

- interrupt – The flag which IRQ to clear.

Return values

- kStatus_Success – Interrupt priority set successfully
- kStatus_Fail – Failed to set the interrupt priority.

static inline uint32_t DisableGlobalIRQ(void)

Disable the global IRQ.

Disable the global interrupt and return the current primask register. User is required to provided the primask register for the *EnableGlobalIRQ()*.

Returns

Current primask value.

static inline void EnableGlobalIRQ(uint32_t primask)

Enable the global IRQ.

Set the primask register with the provided primask value but not just enable the primask. The idea is for the convenience of integration of RTOS. some RTOS get its own management

mechanism of primask. User is required to use the *EnableGlobalIRQ()* and *DisableGlobalIRQ()* in pair.

Parameters

- primask – value of primask register to be restored. The primask value is supposed to be provided by the *DisableGlobalIRQ()*.

static inline bool __SDK_AtomicLocalCompareAndSet(uint32_t *addr, uint32_t expected, uint32_t newValue)

static inline uint32_t __SDK_AtomicTestAndSet(uint32_t *addr, uint32_t newValue)

FSL_DRIVER_TRANSFER_DOUBLE_WEAK_IRQ

Macro to use the default weak IRQ handler in drivers.

MAKE_STATUS(group, code)

Construct a status code value from a group and code number.

MAKE_VERSION(major, minor, bugfix)

Construct the version number for drivers.

The driver version is a 32-bit number, for both 32-bit platforms(such as Cortex M) and 16-bit platforms(such as DSC).

```
| Unused     || Major Version || Minor Version ||  Bug Fix     |
31        25  24          17  16          9  8           0
```

ARRAY_SIZE(x)

Computes the number of elements in an array.

SUPPRESS_FALL_THROUGH_WARNING()

For switch case code block, if case section ends without "break;" statement, there wil be fallthrough warning with compiler flag -Wextra or -Wimplicit-fallthrough=n when using armgcc. To suppress this warning, "SUPPRESS_FALL_THROUGH_WARNING();" need to be added at the end of each case section which misses "break;"statement.

MSDK_REG_SECURE_ADDR(x)

Convert the register address to the one used in secure mode.

MSDK_REG_NONSECURE_ADDR(x)

Convert the register address to the one used in non-secure mode.

MSDK_HAS_DWT_CYCCNT

The chip supports DWT CYCCNT or not.

MSDK_INVALID_IRQ_HANDLER

Invalid IRQ handler address.

2.36 LPADC: 12-bit SAR Analog-to-Digital Converter Driver

enum __lpadc_status_flags

Define hardware flags of the module.

Values:

enumerator kLPADC_ResultFIFO0OverflowFlag

Indicates that more data has been written to the Result FIFO 0 than it can hold.

enumerator kLPADC_ResultFIFO0ReadyFlag

Indicates when the number of valid datawords in the result FIFO 0 is greater than the setting watermark level.

enumerator kLPADC_TriggerExceptionFlag

Indicates that a trigger exception event has occurred.

enumerator kLPADC_TriggerCompletionFlag

Indicates that a trigger completion event has occurred.

enumerator kLPADC_CalibrationReadyFlag

Indicates that the calibration process is done.

enumerator kLPADC_ActiveFlag

Indicates that the ADC is in active state.

enumerator kLPADC_ResultFIFOOverflowFlag

To compilitable with old version, do not recommend using this, please use *kLPADC_ResultFIFO0OverflowFlag* as instead.

enumerator kLPADC_ResultFIFOReadyFlag

To compilitable with old version, do not recommend using this, please use *kLPADC_ResultFIFO0ReadyFlag* as instead.

enum _lpadc_interrupt_enable

Define interrupt switchers of the module.

Note: LPADC of different chips supports different number of trigger sources, please check the Reference Manual for details.

Values:

enumerator kLPADC_ResultFIFO0OverflowInterruptEnable

Configures ADC to generate overflow interrupt requests when FOF0 flag is asserted.

enumerator kLPADC_FIFO0WatermarkInterruptEnable

Configures ADC to generate watermark interrupt requests when RDY0 flag is asserted.

enumerator kLPADC_ResultFIFOOverflowInterruptEnable

To compilitable with old version, do not recommend using this, please use *kLPADC_ResultFIFO0OverflowInterruptEnable* as instead.

enumerator kLPADC_FIFOWatermarkInterruptEnable

To compilitable with old version, do not recommend using this, please use *kLPADC_FIFO0WatermarkInterruptEnable* as instead.

enumerator kLPADC_TriggerExceptionInterruptEnable

Configures ADC to generate trigger exception interrupt.

enumerator kLPADC_Trigger0CompletionInterruptEnable

Configures ADC to generate interrupt when trigger 0 completion.

enumerator kLPADC_Trigger1CompletionInterruptEnable

Configures ADC to generate interrupt when trigger 1 completion.

enum _lpadc_trigger_status_flags

The enumerator of lpadc trigger status flags, including interrupted flags and completed flags.

Note: LPADC of different chips supports different number of trigger sources, please check the Reference Manual for details.

Values:

enumerator kLPADC_Trigger0InterruptedFlag

Trigger 0 is interrupted by a high priority exception.

enumerator kLPADC_Trigger1InterruptedFlag

Trigger 1 is interrupted by a high priority exception.

enumerator kLPADC_Trigger0CompletedFlag

Trigger 0 is completed and trigger 0 has enabled completion interrupts.

enumerator kLPADC_Trigger1CompletedFlag

Trigger 1 is completed and trigger 1 has enabled completion interrupts.

enum _lpadc_sample_scale_mode

Define enumeration of sample scale mode.

The sample scale mode is used to reduce the selected ADC analog channel input voltage level by a factor. The maximum possible voltage on the ADC channel input should be considered when selecting a scale mode to ensure that the reducing factor always results voltage level at or below the VREFH reference. This reducing capability allows conversion of analog inputs higher than VREFH. A-side and B-side channel inputs are both scaled using the scale mode.

Values:

enumerator kLPADC_SamplePartScale

Use divided input voltage signal. (For scale select,please refer to the reference manual).

enumerator kLPADC_SampleFullScale

Full scale (Factor of 1).

enum _lpadc_sample_channel_mode

Define enumeration of channel sample mode.

The channel sample mode configures the channel with single-end/differential/dual-single-end, side A/B.

Values:

enumerator kLPADC_SampleChannelSingleEndSideA
Single-end mode, only A-side channel is converted.

enumerator kLPADC_SampleChannelSingleEndSideB
Single-end mode, only B-side channel is converted.

enumerator kLPADC_SampleChannelDiffBothSideAB
Differential mode, the ADC result is (CHnA-CHnB).

enumerator kLPADC_SampleChannelDiffBothSideBA
Differential mode, the ADC result is (CHnB-CHnA).

enumerator kLPADC_SampleChannelDiffBothSide
Differential mode, the ADC result is (CHnA-CHnB).

enumerator kLPADC_SampleChannelDualSingleEndBothSide
Dual-Single-Ended Mode. Both A side and B side channels are converted independently.

enum _lpadc_hardware_average_mode
Define enumeration of hardware average selection.

It Selects how many ADC conversions are averaged to create the ADC result. An internal storage buffer is used to capture temporary results while the averaging iterations are executed.

Note: Some enumerator values are not available on some devices, mainly depends on the size of AVGS field in CMDH register.

Values:

enumerator kLPADC_HardwareAverageCount1
Single conversion.

enumerator kLPADC_HardwareAverageCount2
2 conversions averaged.

enumerator kLPADC_HardwareAverageCount4
4 conversions averaged.

enumerator kLPADC_HardwareAverageCount8
8 conversions averaged.

enumerator kLPADC_HardwareAverageCount16
16 conversions averaged.

enumerator kLPADC_HardwareAverageCount32
32 conversions averaged.

enumerator kLPADC_HardwareAverageCount64

64 conversions averaged.

enumerator kLPADC_HardwareAverageCount128

128 conversions averaged.

enum _lpadc_sample_time_mode

Define enumeration of sample time selection.

The shortest sample time maximizes conversion speed for lower impedance inputs. Extending sample time allows higher impedance inputs to be accurately sampled. Longer sample times can also be used to lower overall power consumption when command looping and sequencing is configured and high conversion rates are not required.

Values:

enumerator kLPADC_SampleTimeADCK3

3 ADCK cycles total sample time.

enumerator kLPADC_SampleTimeADCK5

5 ADCK cycles total sample time.

enumerator kLPADC_SampleTimeADCK7

7 ADCK cycles total sample time.

enumerator kLPADC_SampleTimeADCK11

11 ADCK cycles total sample time.

enumerator kLPADC_SampleTimeADCK19

19 ADCK cycles total sample time.

enumerator kLPADC_SampleTimeADCK35

35 ADCK cycles total sample time.

enumerator kLPADC_SampleTimeADCK67

69 ADCK cycles total sample time.

enumerator kLPADC_SampleTimeADCK131

131 ADCK cycles total sample time.

enum _lpadc_hardware_compare_mode

Define enumeration of hardware compare mode.

After an ADC channel input is sampled and converted and any averaging iterations are performed, this mode setting guides operation of the automatic compare function to optionally only store when the compare operation is true. When compare is enabled, the conversion result is compared to the compare values.

Values:

enumerator kLPADC_HardwareCompareDisabled

Compare disabled.

enumerator kLPADC_HardwareCompareStoreOnTrue

Compare enabled. Store on true.

enumerator kLPADC_HardwareCompareRepeatUntilTrue

Compare enabled. Repeat channel acquisition until true.

enum _lpadc_conversion_resolution_mode

Define enumeration of conversion resolution mode.

Configure the resolution bit in specific conversion type. For detailed resolution accuracy, see to *lpadc_sample_channel_mode_t*

Values:

enumerator kLPADC_ConversionResolutionStandard

Standard resolution. Single-ended 12-bit conversion, Differential 13-bit conversion with 2's complement output.

enumerator kLPADC_ConversionResolutionHigh

High resolution. Single-ended 16-bit conversion; Differential 16-bit conversion with 2's complement output.

enum _lpadc_conversion_average_mode

Define enumeration of conversion averages mode.

Configure the converion average number for auto-calibration.

Note: Some enumerator values are not available on some devices, mainly depends on the size of CAL_AVGS field in CTRL register.

Values:

enumerator kLPADC_ConversionAverage1

Single conversion.

enumerator kLPADC_ConversionAverage2

2 conversions averaged.

enumerator kLPADC_ConversionAverage4

4 conversions averaged.

enumerator kLPADC_ConversionAverage8

8 conversions averaged.

enumerator kLPADC_ConversionAverage16

16 conversions averaged.

enumerator kLPADC_ConversionAverage32

32 conversions averaged.

enumerator kLPADC_ConversionAverage64
64 conversions averaged.

enumerator kLPADC_ConversionAverage128
128 conversions averaged.

enumerator kLPADC_ConversionAverageMax

enum _lpadc_reference_voltage_mode
Define enumeration of reference voltage source.

For detail information, need to check the SoC's specification.

Values:

enumerator kLPADC_ReferenceVoltageAlt1
Option 1 setting.

enumerator kLPADC_ReferenceVoltageAlt2
Option 2 setting.

enumerator kLPADC_ReferenceVoltageAlt3
Option 3 setting.

enum _lpadc_power_level_mode
Define enumeration of power configuration.

Configures the ADC for power and performance. In the highest power setting the highest conversion rates will be possible. Refer to the device data sheet for power and performance capabilities for each setting.

Values:

enumerator kLPADC_PowerLevelAlt1
Lowest power setting.

enumerator kLPADC_PowerLevelAlt2
Next lowest power setting.

enumerator kLPADC_PowerLevelAlt3
...

enumerator kLPADC_PowerLevelAlt4
Highest power setting.

enum _lpadc_offset_calibration_mode
Define enumeration of offset calibration mode.

Values:

enumerator kLPADC_OffsetCalibration12bitMode
12 bit offset calibration mode.

enumerator kLPADC_OffsetCalibration16bitMode

16 bit offset calibration mode.

enum _lpadc_trigger_priority_policy

Define enumeration of trigger priority policy.

This selection controls how higher priority triggers are handled.

Note: kLPADC_TriggerPriorityPreemptSubsequently is not available on some devices, mainly depends on the size of TPRICTRL field in CFG register.

Values:

enumerator kLPADC_ConvPreemptImmediatelyNotAutoResumed

If a higher priority trigger is detected during command processing, the current conversion is aborted and the new command specified by the trigger is started, when higher priority conversion finishes, the preempted conversion is not automatically resumed or restarted.

enumerator kLPADC_ConvPreemptSoftlyNotAutoResumed

If a higher priority trigger is received during command processing, the current conversion is completed (including averaging iterations and compare function if enabled) and stored to the result FIFO before the higher priority trigger/command is initiated, when higher priority conversion finishes, the preempted conversion is not resumed or restarted.

enumerator kLPADC_ConvPreemptImmediatelyAutoRestarted

If a higher priority trigger is detected during command processing, the current conversion is aborted and the new command specified by the trigger is started, when higher priority conversion finishes, the preempted conversion will automatically be restarted.

enumerator kLPADC_ConvPreemptSoftlyAutoRestarted

If a higher priority trigger is received during command processing, the current conversion is completed (including averaging iterations and compare function if enabled) and stored to the result FIFO before the higher priority trigger/command is initiated, when higher priority conversion finishes, the preempted conversion will automatically be restarted.

enumerator kLPADC_ConvPreemptImmediatelyAutoResumed

If a higher priority trigger is detected during command processing, the current conversion is aborted and the new command specified by the trigger is started, when higher priority conversion finishes, the preempted conversion will automatically be resumed.

enumerator kLPADC_ConvPreemptSoftlyAutoResumed

If a higher priority trigger is received during command processing, the current conversion is completed (including averaging iterations and compare function if enabled) and stored to the result FIFO before the higher priority trigger/command is initiated, when higher priority conversion finishes, the preempted conversion will be automatically be resumed.

enumerator kLPADC_TriggerPriorityPreemptImmediately

Legacy support is not recommended as it only ensures compatibility with older versions.

enumerator kLPADC_TriggerPriorityPreemptSoftly

Legacy support is not recommended as it only ensures compatibility with older versions.

enumerator kLPADC_TriggerPriorityExceptionDisabled

High priority trigger exception disabled.

enum _lpadc_tune_value

Define enumeration of tune value.

Values:

enumerator kLPADC_TuneValue0

Tune value 0.

enumerator kLPADC_TuneValue1

Tune value 1.

enumerator kLPADC_TuneValue2

Tune value 2.

enumerator kLPADC_TuneValue3

Tune value 3.

typedef enum *_lpadc_sample_scale_mode* lpadc_sample_scale_mode_t

Define enumeration of sample scale mode.

The sample scale mode is used to reduce the selected ADC analog channel input voltage level by a factor. The maximum possible voltage on the ADC channel input should be considered when selecting a scale mode to ensure that the reducing factor always results voltage level at or below the VREFH reference. This reducing capability allows conversion of analog inputs higher than VREFH. A-side and B-side channel inputs are both scaled using the scale mode.

typedef enum *_lpadc_sample_channel_mode* lpadc_sample_channel_mode_t

Define enumeration of channel sample mode.

The channel sample mode configures the channel with single-end/differential/dual-single-end, side A/B.

typedef enum *_lpadc_hardware_average_mode* lpadc_hardware_average_mode_t

Define enumeration of hardware average selection.

It Selects how many ADC conversions are averaged to create the ADC result. An internal storage buffer is used to capture temporary results while the averaging iterations are executed.

Note: Some enumerator values are not available on some devices, mainly depends on the size of AVGS field in CMDH register.

typedef enum *_lpadc_sample_time_mode* lpadc_sample_time_mode_t

Define enumeration of sample time selection.

The shortest sample time maximizes conversion speed for lower impedance inputs. Extending sample time allows higher impedance inputs to be accurately sampled. Longer sample times can also be used to lower overall power consumption when command looping and sequencing is configured and high conversion rates are not required.

typedef enum *_lpadc_hardware_compare_mode* lpadc_hardware_compare_mode_t

Define enumeration of hardware compare mode.

After an ADC channel input is sampled and converted and any averaging iterations are performed, this mode setting guides operation of the automatic compare function to optionally only store when the compare operation is true. When compare is enabled, the conversion result is compared to the compare values.

typedef enum *_lpadc_conversion_resolution_mode* lpadc_conversion_resolution_mode_t

Define enumeration of conversion resolution mode.

Configure the resolution bit in specific conversion type. For detailed resolution accuracy, see to *lpadc_sample_channel_mode_t*

typedef enum *_lpadc_conversion_average_mode* lpadc_conversion_average_mode_t

Define enumeration of conversion averages mode.

Configure the converion average number for auto-calibration.

Note: Some enumerator values are not available on some devices, mainly depends on the size of CAL_AVGS field in CTRL register.

---

typedef enum *_lpadc_reference_voltage_mode* lpadc_reference_voltage_source_t

Define enumeration of reference voltage source.

For detail information, need to check the SoC's specification.

typedef enum *_lpadc_power_level_mode* lpadc_power_level_mode_t

Define enumeration of power configuration.

Configures the ADC for power and performance. In the highest power setting the highest conversion rates will be possible. Refer to the device data sheet for power and performance capabilities for each setting.

typedef enum *_lpadc_offset_calibration_mode* lpadc_offset_calibration_mode_t

Define enumeration of offset calibration mode.

typedef enum *_lpadc_trigger_priority_policy* lpadc_trigger_priority_policy_t

Define enumeration of trigger priority policy.

This selection controls how higher priority triggers are handled.

Note: **kLPADC_TriggerPriorityPreemptSubsequently** is not available on some devices, mainly depends on the size of TPRICTRL field in CFG register.

---

typedef enum *_lpadc_tune_value* lpadc_tune_value_t

Define enumeration of tune value.

typedef struct *_lpadc_calibration_value* lpadc_calibration_value_t

A structure of calibration value.

LPADC_CONVERSION_COMPLETE_TIMEOUT

Max loops to wait for LPADC conversion complete.

When doing calibration, driver will wait for the completion of conversion. This parameter defines how many loops to check completion before return timeout. If defined as 0, driver will wait forever until completion.

LPADC_CALIBRATION_READY_TIMEOUT

Max loops to wait for LPADC calibration ready.

Before doing calibration, driver will wait for the calibration ready. This parameter defines how many loops to check the calibration ready. If defined as 0, driver will wait forever until ready.

LPADC_GAIN_CAL_READY_TIMEOUT

Max loops to wait for LPADC gain calibration GAIN_CAL ready.

Before doing calibration, driver will wait for the gain calibration GAIN_CAL ready. This parameter defines how many loops to check the gain calibration GAIN_CAL ready. If defined as 0, driver will wait forever until ready.

LPADC_USE_FIXED_POINT_GAIN_CALCULATION

Use fixed point arithmetic for the auto-calibration gain calculation.

When set to 1, *LPADC_FinishAutoCalibration()* calculates the gain conversion result with integer-only arithmetic instead of the float/double software library, which saves several kB of flash on code size sensitive devices. The fixed point result keeps the gain fraction in Q17.14, so the two least significant bits of the gain conversion result are always zero.

ADC_OFSTRIM_OFSTRIM_MAX

ADC_OFSTRIM_OFSTRIM_SIGN

LPADC_GET_ACTIVE_COMMAND_STATUS(statusVal)

Define the MACRO function to get command status from status value.

The statusVal is the return value from *LPADC_GetStatusFlags()*.

LPADC_GET_ACTIVE_TRIGGER_STATUE(statusVal)

Define the MACRO function to get trigger status from status value.

The statusVal is the return value from *LPADC_GetStatusFlags()*.

void LPADC_Init(ADC_Type *base, const *lpadc_config_t* *config)

Initializes the LPADC module.

Parameters

- base – LPADC peripheral base address.
- config – Pointer to configuration structure. See “lpadc_config_t”.

void LPADC_GetDefaultConfig(*lpadc_config_t* *config)

Gets an available pre-defined settings for initial configuration.

This function initializes the converter configuration structure with an available settings. The default values are:

```
config->enableInDozeMode        = true;
config->enableAnalogPreliminary = false;
config->powerUpDelay            = 0x80;
config->referenceVoltageSource  = kLPADC_ReferenceVoltageAlt1;
config->powerLevelMode          = kLPADC_PowerLevelAlt1;
config->triggerPriorityPolicy   = kLPADC_TriggerPriorityPreemptImmediately;
config->enableConvPause         = false;
config->convPauseDelay          = 0U;
config->FIFOWatermark           = 0U;
```

Parameters

- config – Pointer to configuration structure.

void LPADC_Deinit(ADC_Type *base)

De-initializes the LPADC module.

Parameters

- base – LPADC peripheral base address.

static inline void LPADC_Enable(ADC_Type *base, bool enable)

Switch on/off the LPADC module.

Parameters

- base – LPADC peripheral base address.
- enable – switcher to the module.

static inline void LPADC_DoResetFIFO(ADC_Type *base)

Do reset the conversion FIFO.

Parameters

- base – LPADC peripheral base address.

static inline void LPADC_DoResetConfig(ADC_Type *base)

Do reset the module's configuration.

Reset all ADC internal logic and registers, except the Control Register (ADCx_CTRL).

Parameters

- base – LPADC peripheral base address.

static inline uint32_t LPADC_GetStatusFlags(ADC_Type *base)

Get status flags.

Parameters

- base – LPADC peripheral base address.

Returns

status flags' mask. See to *_lpadc_status_flags*.

static inline void LPADC_ClearStatusFlags(ADC_Type *base, uint32_t mask)

Clear status flags.

Only the flags can be cleared by writing ADCx_STATUS register would be cleared by this API.

Parameters

- base – LPADC peripheral base address.
- mask – Mask value for flags to be cleared. See to *_lpadc_status_flags*.

static inline uint32_t LPADC_GetTriggerStatusFlags(ADC_Type *base)

Get trigger status flags to indicate which trigger sequences have been completed or interrupted by a high priority trigger exception.

Note: On some devices, the trigger completion status may be asserted before the final command in a chained trigger sequence starts to execute. When using chained commands, do not rely on trigger completion status alone to guarantee that all conversion results are already available in the FIFO. Use FIFO ready indication together with result tags, or stall the last command with WAIT_TRIG when that sequencing model is acceptable for the application.

Parameters

- base – LPADC peripheral base address.

Returns

The OR'ed value of *_lpadc_trigger_status_flags*.

static inline void LPADC_ClearTriggerStatusFlags(ADC_Type *base, uint32_t mask)

Clear trigger status flags.

Parameters

- base – LPADC peripheral base address.
- mask – The mask of trigger status flags to be cleared, should be the OR'ed value of *_lpadc_trigger_status_flags*.

static inline void LPADC_EnableInterrupts(ADC_Type *base, uint32_t mask)

Enable interrupts.

Note: When enabling trigger completion interrupts (kLPADC_TriggerXCompletionInterruptEnable) on some devices, the interrupt may occur before the final command in a chained trigger sequence starts to execute. For multi-command trigger sequences, do not use the trigger completion interrupt alone as the indication that all expected results are already stored in the FIFO.

Parameters

- base – LPADC peripheral base address.
- mask – Mask value for interrupt events. See to *_lpadc_interrupt_enable*.

static inline void LPADC_DisableInterrupts(ADC_Type *base, uint32_t mask)

Disable interrupts.

Parameters

- base – LPADC peripheral base address.
- mask – Mask value for interrupt events. See to *_lpadc_interrupt_enable*.

static inline void LPADC_EnableFIFOWatermarkDMA(ADC_Type *base, bool enable)

Switch on/off the DMA trigger for FIFO watermark event.

Parameters

- base – LPADC peripheral base address.
- enable – Switcher to the event.

static inline uint32_t LPADC_GetConvResultCount(ADC_Type *base)

Get the count of result kept in conversion FIFO.

Parameters

- base – LPADC peripheral base address.

Returns

The count of result kept in conversion FIFO.

bool LPADC_GetConvResult(ADC_Type *base, *lpadc_conv_result_t* *result)

Get the result in conversion FIFO.

Parameters

- base – LPADC peripheral base address.
- result – Pointer to structure variable that keeps the conversion result in conversion FIFO.

Returns

Status whether FIFO entry is valid.

void LPADC_GetConvResultBlocking(ADC_Type *base, *lpadc_conv_result_t* *result)

Get the result in conversion FIFO using blocking method.

Parameters

- base – LPADC peripheral base address.
- result – Pointer to structure variable that keeps the conversion result in conversion FIFO.

void LPADC_SetConvTriggerConfig(ADC_Type *base, uint32_t triggerId, const *lpadc_conv_trigger_config_t* *config)

Configure the conversion trigger source.

Each programmable trigger can launch the conversion command in command buffer.

Parameters

- base – LPADC peripheral base address.
- triggerId – ID for each trigger. Typically, the available value range is from 0.
- config – Pointer to configuration structure. See to *lpadc_conv_trigger_config_t*.

void LPADC_GetDefaultConvTriggerConfig(*lpadc_conv_trigger_config_t* *config)

Gets an available pre-defined settings for trigger's configuration.

This function initializes the trigger's configuration structure with an available settings. The default values are:

```
config->targetCommandId       = 0U;
config->delayPower            = 0U;
config->priority              = 0U;
config->channelAFIFOSelect    = 0U;
config->channelBFIFOSelect    = 0U;
config->enableHardwareTrigger = false;
```

Parameters

- config – Pointer to configuration structure.

static inline void LPADC_DoSoftwareTrigger(ADC_Type *base, uint32_t triggerIdMask)

Do software trigger to conversion command.

Parameters

- base – LPADC peripheral base address.
- triggerIdMask – Mask value for software trigger indexes, which count from zero.

static inline void LPADC_EnableHardwareTriggerCommandSelection(ADC_Type *base, uint32_t triggerId, bool enable)

Enable hardware trigger command selection.

This function will use the hardware trigger command from ADC_ETC.The trigger command is then defined by ADC hardware trigger command selection field in ADC_ETC->TRIGx_CHAINy_z_n[CSEL].

Parameters

- base – LPADC peripheral base address.
- triggerId – ID for each trigger. Typically, the available value range is from 0.
- enable – True to enable or flase to disable.

void LPADC_SetConvCommandConfig(ADC_Type *base, uint32_t commandId, const *lpadc_conv_command_config_t* *config)

Configure conversion command.

Note: The number of compare value register on different chips is different, that is mean in some chips, some command buffers do not have the compare functionality.

Parameters

- base – LPADC peripheral base address.
- commandId – ID for command in command buffer. Typically, the available value range is 1 - 15.
- config – Pointer to configuration structure. See to *lpadc_conv_command_config_t*.

void LPADC_GetDefaultConvCommandConfig(*lpadc_conv_command_config_t* *config)

Gets an available pre-defined settings for conversion command's configuration.

This function initializes the conversion command's configuration structure with an available settings. The default values are:

```
config->sampleScaleMode            = kLPADC_SampleFullScale;
config->channelBScaleMode          = kLPADC_SampleFullScale;
config->sampleChannelMode          = kLPADC_SampleChannelSingleEndSideA;
config->channelNumber              = 0U;
config->channelBNumber             = 0U;
config->chainedNextCommandNumber   = 0U;
config->enableAutoChannelIncrement = false;
config->loopCount                  = 0U;
config->hardwareAverageMode        = kLPADC_HardwareAverageCount1;
config->sampleTimeMode             = kLPADC_SampleTimeADCK3;
config->hardwareCompareMode        = kLPADC_HardwareCompareDisabled;
config->hardwareCompareValueHigh   = 0U;
config->hardwareCompareValueLow    = 0U;
```

(continues on next page)

(continued from previous page)

```
config->conversionResolutionMode   = kLPADC_ConversionResolutionStandard;
config->enableWaitTrigger          = false;
config->enableChannelB             = false;
```

Parameters

- config – Pointer to configuration structure.

void LPADC_EnableCalibration(ADC_Type *base, bool enable)

Enable the calibration function.

When CALOFS is set, the ADC is configured to perform a calibration function anytime the ADC executes a conversion. Any channel selected is ignored and the value returned in the RESFIFO is a signed value between -31 and 31. -32 is not a valid and is never a returned value. Software should copy the lower 6- bits of the conversion result stored in the RESFIFO after a completed calibration conversion to the OFSTRIM field. The OFSTRIM field is used in normal operation for offset correction.

Parameters

- base – LPADC peripheral base address.
- enable – switcher to the calibration function.

static inline void LPADC_SetOffsetValue(ADC_Type *base, uint32_t value)

Set proper offset value to trim ADC.

To minimize the offset during normal operation, software should read the conversion result from the RESFIFO calibration operation and write the lower 6 bits to the OFSTRIM register.

Parameters

- base – LPADC peripheral base address.
- value – Setting offset value.

status_t LPADC_DoAutoCalibration(ADC_Type *base)

Do auto calibration.

Calibration function should be executed before using converter in application. It used the software trigger and a dummy conversion, get the offset and write them into the OFSTRIM register. It called some of functional API including:

- LPADC_EnableCalibration(...)
- LPADC_SetOffsetValue(...)
- LPADC_SetConvCommandConfig(...)
- LPADC_SetConvTriggerConfig(...)

Parameters

- base – LPADC peripheral base address.
- base – LPADC peripheral base address.

Return values

- kStatus_Success – Successfully configured.
- kStatus_Timeout – Timeout occurs while waiting completion.

static inline void LPADC_SetOffsetValue(ADC_Type *base, int16_t value)

Set trim value for offset.

Note: For 16-bit conversions, each increment is 1/2 LSB resulting in a programmable offset range of -256 LSB to 255.5 LSB; For 12-bit conversions, each increment is 1/32 LSB resulting in a programmable offset range of -16 LSB to 15.96875 LSB.

Parameters

- base – LPADC peripheral base address.
- value – Offset trim value, is a 10-bit signed value between -512 and 511.

static inline void LPADC_GetOffsetValue(ADC_Type *base, int16_t *pValue)

Get trim value of offset.

Parameters

- base – LPADC peripheral base address.
- pValue – Pointer to the variable in type of int16_t to store offset value.

static inline void LPADC_EnableOffsetCalibration(ADC_Type *base, bool enable)

Enable the offset calibration function.

Parameters

- base – LPADC peripheral base address.
- enable – switcher to the calibration function.

static inline void LPADC_SetOffsetCalibrationMode(ADC_Type *base, *lpadc_offset_calibration_mode_t* mode)

Set offset calibration mode.

Parameters

- base – LPADC peripheral base address.
- mode – set offset calibration mode.see to *lpadc_offset_calibration_mode_t* .

status_t LPADC_DoOffsetCalibration(ADC_Type *base)

Do offset calibration.

Parameters

- base – LPADC peripheral base address.

Return values

- kStatus_Success – Successfully configured.
- kStatus_Timeout – Timeout occurs while waiting completion.

void LPADC_PrepareAutoCalibration(ADC_Type *base)

Prepare auto calibration, LPADC_FinishAutoCalibration has to be called before using the LPADC. LPADC_DoAutoCalibration has been split in two API to avoid to be stuck too long in the function.

Parameters

- base – LPADC peripheral base address.

status_t LPADC_FinishAutoCalibration(ADC_Type *base)

Finish auto calibration start with LPADC_PrepareAutoCalibration.

Note: This feature is used for LPADC with CTRL[CALOFSMODE].

Parameters

- base – LPADC peripheral base address.

Return values

- kStatus_Success – Successfully configured.
- kStatus_Timeout – Timeout occurs while waiting completion.

void LPADC_GetCalibrationValue(ADC_Type *base, *lpadc_calibration_value_t* *ptrCalibrationValue)

Get calibration value into the memory which is defined by invoker.

Note: Please note the ADC will be disabled temporary.

Note: This function should be used after finish calibration.

Parameters

- base – LPADC peripheral base address.
- ptrCalibrationValue – Pointer to *lpadc_calibration_value_t* structure, this memory block should be always powered on even in low power modes.

status_t LPADC_SetCalibrationValue(ADC_Type *base, const *lpadc_calibration_value_t* *ptrCalibrationValue)

Set calibration value into ADC calibration registers.

Note: Please note the ADC will be disabled temporary.

Parameters

- base – LPADC peripheral base address.
- ptrCalibrationValue – Pointer to *lpadc_calibration_value_t* structure which contains ADC's calibration value.

Return values

- kStatus_Success – Successfully configured.
- kStatus_Timeout – Timeout occurs while waiting completion.

static inline void LPADC_RequestHighSpeedModeTrim(ADC_Type *base)

Request high speed mode trim calculation.

Parameters

- base – LPADC peripheral base address.

static inline int8_t LPADC_GetHighSpeedTrimValue(ADC_Type *base)

Get high speed mode trim value, the result is a 5-bit signed value between -16 and 15.

Note: The high speed mode trim value is used to minimize offset for high speed conversion.

Parameters

- base – LPADC peripheral base address.

Returns

The calculated high speed mode trim value.

static inline void LPADC_SetHighSpeedTrimValue(ADC_Type *base, int8_t trimValue)

Set high speed mode trim value.

Note: If is possible to set the trim value manually, but it is recommended to use the LPADC_RequestHighSpeedModeTrim.

Parameters

- base – LPADC peripheral base address.
- trimValue – The trim value to be set.

static inline void LPADC_EnableHighSpeedConversionMode(ADC_Type *base, bool enable)

Enable/disable high speed conversion mode, if enabled conversions complete 2 or 3 ADCK cycles sooner compared to conversion cycle counts when high speed mode is disabled.

Parameters

- base – LPADC peripheral base address.
- enable – Used to enable/disable high speed conversion mode:
  - **true** Enable high speed conversion mode;
  - **false** Disable high speed conversion mode.

static inline void LPADC_EnableExtraCycle(ADC_Type *base, bool enable)

Enable/disable an additional ADCK cycle to conversion.

Parameters

- base – LPADC peripheral base address.
- enable – Used to enable/disable an additional ADCK cycle to conversion:
  - **true** Enable an additional ADCK cycle to conversion;
  - **false** Disable an additional ADCK cycle to conversion.

static inline void LPADC_SetTuneValue(ADC_Type *base, *lpadc_tune_value_t* tuneValue)

Set tune value which provides some variability in how many cycles are needed to complete a conversion.

Parameters

- base – LPADC peripheral base address.
- tuneValue – The tune value to be set, please refer to *lpadc_tune_value_t*.

static inline *lpadc_tune_value_t* LPADC_GetTuneValue(ADC_Type *base)

Get tune value which provides some variability in how many cycles are needed to complete a conversion.

Parameters

- base – LPADC peripheral base address.

Returns

The tune value, please refer to *lpadc_tune_value_t*.

static inline void LPADC_EnableJustifiedLeft(ADC_Type *base, bool enable)

Enable/disable left-justify format in 12-bit single-end mode.

Parameters

- base – LPADC peripheral base address.
- enable – Used to enable/disable left-justify format in 12-bit single-end mode:
  - **true** Enable left-justify format in 12-bit single-end mode;
  - **false** Disable left-justify format in 12-bit single-end mode.

FSL_LPADC_DRIVER_VERSION

LPADC driver version 2.10.3.

struct lpadc_config_t

#include <fsl_lpadc.h> LPADC global configuration.

This structure would used to keep the settings for initialization.

Public Members

bool enableInternalClock

Enables the internally generated clock source. The clock source is used in clock selection logic at the chip level and is optionally used for the ADC clock source.

bool enableVref1LowVoltage

If voltage reference option1 input is below 1.8V, it should be "true". If voltage reference option1 input is above 1.8V, it should be "false".

bool enableInDozeMode

Control system transition to Stop and Wait power modes while ADC is converting. When enabled in Doze mode, immediate entries to Wait or Stop are allowed. When disabled, the ADC will wait for the current averaging iteration/FIFO storage to complete before acknowledging stop or wait mode entry.

lpadc_conversion_average_mode_t conversionAverageMode

Auto-Calibration Averages.

bool enableAnalogPreliminary

ADC analog circuits are pre-enabled and ready to execute conversions without startup delays(at the cost of higher DC current consumption).

uint32_t powerUpDelay

When the analog circuits are not pre-enabled, the ADC analog circuits are only powered while the ADC is active and there is a counted delay defined by this field after an initial trigger transitions the ADC from its Idle state to allow time for the analog circuits to stabilize. The startup delay count of (powerUpDelay * 4) ADCK cycles must result in a longer delay than the analog startup time.

lpadc_reference_voltage_source_t referenceVoltageSource

Selects the voltage reference high used for conversions.

lpadc_power_level_mode_t powerLevelMode

Power Configuration Selection.

lpadc_trigger_priority_policy_t triggerPriorityPolicy

Control how higher priority triggers are handled, see to lpadc_trigger_priority_policy_t.

bool enableConvPause

Enables the ADC pausing function. When enabled, a programmable delay is inserted during command execution sequencing between LOOP iterations, between commands in a sequence, and between conversions when command is executing in "Compare Until True" configuration.

uint32_t convPauseDelay

Controls the duration of pausing during command execution sequencing. The pause delay is a count of (convPauseDelay*4) ADCK cycles. Only available when ADC pausing function is enabled. The available value range is in 9-bit.

uint32_t FIFOWatermark

FIFOWatermark is a programmable threshold setting. When the number of datawords stored in the ADC Result FIFO is greater than the value in this field, the ready flag would be asserted to indicate stored data has reached the programmable threshold.

struct lpadc_conv_command_config_t

#include <fsl_lpadc.h> Define structure to keep the configuration for conversion command.

Public Members

lpadc_sample_scale_mode_t sampleScaleMode

Sample scale mode.

lpadc_sample_scale_mode_t channelBScaleMode

Alternate channe B Scale mode.

lpadc_sample_channel_mode_t sampleChannelMode

Channel sample mode.

uint32_t channelNumber

Channel number, select the channel or channel pair.

uint32_t channelBNumber

Alternate Channel B number, select the channel.

uint32_t chainedNextCommandNumber

Selects the next command to be executed after this command completes. 1-15 is available, 0 is to terminate the chain after this command.

bool enableAutoChannelIncrement

Loop with increment: when disabled, the "loopCount" field selects the number of times the selected channel is converted consecutively; when enabled, the "loopCount" field defines how many consecutive channels are converted as part of the command execution.

uint32_t loopCount

Selects how many times this command executes before finish and transition to the next command or Idle state. Command executes LOOP+1 times. 0-15 is available.

lpadc_hardware_average_mode_t hardwareAverageMode

Hardware average selection.

lpadc_sample_time_mode_t sampleTimeMode

Sample time selection.

lpadc_hardware_compare_mode_t hardwareCompareMode

Hardware compare selection.

uint32_t hardwareCompareValueHigh

Compare Value High. The available value range is in 16-bit.

uint32_t hardwareCompareValueLow

Compare Value Low. The available value range is in 16-bit.

lpadc_conversion_resolution_mode_t conversionResolutionMode

Conversion resolution mode.

bool enableWaitTrigger

Wait for trigger assertion before execution: when disabled, this command will be automatically executed; when enabled, the active trigger must be asserted again before executing this command.

struct lpadc_conv_trigger_config_t

#include <fsl_lpadc.h> Define structure to keep the configuration for conversion trigger.

Public Members

uint32_t targetCommandId

Select the command from command buffer to execute upon detect of the associated trigger event.

uint32_t delayPower

Select the trigger delay duration to wait at the start of servicing a trigger event. When this field is clear, then no delay is incurred. When this field is set to a non-zero value, the duration for the delay is 2^delayPower ADCK cycles. The available value range is 4-bit.

uint32_t priority

Sets the priority of the associated trigger source. If two or more triggers have the same priority level setting, the lower order trigger event has the higher priority. The lower value for this field is for the higher priority, the available value range is 1-bit.

bool enableHardwareTrigger

Enable hardware trigger source to initiate conversion on the rising edge of the input trigger source or not. THe software trigger is always available.

struct lpadc_conv_result_t

#include <fsl_lpadc.h> Define the structure to keep the conversion result.

Public Members

uint32_t commandIdSource

Indicate the command buffer being executed that generated this result.

uint32_t loopCountIndex

Indicate the loop count value during command execution that generated this result.

uint32_t triggerIdSource

Indicate the trigger source that initiated a conversion and generated this result.

uint16_t convValue

Data result.

struct _lpadc_calibration_value

#include <fsl_lpadc.h> A structure of calibration value.

2.37 LPCMP: Low Power Analog Comparator Driver

void LPCMP_Init(LPCMP_Type *base, const *lpcmp_config_t* *config)

Initialize the LPCMP.

This function initializes the LPCMP module. The operations included are:

- Enabling the clock for LPCMP module.
- Configuring the comparator.
- Enabling the LPCMP module. Note: For some devices, multiple LPCMP instance share the same clock gate. In this case, to enable the clock for any instance enables all the LPCMPs. Check the chip reference manual for the clock assignment of the LPCMP.

Parameters

- base – LPCMP peripheral base address.
- config – Pointer to "lpcmp_config_t" structure.

void LPCMP_Deinit(LPCMP_Type *base)

De-initializes the LPCMP module.

This function de-initializes the LPCMP module. The operations included are:

- Disabling the LPCMP module.
- Disabling the clock for LPCMP module.

This function disables the clock for the LPCMP. Note: For some devices, multiple LPCMP instance shares the same clock gate. In this case, before disabling the clock for the LPCMP, ensure that all the LPCMP instances are not used.

Parameters

- base – LPCMP peripheral base address.

void LPCMP_GetDefaultConfig(*lpcmp_config_t* *config)

Gets an available pre-defined settings for the comparator's configuration.

This function initializes the comparator configuration structure to these default values:

```
config->enableStopMode      = false;
config->enableOutputPin     = false;
config->enableCmpToDacLink  = false;
config->useUnfilteredOutput = false;
config->enableInvertOutput  = false;
config->hysteresisMode      = kLPCMP_HysteresisLevel0;
config->powerMode           = kLPCMP_LowSpeedPowerMode;
config->functionalSourceClock = kLPCMP_FunctionalClockSource0;
config->plusInputSrc        = kLPCMP_PlusInputSrcMux;
config->minusInputSrc       = kLPCMP_MinusInputSrcMux;
```

Parameters

- config – Pointer to "lpcmp_config_t" structure.

static inline void LPCMP_Enable(LPCMP_Type *base, bool enable)

Enable/Disable LPCMP module.

Parameters

- base – LPCMP peripheral base address.
- enable – "true" means enable the module, and "false" means disable the module.

void LPCMP_SetInputChannels(LPCMP_Type *base, uint32_t positiveChannel, uint32_t negativeChannel)

Select the input channels for LPCMP. This function determines which input is selected for the negative and positive mux.

Parameters

- base – LPCMP peripheral base address.
- positiveChannel – Positive side input channel number. Available range is 0-7.
- negativeChannel – Negative side input channel number. Available range is 0-7.

static inline void LPCMP_EnableDMA(LPCMP_Type *base, bool enable)

Enables/disables the DMA request for rising/falling events. Normally, the LPCMP generates a CPU interrupt if there is a rising/falling event. When DMA support is enabled and the rising/falling interrupt is enabled , the rising/falling event forces a DMA transfer request rather than a CPU interrupt instead.

Parameters

- base – LPCMP peripheral base address.
- enable – "true" means enable DMA support, and "false" means disable DMA support.

void LPCMP_SetFilterConfig(LPCMP_Type *base, const *lpcmp_filter_config_t* *config)

Configures the filter.

Parameters

- base – LPCMP peripheral base address.
- config – Pointer to "lpcmp_filter_config_t" structure.

void LPCMP_SetDACConfig(LPCMP_Type *base, const *lpcmp_dac_config_t* *config)

Configure the internal DAC module.

Parameters

- base – LPCMP peripheral base address.
- config – Pointer to "lpcmp_dac_config_t" structure. If config is "NULL", disable internal DAC.

static inline void LPCMP_EnableInterrupts(LPCMP_Type *base, uint32_t mask)

Enable the interrupts.

Parameters

- base – LPCMP peripheral base address.
- mask – Mask value for interrupts. See "_lpcmp_interrupt_enable".

static inline void LPCMP_DisableInterrupts(LPCMP_Type *base, uint32_t mask)

Disable the interrupts.

Parameters

- base – LPCMP peripheral base address.
- mask – Mask value for interrupts. See "_lpcmp_interrupt_enable".

static inline uint32_t LPCMP_GetStatusFlags(LPCMP_Type *base)

Get the LPCMP status flags.

Parameters

- base – LPCMP peripheral base address.

Returns

Mask value for the asserted flags. See "_lpcmp_status_flags".

static inline void LPCMP_ClearStatusFlags(LPCMP_Type *base, uint32_t mask)

Clear the LPCMP status flags.

Parameters

- base – LPCMP peripheral base address.
- mask – Mask value for the flags. See "_lpcmp_status_flags".

static inline void LPCMP_EnableWindowMode(LPCMP_Type *base, bool enable)

Enable/Disable window mode.When any windowed mode is active, COUTA is clocked by the bus clock whenever WINDOW = 1. The last latched value is held when WINDOW = 0. The optionally inverted comparator output COUT_RAW is sampled on every bus clock when WINDOW=1 to generate COUTA.

Parameters

- base – LPCMP peripheral base address.
- enable – "true" means enable window mode, and "false" means disable window mode.

void LPCMP_SetWindowControl(LPCMP_Type *base, const *lpcmp_window_control_config_t* *config)

Configure the window control, users can use this API to implement operations on the window, such as inverting the window signal, setting the window closing event(only valid in windowing mode), and setting the COUTA signal after the window is closed(only valid in windowing mode).

Parameters

- base – LPCMP peripheral base address.
- config – Pointer "lpcmp_window_control_config_t" structure.

void LPCMP_SetRoundRobinConfig(LPCMP_Type *base, const *lpcmp_roundrobin_config_t* *config)

Configure the roundrobin mode.

Parameters

- base – LPCMP peripheral base address.
- config – Pointer "lpcmp_roundrobin_config_t" structure.

static inline void LPCMP_EnableRoundRobinMode(LPCMP_Type *base, bool enable)

Enable/Disable roundrobin mode.

Parameters

- base – LPCMP peripheral base address.
- enable – "true" means enable roundrobin mode, and "false" means disable roundrobin mode.

void LPCMP_SetRoundRobinInternalTimer(LPCMP_Type *base, uint32_t value)

brief Configure the roundrobin internal timer reload value.

param base LPCMP peripheral base address. param value RoundRobin internal timer reload value, allowed range:0x0UL-0xFFFFFFFUL.

static inline void LPCMP_EnableRoundRobinInternalTimer(LPCMP_Type *base, bool enable)

Enable/Disable roundrobin internal timer, note that this function is only valid when using the internal trigger source.

Parameters

- base – LPCMP peripheral base address.
- enable – "true" means enable roundrobin internal timer, and "false" means disable roundrobin internal timer.

static inline void LPCMP_SetPreSetValue(LPCMP_Type *base, uint8_t mask)

Set preset value for all channels, users can set all channels' preset vaule through this API, for example, if the mask set to 0x03U means channel0 and channel2's preset value set to 1U and other channels' preset value set to 0U.

Parameters

- base – LPCMP peripheral base address.
- mask – Mask of channel index.

static inline uint8_t LPCMP_GetComparisonResult(LPCMP_Type *base)

Get comparison results for all channels, users can get all channels' comparison results through this API.

Parameters

- base – LPCMP peripheral base address.

Returns

return All channels' comparison result.

static inline void LPCMP_ClearInputChangedFlags(LPCMP_Type *base, uint8_t mask)

Clear input changed flags for single channel or multiple channels, users can clear input changed flag of a single channel or multiple channels through this API, for example, if the mask set to 0x03U means clear channel0 and channel2's input changed flags.

Parameters

- base – LPCMP peripheral base address.
- mask – Mask of channel index.

static inline uint8_t LPCMP_GetInputChangedFlags(LPCMP_Type *base)

Get input changed flags for all channels, Users can get all channels' input changed flags through this API.

Parameters

- base – LPCMP peripheral base address.

Returns

return All channels' changed flag.

FSL_LPCMP_DRIVER_VERSION

LPCMP driver version 2.3.2.

enum _lpcmp_status_flags

LPCMP status falgs mask.

Values:

enumerator kLPCMP_OutputRisingEventFlag

Rising-edge on the comparison output has occurred.

enumerator kLPCMP_OutputFallingEventFlag

Falling-edge on the comparison output has occurred.

enumerator kLPCMP_OutputRoundRobinEventFlag

Detects when any channel's last comparison result is different from the pre-set value in trigger mode.

enumerator kLPCMP_OutputAssertEventFlag

Return the current value of the analog comparator output. The flag does not support W1C.

enum _lpcmp_interrupt_enable

LPCMP interrupt enable/disable mask.

Values:

enumerator kLPCMP_OutputRisingInterruptEnable

Comparator interrupt enable rising.

enumerator kLPCMP_OutputFallingInterruptEnable

Comparator interrupt enable falling.

enumerator kLPCMP_RoundRobinInterruptEnable

Comparator round robin mode interrupt occurred when the comparison result changes for a given channel.

enum _lpcmp_hysteresis_mode

LPCMP hysteresis mode. See chip data sheet to get the actual hystersis value with each level.

Values:

enumerator kLPCMP_HysteresisLevel0

The hard block output has level 0 hysteresis internally.

enumerator kLPCMP_HysteresisLevel1

The hard block output has level 1 hysteresis internally.

enumerator kLPCMP_HysteresisLevel2

The hard block output has level 2 hysteresis internally.

enumerator kLPCMP_HysteresisLevel3

The hard block output has level 3 hysteresis internally.

enum _lpcmp_power_mode

LPCMP nano mode.

Values:

enumerator kLPCMP_LowSpeedPowerMode

Low speed comparison mode is selected.

enumerator kLPCMP_HighSpeedPowerMode

High speed comparison mode is selected.

enumerator kLPCMP_NanoPowerMode

Nano power comparator is enabled.

enum _lpcmp_dac_reference_voltage_source

Internal DAC reference voltage source.

Values:

enumerator kLPCMP_VrefSourceVin1
vrefh_int is selected as resistor ladder network supply reference Vin.

enumerator kLPCMP_VrefSourceVin2
vrefh_ext is selected as resistor ladder network supply reference Vin.

enum _lpcmp_functional_source_clock
LPCMP functional mode clock source selection.

Note: In different devices, the functional mode clock source selection is different, please refer to specific device Reference Manual for details.

Values:

enumerator kLPCMP_FunctionalClockSource0
Select functional mode clock source0.

enumerator kLPCMP_FunctionalClockSource1
Select functional mode clock source1.

enumerator kLPCMP_FunctionalClockSource2
Select functional mode clock source2.

enumerator kLPCMP_FunctionalClockSource3
Select functional mode clock source3.

enum _lpcmp_couta_signal
Set the COUTA signal value when the window is closed.

Values:

enumerator kLPCMP_COUTASignalNoSet
NO set the COUTA signal value when the window is closed.

enumerator kLPCMP_COUTASignalLow
Set COUTA signal low(0) when the window is closed.

enumerator kLPCMP_COUTASignalHigh
Set COUTA signal high(1) when the window is closed.

enum _lpcmp_close_window_event
Set COUT event, which can close the active window in window mode.

Values:

enumerator kLPCMP_CLoseWindowEventNoSet
No Set COUT event, which can close the active window in window mode.

enumerator kLPCMP_CloseWindowEventRisingEdge
Set rising edge COUT signal as COUT event.

enumerator kLPCMP_CloseWindowEventFallingEdge
Set falling edge COUT signal as COUT event.

enumerator kLPCMP_CLoseWindowEventBothEdge
Set both rising and falling edge COUT signal as COUT event.

enum _lpcmp_roundrobin_fixedmuxport
LPCMP round robin mode fixed mux port.

Values:

enumerator kLPCMP_FixedPlusMuxPort
Fixed plus mux port.

enumerator kLPCMP_FixedMinusMuxPort
Fixed minus mux port.

enum _lpcmp_roundrobin_clock_source
LPCMP round robin mode clock source selection.

Note: In different devices,the round robin mode clock source selection is different, please refer to the specific device Reference Manual for details.

Values:

enumerator kLPCMP_RoundRobinClockSource0
Select roundrobin mode clock source0.

enumerator kLPCMP_RoundRobinClockSource1
Select roundrobin mode clock source1.

enumerator kLPCMP_RoundRobinClockSource2
Select roundrobin mode clock source2.

enumerator kLPCMP_RoundRobinClockSource3
Select roundrobin mode clock source3.

enum _lpcmp_roundrobin_trigger_source
LPCMP round robin mode trigger source.

Values:

enumerator kLPCMP_TriggerSourceExternally
Select external trigger source.

enumerator kLPCMP_TriggerSourceInternally
Select internal trigger source.

enum _lpcmp_plus_input_src
LPCMP plus input source.

Values:

enumerator kLPCMP_PlusInputSrcDac

LPCMP plus input source from the internal 8-bit DAC output.

enumerator kLPCMP_PlusInputSrcMux

LPCMP plus input source from the analog 8-1 mux.

enum _lpcmp_minus_input_src

LPCMP minus input source.

Values:

enumerator kLPCMP_MinusInputSrcDac

LPCMP minus input source from the internal 8-bit DAC output.

enumerator kLPCMP_MinusInputSrcMux

LPCMP minus input source from the analog 8-1 mux.

typedef enum *_lpcmp_hysteresis_mode* lpcmp_hysteresis_mode_t

LPCMP hysteresis mode. See chip data sheet to get the actual hystersis value with each level.

typedef enum *_lpcmp_power_mode* lpcmp_power_mode_t

LPCMP nano mode.

typedef enum *_lpcmp_dac_reference_voltage_source* lpcmp_dac_reference_voltage_source_t

Internal DAC reference voltage source.

typedef enum *_lpcmp_functional_source_clock* lpcmp_functional_source_clock_t

LPCMP functional mode clock source selection.

Note: In different devices, the functional mode clock source selection is different, please refer to specific device Reference Manual for details.

typedef enum *_lpcmp_couta_signal* lpcmp_couta_signal_t

Set the COUTA signal value when the window is closed.

typedef enum *_lpcmp_close_window_event* lpcmp_close_window_event_t

Set COUT event, which can close the active window in window mode.

typedef enum *_lpcmp_roundrobin_fixedmuxport* lpcmp_roundrobin_fixedmuxport_t

LPCMP round robin mode fixed mux port.

typedef enum *_lpcmp_roundrobin_clock_source* lpcmp_roundrobin_clock_source_t

LPCMP round robin mode clock source selection.

Note: In different devices,the round robin mode clock source selection is different, please refer to the specific device Reference Manual for details.

typedef enum *_lpcmp_roundrobin_trigger_source* lpcmp_roundrobin_trigger_source_t

LPCMP round robin mode trigger source.

typedef struct *_lpcmp_filter_config* lpcmp_filter_config_t
Configure the filter.

typedef enum *_lpcmp_plus_input_src* lpcmp_plus_input_src_t
LPCMP plus input source.

typedef enum *_lpcmp_minus_input_src* lpcmp_minus_input_src_t
LPCMP minus input source.

typedef struct *_lpcmp_dac_config* lpcmp_dac_config_t
configure the internal DAC.

typedef struct *_lpcmp_config* lpcmp_config_t
Configures the comparator.

typedef struct *_lpcmp_window_control_config* lpcmp_window_control_config_t
Configure the window mode control.

typedef struct *_lpcmp_roundrobin_config* lpcmp_roundrobin_config_t
Configure the round robin mode.

LPCMP_CCR1_COUTA_CFG_MASK

LPCMP_CCR1_COUTA_CFG_SHIFT

LPCMP_CCR1_COUTA_CFG(x)

LPCMP_CCR1_EVT_SEL_CFG_MASK

LPCMP_CCR1_EVT_SEL_CFG_SHIFT

LPCMP_CCR1_EVT_SEL_CFG(x)

struct _lpcmp_filter_config
#include <fsl_lpcmp.h> Configure the filter.

Public Members

bool enableSample
Decide whether to use the external SAMPLE as a sampling clock input.

uint8_t filterSampleCount
Filter Sample Count. Available range is 1-7; 0 disables the filter.

uint8_t filterSamplePeriod
Filter Sample Period. The divider to the bus clock. Available range is 0-255. The sampling clock must be at least 4 times slower than the system clock to the comparator. So if enableSample is “false”, filterSamplePeriod should be set greater than 4.

struct __lpcmp_dac_config

#include <fsl_lpcmp.h> configure the internal DAC.

Public Members

bool enableLowPowerMode

Decide whether to enable DAC low power mode.

lpcmp_dac_reference_voltage_source_t referenceVoltageSource

Internal DAC supply voltage reference source.

uint8_t DACValue

Value for the DAC Output Voltage. Different devices has different available range, for specific values, please refer to the reference manual.

struct __lpcmp_config

#include <fsl_lpcmp.h> Configures the comparator.

Public Members

bool enableStopMode

Decide whether to enable the comparator when in STOP modes.

bool enableCmpToDacLink

Controls the link from the CMP enable to the DAC enable.

bool enableOutputPin

Decide whether to enable the comparator is available in selected pin.

bool useUnfilteredOutput

Decide whether to use unfiltered output.

bool enableInvertOutput

Decide whether to inverts the comparator output.

lpcmp_hysteresis_mode_t hysteresisMode

LPCMP hysteresis mode.

lpcmp_power_mode_t powerMode

LPCMP power mode.

lpcmp_functional_source_clock_t functionalSourceClock

Select LPCMP functional mode clock source.

lpcmp_plus_input_src_t plusInputSrc

Select LPCMP plus input source.

lpcmp_minus_input_src_t minusInputSrc

Select LPCMP minus input source.

struct __lpcmp__window__control__config

#include <fsl_lpcmp.h> Configure the window mode control.

Public Members

bool enableInvertWindowSignal

True: enable invert window signal, False: disable invert window signal.

lpcmp_couta_signal_t COUTASignal

Decide whether to define the COUTA signal value when the window is closed.

lpcmp_close_window_event_t closeWindowEvent

Decide whether to select COUT event signal edge defines a COUT event to close window.

struct __lpcmp__roundrobin__config

#include <fsl_lpcmp.h> Configure the round robin mode.

Public Members

uint8_t initDelayModules

Comparator and DAC initialization delay modulus, See Reference Manual and DataSheet for specific value.

uint8_t sampleClockNumbers

Specify the number of the round robin clock cycles(0~3) to wait after scanning the active channel before sampling the channel's comparison result.

uint8_t channelSampleNumbers

Specify the number of samples for one channel, note that channelSampleNumbers must not smaller than sampleTimeThreshhold.

uint8_t sampleTimeThreshhold

Specify that for one channel, when (sampleTimeThreshhold + 1) sample results are "1",the final result is "1", otherwise the final result is "0", note that the sampleTimeThreshhold must not be larger than channelSampleNumbers.

lpcmp_roundrobin_clock_source_t roundrobinClockSource

Decide which clock source to choose in round robin mode.

lpcmp_roundrobin_trigger_source_t roundrobinTriggerSource

Decide which trigger source to choose in round robin mode.

lpcmp_roundrobin_fixedmuxport_t fixedMuxPort

Decide which mux port to choose as fixed channel in round robin mode.

uint8_t fixedChannel

Indicate which channel of the fixed mux port is used in round robin mode.

uint8_t checkerChannelMask

Indicate which channel of the non-fixed mux port to check its voltage value in round robin mode, for example, if checkerChannelMask set to 0x11U means select channel 0 and channel 4 as checker channel.

2.38 LPI2C: Low Power Inter-Integrated Circuit Driver

void LPI2C_DriverIRQHandler(uint32_t instance)

LPI2C driver IRQ handler common entry.

This function provides the common IRQ request entry for LPI2C.

Parameters

- instance – LPI2C instance.

FSL_LPI2C_DRIVER_VERSION

LPI2C driver version.

LPI2C status return codes.

Values:

enumerator kStatus_LPI2C_Busy

The master is already performing a transfer.

enumerator kStatus_LPI2C_Idle

The slave driver is idle.

enumerator kStatus_LPI2C_Nak

The slave device sent a NAK in response to a byte.

enumerator kStatus_LPI2C_FifoError

FIFO under run or overrun.

enumerator kStatus_LPI2C_BitError

Transferred bit was not seen on the bus.

enumerator kStatus_LPI2C_ArbitrationLost

Arbitration lost error.

enumerator kStatus_LPI2C_PinLowTimeout

SCL or SDA were held low longer than the timeout.

enumerator kStatus_LPI2C_NoTransferInProgress

Attempt to abort a transfer when one is not in progress.

enumerator kStatus_LPI2C_DmaRequestFail

DMA request failed.

enumerator kStatus_LPI2C_Timeout

Timeout polling status flags.

IRQn_Type const kLpi2cMasterIrqs[]

Array to map LPI2C instance number to IRQ number, used internally for LPI2C master interrupt and EDMA transactional APIs.

IRQn_Type const kLpi2cSlaveIrqs[]

lpi2c_master_isr_t s_lpi2cMasterIsr

Pointer to master IRQ handler for each instance, used internally for LPI2C master interrupt and EDMA transactional APIs.

void *s_lpi2cMasterHandle[]

Pointers to master handles for each instance, used internally for LPI2C master interrupt and EDMA transactional APIs.

uint32_t LPI2C_GetInstance(LPI2C_Type *base)

Returns an instance number given a base address.

If an invalid base address is passed, debug builds will assert. Release builds will just return instance number 0.

Parameters

- base – The LPI2C peripheral base address.

Returns

LPI2C instance number starting from 0.

I2C_RETRY_TIMES

Retry times for waiting flag.

2.39 LPI2C Master Driver

void LPI2C_MasterGetDefaultConfig(*lpi2c_master_config_t* *masterConfig)

Provides a default configuration for the LPI2C master peripheral.

This function provides the following default configuration for the LPI2C master peripheral:

```
masterConfig->enableMaster            = true;
masterConfig->debugEnable             = false;
masterConfig->ignoreAck               = false;
masterConfig->pinConfig               = kLPI2C_2PinOpenDrain;
masterConfig->baudRate_Hz             = 100000U;
masterConfig->busIdleTimeout_ns       = 0;
masterConfig->pinLowTimeout_ns        = 0;
masterConfig->sdaGlitchFilterWidth_ns = 0;
masterConfig->sclGlitchFilterWidth_ns = 0;
masterConfig->hostRequest.enable      = false;
masterConfig->hostRequest.source      = kLPI2C_HostRequestExternalPin;
masterConfig->hostRequest.polarity    = kLPI2C_HostRequestPinActiveHigh;
```

After calling this function, you can override any settings in order to customize the configuration, prior to initializing the master driver with *LPI2C_MasterInit()*.

Parameters

- masterConfig – **[out]** User provided configuration structure for default values. Refer to *lpi2c_master_config_t*.

void LPI2C_MasterInit(LPI2C_Type *base, const *lpi2c_master_config_t* *masterConfig, uint32_t sourceClock_Hz)

Initializes the LPI2C master peripheral.

This function enables the peripheral clock and initializes the LPI2C master peripheral as described by the user provided configuration. A software reset is performed prior to configuration.

Parameters

- base – The LPI2C peripheral base address.
- masterConfig – User provided peripheral configuration. Use *LPI2C_MasterGetDefaultConfig()* to get a set of defaults that you can override.
- sourceClock_Hz – Frequency in Hertz of the LPI2C functional clock. Used to calculate the baud rate divisors, filter widths, and timeout periods.

void LPI2C_MasterDeinit(LPI2C_Type *base)

Deinitializes the LPI2C master peripheral.

This function disables the LPI2C master peripheral and gates the clock. It also performs a software reset to restore the peripheral to reset conditions.

Parameters

- base – The LPI2C peripheral base address.

void LPI2C_MasterConfigureDataMatch(LPI2C_Type *base, const *lpi2c_data_match_config_t* *matchConfig)

Configures LPI2C master data match feature.

Parameters

- base – The LPI2C peripheral base address.
- matchConfig – Settings for the data match feature.

status_t LPI2C_MasterCheckAndClearError(LPI2C_Type *base, uint32_t status)

Convert provided flags to status code, and clear any errors if present.

Parameters

- base – The LPI2C peripheral base address.
- status – Current status flags value that will be checked.

Return values

- kStatus_Success –
- kStatus_LPI2C_PinLowTimeout –
- kStatus_LPI2C_ArbitrationLost –
- kStatus_LPI2C_Nak –
- kStatus_LPI2C_FifoError –

status_t LPI2C_CheckForBusyBus(LPI2C_Type *base)

Make sure the bus isn't already busy.

A busy bus is allowed if we are the one driving it.

Parameters

- base – The LPI2C peripheral base address.

Return values

- kStatus_Success –
- kStatus_LPI2C_Busy –

static inline void LPI2C_MasterReset(LPI2C_Type *base)

Performs a software reset.

Restores the LPI2C master peripheral to reset conditions.

Parameters

- base – The LPI2C peripheral base address.

static inline void LPI2C_MasterEnable(LPI2C_Type *base, bool enable)

Enables or disables the LPI2C module as master.

Parameters

- base – The LPI2C peripheral base address.
- enable – Pass true to enable or false to disable the specified LPI2C as master.

static inline uint32_t LPI2C_MasterGetStatusFlags(LPI2C_Type *base)

Gets the LPI2C master status flags.

A bit mask with the state of all LPI2C master status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

See also:

_lpi2c_master_flags

Parameters

- base – The LPI2C peripheral base address.

Returns

State of the status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

static inline void LPI2C_MasterClearStatusFlags(LPI2C_Type *base, uint32_t statusMask)

Clears the LPI2C master status flag state.

The following status register flags can be cleared:

- *kLPI2C_MasterEndOfPacketFlag*
- *kLPI2C_MasterStopDetectFlag*
- *kLPI2C_MasterNackDetectFlag*
- *kLPI2C_MasterArbitrationLostFlag*
- *kLPI2C_MasterFifoErrFlag*
- *kLPI2C_MasterPinLowTimeoutFlag*

- *kLPI2C_MasterDataMatchFlag*

Attempts to clear other flags has no effect.

See also:

_lpi2c_master_flags.

Parameters

- base – The LPI2C peripheral base address.
- statusMask – A bitmask of status flags that are to be cleared. The mask is composed of _lpi2c_master_flags enumerators OR'd together. You may pass the result of a previous call to *LPI2C_MasterGetStatusFlags()*.

static inline void LPI2C_MasterEnableInterrupts(LPI2C_Type *base, uint32_t interruptMask)

Enables the LPI2C master interrupt requests.

All flags except *kLPI2C_MasterBusyFlag* and *kLPI2C_MasterBusBusyFlag* can be enabled as interrupts.

Parameters

- base – The LPI2C peripheral base address.
- interruptMask – Bit mask of interrupts to enable. See _lpi2c_master_flags for the set of constants that should be OR'd together to form the bit mask.

static inline void LPI2C_MasterDisableInterrupts(LPI2C_Type *base, uint32_t interruptMask)

Disables the LPI2C master interrupt requests.

All flags except *kLPI2C_MasterBusyFlag* and *kLPI2C_MasterBusBusyFlag* can be enabled as interrupts.

Parameters

- base – The LPI2C peripheral base address.
- interruptMask – Bit mask of interrupts to disable. See _lpi2c_master_flags for the set of constants that should be OR'd together to form the bit mask.

static inline uint32_t LPI2C_MasterGetEnabledInterrupts(LPI2C_Type *base)

Returns the set of currently enabled LPI2C master interrupt requests.

Parameters

- base – The LPI2C peripheral base address.

Returns

A bitmask composed of _lpi2c_master_flags enumerators OR'd together to indicate the set of enabled interrupts.

static inline void LPI2C_MasterEnableDMA(LPI2C_Type *base, bool enableTx, bool enableRx)

Enables or disables LPI2C master DMA requests.

Parameters

- base – The LPI2C peripheral base address.
- enableTx – Enable flag for transmit DMA request. Pass true for enable, false for disable.
- enableRx – Enable flag for receive DMA request. Pass true for enable, false for disable.

static inline uint32_t LPI2C_MasterGetTxFifoAddress(LPI2C_Type *base)

Gets LPI2C master transmit data register address for DMA transfer.

Parameters

- base – The LPI2C peripheral base address.

Returns

The LPI2C Master Transmit Data Register address.

static inline uint32_t LPI2C_MasterGetRxFifoAddress(LPI2C_Type *base)

Gets LPI2C master receive data register address for DMA transfer.

Parameters

- base – The LPI2C peripheral base address.

Returns

The LPI2C Master Receive Data Register address.

static inline void LPI2C_MasterSetWatermarks(LPI2C_Type *base, size_t txWords, size_t rxWords)

Sets the watermarks for LPI2C master FIFOs.

Parameters

- base – The LPI2C peripheral base address.
- txWords – Transmit FIFO watermark value in words. The *kLPI2C_MasterTxReadyFlag* flag is set whenever the number of words in the transmit FIFO is equal or less than *txWords*. Writing a value equal or greater than the FIFO size is truncated.
- rxWords – Receive FIFO watermark value in words. The *kLPI2C_MasterRxReadyFlag* flag is set whenever the number of words in the receive FIFO is greater than *rxWords*. Writing a value equal or greater than the FIFO size is truncated.

static inline void LPI2C_MasterGetFifoCounts(LPI2C_Type *base, size_t *rxCount, size_t *txCount)

Gets the current number of words in the LPI2C master FIFOs.

Parameters

- base – The LPI2C peripheral base address.
- txCount – **[out]** Pointer through which the current number of words in the transmit FIFO is returned. Pass NULL if this value is not required.
- rxCount – **[out]** Pointer through which the current number of words in the receive FIFO is returned. Pass NULL if this value is not required.

void LPI2C_MasterSetBaudRate(LPI2C_Type *base, uint32_t sourceClock_Hz, uint32_t baudRate_Hz)

Sets the I2C bus frequency for master transactions.

The LPI2C master is automatically disabled and re-enabled as necessary to configure the baud rate. Do not call this function during a transfer, or the transfer is aborted.

Note: Please note that the second parameter is the clock frequency of LPI2C module, the third parameter means user configured bus baudrate, this implementation is different from other I2C drivers which use baudrate configuration as second parameter and source clock frequency as third parameter.

Parameters

- base – The LPI2C peripheral base address.
- sourceClock_Hz – LPI2C functional clock frequency in Hertz.
- baudRate_Hz – Requested bus frequency in Hertz.

static inline bool LPI2C_MasterGetBusIdleState(LPI2C_Type *base)

Returns whether the bus is idle.

Requires the master mode to be enabled.

Parameters

- base – The LPI2C peripheral base address.

Return values

- true – Bus is busy.
- false – Bus is idle.

status_t LPI2C_MasterStart(LPI2C_Type *base, uint8_t address, *lpi2c_direction_t* dir)

Sends a START signal and slave address on the I2C bus.

This function is used to initiate a new master mode transfer. First, the bus state is checked to ensure that another master is not occupying the bus. Then a START signal is transmitted, followed by the 7-bit address specified in the *address* parameter. Note that this function does not actually wait until the START and address are successfully sent on the bus before returning.

Parameters

- base – The LPI2C peripheral base address.
- address – 7-bit slave device address, in bits [6:0].
- dir – Master transfer direction, either *kLPI2C_Read* or *kLPI2C_Write*. This parameter is used to set the R/w bit (bit 0) in the transmitted slave address.

Return values

- kStatus_Success – START signal and address were successfully enqueued in the transmit FIFO.
- kStatus_LPI2C_Busy – Another master is currently utilizing the bus.

static inline *status_t* LPI2C_MasterRepeatedStart(LPI2C_Type *base, uint8_t address, *lpi2c_direction_t* dir)

Sends a repeated START signal and slave address on the I2C bus.

This function is used to send a Repeated START signal when a transfer is already in progress. Like *LPI2C_MasterStart()*, it also sends the specified 7-bit address.

Note: This function exists primarily to maintain compatible APIs between LPI2C and I2C drivers, as well as to better document the intent of code that uses these APIs.

Parameters

- base – The LPI2C peripheral base address.
- address – 7-bit slave device address, in bits [6:0].
- dir – Master transfer direction, either *kLPI2C_Read* or *kLPI2C_Write*. This parameter is used to set the R/w bit (bit 0) in the transmitted slave address.

Return values

- kStatus_Success – Repeated START signal and address were successfully enqueued in the transmit FIFO.
- kStatus_LPI2C_Busy – Another master is currently utilizing the bus.

status_t LPI2C_MasterSend(LPI2C_Type *base, void *txBuff, size_t txSize)

Performs a polling send transfer on the I2C bus.

Sends up to *txSize* number of bytes to the previously addressed slave device. The slave may reply with a NAK to any byte in order to terminate the transfer early. If this happens, this function returns *kStatus_LPI2C_Nak*.

Parameters

- base – The LPI2C peripheral base address.
- txBuff – The pointer to the data to be transferred.
- txSize – The length in bytes of the data to be transferred.

Return values

- kStatus_Success – Data was sent successfully.
- kStatus_LPI2C_Busy – Another master is currently utilizing the bus.
- kStatus_LPI2C_Nak – The slave device sent a NAK in response to a byte.
- kStatus_LPI2C_FifoError – FIFO under run or over run.
- kStatus_LPI2C_ArbitrationLost – Arbitration lost error.
- kStatus_LPI2C_PinLowTimeout – SCL or SDA were held low longer than the timeout.

status_t LPI2C_MasterReceive(LPI2C_Type *base, void *rxBuff, size_t rxSize)

Performs a polling receive transfer on the I2C bus.

Parameters

- base – The LPI2C peripheral base address.
- rxBuff – The pointer to the data to be transferred.
- rxSize – The length in bytes of the data to be transferred.

Return values

- kStatus_Success – Data was received successfully.
- kStatus_LPI2C_Busy – Another master is currently utilizing the bus.
- kStatus_LPI2C_Nak – The slave device sent a NAK in response to a byte.
- kStatus_LPI2C_FifoError – FIFO under run or overrun.
- kStatus_LPI2C_ArbitrationLost – Arbitration lost error.
- kStatus_LPI2C_PinLowTimeout – SCL or SDA were held low longer than the timeout.

status_t LPI2C_MasterStop(LPI2C_Type *base)

Sends a STOP signal on the I2C bus.

This function does not return until the STOP signal is seen on the bus, or an error occurs.

Parameters

- base – The LPI2C peripheral base address.

Return values

- kStatus_Success – The STOP signal was successfully sent on the bus and the transaction terminated.
- kStatus_LPI2C_Busy – Another master is currently utilizing the bus.
- kStatus_LPI2C_Nak – The slave device sent a NAK in response to a byte.
- kStatus_LPI2C_FifoError – FIFO under run or overrun.
- kStatus_LPI2C_ArbitrationLost – Arbitration lost error.
- kStatus_LPI2C_PinLowTimeout – SCL or SDA were held low longer than the timeout.

status_t LPI2C_MasterTransferBlocking(LPI2C_Type *base, *lpi2c_master_transfer_t* *transfer)

Performs a master polling transfer on the I2C bus.

Note: The API does not return until the transfer succeeds or fails due to error happens during transfer.

Parameters

- base – The LPI2C peripheral base address.
- transfer – Pointer to the transfer structure.

Return values

- kStatus_Success – Data was received successfully.
- kStatus_LPI2C_Busy – Another master is currently utilizing the bus.
- kStatus_LPI2C_Nak – The slave device sent a NAK in response to a byte.
- kStatus_LPI2C_FifoError – FIFO under run or overrun.
- kStatus_LPI2C_ArbitrationLost – Arbitration lost error.
- kStatus_LPI2C_PinLowTimeout – SCL or SDA were held low longer than the timeout.

void LPI2C_MasterTransferCreateHandle(LPI2C_Type *base, *lpi2c_master_handle_t* *handle, *lpi2c_master_transfer_callback_t* callback, void *userData)

Creates a new handle for the LPI2C master non-blocking APIs.

The creation of a handle is for use with the non-blocking APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the *LPI2C_MasterTransferAbort()* API shall be called.

Note: The function also enables the NVIC IRQ for the input LPI2C. Need to notice that on some SoCs the LPI2C IRQ is connected to INTMUX, in this case user needs to enable the associated INTMUX IRQ in application.

Parameters

- base – The LPI2C peripheral base address.
- handle – **[out]** Pointer to the LPI2C master driver handle.
- callback – User provided pointer to the asynchronous callback function.
- userData – User provided pointer to the application callback data.

status_t LPI2C_MasterTransferNonBlocking(LPI2C_Type *base, *lpi2c_master_handle_t* *handle, *lpi2c_master_transfer_t* *transfer)

Performs a non-blocking transaction on the I2C bus.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to the LPI2C master driver handle.
- transfer – The pointer to the transfer descriptor.

Return values

- kStatus_Success – The transaction was started successfully.
- kStatus_LPI2C_Busy – Either another master is currently utilizing the bus, or a non-blocking transaction is already in progress.

status_t LPI2C_MasterTransferGetCount(LPI2C_Type *base, *lpi2c_master_handle_t* *handle, size_t *count)

Returns number of bytes transferred so far.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to the LPI2C master driver handle.
- count – **[out]** Number of bytes transferred so far by the non-blocking transaction.

Return values

- kStatus_Success –
- kStatus_NoTransferInProgress – There is not a non-blocking transaction currently in progress.

void LPI2C_MasterTransferAbort(LPI2C_Type *base, *lpi2c_master_handle_t* *handle)

Terminates a non-blocking LPI2C master transmission early.

Note: It is not safe to call this function from an IRQ handler that has a higher priority than the LPI2C peripheral's IRQ priority.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to the LPI2C master driver handle.

void LPI2C_MasterTransferHandleIRQ(LPI2C_Type *base, void *lpi2cMasterHandle)

Reusable routine to handle master interrupts.

Note: This function does not need to be called unless you are reimplementing the non-blocking API's interrupt handler routines to add special functionality.

Parameters

- base – The LPI2C peripheral base address.
- lpi2cMasterHandle – Pointer to the LPI2C master driver handle.

enum _lpi2c_master_flags

LPI2C master peripheral flags.

The following status register flags can be cleared:

- *kLPI2C_MasterEndOfPacketFlag*
- *kLPI2C_MasterStopDetectFlag*
- *kLPI2C_MasterNackDetectFlag*
- *kLPI2C_MasterArbitrationLostFlag*
- *kLPI2C_MasterFifoErrFlag*
- *kLPI2C_MasterPinLowTimeoutFlag*
- *kLPI2C_MasterDataMatchFlag*

All flags except *kLPI2C_MasterBusyFlag* and *kLPI2C_MasterBusBusyFlag* can be enabled as interrupts.

Note: These enums are meant to be OR'd together to form a bit mask.

Values:

enumerator kLPI2C_MasterTxReadyFlag

Transmit data flag

enumerator kLPI2C_MasterRxReadyFlag

Receive data flag

enumerator kLPI2C_MasterEndOfPacketFlag

End Packet flag

enumerator kLPI2C_MasterStopDetectFlag

Stop detect flag

enumerator kLPI2C_MasterNackDetectFlag

NACK detect flag

enumerator kLPI2C_MasterArbitrationLostFlag

Arbitration lost flag

enumerator kLPI2C_MasterFifoErrFlag

FIFO error flag

enumerator kLPI2C_MasterPinLowTimeoutFlag

Pin low timeout flag

enumerator kLPI2C_MasterDataMatchFlag

Data match flag

enumerator kLPI2C_MasterBusyFlag

Master busy flag

enumerator kLPI2C_MasterBusBusyFlag
Bus busy flag

enumerator kLPI2C_MasterClearFlags
All flags which are cleared by the driver upon starting a transfer.

enumerator kLPI2C_MasterIrqFlags
IRQ sources enabled by the non-blocking transactional API.

enumerator kLPI2C_MasterErrorFlags
Errors to check for.

enum _lpi2c_direction
Direction of master and slave transfers.

Values:

enumerator kLPI2C_Write
Master transmit.

enumerator kLPI2C_Read
Master receive.

enum _lpi2c_master_pin_config
LPI2C pin configuration.

Values:

enumerator kLPI2C_2PinOpenDrain
LPI2C Configured for 2-pin open drain mode

enumerator kLPI2C_2PinOutputOnly
LPI2C Configured for 2-pin output only mode (ultra-fast mode)

enumerator kLPI2C_2PinPushPull
LPI2C Configured for 2-pin push-pull mode

enumerator kLPI2C_4PinPushPull
LPI2C Configured for 4-pin push-pull mode

enumerator kLPI2C_2PinOpenDrainWithSeparateSlave
LPI2C Configured for 2-pin open drain mode with separate LPI2C slave

enumerator kLPI2C_2PinOutputOnlyWithSeparateSlave
LPI2C Configured for 2-pin output only mode(ultra-fast mode) with separate LPI2C slave

enumerator kLPI2C_2PinPushPullWithSeparateSlave
LPI2C Configured for 2-pin push-pull mode with separate LPI2C slave

enumerator kLPI2C_4PinPushPullWithInvertedOutput
LPI2C Configured for 4-pin push-pull mode(inverted outputs)

enum _lpi2c_host_request_source
LPI2C master host request selection.
Values:

enumerator kLPI2C_HostRequestExternalPin
Select the LPI2C_HREQ pin as the host request input

enumerator kLPI2C_HostRequestInputTrigger
Select the input trigger as the host request input

enum _lpi2c_host_request_polarity
LPI2C master host request pin polarity configuration.
Values:

enumerator kLPI2C_HostRequestPinActiveLow
Configure the LPI2C_HREQ pin active low

enumerator kLPI2C_HostRequestPinActiveHigh
Configure the LPI2C_HREQ pin active high

enum _lpi2c_data_match_config_mode
LPI2C master data match configuration modes.
Values:

enumerator kLPI2C_MatchDisabled
LPI2C Match Disabled

enumerator kLPI2C_1stWordEqualsM0OrM1
LPI2C Match Enabled and 1st data word equals MATCH0 OR MATCH1

enumerator kLPI2C_AnyWordEqualsM0OrM1
LPI2C Match Enabled and any data word equals MATCH0 OR MATCH1

enumerator kLPI2C_1stWordEqualsM0And2ndWordEqualsM1
LPI2C Match Enabled and 1st data word equals MATCH0, 2nd data equals MATCH1

enumerator kLPI2C_AnyWordEqualsM0AndNextWordEqualsM1
LPI2C Match Enabled and any data word equals MATCH0, next data equals MATCH1

enumerator kLPI2C_1stWordAndM1EqualsM0AndM1
LPI2C Match Enabled and 1st data word and MATCH0 equals MATCH0 and MATCH1

enumerator kLPI2C_AnyWordAndM1EqualsM0AndM1
LPI2C Match Enabled and any data word and MATCH0 equals MATCH0 and MATCH1

enum _lpi2c_master_transfer_flags

Transfer option flags.

Note: These enumerations are intended to be OR'd together to form a bit mask of options for the *_lpi2c_master_transfer::flags* field.

Values:

enumerator kLPI2C_TransferDefaultFlag

Transfer starts with a start signal, stops with a stop signal.

enumerator kLPI2C_TransferNoStartFlag

Don't send a start condition, address, and sub address

enumerator kLPI2C_TransferNoStopFlag

Don't send a stop condition.

typedef enum *_lpi2c_direction* lpi2c_direction_t

Direction of master and slave transfers.

typedef enum *_lpi2c_master_pin_config* lpi2c_master_pin_config_t

LPI2C pin configuration.

typedef enum *_lpi2c_host_request_source* lpi2c_host_request_source_t

LPI2C master host request selection.

typedef enum *_lpi2c_host_request_polarity* lpi2c_host_request_polarity_t

LPI2C master host request pin polarity configuration.

typedef struct *_lpi2c_master_config* lpi2c_master_config_t

Structure with settings to initialize the LPI2C master module.

This structure holds configuration settings for the LPI2C peripheral. To initialize this structure to reasonable defaults, call the *LPI2C_MasterGetDefaultConfig()* function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

typedef enum *_lpi2c_data_match_config_mode* lpi2c_data_match_config_mode_t

LPI2C master data match configuration modes.

typedef struct *_lpi2c_match_config* lpi2c_data_match_config_t

LPI2C master data match configuration structure.

typedef struct *_lpi2c_master_transfer* lpi2c_master_transfer_t

LPI2C master descriptor of the transfer.

typedef struct *_lpi2c_master_handle* lpi2c_master_handle_t

LPI2C master handle of the transfer.

typedef void (*lpi2c_master_transfer_callback_t)(LPI2C_Type *base, *lpi2c_master_handle_t* *handle, *status_t* completionStatus, void *userData)

Master completion callback function pointer type.

This callback is used only for the non-blocking master transfer API. Specify the callback you wish to use in the call to *LPI2C_MasterTransferCreateHandle()*.

Param base
The LPI2C peripheral base address.

Param handle
Pointer to the LPI2C master driver handle.

Param completionStatus
Either kStatus_Success or an error code describing how the transfer completed.

Param userData
Arbitrary pointer-sized value passed from the application.

typedef void (*lpi2c_master_isr_t)(LPI2C_Type *base, void *handle)

Typedef for master interrupt handler, used internally for LPI2C master interrupt and EDMA transactional APIs.

struct _lpi2c_master_config

#include <fsl_lpi2c.h> Structure with settings to initialize the LPI2C master module.

This structure holds configuration settings for the LPI2C peripheral. To initialize this structure to reasonable defaults, call the *LPI2C_MasterGetDefaultConfig()* function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

Public Members

bool enableMaster

Whether to enable master mode.

bool enableDoze

Whether master is enabled in doze mode.

bool debugEnable

Enable transfers to continue when halted in debug mode.

bool ignoreAck

Whether to ignore ACK/NACK.

lpi2c_master_pin_config_t pinConfig

The pin configuration option.

uint32_t baudRate_Hz

Desired baud rate in Hertz.

uint32_t busIdleTimeout_ns
Bus idle timeout in nanoseconds. Set to 0 to disable.

uint32_t pinLowTimeout_ns
Pin low timeout in nanoseconds. Set to 0 to disable.

uint8_t sdaGlitchFilterWidth_ns
Width in nanoseconds of glitch filter on SDA pin. Set to 0 to disable.

uint8_t sclGlitchFilterWidth_ns
Width in nanoseconds of glitch filter on SCL pin. Set to 0 to disable.

struct *_lpi2c_master_config* hostRequest
Host request options.

struct __lpi2c_match_config
#include <fsl_lpi2c.h> LPI2C master data match configuration structure.

Public Members

lpi2c_data_match_config_mode_t matchMode
Data match configuration setting.

bool rxDataMatchOnly
When set to true, received data is ignored until a successful match.

uint32_t match0
Match value 0.

uint32_t match1
Match value 1.

struct __lpi2c_master_transfer
#include <fsl_lpi2c.h> Non-blocking transfer descriptor structure.

This structure is used to pass transaction parameters to the *LPI2C_MasterTransferNonBlocking()* API.

Public Members

uint32_t flags
Bit mask of options for the transfer. See enumeration *_lpi2c_master_transfer_flags* for available options. Set to 0 or *kLPI2C_TransferDefaultFlag* for normal transfers.

uint16_t slaveAddress
The 7-bit slave address.

lpi2c_direction_t direction
Either *kLPI2C_Read* or *kLPI2C_Write*.

uint32_t subaddress
Sub address. Transferred MSB first.

size_t subaddressSize
Length of sub address to send in bytes. Maximum size is 4 bytes.

void *data
Pointer to data to transfer.

size_t dataSize
Number of bytes to transfer.

struct __lpi2c_master_handle
#include <fsl_lpi2c.h> Driver handle for master non-blocking APIs.

Note: The contents of this structure are private and subject to change.

Public Members

uint8_t state
Transfer state machine current state.

uint16_t remainingBytes
Remaining byte count in current state.

uint8_t *buf
Buffer pointer for current state.

uint16_t commandBuffer[6]
LPI2C command sequence. When all 6 command words are used: Start&addr&write[1 word] + subaddr[4 words] + restart&addr&read[1 word]

lpi2c_master_transfer_t transfer
Copy of the current transfer info.

lpi2c_master_transfer_callback_t completionCallback
Callback function pointer.

void *userData
Application data passed to callback.

uint16_t chunkSize
Remaining byte count in current chunk.

struct hostRequest

Public Members

bool enable
Enable host request.

lpi2c_host_request_source_t source
Host request source.

lpi2c_host_request_polarity_t polarity
Host request pin polarity.

2.40 LPI2C Master DMA Driver

void LPI2C_MasterCreateEDMAHandle(LPI2C_Type *base, *lpi2c_master_edma_handle_t* *handle, *edma_handle_t* *rxDmaHandle, *edma_handle_t* *txDmaHandle, *lpi2c_master_edma_transfer_callback_t* callback, void *userData)

Create a new handle for the LPI2C master DMA APIs.

The creation of a handle is for use with the DMA APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the *LPI2C_MasterTransferAbortEDMA()* API shall be called.

For devices where the LPI2C send and receive DMA requests are OR'd together, the *txDmaHandle* parameter is ignored and may be set to NULL.

Parameters

- base – The LPI2C peripheral base address.
- handle – **[out]** Pointer to the LPI2C master driver handle.
- rxDmaHandle – Handle for the eDMA receive channel. Created by the user prior to calling this function.
- txDmaHandle – Handle for the eDMA transmit channel. Created by the user prior to calling this function.
- callback – User provided pointer to the asynchronous callback function.
- userData – User provided pointer to the application callback data.

status_t LPI2C_MasterTransferEDMA(LPI2C_Type *base, *lpi2c_master_edma_handle_t* *handle, *lpi2c_master_transfer_t* *transfer)

Performs a non-blocking DMA-based transaction on the I2C bus.

The callback specified when the *handle* was created is invoked when the transaction has completed.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to the LPI2C master driver handle.
- transfer – The pointer to the transfer descriptor.

Return values

- kStatus_Success – The transaction was started successfully.

- kStatus_LPI2C_Busy – Either another master is currently utilizing the bus, or another DMA transaction is already in progress.

status_t LPI2C_MasterTransferGetCountEDMA(LPI2C_Type *base, *lpi2c_master_edma_handle_t* *handle, size_t *count)

Returns number of bytes transferred so far.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to the LPI2C master driver handle.
- count – **[out]** Number of bytes transferred so far by the non-blocking transaction.

Return values

- kStatus_Success –
- kStatus_NoTransferInProgress – There is not a DMA transaction currently in progress.

status_t LPI2C_MasterTransferAbortEDMA(LPI2C_Type *base, *lpi2c_master_edma_handle_t* *handle)

Terminates a non-blocking LPI2C master transmission early.

Note: It is not safe to call this function from an IRQ handler that has a higher priority than the eDMA peripheral's IRQ priority.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to the LPI2C master driver handle.

Return values

- kStatus_Success – A transaction was successfully aborted.
- kStatus_LPI2C_Idle – There is not a DMA transaction currently in progress.

typedef struct *_lpi2c_master_edma_handle* lpi2c_master_edma_handle_t

LPI2C master EDMA handle of the transfer.

typedef void (*lpi2c_master_edma_transfer_callback_t)(LPI2C_Type *base, *lpi2c_master_edma_handle_t* *handle, *status_t* completionStatus, void *userData)

Master DMA completion callback function pointer type.

This callback is used only for the non-blocking master transfer API. Specify the callback you wish to use in the call to *LPI2C_MasterCreateEDMAHandle()*.

Param base
The LPI2C peripheral base address.

Param handle
Handle associated with the completed transfer.

Param completionStatus
Either kStatus_Success or an error code describing how the transfer completed.

Param userData
Arbitrary pointer-sized value passed from the application.

struct __lpi2c_master_edma_handle

#include <fsl_lpi2c_edma.h> Driver handle for master DMA APIs.

Note: The contents of this structure are private and subject to change.

Public Members

LPI2C_Type *base

LPI2C base pointer.

bool isBusy

Transfer state machine current state.

uint8_t nbytes

eDMA minor byte transfer count initially configured.

uint16_t commandBuffer[20U]

LPI2C command sequence. When all 10 command words are used: Start&addr&write[1 word] + subaddr[4 words] + restart&addr&read[1 word] + receive&Size[4 words]

lpi2c_master_transfer_t transfer

Copy of the current transfer info.

lpi2c_master_edma_transfer_callback_t completionCallback

Callback function pointer.

void *userData

Application data passed to callback.

edma_handle_t *rx

Handle for receive DMA channel.

edma_handle_t *tx

Handle for transmit DMA channel.

edma_tcd_t tcds[3]

Software TCD. Three are allocated to provide enough room to align to 32-bytes.

2.41 LPI2C Slave Driver

void LPI2C_SlaveGetDefaultConfig(*lpi2c_slave_config_t* *slaveConfig)

Provides a default configuration for the LPI2C slave peripheral.

This function provides the following default configuration for the LPI2C slave peripheral:

```
slaveConfig->enableSlave              = true;
slaveConfig->address0                 = 0U;
slaveConfig->address1                 = 0U;
slaveConfig->addressMatchMode         = kLPI2C_MatchAddress0;
slaveConfig->filterDozeEnable         = true;
slaveConfig->filterEnable             = true;
slaveConfig->enableGeneralCall        = false;
slaveConfig->sclStall.enableAck       = false;
slaveConfig->sclStall.enableTx        = true;
slaveConfig->sclStall.enableRx        = true;
slaveConfig->sclStall.enableAddress   = true;
slaveConfig->ignoreAck                = false;
slaveConfig->enableReceivedAddressRead = false;
slaveConfig->sdaGlitchFilterWidth_ns  = 0;
slaveConfig->sclGlitchFilterWidth_ns  = 0;
slaveConfig->dataValidDelay_ns        = 0;
slaveConfig->clockHoldTime_ns         = 0;
```

After calling this function, override any settings to customize the configuration, prior to initializing the master driver with *LPI2C_SlaveInit()*. Be sure to override at least the *address0* member of the configuration structure with the desired slave address.

Parameters

- slaveConfig – **[out]** User provided configuration structure that is set to default values. Refer to *lpi2c_slave_config_t*.

void LPI2C_SlaveInit(LPI2C_Type *base, const *lpi2c_slave_config_t* *slaveConfig, uint32_t sourceClock_Hz)

Initializes the LPI2C slave peripheral.

This function enables the peripheral clock and initializes the LPI2C slave peripheral as described by the user provided configuration.

Parameters

- base – The LPI2C peripheral base address.
- slaveConfig – User provided peripheral configuration. Use *LPI2C_SlaveGetDefaultConfig()* to get a set of defaults that you can override.
- sourceClock_Hz – Frequency in Hertz of the LPI2C functional clock. Used to calculate the filter widths, data valid delay, and clock hold time.

void LPI2C_SlaveDeinit(LPI2C_Type *base)

Deinitializes the LPI2C slave peripheral.

This function disables the LPI2C slave peripheral and gates the clock. It also performs a software reset to restore the peripheral to reset conditions.

Parameters

- base – The LPI2C peripheral base address.

static inline void LPI2C_SlaveReset(LPI2C_Type *base)

Performs a software reset of the LPI2C slave peripheral.

Parameters

- base – The LPI2C peripheral base address.

static inline void LPI2C_SlaveEnable(LPI2C_Type *base, bool enable)

Enables or disables the LPI2C module as slave.

Parameters

- base – The LPI2C peripheral base address.
- enable – Pass true to enable or false to disable the specified LPI2C as slave.

static inline uint32_t LPI2C_SlaveGetStatusFlags(LPI2C_Type *base)

Gets the LPI2C slave status flags.

A bit mask with the state of all LPI2C slave status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

See also:

_lpi2c_slave_flags

Parameters

- base – The LPI2C peripheral base address.

Returns

State of the status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

static inline void LPI2C_SlaveClearStatusFlags(LPI2C_Type *base, uint32_t statusMask)

Clears the LPI2C status flag state.

The following status register flags can be cleared:

- *kLPI2C_SlaveRepeatedStartDetectFlag*
- *kLPI2C_SlaveStopDetectFlag*
- *kLPI2C_SlaveBitErrFlag*
- *kLPI2C_SlaveFifoErrFlag*

Attempts to clear other flags has no effect.

See also:

_lpi2c_slave_flags.

Parameters

- base – The LPI2C peripheral base address.
- statusMask – A bitmask of status flags that are to be cleared. The mask is composed of *_lpi2c_slave_flags* enumerators OR'd together. You may pass the result of a previous call to *LPI2C_SlaveGetStatusFlags()*.

static inline void LPI2C_SlaveEnableInterrupts(LPI2C_Type *base, uint32_t interruptMask)

Enables the LPI2C slave interrupt requests.

All flags except *kLPI2C_SlaveBusyFlag* and *kLPI2C_SlaveBusBusyFlag* can be enabled as interrupts.

Parameters

- base – The LPI2C peripheral base address.

- interruptMask – Bit mask of interrupts to enable. See *_lpi2c_slave_flags* for the set of constants that should be OR'd together to form the bit mask.

static inline void LPI2C_SlaveDisableInterrupts(LPI2C_Type *base, uint32_t interruptMask)

Disables the LPI2C slave interrupt requests.

All flags except *kLPI2C_SlaveBusyFlag* and *kLPI2C_SlaveBusBusyFlag* can be enabled as interrupts.

Parameters

- base – The LPI2C peripheral base address.
- interruptMask – Bit mask of interrupts to disable. See *_lpi2c_slave_flags* for the set of constants that should be OR'd together to form the bit mask.

static inline uint32_t LPI2C_SlaveGetEnabledInterrupts(LPI2C_Type *base)

Returns the set of currently enabled LPI2C slave interrupt requests.

Parameters

- base – The LPI2C peripheral base address.

Returns

A bitmask composed of *_lpi2c_slave_flags* enumerators OR'd together to indicate the set of enabled interrupts.

static inline void LPI2C_SlaveEnableDMA(LPI2C_Type *base, bool enableAddressValid, bool enableRx, bool enableTx)

Enables or disables the LPI2C slave peripheral DMA requests.

Parameters

- base – The LPI2C peripheral base address.
- enableAddressValid – Enable flag for the address valid DMA request. Pass true for enable, false for disable. The address valid DMA request is shared with the receive data DMA request.
- enableRx – Enable flag for the receive data DMA request. Pass true for enable, false for disable.
- enableTx – Enable flag for the transmit data DMA request. Pass true for enable, false for disable.

static inline bool LPI2C_SlaveGetBusIdleState(LPI2C_Type *base)

Returns whether the bus is idle.

Requires the slave mode to be enabled.

Parameters

- base – The LPI2C peripheral base address.

Return values

- true – Bus is busy.
- false – Bus is idle.

static inline void LPI2C_SlaveTransmitAck(LPI2C_Type *base, bool ackOrNack)

Transmits either an ACK or NAK on the I2C bus in response to a byte from the master.

Use this function to send an ACK or NAK when the *kLPI2C_SlaveTransmitAckFlag* is asserted. This only happens if you enable the sclStall.enableAck field of the *lpi2c_slave_config_t* configuration structure used to initialize the slave peripheral.

Parameters

- base – The LPI2C peripheral base address.

- ackOrNack – Pass true for an ACK or false for a NAK.

static inline void LPI2C_SlaveEnableAckStall(LPI2C_Type *base, bool enable)

Enables or disables ACKSTALL.

When enables ACKSTALL, software can transmit either an ACK or NAK on the I2C bus in response to a byte from the master.

Parameters

- base – The LPI2C peripheral base address.
- enable – True will enable ACKSTALL,false will disable ACKSTALL.

static inline uint32_t LPI2C_SlaveGetReceivedAddress(LPI2C_Type *base)

Returns the slave address sent by the I2C master.

This function should only be called if the *kLPI2C_SlaveAddressValidFlag* is asserted.

Parameters

- base – The LPI2C peripheral base address.

Returns

The 8-bit address matched by the LPI2C slave. Bit 0 contains the R/w direction bit, and the 7-bit slave address is in the upper 7 bits.

status_t LPI2C_SlaveSend(LPI2C_Type *base, void *txBuff, size_t txSize, size_t *actualTxSize)

Performs a polling send transfer on the I2C bus.

Parameters

- base – The LPI2C peripheral base address.
- txBuff – The pointer to the data to be transferred.
- txSize – The length in bytes of the data to be transferred.
- actualTxSize – **[out]**

Returns

Error or success status returned by API.

status_t LPI2C_SlaveReceive(LPI2C_Type *base, void *rxBuff, size_t rxSize, size_t *actualRxSize)

Performs a polling receive transfer on the I2C bus.

Parameters

- base – The LPI2C peripheral base address.
- rxBuff – The pointer to the data to be transferred.
- rxSize – The length in bytes of the data to be transferred.
- actualRxSize – **[out]**

Returns

Error or success status returned by API.

void LPI2C_SlaveTransferCreateHandle(LPI2C_Type *base, *lpi2c_slave_handle_t* *handle, *lpi2c_slave_transfer_callback_t* callback, void *userData)

Creates a new handle for the LPI2C slave non-blocking APIs.

The creation of a handle is for use with the non-blocking APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the *LPI2C_SlaveTransferAbort()* API shall be called.

Note: The function also enables the NVIC IRQ for the input LPI2C. Need to notice that on some SoCs the LPI2C IRQ is connected to INTMUX, in this case user needs to enable the associated INTMUX IRQ in application.

Parameters

- base – The LPI2C peripheral base address.
- handle – **[out]** Pointer to the LPI2C slave driver handle.
- callback – User provided pointer to the asynchronous callback function.
- userData – User provided pointer to the application callback data.

status_t LPI2C_SlaveTransferNonBlocking(LPI2C_Type *base, *lpi2c_slave_handle_t* *handle, uint32_t eventMask)

Starts accepting slave transfers.

Call this API after calling I2C_SlaveInit() and *LPI2C_SlaveTransferCreateHandle()* to start processing transactions driven by an I2C master. The slave monitors the I2C bus and pass events to the callback that was passed into the call to *LPI2C_SlaveTransferCreateHandle()*. The callback is always invoked from the interrupt context.

The set of events received by the callback is customizable. To do so, set the *eventMask* parameter to the OR'd combination of *lpi2c_slave_transfer_event_t* enumerators for the events you wish to receive. The *kLPI2C_SlaveTransmitEvent* and *kLPI2C_SlaveReceiveEvent* events are always enabled and do not need to be included in the mask. Alternatively, you can pass 0 to get a default set of only the transmit and receive events that are always enabled. In addition, the *kLPI2C_SlaveAllEvents* constant is provided as a convenient way to enable all events.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to lpi2c_slave_handle_t structure which stores the transfer state.
- eventMask – Bit mask formed by OR'ing together *lpi2c_slave_transfer_event_t* enumerators to specify which events to send to the callback. Other accepted values are 0 to get a default set of only the transmit and receive events, and *kLPI2C_SlaveAllEvents* to enable all events.

Return values

- kStatus_Success – Slave transfers were successfully started.
- kStatus_LPI2C_Busy – Slave transfers have already been started on this handle.

status_t LPI2C_SlaveTransferGetCount(LPI2C_Type *base, *lpi2c_slave_handle_t* *handle, size_t *count)

Gets the slave transfer status during a non-blocking transfer.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to i2c_slave_handle_t structure.
- count – **[out]** Pointer to a value to hold the number of bytes transferred. May be NULL if the count is not required.

Return values

- kStatus_Success –
- kStatus_NoTransferInProgress –

void LPI2C_SlaveTransferAbort(LPI2C_Type *base, *lpi2c_slave_handle_t* *handle)

Aborts the slave non-blocking transfers.

Note: This API could be called at any time to stop slave for handling the bus events.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to lpi2c_slave_handle_t structure which stores the transfer state.

void LPI2C_SlaveTransferHandleIRQ(LPI2C_Type *base, *lpi2c_slave_handle_t* *handle)

Reusable routine to handle slave interrupts.

Note: This function does not need to be called unless you are reimplementing the non blocking API's interrupt handler routines to add special functionality.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to lpi2c_slave_handle_t structure which stores the transfer state.

enum _lpi2c_slave_flags

LPI2C slave peripheral flags.

The following status register flags can be cleared:

- *kLPI2C_SlaveRepeatedStartDetectFlag*
- *kLPI2C_SlaveStopDetectFlag*
- *kLPI2C_SlaveBitErrFlag*
- *kLPI2C_SlaveFifoErrFlag*

All flags except *kLPI2C_SlaveBusyFlag* and *kLPI2C_SlaveBusBusyFlag* can be enabled as interrupts.

Note: These enumerations are meant to be OR'd together to form a bit mask.

Values:

enumerator kLPI2C_SlaveTxReadyFlag

Transmit data flag

enumerator kLPI2C_SlaveRxReadyFlag

Receive data flag

enumerator kLPI2C_SlaveAddressValidFlag

Address valid flag

enumerator kLPI2C_SlaveTransmitAckFlag
Transmit ACK flag

enumerator kLPI2C_SlaveRepeatedStartDetectFlag
Repeated start detect flag

enumerator kLPI2C_SlaveStopDetectFlag
Stop detect flag

enumerator kLPI2C_SlaveBitErrFlag
Bit error flag

enumerator kLPI2C_SlaveFifoErrFlag
FIFO error flag

enumerator kLPI2C_SlaveAddressMatch0Flag
Address match 0 flag

enumerator kLPI2C_SlaveAddressMatch1Flag
Address match 1 flag

enumerator kLPI2C_SlaveGeneralCallFlag
General call flag

enumerator kLPI2C_SlaveBusyFlag
Master busy flag

enumerator kLPI2C_SlaveBusBusyFlag
Bus busy flag

enumerator kLPI2C_SlaveClearFlags
All flags which are cleared by the driver upon starting a transfer.

enumerator kLPI2C_SlaveIrqFlags
IRQ sources enabled by the non-blocking transactional API.

enumerator kLPI2C_SlaveErrorFlags
Errors to check for.

enum _lpi2c_slave_address_match
LPI2C slave address match options.
Values:

enumerator kLPI2C_MatchAddress0
Match only address 0.

enumerator kLPI2C_MatchAddress0OrAddress1
Match either address 0 or address 1.

enumerator kLPI2C_MatchAddress0ThroughAddress1

Match a range of slave addresses from address 0 through address 1.

enum _lpi2c_slave_transfer_event

Set of events sent to the callback for non blocking slave transfers.

These event enumerations are used for two related purposes. First, a bit mask created by OR'ing together events is passed to *LPI2C_SlaveTransferNonBlocking()* in order to specify which events to enable. Then, when the slave callback is invoked, it is passed the current event through its *transfer* parameter.

Note: These enumerations are meant to be OR'd together to form a bit mask of events.

Values:

enumerator kLPI2C_SlaveAddressMatchEvent

Received the slave address after a start or repeated start.

enumerator kLPI2C_SlaveTransmitEvent

Callback is requested to provide data to transmit (slave-transmitter role).

enumerator kLPI2C_SlaveReceiveEvent

Callback is requested to provide a buffer in which to place received data (slave-receiver role).

enumerator kLPI2C_SlaveTransmitAckEvent

Callback needs to either transmit an ACK or NACK.

enumerator kLPI2C_SlaveRepeatedStartEvent

A repeated start was detected.

enumerator kLPI2C_SlaveCompletionEvent

A stop was detected, completing the transfer.

enumerator kLPI2C_SlaveAllEvents

Bit mask of all available events.

typedef enum *_lpi2c_slave_address_match* lpi2c_slave_address_match_t

LPI2C slave address match options.

typedef struct *_lpi2c_slave_config* lpi2c_slave_config_t

Structure with settings to initialize the LPI2C slave module.

This structure holds configuration settings for the LPI2C slave peripheral. To initialize this structure to reasonable defaults, call the *LPI2C_SlaveGetDefaultConfig()* function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

typedef enum *_lpi2c_slave_transfer_event* lpi2c_slave_transfer_event_t

Set of events sent to the callback for non blocking slave transfers.

These event enumerations are used for two related purposes. First, a bit mask created by OR'ing together events is passed to *LPI2C_SlaveTransferNonBlocking()* in order to specify which events to enable. Then, when the slave callback is invoked, it is passed the current event through its *transfer* parameter.

Note: These enumerations are meant to be OR'd together to form a bit mask of events.

typedef struct *_lpi2c_slave_transfer* lpi2c_slave_transfer_t

LPI2C slave transfer structure.

typedef struct *_lpi2c_slave_handle* lpi2c_slave_handle_t

LPI2C slave handle structure.

typedef void (*lpi2c_slave_transfer_callback_t)(LPI2C_Type *base, *lpi2c_slave_transfer_t* *transfer, void *userData)

Slave event callback function pointer type.

This callback is used only for the slave non-blocking transfer API. To install a callback, use the LPI2C_SlaveSetCallback() function after you have created a handle.

Param base
Base address for the LPI2C instance on which the event occurred.

Param transfer
Pointer to transfer descriptor containing values passed to and/or from the callback.

Param userData
Arbitrary pointer-sized value passed from the application.

struct _lpi2c_slave_config

#include <fsl_lpi2c.h> Structure with settings to initialize the LPI2C slave module.

This structure holds configuration settings for the LPI2C slave peripheral. To initialize this structure to reasonable defaults, call the *LPI2C_SlaveGetDefaultConfig()* function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

Public Members

bool enableSlave

Enable slave mode.

uint8_t address0

Slave's 7-bit address.

uint8_t address1

Alternate slave 7-bit address.

lpi2c_slave_address_match_t addressMatchMode

Address matching options.

bool filterDozeEnable
Enable digital glitch filter in doze mode.

bool filterEnable
Enable digital glitch filter.

bool enableGeneralCall
Enable general call address matching.

struct *_lpi2c_slave_config* sclStall
SCL stall enable options.

bool ignoreAck
Continue transfers after a NACK is detected.

bool enableReceivedAddressRead
Enable reading the address received address as the first byte of data.

uint32_t sdaGlitchFilterWidth_ns
Width in nanoseconds of the digital filter on the SDA signal. Set to 0 to disable.

uint32_t sclGlitchFilterWidth_ns
Width in nanoseconds of the digital filter on the SCL signal. Set to 0 to disable.

uint32_t dataValidDelay_ns
Width in nanoseconds of the data valid delay.

uint32_t clockHoldTime_ns
Width in nanoseconds of the clock hold time.

struct __lpi2c__slave__transfer
#include <fsl_lpi2c.h> LPI2C slave transfer structure.

Public Members

lpi2c_slave_transfer_event_t event
Reason the callback is being invoked.

uint8_t receivedAddress
Matching address send by master.

uint8_t *data
Transfer buffer

size_t dataSize
Transfer size

status_t completionStatus

Success or error code describing how the transfer completed. Only applies for *kLPI2C_SlaveCompletionEvent*.

size_t transferredCount

Number of bytes actually transferred since start or last repeated start.

struct __lpi2c__slave__handle

#include <fsl_lpi2c.h> LPI2C slave handle structure.

Note: The contents of this structure are private and subject to change.

Public Members

lpi2c_slave_transfer_t transfer

LPI2C slave transfer copy.

bool isBusy

Whether transfer is busy.

bool wasTransmit

Whether the last transfer was a transmit.

uint32_t eventMask

Mask of enabled events.

uint32_t transferredCount

Count of bytes transferred.

lpi2c_slave_transfer_callback_t callback

Callback function called at transfer event.

void *userData

Callback parameter passed to callback.

struct sclStall

Public Members

bool enableAck

Enables SCL clock stretching during slave-transmit address byte(s) and slave-receiver address and data byte(s) to allow software to write the Transmit ACK Register before the ACK or NACK is transmitted. Clock stretching occurs when transmitting the 9th bit. When enableAckSCLStall is enabled, there is no need to set either enableRxDataSCLStall or enableAddressSCLStall.

bool enableTx

Enables SCL clock stretching when the transmit data flag is set during a slave-transmit transfer.

bool enableRx

Enables SCL clock stretching when receive data flag is set during a slave-receive transfer.

bool enableAddress

Enables SCL clock stretching when the address valid flag is asserted.

2.42 LPIT: Low-Power Interrupt Timer

void LPIT_Init(LPIT_Type *base, const *lpit_config_t* *config)

Ungates the LPIT clock and configures the peripheral for a basic operation.

This function issues a software reset to reset all channels and registers except the Module Control register.

Note: This API should be called at the beginning of the application using the LPIT driver.

Parameters

- base – LPIT peripheral base address.
- config – Pointer to the user configuration structure.

void LPIT_Deinit(LPIT_Type *base)

Disables the module and gates the LPIT clock.

Parameters

- base – LPIT peripheral base address.

void LPIT_GetDefaultConfig(*lpit_config_t* *config)

Fills in the LPIT configuration structure with default settings.

The default values are:

```
config->enableRunInDebug = false;
config->enableRunInDoze = false;
```

Parameters

- config – Pointer to the user configuration structure.

status_t LPIT_SetupChannel(LPIT_Type *base, *lpit_chnl_t* channel, const *lpit_chnl_params_t* *chnlSetup)

Sets up an LPIT channel based on the user's preference.

This function sets up the operation mode to one of the options available in the enumeration *lpit_timer_modes_t*. It sets the trigger source as either internal or external, trigger selection and the timers behaviour when a timeout occurs. It also chains the timer if a prior timer if requested by the user.

Parameters

- base – LPIT peripheral base address.
- channel – Channel that is being configured.
- chnlSetup – Configuration parameters.

static inline void LPIT_EnableInterrupts(LPIT_Type *base, uint32_t mask)

Enables the selected PIT interrupts.

Parameters

- base – LPIT peripheral base address.
- mask – The interrupts to enable. This is a logical OR of members of the enumeration *lpit_interrupt_enable_t*

static inline void LPIT_DisableInterrupts(LPIT_Type *base, uint32_t mask)

Disables the selected PIT interrupts.

Parameters

- base – LPIT peripheral base address.
- mask – The interrupts to enable. This is a logical OR of members of the enumeration *lpit_interrupt_enable_t*

static inline uint32_t LPIT_GetEnabledInterrupts(LPIT_Type *base)

Gets the enabled LPIT interrupts.

Parameters

- base – LPIT peripheral base address.

Returns

The enabled interrupts. This is the logical OR of members of the enumeration *lpit_interrupt_enable_t*

static inline uint32_t LPIT_GetStatusFlags(LPIT_Type *base)

Gets the LPIT status flags.

Parameters

- base – LPIT peripheral base address.

Returns

The status flags. This is the logical OR of members of the enumeration *lpit_status_flags_t*

static inline void LPIT_ClearStatusFlags(LPIT_Type *base, uint32_t mask)

Clears the LPIT status flags.

Parameters

- base – LPIT peripheral base address.
- mask – The status flags to clear. This is a logical OR of members of the enumeration *lpit_status_flags_t*

static inline void LPIT_SetTimerPeriod(LPIT_Type *base, *lpit_chnl_t* channel, uint32_t ticks)

Sets the timer period in units of count.

Timers begin counting down from the value set by this function until it reaches 0, at which point it generates an interrupt and loads this register value again. Writing a new value to this register does not restart the timer. Instead, the value is loaded after the timer expires.

Note: User can call the utility macros provided in fsl_common.h to convert to ticks.

Parameters

- base – LPIT peripheral base address.
- channel – Timer channel number.
- ticks – Timer period in units of ticks.

static inline void LPIT_SetTimerValue(LPIT_Type *base, *lpit_chnl_t* channel, uint32_t ticks)

Sets the timer period in units of count.

In the Dual 16-bit Periodic Counter mode, the counter will load and then the lower 16-bits will decrement down to zero, which will assert the output pre-trigger. The upper 16-bits will then decrement down to zero, which will negate the output pre-trigger and set the timer interrupt flag.

Note: Set TVAL register to 0 or 1 is invalid in compare mode.

Parameters

- base – LPIT peripheral base address.
- channel – Timer channel number.
- ticks – Timer period in units of ticks.

static inline uint32_t LPIT_GetCurrentTimerCount(LPIT_Type *base, *lpit_chnl_t* channel)

Reads the current timer counting value.

This function returns the real-time timer counting value, in a range from 0 to a timer period.

Note: User can call the utility macros provided in fsl_common.h to convert ticks to microseconds or milliseconds.

Parameters

- base – LPIT peripheral base address.
- channel – Timer channel number.

Returns

Current timer counting value in ticks.

static inline void LPIT_StartTimer(LPIT_Type *base, *lpit_chnl_t* channel)

Starts the timer counting.

After calling this function, timers load the period value and count down to 0. When the timer reaches 0, it generates a trigger pulse and sets the timeout interrupt flag.

Parameters

- base – LPIT peripheral base address.
- channel – Timer channel number.

static inline void LPIT_StopTimer(LPIT_Type *base, *lpit_chnl_t* channel)

Stops the timer counting.

Parameters

- base – LPIT peripheral base address.
- channel – Timer channel number.

FSL_LPIT_DRIVER_VERSION

Version 2.1.3

enum _lpit_chnl

List of LPIT channels.

Note: Actual number of available channels is SoC-dependent

Values:

enumerator kLPIT_Chnl_0

LPIT channel number 0

enumerator kLPIT_Chnl_1

LPIT channel number 1

enumerator kLPIT_Chnl_2

LPIT channel number 2

enumerator kLPIT_Chnl_3

LPIT channel number 3

enum _lpit_timer_modes

Mode options available for the LPIT timer.

Values:

enumerator kLPIT_PeriodicCounter

Use the all 32-bits, counter loads and decrements to zero

enumerator kLPIT_DualPeriodicCounter

Counter loads, lower 16-bits decrement to zero, then upper 16-bits decrement

enumerator kLPIT_TriggerAccumulator

Counter loads on first trigger and decrements on each trigger

enumerator kLPIT_InputCapture

Counter loads with 0xFFFFFFFF, decrements to zero. It stores the inverse of the current value when a input trigger is detected

enum _lpit_trigger_select

Trigger options available.

This is used for both internal and external trigger sources. The actual trigger options available is SoC-specific, user should refer to the reference manual.

Values:

enumerator kLPIT_Trigger_TimerChn0

Channel 0 is selected as a trigger source

enumerator kLPIT_Trigger_TimerChn1
Channel 1 is selected as a trigger source

enumerator kLPIT_Trigger_TimerChn2
Channel 2 is selected as a trigger source

enumerator kLPIT_Trigger_TimerChn3
Channel 3 is selected as a trigger source

enumerator kLPIT_Trigger_TimerChn4
Channel 4 is selected as a trigger source

enumerator kLPIT_Trigger_TimerChn5
Channel 5 is selected as a trigger source

enumerator kLPIT_Trigger_TimerChn6
Channel 6 is selected as a trigger source

enumerator kLPIT_Trigger_TimerChn7
Channel 7 is selected as a trigger source

enumerator kLPIT_Trigger_TimerChn8
Channel 8 is selected as a trigger source

enumerator kLPIT_Trigger_TimerChn9
Channel 9 is selected as a trigger source

enumerator kLPIT_Trigger_TimerChn10
Channel 10 is selected as a trigger source

enumerator kLPIT_Trigger_TimerChn11
Channel 11 is selected as a trigger source

enumerator kLPIT_Trigger_TimerChn12
Channel 12 is selected as a trigger source

enumerator kLPIT_Trigger_TimerChn13
Channel 13 is selected as a trigger source

enumerator kLPIT_Trigger_TimerChn14
Channel 14 is selected as a trigger source

enumerator kLPIT_Trigger_TimerChn15
Channel 15 is selected as a trigger source

enum _lpit_trigger_source
Trigger source options available.

Values:

enumerator kLPIT_TriggerSource_External
Use external trigger input

enumerator kLPIT_TriggerSource_Internal
Use internal trigger

enum _lpit_interrupt_enable
List of LPIT interrupts.

Note: Number of timer channels are SoC-specific. See the SoC Reference Manual.

Values:

enumerator kLPIT_Channel0TimerInterruptEnable
Channel 0 Timer interrupt

enumerator kLPIT_Channel1TimerInterruptEnable
Channel 1 Timer interrupt

enumerator kLPIT_Channel2TimerInterruptEnable
Channel 2 Timer interrupt

enumerator kLPIT_Channel3TimerInterruptEnable
Channel 3 Timer interrupt

enum _lpit_status_flags
List of LPIT status flags.

Note: Number of timer channels are SoC-specific. See the SoC Reference Manual.

Values:

enumerator kLPIT_Channel0TimerFlag
Channel 0 Timer interrupt flag

enumerator kLPIT_Channel1TimerFlag
Channel 1 Timer interrupt flag

enumerator kLPIT_Channel2TimerFlag
Channel 2 Timer interrupt flag

enumerator kLPIT_Channel3TimerFlag
Channel 3 Timer interrupt flag

typedef enum *_lpit_chnl* lpit_chnl_t
List of LPIT channels.

Note: Actual number of available channels is SoC-dependent

typedef enum *_lpit_timer_modes* lpit_timer_modes_t

Mode options available for the LPIT timer.

typedef enum *_lpit_trigger_select* lpit_trigger_select_t

Trigger options available.

This is used for both internal and external trigger sources. The actual trigger options available is SoC-specific, user should refer to the reference manual.

typedef enum *_lpit_trigger_source* lpit_trigger_source_t

Trigger source options available.

typedef enum *_lpit_interrupt_enable* lpit_interrupt_enable_t

List of LPIT interrupts.

Note: Number of timer channels are SoC-specific. See the SoC Reference Manual.

typedef enum *_lpit_status_flags* lpit_status_flags_t

List of LPIT status flags.

Note: Number of timer channels are SoC-specific. See the SoC Reference Manual.

typedef struct *_lpit_chnl_params* lpit_chnl_params_t

Structure to configure the channel timer.

typedef struct *_lpit_config* lpit_config_t

LPIT configuration structure.

This structure holds the configuration settings for the LPIT peripheral. To initialize this structure to reasonable defaults, call the *LPIT_GetDefaultConfig()* function and pass a pointer to the configuration structure instance.

The configuration structure can be made constant so as to reside in flash.

static void LPIT_ResetStateDelay(void)

Short wait for LPIT state reset.

After clear or set LPIT_EN, there should be delay longer than 4 LPIT functional clock.

static inline void LPIT_Reset(LPIT_Type *base)

Performs a software reset on the LPIT module.

This resets all channels and registers except the Module Control Register.

Parameters

- base – LPIT peripheral base address.

LPIT_RESET_STATE_DELAY

Delay used in LPIT_Reset.

The macro value should be larger than 4 * core clock / LPIT peripheral clock.

struct _lpit_chnl_params

#include <fsl_lpit.h> Structure to configure the channel timer.

Public Members

bool chainChannel
: true: Timer chained to previous timer; false: Timer not chained

lpit_timer_modes_t timerMode
: Timers mode of operation.

lpit_trigger_select_t triggerSelect
: Trigger selection for the timer

lpit_trigger_source_t triggerSource
: Decides if we use external or internal trigger.

bool enableReloadOnTrigger
: true: Timer reloads when a trigger is detected; false: No effect

bool enableStopOnTimeout
: true: Timer will stop after timeout; false: does not stop after timeout

bool enableStartOnTrigger
: true: Timer starts when a trigger is detected; false: decrement immediately

struct __lpit_config
: *#include <fsl_lpit.h>* LPIT configuration structure.

This structure holds the configuration settings for the LPIT peripheral. To initialize this structure to reasonable defaults, call the *LPIT_GetDefaultConfig()* function and pass a pointer to the configuration structure instance.

The configuration structure can be made constant so as to reside in flash.

Public Members

bool enableRunInDebug
: true: Timers run in debug mode; false: Timers stop in debug mode

bool enableRunInDoze
: true: Timers run in doze mode; false: Timers stop in doze mode

2.43 LPSPI: Low Power Serial Peripheral Interface

2.44 LPSPI Peripheral driver

void LPSPI_MasterInit(LPSPI_Type *base, const *lpspi_master_config_t* *masterConfig, uint32_t srcClock_Hz)

Initializes the LPSPI master.

Parameters

- base – LPSPI peripheral address.
- masterConfig – Pointer to structure lpspi_master_config_t.
- srcClock_Hz – Module source input clock in Hertz

void LPSPI_MasterGetDefaultConfig(*lpspi_master_config_t* *masterConfig)

Sets the lpspi_master_config_t structure to default values.

This API initializes the configuration structure for *LPSPI_MasterInit()*. The initialized structure can remain unchanged in *LPSPI_MasterInit()*, or can be modified before calling the *LPSPI_MasterInit()*. Example:

```
lpspi_master_config_t masterConfig;
LPSPI_MasterGetDefaultConfig(&masterConfig);
```

Parameters

- masterConfig – pointer to lpspi_master_config_t structure

void LPSPI_SlaveInit(LPSPI_Type *base, const *lpspi_slave_config_t* *slaveConfig)

LPSPI slave configuration.

Parameters

- base – LPSPI peripheral address.
- slaveConfig – Pointer to a structure lpspi_slave_config_t.

void LPSPI_SlaveGetDefaultConfig(*lpspi_slave_config_t* *slaveConfig)

Sets the lpspi_slave_config_t structure to default values.

This API initializes the configuration structure for *LPSPI_SlaveInit()*. The initialized structure can remain unchanged in *LPSPI_SlaveInit()* or can be modified before calling the *LPSPI_SlaveInit()*. Example:

```
lpspi_slave_config_t slaveConfig;
LPSPI_SlaveGetDefaultConfig(&slaveConfig);
```

Parameters

- slaveConfig – pointer to lpspi_slave_config_t structure.

void LPSPI_Deinit(LPSPI_Type *base)

De-initializes the LPSPI peripheral. Call this API to disable the LPSPI clock.

Parameters

- base – LPSPI peripheral address.

void LPSPI_Reset(LPSPI_Type *base)

Restores the LPSPI peripheral to reset state. Note that this function sets all registers to reset state. As a result, the LPSPI module can't work after calling this API.

Parameters

- base – LPSPI peripheral address.

uint32_t LPSPI_GetInstance(LPSPI_Type *base)

Get the LPSPI instance from peripheral base address.

Parameters

- base – LPSPI peripheral base address.

Returns

LPSPI instance.

static inline void LPSPI_Enable(LPSPI_Type *base, bool enable)

Enables the LPSPI peripheral and sets the MCR MDIS to 0.

Parameters

- base – LPSPI peripheral address.
- enable – Pass true to enable module, false to disable module.

static inline uint32_t LPSPI_GetStatusFlags(LPSPI_Type *base)

Gets the LPSPI status flag state.

Parameters

- base – LPSPI peripheral address.

Returns

The LPSPI status(in SR register).

static inline uint8_t LPSPI_GetTxFifoSize(LPSPI_Type *base)

Gets the LPSPI Tx FIFO size.

Parameters

- base – LPSPI peripheral address.

Returns

The LPSPI Tx FIFO size.

static inline uint8_t LPSPI_GetRxFifoSize(LPSPI_Type *base)

Gets the LPSPI Rx FIFO size.

Parameters

- base – LPSPI peripheral address.

Returns

The LPSPI Rx FIFO size.

static inline uint32_t LPSPI_GetTxFifoCount(LPSPI_Type *base)

Gets the LPSPI Tx FIFO count.

Parameters

- base – LPSPI peripheral address.

Returns

The number of words in the transmit FIFO.

static inline uint32_t LPSPI_GetRxFifoCount(LPSPI_Type *base)

Gets the LPSPI Rx FIFO count.

Parameters

- base – LPSPI peripheral address.

Returns

The number of words in the receive FIFO.

static inline void LPSPI_ClearStatusFlags(LPSPI_Type *base, uint32_t statusFlags)

Clears the LPSPI status flag.

This function clears the desired status bit by using a write-1-to-clear. The user passes in the base and the desired status flag bit to clear. The list of status flags is defined in the _lpspi_flags. Example usage:

```
LPSPI_ClearStatusFlags(base, kLPSPI_TxDataRequestFlag|kLPSPI_RxDataReadyFlag);
```

Parameters

- base – LPSPI peripheral address.
- statusFlags – The status flag used from type _lpspi_flags.

static inline uint32_t LPSPI_GetTcr(LPSPI_Type *base)

static inline void LPSPI_EnableInterrupts(LPSPI_Type *base, uint32_t mask)

Enables the LPSPI interrupts.

This function configures the various interrupt masks of the LPSPI. The parameters are base and an interrupt mask. Note that, for Tx fill and Rx FIFO drain requests, enabling the interrupt request disables the DMA request.

```
LPSPI_EnableInterrupts(base, kLPSPI_TxInterruptEnable | kLPSPI_RxInterruptEnable );
```

Parameters

- base – LPSPI peripheral address.
- mask – The interrupt mask; Use the enum _lpspi_interrupt_enable.

static inline void LPSPI_DisableInterrupts(LPSPI_Type *base, uint32_t mask)

Disables the LPSPI interrupts.

```
LPSPI_DisableInterrupts(base, kLPSPI_TxInterruptEnable | kLPSPI_RxInterruptEnable );
```

Parameters

- base – LPSPI peripheral address.
- mask – The interrupt mask; Use the enum _lpspi_interrupt_enable.

static inline void LPSPI_EnableDMA(LPSPI_Type *base, uint32_t mask)

Enables the LPSPI DMA request.

This function configures the Rx and Tx DMA mask of the LPSPI. The parameters are base and a DMA mask.

```
LPSPI_EnableDMA(base, kLPSPI_TxDmaEnable | kLPSPI_RxDmaEnable);
```

Parameters

- base – LPSPI peripheral address.
- mask – The interrupt mask; Use the enum _lpspi_dma_enable.

static inline void LPSPI_DisableDMA(LPSPI_Type *base, uint32_t mask)

Disables the LPSPI DMA request.

This function configures the Rx and Tx DMA mask of the LPSPI. The parameters are base and a DMA mask.

```
SPI_DisableDMA(base, kLPSPI_TxDmaEnable | kLPSPI_RxDmaEnable);
```

Parameters

- base – LPSPI peripheral address.
- mask – The interrupt mask; Use the enum _lpspi_dma_enable.

static inline uint32_t LPSPI_GetTxRegisterAddress(LPSPI_Type *base)

Gets the LPSPI Transmit Data Register address for a DMA operation.

This function gets the LPSPI Transmit Data Register address because this value is needed for the DMA operation. This function can be used for either master or slave mode.

Parameters

- base – LPSPI peripheral address.

Returns
The LPSPI Transmit Data Register address.

static inline uint32_t LPSPI_GetRxRegisterAddress(LPSPI_Type *base)

Gets the LPSPI Receive Data Register address for a DMA operation.

This function gets the LPSPI Receive Data Register address because this value is needed for the DMA operation. This function can be used for either master or slave mode.

Parameters

- base – LPSPI peripheral address.

Returns
The LPSPI Receive Data Register address.

bool LPSPI_CheckTransferArgument(LPSPI_Type *base, *lpspi_transfer_t* *transfer, bool isEdma)

Check the argument for transfer .

Parameters

- base – LPSPI peripheral address.
- transfer – the transfer struct to be used.
- isEdma – True to check for EDMA transfer, false to check interrupt non-blocking transfer

Returns
Return true for right and false for wrong.

static inline void LPSPI_SetMasterSlaveMode(LPSPI_Type *base, *lpspi_master_slave_mode_t* mode)

Configures the LPSPI for either master or slave.

Note that the CFGR1 should only be written when the LPSPI is disabled (LPSPIx_CR_MEN = 0).

Parameters

- base – LPSPI peripheral address.
- mode – Mode setting (master or slave) of type lpspi_master_slave_mode_t.

static inline void LPSPI_SelectTransferPCS(LPSPI_Type *base, *lpspi_which_pcs_t* select)

Configures the peripheral chip select used for the transfer.

Parameters

- base – LPSPI peripheral address.
- select – LPSPI Peripheral Chip Select (PCS) configuration.

static inline void LPSPI_SetPCSContinous(LPSPI_Type *base, bool IsContinous)

Set the PCS signal to continuous or uncontinuous mode.

Note: In master mode, continuous transfer will keep the PCS asserted at the end of the frame size, until a command word is received that starts a new frame. So PCS must be set back to uncontinuous when transfer finishes. In slave mode, when continuous transfer

is enabled, the LPSPI will only transmit the first frame size bits, after that the LPSPI will transmit received data back (assuming a 32-bit shift register).

Parameters

- base – LPSPI peripheral address.
- IsContinous – True to set the transfer PCS to continuous mode, false to set to uncontinuous mode.

static inline bool LPSPI_IsMaster(LPSPI_Type *base)

Returns whether the LPSPI module is in master mode.

Parameters

- base – LPSPI peripheral address.

Returns

Returns true if the module is in master mode or false if the module is in slave mode.

static inline void LPSPI_FlushFifo(LPSPI_Type *base, bool flushTxFifo, bool flushRxFifo)

Flushes the LPSPI FIFOs.

Parameters

- base – LPSPI peripheral address.
- flushTxFifo – Flushes (true) the Tx FIFO, else do not flush (false) the Tx FIFO.
- flushRxFifo – Flushes (true) the Rx FIFO, else do not flush (false) the Rx FIFO.

static inline void LPSPI_SetFifoWatermarks(LPSPI_Type *base, uint32_t txWater, uint32_t rxWater)

Sets the transmit and receive FIFO watermark values.

This function allows the user to set the receive and transmit FIFO watermarks. The function does not compare the watermark settings to the FIFO size. The FIFO watermark should not be equal to or greater than the FIFO size. It is up to the higher level driver to make this check.

Parameters

- base – LPSPI peripheral address.
- txWater – The TX FIFO watermark value. Writing a value equal or greater than the FIFO size is truncated.
- rxWater – The RX FIFO watermark value. Writing a value equal or greater than the FIFO size is truncated.

static inline void LPSPI_SetAllPcsPolarity(LPSPI_Type *base, uint32_t mask)

Configures all LPSPI peripheral chip select polarities simultaneously.

Note that the CFGR1 should only be written when the LPSPI is disabled (LPSPIx_CR_MEN = 0).

This is an example: PCS0 and PCS1 set to active low and other PCSs set to active high. Note that the number of PCS is device-specific.

```
LPSPI_SetAllPcsPolarity(base, kLPSPI_Pcs0ActiveLow | kLPSPI_Pcs1ActiveLow);
```

Parameters

- base – LPSPI peripheral address.

- mask – The PCS polarity mask; Use the enum _lpspi_pcs_polarity.

static inline void LPSPI_SetFrameSize(LPSPI_Type *base, uint32_t frameSize)

Configures the frame size.

The minimum frame size is 8-bits and the maximum frame size is 4096-bits. If the frame size is less than or equal to 32-bits, the word size and frame size are identical. If the frame size is greater than 32-bits, the word size is 32-bits for each word except the last (the last word contains the remainder bits if the frame size is not divisible by 32). The minimum word size is 2-bits. A frame size of 33-bits (or similar) is not supported.

Note 1: The transmit command register should be initialized before enabling the LPSPI in slave mode, although the command register does not update until after the LPSPI is enabled. After it is enabled, the transmit command register should only be changed if the LPSPI is idle.

Note 2: The transmit and command FIFO is a combined FIFO that includes both transmit data and command words. That means the TCR register should be written to when the Tx FIFO is not full.

Parameters

- base – LPSPI peripheral address.
- frameSize – The frame size in number of bits.

uint32_t LPSPI_MasterSetBaudRate(LPSPI_Type *base, uint32_t baudRate_Bps, uint32_t srcClock_Hz, uint32_t *tcrPrescaleValue)

Sets the LPSPI baud rate in bits per second.

This function takes in the desired bitsPerSec (baud rate) and calculates the nearest possible baud rate without exceeding the desired baud rate and returns the calculated baud rate in bits-per-second. It requires the caller to provide the frequency of the module source clock (in Hertz). Note that the baud rate does not go into effect until the Transmit Control Register (TCR) is programmed with the prescale value. Hence, this function returns the prescale tcrPrescaleValue parameter for later programming in the TCR. The higher level peripheral driver should alert the user of an out of range baud rate input.

Note that the LPSPI module must first be disabled before configuring this. Note that the LPSPI module must be configured for master mode before configuring this.

Parameters

- base – LPSPI peripheral address.
- baudRate_Bps – The desired baud rate in bits per second.
- srcClock_Hz – Module source input clock in Hertz.
- tcrPrescaleValue – The TCR prescale value needed to program the TCR.

Returns

The actual calculated baud rate. This function may also return a "0" if the LPSPI is not configured for master mode or if the LPSPI module is not disabled.

void LPSPI_MasterSetDelayScaler(LPSPI_Type *base, uint32_t scaler, *lpspi_delay_type_t* whichDelay)

Manually configures a specific LPSPI delay parameter (module must be disabled to change the delay values).

This function configures the following: SCK to PCS delay, or PCS to SCK delay, or The configurations must occur between the transfer delay.

The delay names are available in type lpspi_delay_type_t.

The user passes the desired delay along with the delay value. This allows the user to directly set the delay values if they have pre-calculated them or if they simply wish to manually increment the value.

Note that the LPSPI module must first be disabled before configuring this. Note that the LPSPI module must be configured for master mode before configuring this.

Parameters

- base – LPSPI peripheral address.
- scaler – The 8-bit delay value 0x00 to 0xFF (255).
- whichDelay – The desired delay to configure, must be of type lpspi_delay_type_t.

uint32_t LPSPI_MasterSetDelayTimes(LPSPI_Type *base, uint32_t delayTimeInNanoSec, *lpspi_delay_type_t* whichDelay, uint32_t srcClock_Hz)

Calculates the delay based on the desired delay input in nanoseconds (module must be disabled to change the delay values).

This function configures the SCK to PCS delay, PCS to SCK delay, or the delay between transfers. The delay names are available in type lpspi_delay_type_t.

The function calculates the value needed for the desired delay parameter and returns the actual calculated delay. An exact delay match may not be possible, in which case the closest match is calculated without going below the desired delay value. If the input exceeds the maximum capability, the maximum supported delay is returned.

Note that the LPSPI module must first be disabled before configuring this. Note that the LPSPI module must be configured for master mode before configuring this.

Parameters

- base – LPSPI peripheral address.
- delayTimeInNanoSec – The desired delay value in nanoseconds.
- whichDelay – The desired delay to configure, must be of type lpspi_delay_type_t.
- srcClock_Hz – Module source input clock in Hertz.

Returns

Actual calculated delay value in nanoseconds.

static inline void LPSPI_WriteData(LPSPI_Type *base, uint32_t data)

Writes data into the transmit data buffer.

This function writes data passed in by the user to the Transmit Data Register (TDR). The user can pass up to 32-bits of data to load into the TDR. If the frame size exceeds 32-bits, the user has to manage sending the data one 32-bit word at a time. Any writes to the TDR result in an immediate push to the transmit FIFO. This function can be used for either master or slave modes.

Parameters

- base – LPSPI peripheral address.
- data – The data word to be sent.

static inline uint32_t LPSPI_ReadData(LPSPI_Type *base)

Reads data from the data buffer.

This function reads the data from the Receive Data Register (RDR). This function can be used for either master or slave mode.

Parameters

- base – LPSPI peripheral address.

Returns
The data read from the data buffer.

void LPSPI_SetDummyData(LPSPI_Type *base, uint8_t dummyData)

Set up the dummy data.

Parameters

- base – LPSPI peripheral address.
- dummyData – Data to be transferred when tx buffer is NULL. Note: This API has no effect when LPSPI in slave interrupt mode, because driver will set the TXMSK bit to 1 if txData is NULL, no data is loaded from transmit FIFO and output pin is tristated.

void LPSPI_MasterTransferCreateHandle(LPSPI_Type *base, *lpspi_master_handle_t* *handle, *lpspi_master_transfer_callback_t* callback, void *userData)

Initializes the LPSPI master handle.

This function initializes the LPSPI handle, which can be used for other LPSPI transactional APIs. Usually, for a specified LPSPI instance, call this API once to get the initialized handle.

Parameters

- base – LPSPI peripheral address.
- handle – LPSPI handle pointer to lpspi_master_handle_t.
- callback – DSPI callback.
- userData – callback function parameter.

status_t LPSPI_MasterTransferBlocking(LPSPI_Type *base, *lpspi_transfer_t* *transfer)

LPSPI master transfer data using a polling method.

This function transfers data using a polling method. This is a blocking function, which does not return until all transfers have been completed.

Note: The transfer data size should be integer multiples of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not integer multiples of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

Parameters

- base – LPSPI peripheral address.
- transfer – pointer to lpspi_transfer_t structure.

Returns
status of status_t.

status_t LPSPI_MasterTransferNonBlocking(LPSPI_Type *base, *lpspi_master_handle_t* *handle, *lpspi_transfer_t* *transfer)

LPSPI master transfer data using an interrupt method.

This function transfers data using an interrupt method. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Note: The transfer data size should be integer multiples of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not integer multiples of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to lpspi_master_handle_t structure which stores the transfer state.
- transfer – pointer to lpspi_transfer_t structure.

Returns
status of status_t.

status_t LPSPI_MasterTransferGetCount(LPSPI_Type *base, *lpspi_master_handle_t* *handle, size_t *count)

Gets the master transfer remaining bytes.

This function gets the master transfer remaining bytes.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to lpspi_master_handle_t structure which stores the transfer state.
- count – Number of bytes transferred so far by the non-blocking transaction.

Returns
status of status_t.

void LPSPI_MasterTransferAbort(LPSPI_Type *base, *lpspi_master_handle_t* *handle)

LPSPI master abort transfer which uses an interrupt method.

This function aborts a transfer which uses an interrupt method.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to lpspi_master_handle_t structure which stores the transfer state.

void LPSPI_MasterTransferHandleIRQ(LPSPI_Type *base, *lpspi_master_handle_t* *handle)

LPSPI Master IRQ handler function.

This function processes the LPSPI transmit and receive IRQ.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to lpspi_master_handle_t structure which stores the transfer state.

void LPSPI_SlaveTransferCreateHandle(LPSPI_Type *base, *lpspi_slave_handle_t* *handle, *lpspi_slave_transfer_callback_t* callback, void *userData)

Initializes the LPSPI slave handle.

This function initializes the LPSPI handle, which can be used for other LPSPI transactional APIs. Usually, for a specified LPSPI instance, call this API once to get the initialized handle.

Parameters

- base – LPSPI peripheral address.
- handle – LPSPI handle pointer to lpspi_slave_handle_t.
- callback – DSPI callback.
- userData – callback function parameter.

status_t LPSPI_SlaveTransferNonBlocking(LPSPI_Type *base, *lpspi_slave_handle_t* *handle, *lpspi_transfer_t* *transfer)

LPSPI slave transfer data using an interrupt method.

This function transfer data using an interrupt method. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Note: The transfer data size should be integer multiples of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not an integer multiple of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to lpspi_slave_handle_t structure which stores the transfer state.
- transfer – pointer to lpspi_transfer_t structure.

Returns
status of status_t.

status_t LPSPI_SlaveTransferGetCount(LPSPI_Type *base, *lpspi_slave_handle_t* *handle, size_t *count)

Gets the slave transfer remaining bytes.

This function gets the slave transfer remaining bytes.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to lpspi_slave_handle_t structure which stores the transfer state.
- count – Number of bytes transferred so far by the non-blocking transaction.

Returns
status of status_t.

void LPSPI_SlaveTransferAbort(LPSPI_Type *base, *lpspi_slave_handle_t* *handle)

LPSPI slave aborts a transfer which uses an interrupt method.

This function aborts a transfer which uses an interrupt method.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to lpspi_slave_handle_t structure which stores the transfer state.

void LPSPI_SlaveTransferHandleIRQ(LPSPI_Type *base, *lpspi_slave_handle_t* *handle)

LPSPI Slave IRQ handler function.

This function processes the LPSPI transmit and receives an IRQ.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to lpspi_slave_handle_t structure which stores the transfer state.

bool LPSPI_WaitTxFifoEmpty(LPSPI_Type *base)

Wait for tx FIFO to be empty.

This function wait the tx fifo empty

Parameters

- base – LPSPI peripheral address.

Returns

true for the tx FIFO is ready, false is not.

void LPSPI_DriverIRQHandler(uint32_t instance)

LPSPI driver IRQ handler common entry.

This function provides the common IRQ request entry for LPSPI.

Parameters

- instance – LPSPI instance.

FSL_LPSPI_DRIVER_VERSION

LPSPI driver version.

Status for the LPSPI driver.

Values:

enumerator kStatus_LPSPI_Busy

LPSPI transfer is busy.

enumerator kStatus_LPSPI_Error

LPSPI driver error.

enumerator kStatus_LPSPI_Idle

LPSPI is idle.

enumerator kStatus_LPSPI_OutOfRange

LPSPI transfer out Of range.

enumerator kStatus_LPSPI_Timeout

LPSPI timeout polling status flags.

enum _lpspi_flags

LPSPI status flags in SPIx_SR register.

Values:

enumerator kLPSPI_TxDataRequestFlag

Transmit data flag

enumerator kLPSPI_RxDataReadyFlag

Receive data flag

enumerator kLPSPI_WordCompleteFlag

Word Complete flag

enumerator kLPSPI_FrameCompleteFlag

Frame Complete flag

enumerator kLPSPI_TransferCompleteFlag
Transfer Complete flag

enumerator kLPSPI_TransmitErrorFlag
Transmit Error flag (FIFO underrun)

enumerator kLPSPI_ReceiveErrorFlag
Receive Error flag (FIFO overrun)

enumerator kLPSPI_DataMatchFlag
Data Match flag

enumerator kLPSPI_ModuleBusyFlag
Module Busy flag

enumerator kLPSPI_AllStatusFlag
Used for clearing all w1c status flags

enum _lpspi_interrupt_enable
LPSPI interrupt source.

Values:

enumerator kLPSPI_TxInterruptEnable
Transmit data interrupt enable

enumerator kLPSPI_RxInterruptEnable
Receive data interrupt enable

enumerator kLPSPI_WordCompleteInterruptEnable
Word complete interrupt enable

enumerator kLPSPI_FrameCompleteInterruptEnable
Frame complete interrupt enable

enumerator kLPSPI_TransferCompleteInterruptEnable
Transfer complete interrupt enable

enumerator kLPSPI_TransmitErrorInterruptEnable
Transmit error interrupt enable(FIFO underrun)

enumerator kLPSPI_ReceiveErrorInterruptEnable
Receive Error interrupt enable (FIFO overrun)

enumerator kLPSPI_DataMatchInterruptEnable
Data Match interrupt enable

enumerator kLPSPI_AllInterruptEnable
All above interrupts enable.

enum _lpspi_dma_enable

LPSPI DMA source.

Values:

enumerator kLPSPI_TxDmaEnable

Transmit data DMA enable

enumerator kLPSPI_RxDmaEnable

Receive data DMA enable

enum _lpspi_master_slave_mode

LPSPI master or slave mode configuration.

Values:

enumerator kLPSPI_Master

LPSPI peripheral operates in master mode.

enumerator kLPSPI_Slave

LPSPI peripheral operates in slave mode.

enum _lpspi_which_pcs_config

LPSPI Peripheral Chip Select (PCS) configuration (which PCS to configure).

Values:

enumerator kLPSPI_Pcs0

PCS[0]

enumerator kLPSPI_Pcs1

PCS[1]

enumerator kLPSPI_Pcs2

PCS[2]

enumerator kLPSPI_Pcs3

PCS[3]

enum _lpspi_pcs_polarity_config

LPSPI Peripheral Chip Select (PCS) Polarity configuration.

Values:

enumerator kLPSPI_PcsActiveHigh

PCS Active High (idles low)

enumerator kLPSPI_PcsActiveLow

PCS Active Low (idles high)

enum _lpspi_pcs_polarity

LPSPI Peripheral Chip Select (PCS) Polarity.

Values:

enumerator kLPSPI_Pcs0ActiveLow

Pcs0 Active Low (idles high).

enumerator kLPSPI_Pcs1ActiveLow

Pcs1 Active Low (idles high).

enumerator kLPSPI_Pcs2ActiveLow

Pcs2 Active Low (idles high).

enumerator kLPSPI_Pcs3ActiveLow

Pcs3 Active Low (idles high).

enumerator kLPSPI_PcsAllActiveLow

Pcs0 to Pcs5 Active Low (idles high).

enum _lpspi_clock_polarity

LPSPI clock polarity configuration.

Values:

enumerator kLPSPI_ClockPolarityActiveHigh

CPOL=0. Active-high LPSPI clock (idles low)

enumerator kLPSPI_ClockPolarityActiveLow

CPOL=1. Active-low LPSPI clock (idles high)

enum _lpspi_clock_phase

LPSPI clock phase configuration.

Values:

enumerator kLPSPI_ClockPhaseFirstEdge

CPHA=0. Data is captured on the leading edge of the SCK and changed on the following edge.

enumerator kLPSPI_ClockPhaseSecondEdge

CPHA=1. Data is changed on the leading edge of the SCK and captured on the following edge.

enum _lpspi_shift_direction

LPSPI data shifter direction options.

Values:

enumerator kLPSPI_MsbFirst

Data transfers start with most significant bit.

enumerator kLPSPI_LsbFirst
Data transfers start with least significant bit.

enum _lpspi_host_request_select
LPSPI Host Request select configuration.
Values:

enumerator kLPSPI_HostReqExtPin
Host Request is an ext pin.

enumerator kLPSPI_HostReqInternalTrigger
Host Request is an internal trigger.

enum _lpspi_match_config
LPSPI Match configuration options.
Values:

enumerator kLPSI_MatchDisabled
LPSPI Match Disabled.

enumerator kLPSI_1stWordEqualsM0orM1
LPSPI Match Enabled.

enumerator kLPSI_AnyWordEqualsM0orM1
LPSPI Match Enabled.

enumerator kLPSI_1stWordEqualsM0and2ndWordEqualsM1
LPSPI Match Enabled.

enumerator kLPSI_AnyWordEqualsM0andNxtWordEqualsM1
LPSPI Match Enabled.

enumerator kLPSI_1stWordAndM1EqualsM0andM1
LPSPI Match Enabled.

enumerator kLPSI_AnyWordAndM1EqualsM0andM1
LPSPI Match Enabled.

enum _lpspi_pin_config
LPSPI pin (SDO and SDI) configuration.
Values:

enumerator kLPSPI_SdiInSdoOut
LPSPI SDI input, SDO output.

enumerator kLPSPI_SdiInSdiOut
LPSPI SDI input, SDI output.

enumerator kLPSPI_SdoInSdoOut
LPSPI SDO input, SDO output.

enumerator kLPSPI_SdoInSdiOut
LPSPI SDO input, SDI output.

enum _lpspi_data_out_config
LPSPI data output configuration.

Values:

enumerator kLpspiDataOutRetained
Data out retains last value when chip select is de-asserted

enumerator kLpspiDataOutTristate
Data out is tristated when chip select is de-asserted

enum _lpspi_transfer_width
LPSPI transfer width configuration.

Values:

enumerator kLPSPI_SingleBitXfer
1-bit shift at a time, data out on SDO, in on SDI (normal mode)

enumerator kLPSPI_TwoBitXfer
2-bits shift out on SDO/SDI and in on SDO/SDI

enumerator kLPSPI_FourBitXfer
4-bits shift out on SDO/SDI/PCS[3:2] and in on SDO/SDI/PCS[3:2]

enum _lpspi_delay_type
LPSPI delay type selection.

Values:

enumerator kLPSPI_PcsToSck
PCS-to-SCK delay.

enumerator kLPSPI_LastSckToPcs
Last SCK edge to PCS delay.

enumerator kLPSPI_BetweenTransfer
Delay between transfers.

enum _lpspi_transfer_config_flag_for_master
Use this enumeration for LPSPI master transfer configFlags.

Values:

enumerator kLPSPI_MasterPcs0
LPSPI master PCS shift macro , internal used. LPSPI master transfer use PCS0 signal

enumerator kLPSPI_MasterPcs1

LPSPI master PCS shift macro , internal used. LPSPI master transfer use PCS1 signal

enumerator kLPSPI_MasterPcs2

LPSPI master PCS shift macro , internal used. LPSPI master transfer use PCS2 signal

enumerator kLPSPI_MasterPcs3

LPSPI master PCS shift macro , internal used. LPSPI master transfer use PCS3 signal

enumerator kLPSPI_MasterPcsContinuous

Is PCS signal continuous

enumerator kLPSPI_MasterByteSwap

Is master swap the byte. For example, when want to send data 1 2 3 4 5 6 7 8 (suppose you set lpspi_shift_direction_t to MSB).

i. If you set bitPerFrame = 8 , no matter the kLPSPI_MasterByteSwapyou flag is used or not, the waveform is 1 2 3 4 5 6 7 8.

ii. If you set bitPerFrame = 16 : (1) the waveform is 2 1 4 3 6 5 8 7 if you do not use the kLPSPI_MasterByteSwap flag. (2) the waveform is 1 2 3 4 5 6 7 8 if you use the kLPSPI_MasterByteSwap flag.

iii. If you set bitPerFrame = 32 : (1) the waveform is 4 3 2 1 8 7 6 5 if you do not use the kLPSPI_MasterByteSwap flag. (2) the waveform is 1 2 3 4 5 6 7 8 if you use the kLPSPI_MasterByteSwap flag.

enum _lpspi_transfer_config_flag_for_slave

Use this enumeration for LPSPI slave transfer configFlags.

Values:

enumerator kLPSPI_SlavePcs0

LPSPI slave PCS shift macro , internal used. LPSPI slave transfer use PCS0 signal

enumerator kLPSPI_SlavePcs1

LPSPI slave PCS shift macro , internal used. LPSPI slave transfer use PCS1 signal

enumerator kLPSPI_SlavePcs2

LPSPI slave PCS shift macro , internal used. LPSPI slave transfer use PCS2 signal

enumerator kLPSPI_SlavePcs3

LPSPI slave PCS shift macro , internal used. LPSPI slave transfer use PCS3 signal

enumerator kLPSPI_SlaveByteSwap

Is slave swap the byte. For example, when want to send data 1 2 3 4 5 6 7 8 (suppose you set lpspi_shift_direction_t to MSB).

i. If you set bitPerFrame = 8 , no matter the kLPSPI_SlaveByteSwap flag is used or not, the waveform is 1 2 3 4 5 6 7 8.

ii. If you set bitPerFrame = 16 : (1) the waveform is 2 1 4 3 6 5 8 7 if you do not use the kLPSPI_SlaveByteSwap flag. (2) the waveform is 1 2 3 4 5 6 7 8 if you use the kLPSPI_SlaveByteSwap flag.

iii. If you set bitPerFrame = 32 : (1) the waveform is 4 3 2 1 8 7 6 5 if you do not use the kLPSPI_SlaveByteSwap flag. (2) the waveform is 1 2 3 4 5 6 7 8 if you use the kLPSPI_SlaveByteSwap flag.

enum __lpspi_transfer_state

LPSPI transfer state, which is used for LPSPI transactional API state machine.

Values:

enumerator kLPSPI_Idle

Nothing in the transmitter/receiver.

enumerator kLPSPI_Busy

Transfer queue is not finished.

enumerator kLPSPI_Error

Transfer error.

typedef enum *_lpspi_master_slave_mode* lpspi_master_slave_mode_t

LPSPI master or slave mode configuration.

typedef enum *_lpspi_which_pcs_config* lpspi_which_pcs_t

LPSPI Peripheral Chip Select (PCS) configuration (which PCS to configure).

typedef enum *_lpspi_pcs_polarity_config* lpspi_pcs_polarity_config_t

LPSPI Peripheral Chip Select (PCS) Polarity configuration.

typedef enum *_lpspi_clock_polarity* lpspi_clock_polarity_t

LPSPI clock polarity configuration.

typedef enum *_lpspi_clock_phase* lpspi_clock_phase_t

LPSPI clock phase configuration.

typedef enum *_lpspi_shift_direction* lpspi_shift_direction_t

LPSPI data shifter direction options.

typedef enum *_lpspi_host_request_select* lpspi_host_request_select_t

LPSPI Host Request select configuration.

typedef enum *_lpspi_match_config* lpspi_match_config_t

LPSPI Match configuration options.

typedef enum *_lpspi_pin_config* lpspi_pin_config_t

LPSPI pin (SDO and SDI) configuration.

typedef enum *_lpspi_data_out_config* lpspi_data_out_config_t

LPSPI data output configuration.

typedef enum *_lpspi_transfer_width* lpspi_transfer_width_t

LPSPI transfer width configuration.

typedef enum *_lpspi_delay_type* lpspi_delay_type_t

LPSPI delay type selection.

typedef struct *_lpspi_master_config* lpspi_master_config_t

LPSPI master configuration structure.

typedef struct *_lpspi_slave_config* lpspi_slave_config_t

LPSPI slave configuration structure.

typedef struct *_lpspi_master_handle* lpspi_master_handle_t

Forward declaration of the *_lpspi_master_handle* typedefs.

typedef struct *_lpspi_slave_handle* lpspi_slave_handle_t

Forward declaration of the *_lpspi_slave_handle* typedefs.

typedef void (*lpspi_master_transfer_callback_t)(LPSPI_Type *base, *lpspi_master_handle_t* *handle, *status_t* status, void *userData)

Master completion callback function pointer type.

Param base
LPSPI peripheral address.

Param handle
Pointer to the handle for the LPSPI master.

Param status
Success or error code describing whether the transfer is completed.

Param userData
Arbitrary pointer-dataSized value passed from the application.

typedef void (*lpspi_slave_transfer_callback_t)(LPSPI_Type *base, *lpspi_slave_handle_t* *handle, *status_t* status, void *userData)

Slave completion callback function pointer type.

Param base
LPSPI peripheral address.

Param handle
Pointer to the handle for the LPSPI slave.

Param status
Success or error code describing whether the transfer is completed.

Param userData
Arbitrary pointer-dataSized value passed from the application.

typedef struct *_lpspi_transfer* lpspi_transfer_t

LPSPI master/slave transfer structure.

volatile uint8_t g_lpspiDummyData[]

Global variable for dummy data value setting.

LPSPI_DUMMY_DATA

LPSPI dummy data if no Tx data.

Dummy data used for tx if there is not txData.

SPI_RETRY_TIMES

Retry times for waiting flag.

LPSPI_MASTER_PCS_SHIFT

LPSPI master PCS shift macro , internal used.

LPSPI_MASTER_PCS_MASK

LPSPI master PCS shift macro , internal used.

LPSPI_SLAVE_PCS_SHIFT

LPSPI slave PCS shift macro , internal used.

LPSPI_SLAVE_PCS_MASK

LPSPI slave PCS shift macro , internal used.

struct _lpspi_master_config

#include <fsl_lpspi.h> LPSPI master configuration structure.

Public Members

uint32_t baudRate

Baud Rate for LPSPI.

uint32_t bitsPerFrame

Bits per frame, minimum 8, maximum 4096.

lpspi_clock_polarity_t cpol

Clock polarity.

lpspi_clock_phase_t cpha

Clock phase.

lpspi_shift_direction_t direction

MSB or LSB data shift direction.

uint32_t pcsToSckDelayInNanoSec

PCS to SCK delay time in nanoseconds, setting to 0 sets the minimum delay. It sets the boundary value if out of range.

uint32_t lastSckToPcsDelayInNanoSec

Last SCK to PCS delay time in nanoseconds, setting to 0 sets the minimum delay. It sets the boundary value if out of range.

uint32_t betweenTransferDelayInNanoSec

After the SCK delay time with nanoseconds, setting to 0 sets the minimum delay. It sets the boundary value if out of range.

lpspi_which_pcs_t whichPcs
Desired Peripheral Chip Select (PCS).

lpspi_pcs_polarity_config_t pcsActiveHighOrLow
Desired PCS active high or low

lpspi_pin_config_t pinCfg
Configures which pins are used for input and output data during single bit transfers.

lpspi_data_out_config_t dataOutConfig
Configures if the output data is tristated between accesses (LPSPI_PCS is negated).

bool enableInputDelay
Enable master to sample the input data on a delayed SCK. This can help improve slave setup time. Refer to device data sheet for specific time length.

struct __lpspi__slave__config
#include <fsl_lpspi.h> LPSPI slave configuration structure.

Public Members

uint32_t bitsPerFrame
Bits per frame, minimum 8, maximum 4096.

lpspi_clock_polarity_t cpol
Clock polarity.

lpspi_clock_phase_t cpha
Clock phase.

lpspi_shift_direction_t direction
MSB or LSB data shift direction.

lpspi_which_pcs_t whichPcs
Desired Peripheral Chip Select (pcs)

lpspi_pcs_polarity_config_t pcsActiveHighOrLow
Desired PCS active high or low

lpspi_pin_config_t pinCfg
Configures which pins are used for input and output data during single bit transfers.

lpspi_data_out_config_t dataOutConfig
Configures if the output data is tristated between accesses (LPSPI_PCS is negated).

struct __lpspi__transfer
#include <fsl_lpspi.h> LPSPI master/slave transfer structure.

Public Members

const uint8_t *txData
Send buffer.

uint8_t *rxData
Receive buffer.

volatile size_t dataSize
Transfer bytes.

uint32_t configFlags
Transfer transfer configuration flags. Set from _lpspi_transfer_config_flag_for_master if the transfer is used for master or _lpspi_transfer_config_flag_for_slave enumeration if the transfer is used for slave.

struct __lpspi__master__handle
#include <fsl_lpspi.h> LPSPI master transfer handle structure used for transactional API.

Public Members

volatile bool isPcsContinuous
Is PCS continuous in transfer.

volatile bool writeTcrInIsr
A flag that whether should write TCR in ISR.

volatile bool isByteSwap
A flag that whether should byte swap.

volatile bool isTxMask
A flag that whether TCR[TXMSK] is set.

volatile uint16_t bytesPerFrame
Number of bytes in each frame

volatile uint16_t frameSize
Backup of TCR[FRAMESZ]

volatile uint8_t fifoSize
FIFO dataSize.

volatile uint8_t rxWatermark
Rx watermark.

volatile uint8_t bytesEachWrite
Bytes for each write TDR.

volatile uint8_t bytesEachRead
Bytes for each read RDR.

const uint8_t *volatile txData
Send buffer.

uint8_t *volatile rxData
Receive buffer.

volatile size_t txRemainingByteCount
Number of bytes remaining to send.

volatile size_t rxRemainingByteCount
Number of bytes remaining to receive.

volatile uint32_t writeRegRemainingTimes
Write TDR register remaining times.

volatile uint32_t readRegRemainingTimes
Read RDR register remaining times.

uint32_t totalByteCount
Number of transfer bytes

uint32_t txBuffIfNull
Used if the txData is NULL.

volatile uint8_t state
LPSPI transfer state , _lpspi_transfer_state.

lpspi_master_transfer_callback_t callback
Completion callback.

void *userData
Callback user data.

struct __lpspi__slave__handle
#include <fsl_lpspi.h> LPSPI slave transfer handle structure used for transactional API.

Public Members

volatile bool isByteSwap
A flag that whether should byte swap.

volatile uint8_t fifoSize
FIFO dataSize.

volatile uint8_t rxWatermark
Rx watermark.

volatile uint8_t bytesEachWrite
Bytes for each write TDR.

volatile uint8_t bytesEachRead
Bytes for each read RDR.

const uint8_t *volatile txData
Send buffer.

uint8_t *volatile rxData
Receive buffer.

volatile size_t txRemainingByteCount
Number of bytes remaining to send.

volatile size_t rxRemainingByteCount
Number of bytes remaining to receive.

volatile uint32_t writeRegRemainingTimes
Write TDR register remaining times.

volatile uint32_t readRegRemainingTimes
Read RDR register remaining times.

uint32_t totalByteCount
Number of transfer bytes

volatile uint8_t state
LPSPI transfer state , _lpspi_transfer_state.

volatile uint32_t errorCount
Error count for slave transfer.

lpspi_slave_transfer_callback_t callback
Completion callback.

void *userData
Callback user data.

2.45 LPSPI eDMA Driver

FSL_LPSPI_EDMA_DRIVER_VERSION
LPSPI EDMA driver version.

DMA_MAX_TRANSFER_COUNT

DMA max transfer size.

typedef struct *_lpspi_master_edma_handle* lpspi_master_edma_handle_t

Forward declaration of the *_lpspi_master_edma_handle* typedefs.

typedef struct *_lpspi_slave_edma_handle* lpspi_slave_edma_handle_t

Forward declaration of the *_lpspi_slave_edma_handle* typedefs.

typedef void (*lpspi_master_edma_transfer_callback_t)(LPSPI_Type *base, *lpspi_master_edma_handle_t* *handle, *status_t* status, void *userData)

Completion callback function pointer type.

Param base
LPSPI peripheral base address.

Param handle
Pointer to the handle for the LPSPI master.

Param status
Success or error code describing whether the transfer completed.

Param userData
Arbitrary pointer-dataSized value passed from the application.

typedef void (*lpspi_slave_edma_transfer_callback_t)(LPSPI_Type *base, *lpspi_slave_edma_handle_t* *handle, *status_t* status, void *userData)

Completion callback function pointer type.

Param base
LPSPI peripheral base address.

Param handle
Pointer to the handle for the LPSPI slave.

Param status
Success or error code describing whether the transfer completed.

Param userData
Arbitrary pointer-dataSized value passed from the application.

void LPSPI_MasterTransferCreateHandleEDMA(LPSPI_Type *base, *lpspi_master_edma_handle_t* *handle, *lpspi_master_edma_transfer_callback_t* callback, void *userData, *edma_handle_t* *edmaRxRegToRxDataHandle, *edma_handle_t* *edmaTxDataToTxRegHandle)

Initializes the LPSPI master eDMA handle.

This function initializes the LPSPI eDMA handle which can be used for other LPSPI transactional APIs. Usually, for a specified LPSPI instance, call this API once to get the initialized handle.

Note that the LPSPI eDMA has a separated (Rx and Tx as two sources) or shared (Rx and Tx are the same source) DMA request source. (1) For a separated DMA request source, enable and set the Rx DMAMUX source for edmaRxRegToRxDataHandle and Tx DMAMUX source for edmaTxDataToTxRegHandle. (2) For a shared DMA request source, enable and set the Rx/Tx DMAMUX source for edmaRxRegToRxDataHandle.

Parameters

- base – LPSPI peripheral base address.

- handle – LPSPI handle pointer to lpspi_master_edma_handle_t.
- callback – LPSPI callback.
- userData – callback function parameter.
- edmaRxRegToRxDataHandle – edmaRxRegToRxDataHandle pointer to edma_handle_t.
- edmaTxDataToTxRegHandle – edmaTxDataToTxRegHandle pointer to edma_handle_t.

status_t LPSPI_MasterTransferEDMA(LPSPI_Type *base, *lpspi_master_edma_handle_t* *handle, *lpspi_transfer_t* *transfer)

LPSPI master transfer data using eDMA.

This function transfers data using eDMA. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Note: The transfer data size should be an integer multiple of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not an integer multiple of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

Parameters

- base – LPSPI peripheral base address.
- handle – pointer to lpspi_master_edma_handle_t structure which stores the transfer state.
- transfer – pointer to lpspi_transfer_t structure.

Returns

status of status_t.

status_t LPSPI_MasterTransferPrepareEDMALite(LPSPI_Type *base, *lpspi_master_edma_handle_t* *handle, uint32_t configFlags)

LPSPI master config transfer parameter while using eDMA.

This function is preparing to transfer data using eDMA, work with LPSPI_MasterTransferEDMALite.

Parameters

- base – LPSPI peripheral base address.
- handle – pointer to lpspi_master_edma_handle_t structure which stores the transfer state.
- configFlags – transfer configuration flags. *_lpspi_transfer_config_flag_for_master*.

Return values

- kStatus_Success – Execution successfully.
- kStatus_LPSPI_Busy – The LPSPI device is busy.

Returns

Indicates whether LPSPI master transfer was successful or not.

status_t LPSPI_MasterTransferEDMALite(LPSPI_Type *base, *lpspi_master_edma_handle_t* *handle, *lpspi_transfer_t* *transfer)

LPSPI master transfer data using eDMA without configs.

This function transfers data using eDMA. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Note: This API is only for transfer through DMA without configuration. Before calling this API, you must call LPSPI_MasterTransferPrepareEDMALite to configure it once. The transfer data size should be an integer multiple of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not an integer multiple of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

Parameters

- base – LPSPI peripheral base address.
- handle – pointer to lpspi_master_edma_handle_t structure which stores the transfer state.
- transfer – pointer to lpspi_transfer_t structure, config field is not uesed.

Return values

- kStatus_Success – Execution successfully.
- kStatus_LPSPI_Busy – The LPSPI device is busy.
- kStatus_InvalidArgument – The transfer structure is invalid.

Returns

Indicates whether LPSPI master transfer was successful or not.

void LPSPI_MasterTransferAbortEDMA(LPSPI_Type *base, *lpspi_master_edma_handle_t* *handle)

LPSPI master aborts a transfer which is using eDMA.

This function aborts a transfer which is using eDMA.

Parameters

- base – LPSPI peripheral base address.
- handle – pointer to lpspi_master_edma_handle_t structure which stores the transfer state.

status_t LPSPI_MasterTransferGetCountEDMA(LPSPI_Type *base, *lpspi_master_edma_handle_t* *handle, size_t *count)

Gets the master eDMA transfer remaining bytes.

This function gets the master eDMA transfer remaining bytes.

Parameters

- base – LPSPI peripheral base address.
- handle – pointer to lpspi_master_edma_handle_t structure which stores the transfer state.
- count – Number of bytes transferred so far by the EDMA transaction.

Returns

status of status_t.

void LPSPI_SlaveTransferCreateHandleEDMA(LPSPI_Type *base, *lpspi_slave_edma_handle_t* *handle, *lpspi_slave_edma_transfer_callback_t* callback, void *userData, *edma_handle_t* *edmaRxRegToRxDataHandle, *edma_handle_t* *edmaTxDataToTxRegHandle)

Initializes the LPSPI slave eDMA handle.

This function initializes the LPSPI eDMA handle which can be used for other LPSPI transactional APIs. Usually, for a specified LPSPI instance, call this API once to get the initialized handle.

Note that LPSPI eDMA has a separated (Rx and Tx as two sources) or shared (Rx and Tx as the same source) DMA request source.

(1) For a separated DMA request source, enable and set the Rx DMAMUX source for edmaRxRegToRxDataHandle and Tx DMAMUX source for edmaTxDataToTxRegHandle. (2) For a shared DMA request source, enable and set the Rx/Rx DMAMUX source for edmaRxRegToRxDataHandle .

Parameters

- base – LPSPI peripheral base address.
- handle – LPSPI handle pointer to lpspi_slave_edma_handle_t.
- callback – LPSPI callback.
- userData – callback function parameter.
- edmaRxRegToRxDataHandle – edmaRxRegToRxDataHandle pointer to edma_handle_t.
- edmaTxDataToTxRegHandle – edmaTxDataToTxRegHandle pointer to edma_handle_t.

status_t LPSPI_SlaveTransferEDMA(LPSPI_Type *base, *lpspi_slave_edma_handle_t* *handle, *lpspi_transfer_t* *transfer)

LPSPI slave transfers data using eDMA.

This function transfers data using eDMA. This is a non-blocking function, which return right away. When all data is transferred, the callback function is called.

Note: The transfer data size should be an integer multiple of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not an integer multiple of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

Parameters

- base – LPSPI peripheral base address.
- handle – pointer to lpspi_slave_edma_handle_t structure which stores the transfer state.
- transfer – pointer to lpspi_transfer_t structure.

Returns

status of status_t.

void LPSPI_SlaveTransferAbortEDMA(LPSPI_Type *base, *lpspi_slave_edma_handle_t* *handle)

LPSPI slave aborts a transfer which is using eDMA.

This function aborts a transfer which is using eDMA.

Parameters

- base – LPSPI peripheral base address.
- handle – pointer to lpspi_slave_edma_handle_t structure which stores the transfer state.

status_t LPSPI_SlaveTransferGetCountEDMA(LPSPI_Type *base, *lpspi_slave_edma_handle_t* *handle, size_t *count)

Gets the slave eDMA transfer remaining bytes.

This function gets the slave eDMA transfer remaining bytes.

Parameters

- base – LPSPI peripheral base address.

- handle – pointer to lpspi_slave_edma_handle_t structure which stores the transfer state.
- count – Number of bytes transferred so far by the eDMA transaction.

Returns
status of status_t.

struct __lpspi_master_edma_handle

#include <fsl_lpspi_edma.h> LPSPI master eDMA transfer handle structure used for transactional API.

Public Members

volatile bool isPcsContinuous
Is PCS continuous in transfer.

volatile bool isByteSwap
A flag that whether should byte swap.

volatile uint8_t fifoSize
FIFO dataSize.

volatile uint8_t rxWatermark
Rx watermark.

volatile uint8_t bytesEachWrite
Bytes for each write TDR.

volatile uint8_t bytesEachRead
Bytes for each read RDR.

volatile uint8_t bytesLastRead
Bytes for last read RDR.

volatile bool isThereExtraRxBytes
Is there extra RX byte.

const uint8_t *volatile txData
Send buffer.

uint8_t *volatile rxData
Receive buffer.

volatile size_t txRemainingByteCount
Number of bytes remaining to send.

volatile size_t rxRemainingByteCount
Number of bytes remaining to receive.

volatile uint32_t writeRegRemainingTimes
Write TDR register remaining times.

volatile uint32_t readRegRemainingTimes
Read RDR register remaining times.

uint32_t totalByteCount
Number of transfer bytes

edma_tcd_t *lastTimeTCD
Pointer to the lastTime TCD

bool isMultiDMATransmit
Is there multi DMA transmit

volatile uint8_t dmaTransmitTime
DMA Transfer times.

uint32_t lastTimeDataBytes
DMA transmit last Time data Bytes

uint32_t dataBytesEveryTime
Bytes in a time for DMA transfer, default is DMA_MAX_TRANSFER_COUNT

edma_transfer_config_t transferConfigRx
Config of DMA rx channel.

edma_transfer_config_t transferConfigTx
Config of DMA tx channel.

uint32_t txBuffIfNull
Used if there is not txData for DMA purpose.

uint32_t rxBuffIfNull
Used if there is not rxData for DMA purpose.

uint32_t transmitCommand
Used to write TCR for DMA purpose.

volatile uint8_t state
LPSPI transfer state , _lpspi_transfer_state.

uint8_t nbytes
eDMA minor byte transfer count initially configured.

lpspi_master_edma_transfer_callback_t callback
Completion callback.

void *userData
Callback user data.

edma_handle_t *edmaRxRegToRxDataHandle
edma_handle_t handle point used for RxReg to RxData buff

edma_handle_t *edmaTxDataToTxRegHandle
edma_handle_t handle point used for TxData to TxReg buff

edma_tcd_t lpspiSoftwareTCD[3]
SoftwareTCD, internal used

struct __lpspi_slave_edma_handle
#include <fsl_lpspi_edma.h> LPSPI slave eDMA transfer handle structure used for transactional API.

Public Members

volatile bool isByteSwap
A flag that whether should byte swap.

volatile uint8_t fifoSize
FIFO dataSize.

volatile uint8_t rxWatermark
Rx watermark.

volatile uint8_t bytesEachWrite
Bytes for each write TDR.

volatile uint8_t bytesEachRead
Bytes for each read RDR.

volatile uint8_t bytesLastRead
Bytes for last read RDR.

volatile bool isThereExtraRxBytes
Is there extra RX byte.

uint8_t nbytes
eDMA minor byte transfer count initially configured.

const uint8_t *volatile txData
Send buffer.

uint8_t *volatile rxData
Receive buffer.

volatile size_t txRemainingByteCount
Number of bytes remaining to send.

volatile size_t rxRemainingByteCount
Number of bytes remaining to receive.

volatile uint32_t writeRegRemainingTimes
Write TDR register remaining times.

volatile uint32_t readRegRemainingTimes
Read RDR register remaining times.

uint32_t totalByteCount
Number of transfer bytes

uint32_t txBuffIfNull
Used if there is not txData for DMA purpose.

uint32_t rxBuffIfNull
Used if there is not rxData for DMA purpose.

volatile uint8_t state
LPSPI transfer state.

uint32_t errorCount
Error count for slave transfer.

lpspi_slave_edma_transfer_callback_t callback
Completion callback.

void *userData
Callback user data.

edma_handle_t *edmaRxRegToRxDataHandle
edma_handle_t handle point used for RxReg to RxData buff

edma_handle_t *edmaTxDataToTxRegHandle
edma_handle_t handle point used for TxData to TxReg

edma_tcd_t lpspiSoftwareTCD[2]
SoftwareTCD, internal used

2.46 LPTMR: Low-Power Timer

void LPTMR_Init(LPTMR_Type *base, const *lptmr_config_t* *config)
Ungates the LPTMR clock and configures the peripheral for a basic operation.

Note: This API should be called at the beginning of the application using the LPTMR driver.

Parameters

- base – LPTMR peripheral base address
- config – A pointer to the LPTMR configuration structure.

void LPTMR_Deinit(LPTMR_Type *base)

Gates the LPTMR clock.

Parameters

- base – LPTMR peripheral base address

void LPTMR_GetDefaultConfig(*lptmr_config_t* *config)

Fills in the LPTMR configuration structure with default settings.

The default values are as follows.

```
config->timerMode = kLPTMR_TimerModeTimeCounter;
config->pinSelect = kLPTMR_PinSelectInput_0;
config->pinPolarity = kLPTMR_PinPolarityActiveHigh;
config->enableFreeRunning = false;
config->bypassPrescaler = true;
config->prescalerClockSource = kLPTMR_PrescalerClock_1;
config->value = kLPTMR_Prescale_Glitch_0;
```

Parameters

- config – A pointer to the LPTMR configuration structure.

static inline void LPTMR_EnableInterrupts(LPTMR_Type *base, uint32_t mask)

Enables the selected LPTMR interrupts.

Parameters

- base – LPTMR peripheral base address
- mask – The interrupts to enable. This is a logical OR of members of the enumeration *lptmr_interrupt_enable_t*

static inline void LPTMR_DisableInterrupts(LPTMR_Type *base, uint32_t mask)

Disables the selected LPTMR interrupts.

Parameters

- base – LPTMR peripheral base address
- mask – The interrupts to disable. This is a logical OR of members of the enumeration *lptmr_interrupt_enable_t*.

static inline uint32_t LPTMR_GetEnabledInterrupts(LPTMR_Type *base)

Gets the enabled LPTMR interrupts.

Parameters

- base – LPTMR peripheral base address

Returns

The enabled interrupts. This is the logical OR of members of the enumeration *lptmr_interrupt_enable_t*

static inline uint32_t LPTMR_GetStatusFlags(LPTMR_Type *base)

Gets the LPTMR status flags.

Parameters

- base – LPTMR peripheral base address

Returns

The status flags. This is the logical OR of members of the enumeration *lptmr_status_flags_t*

static inline void LPTMR_ClearStatusFlags(LPTMR_Type *base, uint32_t mask)

Clears the LPTMR status flags.

Parameters

- base – LPTMR peripheral base address
- mask – The status flags to clear. This is a logical OR of members of the enumeration *lptmr_status_flags_t*.

static inline void LPTMR_SetTimerPeriod(LPTMR_Type *base, uint32_t ticks)

Sets the timer period in units of count.

Timers counts from 0 until it equals the count value set here. The count value is written to the CMR register.

Note:

a. The TCF flag is set with the CNR equals the count provided here and then increments.

b. Call the utility macros provided in the fsl_common.h to convert to ticks.

Parameters

- base – LPTMR peripheral base address
- ticks – A timer period in units of ticks

static inline uint32_t LPTMR_GetCurrentTimerCount(LPTMR_Type *base)

Reads the current timer counting value.

This function returns the real-time timer counting value in a range from 0 to a timer period.

Note: Call the utility macros provided in the fsl_common.h to convert ticks to usec or msec.

Parameters

- base – LPTMR peripheral base address

Returns

The current counter value in ticks

static inline void LPTMR_StartTimer(LPTMR_Type *base)

Starts the timer.

After calling this function, the timer counts up to the CMR register value. Each time the timer reaches the CMR value and then increments, it generates a trigger pulse and sets the timeout interrupt flag. An interrupt is also triggered if the timer interrupt is enabled.

Parameters

- base – LPTMR peripheral base address

static inline void LPTMR_StopTimer(LPTMR_Type *base)

Stops the timer.

This function stops the timer and resets the timer's counter register.

Parameters

- base – LPTMR peripheral base address

FSL_LPTMR_DRIVER_VERSION

Driver Version

enum _lptmr_pin_select

LPTMR pin selection used in pulse counter mode.

Values:

enumerator kLPTMR_PinSelectInput_0

Pulse counter input 0 is selected

enumerator kLPTMR_PinSelectInput_1

Pulse counter input 1 is selected

enumerator kLPTMR_PinSelectInput_2

Pulse counter input 2 is selected

enumerator kLPTMR_PinSelectInput_3

Pulse counter input 3 is selected

enum _lptmr_pin_polarity

LPTMR pin polarity used in pulse counter mode.

Values:

enumerator kLPTMR_PinPolarityActiveHigh

Pulse Counter input source is active-high

enumerator kLPTMR_PinPolarityActiveLow

Pulse Counter input source is active-low

enum _lptmr_timer_mode

LPTMR timer mode selection.

Values:

enumerator kLPTMR_TimerModeTimeCounter

Time Counter mode

enumerator kLPTMR_TimerModePulseCounter

Pulse Counter mode

enum _lptmr_prescaler_glitch_value

LPTMR prescaler/glitch filter values.

Values:

enumerator kLPTMR_Prescale_Glitch_0
Prescaler divide 2, glitch filter does not support this setting

enumerator kLPTMR_Prescale_Glitch_1
Prescaler divide 4, glitch filter 2

enumerator kLPTMR_Prescale_Glitch_2
Prescaler divide 8, glitch filter 4

enumerator kLPTMR_Prescale_Glitch_3
Prescaler divide 16, glitch filter 8

enumerator kLPTMR_Prescale_Glitch_4
Prescaler divide 32, glitch filter 16

enumerator kLPTMR_Prescale_Glitch_5
Prescaler divide 64, glitch filter 32

enumerator kLPTMR_Prescale_Glitch_6
Prescaler divide 128, glitch filter 64

enumerator kLPTMR_Prescale_Glitch_7
Prescaler divide 256, glitch filter 128

enumerator kLPTMR_Prescale_Glitch_8
Prescaler divide 512, glitch filter 256

enumerator kLPTMR_Prescale_Glitch_9
Prescaler divide 1024, glitch filter 512

enumerator kLPTMR_Prescale_Glitch_10
Prescaler divide 2048 glitch filter 1024

enumerator kLPTMR_Prescale_Glitch_11
Prescaler divide 4096, glitch filter 2048

enumerator kLPTMR_Prescale_Glitch_12
Prescaler divide 8192, glitch filter 4096

enumerator kLPTMR_Prescale_Glitch_13
Prescaler divide 16384, glitch filter 8192

enumerator kLPTMR_Prescale_Glitch_14
Prescaler divide 32768, glitch filter 16384

enumerator kLPTMR_Prescale_Glitch_15
Prescaler divide 65536, glitch filter 32768

enum _lptmr_prescaler_clock_select

LPTMR prescaler/glitch filter clock select.

Note: Clock connections are SoC-specific

Values:

enum _lptmr_interrupt_enable

List of the LPTMR interrupts.

Values:

enumerator kLPTMR_TimerInterruptEnable

Timer interrupt enable

enum _lptmr_status_flags

List of the LPTMR status flags.

Values:

enumerator kLPTMR_TimerCompareFlag

Timer compare flag

typedef enum *_lptmr_pin_select* lptmr_pin_select_t

LPTMR pin selection used in pulse counter mode.

typedef enum *_lptmr_pin_polarity* lptmr_pin_polarity_t

LPTMR pin polarity used in pulse counter mode.

typedef enum *_lptmr_timer_mode* lptmr_timer_mode_t

LPTMR timer mode selection.

typedef enum *_lptmr_prescaler_glitch_value* lptmr_prescaler_glitch_value_t

LPTMR prescaler/glitch filter values.

typedef enum *_lptmr_prescaler_clock_select* lptmr_prescaler_clock_select_t

LPTMR prescaler/glitch filter clock select.

Note: Clock connections are SoC-specific

typedef enum *_lptmr_interrupt_enable* lptmr_interrupt_enable_t

List of the LPTMR interrupts.

typedef enum *_lptmr_status_flags* lptmr_status_flags_t

List of the LPTMR status flags.

typedef struct *_lptmr_config* lptmr_config_t

LPTMR config structure.

This structure holds the configuration settings for the LPTMR peripheral. To initialize this structure to reasonable defaults, call the *LPTMR_GetDefaultConfig()* function and pass a pointer to your configuration structure instance.

The configuration struct can be made constant so it resides in flash.

static inline void LPTMR_EnableTimerDMA(LPTMR_Type *base, bool enable)

Enable or disable timer DMA request.

Toggles CSR[TDRE] on the LPTMR side only: when enabled, every compare event (CSR[TCF]=1) raises an LPTMR DMA request line which is auto-cleared after the DMA controller services it.

Note: LPTMR runs on a low-power clock that is asynchronous to the EDMA bus clock. On some EDMA IP variants the channel has TWO gates that must BOTH be opened for an LPTMR request to be accepted: the base hardware request gate (ERQ, opened by *EDMA_EnableChannelRequest()*) and an extra asynchronous-request gate (opened by *EDMA_EnableAsyncRequest()* on EDMA4; the classic EDMA + DMAMUX combination does not need this).

Parameters

- base – base LPTMR peripheral base address
- enable – Switcher of timer DMA feature. "true" means to enable, "false" means to disable.

struct __lptmr_config

#include <fsl_lptmr.h> LPTMR config structure.

This structure holds the configuration settings for the LPTMR peripheral. To initialize this structure to reasonable defaults, call the *LPTMR_GetDefaultConfig()* function and pass a pointer to your configuration structure instance.

The configuration struct can be made constant so it resides in flash.

Public Members

lptmr_timer_mode_t timerMode

Time counter mode or pulse counter mode

lptmr_pin_select_t pinSelect

LPTMR pulse input pin select; used only in pulse counter mode

lptmr_pin_polarity_t pinPolarity

LPTMR pulse input pin polarity; used only in pulse counter mode

bool enableFreeRunning

True: enable free running, counter is reset on overflow False: counter is reset when the compare flag is set

bool bypassPrescaler

True: bypass prescaler; false: use clock from prescaler

lptmr_prescaler_clock_select_t prescalerClockSource

LPTMR clock source

lptmr_prescaler_glitch_value_t value

Prescaler or glitch filter value

2.47 LPUART: Low Power Universal Asynchronous Receiver/Transmitter Driver

2.48 LPUART Driver

static inline void LPUART_SoftwareReset(LPUART_Type *base)

Resets the LPUART using software.

This function resets all internal logic and registers except the Global Register. Remains set until cleared by software.

Parameters

- base – LPUART peripheral base address.

status_t LPUART_Init(LPUART_Type *base, const *lpuart_config_t* *config, uint32_t srcClock_Hz)

Initializes an LPUART instance with the user configuration structure and the peripheral clock.

This function configures the LPUART module with user-defined settings. Call the *LPUART_GetDefaultConfig()* function to configure the configuration structure and get the default configuration. The example below shows how to use this API to configure the LPUART.

```
lpuart_config_t lpuartConfig;
lpuartConfig.baudRate_Bps = 115200U;
lpuartConfig.parityMode = kLPUART_ParityDisabled;
lpuartConfig.dataBitsCount = kLPUART_EightDataBits;
lpuartConfig.isMsb = false;
lpuartConfig.stopBitCount = kLPUART_OneStopBit;
lpuartConfig.txFifoWatermark = 0;
lpuartConfig.rxFifoWatermark = 1;
LPUART_Init(LPUART1, &lpuartConfig, 20000000U);
```

Parameters

- base – LPUART peripheral base address.
- config – Pointer to a user-defined configuration structure.
- srcClock_Hz – LPUART clock source frequency in HZ.

Return values

- kStatus_LPUART_BaudrateNotSupport – Baudrate is not support in current clock source.
- kStatus_Success – LPUART initialize succeed

status_t LPUART_Deinit(LPUART_Type *base)

Deinitializes a LPUART instance.

This function waits for transmit to complete, disables TX and RX, and disables the LPUART clock.

Parameters

- base – LPUART peripheral base address.

Return values

- kStatus_Success – Deinit is success.
- kStatus_LPUART_Timeout – Timeout during deinit.

void LPUART_GetDefaultConfig(*lpuart_config_t* *config)

Gets the default configuration structure.

This function initializes the LPUART configuration structure to a default value. The default values are: lpuartConfig->baudRate_Bps = 115200U; lpuartConfig->parityMode = kLPUART_ParityDisabled; lpuartConfig->dataBitsCount = kLPUART_EightDataBits; lpuartConfig->isMsb = false; lpuartConfig->stopBitCount = kLPUART_OneStopBit; lpuartConfig->txFifoWatermark = 0; lpuartConfig->rxFifoWatermark = 1; lpuartConfig->rxIdleType = kLPUART_IdleTypeStartBit; lpuartConfig->rxIdleConfig = kLPUART_IdleCharacter1; lpuartConfig->enableTx = false; lpuartConfig->enableRx = false;

Parameters

- config – Pointer to a configuration structure.

status_t LPUART_SetBaudRate(LPUART_Type *base, uint32_t baudRate_Bps, uint32_t srcClock_Hz)

Sets the LPUART instance baudrate.

This function configures the LPUART module baudrate. This function is used to update the LPUART module baudrate after the LPUART module is initialized by the LPUART_Init.

```
LPUART_SetBaudRate(LPUART1, 115200U, 20000000U);
```

Parameters

- base – LPUART peripheral base address.
- baudRate_Bps – LPUART baudrate to be set.
- srcClock_Hz – LPUART clock source frequency in HZ.

Return values

- kStatus_LPUART_BaudrateNotSupport – Baudrate is not supported in the current clock source.
- kStatus_Success – Set baudrate succeeded.

void LPUART_Enable9bitMode(LPUART_Type *base, bool enable)

Enable 9-bit data mode for LPUART.

This function set the 9-bit mode for LPUART module. The 9th bit is not used for parity thus can be modified by user.

Parameters

- base – LPUART peripheral base address.
- enable – true to enable, flase to disable.

static inline void LPUART_SetMatchAddress(LPUART_Type *base, uint16_t address1, uint16_t address2)

Set the LPUART address.

This function configures the address for LPUART module that works as slave in 9-bit data mode. One or two address fields can be configured. When the address field's match enable bit is set, the frame it receices with MSB being 1 is considered as an address frame, otherwise it is considered as data frame. Once the address frame matches one of slave's own addresses, this slave is addressed. This address frame and its following data frames are stored in the receive buffer, otherwise the frames will be discarded. To un-address a slave, just send an address frame with unmatched address.

Note: Any LPUART instance joined in the multi-slave system can work as slave. The position of the address mark is the same as the parity bit when parity is enabled for 8 bit and 9 bit data formats.

Parameters

- base – LPUART peripheral base address.
- address1 – LPUART slave address1.
- address2 – LPUART slave address2.

static inline void LPUART_EnableMatchAddress(LPUART_Type *base, bool match1, bool match2)

Enable the LPUART match address feature.

Parameters

- base – LPUART peripheral base address.
- match1 – true to enable match address1, false to disable.
- match2 – true to enable match address2, false to disable.

static inline void LPUART_SetRxFifoWatermark(LPUART_Type *base, uint8_t water)

Sets the rx FIFO watermark.

Parameters

- base – LPUART peripheral base address.
- water – Rx FIFO watermark.

static inline void LPUART_SetTxFifoWatermark(LPUART_Type *base, uint8_t water)

Sets the tx FIFO watermark.

Parameters

- base – LPUART peripheral base address.
- water – Tx FIFO watermark.

static inline void LPUART_TransferEnable16Bit(*lpuart_handle_t* *handle, bool enable)

Sets the LPUART using 16bit transmit, only for 9bit or 10bit mode.

This function Enable 16bit Data transmit in lpuart_handle_t.

Parameters

- handle – LPUART handle pointer.
- enable – true to enable, false to disable.

uint32_t LPUART_GetStatusFlags(LPUART_Type *base)

Gets LPUART status flags.

This function gets all LPUART status flags. The flags are returned as the logical OR value of the enumerators *_lpuart_flags*. To check for a specific status, compare the return value with enumerators in the *_lpuart_flags*. For example, to check whether the TX is empty:

```
if (kLPUART_TxDataRegEmptyFlag & LPUART_GetStatusFlags(LPUART1))
{
    ...
}
```

Parameters

- base – LPUART peripheral base address.

Returns

LPUART status flags which are ORed by the enumerators in the _lpuart_flags.

status_t LPUART_ClearStatusFlags(LPUART_Type *base, uint32_t mask)

Clears status flags with a provided mask.

This function clears LPUART status flags with a provided mask. Automatically cleared flags can't be cleared by this function. Flags that can only cleared or set by hardware are: kLPUART_TxDataRegEmptyFlag, kLPUART_TransmissionCompleteFlag, kLPUART_RxDataRegFullFlag, kLPUART_RxActiveFlag, kLPUART_NoiseErrorFlag, kLPUART_ParityErrorFlag, kLPUART_TxFifoEmptyFlag,kLPUART_RxFifoEmptyFlag Note: This API should be called when the Tx/Rx is idle, otherwise it takes no effects.

Parameters

- base – LPUART peripheral base address.
- mask – the status flags to be cleared. The user can use the enumerators in the _lpuart_status_flag_t to do the OR operation and get the mask.

Return values

- kStatus_LPUART_FlagCannotClearManually – The flag can't be cleared by this function but it is cleared automatically by hardware.
- kStatus_Success – Status in the mask are cleared.

Returns

0 succeed, others failed.

void LPUART_EnableInterrupts(LPUART_Type *base, uint32_t mask)

Enables LPUART interrupts according to a provided mask.

This function enables the LPUART interrupts according to a provided mask. The mask is a logical OR of enumeration members. See the *_lpuart_interrupt_enable*. This examples shows how to enable TX empty interrupt and RX full interrupt:

```
LPUART_EnableInterrupts(LPUART1,kLPUART_TxDataRegEmptyInterruptEnable | kLPUART_
↪RxDataRegFullInterruptEnable);
```

Parameters

- base – LPUART peripheral base address.
- mask – The interrupts to enable. Logical OR of *_lpuart_interrupt_enable*.

void LPUART_DisableInterrupts(LPUART_Type *base, uint32_t mask)

Disables LPUART interrupts according to a provided mask.

This function disables the LPUART interrupts according to a provided mask. The mask is a logical OR of enumeration members. See *_lpuart_interrupt_enable*. This example shows how to disable the TX empty interrupt and RX full interrupt:

```
LPUART_DisableInterrupts(LPUART1,kLPUART_TxDataRegEmptyInterruptEnable | kLPUART_
↪RxDataRegFullInterruptEnable);
```

Parameters

- base – LPUART peripheral base address.
- mask – The interrupts to disable. Logical OR of *_lpuart_interrupt_enable*.

uint32_t LPUART_GetEnabledInterrupts(LPUART_Type *base)

Gets enabled LPUART interrupts.

This function gets the enabled LPUART interrupts. The enabled interrupts are returned as the logical OR value of the enumerators *_lpuart_interrupt_enable*. To check a specific interrupt enable status, compare the return value with enumerators in *_lpuart_interrupt_enable*. For example, to check whether the TX empty interrupt is enabled:

```
uint32_t enabledInterrupts = LPUART_GetEnabledInterrupts(LPUART1);

if (kLPUART_TxDataRegEmptyInterruptEnable & enabledInterrupts)
{
    ...
}
```

Parameters

- base – LPUART peripheral base address.

Returns

LPUART interrupt flags which are logical OR of the enumerators in *_lpuart_interrupt_enable*.

static inline uintptr_t LPUART_GetDataRegisterAddress(LPUART_Type *base)

Gets the LPUART data register address.

This function returns the LPUART data register address, which is mainly used by the DMA/eDMA.

Parameters

- base – LPUART peripheral base address.

Returns

LPUART data register addresses which are used both by the transmitter and receiver.

static inline void LPUART_EnableTxDMA(LPUART_Type *base, bool enable)

Enables or disables the LPUART transmitter DMA request.

This function enables or disables the transmit data register empty flag, STAT[TDRE], to generate DMA requests.

Parameters

- base – LPUART peripheral base address.
- enable – True to enable, false to disable.

static inline void LPUART_EnableRxDMA(LPUART_Type *base, bool enable)

Enables or disables the LPUART receiver DMA.

This function enables or disables the receiver data register full flag, STAT[RDRF], to generate DMA requests.

Parameters

- base – LPUART peripheral base address.
- enable – True to enable, false to disable.

uint32_t LPUART_GetInstance(LPUART_Type *base)

Get the LPUART instance from peripheral base address.

Parameters

- base – LPUART peripheral base address.

Returns

LPUART instance.

static inline void LPUART_EnableTx(LPUART_Type *base, bool enable)

Enables or disables the LPUART transmitter.

This function enables or disables the LPUART transmitter.

Parameters

- base – LPUART peripheral base address.
- enable – True to enable, false to disable.

static inline void LPUART_EnableRx(LPUART_Type *base, bool enable)

Enables or disables the LPUART receiver.

This function enables or disables the LPUART receiver.

Parameters

- base – LPUART peripheral base address.
- enable – True to enable, false to disable.

static inline void LPUART_WriteByte(LPUART_Type *base, uint8_t data)

Writes to the transmitter register.

This function writes data to the transmitter register directly. The upper layer must ensure that the TX register is empty or that the TX FIFO has room before calling this function.

Parameters

- base – LPUART peripheral base address.
- data – Data write to the TX register.

static inline uint8_t LPUART_ReadByte(LPUART_Type *base)

Reads the receiver register.

This function reads data from the receiver register directly. The upper layer must ensure that the receiver register is full or that the RX FIFO has data before calling this function.

Parameters

- base – LPUART peripheral base address.

Returns

Data read from data register.

static inline uint8_t LPUART_GetRxFifoCount(LPUART_Type *base)

Gets the rx FIFO data count.

Parameters

- base – LPUART peripheral base address.

Returns

rx FIFO data count.

static inline uint8_t LPUART_GetTxFifoCount(LPUART_Type *base)

Gets the tx FIFO data count.

Parameters

- base – LPUART peripheral base address.

Returns

tx FIFO data count.

void LPUART_SendAddress(LPUART_Type *base, uint8_t address)

Transmit an address frame in 9-bit data mode.

Parameters

- base – LPUART peripheral base address.
- address – LPUART slave address.

status_t LPUART_WriteBlocking(LPUART_Type *base, const uint8_t *data, size_t length)

Writes to the transmitter register using a blocking method.

This function polls the transmitter register, first waits for the register to be empty or TX FIFO to have room, and writes data to the transmitter buffer, then waits for the dat to be sent out to the bus.

Parameters

- base – LPUART peripheral base address.
- data – Start address of the data to write.
- length – Size of the data to write.

Return values

- kStatus_LPUART_Timeout – Transmission timed out and was aborted.
- kStatus_Success – Successfully wrote all data.

status_t LPUART_WriteBlocking16bit(LPUART_Type *base, const uint16_t *data, size_t length)

Writes to the transmitter register using a blocking method in 9bit or 10bit mode.

Note: This function only support 9bit or 10bit transfer. Please make sure only 10bit of data is valid and other bits are 0.

Parameters

- base – LPUART peripheral base address.
- data – Start address of the data to write.
- length – Size of the data to write.

Return values

- kStatus_LPUART_Timeout – Transmission timed out and was aborted.
- kStatus_Success – Successfully wrote all data.

status_t LPUART_ReadBlocking(LPUART_Type *base, uint8_t *data, size_t length)

Reads the receiver data register using a blocking method.

This function polls the receiver register, waits for the receiver register full or receiver FIFO has data, and reads data from the TX register.

Parameters

- base – LPUART peripheral base address.
- data – Start address of the buffer to store the received data.
- length – Size of the buffer.

Return values

- kStatus_LPUART_RxHardwareOverrun – Receiver overrun happened while receiving data.
- kStatus_LPUART_NoiseError – Noise error happened while receiving data.
- kStatus_LPUART_FramingError – Framing error happened while receiving data.
- kStatus_LPUART_ParityError – Parity error happened while receiving data.
- kStatus_LPUART_Timeout – Transmission timed out and was aborted.
- kStatus_Success – Successfully received all data.

status_t LPUART_ReadBlocking16bit(LPUART_Type *base, uint16_t *data, size_t length)

Reads the receiver data register in 9bit or 10bit mode.

Note: This function only support 9bit or 10bit transfer.

Parameters

- base – LPUART peripheral base address.
- data – Start address of the buffer to store the received data by 16bit, only 10bit is valid.
- length – Size of the buffer.

Return values

- kStatus_LPUART_RxHardwareOverrun – Receiver overrun happened while receiving data.
- kStatus_LPUART_NoiseError – Noise error happened while receiving data.
- kStatus_LPUART_FramingError – Framing error happened while receiving data.
- kStatus_LPUART_ParityError – Parity error happened while receiving data.
- kStatus_LPUART_Timeout – Transmission timed out and was aborted.
- kStatus_Success – Successfully received all data.

void LPUART_TransferCreateHandle(LPUART_Type *base, *lpuart_handle_t* *handle, *lpuart_transfer_callback_t* callback, void *userData)

Initializes the LPUART handle.

This function initializes the LPUART handle, which can be used for other LPUART transactional APIs. Usually, for a specified LPUART instance, call this API once to get the initialized handle.

The LPUART driver supports the "background" receiving, which means that user can set up an RX ring buffer optionally. Data received is stored into the ring buffer even when the user doesn't call the *LPUART_TransferReceiveNonBlocking()* API. If there is already data received in the ring buffer, the user can get the received data from the ring buffer directly. The ring buffer is disabled if passing NULL as ringBuffer.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.
- callback – Callback function.
- userData – User data.

status_t LPUART_TransferSendNonBlocking(LPUART_Type *base, *lpuart_handle_t* *handle, *lpuart_transfer_t* *xfer)

Transmits a buffer of data using the interrupt method.

This function send data using an interrupt method. This is a non-blocking function, which returns directly without waiting for all data written to the transmitter register. When all data is written to the TX register in the ISR, the LPUART driver calls the callback function and passes the *kStatus_LPUART_TxIdle* as status parameter.

Note: The kStatus_LPUART_TxIdle is passed to the upper layer when all data are written to the TX register. However, there is no check to ensure that all the data sent out. Before disabling the TX, check the kLPUART_TransmissionCompleteFlag to ensure that the transmit is finished.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.
- xfer – LPUART transfer structure, see *lpuart_transfer_t*.

Return values

- kStatus_Success – Successfully start the data transmission.
- kStatus_LPUART_TxBusy – Previous transmission still not finished, data not all written to the TX register.
- kStatus_InvalidArgument – Invalid argument.

void LPUART_TransferStartRingBuffer(LPUART_Type *base, *lpuart_handle_t* *handle, uint8_t *ringBuffer, size_t ringBufferSize)

Sets up the RX ring buffer.

This function sets up the RX ring buffer to a specific UART handle.

When the RX ring buffer is used, data received is stored into the ring buffer even when the user doesn't call the UART_TransferReceiveNonBlocking() API. If there is already data received in the ring buffer, the user can get the received data from the ring buffer directly.

Note: When using RX ring buffer, one byte is reserved for internal use. In other words, if ringBufferSize is 32, then only 31 bytes are used for saving data.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.
- ringBuffer – Start address of ring buffer for background receiving. Pass NULL to disable the ring buffer.
- ringBufferSize – size of the ring buffer.

void LPUART_TransferStopRingBuffer(LPUART_Type *base, *lpuart_handle_t* *handle)

Aborts the background transfer and uninstalls the ring buffer.

This function aborts the background transfer and uninstalls the ring buffer.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.

size_t LPUART_TransferGetRxRingBufferLength(LPUART_Type *base, *lpuart_handle_t* *handle)

Get the length of received data in RX ring buffer.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.

Returns

Length of received data in RX ring buffer.

void LPUART_TransferAbortSend(LPUART_Type *base, *lpuart_handle_t* *handle)

Aborts the interrupt-driven data transmit.

This function aborts the interrupt driven data sending. The user can get the remainBtyes to find out how many bytes are not sent out.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.

status_t LPUART_TransferGetSendCount(LPUART_Type *base, *lpuart_handle_t* *handle, uint32_t *count)

Gets the number of bytes that have been sent out to bus.

This function gets the number of bytes that have been sent out to bus by an interrupt method.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.
- count – Send bytes count.

Return values

- kStatus_NoTransferInProgress – No send in progress.
- kStatus_InvalidArgument – Parameter is invalid.
- kStatus_Success – Get successfully through the parameter count;

status_t LPUART_TransferReceiveNonBlocking(LPUART_Type *base, *lpuart_handle_t* *handle, *lpuart_transfer_t* *xfer, size_t *receivedBytes)

Receives a buffer of data using the interrupt method.

This function receives data using an interrupt method. This is a non-blocking function which returns without waiting to ensure that all data are received. If the RX ring buffer is used and not empty, the data in the ring buffer is copied and the parameter receivedBytes shows how many bytes are copied from the ring buffer. After copying, if the data in the ring buffer is not enough for read, the receive request is saved by the LPUART driver. When the new data arrives, the receive request is serviced first. When all data is received, the LPUART driver notifies the upper layer through a callback function and passes a status parameter kStatus_UART_RxIdle. For example, the upper layer needs 10 bytes but there are only 5 bytes in ring buffer. The 5 bytes are copied to xfer->data, which returns with the parameter receivedBytes set to 5. For the remaining 5 bytes, the newly arrived data is saved from xfer->data[5]. When 5 bytes are received, the LPUART driver notifies the upper layer. If the RX ring buffer is not enabled, this function enables the RX and RX interrupt to receive data to xfer->data. When all data is received, the upper layer is notified.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.
- xfer – LPUART transfer structure, see uart_transfer_t.
- receivedBytes – Bytes received from the ring buffer directly.

Return values

- kStatus_Success – Successfully queue the transfer into the transmit queue.
- kStatus_LPUART_RxBusy – Previous receive request is not finished.
- kStatus_InvalidArgument – Invalid argument.

void LPUART_TransferAbortReceive(LPUART_Type *base, *lpuart_handle_t* *handle)

Aborts the interrupt-driven data receiving.

This function aborts the interrupt-driven data receiving. The user can get the remainBytes to find out how many bytes not received yet.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.

status_t LPUART_TransferGetReceiveCount(LPUART_Type *base, *lpuart_handle_t* *handle, uint32_t *count)

Gets the number of bytes that have been received.

This function gets the number of bytes that have been received.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.
- count – Receive bytes count.

Return values

- kStatus_NoTransferInProgress – No receive in progress.
- kStatus_InvalidArgument – Parameter is invalid.
- kStatus_Success – Get successfully through the parameter count;

void LPUART_TransferHandleIRQ(LPUART_Type *base, void *irqHandle)

LPUART IRQ handle function.

This function handles the LPUART transmit and receive IRQ request.

Parameters

- base – LPUART peripheral base address.
- irqHandle – LPUART handle pointer.

void LPUART_TransferHandleErrorIRQ(LPUART_Type *base, void *irqHandle)

LPUART Error IRQ handle function.

This function handles the LPUART error IRQ request.

Parameters

- base – LPUART peripheral base address.
- irqHandle – LPUART handle pointer.

void LPUART_DriverIRQHandler(uint32_t instance)

LPUART driver IRQ handler common entry.

This function provides the common IRQ request entry for LPUART.

Parameters

- instance – LPUART instance.

FSL_LPUART_DRIVER_VERSION

LPUART driver version.

Error codes for the LPUART driver.

Values:

enumerator kStatus_LPUART_TxBusy

TX busy

enumerator kStatus_LPUART_RxBusy

RX busy

enumerator kStatus_LPUART_TxIdle

LPUART transmitter is idle.

enumerator kStatus_LPUART_RxIdle

LPUART receiver is idle.

enumerator kStatus_LPUART_TxWatermarkTooLarge

TX FIFO watermark too large

enumerator kStatus_LPUART_RxWatermarkTooLarge

RX FIFO watermark too large

enumerator kStatus_LPUART_FlagCannotClearManually

Some flag can't manually clear

enumerator kStatus_LPUART_Error
Error happens on LPUART.

enumerator kStatus_LPUART_RxRingBufferOverrun
LPUART RX software ring buffer overrun.

enumerator kStatus_LPUART_RxHardwareOverrun
LPUART RX receiver overrun.

enumerator kStatus_LPUART_NoiseError
LPUART noise error.

enumerator kStatus_LPUART_FramingError
LPUART framing error.

enumerator kStatus_LPUART_ParityError
LPUART parity error.

enumerator kStatus_LPUART_BaudrateNotSupport
Baudrate is not support in current clock source

enumerator kStatus_LPUART_IdleLineDetected
IDLE flag.

enumerator kStatus_LPUART_Timeout
LPUART times out.

enum _lpuart_parity_mode
LPUART parity mode.

Values:

enumerator kLPUART_ParityDisabled
Parity disabled

enumerator kLPUART_ParityEven
Parity enabled, type even, bit setting: PE|PT = 10

enumerator kLPUART_ParityOdd
Parity enabled, type odd, bit setting: PE|PT = 11

enum _lpuart_data_bits
LPUART data bits count.

Values:

enumerator kLPUART_EightDataBits
Eight data bit

enumerator kLPUART_SevenDataBits
Seven data bit

enum _lpuart_stop_bit_count
LPUART stop bit count.
Values:

enumerator kLPUART_OneStopBit
One stop bit

enumerator kLPUART_TwoStopBit
Two stop bits

enum _lpuart_transmit_cts_source
LPUART transmit CTS source.
Values:

enumerator kLPUART_CtsSourcePin
CTS resource is the LPUART_CTS pin.

enumerator kLPUART_CtsSourceMatchResult
CTS resource is the match result.

enum _lpuart_transmit_cts_config
LPUART transmit CTS configure.
Values:

enumerator kLPUART_CtsSampleAtStart
CTS input is sampled at the start of each character.

enumerator kLPUART_CtsSampleAtIdle
CTS input is sampled when the transmitter is idle

enum _lpuart_transmit_rts_polarity
LPUART transmitter RTS polarity.
Values:

enumerator kLPUART_RtsPolarityLow
Transmitter RTS is active low.

enumerator kLPUART_RtsPolarityHigh
Transmitter RTS is active high.

enum _lpuart_idle_type_select
LPUART idle flag type defines when the receiver starts counting.
Values:

enumerator kLPUART_IdleTypeStartBit
Start counting after a valid start bit.

enumerator kLPUART_IdleTypeStopBit
Start counting after a stop bit.

enum _lpuart_idle_config
LPUART idle detected configuration. This structure defines the number of idle characters that must be received before the IDLE flag is set.

Values:

enumerator kLPUART_IdleCharacter1
the number of idle characters.

enumerator kLPUART_IdleCharacter2
the number of idle characters.

enumerator kLPUART_IdleCharacter4
the number of idle characters.

enumerator kLPUART_IdleCharacter8
the number of idle characters.

enumerator kLPUART_IdleCharacter16
the number of idle characters.

enumerator kLPUART_IdleCharacter32
the number of idle characters.

enumerator kLPUART_IdleCharacter64
the number of idle characters.

enumerator kLPUART_IdleCharacter128
the number of idle characters.

enum _lpuart_interrupt_enable
LPUART interrupt configuration structure, default settings all disabled.

This structure contains the settings for all LPUART interrupt configurations.

Values:

enumerator kLPUART_LinBreakInterruptEnable
LIN break detect. bit 7

enumerator kLPUART_RxActiveEdgeInterruptEnable
Receive Active Edge. bit 6

enumerator kLPUART_TxDataRegEmptyInterruptEnable
Transmit data register empty. bit 23

enumerator kLPUART_TransmissionCompleteInterruptEnable
Transmission complete. bit 22

enumerator kLPUART_RxDataRegFullInterruptEnable
Receiver data register full. bit 21

enumerator kLPUART_IdleLineInterruptEnable
Idle line. bit 20

enumerator kLPUART_RxOverrunInterruptEnable
Receiver Overrun. bit 27

enumerator kLPUART_NoiseErrorInterruptEnable
Noise error flag. bit 26

enumerator kLPUART_FramingErrorInterruptEnable
Framing error flag. bit 25

enumerator kLPUART_ParityErrorInterruptEnable
Parity error flag. bit 24

enumerator kLPUART_Match1InterruptEnable
Parity error flag. bit 15

enumerator kLPUART_Match2InterruptEnable
Parity error flag. bit 14

enumerator kLPUART_TxFifoOverflowInterruptEnable
Transmit FIFO Overflow. bit 9

enumerator kLPUART_RxFifoUnderflowInterruptEnable
Receive FIFO Underflow. bit 8

enumerator kLPUART_AllInterruptEnable

enum _lpuart_flags
LPUART status flags.

This provides constants for the LPUART status flags for use in the LPUART functions.

Values:

enumerator kLPUART_TxDataRegEmptyFlag
Transmit data register empty flag, sets when transmit buffer is empty. bit 23

enumerator kLPUART_TransmissionCompleteFlag
Transmission complete flag, sets when transmission activity complete. bit 22

enumerator kLPUART_RxDataRegFullFlag
Receive data register full flag, sets when the receive data buffer is full. bit 21

enumerator kLPUART_IdleLineFlag

Idle line detect flag, sets when idle line detected. bit 20

enumerator kLPUART_RxOverrunFlag

Receive Overrun, sets when new data is received before data is read from receive register. bit 19

enumerator kLPUART_NoiseErrorFlag

Receive takes 3 samples of each received bit. If any of these samples differ, noise flag sets. bit 18

enumerator kLPUART_FramingErrorFlag

Frame error flag, sets if logic 0 was detected where stop bit expected. bit 17

enumerator kLPUART_ParityErrorFlag

If parity enabled, sets upon parity error detection. bit 16

enumerator kLPUART_LinBreakFlag

LIN break detect interrupt flag, sets when LIN break char detected and LIN circuit enabled. bit 31

enumerator kLPUART_RxActiveEdgeFlag

Receive pin active edge interrupt flag, sets when active edge detected. bit 30

enumerator kLPUART_RxActiveFlag

Receiver Active Flag (RAF), sets at beginning of valid start. bit 24

enumerator kLPUART_DataMatch1Flag

The next character to be read from LPUART_DATA matches MA1. bit 15

enumerator kLPUART_DataMatch2Flag

The next character to be read from LPUART_DATA matches MA2. bit 14

enumerator kLPUART_TxFifoEmptyFlag

TXEMPT bit, sets if transmit buffer is empty. bit 7

enumerator kLPUART_RxFifoEmptyFlag

RXEMPT bit, sets if receive buffer is empty. bit 6

enumerator kLPUART_TxFifoOverflowFlag

TXOF bit, sets if transmit buffer overflow occurred. bit 1

enumerator kLPUART_RxFifoUnderflowFlag

RXUF bit, sets if receive buffer underflow occurred. bit 0

enumerator kLPUART_AllClearFlags

enumerator kLPUART_AllFlags

typedef enum *_lpuart_parity_mode* lpuart_parity_mode_t
LPUART parity mode.

typedef enum *_lpuart_data_bits* lpuart_data_bits_t
LPUART data bits count.

typedef enum *_lpuart_stop_bit_count* lpuart_stop_bit_count_t
LPUART stop bit count.

typedef enum *_lpuart_transmit_cts_source* lpuart_transmit_cts_source_t
LPUART transmit CTS source.

typedef enum *_lpuart_transmit_cts_config* lpuart_transmit_cts_config_t
LPUART transmit CTS configure.

typedef enum *_lpuart_transmit_rts_polarity* lpuart_transmit_rts_polarity_t
LPUART transmitter RTS polarity.

typedef enum *_lpuart_idle_type_select* lpuart_idle_type_select_t
LPUART idle flag type defines when the receiver starts counting.

typedef enum *_lpuart_idle_config* lpuart_idle_config_t
LPUART idle detected configuration. This structure defines the number of idle characters that must be received before the IDLE flag is set.

typedef struct *_lpuart_config* lpuart_config_t
LPUART configuration structure.

typedef struct *_lpuart_transfer* lpuart_transfer_t
LPUART transfer structure.

typedef struct *_lpuart_handle* lpuart_handle_t

typedef void (*lpuart_transfer_callback_t)(LPUART_Type *base, *lpuart_handle_t* *handle, *status_t* status, void *userData)
LPUART transfer callback function.

typedef void (*lpuart_isr_t)(LPUART_Type *base, void *handle)

void *s_lpuartHandle[]

const IRQn_Type s_lpuartTxIRQ[]

lpuart_isr_t s_lpuartIsr[]

UART_RETRY_TIMES
Retry times for waiting flag.

struct _lpuart_config
#include <fsl_lpuart.h> LPUART configuration structure.

Public Members

uint32_t baudRate_Bps
LPUART baud rate

lpuart_parity_mode_t parityMode
Parity mode, disabled (default), even, odd

lpuart_data_bits_t dataBitsCount
Data bits count, eight (default), seven

bool isMsb
Data bits order, LSB (default), MSB

lpuart_stop_bit_count_t stopBitCount
Number of stop bits, 1 stop bit (default) or 2 stop bits

uint8_t txFifoWatermark
TX FIFO watermark

uint8_t rxFifoWatermark
RX FIFO watermark

bool enableRxRTS
RX RTS enable

bool enableTxRTS
TX RTS enable

bool enableTxCTS
TX CTS enable

lpuart_transmit_cts_source_t txCtsSource
TX CTS source

lpuart_transmit_cts_config_t txCtsConfig
TX CTS configure

lpuart_transmit_rts_polarity_t txRtsPolarity
TX RTS polarity

uint8_t rtsWatermark
RTS watermark

lpuart_idle_type_select_t rxIdleType
RX IDLE type.

lpuart_idle_config_t rxIdleConfig
RX IDLE configuration.

bool enableTx
Enable TX

bool enableRx
Enable RX

bool swapTxdRxd
Swap TXD and RXD pins

bool inverseTxd
Transmit Data Inversion - Setting true reverses the polarity of the transmitted data output

struct __lpuart_transfer
#include <fsl_lpuart.h> LPUART transfer structure.

Public Members

size_t dataSize
The byte count to be transfer.

struct __lpuart_handle
#include <fsl_lpuart.h> LPUART handle structure.

Public Members

volatile size_t txDataSize
Size of the remaining data to send.

size_t txDataSizeAll
Size of the data to send out.

volatile size_t rxDataSize
Size of the remaining data to receive.

size_t rxDataSizeAll
Size of the data to receive.

size_t rxRingBufferSize
Size of the ring buffer.

volatile uint16_t rxRingBufferHead
Index for the driver to store received data into ring buffer.

volatile uint16_t rxRingBufferTail
Index for the user to get data from the ring buffer.

lpuart_transfer_callback_t callback
Callback function.

void *userData
LPUART callback function parameter.

volatile uint8_t txState
TX transfer state.

volatile uint8_t rxState
RX transfer state.

bool isSevenDataBits
Seven data bits flag.

bool is16bitData
16bit data bits flag, only used for 9bit or 10bit data

union ___unnamed58___

Public Members

uint8_t *data
The buffer of data to be transfer.

uint8_t *rxData
The buffer to receive data.

uint16_t *rxData16
The buffer to receive data.

const uint8_t *txData
The buffer of data to be sent.

const uint16_t *txData16
The buffer of data to be sent.

union ___unnamed60___

Public Members

const uint8_t *volatile txData
Address of remaining data to send.

const uint16_t *volatile txData16
Address of remaining data to send.

union ___unnamed62___

Public Members

uint8_t *volatile rxData
Address of remaining data to receive.

uint16_t *volatile rxData16
Address of remaining data to receive.

union ___unnamed64___

Public Members

uint8_t *rxRingBuffer
Start address of the receiver ring buffer.

uint16_t *rxRingBuffer16
Start address of the receiver ring buffer.

2.49 LPUART eDMA Driver

void LPUART_TransferCreateHandleEDMA(LPUART_Type *base, *lpuart_edma_handle_t* *handle, *lpuart_edma_transfer_callback_t* callback, void *userData, *edma_handle_t* *txEdmaHandle, *edma_handle_t* *rxEdmaHandle)

Initializes the LPUART handle which is used in transactional functions.

Note: This function disables all LPUART interrupts.

Parameters

- base – LPUART peripheral base address.
- handle – Pointer to lpuart_edma_handle_t structure.
- callback – Callback function.
- userData – User data.
- txEdmaHandle – User requested DMA handle for TX DMA transfer.
- rxEdmaHandle – User requested DMA handle for RX DMA transfer.

status_t LPUART_SendEDMA(LPUART_Type *base, *lpuart_edma_handle_t* *handle, *lpuart_transfer_t* *xfer)

Sends data using eDMA.

This function sends data using eDMA. This is a non-blocking function, which returns right away. When all data is sent, the send callback function is called.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.
- xfer – LPUART eDMA transfer structure. See *lpuart_transfer_t*.

Return values

- kStatus_Success – if succeed, others failed.
- kStatus_LPUART_TxBusy – Previous transfer on going.
- kStatus_InvalidArgument – Invalid argument.

status_t LPUART_ReceiveEDMA(LPUART_Type *base, *lpuart_edma_handle_t* *handle, *lpuart_transfer_t* *xfer)

Receives data using eDMA.

This function receives data using eDMA. This is non-blocking function, which returns right away. When all data is received, the receive callback function is called.

Parameters

- base – LPUART peripheral base address.
- handle – Pointer to lpuart_edma_handle_t structure.
- xfer – LPUART eDMA transfer structure, see *lpuart_transfer_t*.

Return values

- kStatus_Success – if succeed, others fail.
- kStatus_LPUART_RxBusy – Previous transfer ongoing.
- kStatus_InvalidArgument – Invalid argument.

void LPUART_TransferAbortSendEDMA(LPUART_Type *base, *lpuart_edma_handle_t* *handle)

Aborts the sent data using eDMA.

This function aborts the sent data using eDMA.

Parameters

- base – LPUART peripheral base address.
- handle – Pointer to lpuart_edma_handle_t structure.

void LPUART_TransferAbortReceiveEDMA(LPUART_Type *base, *lpuart_edma_handle_t* *handle)

Aborts the received data using eDMA.

This function aborts the received data using eDMA.

Parameters

- base – LPUART peripheral base address.
- handle – Pointer to lpuart_edma_handle_t structure.

status_t LPUART_TransferGetSendCountEDMA(LPUART_Type *base, *lpuart_edma_handle_t* *handle, uint32_t *count)

Gets the number of bytes written to the LPUART TX register.

This function gets the number of bytes written to the LPUART TX register by DMA.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.

- count – Send bytes count.

Return values

- kStatus_NoTransferInProgress – No send in progress.
- kStatus_InvalidArgument – Parameter is invalid.
- kStatus_Success – Get successfully through the parameter count;

status_t LPUART_TransferGetReceiveCountEDMA(LPUART_Type *base, *lpuart_edma_handle_t* *handle, uint32_t *count)

Gets the number of received bytes.

This function gets the number of received bytes.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.
- count – Receive bytes count.

Return values

- kStatus_NoTransferInProgress – No receive in progress.
- kStatus_InvalidArgument – Parameter is invalid.
- kStatus_Success – Get successfully through the parameter count;

void LPUART_TransferEdmaHandleIRQ(LPUART_Type *base, void *lpuartEdmaHandle)

LPUART eDMA IRQ handle function.

This function handles the LPUART tx complete IRQ request and invoke user callback. It is not set to static so that it can be used in user application.

Note: This function is used as default IRQ handler by double weak mechanism. If user's specific IRQ handler is implemented, make sure this function is invoked in the handler.

Parameters

- base – LPUART peripheral base address.
- lpuartEdmaHandle – LPUART handle pointer.

FSL_LPUART_EDMA_DRIVER_VERSION

LPUART EDMA driver version.

typedef struct *_lpuart_edma_handle* lpuart_edma_handle_t

typedef void (*lpuart_edma_transfer_callback_t)(LPUART_Type *base, *lpuart_edma_handle_t* *handle, *status_t* status, void *userData)

LPUART transfer callback function.

struct _lpuart_edma_handle

#include <fsl_lpuart_edma.h> LPUART eDMA handle.

Public Members

lpuart_edma_transfer_callback_t callback
Callback function.

void *userData
LPUART callback function parameter.

size_t rxDataSizeAll
Size of the data to receive.

size_t txDataSizeAll
Size of the data to send out.

edma_handle_t *txEdmaHandle
The eDMA TX channel used.

edma_handle_t *rxEdmaHandle
The eDMA RX channel used.

uint8_t nbytes
eDMA minor byte transfer count initially configured.

volatile uint8_t txState
TX transfer state.

volatile uint8_t rxState
RX transfer state

2.50 LTC: LP Trusted Cryptography

FSL_LTC_DRIVER_VERSION
LTC driver version. Version 2.0.18.

Current version: 2.0.18

Change log:

- Version 2.0.1
  - fixed warning during g++ compilation
- Version 2.0.2
  - fixed [KPSDK-10932][LTC][SHA] *LTC_HASH()* blocks indefinitely when message size exceeds 4080 bytes
- Version 2.0.3
  - fixed *LTC_PKHA_CompareBigNum()* in case an integer argument is an array of all zeroes
- Version 2.0.4
  - constant *LTC_PKHA_CompareBigNum()* processing time

- Version 2.0.5
  - Fix MISRA issues
- Version 2.0.6
  - fixed [KPSDK-23603][LTC] AES Decrypt in ECB and CBC modes fail when ciphertext size > 0xff0 bytes
- Version 2.0.7
  - Fix MISRA-2012 issues
- Version 2.0.8
  - Fix Coverity issues
- Version 2.0.9
  - Fix sign-compare warning in ltc_set_context and in ltc_get_context
- Version 2.0.10
  - Fix MISRA-2012 issues
- Version 2.0.11
  - Fix MISRA-2012 issues
- Version 2.0.12
  - Fix AES Decrypt in CBC modes fail when used kLTC_DecryptKey.
- Version 2.0.13
  - Add feature macro FSL_FEATURE_LTC_HAS_NO_CLOCK_CONTROL_BIT into LTC_Init function.
- Version 2.0.14
  - Add feature macro FSL_FEATURE_LTC_HAS_NO_CLOCK_CONTROL_BIT into LTC_Deinit function.
- Version 2.0.15
  - Fix MISRA-2012 issues
- Version 2.0.16
  - Fix unitialized GCC warning in LTC_AES_GenerateDecryptKey()
- Version 2.0.17
  - Fix CMAC for payloads over one block, and if BRIC is present on the device, remove XCBC and "decrypt key" functionality
- Version 2.0.18
  - Fix CERT INT30-C and INT31-C compliance

void LTC_Init(LTC_Type *base)

Initializes the LTC driver. This function initializes the LTC driver.

Parameters

- base – LTC peripheral base address

void LTC_Deinit(LTC_Type *base)

Deinitializes the LTC driver. This function deinitializes the LTC driver.

Parameters

- base – LTC peripheral base address

void LTC_SetDpaMaskSeed(LTC_Type *base, uint32_t mask)

Sets the DPA Mask Seed register.

The DPA Mask Seed register reseeds the mask that provides resistance against DPA (differential power analysis) attacks on AES or DES keys.

Differential Power Analysis Mask (DPA) resistance uses a randomly changing mask that introduces "noise" into the power consumed by the AES or DES. This reduces the signal-to-noise ratio that differential power analysis attacks use to "guess" bits of the key. This randomly changing mask should be seeded at POR, and continues to provide DPA resistance from that point on. However, to provide even more DPA protection it is recommended that the DPA mask be reseeded after every 50,000 blocks have been processed. At that time, software can opt to write a new seed (preferably obtained from an RNG) into the DPA Mask Seed register (DPAMS), or software can opt to provide the new seed earlier or later, or not at all. DPA resistance continues even if the DPA mask is never reseeded.

Parameters

- base – LTC peripheral base address
- mask – The DPA mask seed.

2.51 LTC AES driver

enum _ltc_aes_key_t

Type of AES key for ECB and CBC decrypt operations.

Values:

enumerator kLTC_EncryptKey

Input key is an encrypt key

typedef enum *_ltc_aes_key_t* ltc_aes_key_t

Type of AES key for ECB and CBC decrypt operations.

status_t LTC_AES_EncryptEcb(LTC_Type *base, const uint8_t *plaintext, uint8_t *ciphertext, uint32_t size, const uint8_t *key, uint32_t keySize)

Encrypts AES using the ECB block mode.

Encrypts AES using the ECB block mode.

Parameters

- base – LTC peripheral base address
- plaintext – Input plain text to encrypt
- ciphertext – **[out]** Output cipher text
- size – Size of input and output data in bytes. Must be multiple of 16 bytes.
- key – Input key to use for encryption
- keySize – Size of the input key, in bytes. Must be 16, 24, or 32.

Returns

Status from encrypt operation

status_t LTC_AES_DecryptEcb(LTC_Type *base, const uint8_t *ciphertext, uint8_t *plaintext, uint32_t size, const uint8_t *key, uint32_t keySize, *ltc_aes_key_t* keyType)

Decrypts AES using ECB block mode.

Decrypts AES using ECB block mode.

Parameters

- base – LTC peripheral base address
- ciphertext – Input cipher text to decrypt
- plaintext – **[out]** Output plain text
- size – Size of input and output data in bytes. Must be multiple of 16 bytes.
- key – Input key.
- keySize – Size of the input key, in bytes. Must be 16, 24, or 32.
- keyType – Input type of the key (allows to directly load decrypt key for AES ECB decrypt operation.)

Returns

Status from decrypt operation

status_t LTC_AES_EncryptCbc(LTC_Type *base, const uint8_t *plaintext, uint8_t *ciphertext, uint32_t size, const uint8_t iv[16], const uint8_t *key, uint32_t keySize)

Encrypts AES using CBC block mode.

Parameters

- base – LTC peripheral base address
- plaintext – Input plain text to encrypt
- ciphertext – **[out]** Output cipher text
- size – Size of input and output data in bytes. Must be multiple of 16 bytes.
- iv – Input initial vector to combine with the first input block.
- key – Input key to use for encryption
- keySize – Size of the input key, in bytes. Must be 16, 24, or 32.

Returns

Status from encrypt operation

status_t LTC_AES_DecryptCbc(LTC_Type *base, const uint8_t *ciphertext, uint8_t *plaintext, uint32_t size, const uint8_t iv[16], const uint8_t *key, uint32_t keySize, *ltc_aes_key_t* keyType)

Decrypts AES using CBC block mode.

Parameters

- base – LTC peripheral base address
- ciphertext – Input cipher text to decrypt
- plaintext – **[out]** Output plain text
- size – Size of input and output data in bytes. Must be multiple of 16 bytes.
- iv – Input initial vector to combine with the first input block.
- key – Input key to use for decryption
- keySize – Size of the input key, in bytes. Must be 16, 24, or 32.
- keyType – Input type of the key (allows to directly load decrypt key for AES CBC decrypt operation.)

Returns
Status from decrypt operation

status_t LTC_AES_CryptCtr(LTC_Type *base, const uint8_t *input, uint8_t *output, uint32_t size, uint8_t counter[16U], const uint8_t *key, uint32_t keySize, uint8_t counterlast[16U], uint32_t *szLeft)

Encrypts or decrypts AES using CTR block mode.

Encrypts or decrypts AES using CTR block mode. AES CTR mode uses only forward AES cipher and same algorithm for encryption and decryption. The only difference between encryption and decryption is that, for encryption, the input argument is plain text and the output argument is cipher text. For decryption, the input argument is cipher text and the output argument is plain text.

Parameters

- base – LTC peripheral base address
- input – Input data for CTR block mode
- output – **[out]** Output data for CTR block mode
- size – Size of input and output data in bytes
- counter – **[inout]** Input counter (updates on return)
- key – Input key to use for forward AES cipher
- keySize – Size of the input key, in bytes. Must be 16, 24, or 32.
- counterlast – **[out]** Output cipher of last counter, for chained CTR calls. NULL can be passed if chained calls are not used.
- szLeft – **[out]** Output number of bytes in left unused in counterlast block. NULL can be passed if chained calls are not used.

Returns
Status from encrypt operation

status_t LTC_AES_EncryptTagGcm(LTC_Type *base, const uint8_t *plaintext, uint8_t *ciphertext, uint32_t size, const uint8_t *iv, uint32_t ivSize, const uint8_t *aad, uint32_t aadSize, const uint8_t *key, uint32_t keySize, uint8_t *tag, uint32_t tagSize)

Encrypts AES and tags using GCM block mode.

Encrypts AES and optionally tags using GCM block mode. If plaintext is NULL, only the GHASH is calculated and output in the 'tag' field.

Parameters

- base – LTC peripheral base address
- plaintext – Input plain text to encrypt
- ciphertext – **[out]** Output cipher text.
- size – Size of input and output data in bytes
- iv – Input initial vector
- ivSize – Size of the IV
- aad – Input additional authentication data
- aadSize – Input size in bytes of AAD
- key – Input key to use for encryption
- keySize – Size of the input key, in bytes. Must be 16, 24, or 32.
- tag – **[out]** Output hash tag. Set to NULL to skip tag processing.

- tagSize – Input size of the tag to generate, in bytes. Must be 4,8,12,13,14,15 or 16.

Returns
Status from encrypt operation

status_t LTC_AES_DecryptTagGcm(LTC_Type *base, const uint8_t *ciphertext, uint8_t *plaintext, uint32_t size, const uint8_t *iv, uint32_t ivSize, const uint8_t *aad, uint32_t aadSize, const uint8_t *key, uint32_t keySize, const uint8_t *tag, uint32_t tagSize)

Decrypts AES and authenticates using GCM block mode.

Decrypts AES and optionally authenticates using GCM block mode. If ciphertext is NULL, only the GHASH is calculated and compared with the received GHASH in 'tag' field.

Parameters

- base – LTC peripheral base address
- ciphertext – Input cipher text to decrypt
- plaintext – **[out]** Output plain text.
- size – Size of input and output data in bytes
- iv – Input initial vector
- ivSize – Size of the IV
- aad – Input additional authentication data
- aadSize – Input size in bytes of AAD
- key – Input key to use for encryption
- keySize – Size of the input key, in bytes. Must be 16, 24, or 32.
- tag – Input hash tag to compare. Set to NULL to skip tag processing.
- tagSize – Input size of the tag, in bytes. Must be 4, 8, 12, 13, 14, 15, or 16.

Returns
Status from decrypt operation

status_t LTC_AES_EncryptTagCcm(LTC_Type *base, const uint8_t *plaintext, uint8_t *ciphertext, uint32_t size, const uint8_t *iv, uint32_t ivSize, const uint8_t *aad, uint32_t aadSize, const uint8_t *key, uint32_t keySize, uint8_t *tag, uint32_t tagSize)

Encrypts AES and tags using CCM block mode.

Encrypts AES and optionally tags using CCM block mode.

Parameters

- base – LTC peripheral base address
- plaintext – Input plain text to encrypt
- ciphertext – **[out]** Output cipher text.
- size – Size of input and output data in bytes. Zero means authentication only.
- iv – Nonce
- ivSize – Length of the Nonce in bytes. Must be 7, 8, 9, 10, 11, 12, or 13.
- aad – Input additional authentication data. Can be NULL if aadSize is zero.
- aadSize – Input size in bytes of AAD. Zero means data mode only (authentication skipped).

- key – Input key to use for encryption
- keySize – Size of the input key, in bytes. Must be 16, 24, or 32.
- tag – **[out]** Generated output tag. Set to NULL to skip tag processing.
- tagSize – Input size of the tag to generate, in bytes. Must be 4, 6, 8, 10, 12, 14, or 16.

Returns
Status from encrypt operation

status_t LTC_AES_DecryptTagCcm(LTC_Type *base, const uint8_t *ciphertext, uint8_t *plaintext, uint32_t size, const uint8_t *iv, uint32_t ivSize, const uint8_t *aad, uint32_t aadSize, const uint8_t *key, uint32_t keySize, const uint8_t *tag, uint32_t tagSize)

Decrypts AES and authenticates using CCM block mode.

Decrypts AES and optionally authenticates using CCM block mode.

Parameters

- base – LTC peripheral base address
- ciphertext – Input cipher text to decrypt
- plaintext – **[out]** Output plain text.
- size – Size of input and output data in bytes. Zero means authentication only.
- iv – Nonce
- ivSize – Length of the Nonce in bytes. Must be 7, 8, 9, 10, 11, 12, or 13.
- aad – Input additional authentication data. Can be NULL if aadSize is zero.
- aadSize – Input size in bytes of AAD. Zero means data mode only (authentication skipped).
- key – Input key to use for decryption
- keySize – Size of the input key, in bytes. Must be 16, 24, or 32.
- tag – Received tag. Set to NULL to skip tag processing.
- tagSize – Input size of the received tag to compare with the computed tag, in bytes. Must be 4, 6, 8, 10, 12, 14, or 16.

Returns
Status from decrypt operation

LTC_AES_BLOCK_SIZE

AES block size in bytes

LTC_AES_IV_SIZE

AES Input Vector size in bytes

LTC_KEY_REGISTER_READABLE

LTC_AES_DecryptCtr(base, input, output, size, counter, key, keySize, counterlast, szLeft)

AES CTR decrypt is mapped to the AES CTR generic operation

LTC_AES_EncryptCtr(base, input, output, size, counter, key, keySize, counterlast, szLeft)

AES CTR encrypt is mapped to the AES CTR generic operation

2.52 LTC DES driver

status_t LTC_DES_EncryptEcb(LTC_Type *base, const uint8_t *plaintext, uint8_t *ciphertext, uint32_t size, const uint8_t key[8])

Encrypts DES using ECB block mode.

Encrypts DES using ECB block mode.

Parameters

- base – LTC peripheral base address
- plaintext – Input plaintext to encrypt
- ciphertext – **[out]** Output ciphertext
- size – Size of input and output data in bytes. Must be multiple of 8 bytes.
- key – Input key to use for encryption

Returns

Status from encrypt/decrypt operation

status_t LTC_DES_DecryptEcb(LTC_Type *base, const uint8_t *ciphertext, uint8_t *plaintext, uint32_t size, const uint8_t key[8])

Decrypts DES using ECB block mode.

Decrypts DES using ECB block mode.

Parameters

- base – LTC peripheral base address
- ciphertext – Input ciphertext to decrypt
- plaintext – **[out]** Output plaintext
- size – Size of input and output data in bytes. Must be multiple of 8 bytes.
- key – Input key to use for decryption

Returns

Status from encrypt/decrypt operation

status_t LTC_DES_EncryptCbc(LTC_Type *base, const uint8_t *plaintext, uint8_t *ciphertext, uint32_t size, const uint8_t iv[8], const uint8_t key[8])

Encrypts DES using CBC block mode.

Encrypts DES using CBC block mode.

Parameters

- base – LTC peripheral base address
- plaintext – Input plaintext to encrypt
- ciphertext – **[out]** Ouput ciphertext
- size – Size of input and output data in bytes
- iv – Input initial vector to combine with the first plaintext block. The iv does not need to be secret, but it must be unpredictable.
- key – Input key to use for encryption

Returns

Status from encrypt/decrypt operation

status_t LTC_DES_DecryptCbc(LTC_Type *base, const uint8_t *ciphertext, uint8_t *plaintext, uint32_t size, const uint8_t iv[8], const uint8_t key[8])

Decrypts DES using CBC block mode.

Decrypts DES using CBC block mode.

Parameters

- base – LTC peripheral base address
- ciphertext – Input ciphertext to decrypt
- plaintext – **[out]** Output plaintext
- size – Size of input data in bytes
- iv – Input initial vector to combine with the first plaintext block. The iv does not need to be secret, but it must be unpredictable.
- key – Input key to use for decryption

Returns

Status from encrypt/decrypt operation

status_t LTC_DES_EncryptCfb(LTC_Type *base, const uint8_t *plaintext, uint8_t *ciphertext, uint32_t size, const uint8_t iv[8], const uint8_t key[8])

Encrypts DES using CFB block mode.

Encrypts DES using CFB block mode.

Parameters

- base – LTC peripheral base address
- plaintext – Input plaintext to encrypt
- size – Size of input data in bytes
- iv – Input initial block.
- key – Input key to use for encryption
- ciphertext – **[out]** Output ciphertext

Returns

Status from encrypt/decrypt operation

status_t LTC_DES_DecryptCfb(LTC_Type *base, const uint8_t *ciphertext, uint8_t *plaintext, uint32_t size, const uint8_t iv[8], const uint8_t key[8])

Decrypts DES using CFB block mode.

Decrypts DES using CFB block mode.

Parameters

- base – LTC peripheral base address
- ciphertext – Input ciphertext to decrypt
- plaintext – **[out]** Output plaintext
- size – Size of input and output data in bytes
- iv – Input initial block.
- key – Input key to use for decryption

Returns

Status from encrypt/decrypt operation

status_t LTC_DES_EncryptOfb(LTC_Type *base, const uint8_t *plaintext, uint8_t *ciphertext, uint32_t size, const uint8_t iv[8], const uint8_t key[8])

Encrypts DES using OFB block mode.

Encrypts DES using OFB block mode.

Parameters

- base – LTC peripheral base address
- plaintext – Input plaintext to encrypt
- ciphertext – **[out]** Output ciphertext
- size – Size of input and output data in bytes
- iv – Input unique input vector. The OFB mode requires that the IV be unique for each execution of the mode under the given key.
- key – Input key to use for encryption

Returns

Status from encrypt/decrypt operation

status_t LTC_DES_DecryptOfb(LTC_Type *base, const uint8_t *ciphertext, uint8_t *plaintext, uint32_t size, const uint8_t iv[8], const uint8_t key[8])

Decrypts DES using OFB block mode.

Decrypts DES using OFB block mode.

Parameters

- base – LTC peripheral base address
- ciphertext – Input ciphertext to decrypt
- plaintext – **[out]** Output plaintext
- size – Size of input and output data in bytes. Must be multiple of 8 bytes.
- iv – Input unique input vector. The OFB mode requires that the IV be unique for each execution of the mode under the given key.
- key – Input key to use for decryption

Returns

Status from encrypt/decrypt operation

status_t LTC_DES2_EncryptEcb(LTC_Type *base, const uint8_t *plaintext, uint8_t *ciphertext, uint32_t size, const uint8_t key1[8], const uint8_t key2[8])

Encrypts triple DES using ECB block mode with two keys.

Encrypts triple DES using ECB block mode with two keys.

Parameters

- base – LTC peripheral base address
- plaintext – Input plaintext to encrypt
- ciphertext – **[out]** Output ciphertext
- size – Size of input and output data in bytes. Must be multiple of 8 bytes.
- key1 – First input key for key bundle
- key2 – Second input key for key bundle

Returns

Status from encrypt/decrypt operation

status_t LTC_DES2_DecryptEcb(LTC_Type *base, const uint8_t *ciphertext, uint8_t *plaintext, uint32_t size, const uint8_t key1[8], const uint8_t key2[8])

Decrypts triple DES using ECB block mode with two keys.

Decrypts triple DES using ECB block mode with two keys.

Parameters

- base – LTC peripheral base address
- ciphertext – Input ciphertext to decrypt
- plaintext – **[out]** Output plaintext
- size – Size of input and output data in bytes. Must be multiple of 8 bytes.
- key1 – First input key for key bundle
- key2 – Second input key for key bundle

Returns

Status from encrypt/decrypt operation

status_t LTC_DES2_EncryptCbc(LTC_Type *base, const uint8_t *plaintext, uint8_t *ciphertext, uint32_t size, const uint8_t iv[8], const uint8_t key1[8], const uint8_t key2[8])

Encrypts triple DES using CBC block mode with two keys.

Encrypts triple DES using CBC block mode with two keys.

Parameters

- base – LTC peripheral base address
- plaintext – Input plaintext to encrypt
- ciphertext – **[out]** Output ciphertext
- size – Size of input and output data in bytes
- iv – Input initial vector to combine with the first plaintext block. The iv does not need to be secret, but it must be unpredictable.
- key1 – First input key for key bundle
- key2 – Second input key for key bundle

Returns

Status from encrypt/decrypt operation

status_t LTC_DES2_DecryptCbc(LTC_Type *base, const uint8_t *ciphertext, uint8_t *plaintext, uint32_t size, const uint8_t iv[8], const uint8_t key1[8], const uint8_t key2[8])

Decrypts triple DES using CBC block mode with two keys.

Decrypts triple DES using CBC block mode with two keys.

Parameters

- base – LTC peripheral base address
- ciphertext – Input ciphertext to decrypt
- plaintext – **[out]** Output plaintext
- size – Size of input and output data in bytes
- iv – Input initial vector to combine with the first plaintext block. The iv does not need to be secret, but it must be unpredictable.
- key1 – First input key for key bundle

- key2 – Second input key for key bundle

Returns
Status from encrypt/decrypt operation

status_t LTC_DES2_EncryptCfb(LTC_Type *base, const uint8_t *plaintext, uint8_t *ciphertext, uint32_t size, const uint8_t iv[8], const uint8_t key1[8], const uint8_t key2[8])

Encrypts triple DES using CFB block mode with two keys.

Encrypts triple DES using CFB block mode with two keys.

Parameters

- base – LTC peripheral base address
- plaintext – Input plaintext to encrypt
- ciphertext – **[out]** Output ciphertext
- size – Size of input and output data in bytes
- iv – Input initial block.
- key1 – First input key for key bundle
- key2 – Second input key for key bundle

Returns
Status from encrypt/decrypt operation

status_t LTC_DES2_DecryptCfb(LTC_Type *base, const uint8_t *ciphertext, uint8_t *plaintext, uint32_t size, const uint8_t iv[8], const uint8_t key1[8], const uint8_t key2[8])

Decrypts triple DES using CFB block mode with two keys.

Decrypts triple DES using CFB block mode with two keys.

Parameters

- base – LTC peripheral base address
- ciphertext – Input ciphertext to decrypt
- plaintext – **[out]** Output plaintext
- size – Size of input and output data in bytes
- iv – Input initial block.
- key1 – First input key for key bundle
- key2 – Second input key for key bundle

Returns
Status from encrypt/decrypt operation

status_t LTC_DES2_EncryptOfb(LTC_Type *base, const uint8_t *plaintext, uint8_t *ciphertext, uint32_t size, const uint8_t iv[8], const uint8_t key1[8], const uint8_t key2[8])

Encrypts triple DES using OFB block mode with two keys.

Encrypts triple DES using OFB block mode with two keys.

Parameters

- base – LTC peripheral base address
- plaintext – Input plaintext to encrypt
- ciphertext – **[out]** Output ciphertext

- size – Size of input and output data in bytes
- iv – Input unique input vector. The OFB mode requires that the IV be unique for each execution of the mode under the given key.
- key1 – First input key for key bundle
- key2 – Second input key for key bundle

Returns
Status from encrypt/decrypt operation

status_t LTC_DES2_DecryptOfb(LTC_Type *base, const uint8_t *ciphertext, uint8_t *plaintext, uint32_t size, const uint8_t iv[8], const uint8_t key1[8], const uint8_t key2[8])

Decrypts triple DES using OFB block mode with two keys.

Decrypts triple DES using OFB block mode with two keys.

Parameters

- base – LTC peripheral base address
- ciphertext – Input ciphertext to decrypt
- plaintext – **[out]** Output plaintext
- size – Size of input and output data in bytes
- iv – Input unique input vector. The OFB mode requires that the IV be unique for each execution of the mode under the given key.
- key1 – First input key for key bundle
- key2 – Second input key for key bundle

Returns
Status from encrypt/decrypt operation

status_t LTC_DES3_EncryptEcb(LTC_Type *base, const uint8_t *plaintext, uint8_t *ciphertext, uint32_t size, const uint8_t key1[8], const uint8_t key2[8], const uint8_t key3[8])

Encrypts triple DES using ECB block mode with three keys.

Encrypts triple DES using ECB block mode with three keys.

Parameters

- base – LTC peripheral base address
- plaintext – Input plaintext to encrypt
- ciphertext – **[out]** Output ciphertext
- size – Size of input and output data in bytes. Must be multiple of 8 bytes.
- key1 – First input key for key bundle
- key2 – Second input key for key bundle
- key3 – Third input key for key bundle

Returns
Status from encrypt/decrypt operation

status_t LTC_DES3_DecryptEcb(LTC_Type *base, const uint8_t *ciphertext, uint8_t *plaintext, uint32_t size, const uint8_t key1[8], const uint8_t key2[8], const uint8_t key3[8])

Decrypts triple DES using ECB block mode with three keys.

Decrypts triple DES using ECB block mode with three keys.

Parameters

- base – LTC peripheral base address
- ciphertext – Input ciphertext to decrypt
- plaintext – **[out]** Output plaintext
- size – Size of input and output data in bytes. Must be multiple of 8 bytes.
- key1 – First input key for key bundle
- key2 – Second input key for key bundle
- key3 – Third input key for key bundle

Returns

Status from encrypt/decrypt operation

status_t LTC_DES3_EncryptCbc(LTC_Type *base, const uint8_t *plaintext, uint8_t *ciphertext, uint32_t size, const uint8_t iv[8], const uint8_t key1[8], const uint8_t key2[8], const uint8_t key3[8])

Encrypts triple DES using CBC block mode with three keys.

Encrypts triple DES using CBC block mode with three keys.

Parameters

- base – LTC peripheral base address
- plaintext – Input plaintext to encrypt
- ciphertext – **[out]** Output ciphertext
- size – Size of input data in bytes
- iv – Input initial vector to combine with the first plaintext block. The iv does not need to be secret, but it must be unpredictable.
- key1 – First input key for key bundle
- key2 – Second input key for key bundle
- key3 – Third input key for key bundle

Returns

Status from encrypt/decrypt operation

status_t LTC_DES3_DecryptCbc(LTC_Type *base, const uint8_t *ciphertext, uint8_t *plaintext, uint32_t size, const uint8_t iv[8], const uint8_t key1[8], const uint8_t key2[8], const uint8_t key3[8])

Decrypts triple DES using CBC block mode with three keys.

Decrypts triple DES using CBC block mode with three keys.

Parameters

- base – LTC peripheral base address
- ciphertext – Input ciphertext to decrypt
- plaintext – **[out]** Output plaintext
- size – Size of input and output data in bytes
- iv – Input initial vector to combine with the first plaintext block. The iv does not need to be secret, but it must be unpredictable.
- key1 – First input key for key bundle
- key2 – Second input key for key bundle
- key3 – Third input key for key bundle

Returns
Status from encrypt/decrypt operation

status_t LTC_DES3_EncryptCfb(LTC_Type *base, const uint8_t *plaintext, uint8_t *ciphertext, uint32_t size, const uint8_t iv[8], const uint8_t key1[8], const uint8_t key2[8], const uint8_t key3[8])

Encrypts triple DES using CFB block mode with three keys.

Encrypts triple DES using CFB block mode with three keys.

Parameters

- base – LTC peripheral base address
- plaintext – Input plaintext to encrypt
- ciphertext – **[out]** Output ciphertext
- size – Size of input and ouput data in bytes
- iv – Input initial block.
- key1 – First input key for key bundle
- key2 – Second input key for key bundle
- key3 – Third input key for key bundle

Returns
Status from encrypt/decrypt operation

status_t LTC_DES3_DecryptCfb(LTC_Type *base, const uint8_t *ciphertext, uint8_t *plaintext, uint32_t size, const uint8_t iv[8], const uint8_t key1[8], const uint8_t key2[8], const uint8_t key3[8])

Decrypts triple DES using CFB block mode with three keys.

Decrypts triple DES using CFB block mode with three keys.

Parameters

- base – LTC peripheral base address
- ciphertext – Input ciphertext to decrypt
- plaintext – **[out]** Output plaintext
- size – Size of input data in bytes
- iv – Input initial block.
- key1 – First input key for key bundle
- key2 – Second input key for key bundle
- key3 – Third input key for key bundle

Returns
Status from encrypt/decrypt operation

status_t LTC_DES3_EncryptOfb(LTC_Type *base, const uint8_t *plaintext, uint8_t *ciphertext, uint32_t size, const uint8_t iv[8], const uint8_t key1[8], const uint8_t key2[8], const uint8_t key3[8])

Encrypts triple DES using OFB block mode with three keys.

Encrypts triple DES using OFB block mode with three keys.

Parameters

- base – LTC peripheral base address
- plaintext – Input plaintext to encrypt

- ciphertext – **[out]** Output ciphertext
- size – Size of input and output data in bytes
- iv – Input unique input vector. The OFB mode requires that the IV be unique for each execution of the mode under the given key.
- key1 – First input key for key bundle
- key2 – Second input key for key bundle
- key3 – Third input key for key bundle

Returns
Status from encrypt/decrypt operation

status_t LTC_DES3_DecryptOfb(LTC_Type *base, const uint8_t *ciphertext, uint8_t *plaintext, uint32_t size, const uint8_t iv[8], const uint8_t key1[8], const uint8_t key2[8], const uint8_t key3[8])

Decrypts triple DES using OFB block mode with three keys.

Decrypts triple DES using OFB block mode with three keys.

Parameters

- base – LTC peripheral base address
- ciphertext – Input ciphertext to decrypt
- plaintext – **[out]** Output plaintext
- size – Size of input and output data in bytes
- iv – Input unique input vector. The OFB mode requires that the IV be unique for each execution of the mode under the given key.
- key1 – First input key for key bundle
- key2 – Second input key for key bundle
- key3 – Third input key for key bundle

Returns
Status from encrypt/decrypt operation

LTC_DES_KEY_SIZE

LTC DES key size - 64 bits.

LTC_DES_IV_SIZE

LTC DES IV size - 8 bytes.

2.53 LTC HASH driver

enum _ltc_hash_algo_t

Supported cryptographic block cipher functions for HASH creation

Values:

enumerator kLTC_Cmac

CMAC (AES engine)

enumerator kLTC_Sha1

SHA_1 (MDHA engine)

enumerator kLTC_Sha224

SHA_224 (MDHA engine)

enumerator kLTC_Sha256

SHA_256 (MDHA engine)

typedef enum *_ltc_hash_algo_t* ltc_hash_algo_t

Supported cryptographic block cipher functions for HASH creation

typedef struct *_ltc_hash_ctx_t* ltc_hash_ctx_t

Storage type used to save hash context.

status_t LTC_HASH_Init(LTC_Type *base, *ltc_hash_ctx_t* *ctx, *ltc_hash_algo_t* algo, const uint8_t *key, uint32_t keySize)

Initialize HASH context.

This function initialize the HASH. Key shall be supplied if the underlaying algoritm is AES XCBC-MAC or CMAC. Key shall be NULL if the underlaying algoritm is SHA.

For XCBC-MAC, the key length must be 16. For CMAC, the key length can be the AES key lengths supported by AES engine. For MDHA the key length argument is ignored.

Parameters

- base – LTC peripheral base address
- ctx – **[out]** Output hash context
- algo – Underlaying algorithm to use for hash computation.
- key – Input key (NULL if underlaying algorithm is SHA)
- keySize – Size of input key in bytes

Returns

Status of initialization

status_t LTC_HASH_Update(*ltc_hash_ctx_t* *ctx, const uint8_t *input, uint32_t inputSize)

Add data to current HASH.

Add data to current HASH. This can be called repeatedly with an arbitrary amount of data to be hashed.

Parameters

- ctx – **[inout]** HASH context
- input – Input data
- inputSize – Size of input data in bytes

Returns

Status of the hash update operation

status_t LTC_HASH_Finish(*ltc_hash_ctx_t* *ctx, uint8_t *output, uint32_t *outputSize)

Finalize hashing.

Outputs the final hash and erases the context.

Parameters

- ctx – **[inout]** Input hash context

- output – **[out]** Output hash data
- outputSize – **[out]** Output parameter storing the size of the output hash in bytes

Returns
Status of the hash finish operation

status_t LTC_HASH(LTC_Type *base, *ltc_hash_algo_t* algo, const uint8_t *input, uint32_t inputSize, const uint8_t *key, uint32_t keySize, uint8_t *output, uint32_t *outputSize)

Create HASH on given data.

Perform the full keyed HASH in one function call.

Parameters

- base – LTC peripheral base address
- algo – Block cipher algorithm to use for CMAC creation
- input – Input data
- inputSize – Size of input data in bytes
- key – Input key
- keySize – Size of input key in bytes
- output – **[out]** Output hash data
- outputSize – **[out]** Output parameter storing the size of the output hash in bytes

Returns
Status of the one call hash operation.

LTC_HASH_CTX_SIZE

LTC HASH Context size.

struct _ltc_hash_ctx_t

#include <fsl_ltc.h> Storage type used to save hash context.

2.54 LTC PKHA driver

enum _ltc_pkha_timing_t

Use of timing equalized version of a PKHA function.

Values:

enumerator kLTC_PKHA_NoTimingEqualized

Normal version of a PKHA operation

enumerator kLTC_PKHA_TimingEqualized

Timing-equalized version of a PKHA operation

enum _ltc_pkha_f2m_t

Integer vs binary polynomial arithmetic selection.

Values:

enumerator kLTC_PKHA_IntegerArith
Use integer arithmetic

enumerator kLTC_PKHA_F2mArith
Use binary polynomial arithmetic

enum _ltc_pkha_montgomery_form_t
Montgomery or normal PKHA input format.

Values:

enumerator kLTC_PKHA_NormalValue
PKHA number is normal integer

enumerator kLTC_PKHA_MontgomeryFormat
PKHA number is in montgomery format

typedef struct *_ltc_pkha_ecc_point_t* ltc_pkha_ecc_point_t
PKHA ECC point structure

typedef enum *_ltc_pkha_timing_t* ltc_pkha_timing_t
Use of timing equalized version of a PKHA function.

typedef enum *_ltc_pkha_f2m_t* ltc_pkha_f2m_t
Integer vs binary polynomial arithmetic selection.

typedef enum *_ltc_pkha_montgomery_form_t* ltc_pkha_montgomery_form_t
Montgomery or normal PKHA input format.

int LTC_PKHA_CompareBigNum(const uint8_t *a, size_t sizeA, const uint8_t *b, size_t sizeB)
Compare two PKHA big numbers.

Compare two PKHA big numbers. Return 1 for a > b, -1 for a < b and 0 if they are same. PKHA big number is lsbyte first. Thus the comparison starts at msbyte which is the last member of tested arrays.

Parameters

- a – First integer represented as an array of bytes, lsbyte first.
- sizeA – Size in bytes of the first integer.
- b – Second integer represented as an array of bytes, lsbyte first.
- sizeB – Size in bytes of the second integer.

Returns
1 if a > b.

Returns
-1 if a < b.

Returns
0 if a = b.

status_t LTC_PKHA_NormalToMontgomery(LTC_Type *base, const uint8_t *N, uint16_t sizeN, uint8_t *A, uint16_t *sizeA, uint8_t *B, uint16_t *sizeB, uint8_t *R2, uint16_t *sizeR2, *ltc_pkha_timing_t* equalTime, *ltc_pkha_f2m_t* arithType)

Converts from integer to Montgomery format.

This function computes R2 mod N and optionally converts A or B into Montgomery format of A or B.

Parameters

- base – LTC peripheral base address
- N – modulus
- sizeN – size of N in bytes
- A – **[inout]** The first input in non-Montgomery format. Output Montgomery format of the first input.
- sizeA – **[inout]** pointer to size variable. On input it holds size of input A in bytes. On output it holds size of Montgomery format of A in bytes.
- B – **[inout]** Second input in non-Montgomery format. Output Montgomery format of the second input.
- sizeB – **[inout]** pointer to size variable. On input it holds size of input B in bytes. On output it holds size of Montgomery format of B in bytes.
- R2 – **[out]** Output Montgomery factor R2 mod N.
- sizeR2 – **[out]** pointer to size variable. On output it holds size of Montgomery factor R2 mod N in bytes.
- equalTime – Run the function time equalized or no timing equalization.
- arithType – Type of arithmetic to perform (integer or F2m)

Returns

Operation status.

status_t LTC_PKHA_MontgomeryToNormal(LTC_Type *base, const uint8_t *N, uint16_t sizeN, uint8_t *A, uint16_t *sizeA, uint8_t *B, uint16_t *sizeB, *ltc_pkha_timing_t* equalTime, *ltc_pkha_f2m_t* arithType)

Converts from Montgomery format to int.

This function converts Montgomery format of A or B into int A or B.

Parameters

- base – LTC peripheral base address
- N – modulus.
- sizeN – size of N modulus in bytes.
- A – **[inout]** Input first number in Montgomery format. Output is non-Montgomery format.
- sizeA – **[inout]** pointer to size variable. On input it holds size of the input A in bytes. On output it holds size of non-Montgomery A in bytes.
- B – **[inout]** Input first number in Montgomery format. Output is non-Montgomery format.
- sizeB – **[inout]** pointer to size variable. On input it holds size of the input B in bytes. On output it holds size of non-Montgomery B in bytes.
- equalTime – Run the function time equalized or no timing equalization.
- arithType – Type of arithmetic to perform (integer or F2m)

Returns
Operation status.

status_t LTC_PKHA_ModAdd(LTC_Type *base, const uint8_t *A, uint16_t sizeA, const uint8_t *B, uint16_t sizeB, const uint8_t *N, uint16_t sizeN, uint8_t *result, uint16_t *resultSize, *ltc_pkha_f2m_t* arithType)

Performs modular addition - (A + B) mod N.

This function performs modular addition of (A + B) mod N, with either integer or binary polynomial (F2m) inputs. In the F2m form, this function is equivalent to a bitwise XOR and it is functionally the same as subtraction.

Parameters

- base – LTC peripheral base address
- A – first addend (integer or binary polynomial)
- sizeA – Size of A in bytes
- B – second addend (integer or binary polynomial)
- sizeB – Size of B in bytes
- N – modulus. For F2m operation this can be NULL, as N is ignored during F2m polynomial addition.
- sizeN – Size of N in bytes. This must be given for both integer and F2m polynomial additions.
- result – **[out]** Output array to store result of operation
- resultSize – **[out]** Output size of operation in bytes
- arithType – Type of arithmetic to perform (integer or F2m)

Returns
Operation status.

status_t LTC_PKHA_ModSub1(LTC_Type *base, const uint8_t *A, uint16_t sizeA, const uint8_t *B, uint16_t sizeB, const uint8_t *N, uint16_t sizeN, uint8_t *result, uint16_t *resultSize)

Performs modular subtraction - (A - B) mod N.

This function performs modular subtraction of (A - B) mod N with integer inputs.

Parameters

- base – LTC peripheral base address
- A – first addend (integer or binary polynomial)
- sizeA – Size of A in bytes
- B – second addend (integer or binary polynomial)
- sizeB – Size of B in bytes
- N – modulus
- sizeN – Size of N in bytes
- result – **[out]** Output array to store result of operation
- resultSize – **[out]** Output size of operation in bytes

Returns
Operation status.

status_t LTC_PKHA_ModSub2(LTC_Type *base, const uint8_t *A, uint16_t sizeA, const uint8_t *B, uint16_t sizeB, const uint8_t *N, uint16_t sizeN, uint8_t *result, uint16_t *resultSize)

Performs modular subtraction - (B - A) mod N.

This function performs modular subtraction of (B - A) mod N, with integer inputs.

Parameters

- base – LTC peripheral base address
- A – first addend (integer or binary polynomial)
- sizeA – Size of A in bytes
- B – second addend (integer or binary polynomial)
- sizeB – Size of B in bytes
- N – modulus
- sizeN – Size of N in bytes
- result – **[out]** Output array to store result of operation
- resultSize – **[out]** Output size of operation in bytes

Returns

Operation status.

status_t LTC_PKHA_ModMul(LTC_Type *base, const uint8_t *A, uint16_t sizeA, const uint8_t *B, uint16_t sizeB, const uint8_t *N, uint16_t sizeN, uint8_t *result, uint16_t *resultSize, *ltc_pkha_f2m_t* arithType, *ltc_pkha_montgomery_form_t* montIn, *ltc_pkha_montgomery_form_t* montOut, *ltc_pkha_timing_t* equalTime)

Performs modular multiplication - (A x B) mod N.

This function performs modular multiplication with either integer or binary polynomial (F2m) inputs. It can optionally specify whether inputs and/or outputs will be in Montgomery form or not.

Parameters

- base – LTC peripheral base address
- A – first addend (integer or binary polynomial)
- sizeA – Size of A in bytes
- B – second addend (integer or binary polynomial)
- sizeB – Size of B in bytes
- N – modulus.
- sizeN – Size of N in bytes
- result – **[out]** Output array to store result of operation
- resultSize – **[out]** Output size of operation in bytes
- arithType – Type of arithmetic to perform (integer or F2m)
- montIn – Format of inputs
- montOut – Format of output
- equalTime – Run the function time equalized or no timing equalization. This argument is ignored for F2m modular multiplication.

Returns
Operation status.

status_t LTC_PKHA_ModExp(LTC_Type *base, const uint8_t *A, uint16_t sizeA, const uint8_t *N, uint16_t sizeN, const uint8_t *E, uint16_t sizeE, uint8_t *result, uint16_t *resultSize, *ltc_pkha_f2m_t* arithType, *ltc_pkha_montgomery_form_t* montIn, *ltc_pkha_timing_t* equalTime)

Performs modular exponentiation - (A^E) mod N.

This function performs modular exponentiation with either integer or binary polynomial (F2m) inputs.

Parameters

- base – LTC peripheral base address
- A – first addend (integer or binary polynomial)
- sizeA – Size of A in bytes
- N – modulus
- sizeN – Size of N in bytes
- E – exponent
- sizeE – Size of E in bytes
- result – **[out]** Output array to store result of operation
- resultSize – **[out]** Output size of operation in bytes
- montIn – Format of A input (normal or Montgomery)
- arithType – Type of arithmetic to perform (integer or F2m)
- equalTime – Run the function time equalized or no timing equalization.

Returns
Operation status.

status_t LTC_PKHA_ModRed(LTC_Type *base, const uint8_t *A, uint16_t sizeA, const uint8_t *N, uint16_t sizeN, uint8_t *result, uint16_t *resultSize, *ltc_pkha_f2m_t* arithType)

Performs modular reduction - (A) mod N.

This function performs modular reduction with either integer or binary polynomial (F2m) inputs.

Parameters

- base – LTC peripheral base address
- A – first addend (integer or binary polynomial)
- sizeA – Size of A in bytes
- N – modulus
- sizeN – Size of N in bytes
- result – **[out]** Output array to store result of operation
- resultSize – **[out]** Output size of operation in bytes
- arithType – Type of arithmetic to perform (integer or F2m)

Returns
Operation status.

status_t LTC_PKHA_ModInv(LTC_Type *base, const uint8_t *A, uint16_t sizeA, const uint8_t *N, uint16_t sizeN, uint8_t *result, uint16_t *resultSize, *ltc_pkha_f2m_t* arithType)

Performs modular inversion - (A^-1) mod N.

This function performs modular inversion with either integer or binary polynomial (F2m) inputs.

Parameters

- base – LTC peripheral base address
- A – first addend (integer or binary polynomial)
- sizeA – Size of A in bytes
- N – modulus
- sizeN – Size of N in bytes
- result – **[out]** Output array to store result of operation
- resultSize – **[out]** Output size of operation in bytes
- arithType – Type of arithmetic to perform (integer or F2m)

Returns

Operation status.

status_t LTC_PKHA_ModR2(LTC_Type *base, const uint8_t *N, uint16_t sizeN, uint8_t *result, uint16_t *resultSize, *ltc_pkha_f2m_t* arithType)

Computes integer Montgomery factor R^2 mod N.

This function computes a constant to assist in converting operands into the Montgomery residue system representation.

Parameters

- base – LTC peripheral base address
- N – modulus
- sizeN – Size of N in bytes
- result – **[out]** Output array to store result of operation
- resultSize – **[out]** Output size of operation in bytes
- arithType – Type of arithmetic to perform (integer or F2m)

Returns

Operation status.

status_t LTC_PKHA_GCD(LTC_Type *base, const uint8_t *A, uint16_t sizeA, const uint8_t *N, uint16_t sizeN, uint8_t *result, uint16_t *resultSize, *ltc_pkha_f2m_t* arithType)

Calculates the greatest common divisor - GCD (A, N).

This function calculates the greatest common divisor of two inputs with either integer or binary polynomial (F2m) inputs.

Parameters

- base – LTC peripheral base address
- A – first value (must be smaller than or equal to N)
- sizeA – Size of A in bytes
- N – second value (must be non-zero)
- sizeN – Size of N in bytes

- result – **[out]** Output array to store result of operation
- resultSize – **[out]** Output size of operation in bytes
- arithType – Type of arithmetic to perform (integer or F2m)

Returns
Operation status.

status_t LTC_PKHA_PrimalityTest(LTC_Type *base, const uint8_t *A, uint16_t sizeA, const uint8_t *B, uint16_t sizeB, const uint8_t *N, uint16_t sizeN, bool *res)

Executes Miller-Rabin primality test.

This function calculates whether or not a candidate prime number is likely to be a prime.

Parameters

- base – LTC peripheral base address
- A – initial random seed
- sizeA – Size of A in bytes
- B – number of trial runs
- sizeB – Size of B in bytes
- N – candidate prime integer
- sizeN – Size of N in bytes
- res – **[out]** True if the value is likely prime or false otherwise

Returns
Operation status.

status_t LTC_PKHA_ECC_PointAdd(LTC_Type *base, const *ltc_pkha_ecc_point_t* *A, const *ltc_pkha_ecc_point_t* *B, const uint8_t *N, const uint8_t *R2modN, const uint8_t *aCurveParam, const uint8_t *bCurveParam, uint8_t size, *ltc_pkha_f2m_t* arithType, *ltc_pkha_ecc_point_t* *result)

Adds elliptic curve points - A + B.

This function performs ECC point addition over a prime field (Fp) or binary field (F2m) using affine coordinates.

Parameters

- base – LTC peripheral base address
- A – Left-hand point
- B – Right-hand point
- N – Prime modulus of the field
- R2modN – NULL (the function computes R2modN internally) or pointer to pre-computed R2modN (obtained from *LTC_PKHA_ModR2()* function).
- aCurveParam – A parameter from curve equation
- bCurveParam – B parameter from curve equation (constant)
- size – Size in bytes of curve points and parameters
- arithType – Type of arithmetic to perform (integer or F2m)
- result – **[out]** Result point

Returns
Operation status.

status_t LTC_PKHA_ECC_PointDouble(LTC_Type *base, const *ltc_pkha_ecc_point_t* *B, const uint8_t *N, const uint8_t *aCurveParam, const uint8_t *bCurveParam, uint8_t size, *ltc_pkha_f2m_t* arithType, *ltc_pkha_ecc_point_t* *result)

Doubles elliptic curve points - B + B.

This function performs ECC point doubling over a prime field (Fp) or binary field (F2m) using affine coordinates.

Parameters

- base – LTC peripheral base address
- B – Point to double
- N – Prime modulus of the field
- aCurveParam – A parameter from curve equation
- bCurveParam – B parameter from curve equation (constant)
- size – Size in bytes of curve points and parameters
- arithType – Type of arithmetic to perform (integer or F2m)
- result – **[out]** Result point

Returns

Operation status.

status_t LTC_PKHA_ECC_PointMul(LTC_Type *base, const *ltc_pkha_ecc_point_t* *A, const uint8_t *E, uint8_t sizeE, const uint8_t *N, const uint8_t *R2modN, const uint8_t *aCurveParam, const uint8_t *bCurveParam, uint8_t size, *ltc_pkha_timing_t* equalTime, *ltc_pkha_f2m_t* arithType, *ltc_pkha_ecc_point_t* *result, bool *infinity)

Multiplies an elliptic curve point by a scalar - E x (A0, A1).

This function performs ECC point multiplication to multiply an ECC point by a scalar integer multiplier over a prime field (Fp) or a binary field (F2m).

Parameters

- base – LTC peripheral base address
- A – Point as multiplicand
- E – Scalar multiple
- sizeE – The size of E, in bytes
- N – Modulus, a prime number for the Fp field or Irreducible polynomial for F2m field.
- R2modN – NULL (the function computes R2modN internally) or pointer to pre-computed R2modN (obtained from *LTC_PKHA_ModR2()* function).
- aCurveParam – A parameter from curve equation
- bCurveParam – B parameter from curve equation (C parameter for operation over F2m).
- size – Size in bytes of curve points and parameters
- equalTime – Run the function time equalized or no timing equalization.
- arithType – Type of arithmetic to perform (integer or F2m)
- result – **[out]** Result point

- infinity – **[out]** Output true if the result is point of infinity, and false otherwise. Writing of this output will be ignored if the argument is NULL.

Returns
Operation status.

struct _ltc_pkha_ecc_point_t

#include <fsl_ltc.h> PKHA ECC point structure

Public Members

uint8_t *X
X coordinate (affine)

uint8_t *Y
Y coordinate (affine)

2.55 LTC Blocking APIs

2.56 MCM: Miscellaneous Control Module

FSL_MCM_DRIVER_VERSION
MCM driver version.

Enum _mcm_interrupt_flag. Interrupt status flag mask. .

Values:

enumerator kMCM_CacheWriteBuffer
Cache Write Buffer Error Enable.

enumerator kMCM_ParityError
Cache Parity Error Enable.

enumerator kMCM_FPUInvalidOperation
FPU Invalid Operation Interrupt Enable.

enumerator kMCM_FPUDivideByZero
FPU Divide-by-zero Interrupt Enable.

enumerator kMCM_FPUOverflow
FPU Overflow Interrupt Enable.

enumerator kMCM_FPUUnderflow
FPU Underflow Interrupt Enable.

enumerator kMCM_FPUInexact
FPU Inexact Interrupt Enable.

enumerator kMCM_FPUInputDenormalInterrupt
FPU Input Denormal Interrupt Enable.

typedef union *_mcm_buffer_fault_attribute* mcm_buffer_fault_attribute_t
The union of buffer fault attribute.

typedef union *_mcm_lmem_fault_attribute* mcm_lmem_fault_attribute_t
The union of LMEM fault attribute.

static inline void MCM_EnableCrossbarRoundRobin(MCM_Type *base, bool enable)
Enables/Disables crossbar round robin.

Parameters

- base – MCM peripheral base address.
- enable – Used to enable/disable crossbar round robin.
  - **true** Enable crossbar round robin.
  - **false** disable crossbar round robin.

static inline void MCM_EnableInterruptStatus(MCM_Type *base, uint32_t mask)
Enables the interrupt.

Parameters

- base – MCM peripheral base address.
- mask – Interrupt status flags mask(*_mcm_interrupt_flag*).

static inline void MCM_DisableInterruptStatus(MCM_Type *base, uint32_t mask)
Disables the interrupt.

Parameters

- base – MCM peripheral base address.
- mask – Interrupt status flags mask(*_mcm_interrupt_flag*).

static inline uint16_t MCM_GetInterruptStatus(MCM_Type *base)
Gets the Interrupt status .

Parameters

- base – MCM peripheral base address.

static inline void MCM_ClearCacheWriteBufferErroStatus(MCM_Type *base)
Clears the Interrupt status .

Parameters

- base – MCM peripheral base address.

static inline uint32_t MCM_GetBufferFaultAddress(MCM_Type *base)
Gets buffer fault address.

Parameters

- base – MCM peripheral base address.

static inline void MCM_GetBufferFaultAttribute(MCM_Type *base, *mcm_buffer_fault_attribute_t* *bufferfault)

Gets buffer fault attributes.

Parameters

- base – MCM peripheral base address.
- bufferfault – Structure to store the result.

static inline uint32_t MCM_GetBufferFaultData(MCM_Type *base)

Gets buffer fault data.

Parameters

- base – MCM peripheral base address.

static inline void MCM_LimitCodeCachePeripheralWriteBuffering(MCM_Type *base, bool enable)

Limit code cache peripheral write buffering.

Parameters

- base – MCM peripheral base address.
- enable – Used to enable/disable limit code cache peripheral write buffering.
  - **true** Enable limit code cache peripheral write buffering.
  - **false** disable limit code cache peripheral write buffering.

static inline void MCM_BypassFixedCodeCacheMap(MCM_Type *base, bool enable)

Bypass fixed code cache map.

Parameters

- base – MCM peripheral base address.
- enable – Used to enable/disable bypass fixed code cache map.
  - **true** Enable bypass fixed code cache map.
  - **false** disable bypass fixed code cache map.

static inline void MCM_EnableCodeBusCache(MCM_Type *base, bool enable)

Enables/Disables code bus cache.

Parameters

- base – MCM peripheral base address.
- enable – Used to disable/enable code bus cache.
  - **true** Enable code bus cache.
  - **false** disable code bus cache.

static inline void MCM_ForceCodeCacheToNoAllocation(MCM_Type *base, bool enable)

Force code cache to no allocation.

Parameters

- base – MCM peripheral base address.
- enable – Used to force code cache to allocation or no allocation.
  - **true** Force code cache to no allocation.
  - **false** Force code cache to allocation.

static inline void MCM_EnableCodeCacheWriteBuffer(MCM_Type *base, bool enable)

Enables/Disables code cache write buffer.

Parameters

- base – MCM peripheral base address.
- enable – Used to enable/disable code cache write buffer.
  - **true** Enable code cache write buffer.
  - **false** Disable code cache write buffer.

static inline void MCM_ClearCodeBusCache(MCM_Type *base)

Clear code bus cache.

Parameters

- base – MCM peripheral base address.

static inline void MCM_EnablePcParityFaultReport(MCM_Type *base, bool enable)

Enables/Disables PC Parity Fault Report.

Parameters

- base – MCM peripheral base address.
- enable – Used to enable/disable PC Parity Fault Report.
  - **true** Enable PC Parity Fault Report.
  - **false** disable PC Parity Fault Report.

static inline void MCM_EnablePcParity(MCM_Type *base, bool enable)

Enables/Disables PC Parity.

Parameters

- base – MCM peripheral base address.
- enable – Used to enable/disable PC Parity.
  - **true** Enable PC Parity.
  - **false** disable PC Parity.

static inline void MCM_LockConfigState(MCM_Type *base)

Lock the configuration state.

Parameters

- base – MCM peripheral base address.

static inline void MCM_EnableCacheParityReporting(MCM_Type *base, bool enable)

Enables/Disables cache parity reporting.

Parameters

- base – MCM peripheral base address.
- enable – Used to enable/disable cache parity reporting.
  - **true** Enable cache parity reporting.
  - **false** disable cache parity reporting.

static inline uint32_t MCM_GetLmemFaultAddress(MCM_Type *base)

Gets LMEM fault address.

Parameters

- base – MCM peripheral base address.

static inline void MCM_GetLmemFaultAttribute(MCM_Type *base, *mcm_lmem_fault_attribute_t* *lmemFault)

Get LMEM fault attributes.

Parameters

- base – MCM peripheral base address.
- lmemFault – Structure to store the result.

static inline uint64_t MCM_GetLmemFaultData(MCM_Type *base)

Gets LMEM fault data.

Parameters

- base – MCM peripheral base address.

MCM_LMFATR_TYPE_MASK

MCM_LMFATR_MODE_MASK

MCM_LMFATR_BUFF_MASK

MCM_LMFATR_CACH_MASK

MCM_ISCR_STAT_MASK

FSL_COMPONENT_ID

union __mcm_buffer_fault_attribute

#include <fsl_mcm.h> The union of buffer fault attribute.

Public Members

uint32_t attribute

Indicates the faulting attributes, when a properly-enabled cache write buffer error interrupt event is detected.

struct *_mcm_buffer_fault_attribute*.*_mcm_buffer_fault_attribut* attribute_memory

struct __mcm_buffer_fault_attribut

#include <fsl_mcm.h>

Public Members

uint32_t busErrorDataAccessType

Indicates the type of cache write buffer access.

uint32_t busErrorPrivilegeLevel

Indicates the privilege level of the cache write buffer access.

uint32_t busErrorSize
Indicates the size of the cache write buffer access.

uint32_t busErrorAccess
Indicates the type of system bus access.

uint32_t busErrorMasterID
Indicates the crossbar switch bus master number of the captured cache write buffer bus error.

uint32_t busErrorOverrun
Indicates if another cache write buffer bus error is detected.

union __mcm_lmem_fault_attribute
#include <fsl_mcm.h> The union of LMEM fault attribute.

Public Members

uint32_t attribute
Indicates the attributes of the LMEM fault detected.

struct *_mcm_lmem_fault_attribute*.*_mcm_lmem_fault_attribut* attribute_memory

struct __mcm_lmem_fault_attribut
#include <fsl_mcm.h>

Public Members

uint32_t parityFaultProtectionSignal
Indicates the features of parity fault protection signal.

uint32_t parityFaultMasterSize
Indicates the parity fault master size.

uint32_t parityFaultWrite
Indicates the parity fault is caused by read or write.

uint32_t backdoorAccess
Indicates the LMEM access fault is initiated by core access or backdoor access.

uint32_t parityFaultSyndrome
Indicates the parity fault syndrome.

uint32_t overrun
Indicates the number of faultss.

2.57 MSCM: Miscellaneous System Control

FSL_MSCM_DRIVER_VERSION

MSCM driver version 2.0.0.

typedef struct *_mscm_uid* mscm_uid_t

static inline void MSCM_GetUID(MSCM_Type *base, *mscm_uid_t* *uid)

Get MSCM UID.

Parameters

- base – MSCM peripheral base address.
- uid – Pointer to an uid struct.

static inline void MSCM_SetSecureIrqParameter(MSCM_Type *base, const uint32_t parameter)

Set MSCM Secure Irq.

Parameters

- base – MSCM peripheral base address.
- parameter – Value to be write to SECURE_IRQ.

static inline uint32_t MSCM_GetSecureIrq(MSCM_Type *base)

Get MSCM Secure Irq.

Parameters

- base – MSCM peripheral base address.

Returns

MSCM Secure Irq.

FSL_COMPONENT_ID

struct _mscm_uid

#include <fsl_mscm.h>

2.58 MU: Messaging Unit

void MU_Init(MU_Type *base)

Initializes the MU module.

This function enables the MU clock only.

Parameters

- base – MU peripheral base address.

void MU_Deinit(MU_Type *base)

De-initializes the MU module.

This function disables the MU clock only.

Parameters

- base – MU peripheral base address.

static inline void MU_SendMsgNonBlocking(MU_Type *base, uint32_t regIndex, uint32_t msg)

Writes a message to the TX register.

This function writes a message to the specific TX register. It does not check whether the TX register is empty or not. The upper layer should make sure the TX register is empty before calling this function. This function can be used in ISR for better performance.

```
while (!(kMU_Tx0EmptyFlag & MU_GetStatusFlags(base))) { }  Wait for TX0 register empty.
MU_SendMsgNonBlocking(base, kMU_MsgReg0, MSG_VAL);  Write message to the TX0 register.
```

Parameters

- base – MU peripheral base address.
- regIndex – TX register index, see *mu_msg_reg_index_t*.
- msg – Message to send.

status_t MU_SendMsg(MU_Type *base, uint32_t regIndex, uint32_t msg)

Blocks to send a message.

This function waits until the TX register is empty and sends the message. If MU1_BUSY_POLL_COUNT is defined and non-zero, the function will timeout after the specified number of polling iterations and returns kStatus_Timeout.

Parameters

- base – MU peripheral base address.
- regIndex – MU message register, see *mu_msg_reg_index_t*.
- msg – Message to send.

Return values

- kStatus_Success – Message sent successfully.
- kStatus_Timeout – Timeout occurred while waiting for TX register to be empty.

Returns

status_t

static inline uint32_t MU_ReceiveMsgNonBlocking(MU_Type *base, uint32_t regIndex)

Reads a message from the RX register.

This function reads a message from the specific RX register. It does not check whether the RX register is full or not. The upper layer should make sure the RX register is full before calling this function. This function can be used in ISR for better performance.

```
uint32_t msg;
while (!(kMU_Rx0FullFlag & MU_GetStatusFlags(base)))
{
}  Wait for the RX0 register full.

msg = MU_ReceiveMsgNonBlocking(base, kMU_MsgReg0);  Read message from RX0 register.
```

Parameters

- base – MU peripheral base address.
- regIndex – RX register index, see *mu_msg_reg_index_t*.

Returns

The received message.

status_t MU_ReceiveMsgTimeout(MU_Type *base, uint32_t regIndex, uint32_t *readValue)

Blocks to receive a message with timeout protection.

This function waits until the RX register is full and receives the message. If MU1_BUSY_POLL_COUNT is defined and non-zero, the function will timeout after the specified number of polling iterations and return kStatus_Timeout.

This function provides the same blocking behavior as *MU_ReceiveMsg()* but with additional timeout protection to prevent system hangs if the other core becomes unresponsive or if hardware issues occur.

Note: Both *MU_ReceiveMsg()* and *MU_ReceiveMsgTimeout()* are blocking functions. The difference is that this function includes timeout protection while *MU_ReceiveMsg()* waits indefinitely.

Parameters

- base – MU peripheral base address.
- regIndex – RX register index, see *mu_msg_reg_index_t*.
- readValue – Pointer to store the received message.

Return values

- kStatus_Success – Message received successfully.
- kStatus_InvalidArgument – Invalid readValue pointer.
- kStatus_Timeout – Timeout occurred while waiting for RX register to be full.

Returns

status_t

uint32_t MU_ReceiveMsg(MU_Type *base, uint32_t regIndex)

Blocks to receive a message (infinite wait, no timeout protection).

This function waits until the RX register is full and receives the message. This function will wait indefinitely until a message is received.

Note: Both *MU_ReceiveMsg()* and *MU_ReceiveMsgTimeout()* are blocking functions. The difference is that *MU_ReceiveMsgTimeout()* includes timeout protection while this function waits indefinitely.

Warning: This function does not include timeout protection and may cause system hangs if the other core becomes unresponsive. For applications requiring timeout protection, use *MU_ReceiveMsgTimeout()* instead.

Parameters

- base – MU peripheral base address.
- regIndex – RX register index, see *mu_msg_reg_index_t*.

Returns

The received message.

static inline void MU_SetFlagsNonBlocking(MU_Type *base, uint32_t flags)

Sets the 3-bit MU flags reflect on the other MU side.

This function sets the 3-bit MU flags directly. Every time the 3-bit MU flags are changed, the status flag kMU_FlagsUpdatingFlag asserts indicating the 3-bit MU flags are updating to the other side. After the 3-bit MU flags are updated, the status flag kMU_FlagsUpdatingFlag is cleared by hardware. During the flags updating period, the flags cannot be changed. The upper layer should make sure the status flag kMU_FlagsUpdatingFlag is cleared before calling this function.

```
while (kMU_FlagsUpdatingFlag & MU_GetStatusFlags(base))
{
}  Wait for previous MU flags updating.

MU_SetFlagsNonBlocking(base, 0U);  Set the mU flags.
```

Parameters

- base – MU peripheral base address.
- flags – The 3-bit MU flags to set.

status_t MU_SetFlags(MU_Type *base, uint32_t flags)

brief Blocks setting the 3-bit MU flags reflect on the other MU side.

This function blocks setting the 3-bit MU flags. Every time the 3-bit MU flags are changed, the status flag kMU_FlagsUpdatingFlag asserts indicating the 3-bit MU flags are updating to the other side. After the 3-bit MU flags are updated, the status flag kMU_FlagsUpdatingFlag is cleared by hardware. During the flags updating period, the flags cannot be changed. This function waits for the MU status flag kMU_FlagsUpdatingFlag cleared and sets the 3-bit MU flags.

If MU1_BUSY_POLL_COUNT is defined and non-zero, the function will timeout after the specified number of polling iterations and return kStatus_Timeout.

return status_t retval kStatus_Success Flags were set successfully. retval kStatus_Timeout Timeout occurred while waiting for flags to update.

Parameters

- base – MU peripheral base address.
- flags – The 3-bit MU flags to set.

static inline uint32_t MU_GetFlags(MU_Type *base)

Gets the current value of the 3-bit MU flags set by the other side.

This function gets the current 3-bit MU flags on the current side.

Parameters

- base – MU peripheral base address.

Returns

flags Current value of the 3-bit flags.

uint32_t MU_GetStatusFlags(MU_Type *base)

Gets the MU status flags.

This function returns the bit mask of the MU status flags. See _mu_status_flags.

```
uint32_t flags;
flags = MU_GetStatusFlags(base);  Get all status flags.
if (kMU_Tx0EmptyFlag & flags)
{
    The TX0 register is empty. Message can be sent.
    MU_SendMsgNonBlocking(base, kMU_MsgReg0, MSG0_VAL);
}
if (kMU_Tx1EmptyFlag & flags)
{
    The TX1 register is empty. Message can be sent.
    MU_SendMsgNonBlocking(base, kMU_MsgReg1, MSG1_VAL);
}
```

If there are more than 4 general purpose interrupts, use *MU_GetGeneralPurposeStatusFlags*.

Parameters

- base – MU peripheral base address.

Returns

Bit mask of the MU status flags, see _mu_status_flags.

static inline uint32_t MU_GetInterruptsPending(MU_Type *base)

Gets the MU IRQ pending status of enabled interrupts.

This function returns the bit mask of the pending MU IRQs of enabled interrupts. Only these flags are checked. kMU_Tx0EmptyFlag kMU_Tx1EmptyFlag kMU_Tx2EmptyFlag kMU_Tx3EmptyFlag kMU_Rx0FullFlag kMU_Rx1FullFlag kMU_Rx2FullFlag kMU_Rx3FullFlag kMU_GenInt0Flag kMU_GenInt1Flag kMU_GenInt2Flag kMU_GenInt3Flag

Parameters

- base – MU peripheral base address.

Returns

Bit mask of the MU IRQs pending.

static inline void MU_ClearStatusFlags(MU_Type *base, uint32_t flags)

Clears the specific MU status flags.

This function clears the specific MU status flags. The flags to clear should be passed in as bit mask. See _mu_status_flags.

```
Clear general interrupt 0 and general interrupt 1 pending flags.
MU_ClearStatusFlags(base, kMU_GenInt0Flag | kMU_GenInt1Flag);
```

If there are more than 4 general purpose interrupts, use *MU_ClearGeneralPurposeStatusFlags*.

Parameters

- base – MU peripheral base address.
- flags – Bit mask of the MU status flags. See _mu_status_flags. Only the following flags can be cleared by software (if applicable for particular device), other flags are cleared by hardware:
  - kMU_GenInt0Flag
  - kMU_GenInt1Flag
  - kMU_GenInt2Flag
  - kMU_GenInt3Flag
  - kMU_MuResetInterruptFlag

- kMU_OtherSideEnterRunInterruptFlag
- kMU_OtherSideEnterHaltInterruptFlag
- kMU_OtherSideEnterWaitInterruptFlag
- kMU_OtherSideEnterStopInterruptFlag
- kMU_OtherSideEnterPowerDownInterruptFlag
- kMU_ResetAssertInterruptFlag
- kMU_HardwareResetInterruptFlag

static inline void MU_EnableInterrupts(MU_Type *base, uint32_t interrupts)

Enables the specific MU interrupts.

This function enables the specific MU interrupts. The interrupts to enable should be passed in as bit mask. See _mu_interrupt_enable.

```
Enable general interrupt 0 and TX0 empty interrupt.
MU_EnableInterrupts(base, kMU_GenInt0InterruptEnable | kMU_Tx0EmptyInterruptEnable);
```

If there are more than 4 general purpose interrupts, use *MU_EnableGeneralPurposeInterrupts*.

Parameters

- base – MU peripheral base address.
- interrupts – Bit mask of the MU interrupts. See _mu_interrupt_enable.

static inline void MU_DisableInterrupts(MU_Type *base, uint32_t interrupts)

Disables the specific MU interrupts.

This function disables the specific MU interrupts. The interrupts to disable should be passed in as bit mask. See _mu_interrupt_enable.

```
Disable general interrupt 0 and TX0 empty interrupt.
MU_DisableInterrupts(base, kMU_GenInt0InterruptEnable | kMU_Tx0EmptyInterruptEnable);
```

If there are more than 4 general purpose interrupts, use *MU_DisableGeneralPurposeInterrupts*.

Parameters

- base – MU peripheral base address.
- interrupts – Bit mask of the MU interrupts. See _mu_interrupt_enable.

status_t MU_TriggerInterrupts(MU_Type *base, uint32_t interrupts)

Triggers interrupts to the other core.

This function triggers the specific interrupts to the other core. The interrupts to trigger are passed in as bit mask. See *_mu_interrupt_trigger*. The MU should not trigger an interrupt to the other core when the previous interrupt has not been processed by the other core. This function checks whether the previous interrupts have been processed. If not, it returns an error.

```
if (kStatus_Success != MU_TriggerInterrupts(base, kMU_GenInt0InterruptTrigger | kMU_
↪GenInt2InterruptTrigger))
{
    Previous general purpose interrupt 0 or general purpose interrupt 2
    has not been processed by the other core.
}
```

If there are more than 4 general purpose interrupts, use *MU_TriggerGeneralPurposeInterrupts*.

Parameters

- base – MU peripheral base address.
- interrupts – Bit mask of the interrupts to trigger. See _mu_interrupt_trigger.

Return values

- kStatus_Success – Interrupts have been triggered successfully.
- kStatus_Fail – Previous interrupts have not been accepted.

static inline void MU_EnableGeneralPurposeInterrupts(MU_Type *base, uint32_t interrupts)

Enables the MU general purpose interrupts.

This function enables the MU general purpose interrupts. The interrupts to enable should be passed in as bit mask of *mu_general_purpose_interrupt_t*. The function *MU_EnableInterrupts* only support general interrupt 0~3, this function supports all general interrupts.

For example, to enable general purpose interrupt 0 and 3, use like this:

```
MU_EnableGeneralPurposeInterrupts(MU, kMU_GeneralPurposeInterrupt0 | kMU_
↪GeneralPurposeInterrupt3);
```

Parameters

- base – MU peripheral base address.
- interrupts – Bit mask of the MU general purpose interrupts, see *mu_general_purpose_interrupt_t*.

static inline void MU_DisableGeneralPurposeInterrupts(MU_Type *base, uint32_t interrupts)

Disables the MU general purpose interrupts.

This function disables the MU general purpose interrupts. The interrupts to disable should be passed in as bit mask of *mu_general_purpose_interrupt_t*. The function *MU_DisableInterrupts* only support general interrupt 0~3, this function supports all general interrupts.

For example, to disable general purpose interrupt 0 and 3, use like this:

```
MU_EnableGeneralPurposeInterrupts(MU, kMU_GeneralPurposeInterrupt0 | kMU_
↪GeneralPurposeInterrupt3);
```

Parameters

- base – MU peripheral base address.
- interrupts – Bit mask of the MU general purpose interrupts. see *mu_general_purpose_interrupt_t*.

static inline uint32_t MU_GetGeneralPurposeStatusFlags(MU_Type *base)

Gets the MU general purpose interrupt status flags.

This function returns the bit mask of the MU general purpose interrupt status flags. *MU_GetStatusFlags* can only get general purpose interrupt status 0~3, this function can get all general purpose interrupts status.

This example shows to check whether general purpose interrupt 0 and 3 happened.

```
uint32_t flags;
flags = MU_GetGeneralPurposeStatusFlags(base);
if (kMU_GeneralPurposeInterrupt0 & flags)
{
```

(continues on next page)

(continued from previous page)

```
}
if (kMU_GeneralPurposeInterrupt3 & flags)
{
}
```

Parameters

- base – MU peripheral base address.

Returns

Bit mask of the MU general purpose interrupt status flags.

static inline void MU_ClearGeneralPurposeStatusFlags(MU_Type *base, uint32_t flags)

Clear the MU general purpose interrupt status flags.

This function clears the specific MU general purpose interrupt status flags. The flags to clear should be passed in as bit mask. mu_general_purpose_interrupt_t_mu_status_flags.

Example to clear general purpose interrupt 0 and general interrupt 1 pending flags.

```
MU_ClearGeneralPurposeStatusFlags(base, kMU_GeneralPurposeInterrupt0 | kMU_
↪GeneralPurposeInterrupt1);
```

Parameters

- base – MU peripheral base address.
- flags – Bit mask of the MU general purpose interrupt status flags. See *mu_general_purpose_interrupt_t*.

static inline uint32_t MU_GetRxStatusFlags(MU_Type *base)

Return the RX status flags in reverse numerical order.

This function return the RX status flags in reverse order. Note: RFn bits of SR[3-0](mu status register) are mapped in ascending numerical order: RF0 -> SR[0] RF1 -> SR[1] RF2 -> SR[2] RF3 -> SR[3] This function will return these bits in reverse numerical order(RF3->RF1) to comply with *MU_GetRxStatusFlags()* of mu driver. See *MU_GetRxStatusFlags()* from drivers/mu/fsl_mu.h

```
status_reg = MU_GetRxStatusFlags(base);
```

Parameters

- base – MU peripheral base address.

Returns

MU RX status flags in reverse order

status_t MU_TriggerGeneralPurposeInterrupts(MU_Type *base, uint32_t interrupts)

Triggers general purpose interrupts to the other core.

This function triggers the specific general purpose interrupts to the other core. The interrupts to trigger are passed in as bit mask. See *mu_general_purpose_interrupt_t*. The MU should not trigger an interrupt to the other core when the previous interrupt has not been processed by the other core. This function checks whether the previous interrupts have been processed. If not, it returns an error.

```
status_t status;
status = MU_TriggerGeneralPurposeInterrupts(base, kMU_GeneralPurposeInterrupt0 | kMU_
↪GeneralPurposeInterrupt2);
```

(continues on next page)

(continued from previous page)

```
if (kStatus_Success != status)
{
    Previous general purpose interrupt 0 or general purpose interrupt 2
    has not been processed by the other core.
}
```

Parameters

- base – MU peripheral base address.
- interrupts – Bit mask of the interrupts to trigger. See *mu_general_purpose_interrupt_t*.

Return values

- kStatus_Success – Interrupts have been triggered successfully.
- kStatus_Fail – Previous interrupts have not been accepted.

void MU_BootOtherCore(MU_Type *base, mu_core_boot_mode_t mode)

Boots the other core.

This function boots the other core with a boot configuration.

Parameters

- base – MU peripheral base address.
- mode – The other core boot mode.

void MU_HoldOtherCoreReset(MU_Type *base)

Holds the other core reset.

This function causes the other core to be held in reset following any reset event.

Parameters

- base – MU peripheral base address.

static inline *status_t* MU_ResetBothSides(MU_Type *base)

Resets the MU for both A side and B side.

This function resets the MU for both A side and B side. Before reset, it is recommended to interrupt processor B, because this function may affect the ongoing processor B programs.

If MU1_BUSY_POLL_COUNT is defined and non-zero, the function will timeout after the specified number of polling iterations if waiting for the other side to come out of reset takes too long.

Note: For some platforms, only MU side A could use this function, check reference manual for details.

Parameters

- base – MU peripheral base address.

Return values

- kStatus_Success – The MU was reset successfully.
- kStatus_Timeout – Timeout occurred while waiting for the other side to come out of reset.

Returns
status_t

status_t MU_HardwareResetOtherCore(MU_Type *base, bool waitReset, bool holdReset, mu_core_boot_mode_t bootMode)

Hardware reset the other core.

This function resets the other core, the other core could mask the hardware reset by calling MU_MaskHardwareReset. The hardware reset mask feature is only available for some platforms. This function could be used together with MU_BootOtherCore to control the other core reset workflow.

If MU1_BUSY_POLL_COUNT is defined and non-zero, the function will timeout after the specified number of polling iterations and return kStatus_Timeout if waiting for the other core to enter or exit reset takes too long.

Example 1: Reset the other core, and no hold reset

```
MU_HardwareResetOtherCore(MU_A, true, false, bootMode);
```

In this example, the core at MU side B will reset with the specified boot mode.

Example 2: Reset the other core and hold it, then boot the other core later. Here the other core enters reset, and the reset is hold

```
MU_HardwareResetOtherCore(MU_A, true, true, modeDontCare);
```

Current core boot the other core when necessary.

```
MU_BootOtherCore(MU_A, bootMode);
```

Note: The feature waitReset, holdReset, and bootMode might be not supported for some platforms. waitReset is only available for platforms that FSL_FEATURE_MU_NO_CORE_STATUS not defined as 1 and FSL_FEATURE_MU_HAS_RESET_ASSERT_INT not defined as 0. holdReset is only available for platforms that FSL_FEATURE_MU_HAS_RSTH not defined as 0. bootMode is only available for platforms that FSL_FEATURE_MU_HAS_BOOT not defined as 0.

Parameters

- base – MU peripheral base address.
- waitReset – Wait the other core enters reset. Only work when there is CSSR0[RAIP].
  - true: Wait until the other core enters reset, if the other core has masked the hardware reset, then this function will be blocked.
  - false: Don't wait the reset.
- holdReset – Hold the other core reset or not. Only work when there is CCR0[RSTH].
  - true: Hold the other core in reset, this function returns directly when the other core enters reset.
  - false: Don't hold the other core in reset, this function waits until the other core out of reset.
- bootMode – Boot mode of the other core, if holdReset is true, this parameter is useless.

Return values

- kStatus_Success – The other core was reset successfully.
- kStatus_Timeout – Timeout occurred while waiting for the other core to enter or exit reset.

Returns
status_t

FSL_MU_DRIVER_VERSION
MU driver version.

enum _mu_status_flags
MU status flags.

Values:

enumerator kMU_Tx0EmptyFlag
TX0 empty.

enumerator kMU_Tx1EmptyFlag
TX1 empty.

enumerator kMU_Tx2EmptyFlag
TX2 empty.

enumerator kMU_Tx3EmptyFlag
TX3 empty.

enumerator kMU_Rx0FullFlag
RX0 full.

enumerator kMU_Rx1FullFlag
RX1 full.

enumerator kMU_Rx2FullFlag
RX2 full.

enumerator kMU_Rx3FullFlag
RX3 full.

enumerator kMU_GenInt0Flag
General purpose interrupt 0 pending.

enumerator kMU_GenInt1Flag
General purpose interrupt 1 pending.

enumerator kMU_GenInt2Flag
General purpose interrupt 2 pending.

enumerator kMU_GenInt3Flag
General purpose interrupt 3 pending.

enumerator kMU_RxFullPendingFlag
Any RX full flag is pending.

enumerator kMU_TxEmptyPendingFlag
Any TX empty flag is pending.

enumerator kMU_GenIntPendingFlag
Any general interrupt flag is pending.

enumerator kMU_EventPendingFlag
MU event pending.

enumerator kMU_FlagsUpdatingFlag
MU flags update is on-going.

enumerator kMU_MuInResetFlag
MU of any side is in reset.

enumerator kMU_MuResetInterruptFlag
The other side initializes MU reset.

enum _mu_interrupt_enable
MU interrupt source to enable.
Values:

enumerator kMU_Tx0EmptyInterruptEnable
TX0 empty.

enumerator kMU_Tx1EmptyInterruptEnable
TX1 empty.

enumerator kMU_Tx2EmptyInterruptEnable
TX2 empty.

enumerator kMU_Tx3EmptyInterruptEnable
TX3 empty.

enumerator kMU_Rx0FullInterruptEnable
RX0 full.

enumerator kMU_Rx1FullInterruptEnable
RX1 full.

enumerator kMU_Rx2FullInterruptEnable
RX2 full.

enumerator kMU_Rx3FullInterruptEnable
RX3 full.

enumerator kMU_GenInt0InterruptEnable
General purpose interrupt 0.

enumerator kMU_GenInt1InterruptEnable
General purpose interrupt 1.

enumerator kMU_GenInt2InterruptEnable
General purpose interrupt 2.

enumerator kMU_GenInt3InterruptEnable
General purpose interrupt 3.

enumerator kMU_MuResetInterruptEnable
The other side initializes MU reset.

enum _mu_interrupt_trigger
MU interrupt that could be triggered to the other core.

Values:

enumerator kMU_GenInt0InterruptTrigger
General purpose interrupt 0.

enumerator kMU_GenInt1InterruptTrigger
General purpose interrupt 1.

enumerator kMU_GenInt2InterruptTrigger
General purpose interrupt 2.

enumerator kMU_GenInt3InterruptTrigger
General purpose interrupt 3.

enum _mu_msg_reg_index
MU message register index.

Values:

enumerator kMU_MsgReg0
Message register 0.

enumerator kMU_MsgReg1
Message register 1.

enumerator kMU_MsgReg2
Message register 2.

enumerator kMU_MsgReg3
Message register 3.

enum _mu_general_purpose_interrupt

MU general purpose interrupts.

Values:

enumerator kMU_GeneralPurposeInterrupt0

General purpose interrupt 0

enumerator kMU_GeneralPurposeInterrupt1

General purpose interrupt 1

enumerator kMU_GeneralPurposeInterrupt2

General purpose interrupt 2

enumerator kMU_GeneralPurposeInterrupt3

General purpose interrupt 3

typedef enum *_mu_msg_reg_index* mu_msg_reg_index_t

MU message register index.

typedef enum *_mu_general_purpose_interrupt* mu_general_purpose_interrupt_t

MU general purpose interrupts.

MU_CORE_INTR(intr)

MU_MISC_INTR(intr)

MU_TX_INTR(intr)

MU_RX_INTR(intr)

MU_GI_INTR(intr)

MU_GET_CORE_INTR(intrs)

MU_GET_TX_INTR(intrs)

MU_GET_RX_INTR(intrs)

MU_GET_GI_INTR(intrs)

MU_CORE_FLAG(flag)

MU_STAT_FLAG(flag)

MU_TX_FLAG(flag)

MU_RX_FLAG(flag)

MU_GI_FLAG(flag)

MU_GET_CORE_FLAG(flags)

MU_GET_STAT_FLAG(flags)

MU_GET_TX_FLAG(flags)

MU_GET_RX_FLAG(flags)

MU_GET_GI_FLAG(flags)

MU1_BUSY_POLL_COUNT

Maximum polling iterations for MU waiting loops.

This parameter defines the maximum number of iterations for any polling loop in the MU code before timing out and returning an error.

It applies to all waiting loops in MU driver, such as waiting for TX register to be empty or waiting for RX register to be full.

This is a count of loop iterations, not a time-based value.

If defined as 0, polling loops will continue indefinitely until their exit condition is met, which could potentially cause the system to hang if a core becomes unresponsive.

2.59 PORT: Port Control and Interrupts

static inline void PORT_GetVersionInfo(PORT_Type *base, *port_version_info_t* *info)

Get PORT version information.

Parameters

- base – PORT peripheral base pointer
- info – PORT version information

static inline void PORT_SecletPortVoltageRange(PORT_Type *base, *port_voltage_range_t* range)

Get PORT version information.

Note: : PORTA_CONFIG[RANGE] controls the voltage ranges of Port A, B, and C. Read or write PORTB_CONFIG[RANGE] and PORTC_CONFIG[RANGE] does not take effect.

Parameters

- base – PORT peripheral base pointer
- range – port voltage range

static inline void PORT_SetPinConfig(PORT_Type *base, uint32_t pin, const *port_pin_config_t* *config)

Sets the port PCR register.

This is an example to define an input pin or output pin PCR configuration.

```
// Define a digital input pin PCR configuration
port_pin_config_t config = {
    kPORT_PullUp,
    kPORT_FastSlewRate,
    kPORT_PassiveFilterDisable,
    kPORT_OpenDrainDisable,
    kPORT_LowDriveStrength,
    kPORT_MuxAsGpio,
    kPORT_UnLockRegister,
};
```

Parameters

- base – PORT peripheral base pointer.

- pin – PORT pin number.
- config – PORT PCR register configuration structure.

static inline void PORT_SetMultiplePinsConfig(PORT_Type *base, uint32_t mask, const *port_pin_config_t* *config)

Sets the port PCR register for multiple pins.

This is an example to define input pins or output pins PCR configuration.

```
Define a digital input pin PCR configuration
port_pin_config_t config = {
    kPORT_PullUp ,
    kPORT_PullEnable,
    kPORT_FastSlewRate,
    kPORT_PassiveFilterDisable,
    kPORT_OpenDrainDisable,
    kPORT_LowDriveStrength,
    kPORT_MuxAsGpio,
    kPORT_UnlockRegister,
};
```

Parameters

- base – PORT peripheral base pointer.
- mask – PORT pin number macro.
- config – PORT PCR register configuration structure.

static inline void PORT_SetPinMux(PORT_Type *base, uint32_t pin, *port_mux_t* mux)

Configures the pin muxing.

Note: : This function is NOT recommended to use together with the PORT_SetPinsConfig, because the PORT_SetPinsConfig need to configure the pin mux anyway (Otherwise the pin mux is reset to zero : kPORT_PinDisabledOrAnalog). This function is recommended to use to reset the pin mux

---

Parameters

- base – PORT peripheral base pointer.
- pin – PORT pin number.
- mux – pin muxing slot selection.
  - *kPORT_PinDisabledOrAnalog*: Pin disabled or work in analog function.
  - *kPORT_MuxAsGpio* : Set as GPIO.
  - *kPORT_MuxAlt2* : chip-specific.
  - *kPORT_MuxAlt3* : chip-specific.
  - *kPORT_MuxAlt4* : chip-specific.
  - *kPORT_MuxAlt5* : chip-specific.
  - *kPORT_MuxAlt6* : chip-specific.
  - *kPORT_MuxAlt7* : chip-specific.

static inline void PORT_EnablePinsDigitalFilter(PORT_Type *base, uint32_t mask, bool enable)

Enables the digital filter in one port, each bit of the 32-bit register represents one pin.

Parameters

- base – PORT peripheral base pointer.
- mask – PORT pin number macro.
- enable – PORT digital filter configuration.

static inline void PORT_SetDigitalFilterConfig(PORT_Type *base, const *port_digital_filter_config_t* *config)

Sets the digital filter in one port, each bit of the 32-bit register represents one pin.

Parameters

- base – PORT peripheral base pointer.
- config – PORT digital filter configuration structure.

static inline void PORT_SetPinDriveStrength(PORT_Type *base, uint32_t pin, uint8_t strength)

Configures the port pin drive strength.

Parameters

- base – PORT peripheral base pointer.
- pin – PORT pin number.
- strength – PORT pin drive strength
  - *kPORT_LowDriveStrength* = 0U - Low-drive strength is configured.
  - *kPORT_HighDriveStrength* = 1U - High-drive strength is configured.

static inline void PORT_EnablePinDoubleDriveStrength(PORT_Type *base, uint32_t pin, bool enable)

Enables the port pin double drive strength.

Parameters

- base – PORT peripheral base pointer.
- pin – PORT pin number.
- enable – PORT pin drive strength configuration.

static inline void PORT_SetPinPullValue(PORT_Type *base, uint32_t pin, uint8_t value)

Configures the port pin pull value.

Parameters

- base – PORT peripheral base pointer.
- pin – PORT pin number.
- value – PORT pin pull value
  - *kPORT_LowPullResistor* = 0U - Low internal pull resistor value is selected.
  - *kPORT_HighPullResistor* = 1U - High internal pull resistor value is selected.

static inline uint32_t PORT_GetEFTDetectFlags(PORT_Type *base)

Get EFT detect flags.

Parameters

- base – PORT peripheral base pointer

Returns
EFT detect flags

static inline void PORT_EnableEFTDetectInterrupts(PORT_Type *base, uint32_t interrupt)
Enable EFT detect interrupts.

Parameters

- base – PORT peripheral base pointer
- interrupt – EFT detect interrupt

static inline void PORT_DisableEFTDetectInterrupts(PORT_Type *base, uint32_t interrupt)
Disable EFT detect interrupts.

Parameters

- base – PORT peripheral base pointer
- interrupt – EFT detect interrupt

static inline void PORT_ClearAllLowEFTDetectors(PORT_Type *base)
Clear all low EFT detector.

Note: : Port B and Port C pins share the same EFT detector clear control from PORTC_EDCR register. Any write to the PORTB_EDCR does not take effect.

Parameters

- base – PORT peripheral base pointer

static inline void PORT_ClearAllHighEFTDetectors(PORT_Type *base)
Clear all high EFT detector.

Parameters

- base – PORT peripheral base pointer

FSL_PORT_DRIVER_VERSION
PORT driver version.

enum _port_pull
Internal resistor pull feature selection.

Values:

enumerator kPORT_PullDisable
Internal pull-up/down resistor is disabled.

enumerator kPORT_PullDown
Internal pull-down resistor is enabled.

enumerator kPORT_PullUp
Internal pull-up resistor is enabled.

enum _port_pull_value
Internal resistor pull value selection.

Values:

enumerator kPORT_LowPullResistor
Low internal pull resistor value is selected.

enumerator kPORT_HighPullResistor
High internal pull resistor value is selected.

enum _port_slew_rate
Slew rate selection.
Values:

enumerator kPORT_FastSlewRate
Fast slew rate is configured.

enumerator kPORT_SlowSlewRate
Slow slew rate is configured.

enum _port_open_drain_enable
Open Drain feature enable/disable.
Values:

enumerator kPORT_OpenDrainDisable
Open drain output is disabled.

enumerator kPORT_OpenDrainEnable
Open drain output is enabled.

enum _port_passive_filter_enable
Passive filter feature enable/disable.
Values:

enumerator kPORT_PassiveFilterDisable
Passive input filter is disabled.

enumerator kPORT_PassiveFilterEnable
Passive input filter is enabled.

enum _port_drive_strength
Configures the drive strength.
Values:

enumerator kPORT_LowDriveStrength
Low-drive strength is configured.

enumerator kPORT_HighDriveStrength
High-drive strength is configured.

enum _port_drive_strength1

Configures the drive strength1.

Values:

enumerator kPORT_NormalDriveStrength

Normal drive strength

enumerator kPORT_DoubleDriveStrength

Double drive strength

enum _port_input_buffer

input buffer disable/enable.

Values:

enumerator kPORT_InputBufferDisable

Digital input is disabled

enumerator kPORT_InputBufferEnable

Digital input is enabled

enum _port_invet_input

Digital input is not inverted or it is inverted.

Values:

enumerator kPORT_InputNormal

Digital input is not inverted

enumerator kPORT_InputInvert

Digital input is inverted

enum _port_lock_register

Unlock/lock the pin control register field[15:0].

Values:

enumerator kPORT_UnlockRegister

Pin Control Register fields [15:0] are not locked.

enumerator kPORT_LockRegister

Pin Control Register fields [15:0] are locked.

enum _port_mux

Pin mux selection.

Values:

enumerator kPORT_PinDisabledOrAnalog

Corresponding pin is disabled, but is used as an analog pin.

enumerator kPORT_MuxAsGpio
Corresponding pin is configured as GPIO.

enumerator kPORT_MuxAlt0
Chip-specific

enumerator kPORT_MuxAlt1
Chip-specific

enumerator kPORT_MuxAlt2
Chip-specific

enumerator kPORT_MuxAlt3
Chip-specific

enumerator kPORT_MuxAlt4
Chip-specific

enumerator kPORT_MuxAlt5
Chip-specific

enumerator kPORT_MuxAlt6
Chip-specific

enumerator kPORT_MuxAlt7
Chip-specific

enumerator kPORT_MuxAlt8
Chip-specific

enumerator kPORT_MuxAlt9
Chip-specific

enumerator kPORT_MuxAlt10
Chip-specific

enumerator kPORT_MuxAlt11
Chip-specific

enumerator kPORT_MuxAlt12
Chip-specific

enumerator kPORT_MuxAlt13
Chip-specific

enumerator kPORT_MuxAlt14
Chip-specific

enumerator kPORT_MuxAlt15
Chip-specific

enum _port_digital_filter_clock_source
Digital filter clock source selection.
Values:

enumerator kPORT_BusClock
Digital filters are clocked by the bus clock.

enumerator kPORT_LpoClock
Digital filters are clocked by the 1 kHz LPO clock.

enum _port_voltage_range
PORT voltage range.
Values:

enumerator kPORT_VoltageRange1Dot71V_3Dot6V
Port voltage range is 1.71 V - 3.6 V.

enumerator kPORT_VoltageRange2Dot70V_3Dot6V
Port voltage range is 2.70 V - 3.6 V.

typedef enum *_port_mux* port_mux_t
Pin mux selection.

typedef enum *_port_digital_filter_clock_source* port_digital_filter_clock_source_t
Digital filter clock source selection.

typedef struct *_port_digital_filter_config* port_digital_filter_config_t
PORT digital filter feature configuration definition.

typedef struct *_port_pin_config* port_pin_config_t
PORT pin configuration structure.

typedef struct *_port_version_info* port_version_info_t
PORT version information.

typedef enum *_port_voltage_range* port_voltage_range_t
PORT voltage range.

FSL_COMPONENT_ID

struct _port_digital_filter_config
#include <fsl_port.h> PORT digital filter feature configuration definition.

Public Members

uint32_t digitalFilterWidth
Set digital filter width

port_digital_filter_clock_source_t clockSource
Set digital filter clockSource

struct __port__pin__config
#include <fsl_port.h> PORT pin configuration structure.

Public Members

uint16_t pullSelect
No-pull/pull-down/pull-up select

uint16_t pullValueSelect
Pull value select

uint16_t slewRate
Fast/slow slew rate Configure

uint16_t passiveFilterEnable
Passive filter enable/disable

uint16_t openDrainEnable
Open drain enable/disable

uint16_t driveStrength
Fast/slow drive strength configure

uint16_t driveStrength1
Normal/Double drive strength enable/disable

uint16_t inputBuffer
Input Buffer Configure

uint16_t invertInput
Invert Input Configure

uint16_t lockRegister
Lock/unlock the PCR field[15:0]

struct __port__version__info
#include <fsl_port.h> PORT version information.

Public Members

uint16_t feature
Feature Specification Number.

uint8_t minor
Minor Version Number.

uint8_t major
Major Version Number.

2.60 RTC: Real Time Clock

void RTC_Init(RTC_Type *base, const *rtc_config_t* *config)
Ungates the RTC clock and configures the peripheral for basic operation.

This function issues a software reset if the timer invalid flag is set.

Note: This API should be called at the beginning of the application using the RTC driver.

Parameters

- base – RTC peripheral base address
- config – Pointer to the user's RTC configuration structure.

static inline void RTC_Deinit(RTC_Type *base)
Stops the timer and gate the RTC clock.

Parameters

- base – RTC peripheral base address

void RTC_GetDefaultConfig(*rtc_config_t* *config)
Fills in the RTC config struct with the default settings.

The default values are as follows.

```
config->clockOutput = false;
config->wakeupSelect = false;
config->updateMode = false;
config->supervisorAccess = false;
config->compensationInterval = 0;
config->compensationTime = 0;
```

Parameters

- config – Pointer to the user's RTC configuration structure.

status_t RTC_SetDatetime(RTC_Type *base, const *rtc_datetime_t* *datetime)
Sets the RTC date and time according to the given time structure.

The RTC counter must be stopped prior to calling this function because writes to the RTC seconds register fail if the RTC counter is running.

Parameters

- base – RTC peripheral base address
- datetime – Pointer to the structure where the date and time details are stored.

Returns

kStatus_Success: Success in setting the time and starting the RTC kStatus_InvalidArgument: Error because the datetime format is incorrect

void RTC_GetDatetime(RTC_Type *base, *rtc_datetime_t* *datetime)

Gets the RTC time and stores it in the given time structure.

Parameters

- base – RTC peripheral base address
- datetime – Pointer to the structure where the date and time details are stored.

status_t RTC_SetAlarm(RTC_Type *base, const *rtc_datetime_t* *alarmTime)

Sets the RTC alarm time.

The function checks whether the specified alarm time is greater than the present time. If not, the function does not set the alarm and returns an error.

Parameters

- base – RTC peripheral base address
- alarmTime – Pointer to the structure where the alarm time is stored.

Returns

kStatus_Success: success in setting the RTC alarm kStatus_InvalidArgument: Error because the alarm datetime format is incorrect kStatus_Fail: Error because the alarm time has already passed

void RTC_GetAlarm(RTC_Type *base, *rtc_datetime_t* *datetime)

Returns the RTC alarm time.

Parameters

- base – RTC peripheral base address
- datetime – Pointer to the structure where the alarm date and time details are stored.

void RTC_EnableInterrupts(RTC_Type *base, uint32_t mask)

Enables the selected RTC interrupts.

Parameters

- base – RTC peripheral base address
- mask – The interrupts to enable. This is a logical OR of members of the enumeration *rtc_interrupt_enable_t*

void RTC_DisableInterrupts(RTC_Type *base, uint32_t mask)

Disables the selected RTC interrupts.

Parameters

- base – RTC peripheral base address
- mask – The interrupts to enable. This is a logical OR of members of the enumeration *rtc_interrupt_enable_t*

uint32_t RTC_GetEnabledInterrupts(RTC_Type *base)

Gets the enabled RTC interrupts.

Parameters

- base – RTC peripheral base address

Returns

The enabled interrupts. This is the logical OR of members of the enumeration *rtc_interrupt_enable_t*

uint32_t RTC_GetStatusFlags(RTC_Type *base)

Gets the RTC status flags.

Parameters

- base – RTC peripheral base address

Returns

The status flags. This is the logical OR of members of the enumeration *rtc_status_flags_t*

void RTC_ClearStatusFlags(RTC_Type *base, uint32_t mask)

Clears the RTC status flags.

Parameters

- base – RTC peripheral base address
- mask – The status flags to clear. This is a logical OR of members of the enumeration *rtc_status_flags_t*

static inline void RTC_EnableLPOClock(RTC_Type *base, bool enable)

Enable/Disable RTC 1kHz LPO clock.

Note: After setting this bit, RTC prescaler increments using the LPO 1kHz clock and not the RTC 32kHz crystal clock.

Parameters

- base – RTC peripheral base address
- enable – Enable/Disable RTC 1kHz LPO clock

static inline void RTC_StartTimer(RTC_Type *base)

Starts the RTC time counter.

After calling this function, the timer counter increments once a second provided SR[TOF] or SR[TIF] are not set.

Parameters

- base – RTC peripheral base address

static inline void RTC_StopTimer(RTC_Type *base)

Stops the RTC time counter.

RTC's seconds register can be written to only when the timer is stopped.

Parameters

- base – RTC peripheral base address

void RTC_GetMonotonicCounter(RTC_Type *base, uint64_t *counter)

Reads the values of the Monotonic Counter High and Monotonic Counter Low and returns them as a single value.

Parameters

- base – RTC peripheral base address
- counter – Pointer to variable where the value is stored.

void RTC_SetMonotonicCounter(RTC_Type *base, uint64_t counter)

Writes values Monotonic Counter High and Monotonic Counter Low by decomposing the given single value. The Monotonic Overflow Flag in RTC_SR is cleared due to the API.

Parameters

- base – RTC peripheral base address
- counter – Counter value

status_t RTC_IncrementMonotonicCounter(RTC_Type *base)

Increments the Monotonic Counter by one.

Increments the Monotonic Counter (registers RTC_MCLR and RTC_MCHR accordingly) by setting the monotonic counter enable (MER[MCE]) and then writing to the RTC_MCLR register. A write to the monotonic counter low that causes it to overflow also increments the monotonic counter high.

Parameters

- base – RTC peripheral base address

Returns

kStatus_Success: success kStatus_Fail: error occurred, either time invalid or monotonic overflow flag was found

FSL_RTC_DRIVER_VERSION

Version 2.4.0

enum _rtc_interrupt_enable

List of RTC interrupts.

Values:

enumerator kRTC_TimeInvalidInterruptEnable

Time invalid interrupt.

enumerator kRTC_TimeOverflowInterruptEnable

Time overflow interrupt.

enumerator kRTC_AlarmInterruptEnable

Alarm interrupt.

enumerator kRTC_MonotonicOverflowInterruptEnable

Monotonic Overflow Interrupt Enable

enumerator kRTC_SecondsInterruptEnable

Seconds interrupt.

enumerator kRTC_TestModeInterruptEnable

enumerator kRTC_FlashSecurityInterruptEnable

enumerator kRTC_TamperPinInterruptEnable

enumerator kRTC_SecurityModuleInterruptEnable

enumerator kRTC_LossOfClockInterruptEnable

enum _rtc_status_flags

List of RTC flags.

Values:

enumerator kRTC_TimeInvalidFlag

Time invalid flag

enumerator kRTC_TimeOverflowFlag

Time overflow flag

enumerator kRTC_AlarmFlag

Alarm flag

enumerator kRTC_MonotonicOverflowFlag

Monotonic Overflow Flag

enumerator kRTC_TamperInterruptDetectFlag

Tamper interrupt detect flag

enumerator kRTC_TestModeFlag

enumerator kRTC_FlashSecurityFlag

enumerator kRTC_TamperPinFlag

enumerator kRTC_SecurityTamperFlag

enumerator kRTC_LossOfClockTamperFlag

enum _rtc_osc_cap_load

List of RTC Oscillator capacitor load settings.

Values:

enumerator kRTC_Capacitor_2p

2 pF capacitor load

enumerator kRTC_Capacitor_4p
4 pF capacitor load

enumerator kRTC_Capacitor_8p
8 pF capacitor load

enumerator kRTC_Capacitor_16p
16 pF capacitor load

enum _rtc_timer_seconds_interrupt_frequency
List of RTC Timer Seconds Interrupt Frequencies.
Values:

enumerator kRTC_TimerSecondsFrequency1Hz
Timer seconds frequency is 1Hz

enumerator kRTC_TimerSecondsFrequency2Hz
Timer seconds frequency is 2Hz

enumerator kRTC_TimerSecondsFrequency4Hz
Timer seconds frequency is 4Hz

enumerator kRTC_TimerSecondsFrequency8Hz
Timer seconds frequency is 8Hz

enumerator kRTC_TimerSecondsFrequency16Hz
Timer seconds frequency is 16Hz

enumerator kRTC_TimerSecondsFrequency32Hz
Timer seconds frequency is 32Hz

enumerator kRTC_TimerSecondsFrequency64Hz
Timer seconds frequency is 64Hz

enumerator kRTC_TimerSecondsFrequency128Hz
Timer seconds frequency is 128Hz

typedef enum *_rtc_interrupt_enable* rtc_interrupt_enable_t
List of RTC interrupts.

typedef enum *_rtc_status_flags* rtc_status_flags_t
List of RTC flags.

typedef enum *_rtc_osc_cap_load* rtc_osc_cap_load_t
List of RTC Oscillator capacitor load settings.

typedef enum *_rtc_timer_seconds_interrupt_frequency* rtc_timer_seconds_interrupt_frequency_t
List of RTC Timer Seconds Interrupt Frequencies.

typedef struct *_rtc_datetime* rtc_datetime_t

Structure is used to hold the date and time.

typedef struct *_rtc_pin_config* rtc_pin_config_t

RTC pin config structure.

typedef struct *_rtc_config* rtc_config_t

RTC config structure.

This structure holds the configuration settings for the RTC peripheral. To initialize this structure to reasonable defaults, call the *RTC_GetDefaultConfig()* function and pass a pointer to your config structure instance.

The config struct can be made const so it resides in flash

static inline uint32_t RTC_GetTamperTimeSeconds(RTC_Type *base)

Get the RTC tamper time seconds.

Parameters

- base – RTC peripheral base address

static inline void RTC_SetOscCapLoad(RTC_Type *base, uint32_t capLoad)

This function sets the specified capacitor configuration for the RTC oscillator.

Parameters

- base – RTC peripheral base address
- capLoad – Oscillator loads to enable. This is a logical OR of members of the enumeration *rtc_osc_cap_load_t*

static inline void RTC_Reset(RTC_Type *base)

Performs a software reset on the RTC module.

This resets all RTC registers except for the SWR bit and the RTC_WAR and RTC_RAR registers. The SWR bit is cleared by software explicitly clearing it.

Parameters

- base – RTC peripheral base address

static inline void RTC_EnableWakeUpPin(RTC_Type *base, bool enable)

Enables or disables the RTC Wakeup Pin Operation.

This function enable or disable RTC Wakeup Pin. The wakeup pin is optional and not available on all devices.

Parameters

- base – RTC_Type base pointer.
- enable – true to enable, false to disable.

static inline void RTC_EnableClockOutput(RTC_Type *base, bool enable)

Enables or disables the RTC 32 kHz clock output.

This function enables or disables the RTC 32 kHz clock output.

Parameters

- base – RTC_Type base pointer.
- enable – true to enable, false to disable.

void RTC_SetTimerSecondsInterruptFrequency(RTC_Type *base, *rtc_timer_seconds_interrupt_frequency_t* freq)

Sets the RTC timer seconds interrupt frequency.

This function sets the RTC timer seconds interrupt frequency.

Parameters

- base – RTC peripheral base address
- freq – The timer seconds interrupt frequency. This is a member of the enumeration *rtc_timer_seconds_interrupt_frequency_t*

struct __rtc_datetime

#include <fsl_rtc.h> Structure is used to hold the date and time.

Public Members

uint16_t year

Range from 1970 to 2099.

uint8_t month

Range from 1 to 12.

uint8_t day

Range from 1 to 31 (depending on month).

uint8_t hour

Range from 0 to 23.

uint8_t minute

Range from 0 to 59.

uint8_t second

Range from 0 to 59.

struct __rtc_pin_config

#include <fsl_rtc.h> RTC pin config structure.

Public Members

bool inputLogic

true: Tamper pin input data is logic one. false: Tamper pin input data is logic zero.

bool pinActiveLow

true: Tamper pin is active low. false: Tamper pin is active high.

bool filterEnable

true: Input filter is enabled on the tamper pin. false: Input filter is disabled on the tamper pin.

bool pullSelectNegate

true: Tamper pin pull resistor direction will negate the tamper pin. false: Tamper pin pull resistor direction will assert the tamper pin.

bool pullEnable

true: Pull resistor is enabled on tamper pin. false: Pull resistor is disabled on tamper pin.

struct __rtc_config

#include <fsl_rtc.h> RTC config structure.

This structure holds the configuration settings for the RTC peripheral. To initialize this structure to reasonable defaults, call the *RTC_GetDefaultConfig()* function and pass a pointer to your config structure instance.

The config struct can be made const so it resides in flash

Public Members

bool clockOutput

true: The 32 kHz clock is not output to other peripherals; false: The 32 kHz clock is output to other peripherals

bool wakeupSelect

true: Wakeup pin outputs the 32 KHz clock; false:Wakeup pin used to wakeup the chip

bool updateMode

true: Registers can be written even when locked under certain conditions, false: No writes allowed when registers are locked

bool supervisorAccess

true: Non-supervisor accesses are allowed; false: Non-supervisor accesses are not supported

uint32_t compensationInterval

Compensation interval that is written to the CIR field in RTC TCR Register

uint32_t compensationTime

Compensation time that is written to the TCR field in RTC TCR Register

2.61 SEMA42: Hardware Semaphores Driver

FSL_SEMA42_DRIVER_VERSION

SEMA42 driver version.

SEMA42 status return codes.

Values:

enumerator kStatus_SEMA42_Busy
SEMA42 gate has been locked by other processor.

enumerator kStatus_SEMA42_Reseting
SEMA42 gate reseting is ongoing.

enum _sema42_gate_status
SEMA42 gate lock status.

Values:

enumerator kSEMA42_Unlocked
The gate is unlocked.

enumerator kSEMA42_LockedByProc0
The gate is locked by processor 0.

enumerator kSEMA42_LockedByProc1
The gate is locked by processor 1.

enumerator kSEMA42_LockedByProc2
The gate is locked by processor 2.

enumerator kSEMA42_LockedByProc3
The gate is locked by processor 3.

enumerator kSEMA42_LockedByProc4
The gate is locked by processor 4.

enumerator kSEMA42_LockedByProc5
The gate is locked by processor 5.

enumerator kSEMA42_LockedByProc6
The gate is locked by processor 6.

enumerator kSEMA42_LockedByProc7
The gate is locked by processor 7.

enumerator kSEMA42_LockedByProc8
The gate is locked by processor 8.

enumerator kSEMA42_LockedByProc9
The gate is locked by processor 9.

enumerator kSEMA42_LockedByProc10
The gate is locked by processor 10.

enumerator kSEMA42_LockedByProc11
The gate is locked by processor 11.

enumerator kSEMA42_LockedByProc12

The gate is locked by processor 12.

enumerator kSEMA42_LockedByProc13

The gate is locked by processor 13.

enumerator kSEMA42_LockedByProc14

The gate is locked by processor 14.

typedef enum *_sema42_gate_status* sema42_gate_status_t

SEMA42 gate lock status.

void SEMA42_Init(SEMA42_Type *base)

Initializes the SEMA42 module.

This function initializes the SEMA42 module. It only enables the clock but does not reset the gates because the module might be used by other processors at the same time. To reset the gates, call either SEMA42_ResetGate or SEMA42_ResetAllGates function.

Parameters

- base – SEMA42 peripheral base address.

void SEMA42_Deinit(SEMA42_Type *base)

De-initializes the SEMA42 module.

This function de-initializes the SEMA42 module. It only disables the clock.

Parameters

- base – SEMA42 peripheral base address.

status_t SEMA42_TryLock(SEMA42_Type *base, uint8_t gateNum, uint8_t procNum)

Tries to lock the SEMA42 gate.

This function tries to lock the specific SEMA42 gate. If the gate has been locked by another processor, this function returns an error code.

Parameters

- base – SEMA42 peripheral base address.
- gateNum – Gate number to lock.
- procNum – Current processor number.

Return values

- kStatus_Success – Lock the sema42 gate successfully.
- kStatus_SEMA42_Busy – Sema42 gate has been locked by another processor.

status_t SEMA42_Lock(SEMA42_Type *base, uint8_t gateNum, uint8_t procNum)

Locks the SEMA42 gate.

This function locks the specific SEMA42 gate. If the gate has been locked by other processors, this function waits until it is unlocked and then lock it.

If SEMA42_BUSY_POLL_COUNT is defined and non-zero, the function will timeout after the specified number of polling iterations and return kStatus_Timeout.

Parameters

- base – SEMA42 peripheral base address.

- gateNum – Gate number to lock.
- procNum – Current processor number.

Return values

- kStatus_Success – The gate was successfully locked.
- kStatus_Timeout – Timeout occurred while waiting for the gate to be unlocked.

Returns
status_t

static inline void SEMA42_Unlock(SEMA42_Type *base, uint8_t gateNum)

Unlocks the SEMA42 gate.

This function unlocks the specific SEMA42 gate. It only writes unlock value to the SEMA42 gate register. However, it does not check whether the SEMA42 gate is locked by the current processor or not. As a result, if the SEMA42 gate is not locked by the current processor, this function has no effect.

Parameters

- base – SEMA42 peripheral base address.
- gateNum – Gate number to unlock.

static inline *sema42_gate_status_t* SEMA42_GetGateStatus(SEMA42_Type *base, uint8_t gateNum)

Gets the status of the SEMA42 gate.

This function checks the lock status of a specific SEMA42 gate.

Parameters

- base – SEMA42 peripheral base address.
- gateNum – Gate number.

Returns
status Current status.

status_t SEMA42_ResetGate(SEMA42_Type *base, uint8_t gateNum)

Resets the SEMA42 gate to an unlocked status.

This function resets a SEMA42 gate to an unlocked status.

Parameters

- base – SEMA42 peripheral base address.
- gateNum – Gate number.

Return values

- kStatus_Success – SEMA42 gate is reset successfully.
- kStatus_SEMA42_Reseting – Some other reset process is ongoing.

static inline *status_t* SEMA42_ResetAllGates(SEMA42_Type *base)

Resets all SEMA42 gates to an unlocked status.

This function resets all SEMA42 gate to an unlocked status.

Parameters

- base – SEMA42 peripheral base address.

Return values

- kStatus_Success – SEMA42 is reset successfully.

- kStatus_SEMA42_Reseting – Some other reset process is ongoing.

SEMA42_GATE_NUM_RESET_ALL

The number to reset all SEMA42 gates.

SEMA42_GATEn(base, n)

SEMA42 gate n register address.

The SEMA42 gates are sorted in the order 3, 2, 1, 0, 7, 6, 5, 4, ... not in the order 0, 1, 2, 3, 4, 5, 6, 7, ... The macro SEMA42_GATEn gets the SEMA42 gate based on the gate index.

The input gate index is XOR'ed with 3U: 0 ^ 3 = 3 1 ^ 3 = 2 2 ^ 3 = 1 3 ^ 3 = 0 4 ^ 3 = 7 5 ^ 3 = 6 6 ^ 3 = 5 7 ^ 3 = 4 ...

SEMA42_BUSY_POLL_COUNT

Maximum polling iterations for SEMA42 waiting loops.

This parameter defines the maximum number of iterations for any polling loop in the SEMA42 driver code before timing out and returning an error.

It applies to all waiting loops in SEMA42 driver, such as waiting for a gate to be unlocked, waiting for a reset to complete, or waiting for a resource to become available.

This is a count of loop iterations, not a time-based value.

If defined as 0, polling loops will continue indefinitely until their exit condition is met, which could potentially cause the system to hang if hardware doesn't respond or if a resource is never released.

2.62 SFA: Signal Frequency Analyser

void SFA_GetDefaultConfig(*sfa_config_t* *config)

Fill the SFA configuration structure with default settings.

The default values are:

```
config->mode = kSFA_FrequencyMeasurement0;
config->cutSelect = kSFA_CUTSelect0;
config->refSelect = kSFA_REFSelect0;
config->prediv = 0U;
config->trigStart = kSFA_TriggerStartSelect0;
config->startPolarity = kSFA_TriggerStartPolarityRiseEdge;
config->trigEnd = kSFA_TriggerEndSelect0;
config->endPolarity = kSFA_TriggerEndPolarityRiseEdge;
config->enableTrigMeasurement = false;
config->enableCUTPin = false;
config->cutTarget = 0xffffU;
config->refTarget = 0xffffffffU;
```

Parameters

- config – Pointer to the user configuration structure.

void SFA_Init(SFA_Type *base)

Initialize SFA.

Parameters

- base – SFA peripheral base address.

status_t SFA_Deinit(SFA_Type *base)

Clear counter, disable SFA and gate the SFA clock.

Parameters

- base – SFA peripheral base address.

Return values

- kStatus_Success – run success.
- kStatus_Timeout – timeout occurs.

static inline void SFA_EnableCUTPin(SFA_Type *base, bool enable)

Control the connection of the clock under test to an external pin.

Parameters

- base – SFA peripheral base address.
- enable – true: connect the clock under test and external pin. false: Disconnect the clock under test and external pin.

static inline uint32_t SFA_GetStatusFlags(SFA_Type *base)

Get SFA status flags.

Parameters

- base – SFA peripheral base address.

void SFA_ClearStatusFlag(SFA_Type *base, uint32_t mask)

Clear the SFA status flags.

Note: To clear kSFA_RefStoppedFlag, kSFA_CUTStoppedFlag, kSFA_MeasurementStartedFlag, and kSFA_ReferenceCounterTimeOutFlag, each counter will also be cleared.

Parameters

- base – SFA peripheral base address.
- mask – SFA status flag mask (see *_sfa_status_flags* for bit definition).

static inline void SFA_EnableInterrupts(SFA_Type *base, uint32_t mask)

Enable the selected SFA interrupt.

Parameters

- base – SFA peripheral base address.
- mask – The interrupt to enable (see *_sfa_interrupts_enable* for definition).

static inline void SFA_DisableInterrupts(SFA_Type *base, uint32_t mask)

Disable the selected SFA interrupt.

Parameters

- base – SFA peripheral base address.
- mask – The interrupt to disable (see *_sfa_interrupts_enable* for definition).

static inline uint8_t SFA_GetMode(SFA_Type *base)

Get SFA measurement mode.

Parameters

- base – SFA peripheral base address.

static inline uint8_t SFA_GetCUTPredivide(SFA_Type *base)

Get CUT predivide value.

Parameters

- base – SFA peripheral base address.

void SFA_InstallCallback(SFA_Type *base, *sfa_callback_t* function)

Install the callback function to be called when IRQ happens or measurement completes.

Parameters

- base – SFA peripheral base address.
- function – the SFA measure completed callback function.

void SFA_SetMeasureConfig(SFA_Type *base, const *sfa_config_t* *config)

Set Measurement options with the passed in configuration structure.

Parameters

- base – SFA peripheral base address.
- config – SFA configuration structure.

status_t SFA_MeasureBlocking(SFA_Type *base)

Start SFA measurement in blocking mode.

Parameters

- base – SFA peripheral base address.

Return values

- kStatus_SFA_MeasurementCompleted – SFA measure completes.
- kStatus_SFA_ReferenceCounterTimeout – reference counter timeout error happens.
- kStatus_SFA_CUTCounterTimeout – CUT counter time out happens.

void SFA_MeasureNonBlocking(SFA_Type *base)

Start measure sequence in NonBlocking mode.

This function performs nonblocking measurement by enabling sfa interrupt (Please enable the FreqGreaterThanMax and FreqLessThanMin interrupts individually as needed). The callback function must be installed before invoking this function.

Note: This function has different functions for different instances.

Parameters

- base – SFA peripheral base address.

void SFA_AbortMeasureSequence(SFA_Type *base)

Abort SFA measurement sequence.

Parameters

- base – SFA peripheral base address.

uint32_t SFA_CalculateFrequencyOrPeriod(SFA_Type *base, uint32_t refFrequency)

Calculate the frequency or period.

Parameters

- base – SFA peripheral base address.

- refFrequency – The reference clock frequency(BUS clock recommended).

static inline uint32_t SFA_GetCUTCounter(SFA_Type *base)

Get current count of the clock under test.

Parameters

- base – SFA peripheral base address.

static inline void SFA_SetCUTTargetCount(SFA_Type *base, uint32_t count)

Set the target count for the clock under test.

Parameters

- base – SFA peripheral base address.
- count – target count for CUT.

static inline uint32_t SFA_GetCUTTargetCount(SFA_Type *base)

Get the target count of the clock under test.

Parameters

- base – SFA peripheral base address.

static inline void SFA_SetCUTLowLimitClockCount(SFA_Type *base, uint32_t count)

Set CUT low limit clock count.

Parameters

- base – SFA peripheral base address.
- count – low limit count for CUT clock.

static inline uint32_t SFA_GetCUTLowLimitClockCount(SFA_Type *base)

Get CUT low limit clock count.

Parameters

- base – SFA peripheral base address.

static inline void SFA_SetCUTHighLimitClockCount(SFA_Type *base, uint32_t count)

Set CUT high limit clock count.

Parameters

- base – SFA peripheral base address.
- count – high limit count for CUT clock.

static inline uint32_t SFA_GetCUTHighLimitClockCount(SFA_Type *base)

Get CUT high limit clock count.

Parameters

- base – SFA peripheral base address.

static inline uint32_t SFA_GetREFCounter(SFA_Type *base)

Get current count of the reference clock.

Parameters

- base – SFA peripheral base address.

static inline void SFA_SetREFTargetCount(SFA_Type *base, uint32_t count)

Set the target count for the reference clock.

Parameters

- base – SFA peripheral base address.

- count – target count for reference clock.

static inline uint32_t SFA_GetREFTargetCount(SFA_Type *base)

Get the target count of the reference clock.

Parameters

- base – SFA peripheral base address.

static inline uint32_t SFA_GetREFStartCount(SFA_Type *base)

Get saved reference clock counter which is loaded when measurement start.

Parameters

- base – SFA peripheral base address.

static inline uint32_t SFA_GetREFEndCount(SFA_Type *base)

Get saved reference clock counter which is loaded when measurement complete.

Parameters

- base – SFA peripheral base address.

static inline void SFA_SetREFLowLimitClockCount(SFA_Type *base, uint32_t count)

Set REF low limit clock count.

Parameters

- base – SFA peripheral base address.
- count – low limit count for REF clock.

static inline uint32_t SFA_GetREFLowLimitClockCount(SFA_Type *base)

Get REF low limit clock count.

Parameters

- base – SFA peripheral base address.

static inline void SFA_SetREFHighLimitClockCount(SFA_Type *base, uint32_t count)

Set REF high limit clock count.

Parameters

- base – SFA peripheral base address.
- count – high limit count for REF clock.

static inline uint32_t SFA_GetREFHighLimitClockCount(SFA_Type *base)

Get REF high limit clock count.

Parameters

- base – SFA peripheral base address.

FSL_SFA_DRIVER_VERSION

SFA driver version 2.1.8.

SFA_MEASUREMENT_START_TIMEOUT

Max loops to wait for SFA measurement started.

This parameter defines how many loops to check completion before return timeout. If defined as 0, driver will wait forever until completion.

SFA_CUT_COUNTER_STOP_TIMEOUT

Max loops to wait for SFA CUT counter has stopped.

This parameter defines how many loops to check completion before return timeout. If defined as 0, driver will wait forever until completion.

SFA_REF_COUNTER_STOP_TIMEOUT

Max loops to wait for SFA REF counter has stopped.

This parameter defines how many loops to check completion before return timeout. If defined as 0, driver will wait forever until completion.

SFA status return codes.

enumeration _sfa_status

Values:

enumerator kStatus_SFA_MeasurementCompleted

Measurement completed

enumerator kStatus_SFA_ReferenceCounterTimeout

Reference counter timeout

enumerator kStatus_SFA_CUTCounterTimeout

CUT counter timeout

enumerator kStatus_SFA_CUTClockFreqLessThanMinLimit

CUT clock frequency less than minimum limit

enumerator kStatus_SFA_CUTClockFreqGreaterThanMaxLimit

CUT clock frequency greater than maximum limit

enum _sfa_status_flags

List of SFA status flags.

The following status register flags can be cleared on any write to REF_CNT.

- kSFA_RefStoppedFlag
- kSFA_CutStoppedFlag
- kSFA_MeasurementStartedFlag
- kSFA_ReferenceCounterTimeOutFlag

Note: These enums are meant to be OR'd together to from a bit mask.

Values:

enumerator kSFA_RefStoppedFlag

Reference counter stopped flag

enumerator kSFA_CutStoppedFlag
CUT counter stopped flag

enumerator kSFA_MeasurementStartedFlag
Measurement Started flag

enumerator kSFA_ReferenceCounterTimeOutFlag
Reference counter time out flag

enumerator kSFA_InterruptRequestFlag
SFA interrupt request flag

enumerator kSFA_FreqGreaterThanMaxInterruptFlag
FREQ_GT_MAX interrupt flag

enumerator kSFA_FreqLessThanMinInterruptFlag
FREQ_LT_MIN interrupt flag

enumerator kSFA_AllStatusFlags

enum _sfa_interrupts_enable
List of SFA interrupt.

Values:

enumerator kSFA_InterruptEnable
SFA interrupt enable

enumerator kSFA_FreqGreaterThanMaxInterruptEnable
FREQ_GT_MAX interrupt enable

enumerator kSFA_FreqLessThanMinInterruptEnable
FREQ_LT_MIN interrupt enable

enum _sfa_measurement_mode
List of SFA measurement mode(Please check the mode configuration according to the manual).

Values:

enumerator kSFA_FrequencyMeasurement0
Frequency measurement performed with REF frequency > CUT frequency

enumerator kSFA_FrequencyMeasurement1
Frequency measurement performed with REF frequency < CUT frequency

enumerator kSFA_CUTPeriodMeasurement
CUT period measurement performed

enumerator kSFA_TriggerBasedMeasurement

Trigger based measurement performed

enum _sfa_cut_select

List of CUT which is connected to the CUT counter (Please refer to the manual for configuration).

Values:

enumerator kSFA_CUTSelect0

enumerator kSFA_CUTSelect1

enumerator kSFA_CUTSelect2

enumerator kSFA_CUTSelect3

enumerator kSFA_CUTSelect4

enumerator kSFA_CUTSelect5

enumerator kSFA_CUTSelect6

enumerator kSFA_CUTSelect7

enumerator kSFA_CUTSelect8

enumerator kSFA_CUTSelect9

enumerator kSFA_CUTSelect10

enumerator kSFA_CUTSelect11

enumerator kSFA_CUTSelect12

enumerator kSFA_CUTSelect13

enumerator kSFA_CUTSelect14

enumerator kSFA_CUTSelect15

enum _sfa_ref_select

List of REF which is connected to the REF counter (Please refer to the manual for configuration).

Values:

enumerator kSFA_REFSelect0

enumerator kSFA_REFSelect1

enumerator kSFA_REFSelect2

enum _sfa_trigger_start_select

List of Signal MUX for Trigger Based Measurement Start.

Values:

enumerator kSFA_TriggerStartSelect0

enumerator kSFA_TriggerStartSelect1

enum _sfa_trigger_end_select

List of Signal MUX for Trigger Based Measurement End.

Values:

enumerator kSFA_TriggerEndSelect0

enumerator kSFA_TriggerEndSelect1

enum _sfa_trigger_start_polarity

List of Trigger Start Polarity.

Values:

enumerator kSFA_TriggerStartPolarityRiseEdge

Rising edge will begin the measurement sequence

enumerator kSFA_TriggerStartPolarityFallEdge

Falling edge will begin the measurement sequence

enum _sfa_trigger_end_polarity

List of Trigger End Polarity.

Values:

enumerator kSFA_TriggerEndPolarityRiseEdge

Rising edge will end the measurement sequence

enumerator kSFA_TriggerEndPolarityFallEdge

Falling edge will end the measurement sequence

typedef void (*sfa_callback_t)(*status_t* status)

sfa measure completion callback function pointer type

This callback can be used in non blocking IRQHandler. Specify the callback you want in the call to *SFA_InstallCallback()*.

Param status

The runtime measurement status. kStatus_SFA_MeasurementCompleted: The measurement completes. kStatus_SFA_ReferenceCounterTimeout: Reference

counter timeout happenes. kStatus_SFA_CUTCounterTimeout: CUT counter timeout happenes.

typedef enum *_sfa_measurement_mode* sfa_measurement_mode_t

List of SFA measurement mode(Please check the mode configuration according to the manual).

typedef enum *_sfa_cut_select* sfa_cut_select_t

List of CUT which is connected to the CUT counter (Please refer to the manual for configuration).

typedef enum *_sfa_ref_select* sfa_ref_select_t

List of REF which is connected to the REF counter (Please refer to the manual for configuration).

typedef enum *_sfa_trigger_start_select* sfa_trigger_start_select_t

List of Signal MUX for Trigger Based Measurement Start.

typedef enum *_sfa_trigger_end_select* sfa_trigger_end_select_t

List of Signal MUX for Trigger Based Measurement End.

typedef enum *_sfa_trigger_start_polarity* sfa_trigger_start_polarity_t

List of Trigger Start Polarity.

typedef enum *_sfa_trigger_end_polarity* sfa_trigger_end_polarity_t

List of Trigger End Polarity.

typedef struct *_sfa_init_config* sfa_config_t

Structure with setting to initialize the SFA module.

This structure holds configuration setting for the SFA peripheral. To initialize this structure to reasonable defaults, call the *SFA_GetDefaultConfig()* function and pass a pointer to your configuration structure instance.

SFA_CUT_CLK_Enable(val)

struct __sfa_init_config

#include <fsl_sfa.h> Structure with setting to initialize the SFA module.

This structure holds configuration setting for the SFA peripheral. To initialize this structure to reasonable defaults, call the *SFA_GetDefaultConfig()* function and pass a pointer to your configuration structure instance.

Public Members

sfa_measurement_mode_t mode

measurement mode

sfa_cut_select_t cutSelect

Select clock connected to the clock under test counter

sfa_ref_select_t refSelect

Selcet REF connected the bus clock

uint8_t prediv

Integer divide of the Input CUT signal

sfa_trigger_start_select_t trigStart

Select the signal will be used to end a trigger based measurement

sfa_trigger_start_polarity_t startPolarity

Select the polarity of the start trigger signal

sfa_trigger_end_select_t trigEnd

Select the signal will be used to commence a trigger based measurement

sfa_trigger_end_polarity_t endPolarity

Select the polarity of the end trigger signal

bool enableTrigMeasurement

false: The measurement will start by default with a dummy write to the CUT counter; true : The measurement will start after receiving a dummy write to the REF_CNT followed by receiving the trigger edge

bool enableCUTPin

Control the connection of the clock under test to an external pin.

uint32_t refTarget

Reference counter target counts

uint32_t cutTarget

CUT counter target counts

2.63 SMSCM: Secure Miscellaneous System Control Module

FSL_MSCM_DRIVER_VERSION

SMSCM driver version 2.0.0.

enum _smscm_debug

SMSCM debug enable type.

Values:

enumerator kSMSCM_InvasiveDebug

enumerator kSMSCM_SecureInvasiveDebug

enumerator kSMSCM_NonInvasiveDebug

enumerator kSMSCM_SecureNonInvasiveDebug

enumerator kSMSCM_AltInvasiveDebug

enumerator kSMSCM_AltDebug

enum _smscm_mem

SMSCM On-Chip Memory Descriptor Register.

Values:

enumerator kSMSCM_Mem0

enumerator kSMSCM_Mem2

enumerator kSMSCM_Mem3

enumerator kSMSCM_Mem5

enum _smscm_ecc_ctrl

SMSCM ECC control type of On-Chip Memory.

Values:

enumerator kSMSCM_EccDisable

enumerator kSMSCM_EccEnableOnWrite

enumerator kSMSCM_EccEnableOnRead

enumerator kSMSCM_EccEnableOnWriteAndRead

typedef enum *_smscm_mem* smscm_mem_t

SMSCM On-Chip Memory Descriptor Register.

typedef enum *_smscm_ecc_ctrl* smscm_ecc_ctrl_t

SMSCM ECC control type of On-Chip Memory.

typedef struct *smscm_ecc_fault_attr* smscm_ecc_fault_attr_t

SMSCM attribute.

static inline void SMSCM_EnableDebug(SMSCM_Type *base, uint32_t debugToEnable)

Enable the Debug function. DebugToEnable could be bitwise OR of _smscm_debug.

Parameters

- base – SMSCM peripheral address.
- debugToEnable – debug enable type.

static inline void SMSCM_DisableDebug(SMSCM_Type *base, uint32_t debugToDisable)

Disables the Debug function. DebugToDisable could be bitwise OR of _smscm_debug.

Parameters

- base – SMSCM peripheral address.
- debugToDisable – debug disable type.

static inline void SMSCM_DebugLock(SMSCM_Type *base)

Lock the debug function.

Parameters

- base – SMSCM peripheral base address.

static inline uint32_t SMSCM_GetSecurityCount(SMSCM_Type *base)

Get value in Security Counter Register (SCTR).

Parameters

- base – SMSCM peripheral base address.

Returns

SMSCM SCTR value.

static inline void SMSCM_SetSecurityCount(SMSCM_Type *base, uint32_t val)

Set value in Security Counter Register (SCTR).

Parameters

- base – SMSCM peripheral base address.
- val – SCTR value to set.

static inline void SMSCM_IncreaseSecurityCount(SMSCM_Type *base, uint32_t val)

Write value to be plused in Security Counter Register (SCTR).

The entire contents of the write data word are added to the security counter, and next-state SCTR =current-state SCTR + DATA32.

Parameters

- base – SMSCM peripheral base address.
- val – SCTR value to plus.

static inline void SMSCM_DecreaseSecurityCount(SMSCM_Type *base, uint32_t val)

Write value to be minused in Security Counter Register (SCTR).

The entire contents of the write data word are added to the security counter, and next-state SCTR =current-state SCTR - DATA32.

Parameters

- base – SMSCM peripheral base address.
- val – SCTR value to be minused.

static inline void SMSCM_IncreaseSecurityCountBy1(SMSCM_Type *base)

Increase security counter rigister by 1.

Parameters

- base – SMSCM peripheral base address.

static inline void SMSCM_DecreaseSecurityCountBy1(SMSCM_Type *base)

Decrease security counter rigister by 1.

Parameters

- base – SMSCM peripheral base address.

static inline void SMSCM_LockMemControlReg(SMSCM_Type *base, *smscm_mem_t* mem)

Lock the on-chip memory descriptor. This register bit provides a mechanism to "lock" the configuration state defined by OCMDRn[11:0]. Once asserted, attempted writes to the OCMDRn[11:0] register are ignored until the next reset clears the flag.

Parameters

- base – SMSCM peripheral address.
- mem – Select OCMDRn to enable read-only mode.

static inline void SMSCM_EnableFlashCache(SMSCM_Type *base, bool enable)

Enable or disable the on-chip memory flash cache.

Parameters

- base – SMSCM peripheral address.
- enable – Use true to enable, false to disable.

static inline void SMSCM_EnableFlashInstructionCache(SMSCM_Type *base, bool enable)

Enable flash instruction cache.

Parameters

- base – SMSCM peripheral address.
- enable – Use true to enable, false to disable.

static inline void SMSCM_EnableFlashDataCache(SMSCM_Type *base, bool enable)

Enable flash data cache.

Parameters

- base – SMSCM peripheral address.
- enable – Use true to enable, false to disable.

static inline void SMSCM_ClearFlashCache(SMSCM_Type *base)

Clear the on-chip memory flash cache.

Parameters

- base – SMSCM peripheral address.

static inline void SMSCM_LockFlashIFR1(SMSCM_Type *base)

Lock IFR1 by flash controller.

Parameters

- base – SMSCM peripheral address.

static inline void SMSCM_EnableFlashSpeculate(SMSCM_Type *base, bool enable)

SMSCM Flash Speculate enable.

Parameters

- base – SMSCM peripheral address.
- enable – Use true to enable, false to disable.

static inline void SMSCM_EnableDataPrefetch(SMSCM_Type *base, bool enable)

SMSCM Data Prefetch enable.

Parameters

- base – SMSCM peripheral address.
- enable – Use true to enable, false to disable.

static inline void SMSCM_EnableFlashDataNonCorrectableBusError(SMSCM_Type *base, bool enable)

Disable non-correctable bus errors on flash data fetches.

Parameters

- base – SMSCM peripheral address.
- enable – Use true to enable, false to disable.

static inline void SMSCM_EnableFlashInstructionNonCorrectableBusError(SMSCM_Type *base, bool enable)

Disable non-correctable bus errors on flash instruction fetches.

Parameters

- base – SMSCM peripheral address.
- enable – Use true to enable, false to disable.

static inline void SMSCM_SetMemEccControl(SMSCM_Type *base, *smscm_mem_t* mem, *smscm_ecc_ctrl_t* eccCtrl)

Select ecc control type in OCMDRn.

Parameters

- base – SMSCM peripheral address.
- mem – Select OCMDRn.
- eccCtrl – Select ecc control type.

static inline void SMSCM_EnableEccReport(SMSCM_Type *base)

Enable RAM ECC 1 bit and non-correctable reporting.

Parameters

- base – SMSCM peripheral base address.

static inline bool SMSCM_GetEccValid(SMSCM_Type *base)

Get the ECC location valid states.

Parameters

- base – SMSCM peripheral base address.

Returns

State of ECC Error Location field.

static inline uint8_t SMSCM_GetEccLocation(SMSCM_Type *base)

Get the ECC location.

Parameters

- base – SMSCM peripheral base address.

Returns

ECC fault location.

void SMSCM_ClearEccError(SMSCM_Type *base, uint8_t errLocation)

Clear each 1-bit correctable or non-correctable error.

Parameters

- base – SMSCM peripheral address.
- errLocation – ECC Error Location

static inline uint32_t SMSCM_GetEccAddress(SMSCM_Type *base)

Get the ECC fault address.

Parameters

- base – SMSCM peripheral base address.

Returns

ECC fault address.

void SMSCM_GetEccAttribute(SMSCM_Type *base, *smscm_ecc_fault_attr_t* *eccAttribute)

Get ECC attribute.

Parameters

- base – SMSCM peripheral address.
- eccAttribute – Ecc attribute.

static inline uint32_t SMSCM_GetEccFaultDataHigh(SMSCM_Type *base)

Get ECC Fault Data High.

This read-only field specifies the upper 32-bit read data word (data[63:32]) from the last captured ECCevent. For ECC events that occur in 32-bit RAMs, this 32-bit field will return 32'h0.

Parameters

- base – SMSCM peripheral base address.

Returns

The higher 32-bit read data word.

static inline uint32_t SMSCM_GetEccFaultDataLow(SMSCM_Type *base)

Get ECC Fault Data Low.

Parameters

- base – SMSCM peripheral base address.

Returns

The lower 32-bit read data word.

struct smscm_ecc_fault_attr

#include <fsl_smscm.h> SMSCM attribute.

2.64 SPC: System Power Control driver

SPC status enumeration.

Values:

enumerator kStatus_SPC_Busy

The SPC instance is busy executing any type of power mode transition.

enumerator kStatus_SPC_DCDCLowDriveStrengthIgnore

DCDC Low drive strength setting be ignored for LVD/HVD enabled.

enumerator kStatus_SPC_DCDCPulseRefreshModeIgnore
DCDC Pulse Refresh Mode drive strength setting be ignored for LVD/HVD enabled.

enumerator kStatus_SPC_SYSLDOOverDriveVoltageFail
SYS LDO regulate to Over drive voltage failed for SYS LDO HVD must be disabled.

enumerator kStatus_SPC_SYSLDOLowDriveStrengthIgnore
SYS LDO Low driver strength setting be ignored for LDO LVD/HVD enabled.

enumerator kStatus_SPC_CORELDOLowDriveStrengthIgnore
CORE LDO Low driver strength setting be ignored for LDO LVD/HVD enabled.

enumerator kStatus_SPC_CORELDOVoltageWrong
Core LDO voltage is wrong.

enumerator kStatus_SPC_CORELDOVoltageSetFail
Core LDO voltage set fail.

enumerator kStatus_SPC_BandgapModeWrong
Selected Bandgap Mode wrong.

enum _spc_voltage_detect_flags
Voltage Detect Status Flags.

Note: VDD_SYS vs VDD1P8 — these flags correspond to DIFFERENT physical rails on different derivatives, not a renamed register. The registers at SPC offset 0x138 (VD_SYS_CFG vs VD_VDD1P8_CFG) and their feature flags (FSL_FEATURE_SPC_HAS_VDD_SYS vs FSL_FEATURE_SPC_HAS_VDD1P8_LVD) are intentionally separate and must not be merged.

- VDD_SYS variant: monitors VDD_SYS (~1.8 V from LDO_SYS, powers AON domain); supports HVD + LVD with LVDRE/LVDIE/HVDRE/HVDIE/LVSEL fields.
- VDD1P8 variant: monitors VDD_FRO (~1.6 V from LDO_FRO, powers FRO block); supports LVD only with LVDIE field. AON on these parts is supplied by VDD_SYS_LV (from LDO_LV_SYS) and has no user-accessible detector.

Values:

enumerator kSPC_IOVDDHighVoltageDetectFlag
IO VDD High-Voltage detect flag.

enumerator kSPC_SystemVDDHighVoltageDetectFlag
System VDD High-Voltage detect flag.

enumerator kSPC_CoreVDDHighVoltageDetectFlag
Core VDD High-Voltage detect flag.

enumerator kSPC_IOVDDLowVoltageDetectFlag
IO VDD Low-Voltage detect flag.

enumerator kSPC_VDD1P8LowVoltageDetectFlag
VDD1P8 Low-Voltage detect flag. Monitors VDD_FRO; see enum

Note: .

enumerator kSPC_SystemVDDLowVoltageDetectFlag
System VDD Low-Voltage detect flag.

enumerator kSPC_CoreVDDLowVoltageDetectFlag
Core VDD Low-Voltage detect flag.

enum _spc_power_domains
SPC power domain isolation status.

Values:

enumerator kSPC_MAINPowerDomainRetain
Peripherals and IO pads retain in MAIN Power Domain.

enumerator kSPC_WAKEPowerDomainRetain
Peripherals and IO pads retain in WAKE Power Domain.

enumerator kSPC_2P4GPowerDoaminRetain
Peripherals and IO pads retion in 2.4G Power Domain.

enumerator kSPC_INFRAPowerDomainRetain
Peripherals and IO pads retain in INFRA Power Domain.

enum _spc_power_domain_id
The enumeration of spc power domain, the connected power domain is chip specfic, please refer to chip's RM for details.

Values:

enumerator kSPC_PowerDomain0
Power domain0, the connected power domain is chip specific.

enumerator kSPC_PowerDomain1
Power domain1, the connected power domain is chip specific.

enumerator kSPC_PowerDomain2
Power domain2, the connected power domain is chip specific.

enumerator kSPC_PowerDomain3
Power domain3, the connected power domain is chip specific.

enum _spc_power_domain_low_power_mode
The enumeration of Power domain's low power mode.

Values:

enumerator kSPC_SleepWithSYSClockRunning
Power domain request SLEEP mode with SYS clock running.

enumerator kSPC_SleepWithSysClockOff
Power domain request SLEEP mode with SYS clock off.

enumerator kSPC_DeepSleepSysClockOff
Power domain request DEEP SLEEP mode with SYS clock off.

enumerator kSPC_PowerDownWithSysClockOff
Power domain request POWER DOWN mode with SYS clock off.

enumerator kSPC_DeepPowerDownWithSysClockOff
Power domain request DEEP POWER DOWN mode with SYS clock off.

enum _spc_lowPower_request_pin_polarity
SPC low power request output pin polarity.
Values:

enumerator kSPC_HighTruePolarity
Control the High Polarity of the Low Power Reqest Pin.

enumerator kSPC_LowTruePolarity
Control the Low Polarity of the Low Power Reqest Pin.

enum _spc_lowPower_request_output_override
SPC low power request output override.
Values:

enumerator kSPC_LowPowerRequestNotForced
Not Forced.

enumerator kSPC_LowPowerRequestReserved
Reserved.

enumerator kSPC_LowPowerRequestForcedLow
Forced Low (Ignore LowPower request output polarity setting.)

enumerator kSPC_LowPowerRequestForcedHigh
Forced High (Ignore LowPower request output polarity setting.)

enum _spc_bandgap_mode
SPC Bandgap mode enumeration in Active mode or Low Power mode.
Values:

enumerator kSPC_BandgapDisabled
Bandgap disabled.

enumerator kSPC_BandgapEnabledBufferDisabled
Bandgap enabled with Buffer disabled.

enumerator kSPC_BandgapEnabledBufferEnabled
Bandgap enabled with Buffer enabled.

enumerator kSPC_BandgapReserved
Reserved.

enum _spc_dcdc_voltage_level
DCDC regulator voltage level enumeration in Active mode or Low Power Mode.

Values:

enumerator kSPC_DCDC_SafeModeVoltage
DCDC VDD Regulator regulate to Safe-Mode Voltage.

enumerator kSPC_DCDC_NormalVoltage
DCDC VDD Regulator regulate to Normal Voltage.

enumerator kSPC_DCDC_MidVoltage
DCDC VDD Regulator regulate to Mid Voltage.

enumerator kSPC_DCDC_LowUnderVoltage
DCDC VDD Regulator regulate to Low Under Voltage.

enum _spc_dcdc_drive_strength
DCDC regulator Drive Strength enumeration in Active mode or Low Power Mode.

Values:

enumerator kSPC_DCDC_PulseRefreshMode
DCDC VDD Regulator Drive Strength set to Pulse Refresh Mode. This enum member is only useful for Low Power Mode config.

enumerator kSPC_DCDC_LowDriveStrength
DCDC VDD regulator Drive Strength set to low.

enumerator kSPC_DCDC_NormalDriveStrength
DCDC VDD regulator Drive Strength set to Normal.

enumerator kSPC_DCDC_Reserved
Reserved.

enum _spc_sys_ldo_voltage_level
SYS LDO regulator voltage level enumeration in Active mode.

Values:

enumerator kSPC_SysLDO_NormalVoltage
SYS LDO VDD Regulator regulate to Normal Voltage(1.8V).

enumerator kSPC_SysLDO_OverDriveVoltage
SYS LDO VDD Regulator regulate to Over Drive Voltage(2.5V).

enum _spc_sys_ldo_drive_strength
SYS LDO regulator Drive Strength enumeration in Active mode or Low Power mode.
Values:

enumerator kSPC_SysLDO_LowDriveStrength
SYS LDO VDD regulator Drive Strength set to low.

enumerator kSPC_SysLDO_NormalDriveStrength
SYS LDO VDD regulator Drive Strength set to Normal.

enum _spc_core_ldo_voltage_level
Core LDO regulator voltage level enumeration in Active mode or Low Power mode.
Values:

enumerator kSPC_CoreLDO_NormalVoltage
Core LDO VDD regulator regulate to Normal Voltage.

enumerator kSPC_CoreLDO_MidDriveVoltage
Core LDO VDD regulator regulate to Mid Drive Voltage.

enumerator kSPC_CoreLDO_UnderDriveVoltage
Core LDO VDD regulator regulate to Under Drive Voltage.

enumerator kSPC_CoreLDO_SafeModeVoltage
Core LDO VDD regulator regulate to Safe-Mode Voltage.

enum _spc_core_ldo_drive_strength
CORE LDO VDD regulator Drive Strength enumeration in Low Power mode.
Values:

enumerator kSPC_CoreLDO_LowDriveStrength
Core LDO VDD regulator Drive Strength set to low.

enumerator kSPC_CoreLDO_NormalDriveStrength
Core LDO VDD regulator Drive Strength set to Normal.

enum _spc_low_voltage_level_select
System/IO VDD Low-Voltage Level Select.
Values:

enumerator kSPC_LowVoltageNormalLevel
Trip point set to Normal level.

enumerator kSPC_LowVoltageSafeLevel
Trip point set to Safe level.

enum _spc_sram_operat_voltage

SRAM operate voltage enumeration.

Values:

enumerator kSPC_SRAM_OperatVoltage1P0V

SRAM operate voltage set to 1.0V.

enumerator kSPC_SRAM_OperatVoltage1P1V

SRAM operate voltage set to 1.1V.

enum _spc_hp_request_override_option

The enumeration of high power request override option.

Values:

enumerator kSPC_HpRequestOverrideDisable

Disable high power request override feature.

enumerator kSPC_HpRequestOverride0

Enable high power request override feature and force value as 0.

enumerator kSPC_HpReqestOverride1

Enable high power request override feature and force value as 1.

enum _spc_vdd_core_glitch_ripple_counter_select

Used to select output of 4-bit ripple counter is used to monitor a glitch on VDD core.

Values:

enumerator kSPC_selectBit0Of4bitRippleCounter

Select bit-0 of 4-bit Ripple Counter to detect glitch on VDD Core.

enumerator kSPC_selectBit1Of4bitRippleCounter

Select bit-1 of 4-bit Ripple Counter to detect glitch on VDD Core.

enumerator kSPC_selectBit2Of4bitRippleCounter

Select bit-2 of 4-bit Ripple Counter to detect glitch on VDD Core.

enumerator kSPC_selectBit3Of4bitRippleCounter

Select bit-3 of 4-bit Ripple Counter to detect glitch on VDD Core.

typedef enum *_spc_power_domain_id* spc_power_domain_id_t

The enumeration of spc power domain, the connected power domain is chip specfic, please refer to chip's RM for details.

typedef enum *_spc_power_domain_low_power_mode* spc_power_domain_low_power_mode_t

The enumeration of Power domain's low power mode.

typedef enum *_spc_lowPower_request_pin_polarity* spc_lowpower_request_pin_polarity_t

SPC low power request output pin polarity.

typedef enum *_spc_lowPower_request_output_override* spc_lowpower_request_output_override_t
SPC low power request output override.

typedef enum *_spc_bandgap_mode* spc_bandgap_mode_t
SPC Bandgap mode enumeration in Active mode or Low Power mode.

typedef enum *_spc_dcdc_voltage_level* spc_dcdc_voltage_level_t
DCDC regulator voltage level enumeration in Active mode or Low Power Mode.

typedef enum *_spc_dcdc_drive_strength* spc_dcdc_drive_strength_t
DCDC regulator Drive Strength enumeration in Active mode or Low Power Mode.

typedef enum *_spc_sys_ldo_voltage_level* spc_sys_ldo_voltage_level_t
SYS LDO regulator voltage level enumeration in Active mode.

typedef enum *_spc_sys_ldo_drive_strength* spc_sys_ldo_drive_strength_t
SYS LDO regulator Drive Strength enumeration in Active mode or Low Power mode.

typedef enum *_spc_core_ldo_voltage_level* spc_core_ldo_voltage_level_t
Core LDO regulator voltage level enumeration in Active mode or Low Power mode.

typedef enum *_spc_core_ldo_drive_strength* spc_core_ldo_drive_strength_t
CORE LDO VDD regulator Drive Strength enumeration in Low Power mode.

typedef enum *_spc_low_voltage_level_select* spc_low_voltage_level_select_t
System/IO VDD Low-Voltage Level Select.

typedef enum *_spc_sram_operat_voltage* spc_sram_operat_voltage_t
SRAM operate voltage enumeration.

typedef struct *_spc_lowpower_request_config* spc_lowpower_request_config_t
Low Power Request output pin configuration.

typedef struct *_spc_intergrated_power_switch_config* spc_intergrated_power_switch_config_t
Integrated power switch configuration.

Note: Legacy structure, will be removed.

typedef struct *_spc_active_mode_core_ldo_option* spc_active_mode_core_ldo_option_t
Core LDO regulator options in Active mode.

typedef struct *_spc_active_mode_sys_ldo_option* spc_active_mode_sys_ldo_option_t
System LDO regulator options in Active mode.

typedef struct *_spc_active_mode_dcdc_option* spc_active_mode_dcdc_option_t
DCDC regulator options in Active mode.

typedef struct *_spc_lowpower_mode_core_ldo_option* spc_lowpower_mode_core_ldo_option_t
Core LDO regulator options in Low Power mode.

typedef struct *_spc_lowpower_mode_sys_ldo_option* spc_lowpower_mode_sys_ldo_option_t
System LDO regulator options in Low Power mode.

typedef struct *_spc_lowpower_mode_dcdc_option* spc_lowpower_mode_dcdc_option_t
DCDC regulator options in Low Power mode.

typedef struct *_spc_voltage_detect_option* spc_voltage_detect_option_t
CORE/SYS/IO VDD Voltage Detect options.

typedef struct *_spc_dcdc_burst_config* spc_dcdc_burst_config_t
DCDC Burst configuration.

typedef struct *_spc_core_voltage_detect_config* spc_core_voltage_detect_config_t
Core Voltage Detect configuration.

typedef struct *_spc_vdd1p8_voltage_detect_config* spc_vdd1p8_voltage_detect_config_t
VDD1P8 Voltage Detect configuration.

typedef struct *_spc_system_voltage_detect_config* spc_system_voltage_detect_config_t
System Voltage Detect Configuration.

typedef struct *_spc_io_voltage_detect_config* spc_io_voltage_detect_config_t
IO Voltage Detect Configuration.

typedef struct *_spc_active_mode_regulators_config* spc_active_mode_regulators_config_t
Active mode configuration.

typedef struct *_spc_lowpower_mode_regulators_config* spc_lowpower_mode_regulators_config_t
Low Power Mode configuration.

typedef enum *_spc_hp_request_override_option* spc_hp_override_request_option_t
The enumeration of high power request override option.

typedef *spc_active_mode_dcdc_option_t* spc_hp_mode_dcdc_option_t

typedef *spc_active_mode_sys_ldo_option_t* spc_hp_mode_sys_ldo_option_t

typedef *spc_active_mode_core_ldo_option_t* spc_hp_mode_core_ldo_option_t

typedef *spc_active_mode_regulators_config_t* spc_hp_mode_regulators_config_t

typedef enum *_spc_vdd_core_glitch_ripple_counter_select*
spc_vdd_core_glitch_ripple_counter_select_t
Used to select output of 4-bit ripple counter is used to monitor a glitch on VDD core.

typedef struct *_spc_vdd_core_glitch_detector_config* spc_vdd_core_glitch_detector_config_t

The configuration of VDD Core glitch detector.

SPC_BUSY_TIMEOUT

Max loops to wait for SPC to stop being busy.

The BUSY bitfield will be set when the SPC performs any kind of power mode transition in active mode or any chip low power mode. You need to wait until this flag is cleared before changing the power mode configuration registers. This parameter defines how many loops to check completion before return timeout. If defined as 0, the driver will wait until completion.

SPC_SRAM_ACK_TIMEOUT

Max loops to wait for SPC to stop being busy.

When changing SRAM voltage, need to wait for SRAM voltage update request acknowledgment. This parameter defines how many loops to check completion before return timeout. If defined as 0, the driver will wait until completion.

SPC_DCDC_ACK_TIMEOUT

Max loops to wait for DCDC burst completed.

When the DCDC burst is requested, it is necessary to wait for the DCDC burst to complete. This parameter defines how many loops to check completion before return timeout. If defined as 0, the driver will wait until it completes.

SPC_VD_STAT_VALID_FLAGS_BASE_MASK

SPC_VD_STAT_VALID_FLAGS_SYS_MASK

SPC_VD_STAT_VALID_FLAGS_MASK

SPC_HP_CNFG_CTRL_OVERRIDE_OPT_MASK

SPC_HP_CNFG_CTRL_OVERRIDE_OPT_SHIFT

SPC_HP_CNFG_CTRL_OVERRIDE_OPT(x)

typedef struct *_spc_dcdc_ramp_config* spc_dcdc_ramp_config_t

DCDC output voltage ramp control configuration.

Used by *SPC_SetDCDCRampControlConfig*. The ramp trim fields select the DCDC output voltage ramp slope; valid values are 0..7.

static inline bool SPC_CheckHPCfgSelected(SPC_Type *base)

Check if HP_CFG selected as active configuration register.

Parameters

- base – SPC peripheral base address.

Return values

- false – ACTIVE_CFG selected as the active configuration register.
- true – HP_CFG selected as the active configuration register.

static inline void SPC_EnableHighPowerRequest(SPC_Type *base, bool enable)

Enable/disable high power request feature.

Parameters

- base – SPC peripheral base address.
- enable – Used to specify enable/disable the high power request feature:
  - **true** Enable high power request;
  - **false** Disable high power reques.

static inline void SPC_OverrideHighPowerRequest(SPC_Type *base, *spc_hp_override_request_option_t* opt)

Override high power request manually.

Parameters

- base – SPC peripheral base address.
- opt – Specify the option of high power override request, please refer to *spc_hp_override_request_option_t*.

static inline *spc_core_ldo_voltage_level_t* SPC_GetHighPowerModeCoreLDOVDDVoltageLevel(SPC_Type *base)

Get voltage level of CORE LDO in high power mode.

Parameters

- base – SPC peripheral base address.

Returns

The voltage level of CORE LDO in high power mode, please refer to *spc_core_ldo_voltage_level_t*.

static inline *spc_bandgap_mode_t* SPC_GetHighPowerModeBandgapMode(SPC_Type *base)

Get bandgap mode in high power mode.

Parameters

- base – SPC peripheral base address.

Returns

The bandgap mode in high power mode, please refer to *spc_bandgap_mode_t*.

static inline uint32_t SPC_GetHighPowerModeVoltageDetectStatus(SPC_Type *base)

Get enabled state of all voltage detectors.

Parameters

- base – SPC peripheral base address.

Returns

All enabled status of all voltage detectors, 1b1 means the corrsponding voltage detector is enabled.

status_t SPC_SetHighPowerModeBandgapModeConfig(SPC_Type *base, *spc_bandgap_mode_t* mode)

Set bandgap mode in high power mode.

Parameters

- base – SPC peripheral base address.
- mode – Specify the bandgap mode in high power mode.

Return values

- kStatus_SPC_BandgapModeWrong – The Bandgap can not be disabled in high power mode.
- kStatus_Success – Config Bandgap mode in high power mode successful.

static inline void SPC_EnableHighPowerModeCMPBandgapBuffer(SPC_Type *base, bool enable)

Enable/disable CMP buffer in high power mode.

Parameters

- base – SPC peripheral base address.
- enable – Used to enable/disable CMP buffer:
  - **true** Enable CMP buffer in high power mode;
  - **false** Disable CMP buffer in high power mode.

static inline void SPC_DisableHighPowerModeVddCoreGlitchDetect(SPC_Type *base, bool disable)

Disable/enable VDD Core Glitch detect feature in high power mode.

Parameters

- base – SPC peripheral base address.
- disable – Used to disable/enable VDD Core Glitch detect feature:
  - **true** Disable VDD Core Glitch detect feature;
  - **false** Enable VDD Core Glitch detect feature.

status_t SPC_SetHighPowerModeCoreLDORegulatorConfig(SPC_Type *base, const *spc_hp_mode_core_ldo_option_t* *option)

Configure CORE LDO regulator in high power mode.

Parameters

- base – SPC peripheral base address.
- option – Pointer to the CORE LDO regulator configuration, please refer to *spc_hp_mode_core_ldo_option_t*.

Return values

- kStatus_Success – Config Core LDO regulator in High power mode successful.
- kStatus_SPC_Busy – The SPC instance is busy to execute any type of power mode transition.
- kStatus_SPC_CORELDOLowDriveStrengthIgnore – If any voltage detect enabled, core_ldo's drive strength can not set to low.
- kStatus_SPC_CORELDOVoltageWrong – The selected voltage level in high power mode is not allowed.
- kStatus_Timeout – Timeout occurs while waiting completion.

status_t SPC_SetHighPowerModeSystemLDORegulatorConfig(SPC_Type *base, const *spc_hp_mode_sys_ldo_option_t* *option)

Configure System LDO regulator in high power mode.

Parameters

- base – SPC peripheral base address.
- option – Pointer to the SYSTEM LDO regulator configuration, please refer to *spc_hp_mode_sys_ldo_option_t*.

Return values

- kStatus_Success – Config System LDO regulator in Active power mode successful.
- kStatus_SPC_Busy – The SPC instance is busy to execute any type of power mode transition.
- kStatus_SPC_SYSLDOOverDriveVoltageFail – Fail to regulator to Over Drive Voltage.
- kStatus_SPC_SYSLDOLowDriveStrengthIgnore – Set driver strength to Low will be ignored.
- kStatus_Timeout – Timeout occurs while waiting completion.

status_t SPC_SetHighPowerModeDCDCRegulatorConfig(SPC_Type *base, const *spc_hp_mode_dcdc_option_t* *option)

Configure DCDC regulator in high power mode.

Parameters

- base – SPC peripheral base address.
- option – Pointer to the DCDC regulator configuration, please refer to *spc_hp_mode_dcdc_option_t*.

Return values

- kStatus_Success – Config DCDC regulator in Active power mode successful.
- kStatus_SPC_Busy – The SPC instance is busy to execute any type of power mode transition.
- kStatus_SPC_DCDCLowDriveStrengthIgnore – Set driver strength to Low will be ignored.
- kStatus_Timeout – Timeout occurs while waiting completion.

status_t SPC_EnableActiveModeDCDC2P5VOutput(SPC_Type *base, bool enable)

Enable/disable the DCDC 2.5V output in Active mode.

The DCDC 2.5V output select bit resides in different registers depending on the SPC instance: in the per-mode configuration registers (ACTIVE_CFG/HP_CFG) on some SoCs (e.g. KW43/MCXW70), and in the global DCDC_CFG register on others (e.g. KW45/KW47/MCXW71/MCXW72). This function selects the 2.5V DCDC output, writing whichever register applies. The 2.5V output is intended to be used together with DCDC Normal drive strength and Normal voltage level (see *SPC_SetActiveModeDCDCRegulatorConfig*).

Parameters

- base – SPC peripheral base address.
- enable – Select the DCDC 2.5V output in Active mode.
  - true : Select the 2.5V DCDC output.
  - false : Use the level selected by DCDC_VDD_LVL.

Return values

- kStatus_Success – The 2.5V output selection was applied successfully.
- kStatus_SPC_Busy – The SPC instance is busy to execute any type of power mode transition.
- kStatus_Timeout – Timeout occurs while waiting completion.

status_t SPC_EnableHighPowerModeDCDC2P5VOutput(SPC_Type *base, bool enable)

Enable/disable the DCDC 2.5V output in High Power mode.

Same as *SPC_EnableActiveModeDCDC2P5VOutput* but applies to the High Power mode configuration register (HP_CFG). Only available on SoCs whose 2.5V output select is in the per-mode configuration registers and that have a High Power mode (HP_CFG).

Parameters

- base – SPC peripheral base address.
- enable – Select the DCDC 2.5V output in High Power mode.
  - true : Select the 2.5V DCDC output.
  - false : Use the level selected by DCDC_VDD_LVL.

Return values

- kStatus_Success – The 2.5V output selection was applied successfully.
- kStatus_SPC_Busy – The SPC instance is busy to execute any type of power mode transition.
- kStatus_Timeout – Timeout occurs while waiting completion.

status_t SPC_SetDCDCRampControlConfig(SPC_Type *base, const *spc_dcdc_ramp_config_t* *config)

Configure DCDC output voltage ramp control.

Enabling ramp control smooths DCDC output voltage transitions, for example when changing the DCDC voltage level or drive strength. When ramp control is enabled the ramp enable bit is programmed first and allowed to settle before the ramp trim values are applied, matching the recommended hardware programming sequence.

Parameters

- base – SPC peripheral base address.
- config – Pointer to the ramp control configuration, see *spc_dcdc_ramp_config_t*.

Return values

- kStatus_Success – Ramp control configuration applied successfully.
- kStatus_SPC_Busy – The SPC instance is busy to execute any type of power mode transition.
- kStatus_Timeout – Timeout occurs while waiting completion.

status_t SPC_SetHighPowerModeRegulatorsConfig(SPC_Type *base, const *spc_hp_mode_regulators_config_t* *config)

Set configuration of regulators in high power mode.

Parameters

- base – SPC peripheral base address.
- config – Pointer to the regulator configuration, please refer to *spc_hp_mode_regulators_config_t*.

Return values

- kStatus_Success – Config regulators in High power mode successful.
- kStatus_SPC_BandgapModeWrong – The bandgap mode setting in high power mode is wrong.
- kStatus_SPC_Busy – The SPC instance is busy to execute any type of power mode transition.

- kStatus_SPC_CORELDOVoltageWrong – The selected voltage level in high mode is not allowed.
- kStatus_SPC_SYSLDOOverDriveVoltageFail – Fail to regulator to Over Drive Voltage.
- kStatus_SPC_SYSLDOLowDriveStrengthIgnore – Set driver strength to Low will be ignored.
- kStatus_SPC_DCDCLowDriveStrengthIgnore – Set driver strength to Low will be ignored.

static inline void SPC_EnableHighPowerModeCoreLowVoltageDetect(SPC_Type *base, bool enable)

Enable/disable low voltage detect for VDD_CORE in high power mode.

Parameters

- base – SPC peripheral base address.
- enable – Used to enable/disable low voltage detect feature for VDD_CORE in high power mode:
  - **true** Enable low voltage detect feature for VDD_CORE in high power mode;
  - **false** Disable low voltage detect feature for VDD_CORE in high power mode.

static inline void SPC_EnableHighPowerModeCoreHighVoltageDetect(SPC_Type *base, bool enable)

Enable/disable high voltage detect for VDD_CORE in high power mode.

Parameters

- base – SPC peripheral base address.
- enable – Used to enable/disable high voltage detect feature for VDD_CORE in high power mode:
  - **true** Enable low voltage detect feature for VDD_CORE in high power mode;
  - **false** Disable low voltage detect feature for VDD_CORE in high power mode.

static inline void SPC_EnableHighPowerModeSystemLowVoltageDetect(SPC_Type *base, bool enable)

Enable/disable low voltage detect for VDD_SYS in high power mode.

Parameters

- base – SPC peripheral base address.
- enable – Used to enable/disable low voltage detect feature for VDD_SYS in high power mode:
  - **true** Enable low voltage detect feature for VDD_SYS in high power mode;
  - **false** Disable low voltage detect feature for VDD_SYS in high power mode.

static inline void SPC_EnableHighPowerModeSystemHighVoltageDetect(SPC_Type *base, bool enable)

Enable/disable high voltage detect for VDD_SYS in high power mode.

Parameters

- base – SPC peripheral base address.

- enable – Used to enable/disable high voltage detect feature for VDD_SYS in high power mode:
  - **true** Enable high voltage detect feature for VDD_SYS in high power mode;
  - **false** Disable high voltage detect feature for VDD_SYS in high power mode.

static inline void SPC_EnableHighPowerModeVDD1P8LowVoltageDetect(SPC_Type *base, bool enable)

Enable/disable low voltage detect for VDD1P8 in high power mode.

Parameters

- base – SPC peripheral base address.
- enable – Used to enable/disable low voltage detect feature for VDD1P8 in high power mode:
  - **true** Enable low voltage detect feature for VDD1P8 in high power mode;
  - **false** Disable low voltage detect feature for VDD1P8 in high power mode.

static inline void SPC_EnableHighPowerModeIOLowVoltageDetect(SPC_Type *base, bool enable)

Enable/disable low voltage detect for VDD_IO_ABC in high power mode.

Parameters

- base – SPC peripheral base address.
- enable – Used to enable/disable low voltage detect feature for VDD_IO_ABC in high power mode:
  - **true** Enable low voltage detect feature for VDD_IO_ABC in high power mode;
  - **false** Disable low voltage detect feature for VDD_IO_ABC in high power mode.

static inline void SPC_EnableHighPowerModeIOHighVoltageDetect(SPC_Type *base, bool enable)

Enable/disable high voltage detect for VDD_IO_ABC in high power mode.

Parameters

- base – SPC peripheral base address.
- enable – Used to enable/disable high voltage detect feature for VDD_IO_ABC in high power mode:
  - **true** Enable high voltage detect feature for VDD_IO_ABC in high power mode;
  - **false** Disable high voltage detect feature for VDD_IO_ABC in high power mode.

uint8_t SPC_GetPeriphIOIsolationStatus(SPC_Type *base)

Gets Isolation status for each power domains.

This function gets the status which indicates whether certain peripheral and the IO pads are in a latched state as a result of having been in POWERDOWN mode.

Parameters

- base – SPC peripheral base address.

Returns

Current isolation status for each power domains. See *_spc_power_domains* for details.

static inline void SPC_ClearPeriphIOIsolationFlag(SPC_Type *base)

Clears peripherals and I/O pads isolation flags for each power domains.

This function clears peripherals and I/O pads isolation flags for each power domains. After recovering from the POWERDOWN mode, user must invoke this function to release the I/O pads and certain peripherals to their normal run mode state. Before invoking this function, user must restore chip configuration in particular pin configuration for enabled WUU wakeup pins.

Parameters

- base – SPC peripheral base address.

static inline bool SPC_GetBusyStatusFlag(SPC_Type *base)

Gets SPC busy status flag.

This function gets SPC busy status flag. When SPC executing any type of power mode transition in ACTIVE mode or any of the SOC low power mode, the SPC busy status flag is set and this function returns true. When changing CORE LDO voltage level and DCDC voltage level in ACTIVE mode, the SPC busy status flag is set and this function return true.

Parameters

- base – SPC peripheral base address.

Returns

Ack busy flag. true - SPC is busy. false - SPC is not busy.

static inline bool SPC_CheckLowPowerReqest(SPC_Type *base)

Checks system low power request.

Note: Only when all power domains request low power mode entry, the result of this function is true. That means when all power domains request low power mode entry, the SPC regulators will be controlled by LP_CFG register.

Parameters

- base – SPC peripheral base address.

Returns

The system low power request check result.

- **true** All power domains have requested low power mode and SPC has entered a low power state and power mode configuration are based on the LP_CFG configuration register.
- **false** SPC in active mode and ACTIVE_CFG register control system power supply.

static inline void SPC_ClearLowPowerRequest(SPC_Type *base)

Clears system low power request, set SPC in active mode.

Parameters

- base – SPC peripheral base address.

static inline bool SPC_CheckPowerSwitchState(SPC_Type *base)

Checks power switch state.

Parameters

- base – SPC peripheral base address.

Returns

The state(ON/OFF) of power switch.

- **true** Indicates the power switch is ON.
- **false** Indicates the power switch is OFF.

spc_power_domain_low_power_mode_t SPC_GetPowerDomainLowPowerMode(SPC_Type *base, *spc_power_domain_id_t* powerDomainId)

Gets selected power domain's requested low power mode.

Parameters

- base – SPC peripheral base address.
- powerDomainId – Power Domain Id, please refer to *spc_power_domain_id_t*.

Returns

The selected power domain's requested low power mode, please refer to *spc_power_domain_low_power_mode_t*.

static inline bool SPC_CheckPowerDomainLowPowerRequest(SPC_Type *base, *spc_power_domain_id_t* powerDomainId)

Checks power domain's low power request.

Parameters

- base – SPC peripheral base address.
- powerDomainId – Power Domain Id, please refer to *spc_power_domain_id_t*.

Returns

The result of power domain's low power request.

- **true** The selected power domain requests low power mode entry.
- **false** The selected power domain does not request low power mode entry.

static inline void SPC_ClearPowerDomainLowPowerRequestFlag(SPC_Type *base, *spc_power_domain_id_t* powerDomainId)

Clears selected power domain's low power request flag.

Parameters

- base – SPC peripheral base address.
- powerDomainId – Power Domain Id, please refer to *spc_power_domain_id_t*.

void SPC_SetLowPowerRequestConfig(SPC_Type *base, const *spc_lowpower_request_config_t* *config)

Configs Low power request output pin.

This function config the low power request output pin

Parameters

- base – SPC peripheral base address.
- config – Pointer the *spc_lowpower_request_config_t* structure.

static inline void SPC_SoftwareGatePowerSwitch(SPC_Type *base, bool gate)

Gates/Un-gates power switch in software mode.

Parameters

- base – SPC peripheral base address.
- gate – Used to gate/ungate power switch
  - **true** The power switch will be gated.

– **false** The power switch will be un-gated.

static inline void SPC_PowerModeControlPowerSwitch(SPC_Type *base)

Gates power switch at low power modes entry, and un-gates power switch at low power mode wakeup.

Parameters

- base – SPC peripheral base address.

static inline void SPC_SetWakeUpValue(SPC_Type *base, uint32_t data)

Set the address of the function/image to be executed if chip wake from power down or deep power down mode.

Note: Data written by this function is used by BootROM to quickly retore ARM Core context, or to switch execution to a defined address in Flash/SRAM on WakeUp.

Note: The first word must be SP, and the second word must be PC.

Note: Please remember to calculate the CRC value of the first 48 bytes of image/function and save the result to REGFILE1->REG[0]. The BootROM will check this CRC value, if authenticated successfully then the image/function will be executed.

Parameters

- base – SPC peripheral base address.
- data – The address of the function/image to be executed if wakeup from low power mode.

static inline uint32_t SPC_GetWakeUpValue(SPC_Type *base)

Gets back the WakeUp value.

Parameters

- base – SPC peripheral base address.

Returns

The WakeUp value.

static inline *spc_core_ldo_voltage_level_t* SPC_GetActiveModeCoreLDOVDDVoltageLevel(SPC_Type *base)

Gets CORE LDO VDD Regulator Voltage level.

This function returns the voltage level of CORE LDO Regulator in Active mode.

Parameters

- base – SPC peripheral base address.

Returns

Voltage level of CORE LDO in type of *spc_core_ldo_voltage_level_t* enumeration.

static inline *spc_bandgap_mode_t* SPC_GetActiveModeBandgapMode(SPC_Type *base)

Gets the Bandgap mode in Active mode.

Parameters

- base – SPC peripheral base address.

Returns
Bandgap mode in the type of *spc_bandgap_mode_t* enumeration.

static inline uint32_t SPC_GetActiveModeVoltageDetectStatus(SPC_Type *base)

Gets all voltage detectors status in Active mode.

Parameters

- base – SPC peripheral base address.

Returns
All voltage detectors status in Active mode.

void SPC_SetActiveModeIntegratedPowerSwitchConfig(SPC_Type *base, const *spc_intergrated_power_switch_config_t* *config)

Configs Integrated power switch in active mode.

Note: Legacy API and will be removed.

Parameters

- base – SPC peripheral base address.
- config – Pointer to spc_intergrated_power_switch_config_t pointer.

status_t SPC_SetActiveModeBandgapModeConfig(SPC_Type *base, *spc_bandgap_mode_t* mode)

Configs Bandgap mode in Active mode.

Note: In active mode, because CORELDO_VDD_DS is reserved and set to Normal, so it is impossible to disable Bandgap in active mode

Parameters

- base – SPC peripheral base address.
- mode – The Bandgap mode be selected.

Return values

- kStatus_SPC_BandgapModeWrong – The Bandgap can not be disabled in active mode.
- kStatus_Success – Config Bandgap mode in Active power mode successful.

static inline void SPC_EnableActiveModeCMPBandgapBuffer(SPC_Type *base, bool enable)

Enables/Disable the CMP Bandgap Buffer in Active mode.

Parameters

- base – SPC peripheral base address.
- enable – Enable/Disable CMP Bandgap buffer. true - Enable Buffer Stored Reference voltage to CMP. false - Disable Buffer Stored Reference voltage to CMP.

static inline void SPC_SetActiveModeVoltageTrimDelay(SPC_Type *base, uint16_t delay)

Sets the delay when the regulators change voltage level in Active mode.

Parameters

- base – SPC peripheral base address.

- delay – The number of SPC timer clock cycles.

status_t SPC_SetActiveModeRegulatorsConfig(SPC_Type *base, const *spc_active_mode_regulators_config_t* *config)

Configs regulators in Active mode.

This function provides the method to config all on-chip regulators in active mode.

Parameters

- base – SPC peripheral base address.
- config – Pointer to spc_active_mode_regulators_config_t structure.

Return values

- kStatus_Success – Config regulators in Active power mode successful.
- kStatus_SPC_BandgapModeWrong – The bandgap mode setting in Active mode is wrong.
- kStatus_SPC_Busy – The SPC instance is busy to execute any type of power mode transition.
- kStatus_SPC_CORELDOVoltageWrong – The selected voltage level in active mode is not allowed.
- kStatus_SPC_SYSLDOOverDriveVoltageFail – Fail to regulator to Over Drive Voltage.
- kStatus_SPC_SYSLDOLowDriveStrengthIgnore – Set driver strength to Low will be ignored.
- kStatus_SPC_DCDCLowDriveStrengthIgnore – Set driver strength to Low will be ignored.
- kStatus_Timeout – Timeout occurs while waiting completion.

static inline void SPC_DisableActiveModeVddCoreGlitchDetect(SPC_Type *base, bool disable)

Disable/Enable VDD Core Glitch Detect in Active mode.

Note: State of glitch detect disable feature will be ignored if bandgap is disabled and glitch detect hardware will be forced to OFF state.

Parameters

- base – SPC peripheral base address.
- disable – Used to disable/enable VDD Core Glitch detect feature.
  - **true** Disable VDD Core Low Voltage detect;
  - **false** Enable VDD Core Low Voltage detect.

void SPC_SetLowPowerModeIntegratedPowerSwitchConfig(SPC_Type *base, const *spc_intergrated_power_switch_config_t* *config)

Configs Integrated power switch in Low Power mode.

Note: Legacy API, will be removed.

Parameters

- base – SPC peripheral base address.

- config – Pointer to spc_intergrated_power_switch_config_t pointer.

static inline void SPC_EnableLowPowerModeVDDCWellBias(SPC_Type *base, bool enable)

Enables/Disables VDDC Well Bias in low power mode.

Parameters

- base – SPC peripheral base address.
- enable – Enable/Disable the VDDC Well Bias. true - Enable Vddc Well Bias. false - Disable Vddc Well Bias.

static inline *spc_core_ldo_drive_strength_t* SPC_GetLowPowerCoreLDOVDDDriveStrength(SPC_Type *base)

Gets CORE LDO VDD Drive Strength for Low Power modes.

Parameters

- base – SPC peripheral base address.

Returns

The CORE LDO's VDD Drive Strength.

static inline *spc_core_ldo_voltage_level_t* SPC_GetLowPowerCoreLDOVDDVoltageLevel(SPC_Type *base)

Gets the CORE LDO VDD Regulator Voltage Level for Low Power modes.

Parameters

- base – SPC peripheral base address.

Returns

The CORE LDO VDD Regulator's voltage level.

static inline *spc_bandgap_mode_t* SPC_GetLowPowerModeBandgapMode(SPC_Type *base)

Gets the Bandgap mode in Low Power mode.

Parameters

- base – SPC peripheral base address.

Returns

Bandgap mode in the type of *spc_bandgap_mode_t* enumeration.

static inline uint32_t SPC_GetLowPowerModeVoltageDetectStatus(SPC_Type *base)

Gets the status of all voltage detectors in Low Power mode.

Parameters

- base – SPC peripheral base address.

Returns

The status of all voltage detectors in low power mode.

static inline void SPC_EnableLowPowerModeLowPowerIREF(SPC_Type *base, bool enable)

Enables/Disables Low Power IREF in low power modes.

This function enables/disables Low Power IREF. Low Power IREF can only get disabled in Deep power down mode. In other low power modes, the Low Power IREF is always enabled.

Parameters

- base – SPC peripheral base address.
- enable – Enable/Disable Low Power IREF. true - Enable Low Power IREF for Low Power modes. false - Disable Low Power IREF for Deep Power Down mode.

status_t SPC_SetLowPowerModeBandgapmodeConfig(SPC_Type *base, *spc_bandgap_mode_t* mode)

Configs Bandgap mode in Low Power mode.

This function configs Bandgap mode in Low Power mode. IF user want to disable Bandgap while keeping any of the Regulator in Normal Driver Strength or if any of the High voltage detectors/Low voltage detectors are kept enabled, the Bandgap mode will be set as Bandgap Enabled with Buffer Disabled.

Note: This API shall be invoked following set HVDs/LVDs and regulators' driver strength.

Parameters

- base – SPC peripheral base address.
- mode – The Bandgap mode be selected.

Return values

- kStatus_SPC_BandgapModeWrong – The bandgap mode setting in Low Power mode is wrong.
- kStatus_Success – Config Bandgap mode in Low Power power mode successful.

static inline void SPC_EnableLowPowerModeCMPBandgapBufferMode(SPC_Type *base, bool enable)

Enables/Disables CMP Bandgap Buffer.

This function gates CMP bandgap buffer. CMP bandgap buffer is automatically disabled and turned off in Deep Power Down mode.

Parameters

- base – SPC peripheral base address.
- enable – Enable/Disable CMP Bandgap buffer. true - Enable Buffer Stored Reference Voltage to CMP. false - Disable Buffer Stored Reference Voltage to CMP.

static inline void SPC_EnableLowPowerModeCoreVDDInternalVoltageScaling(SPC_Type *base, bool enable)

Enables/Disables CORE VDD IVS(Internal Voltage Scaling) in low power modes.

This function gates CORE VDD IVS. When enabled, the IVS regulator will scale the external input CORE VDD to a lower voltage level to reduce internal leakage. IVS is invalid in Sleep or Deep power down mode.

Parameters

- base – SPC peripheral base address.
- enable – Enable/Disable IVS. true - enable CORE VDD IVS in Deep Sleep mode or Power Down mode. false - disable CORE VDD IVS in Deep Sleep mode or Power Down mode.

static inline void SPC_SetLowPowerWakeUpDelay(SPC_Type *base, uint16_t delay)

Sets the delay when exit the low power modes.

Parameters

- base – SPC peripheral base address.
- delay – The number of SPC timer clock cycles that the SPC waits on exit from low power modes.

status_t SPC_SetLowPowerModeRegulatorsConfig(SPC_Type *base, const *spc_lowpower_mode_regulators_config_t* *config)

Configs regulators in Low Power mode.

This function provides the method to config all on-chip regulators in Low Power mode.

Parameters

- base – SPC peripheral base address.
- config – Pointer to spc_lowpower_mode_regulators_config_t structure.

Return values

- kStatus_Success – Config regulators in Low power mode successful.
- kStatus_SPC_BandgapModeWrong – The bandgap mode setting in Low Power mode is wrong.
- kStatus_SPC_Busy – The SPC instance is busy to execute any type of power mode transition.
- kStatus_SPC_CORELDOVoltageWrong – The selected voltage level is wrong.
- kStatus_SPC_CORELDOLowDriveStrengthIgnore – Set driver strength to low will be ignored.
- kStatus_SPC_CORELDOVoltageSetFail – Fail to change Core LDO voltage level.
- kStatus_SPC_SYSLDOLowDriveStrengthIgnore – Set driver strength to low will be ignored.
- kStatus_SPC_DCDCPulseRefreshModeIgnore – Set driver strength to Pulse Refresh mode will be ignored.
- kStatus_SPC_DCDCLowDriveStrengthIgnore – Set driver strength to Low Drive Strength will be ignored.

static inline void SPC_DisableLowPowerModeVddCoreGlitchDetect(SPC_Type *base, bool disable)

Disable/Enable VDD Core Glitch Detect in low power mode.

Note: State of glitch detect disable feature will be ignored if bandgap is disabled and glitch detect hardware will be forced to OFF state.

Parameters

- base – SPC peripheral base address.
- disable – Used to disable/enable VDD Core Glitch detect feature.
  - **true** Disable VDD Core Low Voltage detect;
  - **false** Enable VDD Core Low Voltage detect.

static inline uint8_t SPC_GetVoltageDetectStatusFlag(SPC_Type *base)

Get Voltage Detect Status Flags.

Parameters

- base – SPC peripheral base address.

Returns

Voltage Detect Status Flags. See *_spc_voltage_detect_flags* for details.

static inline void SPC_ClearVoltageDetectStatusFlag(SPC_Type *base, uint8_t mask)

Clear Voltage Detect Status Flags.

Parameters

- base – SPC peripheral base address.
- mask – The mask of the voltage detect status flags. See *_spc_voltage_detect_flags* for details.

void SPC_SetCoreVoltageDetectConfig(SPC_Type *base, const *spc_core_voltage_detect_config_t* *config)

Configs CORE voltage detect options.

This function config CORE voltage detect options.

Note: : Setting both the voltage detect interrupt and reset enable will cause interrupt to be generated on exit from reset. If those conditioned is not desired, interrupt/reset so only one is enabled.

Parameters

- base – SPC peripheral base address.
- config – Pointer to spc_core_voltage_detect_config_t structure.

static inline void SPC_LockCoreVoltageDetectResetSetting(SPC_Type *base)

Locks Core voltage detect reset setting.

This function locks core voltage detect reset setting. After invoking this function any configuration of Core voltage detect reset will be ignored.

Parameters

- base – SPC peripheral base address.

static inline void SPC_UnlockCoreVoltageDetectResetSetting(SPC_Type *base)

Unlocks Core voltage detect reset setting.

This function unlocks core voltage detect reset setting. If locks the Core voltage detect reset setting, invoking this function to unlock.

Parameters

- base – SPC peripheral base address.

status_t SPC_EnableActiveModeCoreHighVoltageDetect(SPC_Type *base, bool enable)

Enables/Disables the Core High Voltage Detector in Active mode.

Note: If the CORE_LDO high voltage detect is enabled in Active mode, please note that the bandgap must be enabled and the drive strength of each regulator must not set to low.

Parameters

- base – SPC peripheral base address.
- enable – Enable/Disable Core HVD. true - Enable Core High voltage detector in active mode. false - Disable Core High voltage detector in active mode.

Return values

kStatus_Success – Enable/Disable Core High Voltage Detect successfully.

status_t SPC_EnableActiveModeCoreLowVoltageDetect(SPC_Type *base, bool enable)

Enables/Disables the Core Low Voltage Detector in Active mode.

Note: If the CORE_LDO low voltage detect is enabled in Active mode, please note that the bandgap must be enabled and the drive strength of each regulator must not set to low.

Parameters

- base – SPC peripheral base address.
- enable – Enable/Disable Core LVD. true - Enable Core Low voltage detector in active mode. false - Disable Core Low voltage detector in active mode.

Return values

kStatus_Success – Enable/Disable Core Low Voltage Detect successfully.

status_t SPC_EnableLowPowerModeCoreHighVoltageDetect(SPC_Type *base, bool enable)

Enables/Disables the Core High Voltage Detector in Low Power mode.

This function enables/disables the Core High Voltage Detector. If enabled the Core High Voltage detector. The Bandgap mode in low power mode must be programmed so that Bandgap is enabled.

Note: If the CORE_LDO high voltage detect is enabled in Low Power mode, please note that the bandgap must be enabled and the drive strength of each regulator must not set to low in low power mode.

Parameters

- base – SPC peripheral base address.
- enable – Enable/Disable Core HVD. true - Enable Core High voltage detector in low power mode. false - Disable Core High voltage detector in low power mode.

Return values

kStatus_Success – Enable/Disable Core High Voltage Detect in low power mode successfully.

status_t SPC_EnableLowPowerModeCoreLowVoltageDetect(SPC_Type *base, bool enable)

Enables/Disables the Core Low Voltage Detector in Low Power mode.

This function enables/disables the Core Low Voltage Detector. If enabled the Core Low Voltage detector. The Bandgap mode in low power mode must be programmed so that Bandgap is enabled.

Note: If the CORE_LDO low voltage detect is enabled in Low Power mode, please note that the bandgap must be enabled and the drive strength of each regulator must not set to low in Low Power mode.

Parameters

- base – SPC peripheral base address.
- enable – Enable/Disable Core HVD. true - Enable Core Low voltage detector in low power mode. false - Disable Core Low voltage detector in low power mode.

Return values

kStatus_Success – Enable/Disable Core Low Voltage Detect in low power mode successfully.

void SPC_SetVDD1P8VoltageDetectConfig(SPC_Type *base, const *spc_vdd1p8_voltage_detect_config_t* *config)

Configs VDD1P8 low voltage detect options.

This function configures the VDD1P8 low-voltage detect interrupt.

Parameters

- base – SPC peripheral base address.
- config – Pointer to spc_vdd1p8_voltage_detect_config_t structure.

status_t SPC_EnableActiveModeVDD1P8LowVoltageDetect(SPC_Type *base, bool enable)

Enables/Disables the VDD1P8 Low Voltage Detector in Active mode.

Note: If the VDD1P8 low voltage detect is enabled in Active mode, please note that the bandgap must be enabled and the drive strength of each regulator must not set to low in Active mode.

Parameters

- base – SPC peripheral base address.
- enable – Enable/Disable VDD1P8 LVD. true - Enable VDD1P8 low voltage detector in active mode. false - Disable VDD1P8 low voltage detector in active mode.

Return values

kStatus_Success – Enable/Disable VDD1P8 Low Voltage Detect successfully.

status_t SPC_EnableLowPowerModeVDD1P8LowVoltageDetect(SPC_Type *base, bool enable)

Enables/Disables the VDD1P8 Low Voltage Detector in Low Power mode.

Note: If the VDD1P8 low voltage detect is enabled in Low Power mode, please note that the bandgap must be enabled and the drive strength of each regulator must not set to low in Low Power mode.

Parameters

- base – SPC peripheral base address.
- enable – Enable/Disable VDD1P8 LVD. true - Enable VDD1P8 low voltage detector in low power mode. false - Disable VDD1P8 low voltage detector in low power mode.

Return values

kStatus_Success – Enable/Disable VDD1P8 Low Voltage Detect in low power mode successfully.

void SPC_SetSystemVDDLowVoltageLevel(SPC_Type *base, *spc_low_voltage_level_select_t* level)

Set system VDD Low-voltage level selection.

This function selects the system VDD low-voltage level. Changing system VDD low-voltage level must be done after disabling the System VDD low voltage reset and interrupt.

Deprecated:

In latest RM, reserved for all devices, will removed in next release.

Parameters

- base – SPC peripheral base address.
- level – System VDD Low-Voltage level selection.

void SPC_SetSystemVoltageDetectConfig(SPC_Type *base, const *spc_system_voltage_detect_config_t* *config)

Configs SYS voltage detect options.

This function config SYS voltage detect options.

Note: : Setting both the voltage detect interrupt and reset enable will cause interrupt to be generated on exit from reset. If those conditioned is not desired, interrupt/reset so only one is enabled.

Parameters

- base – SPC peripheral base address.
- config – Pointer to spc_system_voltage_detect_config_t structure.

static inline void SPC_LockSystemVoltageDetectResetSetting(SPC_Type *base)

Lock System voltage detect reset setting.

This function locks system voltage detect reset setting. After invoking this function any configuration of System Voltage detect reset will be ignored.

Parameters

- base – SPC peripheral base address.

static inline void SPC_UnlockSystemVoltageDetectResetSetting(SPC_Type *base)

Unlock System voltage detect reset setting.

This function unlocks system voltage detect reset setting. If locks the System voltage detect reset setting, invoking this function to unlock.

Parameters

- base – SPC peripheral base address.

status_t SPC_EnableActiveModeSystemHighVoltageDetect(SPC_Type *base, bool enable)

Enables/Disables the System High Voltage Detector in Active mode.

Note: If the System_LDO high voltage detect is enabled in Active mode, please note that the bandgap must be enabled and the drive strength of each regulator must not set to low in Active mode.

Parameters

- base – SPC peripheral base address.
- enable – Enable/Disable System HVD. true - Enable System High voltage detector in active mode. false - Disable System High voltage detector in active mode.

Return values

kStatus_Success – Enable/Disable System High Voltage Detect successfully.

status_t SPC_EnableActiveModeSystemLowVoltageDetect(SPC_Type *base, bool enable)

Enables/Disable the System Low Voltage Detector in Active mode.

Note: If the System_LDO low voltage detect is enabled in Active mode, please note that the bandgap must be enabled and the drive strength of each regulator must not set to low in Active mode.

Parameters

- base – SPC peripheral base address.
- enable – Enable/Disable System LVD. true - Enable System Low voltage detector in active mode. false - Disable System Low voltage detector in active mode.

Return values

kStatus_Success – Enable/Disable the System Low Voltage Detect successfully.

status_t SPC_EnableLowPowerModeSystemHighVoltageDetect(SPC_Type *base, bool enable)

Enables/Disables the System High Voltage Detector in Low Power mode.

Note: If the System_LDO high voltage detect is enabled in Low Power mode, please note that the bandgap must be enabled and the drive strength of each regulator must not set to low in Low Power mode.

Parameters

- base – SPC peripheral base address.
- enable – Enable/Disable System HVD. true - Enable System High voltage detector in low power mode. false - Disable System High voltage detector in low power mode.

Return values

kStatus_Success – Enable/Disable System High Voltage Detect in low power mode successfully.

status_t SPC_EnableLowPowerModeSystemLowVoltageDetect(SPC_Type *base, bool enable)

Enables/Disables the System Low Voltage Detector in Low Power mode.

Note: If the System_LDO low voltage detect is enabled in Low Power mode, please note that the bandgap must be enabled and the drive strength of each regulator must not set to low in Low Power mode.

Parameters

- base – SPC peripheral base address.
- enable – Enable/Disable System HVD. true - Enable System Low voltage detector in low power mode. false - Disable System Low voltage detector in low power mode.

Return values

kStatus_Success – Enables System Low Voltage Detect in low power mode successfully.

void SPC_SetIOVDDLowVoltageLevel(SPC_Type *base, *spc_low_voltage_level_select_t* level)

Set IO VDD Low-Voltage level selection.

This function selects the IO VDD Low-voltage level. Changing IO VDD low-voltage level must be done after disabling the IO VDD low voltage reset and interrupt.

Parameters

- base – SPC peripheral base address.
- level – IO VDD Low-voltage level selection.

void SPC_SetIOVoltageDetectConfig(SPC_Type *base, const *spc_io_voltage_detect_config_t* *config)

Configs IO voltage detect options.

This function config IO voltage detect options.

Note: : Setting both the voltage detect interrupt and reset enable will cause interrupt to be generated on exit from reset. If those conditioned is not desired, interrupt/reset so only one is enabled.

Parameters

- base – SPC peripheral base address.
- config – Pointer to spc_voltage_detect_config_t structure.

static inline void SPC_LockIOVoltageDetectResetSetting(SPC_Type *base)

Lock IO Voltage detect reset setting.

This function locks IO voltage detect reset setting. After invoking this function any configuration of system voltage detect reset will be ignored.

Parameters

- base – SPC peripheral base address.

static inline void SPC_UnlockIOVoltageDetectResetSetting(SPC_Type *base)

Unlock IO voltage detect reset setting.

This function unlocks IO voltage detect reset setting. If locks the IO voltage detect reset setting, invoking this function to unlock.

Parameters

- base – SPC peripheral base address.

status_t SPC_EnableActiveModeIOHighVoltageDetect(SPC_Type *base, bool enable)

Enables/Disables the IO High Voltage Detector in Active mode.

Note: If the IO high voltage detect is enabled in Active mode, please note that the bandgap must be enabled and the drive strength of each regulator must not set to low in Active mode.

Parameters

- base – SPC peripheral base address.
- enable – Enable/Disable IO HVD. true - Enable IO High voltage detector in active mode. false - Disable IO High voltage detector in active mode.

Return values

kStatus_Success – Enable/Disable IO High Voltage Detect successfully.

status_t SPC_EnableActiveModeIOLowVoltageDetect(SPC_Type *base, bool enable)

Enables/Disables the IO Low Voltage Detector in Active mode.

Note: If the IO low voltage detect is enabled in Active mode, please note that the bandgap must be enabled and the drive strength of each regulator must not set to low in Active mode.

Parameters

- base – SPC peripheral base address.
- enable – Enable/Disable IO LVD. true - Enable IO Low voltage detector in active mode. false - Disable IO Low voltage detector in active mode.

Return values

kStatus_Success – Enable IO Low Voltage Detect successfully.

status_t SPC_EnableLowPowerModeIOHighVoltageDetect(SPC_Type *base, bool enable)

Enables/Disables the IO High Voltage Detector in Low Power mode.

Note: If the IO high voltage detect is enabled in Low Power mode, please note that the bandgap must be enabled and the drive strength of each regulator must not set to low in Low Power mode.

Parameters

- base – SPC peripheral base address.
- enable – Enable/Disable IO HVD. true - Enable IO High voltage detector in low power mode. false - Disable IO High voltage detector in low power mode.

Return values

kStatus_Success – Enable IO High Voltage Detect in low power mode successfully.

status_t SPC_EnableLowPowerModeIOLowVoltageDetect(SPC_Type *base, bool enable)

Enables/Disables the IO Low Voltage Detector in Low Power mode.

Note: If the IO low voltage detect is enabled in Low Power mode, please note that the bandgap must be enabled and the drive strength of each regulator must not set to low in Low Power mode.

Parameters

- base – SPC peripheral base address.
- enable – Enable/Disable IO LVD. true - Enable IO Low voltage detector in low power mode. false - Disable IO Low voltage detector in low power mode.

Return values

kStatus_Success – Enable/Disable IO Low Voltage Detect in low power mode successfully.

void SPC_SetExternalVoltageDomainsConfig(SPC_Type *base, uint8_t lowPowerIsoMask, uint8_t IsoMask)

Configs external voltage domains.

This function configs external voltage domains isolation.

Parameters

- base – SPC peripheral base address.
- lowPowerIsoMask – The mask of external domains isolate enable during low power mode. Please read the Reference Manual for the Bitmap.
- IsoMask – The mask of external domains isolate. Please read the Reference Manual for the Bitmap.

static inline uint8_t SPC_GetExternalDomainsStatus(SPC_Type *base)

Gets External Domains status.

This function configs external voltage domains status.

Parameters

- base – SPC peripheral base address.

Returns

The status of each external domain.

static inline void SPC_EnableCoreLDORegulator(SPC_Type *base, bool enable)

Enable/Disable Core LDO regulator.

Note: The CORE LDO enable bit is write-once.

Parameters

- base – SPC peripheral base address.
- enable – Enable/Disable CORE LDO Regulator. true - Enable CORE LDO Regulator. false - Disable CORE LDO Regulator.

static inline void SPC_PullDownCoreLDORegulator(SPC_Type *base, bool pulldown)

Enable/Disable the CORE LDO Regulator pull down in Deep Power Down.

Note: This function only useful when enabled the CORE LDO Regulator.

Parameters

- base – SPC peripheral base address.
- pulldown – Enable/Disable CORE LDO pulldown in Deep Power Down mode. true - CORE LDO Regulator will discharge in Deep Power Down mode. false - CORE LDO Regulator will not discharge in Deep Power Down mode.

status_t SPC_SetActiveModeCoreLDORegulatorConfig(SPC_Type *base, const *spc_active_mode_core_ldo_option_t* *option)

Configs Core LDO VDD Regulator in Active mode.

Note: If any voltage detect feature is enabled in Active mode, then CORE_LDO's drive strength must not set to low.

Note: Core VDD level for the Core LDO low power regulator can only be changed when CORELDO_VDD_DS is normal

Parameters

- base – SPC peripheral base address.
- option – Pointer to the spc_active_mode_core_ldo_option_t structure.

Return values

- kStatus_Success – Config Core LDO regulator in Active power mode successful.
- kStatus_SPC_Busy – The SPC instance is busy to execute any type of power mode transition.
- kStatus_SPC_CORELDOLowDriveStrengthIgnore – If any voltage detect enabled, core_ldo's drive strength can not set to low.
- kStatus_SPC_CORELDOVoltageWrong – The selected voltage level in active mode is not allowed.
- kStatus_Timeout – Timeout occurs while waiting completion.

status_t SPC_SetLowPowerModeCoreLDORegulatorConfig(SPC_Type *base, const *spc_lowpower_mode_core_ldo_option_t* *option)

Configs CORE LDO Regulator in low power mode.

This function configs CORE LDO Regulator in Low Power mode. If CORE LDO VDD Drive Strength is set to Normal, the CORE LDO VDD regulator voltage level in Active mode must be equal to the voltage level in Low power mode. And the Bandgap must be programmed to select bandgap enabled. Core VDD voltage levels for the Core LDO low power regulator can only be changed when the CORE LDO Drive Strength set as Normal.

Parameters

- base – SPC peripheral base address.
- option – Pointer to the spc_lowpower_mode_core_ldo_option_t structure.

Return values

- kStatus_Success – Config Core LDO regulator in power mode successfully.
- kStatus_SPC_Busy – The SPC instance is busy to execute any type of power mode transition.
- kStatus_SPC_CORELDOLowDriveStrengthIgnore – Set driver strength to low will be ignored.
- kStatus_SPC_CORELDOVoltageSetFail – Fail to change Core LDO voltage level.
- kStatus_Timeout – Timeout occurs while waiting completion.

static inline void SPC_EnableSystemLDORegulator(SPC_Type *base, bool enable)

Enable/Disable System LDO regulator.

Note: The SYSTEM LDO enable bit is write-once.

Parameters

- base – SPC peripheral base address.
- enable – Enable/Disable System LDO Regulator. true - Enable System LDO Regulator. false - Disable System LDO Regulator.

static inline void SPC_EnableSystemLDOSinkFeature(SPC_Type *base, bool sink)

Enable/Disable current sink feature of System LDO Regulator.

Parameters

- base – SPC peripheral base address.
- sink – Enable/Disable current sink feature. true - Enable current sink feature of System LDO Regulator. false - Disable current sink feature of System LDO Regulator.

status_t SPC_SetActiveModeSystemLDORegulatorConfig(SPC_Type *base, const *spc_active_mode_sys_ldo_option_t* *option)

Configs System LDO VDD Regulator in Active mode.

This function configs System LDO VDD Regulator in Active mode. If System LDO VDD Drive Strength is set to Normal, the Bandgap mode in Active mode must be programmed to a value that enables the bandgap. If any voltage detects are kept enabled, configuration to set System LDO VDD drive strength to low will be ignored. If select System LDO VDD Regulator voltage level to Over Drive Voltage, the Drive Strength of System LDO VDD Regulator must be set to Normal otherwise the regulator Drive Strength will be forced to Normal. If select System LDO VDD Regulator voltage level to Over Drive Voltage, the High voltage detect must be disabled. Otherwise it will be fail to regulator to Over Drive Voltage.

Parameters

- base – SPC peripheral base address.
- option – Pointer to the spc_active_mode_sys_ldo_option_t structure.

Return values

- kStatus_Success – Config System LDO regulator in Active power mode successful.
- kStatus_SPC_Busy – The SPC instance is busy to execute any type of power mode transition.
- kStatus_SPC_SYSLDOOverDriveVoltageFail – Fail to regulator to Over Drive Voltage.
- kStatus_SPC_SYSLDOLowDriveStrengthIgnore – Set driver strength to Low will be ignored.
- kStatus_Timeout – Timeout occurs while waiting completion.

status_t SPC_SetLowPowerModeSystemLDORegulatorConfig(SPC_Type *base, const *spc_lowpower_mode_sys_ldo_option_t* *option)

Configs System LDO regulator in low power modes.

This function configs System LDO regulator in low power modes. If System LDO VDD Regulator Drive strength is set to normal, bandgap mode in low power mode must be programmed to a value that enables the Bandgap. If any High voltage detectors or Low Voltage detectors are kept enabled, configuration to set System LDO Regulator drive strength as Low will be ignored.

Parameters

- base – SPC peripheral base address.

- option – Pointer to spc_lowpower_mode_sys_ldo_option_t structure.

Return values

- kStatus_Success – Config System LDO regulator in Low Power Mode successfully.
- kStatus_SPC_Busy – The SPC instance is busy to execute any type of power mode transition.
- kStatus_SPC_SYSLDOLowDriveStrengthIgnore – Set driver strength to low will be ignored.
- kStatus_Timeout – Timeout occurs while waiting completion.

static inline void SPC_EnableDCDCRegulator(SPC_Type *base, bool enable)

Enable/Disable DCDC Regulator.

Note: The DCDC enable bit is write-once.

Parameters

- base – SPC peripheral base address.
- enable – Enable/Disable DCDC Regulator. true - Enable DCDC Regulator. false - Disable DCDC Regulator.

status_t SPC_SetDCDCBurstConfig(SPC_Type *base, *spc_dcdc_burst_config_t* *config)

Config DCDC Burst options.

Parameters

- base – SPC peripheral base address.
- config – Pointer to spc_dcdc_burst_config_t structure.

Return values

- kStatus_Success – Successfully configured.
- kStatus_Timeout – Timeout occurs while waiting completion.

void SPC_SetDCDCRefreshCount(SPC_Type *base, uint16_t count)

Set the count value of the reference clock.

This function set the count value of the reference clock to control the frequency of dcdc refresh when dcdc is configured in Pulse Refresh mode.

Parameters

- base – SPC peripheral base address.
- count – The count value, 16 bit width.

status_t SPC_SetActiveModeDCDCRegulatorConfig(SPC_Type *base, const *spc_active_mode_dcdc_option_t* *option)

Configs DCDC VDD Regulator in Active mode.

This function configs DCDC VDD Regulator in Active mode. If DCDDC VDD Drive Strength is set to Normal, the Bandgap mode in Active mode must be programmed to a value that enable the bandgap. If any voltage detects are kept enabled, configuration to set DCDC VDD drive strength to low will be ignored. When switching DCDC from low drive strength to Normal driver strength, make sure the DCDC high VDD LVL setting to the same level that was set prior to switching the DCDC to low drive strength.

Parameters

- base – SPC peripheral base address.
- option – Pointer to the spc_active_mode_dcdc_option_t structure.

Return values

- kStatus_Success – Config DCDC regulator in Active power mode successful.
- kStatus_SPC_Busy – The SPC instance is busy to execute any type of power mode transition.
- kStatus_SPC_DCDCLowDriveStrengthIgnore – Set driver strength to Low will be ignored.
- kStatus_Timeout – Timeout occurs while waiting completion.

status_t SPC_SetLowPowerModeDCDCRegulatorConfig(SPC_Type *base, const *spc_lowpower_mode_dcdc_option_t* *option)

Configs DCDC VDD Regulator in Low power modes.

This function configs DCDC VDD Regulator in Low Power modes. If DCDC VDD Drive Strength is set to Normal, the Bandgap mode in Low Power mode must be programmed to a value that enables the Bandgap. If any of voltage detectors are kept enabled, configuration to set DCDC VDD Drive Strength to Low or Pulse mode will be ignored. In Deep Power Down mode, DCDC regulator is always turned off.

Parameters

- base – SPC peripheral base address.
- option – Pointer to the spc_lowpower_mode_dcdc_option_t structure.

Return values

- kStatus_Success – Config DCDC regulator in low power mode successfully.
- kStatus_SPC_Busy – The SPC instance is busy to execute any type of power mode transition.
- kStatus_SPC_DCDCPulseRefreshModeIgnore – Set driver strength to Pulse Refresh mode will be ignored.
- kStatus_SPC_DCDCLowDriveStrengthIgnore – Set driver strength to Low Drive Strength will be ignored.
- kStatus_Timeout – Timeout occurs while waiting completion.

status_t SPC_SetSRAMOperateVoltage(SPC_Type *base, *spc_sram_operat_voltage_t* voltage)

Set the SRAM operate voltage level.

Parameters

- base – SPC peripheral base address.
- voltage – Target SRAM operate voltage level, please refer to *spc_sram_operat_voltage_t*.

Return values

- kStatus_Success – Successfully configured.
- kStatus_Timeout – Timeout occurs while waiting completion.

void SPC_ConfigVddCoreGlitchDetector(SPC_Type *base, const *spc_vdd_core_glitch_detector_config_t* *config)

Configures VDD Core Glitch detector, including ripple counter selection, timeout value and so on.

Parameters

- base – SPC peripheral base address.
- config – Pointer to the structure in type of *spc_vdd_core_glitch_detector_config_t*.

static inline bool SPC_CheckGlitchRippleCounterOutput(SPC_Type *base, *spc_vdd_core_glitch_ripple_counter_select_t* rippleCounter)

Checks selected 4-bit glitch ripple counter's output.

Parameters

- base – SPC peripheral base address.
- rippleCounter – The ripple counter to check, please refer to *spc_vdd_core_glitch_ripple_counter_select_t*.

Return values

- true – The selected ripple counter output is 1, will generate interrupt or reset based on settings.
- false – The selected ripple counter output is 0.

static inline void SPC_ClearGlitchRippleCounterOutput(SPC_Type *base, *spc_vdd_core_glitch_ripple_counter_select_t* rippleCounter)

Clears output of selected glitch ripple counter.

Parameters

- base – SPC peripheral base address.
- rippleCounter – The ripple counter to check, please refer to *spc_vdd_core_glitch_ripple_counter_select_t*.

static inline void SPC_LockVddCoreVoltageGlitchDetectResetControl(SPC_Type *base)

After invoking this function, writes to SPC_VDD_CORE_GLITCH_DETECT_SC[RE] register are ignored.

Parameters

- base – SPC peripheral base address.

static inline void SPC_UnlockVddCoreVoltageGlitchDetectResetControl(SPC_Type *base)

After invoking this function, writes to SPC_VDD_CORE_GLITCH_DETECT_SC[RE] register are allowed.

Parameters

- base – SPC peripheral base address.

static inline bool SPC_CheckVddCoreVoltageGlitchResetControlState(SPC_Type *base)

Checks if SPC_VDD_CORE_GLITCH_DETECT_SC[RE] register is writable.

Parameters

- base – SPC peripheral base address.

Return values

- true – SPC_VDD_CORE_GLITCH_DETECT_SC[RE] register is writable.
- false – SPC_VDD_CORE_GLITCH_DETECT_SC[RE] register is not writable.

FSL_SPC_DRIVER_VERSION

SPC driver version 2.9.0.

SPC_EVD_CFG_REG_EVDISO_SHIFT

SPC_EVD_CFG_REG_EVDLPISO_SHIFT

SPC_EVD_CFG_REG_EVDSTAT_SHIFT

SPC_EVD_CFG_REG_EVDISO(x)

SPC_EVD_CFG_REG_EVDLPISO(x)

SPC_EVD_CFG_REG_EVDSTAT(x)

struct _spc_lowpower_request_config

#include <fsl_spc.h> Low Power Request output pin configuration.

Public Members

bool enable

Low Power Request Output enable.

spc_lowpower_request_pin_polarity_t polarity

Low Power Request Output pin polarity select.

spc_lowpower_request_output_override_t override

Low Power Request Output Override.

struct _spc_intergrated_power_switch_config

#include <fsl_spc.h> Integrated power switch configuration.

Note: Legacy structure, will be removed.

Public Members

bool wakeup

Assert an output pin to un-gate the integrated power switch.

bool sleep

Assert an output pin to power gate the intergrated power switch.

struct _spc_active_mode_core_ldo_option

#include <fsl_spc.h> Core LDO regulator options in Active mode.

Public Members

spc_core_ldo_voltage_level_t CoreLDOVoltage

Core LDO Regulator Voltage Level selection in Active mode.

spc_core_ldo_drive_strength_t CoreLDODriveStrength
Core LDO Regulator Drive Strength selection in Active mode

struct __spc__active__mode__sys__ldo__option
#include <fsl_spc.h> System LDO regulator options in Active mode.

Public Members

spc_sys_ldo_voltage_level_t SysLDOVoltage
System LDO Regulator Voltage Level selection in Active mode.

spc_sys_ldo_drive_strength_t SysLDODriveStrength
System LDO Regulator Drive Strength selection in Active mode.

struct __spc__active__mode__dcdc__option
#include <fsl_spc.h> DCDC regulator options in Active mode.

Public Members

spc_dcdc_voltage_level_t DCDCVoltage
DCDC Regulator Voltage Level selection in Active mode.

spc_dcdc_drive_strength_t DCDCDriveStrength
DCDC VDD Regulator Drive Strength selection in Active mode.

struct __spc__lowpower__mode__core__ldo__option
#include <fsl_spc.h> Core LDO regulator options in Low Power mode.

Public Members

spc_core_ldo_voltage_level_t CoreLDOVoltage
Core LDO Regulator Voltage Level selection in Low Power mode.

spc_core_ldo_drive_strength_t CoreLDODriveStrength
Core LDO Regulator Drive Strength selection in Low Power mode

struct __spc__lowpower__mode__sys__ldo__option
#include <fsl_spc.h> System LDO regulator options in Low Power mode.

Public Members

spc_sys_ldo_drive_strength_t SysLDODriveStrength
System LDO Regulator Drive Strength selection in Low Power mode.

struct __spc__lowpower__mode__dcdc__option
#include <fsl_spc.h> DCDC regulator options in Low Power mode.

Public Members

spc_dcdc_voltage_level_t DCDCVoltage
DCDC Regulator Voltage Level selection in Low Power mode.

spc_dcdc_drive_strength_t DCDCDriveStrength
DCDC VDD Regulator Drive Strength selection in Low Power mode.

struct __spc__voltage__detect__option
#include <fsl_spc.h> CORE/SYS/IO VDD Voltage Detect options.

Public Members

bool HVDInterruptEnable
CORE/SYS/IO VDD High Voltage Detect interrupt enable.

bool HVDResetEnable
CORE/SYS/IO VDD High Voltage Detect reset enable.

bool LVDInterruptEnable
CORE/SYS/IO VDD Low Voltage Detect interrupt enable.

bool LVDResetEnable
CORE/SYS/IO VDD Low Voltage Detect reset enable.

struct __spc__dcdc__burst__config
#include <fsl_spc.h> DCDC Burst configuration.

Public Members

bool sofwareBurstRequest
Enable/Disable DCDC Software Burst Request.

bool externalBurstRequest
Enable/Disable DCDC External Burst Request.

bool stabilizeBurstFreq
Enable/Disable DCDC frequency stabilization.

uint8_t freq
The frequency of the current burst.

struct __spc__core__voltage__detect__config
#include <fsl_spc.h> Core Voltage Detect configuration.

Public Members

spc_voltage_detect_option_t option
Core VDD Voltage Detect option.

struct __spc__vdd1p8__voltage__detect__config
#include <fsl_spc.h> VDD1P8 Voltage Detect configuration.

Public Members

bool LVDInterruptEnable
VDD1P8 Low Voltage Detect interrupt enable.

struct __spc__system__voltage__detect__config
#include <fsl_spc.h> System Voltage Detect Configuration.

Public Members

spc_voltage_detect_option_t option
System VDD Voltage Detect option.

spc_low_voltage_level_select_t level

Deprecated:
, reserved for all devices, will removed in next release.

struct __spc__io__voltage__detect__config
#include <fsl_spc.h> IO Voltage Detect Configuration.

Public Members

spc_voltage_detect_option_t option
IO VDD Voltage Detect option.

spc_low_voltage_level_select_t level
IO VDD Low-voltage level selection.

struct __spc__active__mode__regulators__config
#include <fsl_spc.h> Active mode configuration.

struct __spc__lowpower__mode__regulators__config
#include <fsl_spc.h> Low Power Mode configuration.

struct __spc__vdd__core__glitch__detector__config
#include <fsl_spc.h> The configuration of VDD Core glitch detector.

Public Members

spc_vdd_core_glitch_ripple_counter_select_t rippleCounterSelect

Used to set ripple counter.

uint8_t resetTimeoutValue

The timeout value used to reset glitch detect/compare logic after an initial glitch is detected.

bool enableReset

Used to enable/disable POR/LVD reset that caused by CORE VDD glitch detect error.

bool enableInterrupt

Used to enable/disable hardware interrupt if CORE VDD glitch detect error.

struct _spc_dcdc_ramp_config

#include <fsl_spc.h> DCDC output voltage ramp control configuration.

Used by *SPC_SetDCDCRampControlConfig*. The ramp trim fields select the DCDC output voltage ramp slope; valid values are 0..7.

Public Members

bool enableRampControl

Enable/disable DCDC output voltage ramp control.

uint8_t rampControl

Active/High Power mode ramp control trim, range 0..7.

uint8_t rampControlLowPower

Low Power mode ramp control trim, range 0..7.

2.65 SYSPM: System Performance Monitor

enum _syspm_monitor

syspm select control monitor

Values:

enumerator kSYSPM_Monitor0

Monitor 0

enum _syspm_event

syspm select event

Values:

enumerator kSYSPM_Event1
Event 1

enumerator kSYSPM_Event2
Event 2

enumerator kSYSPM_Event3
Event 3

enum _syspm_mode
syspm set count mode
Values:

enumerator kSYSPM_BothMode
count in both modes

enumerator kSYSPM_UserMode
count only in user mode

enumerator kSYSPM_PrivilegedMode
count only in privileged mode

enum _syspm_startstop_control
syspm start/stop control
Values:

enumerator kSYSPM_Idle
idle >

enumerator kSYSPM_LocalStop
local stop

enumerator kSYSPM_LocalStart
local start

enumerator KSYSPM_EnableTraceControl
enable global TSTART/TSTOP

enumerator kSYSPM_GlobalStart
global stop

enumerator kSYSPM_GlobalStop
global start

typedef enum *_syspm_monitor* syspm_monitor_t
syspm select control monitor

typedef enum *_syspm_event* syspm_event_t

sysp m select event

typedef enum *_syspm_mode* syspm_mode_t

sysp m set count mode

typedef enum *_syspm_startstop_control* syspm_startstop_control_t

sysp m start/stop control

void SYSPM_Init(SYSPM_Type *base)

Initializes the SYSPM.

This function enables the SYSPM clock.

Parameters

- base – SYSPM peripheral base address.

void SYSPM_Deinit(SYSPM_Type *base)

Deinitializes the SYSPM.

This function disables the SYSPM clock.

Parameters

- base – SYSPM peripheral base address.

void SYSPM_SelectEvent(SYSPM_Type *base, *syspm_monitor_t* monitor, *syspm_event_t* event, uint8_t eventCode)

Select event counters.

Parameters

- base – SYSPM peripheral base address.
- monitor – syspm control monitor, see to *syspm_monitor_t*.
- event – syspm select event, see to *syspm_event_t*.
- eventCode – select which event to be counted in PMECTRx., see to table Events.

void SYSPM_ResetEvent(SYSPM_Type *base, *syspm_monitor_t* monitor, *syspm_event_t* event)

Reset event counters.

Parameters

- base – SYSPM peripheral base address.
- monitor – syspm control monitor, see to *syspm_monitor_t*.
- event – syspm select event, see to *syspm_event_t*.

void SYSPM_ResetInstructionEvent(SYSPM_Type *base, *syspm_monitor_t* monitor)

Reset Instruction Counter.

Parameters

- base – SYSPM peripheral base address.
- monitor – syspm control monitor, see to *syspm_monitor_t*.

void SYSPM_SetCountMode(SYSPM_Type *base, *syspm_monitor_t* monitor, *syspm_mode_t* mode)

Set count mode.

Parameters

- base – SYSPM peripheral base address.
- monitor – syspm control monitor, see to *syspm_monitor_t*.
- mode – syspm select counter mode, see to *syspm_mode_t*.

void SYSPM_SetStartStopControl(SYSPM_Type *base, *syspm_monitor_t* monitor, *syspm_startstop_control_t* ssc)

Set Start/Stop Control.

Parameters

- base – SYSPM peripheral base address.
- monitor – syspm control monitor, see to *syspm_monitor_t*.
- ssc – This 3-bit field provides a three-phase mechanism to start/stop the counters. It includes a prioritized scheme with local start > local stop > global start > global stop > conditional TSTART > TSTOP. The global and conditional start/stop affect all configured PM/PSAM module concurrently so counters are "coherent". see to *syspm_startstop_control_t*

void SYSPM_DisableCounter(SYSPM_Type *base, *syspm_monitor_t* monitor)

Disable Counters if Stopped or Halted.

Parameters

- base – SYSPM peripheral base address.
- monitor – syspm control monitor, see to *syspm_monitor_t*.

uint64_t SYSPM_GetEventCounter(SYSPM_Type *base, *syspm_monitor_t* monitor, *syspm_event_t* event)

This is the the 40-bits of eventx counter. The value in this register increments each time the event selected in PMCRx[SELEVTx] occurs.

Parameters

- base – SYSPM peripheral base address.
- monitor – syspm control monitor, see to *syspm_monitor_t*.
- event – syspm select event, see to *syspm_event_t*.

Returns

- When the return value is not equal to *SYSPM_COUNT_STABLE_TIMEOUT_RETURN_VALUE*, the return value represents a 40 bits eventx counter.
- When the return value is equal to *SYSPM_COUNT_STABLE_TIMEOUT_RETURN_VALUE*, the return value represents timeout occured.

uint64_t SYSPM_GetInstructionCounter(SYSPM_Type *base, *syspm_monitor_t* monitor)

This is the the 40-bits of instructionx counter. The value in this register increments each time the CPU count signals occurs.

Parameters

- base – SYSPM peripheral base address.
- monitor – syspm control monitor, see to *syspm_monitor_t*.

Returns

- When the return value is not equal to *SYSPM_COUNT_STABLE_TIMEOUT_RETURN_VALUE*, the return value represents a 40 bits instruction counter.

- When the return value is equal to *SYSPM_COUNT_STABLE_TIMEOUT_RETURN_VALUE*, the return value represents timeout occured.

EVENT_COUNT_STABLE_TIMEOUT

Max loops to wait for SYSPM event count stable (0 means wait forever)

INSTRUCTION_COUNT_STABLE_TIMEOUT

Max loops to wait for SYSPM instruction count stable (0 means wait forever)

FSL_SYSPM_DRIVER_VERSION

SYSPM driver version.

SYSPM_COUNT_STABLE_TIMEOUT_RETURN_VALUE

2.66 TPM: Timer PWM Module

uint32_t TPM_GetInstance(TPM_Type *base)

Gets the instance from the base address.

Parameters

- base – TPM peripheral base address

Returns

The TPM instance

void TPM_Init(TPM_Type *base, const *tpm_config_t* *config)

Ungates the TPM clock and configures the peripheral for basic operation.

Note: This API should be called at the beginning of the application using the TPM driver.

Parameters

- base – TPM peripheral base address
- config – Pointer to user's TPM config structure.

void TPM_Deinit(TPM_Type *base)

Stops the counter and gates the TPM clock.

Parameters

- base – TPM peripheral base address

void TPM_GetDefaultConfig(*tpm_config_t* *config)

Fill in the TPM config struct with the default settings.

The default values are:

```
config->prescale = kTPM_Prescale_Divide_1;
config->useGlobalTimeBase = false;
config->syncGlobalTimeBase = false;
config->dozeEnable = false;
config->dbgMode = false;
config->enableReloadOnTrigger = false;
config->enableStopOnOverflow = false;
```

(continues on next page)

(continued from previous page)

```
    config->enableStartOnTrigger = false;
#if FSL_FEATURE_TPM_HAS_PAUSE_COUNTER_ON_TRIGGER
    config->enablePauseOnTrigger = false;
#endif
    config->triggerSelect = kTPM_Trigger_Select_0;
#if FSL_FEATURE_TPM_HAS_EXTERNAL_TRIGGER_SELECTION
    config->triggerSource = kTPM_TriggerSource_External;
    config->extTriggerPolarity = kTPM_ExtTrigger_Active_High;
#endif
#if defined(FSL_FEATURE_TPM_HAS_POL) && FSL_FEATURE_TPM_HAS_POL
    config->chnlPolarity = 0U;
#endif
```

Parameters

- config – Pointer to user's TPM config structure.

tpm_clock_prescale_t TPM_CalculateCounterClkDiv(TPM_Type *base, uint32_t counterPeriod_Hz, uint32_t srcClock_Hz)

Calculates the counter clock prescaler.

This function calculates the values for SC[PS].

return Calculated clock prescaler value.

Parameters

- base – TPM peripheral base address
- counterPeriod_Hz – The desired frequency in Hz which corresponding to the time when the counter reaches the mod value
- srcClock_Hz – TPM counter clock in Hz

static inline void TPM_Reset(TPM_Type *base)

Performs a software reset on the TPM module.

Reset all internal logic and registers, except the Global Register. Remains set until cleared by software.

Note: TPM software reset is available on certain SoC's only

Parameters

- base – TPM peripheral base address

status_t TPM_SetupPwm(TPM_Type *base, const *tpm_chnl_pwm_signal_param_t* *chnlParams, uint8_t numOfChnls, *tpm_pwm_mode_t* mode, uint32_t pwmFreq_Hz, uint32_t srcClock_Hz)

Configures the PWM signal parameters.

User calls this function to configure the PWM signals period, mode, dutycycle and edge. Use this function to configure all the TPM channels that will be used to output a PWM signal

Parameters

- base – TPM peripheral base address
- chnlParams – Array of PWM channel parameters to configure the channel(s)

- numOfChnls – Number of channels to configure, this should be the size of the array passed in
- mode – PWM operation mode, options available in enumeration *tpm_pwm_mode_t*
- pwmFreq_Hz – PWM signal frequency in Hz
- srcClock_Hz – TPM counter clock in Hz

Returns

kStatus_Success PWM setup successful kStatus_Error PWM setup failed kStatus_Timeout PWM setup timeout when write register CnV or MOD

status_t TPM_UpdatePwmDutycycle(TPM_Type *base, *tpm_chnl_t* chnlNumber, *tpm_pwm_mode_t* currentPwmMode, uint8_t dutyCyclePercent)

Update the duty cycle of an active PWM signal.

Parameters

- base – TPM peripheral base address
- chnlNumber – The channel number. In combined mode, this represents the channel pair number
- currentPwmMode – The current PWM mode set during PWM setup
- dutyCyclePercent – New PWM pulse width, value should be between 0 to 100 0=inactive signal(0% duty cycle)... 100=active signal (100% duty cycle)

Returns

kStatus_Success if the PWM setup was successful, kStatus_Error on failure

void TPM_UpdateChnlEdgeLevelSelect(TPM_Type *base, *tpm_chnl_t* chnlNumber, uint8_t level)

Update the edge level selection for a channel.

Note: When the TPM has PWM pause level select feature (FSL_FEATURE_TPM_HAS_PAUSE_LEVEL_SELECT = 1), the PWM output cannot be turned off by selecting the output level. In this case, must use TPM_DisableChannel API to close the PWM output.

Parameters

- base – TPM peripheral base address
- chnlNumber – The channel number
- level – The level to be set to the ELSnB:ELSnA field; valid values are 00, 01, 10, 11. See the appropriate SoC reference manual for details about this field.

static inline uint8_t TPM_GetChannelContorlBits(TPM_Type *base, *tpm_chnl_t* chnlNumber)

Get the channel control bits value (mode, edge and level bit fields).

Deprecated:

Please use *TPM_GetChannelControlBits()* instead.

Parameters

- base – TPM peripheral base address
- chnlNumber – The channel number

Returns
The contorl bits value. This is the logical OR of members of the enumeration *tpm_chnl_control_bit_mask_t*.

static inline uint8_t TPM_GetChannelControlBits(TPM_Type *base, *tpm_chnl_t* chnlNumber)

Get the channel control bits value (mode, edge and level bit fields).

Parameters

- base – TPM peripheral base address
- chnlNumber – The channel number

Returns
The control bits value. This is the logical OR of members of the enumeration *tpm_chnl_control_bit_mask_t*.

static inline *status_t* TPM_DisableChannel(TPM_Type *base, *tpm_chnl_t* chnlNumber)

Dsiable the channel.

This function disable the channel by clear all mode and level control bits.

Parameters

- base – TPM peripheral base address
- chnlNumber – The channel number

Returns
kStatus_Success PWM setup successful kStatus_Timeout PWM setup timeout when write register CnSC

static inline *status_t* TPM_EnableChannel(TPM_Type *base, *tpm_chnl_t* chnlNumber, uint8_t control)

Enable the channel according to mode and level configs.

This function enable the channel output according to input mode/level config parameters.

Parameters

- base – TPM peripheral base address
- chnlNumber – The channel number
- control – The contorl bits value. This is the logical OR of members of the enumeration *tpm_chnl_control_bit_mask_t*.

Returns
kStatus_Success PWM setup successful kStatus_Timeout PWM setup timeout when write register CnSC

void TPM_SetupInputCapture(TPM_Type *base, *tpm_chnl_t* chnlNumber, *tpm_input_capture_edge_t* captureMode)

Enables capturing an input signal on the channel using the function parameters.

When the edge specified in the captureMode argument occurs on the channel, the TPM counter is captured into the CnV register. The user has to read the CnV register separately to get this value.

Parameters

- base – TPM peripheral base address
- chnlNumber – The channel number
- captureMode – Specifies which edge to capture

status_t TPM_SetupOutputCompare(TPM_Type *base, *tpm_chnl_t* chnlNumber, *tpm_output_compare_mode_t* compareMode, uint32_t compareValue)

Configures the TPM to generate timed pulses.

When the TPM counter matches the value of compareVal argument (this is written into CnV reg), the channel output is changed based on what is specified in the compareMode argument.

Parameters

- base – TPM peripheral base address
- chnlNumber – The channel number
- compareMode – Action to take on the channel output when the compare condition is met
- compareValue – Value to be programmed in the CnV register.

Returns

kStatus_Success PWM setup successful kStatus_Timeout PWM setup timeout when write register CnV

void TPM_SetupDualEdgeCapture(TPM_Type *base, *tpm_chnl_t* chnlPairNumber, const *tpm_dual_edge_capture_param_t* *edgeParam, uint32_t filterValue)

Configures the dual edge capture mode of the TPM.

This function allows to measure a pulse width of the signal on the input of channel of a channel pair. The filter function is disabled if the filterVal argument passed is zero.

Parameters

- base – TPM peripheral base address
- chnlPairNumber – The TPM channel pair number; options are 0, 1, 2, 3
- edgeParam – Sets up the dual edge capture function
- filterValue – Filter value, specify 0 to disable filter.

void TPM_SetupQuadDecode(TPM_Type *base, const *tpm_phase_params_t* *phaseAParams, const *tpm_phase_params_t* *phaseBParams, *tpm_quad_decode_mode_t* quadMode)

Configures the parameters and activates the quadrature decode mode.

Parameters

- base – TPM peripheral base address
- phaseAParams – Phase A configuration parameters
- phaseBParams – Phase B configuration parameters
- quadMode – Selects encoding mode used in quadrature decoder mode

static inline void TPM_SetChannelPolarity(TPM_Type *base, *tpm_chnl_t* chnlNumber, bool enable)

Set the input and output polarity of each of the channels.

Parameters

- base – TPM peripheral base address
- chnlNumber – The channel number
- enable – true: Set the channel polarity to active high; false: Set the channel polarity to active low;

static inline void TPM_EnableChannelExtTrigger(TPM_Type *base, *tpm_chnl_t* chnlNumber, bool enable)

Enable external trigger input to be used by channel.

In input capture mode, configures the trigger input that is used by the channel to capture the counter value. In output compare or PWM mode, configures the trigger input used to modulate the channel output. When modulating the output, the output is forced to the channel initial value whenever the trigger is not asserted.

Note: No matter how many external trigger sources there are, only input trigger 0 and 1 are used. The even numbered channels share the input trigger 0 and the odd numbered channels share the second input trigger 1.

Parameters

- base – TPM peripheral base address
- chnlNumber – The channel number
- enable – true: Configures trigger input 0 or 1 to be used by channel; false: Trigger input has no effect on the channel

void TPM_EnableInterrupts(TPM_Type *base, uint32_t mask)

Enables the selected TPM interrupts.

Parameters

- base – TPM peripheral base address
- mask – The interrupts to enable. This is a logical OR of members of the enumeration *tpm_interrupt_enable_t*

void TPM_DisableInterrupts(TPM_Type *base, uint32_t mask)

Disables the selected TPM interrupts.

Parameters

- base – TPM peripheral base address
- mask – The interrupts to disable. This is a logical OR of members of the enumeration *tpm_interrupt_enable_t*

uint32_t TPM_GetEnabledInterrupts(TPM_Type *base)

Gets the enabled TPM interrupts.

Parameters

- base – TPM peripheral base address

Returns

The enabled interrupts. This is the logical OR of members of the enumeration *tpm_interrupt_enable_t*

void TPM_RegisterCallBack(TPM_Type *base, *tpm_callback_t* callback)

Register callback.

If channel or overflow interrupt is enabled by the user, then a callback can be registered which will be invoked when the interrupt is triggered.

Parameters

- base – TPM peripheral base address
- callback – Callback function

void TPM_DriverIRQHandler(uint32_t instance)

TPM driver IRQ handler common entry.

This function provides the common IRQ request entry for TPM.

Parameters

- instance – TPM instance.

static inline uint32_t TPM_GetChannelValue(TPM_Type *base, *tpm_chnl_t* chnlNumber)

Gets the TPM channel value.

Note: The TPM channel value contain the captured TPM counter value for the input modes or the match value for the output modes.

Parameters

- base – TPM peripheral base address
- chnlNumber – The channel number

Returns

The channle CnV regisyer value.

static inline uint32_t TPM_GetStatusFlags(TPM_Type *base)

Gets the TPM status flags.

Parameters

- base – TPM peripheral base address

Returns

The status flags. This is the logical OR of members of the enumeration *tpm_status_flags_t*

static inline void TPM_ClearStatusFlags(TPM_Type *base, uint32_t mask)

Clears the TPM status flags.

Parameters

- base – TPM peripheral base address
- mask – The status flags to clear. This is a logical OR of members of the enumeration *tpm_status_flags_t*

static inline *status_t* TPM_SetTimerPeriod(TPM_Type *base, uint32_t ticks)

Sets the timer period in units of ticks.

Timers counts from 0 until it equals the count value set here. The count value is written to the MOD register.

Note:

a. This API allows the user to use the TPM module as a timer. Do not mix usage of this API with TPM's PWM setup API's.

b. Call the utility macros provided in the fsl_common.h to convert usec or msec to ticks.

Parameters

- base – TPM peripheral base address
- ticks – A timer period in units of ticks, which should be equal or greater than 1.

Returns
kStatus_Success PWM setup successful kStatus_Timeout PWM setup timeout when write register CnSC

static inline uint32_t TPM_GetCurrentTimerCount(TPM_Type *base)

Reads the current timer counting value.

This function returns the real-time timer counting value in a range from 0 to a timer period.

Note: Call the utility macros provided in the fsl_common.h to convert ticks to usec or msec.

Parameters

- base – TPM peripheral base address

Returns
The current counter value in ticks

static inline void TPM_StartTimer(TPM_Type *base, *tpm_clock_source_t* clockSource)

Starts the TPM counter.

Parameters

- base – TPM peripheral base address
- clockSource – TPM clock source; once clock source is set the counter will start running

static inline *status_t* TPM_StopTimer(TPM_Type *base)

Stops the TPM counter.

Parameters

- base – TPM peripheral base address

Returns
kStatus_Success PWM setup successful kStatus_Timeout PWM setup timeout when write register CnSC

FSL_TPM_DRIVER_VERSION

TPM driver version 2.5.0.

enum _tpm_chnl

List of TPM channels.

Note: Actual number of available channels is SoC dependent

Values:

enumerator kTPM_Chnl_0

TPM channel number 0

enumerator kTPM_Chnl_1

TPM channel number 1

enumerator kTPM_Chnl_2

TPM channel number 2

enumerator kTPM_Chnl_3
TPM channel number 3

enumerator kTPM_Chnl_4
TPM channel number 4

enumerator kTPM_Chnl_5
TPM channel number 5

enumerator kTPM_Chnl_6
TPM channel number 6

enumerator kTPM_Chnl_7
TPM channel number 7

enum _tpm_pwm_mode
TPM PWM operation modes.

Values:

enumerator kTPM_EdgeAlignedPwm
Edge aligned PWM

enumerator kTPM_CombinedPwm
Combined PWM (Edge-aligned, center-aligned, or asymmetrical PWMs can be obtained in combined mode using different software configurations)

enum _tpm_pwm_level_select
TPM PWM output pulse mode: high-true, low-true or no output.

Note: When the TPM has PWM pause level select feature, the PWM output cannot be turned off by selecting the output level. In this case, the channel must be closed to close the PWM output.

Values:

enumerator kTPM_HighTrue
High true pulses

enumerator kTPM_LowTrue
Low true pulses

enum _tpm_pwm_pause_level_select
TPM PWM output when first enabled or paused: set or clear.

Values:

enumerator kTPM_ClearOnPause
Clear Output when counter first enabled or paused.

enumerator kTPM_SetOnPause

Set Output when counter first enabled or paused.

enum _tpm_chnl_control_bit_mask

List of TPM channel modes and level control bit mask.

Values:

enumerator kTPM_ChnlELSnAMask

Channel ELSA bit mask.

enumerator kTPM_ChnlELSnBMask

Channel ELSB bit mask.

enumerator kTPM_ChnlMSAMask

Channel MSA bit mask.

enumerator kTPM_ChnlMSBMask

Channel MSB bit mask.

enum _tpm_trigger_select

Trigger sources available.

This is used for both internal & external trigger sources (external trigger sources available in certain SoC's)

Note: The actual trigger sources available is SoC-specific.

Values:

enumerator kTPM_Trigger_Select_0

enumerator kTPM_Trigger_Select_1

enumerator kTPM_Trigger_Select_2

enumerator kTPM_Trigger_Select_3

enumerator kTPM_Trigger_Select_4

enumerator kTPM_Trigger_Select_5

enumerator kTPM_Trigger_Select_6

enumerator kTPM_Trigger_Select_7

enumerator kTPM_Trigger_Select_8

enumerator kTPM_Trigger_Select_9

enumerator kTPM_Trigger_Select_10

enumerator kTPM_Trigger_Select_11

enumerator kTPM_Trigger_Select_12

enumerator kTPM_Trigger_Select_13

enumerator kTPM_Trigger_Select_14

enumerator kTPM_Trigger_Select_15

enum _tpm_trigger_source

Trigger source options available.

Note: This selection is available only on some SoC's. For SoC's without this selection, the only trigger source available is internal triger.

Values:

enumerator kTPM_TriggerSource_External

Use external trigger input

enumerator kTPM_TriggerSource_Internal

Use internal trigger (channel pin input capture)

enum _tpm_ext_trigger_polarity

External trigger source polarity.

Note: Selects the polarity of the external trigger source.

Values:

enumerator kTPM_ExtTrigger_Active_High

External trigger input is active high

enumerator kTPM_ExtTrigger_Active_Low

External trigger input is active low

enum _tpm_output_compare_mode

TPM output compare modes.

Values:

enumerator kTPM_NoOutputSignal

No channel output when counter reaches CnV

enumerator kTPM_ToggleOnMatch
Toggle output

enumerator kTPM_ClearOnMatch
Clear output

enumerator kTPM_SetOnMatch
Set output

enumerator kTPM_HighPulseOutput
Pulse output high

enumerator kTPM_LowPulseOutput
Pulse output low

enum _tpm_input_capture_edge
TPM input capture edge.

Values:

enumerator kTPM_RisingEdge
Capture on rising edge only

enumerator kTPM_FallingEdge
Capture on falling edge only

enumerator kTPM_RiseAndFallEdge
Capture on rising or falling edge

enum _tpm_quad_decode_mode
TPM quadrature decode modes.

Note: This mode is available only on some SoC's.

Values:

enumerator kTPM_QuadPhaseEncode
Phase A and Phase B encoding mode

enumerator kTPM_QuadCountAndDir
Count and direction encoding mode

enum _tpm_phase_polarity
TPM quadrature phase polarities.

Values:

enumerator kTPM_QuadPhaseNormal
Phase input signal is not inverted

enumerator kTPM_QuadPhaseInvert
Phase input signal is inverted

enum _tpm_clock_source
TPM clock source selection.
Values:

enumerator kTPM_SystemClock
System clock

enumerator kTPM_ExternalClock
External TPM_EXTCLK pin clock

enumerator kTPM_ExternalInputTriggerClock
Selected external input trigger clock

enum _tpm_clock_prescale
TPM prescale value selection for the clock source.
Values:

enumerator kTPM_Prescale_Divide_1
Divide by 1

enumerator kTPM_Prescale_Divide_2
Divide by 2

enumerator kTPM_Prescale_Divide_4
Divide by 4

enumerator kTPM_Prescale_Divide_8
Divide by 8

enumerator kTPM_Prescale_Divide_16
Divide by 16

enumerator kTPM_Prescale_Divide_32
Divide by 32

enumerator kTPM_Prescale_Divide_64
Divide by 64

enumerator kTPM_Prescale_Divide_128
Divide by 128

enum _tpm_interrupt_enable
List of TPM interrupts.
Values:

enumerator kTPM_Chnl0InterruptEnable
Channel 0 interrupt.

enumerator kTPM_Chnl1InterruptEnable
Channel 1 interrupt.

enumerator kTPM_Chnl2InterruptEnable
Channel 2 interrupt.

enumerator kTPM_Chnl3InterruptEnable
Channel 3 interrupt.

enumerator kTPM_Chnl4InterruptEnable
Channel 4 interrupt.

enumerator kTPM_Chnl5InterruptEnable
Channel 5 interrupt.

enumerator kTPM_Chnl6InterruptEnable
Channel 6 interrupt.

enumerator kTPM_Chnl7InterruptEnable
Channel 7 interrupt.

enumerator kTPM_TimeOverflowInterruptEnable
Time overflow interrupt.

enum _tpm_status_flags
List of TPM flags.

Values:

enumerator kTPM_Chnl0Flag
Channel 0 flag

enumerator kTPM_Chnl1Flag
Channel 1 flag

enumerator kTPM_Chnl2Flag
Channel 2 flag

enumerator kTPM_Chnl3Flag
Channel 3 flag

enumerator kTPM_Chnl4Flag
Channel 4 flag

enumerator kTPM_Chnl5Flag
Channel 5 flag

enumerator kTPM_Chnl6Flag
Channel 6 flag

enumerator kTPM_Chnl7Flag
Channel 7 flag

enumerator kTPM_TimeOverflowFlag
Time overflow flag

typedef enum *_tpm_chnl* tpm_chnl_t
List of TPM channels.

Note: Actual number of available channels is SoC dependent

typedef enum *_tpm_pwm_mode* tpm_pwm_mode_t
TPM PWM operation modes.

typedef enum *_tpm_pwm_level_select* tpm_pwm_level_select_t
TPM PWM output pulse mode: high-true, low-true or no output.

Note: When the TPM has PWM pause level select feature, the PWM output cannot be turned off by selecting the output level. In this case, the channel must be closed to close the PWM output.

typedef enum *_tpm_pwm_pause_level_select* tpm_pwm_pause_level_select_t
TPM PWM output when first enabled or paused: set or clear.

typedef enum *_tpm_chnl_control_bit_mask* tpm_chnl_control_bit_mask_t
List of TPM channel modes and level control bit mask.

typedef struct *_tpm_chnl_pwm_signal_param* tpm_chnl_pwm_signal_param_t
Options to configure a TPM channel's PWM signal.

typedef enum *_tpm_trigger_select* tpm_trigger_select_t
Trigger sources available.

This is used for both internal & external trigger sources (external trigger sources available in certain SoC's)

Note: The actual trigger sources available is SoC-specific.

typedef enum *_tpm_trigger_source* tpm_trigger_source_t
Trigger source options available.

Note: This selection is available only on some SoC's. For SoC's without this selection, the only trigger source available is internal triger.

typedef enum *_tpm_ext_trigger_polarity* tpm_ext_trigger_polarity_t

External trigger source polarity.

Note: Selects the polarity of the external trigger source.

typedef enum *_tpm_output_compare_mode* tpm_output_compare_mode_t

TPM output compare modes.

typedef enum *_tpm_input_capture_edge* tpm_input_capture_edge_t

TPM input capture edge.

typedef struct *_tpm_dual_edge_capture_param* tpm_dual_edge_capture_param_t

TPM dual edge capture parameters.

Note: This mode is available only on some SoC's.

typedef enum *_tpm_quad_decode_mode* tpm_quad_decode_mode_t

TPM quadrature decode modes.

Note: This mode is available only on some SoC's.

typedef enum *_tpm_phase_polarity* tpm_phase_polarity_t

TPM quadrature phase polarities.

typedef struct *_tpm_phase_param* tpm_phase_params_t

TPM quadrature decode phase parameters.

typedef enum *_tpm_clock_source* tpm_clock_source_t

TPM clock source selection.

typedef enum *_tpm_clock_prescale* tpm_clock_prescale_t

TPM prescale value selection for the clock source.

typedef struct *_tpm_config* tpm_config_t

TPM config structure.

This structure holds the configuration settings for the TPM peripheral. To initialize this structure to reasonable defaults, call the *TPM_GetDefaultConfig()* function and pass a pointer to your config structure instance.

The config struct can be made const so it resides in flash

typedef enum *_tpm_interrupt_enable* tpm_interrupt_enable_t

List of TPM interrupts.

typedef enum *_tpm_status_flags* tpm_status_flags_t

List of TPM flags.

typedef void (*tpm_callback_t)(TPM_Type *base)

TPM callback function pointer.

Param base
TPM peripheral base address.

TPM_TIMEOUT

Max loops to wait for writing register.

When writing MOD CnV CnSC and SC register, driver will wait until register is updated. This parameter defines how many loops to check completion before return timeout. If defined as 0, driver will wait forever until completion.

TPM_MAX_COUNTER_VALUE(x)

Help macro to get the max counter value.

struct _tpm_chnl_pwm_signal_param

#include <fsl_tpm.h> Options to configure a TPM channel's PWM signal.

Public Members

tpm_chnl_t chnlNumber

TPM channel to configure. In combined mode (available in some SoC's), this represents the channel pair number

tpm_pwm_pause_level_select_t pauseLevel

PWM output level when counter first enabled or paused

tpm_pwm_level_select_t level

PWM output active level select

uint8_t dutyCyclePercent

PWM pulse width, value should be between 0 to 100 0=inactive signal(0% duty cycle)... 100=always active signal (100% duty cycle)

uint8_t firstEdgeDelayPercent

Used only in combined PWM mode to generate asymmetrical PWM. Specifies the delay to the first edge in a PWM period. If unsure, leave as 0. Should be specified as percentage of the PWM period, (dutyCyclePercent + firstEdgeDelayPercent) value should be not greate than 100.

bool enableComplementary

Used only in combined PWM mode. true: The combined channels output complementary signals; false: The combined channels output same signals;

tpm_pwm_pause_level_select_t secPauseLevel

Used only in combined PWM mode. Define the second channel output level when counter first enabled or paused

uint8_t deadTimeValue[2]

The dead time value for channel n and n+1 in combined complementary PWM mode. Deadtime insertion is disabled when this value is zero, otherwise deadtime insertion for channel n/n+1 is configured as (deadTimeValue * 4) clock cycles. deadTimeValue's available range is 0 ~ 15.

struct __tpm_dual_edge_capture_param

#include <fsl_tpm.h> TPM dual edge capture parameters.

Note: This mode is available only on some SoC's.

Public Members

bool enableSwap

true: Use channel n+1 input, channel n input is ignored; false: Use channel n input, channel n+1 input is ignored

tpm_input_capture_edge_t currChanEdgeMode

Input capture edge select for channel n

tpm_input_capture_edge_t nextChanEdgeMode

Input capture edge select for channel n+1

struct __tpm_phase_param

#include <fsl_tpm.h> TPM quadrature decode phase parameters.

Public Members

uint32_t phaseFilterVal

Filter value, filter is disabled when the value is zero

tpm_phase_polarity_t phasePolarity

Phase polarity

struct __tpm_config

#include <fsl_tpm.h> TPM config structure.

This structure holds the configuration settings for the TPM peripheral. To initialize this structure to reasonable defaults, call the *TPM_GetDefaultConfig()* function and pass a pointer to your config structure instance.

The config struct can be made const so it resides in flash

Public Members

tpm_clock_prescale_t prescale

Select TPM clock prescale value

bool useGlobalTimeBase

true: The TPM channels use an external global time base (the local counter still use for generate overflow interrupt and DMA request); false: All TPM channels use the local counter as their timebase

bool syncGlobalTimeBase

true: The TPM counter is synchronized to the global time base; false: disabled

tpm_trigger_select_t triggerSelect

Input trigger to use for controlling the counter operation

tpm_trigger_source_t triggerSource

Decides if we use external or internal trigger.

tpm_ext_trigger_polarity_t extTriggerPolarity

when using external trigger source, need selects the polarity of it.

bool enableDoze

true: TPM counter is paused in doze mode; false: TPM counter continues in doze mode

bool enableDebugMode

true: TPM counter continues in debug mode; false: TPM counter is paused in debug mode

bool enableReloadOnTrigger

true: TPM counter is reloaded on trigger; false: TPM counter not reloaded

bool enableStopOnOverflow

true: TPM counter stops after overflow; false: TPM counter continues running after overflow

bool enableStartOnTrigger

true: TPM counter only starts when a trigger is detected; false: TPM counter starts immediately

bool enablePauseOnTrigger

true: TPM counter will pause while trigger remains asserted; false: TPM counter continues running

uint8_t chnlPolarity

Defines the input/output polarity of the channels in POL register

2.67 TRDC: Trusted Resource Domain Controller

void TRDC_Init(TRDC_Type *base)

Initializes the TRDC module.

This function enables the TRDC clock.

Parameters

- base – TRDC peripheral base address.

void TRDC_Deinit(TRDC_Type *base)

De-initializes the TRDC module.

This function disables the TRDC clock.

Parameters

- base – TRDC peripheral base address.

static inline uint8_t TRDC_GetCurrentMasterDomainId(TRDC_Type *base)

Gets the domain ID of the current bus master.

Parameters

- base – TRDC peripheral base address.

Returns

Domain ID of current bus master.

void TRDC_GetHardwareConfig(TRDC_Type *base, *trdc_hardware_config_t* *config)

Gets the TRDC hardware configuration.

This function gets the TRDC hardware configurations, including number of bus masters, number of domains, number of MRCs and number of PACs.

Parameters

- base – TRDC peripheral base address.
- config – Pointer to the structure to get the configuration.

static inline void TRDC_SetDacGlobalValid(TRDC_Type *base)

Sets the TRDC DAC(Domain Assignment Controllers) global valid.

Once enabled, it will remain enabled until next reset.

Parameters

- base – TRDC peripheral base address.

static inline void TRDC_LockMasterDomainAssignment(TRDC_Type *base, uint8_t master)

Locks the bus master domain assignment register.

This function locks the master domain assignment. After it is locked, the register can't be changed until next reset.

Parameters

- base – TRDC peripheral base address.
- master – Which master to configure.

static inline void TRDC_SetMasterDomainAssignmentValid(TRDC_Type *base, uint8_t master, bool valid)

Sets the master domain assignment as valid or invalid.

This function sets the master domain assignment as valid or invalid.

Parameters

- base – TRDC peripheral base address.
- master – Which master to configure.
- valid – True to set valid, false to set invalid.

void TRDC_GetDefaultProcessorDomainAssignment(*trdc_processor_domain_assignment_t* *domainAssignment)

Gets the default master domain assignment for the processor bus master.

This function gets the default master domain assignment for the processor bus master. It should only be used for the processor bus masters, such as CORE0. This function sets the assignment as follows:

```
assignment->domainId          = 0U;
assignment->domainIdSelect    = kTRDC_DidMda;
assignment->lock              = 0U;
```

Parameters

- domainAssignment – Pointer to the assignment structure.

void TRDC_GetDefaultNonProcessorDomainAssignment(*trdc_non_processor_domain_assignment_t* *domainAssignment)

Gets the default master domain assignment for non-processor bus master.

This function gets the default master domain assignment for non-processor bus master. It should only be used for the non-processor bus masters, such as DMA. This function sets the assignment as follows:

```
assignment->domainId          = 0U;
assignment->privilegeAttr     = kTRDC_ForceUser;
assignment->secureAttr        = kTRDC_ForceSecure;
assignment->bypassDomainId    = 0U;
assignment->lock              = 0U;
```

Parameters

- domainAssignment – Pointer to the assignment structure.

void TRDC_SetProcessorDomainAssignment(TRDC_Type *base, const *trdc_processor_domain_assignment_t* *domainAssignment)

Sets the processor bus master domain assignment.

This function sets the processor master domain assignment as valid. One bus master might have multiple domain assignment registers. The parameter assignIndex specifies which assignment register to set.

Example: Set domain assignment for core 0.

```
trdc_processor_domain_assignment_t processorAssignment;

TRDC_GetDefaultProcessorDomainAssignment(&processorAssignment);

processorAssignment.domainId = 0;
processorAssignment.xxx      = xxx;
TRDC_SetMasterDomainAssignment(TRDC, &processorAssignment);
```

Parameters

- base – TRDC peripheral base address.
- domainAssignment – Pointer to the assignment structure.

static inline void TRDC_EnableProcessorDomainAssignment(TRDC_Type *base, bool enable)

Enables the processor bus master domain assignment.

Parameters

- base – TRDC peripheral base address.
- enable – True to enable, false to disable.

void TRDC_SetNonProcessorDomainAssignment(TRDC_Type *base, uint8_t master, const *trdc_non_processor_domain_assignment_t* *domainAssignment)

Sets the non-processor bus master domain assignment.

This function sets the non-processor master domain assignment as valid. One bus master might have multiple domain assignment registers. The parameter assignIndex specifies which assignment register to set.

Example: Set domain assignment for DMA0.

```
trdc_non_processor_domain_assignment_t nonProcessorAssignment;

TRDC_GetDefaultNonProcessorDomainAssignment(&nonProcessorAssignment);
nonProcessorAssignment.domainId = 1;
nonProcessorAssignment.xxx      = xxx;

TRDC_SetMasterDomainAssignment(TRDC, kTrdcMasterDma0, 0U, &nonProcessorAssignment);
```

Parameters

- base – TRDC peripheral base address.
- master – Which master to configure, refer to trdc_master_t in processor header file.
- domainAssignment – Pointer to the assignment structure.

void TRDC_GetDefaultIDAUConfig(*trdc_idau_config_t* *idauConfiguration)

Gets the default IDAU(Implementation-Defined Attribution Unit) configuration.

```
config->lockSecureVTOR    = false;
config->lockNonsecureVTOR = false;
config->lockSecureMPU     = false;
config->lockNonsecureMPU  = false;
config->lockSAU           = false;
```

Parameters

- idauConfiguration – Pointer to the configuration structure.

void TRDC_SetIDAU(TRDC_Type *base, const *trdc_idau_config_t* *idauConfiguration)

Sets the IDAU(Implementation-Defined Attribution Unit) control configuration.

Example: Lock the secure and non-secure MPU registers.

```
trdc_idau_config_t idauConfiguration;

TRDC_GetDefaultIDAUConfig(&idauConfiguration);

idauConfiguration.lockSecureMPU = true;
idauConfiguration.lockNonsecureMPU      = true;
TRDC_SetIDAU(TRDC, &idauConfiguration);
```

Parameters

- base – TRDC peripheral base address.
- idauConfiguration – Pointer to the configuration structure.

static inline void TRDC_EnableFlashLogicalWindow(TRDC_Type *base, bool enable)

Enables/disables the FLW(flash logical window) function.

Parameters

- base – TRDC peripheral base address.
- enable – True to enable, false to disable.

static inline void TRDC_LockFlashLogicalWindow(TRDC_Type *base)

Locks FLW registers. Once locked the registers can noy be updated until next reset.

Parameters

- base – TRDC peripheral base address.

static inline uint32_t TRDC_GetFlashLogicalWindowPbase(TRDC_Type *base)

Gets the FLW physical base address.

Parameters

- base – TRDC peripheral base address.

Returns

Physical address of the FLW function.

static inline void TRDC_GetSetFlashLogicalWindowSize(TRDC_Type *base, uint16_t size)

Sets the FLW size.

Parameters

- base – TRDC peripheral base address.
- size – Size of the FLW in unit of 32k bytes.

void TRDC_GetDefaultFlashLogicalWindowConfig(*trdc_flw_config_t* *flwConfiguration)

Gets the default FLW(Flsh Logical Window) configuration.

```
config->blockCount     = false;
config->arrayBaseAddr = false;
config->lock      = false;
config->enable   = false;
```

Parameters

- flwConfiguration – Pointer to the configuration structure.

void TRDC_SetFlashLogicalWindow(TRDC_Type *base, const *trdc_flw_config_t* *flwConfiguration)

Sets the FLW function's configuration.

```
trdc_flw_config_t flwConfiguration;

TRDC_GetDefaultIDAUConfig(&flwConfiguration);

flwConfiguration.blockCount = 32U;
flwConfiguration.arrayBaseAddr = 0xXXXXXXXX;
TRDC_SetIDAU(TRDC, &flwConfiguration);
```

Parameters

- base – TRDC peripheral base address.

- flwConfiguration – Pointer to the configuration structure.

status_t TRDC_GetAndClearFirstDomainError(TRDC_Type *base, *trdc_domain_error_t* *error)

Gets and clears the first domain error of the current domain.

This function gets the first access violation information for the current domain and clears the pending flag. There might be multiple access violations pending for the current domain. This function only processes the first error.

Parameters

- base – TRDC peripheral base address.
- error – Pointer to the error information.

Returns

If the access violation is captured, this function returns the kStatus_Success. The error information can be obtained from the parameter error. If no access violation is captured, this function returns the kStatus_NoData.

status_t TRDC_GetAndClearFirstSpecificDomainError(TRDC_Type *base, *trdc_domain_error_t* *error, uint8_t domainId)

Gets and clears the first domain error of the specific domain.

This function gets the first access violation information for the specific domain and clears the pending flag. There might be multiple access violations pending for the current domain. This function only processes the first error.

Parameters

- base – TRDC peripheral base address.
- error – Pointer to the error information.
- domainId – The error of which domain to get and clear.

Returns

If the access violation is captured, this function returns the kStatus_Success. The error information can be obtained from the parameter error. If no access violation is captured, this function returns the kStatus_NoData.

static inline void TRDC_SetMrcGlobalValid(TRDC_Type *base)

Sets the TRDC MRC(Memory Region Checkers) global valid.

Once enabled, it will remain enabled until next reset.

Parameters

- base – TRDC peripheral base address.

static inline uint8_t TRDC_GetMrcRegionNumber(TRDC_Type *base, uint8_t mrcIdx)

Gets the TRDC MRC(Memory Region Checkers) region number valid.

Parameters

- base – TRDC peripheral base address.
- mrcIdx – MRC index.

Returns

the region number of the given MRC instance

void TRDC_MrcSetMemoryAccessConfig(TRDC_Type *base, const *trdc_memory_access_control_config_t* *config, uint8_t mrcIdx, uint8_t regIdx)

Sets the memory access configuration for one of the access control register of one MRC.

Example: Enable the secure operations and lock the configuration for MRC0 region 1.

```
trdc_memory_access_control_config_t config;

config.securePrivX = true;
config.securePrivW = true;
config.securePrivR = true;
config.lock = true;
TRDC_SetMrcMemoryAccess(TRDC, &config, 0, 1);
```

Parameters

- base – TRDC peripheral base address.
- config – Pointer to the configuration structure.
- mrcIdx – MRC index.
- regIdx – Register number.

void TRDC_MrcEnableDomainNseUpdate(TRDC_Type *base, uint8_t mrcIdx, uint16_t domainMask, bool enable)

Enables the update of the selected domains.

After the domains' update are enabled, their regions' NSE bits can be set or clear.

Parameters

- base – TRDC peripheral base address.
- mrcIdx – MRC index.
- domainMask – Bit mask of the domains to be enabled.
- enable – True to enable, false to disable.

void TRDC_MrcRegionNseSet(TRDC_Type *base, uint8_t mrcIdx, uint16_t regionMask)

Sets the NSE bits of the selected regions for domains.

This function sets the NSE bits for the selected regions for the domains whose update are enabled.

Parameters

- base – TRDC peripheral base address.
- mrcIdx – MRC index.
- regionMask – Bit mask of the regions whose NSE bits to set.

void TRDC_MrcRegionNseClear(TRDC_Type *base, uint8_t mrcIdx, uint16_t regionMask)

Clears the NSE bits of the selected regions for domains.

This function clears the NSE bits for the selected regions for the domains whose update are enabled.

Parameters

- base – TRDC peripheral base address.
- mrcIdx – MRC index.
- regionMask – Bit mask of the regions whose NSE bits to clear.

void TRDC_MrcDomainNseClear(TRDC_Type *base, uint8_t mrcIdx, uint16_t domainMask)

Clears the NSE bits for all the regions of the selected domains.

This function clears the NSE bits for all regions of selected domains whose update are enabled.

Parameters

- base – TRDC peripheral base address.
- mrcIdx – MRC index.
- domainMask – Bit mask of the domains whose NSE bits to clear.

void TRDC_MrcSetRegionDescriptorConfig(TRDC_Type *base, const *trdc_mrc_region_descriptor_config_t* *config)

Sets the configuration for one of the region descriptor per domain per MRC instnce.

This function sets the configuration for one of the region descriptor, including the start and end address of the region, memory access control policy and valid.

Parameters

- base – TRDC peripheral base address.
- config – Pointer to region descriptor configuration structure.

static inline void TRDC_SetMbcGlobalValid(TRDC_Type *base)

Sets the TRDC MBC(Memory Block Checkers) global valid.

Once enabled, it will remain enabled until next reset.

Parameters

- base – TRDC peripheral base address.

void TRDC_GetMbcHardwareConfig(TRDC_Type *base, *trdc_slave_memory_hardware_config_t* *config, uint8_t mbcIdx, uint8_t slvIdx)

Gets the hardware configuration of the one of two slave memories within each MBC(memory block checker).

Parameters

- base – TRDC peripheral base address.
- config – Pointer to the structure to get the configuration.
- mbcIdx – MBC number.
- slvIdx – Slave number.

void TRDC_MbcSetNseUpdateConfig(TRDC_Type *base, const *trdc_mbc_nse_update_config_t* *config, uint8_t mbcIdx)

Sets the NSR update configuration for one of the MBC instance.

After set the NSE configuration, the configured memory area can be updateby NSE set/clear.

Parameters

- base – TRDC peripheral base address.
- config – Pointer to NSE update configuration structure.
- mbcIdx – MBC index.

void TRDC_MbcWordNseSet(TRDC_Type *base, uint8_t mbcIdx, uint32_t bitMask)

Sets the NSE bits of the selected configuration words according to NSE update configuration.

This function sets the NSE bits of the word for the configured regio, memory.

Parameters

- base – TRDC peripheral base address.
- mbcIdx – MBC index.
- bitMask – Mask of the bits whose NSE bits to set.

void TRDC_MbcWordNseClear(TRDC_Type *base, uint8_t mbcIdx, uint32_t bitMask)

Clears the NSE bits of the selected configuration words according to NSE update configuration.

This function sets the NSE bits of the word for the configured regio, memory.

Parameters

- base – TRDC peripheral base address.
- mbcIdx – MBC index.
- bitMask – Mask of the bits whose NSE bits to clear.

void TRDC_MbcNseClearAll(TRDC_Type *base, uint8_t mbcIdx, uint16_t domainMask, uint8_t slaveMask)

Clears all configuration words' NSE bits of the selected domain and memory.

Parameters

- base – TRDC peripheral base address.
- mbcIdx – MBC index.
- domainMask – Mask of the domains whose NSE bits to clear, 0b110 means clear domain 1&2.
- slaveMask – Mask of the slaves whose NSE bits to clear, 0x11 means clear all slave 0&1's NSE bits.

void TRDC_MbcSetMemoryAccessConfig(TRDC_Type *base, const *trdc_memory_access_control_config_t* *config, uint8_t mbcIdx, uint8_t rgdIdx)

Sets the memory access configuration for one of the region descriptor of one MBC.

Example: Enable the secure operations and lock the configuration for MRC0 region 1.

```
trdc_memory_access_control_config_t config;

config.securePrivX = true;
config.securePrivW = true;
config.securePrivR = true;
config.lock = true;
TRDC_SetMbcMemoryAccess(TRDC, &config, 0, 1);
```

Parameters

- base – TRDC peripheral base address.
- config – Pointer to the configuration structure.
- mbcIdx – MBC index.
- rgdIdx – Region descriptor number.

void TRDC_MbcSetMemoryBlockConfig(TRDC_Type *base, const *trdc_mbc_memory_block_config_t* *config)

Sets the configuration for one of the memory block per domain per MBC instnce.

This function sets the configuration for one of the memory block, including the memory access control policy and nse enable.

Parameters

- base – TRDC peripheral base address.
- config – Pointer to memory block configuration structure.

enum __trdc__did__sel

TRDC domain ID select method, the register bit TRDC_MDA_W0_0_DFMT0[DIDS], used for domain hit evaluation.

Values:

enumerator kTRDC__DidMda

Use MDAn[2:0] as DID.

enumerator kTRDC__DidInput

Use the input DID (DID_in) as DID.

enumerator kTRDC__DidMdaAndInput

Use MDAn[2] concatenated with DID_in[1:0] as DID.

enumerator kTRDC__DidReserved

Reserved.

enum __trdc__secure__attr

TRDC secure attribute, the register bit TRDC_MDA_W0_0_DFMT0[SA], used for bus master domain assignment.

Values:

enumerator kTRDC__ForceSecure

Force the bus attribute for this master to secure.

enumerator kTRDC__ForceNonSecure

Force the bus attribute for this master to non-secure.

enumerator kTRDC__MasterSecure

Use the bus master's secure/nonsecure attribute directly.

enumerator kTRDC__MasterSecure1

Use the bus master's secure/nonsecure attribute directly.

enum __trdc__privilege__attr

TRDC privileged attribute, the register bit TRDC_MDA_W0_x_DFMT1[PA], used for non-processor bus master domain assignment.

Values:

enumerator kTRDC__ForceUser

Force the bus attribute for this master to user.

enumerator kTRDC__ForcePrivilege

Force the bus attribute for this master to privileged.

enumerator kTRDC__MasterPrivilege

Use the bus master's attribute directly.

enumerator kTRDC_MasterPrivilege1

Use the bus master's attribute directly.

enum _trdc_controller

TRDC controller definition for domain error check. Each TRDC instance may have different MRC or MBC count, call TRDC_GetHardwareConfig to get the actual count.

Values:

enumerator kTRDC_MemBlockController0

Memory block checker 0.

enumerator kTRDC_MemBlockController1

Memory block checker 1.

enumerator kTRDC_MemBlockController2

Memory block checker 2.

enumerator kTRDC_MemBlockController3

Memory block checker 3.

enumerator kTRDC_MemRegionChecker0

Memory region checker 0.

enumerator kTRDC_MemRegionChecker1

Memory region checker 1.

enumerator kTRDC_MemRegionChecker2

Memory region checker 2.

enumerator kTRDC_MemRegionChecker3

Memory region checker 3.

enumerator kTRDC_MemRegionChecker4

Memory region checker 4.

enumerator kTRDC_MemRegionChecker5

Memory region checker 5.

enumerator kTRDC_MemRegionChecker6

Memory region checker 6.

enum _trdc_error_state

TRDC domain error state definition TRDC_MBCn_DERR_W1[EST] or TRDC_MRCn_DERR_W1[EST].

Values:

enumerator kTRDC_ErrorStateNone

No access violation detected.

enumerator kTRDC_ErrorStateNone1
No access violation detected.

enumerator kTRDC_ErrorStateSingle
Single access violation detected.

enumerator kTRDC_ErrorStateMulti
Multiple access violation detected.

enum _trdc_error_attr
TRDC domain error attribute definition TRDC_MBCn_DERR_W1[EATR] or TRDC_MRCn_DERR_W1[EATR].

Values:

enumerator kTRDC_ErrorSecureUserInst
Secure user mode, instruction fetch access.

enumerator kTRDC_ErrorSecureUserData
Secure user mode, data access.

enumerator kTRDC_ErrorSecurePrivilegeInst
Secure privileged mode, instruction fetch access.

enumerator kTRDC_ErrorSecurePrivilegeData
Secure privileged mode, data access.

enumerator kTRDC_ErrorNonSecureUserInst
NonSecure user mode, instruction fetch access.

enumerator kTRDC_ErrorNonSecureUserData
NonSecure user mode, data access.

enumerator kTRDC_ErrorNonSecurePrivilegeInst
NonSecure privileged mode, instruction fetch access.

enumerator kTRDC_ErrorNonSecurePrivilegeData
NonSecure privileged mode, data access.

enum _trdc_error_type
TRDC domain error access type definition TRDC_DERR_W1_n[ERW].

Values:

enumerator kTRDC_ErrorTypeRead
Error occurs on read reference.

enumerator kTRDC_ErrorTypeWrite
Error occurs on write reference.

enum _trdc_region_descriptor

The region descriptor enumeration, used to form a mask to set/clear the NSE bits for one or several regions.

Values:

enumerator kTRDC_RegionDescriptor0
Region descriptor 0.

enumerator kTRDC_RegionDescriptor1
Region descriptor 1.

enumerator kTRDC_RegionDescriptor2
Region descriptor 2.

enumerator kTRDC_RegionDescriptor3
Region descriptor 3.

enumerator kTRDC_RegionDescriptor4
Region descriptor 4.

enumerator kTRDC_RegionDescriptor5
Region descriptor 5.

enumerator kTRDC_RegionDescriptor6
Region descriptor 6.

enumerator kTRDC_RegionDescriptor7
Region descriptor 7.

enumerator kTRDC_RegionDescriptor8
Region descriptor 8.

enumerator kTRDC_RegionDescriptor9
Region descriptor 9.

enumerator kTRDC_RegionDescriptor10
Region descriptor 10.

enumerator kTRDC_RegionDescriptor11
Region descriptor 11.

enumerator kTRDC_RegionDescriptor12
Region descriptor 12.

enumerator kTRDC_RegionDescriptor13
Region descriptor 13.

enumerator kTRDC_RegionDescriptor14
Region descriptor 14.

enumerator kTRDC_RegionDescriptor15
Region descriptor 15.

enum _trdc_MRC_domain
The MRC domain enumeration, used to form a mask to enable/disable the update or clear all NSE bits of one or several domains.

Values:

enumerator kTRDC_MrcDomain0
Domain 0.

enumerator kTRDC_MrcDomain1
Domain 1.

enumerator kTRDC_MrcDomain2
Domain 2.

enumerator kTRDC_MrcDomain3
Domain 3.

enumerator kTRDC_MrcDomain4
Domain 4.

enumerator kTRDC_MrcDomain5
Domain 5.

enumerator kTRDC_MrcDomain6
Domain 6.

enumerator kTRDC_MrcDomain7
Domain 7.

enumerator kTRDC_MrcDomain8
Domain 8.

enumerator kTRDC_MrcDomain9
Domain 9.

enumerator kTRDC_MrcDomain10
Domain 10.

enumerator kTRDC_MrcDomain11
Domain 11.

enumerator kTRDC_MrcDomain12
Domain 12.

enumerator kTRDC_MrcDomain13
Domain 13.

enumerator kTRDC_MrcDomain14
Domain 14.

enumerator kTRDC_MrcDomain15
Domain 15.

enum _trdc_MBC_domain
The MBC domain enumeration, used to form a mask to enable/disable the update or clear NSE bits of one or several domains.

Values:

enumerator kTRDC_MbcDomain0
Domain 0.

enumerator kTRDC_MbcDomain1
Domain 1.

enumerator kTRDC_MbcDomain2
Domain 2.

enumerator kTRDC_MbcDomain3
Domain 3.

enumerator kTRDC_MbcDomain4
Domain 4.

enumerator kTRDC_MbcDomain5
Domain 5.

enumerator kTRDC_MbcDomain6
Domain 6.

enumerator kTRDC_MbcDomain7
Domain 7.

enum _trdc_MBC_memory
The MBC slave memory enumeration, used to form a mask to enable/disable the update or clear NSE bits of one or several memory block.

Values:

enumerator kTRDC_MbcSlaveMemory0
Memory 0.

enumerator kTRDC_MbcSlaveMemory1
Memory 1.

enumerator kTRDC_MbcSlaveMemory2
Memory 2.

enumerator kTRDC_MbcSlaveMemory3
Memory 3.

enum _trdc_MBC_bit
The MBC bit enumeration, used to form a mask to set/clear configured words' NSE.
Values:

enumerator kTRDC_MbcBit0
Bit 0.

enumerator kTRDC_MbcBit1
Bit 1.

enumerator kTRDC_MbcBit2
Bit 2.

enumerator kTRDC_MbcBit3
Bit 3.

enumerator kTRDC_MbcBit4
Bit 4.

enumerator kTRDC_MbcBit5
Bit 5.

enumerator kTRDC_MbcBit6
Bit 6.

enumerator kTRDC_MbcBit7
Bit 7.

enumerator kTRDC_MbcBit8
Bit 8.

enumerator kTRDC_MbcBit9
Bit 9.

enumerator kTRDC_MbcBit10
Bit 10.

enumerator kTRDC_MbcBit11
Bit 11.

enumerator kTRDC_MbcBit12
Bit 12.

enumerator kTRDC_MbcBit13
Bit 13.

enumerator kTRDC_MbcBit14
Bit 14.

enumerator kTRDC_MbcBit15
Bit 15.

enumerator kTRDC_MbcBit16
Bit 16.

enumerator kTRDC_MbcBit17
Bit 17.

enumerator kTRDC_MbcBit18
Bit 18.

enumerator kTRDC_MbcBit19
Bit 19.

enumerator kTRDC_MbcBit20
Bit 20.

enumerator kTRDC_MbcBit21
Bit 21.

enumerator kTRDC_MbcBit22
Bit 22.

enumerator kTRDC_MbcBit23
Bit 23.

enumerator kTRDC_MbcBit24
Bit 24.

enumerator kTRDC_MbcBit25
Bit 25.

enumerator kTRDC_MbcBit26
Bit 26.

enumerator kTRDC_MbcBit27
Bit 27.

enumerator kTRDC_MbcBit28
Bit 28.

enumerator kTRDC_MbcBit29
Bit 29.

enumerator kTRDC_MbcBit30

Bit 30.

enumerator kTRDC_MbcBit31

Bit 31.

typedef struct *_trdc_hardware_config* trdc_hardware_config_t

TRDC hardware configuration.

typedef struct *_trdc_slave_memory_hardware_config* trdc_slave_memory_hardware_config_t

Hardware configuration of the two slave memories within each MBC(memory block checker).

typedef enum *_trdc_did_sel* trdc_did_sel_t

TRDC domain ID select method, the register bit TRDC_MDA_W0_0_DFMT0[DIDS], used for domain hit evaluation.

typedef enum *_trdc_secure_attr* trdc_secure_attr_t

TRDC secure attribute, the register bit TRDC_MDA_W0_0_DFMT0[SA], used for bus master domain assignment.

typedef struct *_trdc_processor_domain_assignment* trdc_processor_domain_assignment_t

Domain assignment for the processor bus master.

typedef enum *_trdc_privilege_attr* trdc_privilege_attr_t

TRDC privileged attribute, the register bit TRDC_MDA_W0_x_DFMT1[PA], used for non-processor bus master domain assignment.

typedef struct *_trdc_non_processor_domain_assignment* trdc_non_processor_domain_assignment_t

Domain assignment for the non-processor bus master.

typedef struct *_trdc_idau_config* trdc_idau_config_t

IDAU(Implementation-Defined Attribution Unit) configuration for TZ-M function control.

typedef struct *_trdc_flw_config* trdc_flw_config_t

FLW(Flash Logical Window) configuration.

typedef enum *_trdc_controller* trdc_controller_t

TRDC controller definition for domain error check. Each TRDC instance may have different MRC or MBC count, call TRDC_GetHardwareConfig to get the actual count.

typedef enum *_trdc_error_state* trdc_error_state_t

TRDC domain error state definition TRDC_MBCn_DERR_W1[EST] or TRDC_MRCn_DERR_W1[EST].

typedef enum *_trdc_error_attr* trdc_error_attr_t

TRDC domain error attribute definition TRDC_MBCn_DERR_W1[EATR] or TRDC_MRCn_DERR_W1[EATR].

typedef enum *_trdc_error_type* trdc_error_type_t

TRDC domain error access type definition TRDC_DERR_W1_n[ERW].

typedef struct *_trdc_domain_error* trdc_domain_error_t

TRDC domain error definition.

typedef struct *_trdc_memory_access_control_config* trdc_memory_access_control_config_t

Memory access control configuration for MBC/MRC.

typedef struct *_trdc_mrc_region_descriptor_config* trdc_mrc_region_descriptor_config_t

The configuration of each region descriptor per domain per MRC instance.

typedef struct *_trdc_mbc_nse_update_config* trdc_mbc_nse_update_config_t

The configuration of MBC NSE update.

typedef struct *_trdc_mbc_memory_block_config* trdc_mbc_memory_block_config_t

The configuration of each memory block per domain per MBC instance.

FSL_TRDC_DRIVER_VERSION

struct _trdc_hardware_config

#include <fsl_trdc.h> TRDC hardware configuration.

Public Members

uint8_t masterNumber

Number of bus masters.

uint8_t domainNumber

Number of domains.

uint8_t mbcNumber

Number of MBCs.

uint8_t mrcNumber

Number of MRCs.

struct _trdc_slave_memory_hardware_config

#include <fsl_trdc.h> Hardware configuration of the two slave memories within each MBC(memory block checker).

Public Members

uint32_t blockNum

Number of blocks.

uint32_t blockSize
Block size.

struct __trdc__processor__domain__assignment
#include <fsl_trdc.h> Domain assignment for the processor bus master.

Public Members

uint32_t domainId
Domain ID.

uint32_t domainIdSelect
Domain ID select method, see *trdc_did_sel_t*.

uint32_t ___pad0___
Reserved.

uint32_t secureAttr
Secure attribute, see *trdc_secure_attr_t*.

uint32_t ___pad1___
Reserved.

uint32_t lock
Lock the register.

uint32_t ___pad2___
Reserved.

struct __trdc__non__processor__domain__assignment
#include <fsl_trdc.h> Domain assignment for the non-processor bus master.

Public Members

uint32_t domainId
Domain ID.

uint32_t privilegeAttr
Privileged attribute, see *trdc_privilege_attr_t*.

uint32_t secureAttr
Secure attribute, see *trdc_secure_attr_t*.

uint32_t bypassDomainId
Bypass domain ID.

uint32_t ___pad0___

Reserved.

uint32_t lock

Lock the register.

uint32_t ___pad1___

Reserved.

struct __trdc__idau__config

#include <fsl_trdc.h> IDAU(Implementation-Defined Attribution Unit) configuration for TZ-M function control.

Public Members

uint32_t ___pad0___

Reserved.

uint32_t lockSecureVTOR

Disable writes to secure VTOR(Vector Table Offset Register).

uint32_t lockNonsecureVTOR

Disable writes to non-secure VTOR, Application interrupt and Reset Control Registers.

uint32_t lockSecureMPU

Disable writes to secure MPU(Memory Protection Unit) from software or from a debug agent connected to the processor in Secure state.

uint32_t lockNonsecureMPU

Disable writes to non-secure MPU(Memory Protection Unit) from software or from a debug agent connected to the processor.

uint32_t lockSAU

Disable writes to SAU(Security Attribution Unit) registers.

uint32_t ___pad1___

Reserved.

struct __trdc__flw__config

#include <fsl_trdc.h> FLW(Flash Logical Window) configuration.

Public Members

uint16_t blockCount

Block count of the Flash Logic Window in 32KByte blocks.

uint32_t arrayBaseAddr
Flash array base address of the Flash Logical Window.

bool lock
Disable writes to FLW registers.

bool enable
Enable FLW function.

struct __trdc__domain__error
#include <fsl_trdc.h> TRDC domain error definition.

Public Members

trdc_controller_t controller
Which controller captured access violation.

uint32_t address
Access address that generated access violation.

trdc_error_state_t errorState
Error state.

trdc_error_attr_t errorAttr
Error attribute.

trdc_error_type_t errorType
Error type.

uint8_t errorPort
Error port.

uint8_t domainId
Domain ID.

uint8_t slaveMemoryIdx
The slave memory index. Only apply when violation in MBC.

struct __trdc__memory__access__control__config
#include <fsl_trdc.h> Memory access control configuration for MBC/MRC.

Public Members

uint32_t nonsecureUsrX
Allow nonsecure user execute access.

uint32_t nonsecureUsrW
Allow nonsecure user write access.

uint32_t nonsecureUsrR
Allow nonsecure user read access.

uint32_t ___pad0___
Reserved.

uint32_t nonsecurePrivX
Allow nonsecure privilege execute access.

uint32_t nonsecurePrivW
Allow nonsecure privilege write access.

uint32_t nonsecurePrivR
Allow nonsecure privilege read access.

uint32_t ___pad1___
Reserved.

uint32_t secureUsrX
Allow secure user execute access.

uint32_t secureUsrW
Allow secure user write access.

uint32_t secureUsrR
Allow secure user read access.

uint32_t ___pad2___
Reserved.

uint32_t securePrivX
Allownsecure privilege execute access.

uint32_t securePrivW
Allownsecure privilege write access.

uint32_t securePrivR
Allownsecure privilege read access.

uint32_t ___pad3___
Reserved.

uint32_t lock
Lock the configuration until next reset, only apply to access control register 0.

struct __trdc__mrc__region__descriptor__config

#include <fsl_trdc.h> The configuration of each region descriptor per domain per MRC instance.

Public Members

uint8_t memoryAccessControlSelect

Select one of the 8 access control policies for this region, for access cotrol policies see *trdc_memory_access_control_config_t*.

uint32_t startAddr

Physical start address.

bool valid

Lock the register.

bool nseEnable

Enable non-secure accesses and disable secure accesses.

uint32_t endAddr

Physical start address.

uint8_t mrcIdx

The index of the MRC for this configuration to take effect.

uint8_t domainIdx

The index of the domain for this configuration to take effect.

uint8_t regionIdx

The index of the region for this configuration to take effect.

struct __trdc__mbc__nse__update__config

#include <fsl_trdc.h> The configuration of MBC NSE update.

Public Members

uint32_t ___pad0___

Reserved.

uint32_t wordIdx

MBC configuration word index to be updated.

uint32_t ___pad1___

Reserved.

uint32_t memorySelect

Bit mask of the selected memory to be updated. *_trdc_MBC_memory*.

uint32_t ___pad2___

Reserved.

uint32_t domianSelect

Bit mask of the selected domain to be updated. *_trdc_MBC_domain*.

uint32_t ___pad3___

Reserved.

uint32_t autoIncrement

Whether to increment the word index after current word is updated using this configuration.

struct __trdc__mbc__memory__block__config

#include <fsl_trdc.h> The configuration of each memory block per domain per MBC instance.

Public Members

uint32_t memoryAccessControlSelect

Select one of the 8 access control policies for this memory block, for access cotrol policies see *trdc_memory_access_control_config_t*.

uint32_t nseEnable

Enable non-secure accesses and disable secure accesses.

uint32_t mbcIdx

The index of the MBC for this configuration to take effect.

uint32_t domainIdx

The index of the domain for this configuration to take effect.

uint32_t slaveMemoryIdx

The index of the slave memory for this configuration to take effect.

uint32_t memoryBlockIdx

The index of the memory block for this configuration to take effect.

2.68 TRGMUX: Trigger Mux Driver

static inline void TRGMUX_LockRegister(TRGMUX_Type *base, uint32_t index)

Sets the flag of the register which is used to mark writeable.

The function sets the flag of the register which is used to mark writeable. Example:

```
TRGMUX_LockRegister(TRGMUX0,kTRGMUX_Trgmux0Dmamux0);
```

Parameters

- base – TRGMUX peripheral base address.

- index – The index of the TRGMUX register, see the enum trgmux_device_t defined in <SOC>.h.

status_t TRGMUX_SetTriggerSource(TRGMUX_Type *base, uint32_t index, *trgmux_trigger_input_t* input, uint32_t trigger_src)

Configures the trigger source of the appointed peripheral.

The function configures the trigger source of the appointed peripheral. Example:

```
TRGMUX_SetTriggerSource(TRGMUX0, kTRGMUX_Trgmux0Dmamux0, kTRGMUX_TriggerInput0,
↪ kTRGMUX_SourcePortPin);
```

Parameters

- base – TRGMUX peripheral base address.
- index – The index of the TRGMUX register, see the enum trgmux_device_t defined in <SOC>.h.
- input – The MUX select for peripheral trigger input
- trigger_src – The trigger inputs for various peripherals. See the enum trgmux_source_t defined in <SOC>.h.

Return values

- kStatus_Success – Configured successfully.
- kStatus_TRGMUX_Locked – Configuration failed because the register is locked.

FSL_TRGMUX_DRIVER_VERSION

TRGMUX driver version.

TRGMUX configure status.

Values:

enumerator kStatus_TRGMUX_Locked

Configure failed for register is locked

enum _trgmux_trigger_input

Defines the MUX select for peripheral trigger input.

Values:

enumerator kTRGMUX_TriggerInput0

The MUX select for peripheral trigger input 0

enumerator kTRGMUX_TriggerInput1

The MUX select for peripheral trigger input 1

enumerator kTRGMUX_TriggerInput2

The MUX select for peripheral trigger input 2

enumerator kTRGMUX_TriggerInput3

The MUX select for peripheral trigger input 3

typedef enum *_trgmux_trigger_input* trgmux_trigger_input_t

Defines the MUX select for peripheral trigger input.

2.69 VBAT: Smart Power Switch

The enumeration of VBAT module status.

Values:

enumerator kStatus_VBAT_Fro16kNotEnabled

Internal 16kHz free running oscillator not enabled.

enumerator kStatus_VBAT_BandgapNotEnabled

Bandgap not enabled.

enum _vbat_status_flag

The enumeration of VBAT status flags.

Values:

enumerator kVBAT_StatusFlagPORDetect

VBAT domain has been reset

enumerator kVBAT_StatusFlagWakeupPin

A falling edge is detected on the wakeup pin.

enumerator kVBAT_StatusFlagBandgapTimer0

Bandgap Timer0 period reached.

enumerator kVBAT_StatusFlagBandgapTimer1

Bandgap Timer1 period reached.

enumerator kVBAT_StatusFlagLdoReady

LDO is enabled and ready.

enum _vbat_interrupt_enable

The enumeration of VBAT interrupt enable.

Values:

enumerator kVBAT_InterruptEnablePORDetect

Enable POR detect interrupt.

enumerator kVBAT_InterruptEnableWakeupPin

Enable the interrupt when a falling edge is detected on the wakeup pin.

enumerator kVBAT_InterruptEnableBandgapTimer0
Enable the interrupt if Bandgap Timer0 period reached.

enumerator kVBAT_InterruptEnableBandgapTimer1
Enable the interrupt if Bandgap Timer1 period reached.

enumerator kVBAT_InterruptEnableLdoReady
Enable LDO ready interrupt.

enumerator kVBAT_AllInterruptsEnable
Enable all interrupts.

enum _vbat_wakeup_enable
The enumeration of VBAT wakeup enable.

Values:

enumerator kVBAT_WakeupEnablePORDetect
Enable POR detect wakeup.

enumerator kVBAT_WakeupEnableWakeupPin
Enable wakeup feature when a falling edge is detected on the wakeup pin.

enumerator kVBAT_WakeupEnableBandgapTimer0
Enable wakeup feature when bandgap timer0 period reached.

enumerator kVBAT_WakeupEnableBandgapTimer1
Enable wakeup feature when bandgap timer1 period reached.

enumerator kVBAT_WakeupEnableLdoReady
Enable wakeup when LDO ready.

enumerator kVBAT_AllWakeupsEnable
Enable all wakeup.

enum _vbat_bandgap_timer_id
The enumeration of bandgap timer id, VBAT support two bandgap timers.

Values:

enumerator kVBAT_BandgapTimer0
Bandgap Timer0.

enumerator kVBAT_BandgapTimer1
Bandgap Timer1.

enum _vbat_bandgap_refresh_period

The enumeration of bandgap refresh period.

Values:

enumerator kVBAT_BandgapRefresh7P8125ms

Bandgap refresh every 7.8125ms.

enumerator kVBAT_BandgapRefresh15P625ms

Bandgap refresh every 15.625ms.

enumerator kVBAT_BandgapRefresh31P25ms

Bandgap refresh every 31.25ms.

enumerator kVBAT_BandgapRefresh62P5ms

Bandgap refresh every 62.5ms.

enum _vbat_bandgap_timer_timeout_period

The enumeration of bandgap timer timeout period.

Values:

enumerator kVBAT_BangapTimerTimeout1s

Bandgap timer timerout every 1s.

enumerator kVBAT_BangapTimerTimeout500ms

Bandgap timer timerout every 500ms.

enumerator kVBAT_BangapTimerTimeout250ms

Bandgap timer timerout every 250ms.

enumerator kVBAT_BangapTimerTimeout125ms

Bandgap timer timerout every 125ms.

enumerator kVBAT_BangapTimerTimeout62P5ms

Bandgap timer timerout every 62.5ms.

enumerator kVBAT_BangapTimerTimeout31P25ms

Bandgap timer timerout every 31.25ms.

typedef enum *_vbat_bandgap_refresh_period* vbat_bandgap_refresh_period_t

The enumeration of bandgap refresh period.

typedef enum *_vbat_bandgap_timer_timeout_period* vbat_bandgap_timer_timeout_period_t

The enumeration of bandgap timer timeout period.

typedef struct *_vbat_fro16k_config* vbat_fro16k_config_t

The structure of internal 16kHz free running oscillator attributes.

VBAT_LDO_READY_TIMEOUT

Max loops to wait for LDO ready.

When configuring the LDO, driver will wait for LDO ready. This parameter defines how many loops to check completion before return timeout. If defined as 0, driver will wait forever until completion.

void VBAT_ConfigFRO16k(VBAT_Type *base, const *vbat_fro16k_config_t* *config)

Configure internal 16kHz free running oscillator, including enabel FRO16k, gate FRO16k output.

Parameters

- base – VBAT peripheral base address.
- config – Pointer to *vbat_fro16k_config_t* structure.

static inline void VBAT_EnableFRO16k(VBAT_Type *base, bool enable)

Enable/disable internal 16kHz free running oscillator.

Parameters

- base – VBAT peripheral base address.
- enable – Used to enable/disable 16kHz FRO.
  - **true** Enable internal 16kHz free running oscillator.
  - **false** Disable internal 16kHz free running oscillator.

static inline bool VBAT_CheckFRO16kEnabled(VBAT_Type *base)

Check if internal 16kHz free running oscillator is enabled.

Parameters

- base – VBAT peripheral base address.

Return values

- true – The internal 16kHz Free running oscillator is enabled.
- false – The internal 16kHz Free running oscillator is enabled.

static inline void VBAT_UngateFRO16k(VBAT_Type *base, bool unGateFRO16k)

Ungate/gate FRO 16kHz output clock to other modules.

Parameters

- base – VBAT peripheral base address.
- unGateFRO16k – Used to gate/ungate FRO 16kHz output.
  - **true** FRO 16kHz output clock to other modules is enabled.
  - **false** FRO 16kHz output clock to other modules is disabled.

static inline void VBAT_LockFRO16kSettings(VBAT_Type *base)

Lock settings of internal 16kHz free running oscillator, please note that if locked 16kHz FRO's settings can not be updated until the next POR.

Note: Please note that the operation to ungate/gate FRO 16kHz output clock can not be locked by this function.

Parameters

- base – VBAT peripheral base address.

status_t VBAT_EnableBandgap(VBAT_Type *base, bool enable)

Enable/disable Bandgap.

Note: The FRO16K must be enabled before enabling the bandgap.

Note: This setting can be locked by *VBAT_LockLdoRamSettings()* function.

Parameters

- base – VBAT peripheral base address.
- enable – Used to enable/disable bandgap.
  - **true** Enable the bandgap.
  - **false** Disable the bandgap.

Return values

- kStatus_Success – Success to enable/disable the bandgap.
- kStatus_VBAT_Fro16kNotEnabled – Fail to enable the bandgap due to FRO16k is not enabled previously.

static inline bool VBAT_CheckBandgapEnabled(VBAT_Type *base)

Check if bandgap is enabled.

Parameters

- base – VBAT peripheral base address.

Return values

- true – The bandgap is enabled.
- false – The bandgap is disabled.

static inline void VBAT_EnableBandgapRefreshMode(VBAT_Type *base, bool enableRefreshMode)

Enable/disable bandgap low power refresh mode.

Note: This setting can be locked by *VBAT_LockLdoRamSettings()* function.

Parameters

- base – VBAT peripheral base address.
- enableRefreshMode – Used to enable/disable bandgap low power refresh mode.
  - **true** Enable bandgap low power refresh mode.
  - **false** Disable bandgap low power refresh mode.

status_t VBAT_EnableBackupSRAMRegulator(VBAT_Type *base, bool enable)

Enable/disable Backup RAM Regulator(RAM_LDO).

Note: This setting can be locked by *VBAT_LockLdoRamSettings()* function.

Parameters

- base – VBAT peripheral base address.
- enable – Used to enable/disable RAM_LDO.
  - **true** Enable backup SRAM regulator.
  - **false** Disable backup SRAM regulator.

Return values

- kStatusSuccess – Success to enable/disable backup SRAM regulator.
- kStatus_VBAT_Fro16kNotEnabled – Fail to enable backup SRAM regulator due to FRO16k is not enabled previously.
- kStatus_VBAT_BandgapNotEnabled – Fail to enable backup SRAM regulator due to the bandgap is not enabled previously.
- kStatus_Timeout – Timeout occurs while waiting completion.

static inline void VBAT_LockLdoRamSettings(VBAT_Type *base)

Lock settings of RAM_LDO, please note that if locked then RAM_LDO's settings can not be updated until the next POR.

Parameters

- base – VBAT peripheral base address.

status_t VBAT_SwitchSRAMPowerByVBAT(VBAT_Type *base)

Switch the SRAM to be powered by VBAT in software mode.

Note: This function can be used to switch the SRAM to the VBAT retention supply at any time, but please note that the SRAM must not be accessed during this time and software must manually invoke *VBAT_SwitchSRAMPowerBySocSupply()* before accessing the SRAM again.

Parameters

- base – VBAT peripheral base address.

Return values

- kStatusSuccess – Success to Switch SRAM powered by VBAT.
- kStatus_VBAT_Fro16kNotEnabled – Fail to switch SRAM powered by VBAT due to FRO16K not enabled previously.

static inline void VBAT_SwitchSRAMPowerBySocSupply(VBAT_Type *base)

Switch the RAM to be powered by Soc Supply in software mode.

Parameters

- base – VBAT peripheral base address.

static inline void VBAT_EnableSRAMArrayRetained(VBAT_Type *base, bool enable)

Enable/disable SRAM array remains powered from Soc power, when LDO_RAM is disabled.

Parameters

- base – VBAT peripheral base address.
- enable – Used to enable/disable SRAM array power retained.
  - **true** SRAM array is retained when powered from VDD_CORE.
  - **false** SRAM array is not retained when powered from VDD_CORE.

static inline void VBAT_EnableSRAMIsolation(VBAT_Type *base, bool enable)

Enable/disable SRAM isolation.

Parameters

- base – VBAT peripheral base address.
- enable – Used to enable/disable SRAM violation.
  - **true** SRAM will be isolated.
  - **false** SRAM state follows the SoC power modes.

status_t VBAT_EnableBandgapTimer(VBAT_Type *base, bool enable, uint8_t timerIdMask)

Enable/disable Bandgap timer.

Note: The bandgap timer is available when the bandgap is enabled and are clocked by the FRO16k.

Parameters

- base – VBAT peripheral base address.
- enable – Used to enable/disable bandgap timer.
- timerIdMask – The mask of bandgap timer Id, should be the OR'ed value of *vbat_bandgap_timer_id_t*.

Return values

- kStatus_Success – Success to enable/disable selected bandgap timer.
- kStatus_VBAT_Fro16kNotEnabled – Fail to enable/disable selected bandgap timer due to FRO16k not enabled previously.
- kStatus_VBAT_BandgapNotEnabled – Fail to enable/disable selected bandgap timer due to bandgap not enabled previously.

void VBAT_SetBandgapTimerTimeoutValue(VBAT_Type *base, *vbat_bandgap_timer_timeout_period_t* timeoutPeriod, uint8_t timerIdMask)

Set bandgap timer timeout value.

Parameters

- base – VBAT peripheral base address.
- timeoutPeriod – Bandgap timer timeout value, please refer to *vbat_bandgap_timer_timeout_period_t*.
- timerIdMask – The mask of bandgap timer Id, should be the OR'ed value of *vbat_bandgap_timer_id_t*.

static inline uint32_t VBAT_GetStatusFlags(VBAT_Type *base)

Get VBAT status flags.

Parameters

- base – VBAT peripheral base address.

Returns

The asserted status flags, should be the OR'ed value of *vbat_status_flag_t*.

static inline void VBAT_ClearStatusFlags(VBAT_Type *base, uint32_t mask)

Clear VBAT status flags.

Parameters

- base – VBAT peripheral base address.
- mask – The mask of status flags to be cleared, should be the OR'ed value of *vbat_status_flag_t* except *kVBAT_StatusFlagLdoReady*.

static inline void VBAT_EnableInterrupts(VBAT_Type *base, uint32_t mask)

Enable interrupts for the VBAT module, such as POR detect interrupt, Wakeup Pin interrupt and so on.

Parameters

- base – VBAT peripheral base address.
- mask – The mask of interrupts to be enabled, should be the OR'ed value of *vbat_interrupt_enable_t*.

static inline void VBAT_DisableInterrupts(VBAT_Type *base, uint32_t mask)

Disable interrupts for the VBAT module, such as POR detect interrupt, wakeup pin interrupt and so on.

Parameters

- base – VBAT peripheral base address.
- mask – The mask of interrupts to be disabled, should be the OR'ed value of *vbat_interrupt_enable_t*.

static inline void VBAT_EnableWakeup(VBAT_Type *base, uint32_t mask)

Enable wakeup for the VBAT module, such as POR detect wakeup, wakeup pin wakeup and so on.

Parameters

- base – VBAT peripheral base address.
- mask – The mask of enumerators in *vbat_wakeup_enable_t*.

static inline void VBAT_DisableWakeup(VBAT_Type *base, uint32_t mask)

Disable wakeup for VBAT module, such as POR detect wakeup, wakeup pin wakeup and so on.

Parameters

- base – VBAT peripheral base address.
- mask – The mask of enumerators in *vbat_wakeup_enable_t*.

static inline void VBAT_LockInterruptWakeupSettings(VBAT_Type *base)

Lock VBAT interrupt and wakeup settings, please note that if locked the interrupt and wakeup settings can not be updated until the next POR.

Parameters

- base – VBAT peripheral base address.

FSL_VBAT_DRIVER_VERSION

VBAT driver version 2.1.1.

struct _vbat_fro16k_config

#include <fsl_vbat.h> The structure of internal 16kHz free running oscillator attributes.

Public Members

bool enableFRO16k

Enable/disable internal 16kHz free running oscillator.

bool enableFRO16kOutput

Enable/disable FRO 16k output clock to other modules.

2.70 VREF: Voltage Reference Driver

void VREF_Init(VREF_Type *base, const *vref_config_t* *config)

Enables the clock gate and configures the VREF module according to the configuration structure.

This function must be called before calling all other VREF driver functions, read/write registers, and configurations with user-defined settings. The example below shows how to set up vref_config_t parameters and how to call the VREF_Init function by passing in these parameters.

```
vref_config_t vrefConfig;
VREF_GetDefaultConfig(&vrefConfig);
vrefConfig.bufferMode = kVREF_ModeHighPowerBuffer;
VREF_Init(VREF, &vrefConfig);
```

Parameters

- base – VREF peripheral address.
- config – Pointer to the configuration structure.

void VREF_Deinit(VREF_Type *base)

Stops and disables the clock for the VREF module.

This function should be called to shut down the module. This is an example.

```
vref_config_t vrefUserConfig;
VREF_GetDefaultConfig(&vrefUserConfig);
VREF_Init(VREF, &vrefUserConfig);
...
VREF_Deinit(VREF);
```

Parameters

- base – VREF peripheral address.

void VREF_GetDefaultConfig(*vref_config_t* *config)

Initializes the VREF configuration structure.

This function initializes the VREF configuration structure to default values. This is an example.

```
config->bufferMode = kVREF_ModeHighPowerBuffer;
config->enableInternalVoltageRegulator = true;
config->enableChopOscillator           = true;
config->enableHCBandgap                = true;
config->enableCurvatureCompensation    = true;
config->enableLowPowerBuff             = true;
```

Parameters

- config – Pointer to the initialization structure.

void VREF_SetVrefTrimVal(VREF_Type *base, uint8_t trimValue)

Sets a TRIM value for the accurate 1.0V bandgap output.

This function sets a TRIM value for the reference voltage. It will trim the accurate 1.0V bandgap by 0.5mV each step.

Parameters

- base – VREF peripheral address.
- trimValue – Value of the trim register to set the output reference voltage (maximum 0x3F (6-bit)).

void VREF_SetTrim21Val(VREF_Type *base, uint8_t trim21Value)

Sets a TRIM value for the accurate buffered VREF output.

This function sets a TRIM value for the reference voltage. If buffer mode be set to other values (Buf21 enabled), it will trim the VREF_OUT by 0.1V each step from 1.0V to 2.1V.

Note: When Buf21 is enabled, the value of UTRIM[TRIM2V1] should be ranged from 0b0000 to 0b1011 in order to trim the output voltage from 1.0V to 2.1V, other values will make the VREF_OUT to default value, 1.0V.

Parameters

- base – VREF peripheral address.
- trim21Value – Value of the trim register to set the output reference voltage (maximum 0xF (4-bit)).

uint8_t VREF_GetVrefTrimVal(VREF_Type *base)

Reads the trim value.

This function gets the TRIM value from the UTRIM register. It reads UTRIM[VREFTRIM] (13:8)

Parameters

- base – VREF peripheral address.

Returns

6-bit value of trim setting.

uint8_t VREF_GetTrim21Val(VREF_Type *base)

Reads the VREF 2.1V trim value.

This function gets the TRIM value from the UTRIM register. It reads UTRIM[TRIM2V1] (3:0),

Parameters

- base – VREF peripheral address.

Returns

4-bit value of trim setting.

static inline void VREF_SetAdcNegChannelRef(VREF_Type *base, bool enable)

Decides whether to enable the ADC negative channel to receive the reference voltage.

Parameters

- base – VREF peripheral address.

- enable – Enable/disable ADC negative channel reference voltage.
  - **true** Enable ADC negative channel reference voltage.
  - **false** Disable ADC negative channel reference voltage.

static inline void VREF_SetAdcPosChannelRef(VREF_Type *base, bool enable)

Decides whether to enable the ADC positive channel to receive the reference voltage.

Parameters

- base – VREF peripheral address.
- enable – Enable/disable ADC positive channel reference voltage.
  - **true** Enable ADC positive channel reference voltage.
  - **false** Disable ADC positive channel reference voltage.

FSL_VREF_DRIVER_VERSION

Version 2.5.0.

enum _vref_buffer_mode

VREF buffer modes.

Values:

enumerator kVREF_ModeBandgapOnly

Bandgap enabled/standby.

enumerator kVREF_ModeLowPowerBuffer

Low-power buffer mode enabled

enumerator kVREF_ModeHighPowerBuffer

High-power buffer mode enabled

typedef enum *_vref_buffer_mode* vref_buffer_mode_t

VREF buffer modes.

typedef struct *_vref_config* vref_config_t

The description structure for the VREF module.

struct _vref_config

#include <fsl_vref.h> The description structure for the VREF module.

Public Members

vref_buffer_mode_t bufferMode

Buffer mode selection

bool enableInternalVoltageRegulator

Provide additional supply noise rejection.

bool enableChopOscillator

Enable Chop oscillator.

bool enableHCBandgap
Enable High-Accurate bandgap.

bool enableCurvatureCompensation
Enable second order curvature compensation.

bool enableLowPowerBuff
Provides bias current for other peripherals.

2.71 WDOG32: 32-bit Watchdog Timer

void WDOG32_GetDefaultConfig(*wdog32_config_t* *config)
Initializes the WDOG32 configuration structure.

This function initializes the WDOG32 configuration structure to default values. The default values are:

```
wdog32Config->enableWdog32 = true;
wdog32Config->clockSource = kWDOG32_ClockSource1;
wdog32Config->prescaler = kWDOG32_ClockPrescalerDivide1;
wdog32Config->workMode.enableWait = true;
wdog32Config->workMode.enableStop = false;
wdog32Config->workMode.enableDebug = false;
wdog32Config->testMode = kWDOG32_TestModeDisabled;
wdog32Config->enableUpdate = true;
wdog32Config->enableInterrupt = false;
wdog32Config->enableWindowMode = false;
wdog32Config->windowValue = 0U;
wdog32Config->timeoutValue = 0xFFFFU;
```

See also:

wdog32_config_t

Parameters

- config – Pointer to the WDOG32 configuration structure.

status_t WDOG32_Init(WDOG_Type *base, const *wdog32_config_t* *config)
Initializes the WDOG32 module.

This function initializes the WDOG32. To reconfigure the WDOG32 without forcing a reset first, enableUpdate must be set to true in the configuration.

Example:

```
wdog32_config_t config;
WDOG32_GetDefaultConfig(&config);
config.timeoutValue = 0x7ffU;
config.enableUpdate = true;
WDOG32_Init(wdog_base,&config);
```

Note: If there is errata ERR010536 (FSL_FEATURE_WDOG_HAS_ERRATA_010536 defined as 1), then after calling this function, user need delay at least 4 LPO clock cycles before accessing other WDOG32 registers.

Parameters

- base – WDOG32 peripheral base address.
- config – The configuration of the WDOG32.

Return values

- kStatus_Success – The initialization was successful
- kStatus_Timeout – The initialization timed out

status_t WDOG32_Deinit(WDOG_Type *base)

De-initializes the WDOG32 module.

This function shuts down the WDOG32. Ensure that the WDOG_CS.UPDATE is 1, which means that the register update is enabled.

Parameters

- base – WDOG32 peripheral base address.

Return values

- kStatus_Success – The de-initialization was successful
- kStatus_Timeout – The de-initialization timed out

status_t WDOG32_Unlock(WDOG_Type *base)

Unlocks the WDOG32 register written.

This function unlocks the WDOG32 register written.

Before starting the unlock sequence and following the configuration, disable the global interrupts. Otherwise, an interrupt could effectively invalidate the unlock sequence and the WCT may expire. After the configuration finishes, re-enable the global interrupts.

Parameters

- base – WDOG32 peripheral base address

Return values

- kStatus_Success – The unlock sequence was successful
- kStatus_Timeout – The unlock sequence timed out

void WDOG32_Enable(WDOG_Type *base)

Enables the WDOG32 module.

This function writes a value into the WDOG_CS register to enable the WDOG32. The WDOG_CS register is a write-once register. Please check the enableUpdate is set to true for calling *WDOG32_Init* to do wdog initialize. Before call the re-configuration APIs, ensure that the WCT window is still open and this register has not been written in this WCT while the function is called.

Parameters

- base – WDOG32 peripheral base address.

void WDOG32_Disable(WDOG_Type *base)

Disables the WDOG32 module.

This function writes a value into the WDOG_CS register to disable the WDOG32. The WDOG_CS register is a write-once register. Please check the enableUpdate is set to true for calling *WDOG32_Init* to do wdog initialize. Before call the re-configuration APIs, ensure that the WCT window is still open and this register has not been written in this WCT while the function is called.

Parameters

- base – WDOG32 peripheral base address

void WDOG32_EnableInterrupts(WDOG_Type *base, uint32_t mask)

Enables the WDOG32 interrupt.

This function writes a value into the WDOG_CS register to enable the WDOG32 interrupt. The WDOG_CS register is a write-once register. Please check the enableUpdate is set to true for calling *WDOG32_Init* to do wdog initialize. Before call the re-configuration APIs, ensure that the WCT window is still open and this register has not been written in this WCT while the function is called.

Parameters

- base – WDOG32 peripheral base address.
- mask – The interrupts to enable. The parameter can be a combination of the following source if defined:
  - kWDOG32_InterruptEnable

void WDOG32_DisableInterrupts(WDOG_Type *base, uint32_t mask)

Disables the WDOG32 interrupt.

This function writes a value into the WDOG_CS register to disable the WDOG32 interrupt. The WDOG_CS register is a write-once register. Please check the enableUpdate is set to true for calling *WDOG32_Init* to do wdog initialize. Before call the re-configuration APIs, ensure that the WCT window is still open and this register has not been written in this WCT while the function is called.

Parameters

- base – WDOG32 peripheral base address.
- mask – The interrupts to disabled. The parameter can be a combination of the following source if defined:
  - kWDOG32_InterruptEnable

static inline uint32_t WDOG32_GetStatusFlags(WDOG_Type *base)

Gets the WDOG32 all status flags.

This function gets all status flags.

Example to get the running flag:

```
uint32_t status;
status = WDOG32_GetStatusFlags(wdog_base) & kWDOG32_RunningFlag;
```

See also:

_wdog32_status_flags_t

- true: related status flag has been set.
- false: related status flag is not set.

Parameters

- base – WDOG32 peripheral base address

Returns

State of the status flag: asserted (true) or not-asserted (false).

void WDOG32_ClearStatusFlags(WDOG_Type *base, uint32_t mask)

Clears the WDOG32 flag.

This function clears the WDOG32 status flag.

Example to clear an interrupt flag:

```
WDOG32_ClearStatusFlags(wdog_base,kWDOG32_InterruptFlag);
```

Parameters

- base – WDOG32 peripheral base address.
- mask – The status flags to clear. The parameter can be any combination of the following values:
  - kWDOG32_InterruptFlag

void WDOG32_SetTimeoutValue(WDOG_Type *base, uint16_t timeoutCount)

Sets the WDOG32 timeout value.

This function writes a timeout value into the WDOG_TOVAL register. The WDOG_TOVAL register is a write-once register. To ensure the reconfiguration fits the timing of WCT, unlock function will be called inline.

Parameters

- base – WDOG32 peripheral base address
- timeoutCount – WDOG32 timeout value, count of WDOG32 clock ticks.

void WDOG32_SetWindowValue(WDOG_Type *base, uint16_t windowValue)

Sets the WDOG32 window value.

This function writes a window value into the WDOG_WIN register. The WDOG_WIN register is a write-once register. Please check the enableUpdate is set to true for calling *WDOG32_Init* to do wdog initialize. Before call the re-configuration APIs, ensure that the WCT window is still open and this register has not been written in this WCT while the function is called.

Parameters

- base – WDOG32 peripheral base address.
- windowValue – WDOG32 window value.

static inline void WDOG32_Refresh(WDOG_Type *base)

Refreshes the WDOG32 timer.

This function feeds the WDOG32. This function should be called before the Watchdog timer is in timeout. Otherwise, a reset is asserted.

Parameters

- base – WDOG32 peripheral base address

static inline uint16_t WDOG32_GetCounterValue(WDOG_Type *base)

Gets the WDOG32 counter value.

This function gets the WDOG32 counter value.

Parameters

- base – WDOG32 peripheral base address.

Returns

Current WDOG32 counter value.

WDOG_FIRST_WORD_OF_UNLOCK

First word of unlock sequence

WDOG_SECOND_WORD_OF_UNLOCK

Second word of unlock sequence

WDOG_FIRST_WORD_OF_REFRESH

First word of refresh sequence

WDOG_SECOND_WORD_OF_REFRESH

Second word of refresh sequence

FSL_WDOG32_DRIVER_VERSION

WDOG32 driver version.

enum _wdog32_clock_source

Max loops to wait for WDOG32 unlock sequence complete.

This is the maximum number of loops to wait for the wdog32 unlock sequence to complete. If set to 0, it will wait indefinitely until the unlock sequence is complete.

Max loops to wait for WDOG32 reconfiguration complete.

This is the maximum number of loops to wait for the wdog32 reconfiguration to complete. If set to 0, it will wait indefinitely until the reconfiguration is complete.

Describes WDOG32 clock source.

Values:

enumerator kWDOG32_ClockSource0

Clock source 0

enumerator kWDOG32_ClockSource1

Clock source 1

enumerator kWDOG32_ClockSource2

Clock source 2

enumerator kWDOG32_ClockSource3

Clock source 3

enum _wdog32_clock_prescaler

Describes the selection of the clock prescaler.

Values:

enumerator kWDOG32_ClockPrescalerDivide1

Divided by 1

enumerator kWDOG32_ClockPrescalerDivide256

Divided by 256

enum _wdog32_test_mode

Describes WDOG32 test mode.

Values:

enumerator kWDOG32_TestModeDisabled

Test Mode disabled

enumerator kWDOG32_UserModeEnabled
User Mode enabled

enumerator kWDOG32_LowByteTest
Test Mode enabled, only low byte is used

enumerator kWDOG32_HighByteTest
Test Mode enabled, only high byte is used

enum _wdog32_interrupt_enable_t
WDOG32 interrupt configuration structure.

This structure contains the settings for all of the WDOG32 interrupt configurations.

Values:

enumerator kWDOG32_InterruptEnable
Interrupt is generated before forcing a reset

enum _wdog32_status_flags_t
WDOG32 status flags.

This structure contains the WDOG32 status flags for use in the WDOG32 functions.

Values:

enumerator kWDOG32_RunningFlag
Running flag, set when WDOG32 is enabled

enumerator kWDOG32_InterruptFlag
Interrupt flag, set when interrupt occurs

typedef enum *_wdog32_clock_source* wdog32_clock_source_t
Max loops to wait for WDOG32 unlock sequence complete.

This is the maximum number of loops to wait for the wdog32 unlock sequence to complete. If set to 0, it will wait indefinitely until the unlock sequence is complete.

Max loops to wait for WDOG32 reconfiguration complete.

This is the maximum number of loops to wait for the wdog32 reconfiguration to complete. If set to 0, it will wait indefinitely until the reconfiguration is complete.

Describes WDOG32 clock source.

typedef enum *_wdog32_clock_prescaler* wdog32_clock_prescaler_t
Describes the selection of the clock prescaler.

typedef struct *_wdog32_work_mode* wdog32_work_mode_t
Defines WDOG32 work mode.

typedef enum *_wdog32_test_mode* wdog32_test_mode_t
Describes WDOG32 test mode.

typedef struct *_wdog32_config* wdog32_config_t

Describes WDOG32 configuration structure.

struct _wdog32_work_mode

#include <fsl_wdog32.h> Defines WDOG32 work mode.

Public Members

bool enableWait

Enables or disables WDOG32 in wait mode

bool enableStop

Enables or disables WDOG32 in stop mode

bool enableDebug

Enables or disables WDOG32 in debug mode

struct _wdog32_config

#include <fsl_wdog32.h> Describes WDOG32 configuration structure.

Public Members

bool enableWdog32

Enables or disables WDOG32

wdog32_clock_source_t clockSource

Clock source select

wdog32_clock_prescaler_t prescaler

Clock prescaler value

wdog32_work_mode_t workMode

Configures WDOG32 work mode in debug stop and wait mode

wdog32_test_mode_t testMode

Configures WDOG32 test mode

bool enableUpdate

Update write-once register enable

bool enableInterrupt

Enables or disables WDOG32 interrupt

bool enableWindowMode

Enables or disables WDOG32 window mode

uint16_t windowValue
Window value

uint16_t timeoutValue
Timeout value

2.72 WUU: Wakeup Unit driver

void WUU_SetExternalWakeUpPinsConfig(WUU_Type *base, uint8_t pinIndex, const *wuu_external_wakeup_pin_config_t* *config)

Enables and Configs External WakeUp Pins.

This function enables/disables the external pin as wakeup input. What's more this function configs pins options, including edge detection wakeup event and operate mode.

Parameters

- base – MUU peripheral base address.
- pinIndex – The index of the external input pin. See Reference Manual for the details.
- config – Pointer to wuu_external_wakeup_pin_config_t structure.

void WUU_ClearExternalWakeupPinsConfig(WUU_Type *base, uint8_t pinIndex)

Disable and clear external wakeup pin settings.

Parameters

- base – MUU peripheral base address.
- pinIndex – The index of the external input pin.

static inline uint32_t WUU_GetExternalWakeUpPinsFlag(WUU_Type *base)

Gets External Wakeup pin flags.

This function return the external wakeup pin flags.

Parameters

- base – WUU peripheral base address.

Returns

Wakeup flags for all external wakeup pins.

static inline void WUU_ClearExternalWakeUpPinsFlag(WUU_Type *base, uint32_t mask)

Clears External WakeUp Pin flags.

This function clears external wakeup pins flags based on the mask.

Parameters

- base – WUU peripheral base address.
- mask – The mask of Wakeup pin index to be cleared.

void WUU_SetInternalWakeUpModulesConfig(WUU_Type *base, uint8_t moduleIndex, *wuu_internal_wakeup_module_event_t* event)

Config Internal modules' event as the wake up soures.

This function configs the internal modules event as the wake up sources.

Parameters

- base – WUU peripheral base address.

- moduleIndex – The selected internal module. See the Reference Manual for the details.
- event – Select interrupt or DMA/Trigger of the internal module as the wake up source.

void WUU_ClearInternalWakeUpModulesConfig(WUU_Type *base, uint8_t moduleIndex, *wuu_internal_wakeup_module_event_t* event)

Disable an on-chip internal modules' event as the wakeup sources.

Parameters

- base – WUU peripheral base address.
- moduleIndex – The selected internal module. See the Reference Manual for the details.
- event – The event(interrupt or DMA/trigger) of the internal module to disable.

static inline uint32_t WUU_GetModuleInterruptFlag(WUU_Type *base)

Get wakeup flags for internal wakeup modules.

Parameters

- base – WUU peripheral base address.

Returns

Wakeup flags for all internal wakeup modules.

static inline bool WUU_GetInternalWakeupModuleFlag(WUU_Type *base, uint32_t moduleIndex)

Gets the internal module wakeup source flag.

This function checks the flag to detect whether the system is woken up by specific on-chip module interrupt.

Parameters

- base – WWU peripheral base address.
- moduleIndex – A module index, which starts from 0.

Returns

True if the specific pin is a wake up source.

void WUU_SetPinFilterConfig(WUU_Type *base, uint8_t filterIndex, const *wuu_pin_filter_config_t* *config)

Configs and Enables Pin filters.

This function configs Pin filter, including pin select, filer operate mode filer wakeup event and filter edge detection.

Parameters

- base – WUU peripheral base address.
- filterIndex – The index of the pin filer.
- config – Pointer to wuu_pin_filter_config_t structure.

bool WUU_GetPinFilterFlag(WUU_Type *base, uint8_t filterIndex)

Gets the pin filter configuration.

This function gets the pin filter flag.

Parameters

- base – WUU peripheral base address.
- filterIndex – A pin filter index, which starts from 1.

Returns
True if the flag is a source of the existing low-leakage power mode.

void WUU_ClearPinFilterFlag(WUU_Type *base, uint8_t filterIndex)

Clears the pin filter configuration.

This function clears the pin filter flag.

Parameters

- base – WUU peripheral base address.
- filterIndex – A pin filter index to clear the flag, starting from 1.

bool WUU_GetExternalWakeupPinFlag(WUU_Type *base, uint32_t pinIndex)

brief Gets the external wakeup source flag.

This function checks the external pin flag to detect whether the MCU is woken up by the specific pin.

param base WUU peripheral base address. param pinIndex A pin index, which starts from 0. return True if the specific pin is a wakeup source.

void WUU_ClearExternalWakeupPinFlag(WUU_Type *base, uint32_t pinIndex)

brief Clears the external wakeup source flag.

This function clears the external wakeup source flag for a specific pin.

param base WUU peripheral base address. param pinIndex A pin index, which starts from 0.

FSL_WUU_DRIVER_VERSION

Defines WUU driver version 2.4.2.

enum _wuu_external_pin_edge_detection

External WakeUp pin edge detection enumeration.

Values:

enumerator kWUU_ExternalPinDisable

External input Pin disabled as wake up input.

enumerator kWUU_ExternalPinRisingEdge

External input Pin enabled with the rising edge detection.

enumerator kWUU_ExternalPinFallingEdge

External input Pin enabled with the falling edge detection.

enumerator kWUU_ExternalPinAnyEdge

External input Pin enabled with any change detection.

enum _wuu_external_wakeup_pin_event

External input wake up pin event enumeration.

Values:

enumerator kWUU_ExternalPinInterrupt

External input Pin configured as interrupt.

enumerator kWUU_ExternalPinDMARequest

External input Pin configured as DMA request.

enumerator kWUU_ExternalPinTriggerEvent

External input Pin configured as Trigger event.

enum _wuu_external_wakeup_pin_mode

External input wake up pin mode enumeration.

Values:

enumerator kWUU_ExternalPinActiveDSPD

External input Pin is active only during Deep Sleep/Power Down Mode. NOTE: This enumerations has been deprecated, please switch to kWUU_ExternalPinActiveLowLeakage.

enumerator kWUU_ExternalPinActiveLowLeakageMode

External input Pin is active only during low-leakage power modes.

enumerator kWUU_ExternalPinActiveAlways

External input Pin is active during all power modes.

enum _wuu_internal_wakeup_module_event

Internal module wake up event enumeration.

Values:

enumerator kWUU_InternalModuleInterrupt

Internal modules' interrupt as a wakeup source.

enumerator kWUU_InternalModuleDMATrigger

Internal modules' DMA/Trigger as a wakeup source.

enum _wuu_filter_edge

Pin filter edge enumeration.

Values:

enumerator kWUU_FilterDisabled

Filter disabled.

enumerator kWUU_FilterPosedgeEnable

Filter posedge detect enabled.

enumerator kWUU_FilterNegedgeEnable

Filter negedge detect enabled.

enumerator kWUU_FilterAnyEdge

Filter any edge detect enabled.

enum _wuu_filter_event

Pin Filter event enumeration.

Values:

enumerator kWUU_FilterInterrupt

Filter output configured as interrupt.

enumerator kWUU_FilterDMARequest

Filter output configured as DMA request.

enumerator kWUU_FilterTriggerEvent

Filter output configured as Trigger event.

enum _wuu_filter_mode

Pin filter mode enumeration.

Values:

enumerator kWUU_FilterActiveDSPD

External input pin filter is active only during Deep Sleep/Power Down Mode. NOTE: This enumerations has been deprecated, please switch to kWUU_FilterActiveLowLeakage.

enumerator kWUU_FilterActiveLowLeakageMode

External input pin filter is active only during low-leakage power modes.

enumerator kWUU_FilterActiveAlways

External input Pin filter is active during all power modes.

typedef enum *_wuu_external_pin_edge_detection* wuu_external_pin_edge_detection_t

External WakeUp pin edge detection enumeration.

typedef enum *_wuu_external_wakeup_pin_event* wuu_external_wakeup_pin_event_t

External input wake up pin event enumeration.

typedef enum *_wuu_external_wakeup_pin_mode* wuu_external_wakeup_pin_mode_t

External input wake up pin mode enumeration.

typedef enum *_wuu_internal_wakeup_module_event* wuu_internal_wakeup_module_event_t

Internal module wake up event enumeration.

typedef enum *_wuu_filter_edge* wuu_filter_edge_t

Pin filter edge enumeration.

typedef enum *_wuu_filter_event* wuu_filter_event_t

Pin Filter event enumeration.

typedef enum *_wuu_filter_mode* wuu_filter_mode_t

Pin filter mode enumeration.

typedef struct *_wuu_external_wakeup_pin_config* wuu_external_wakeup_pin_config_t

External WakeUp pin configuration.

typedef struct *_wuu_pin_filter_config* wuu_pin_filter_config_t

Pin Filter configuration.

struct _wuu_external_wakeup_pin_config

#include <fsl_wuu.h> External WakeUp pin configuration.

Public Members

wuu_external_pin_edge_detection_t edge

External Input pin edge detection.

wuu_external_wakeup_pin_event_t event

External Input wakeup Pin event

wuu_external_wakeup_pin_mode_t mode

External Input wakeup Pin operate mode.

struct _wuu_pin_filter_config

#include <fsl_wuu.h> Pin Filter configuration.

Public Members

uint32_t pinIndex

The index of wakeup pin to be muxxed into filter.

wuu_filter_edge_t edge

The edge of the pin digital filter.

wuu_filter_event_t event

The event of the filter output.

wuu_filter_mode_t mode

The mode of the filter operate.

Chapter 3

Middleware

3.1 Motor Control

3.1.1 FreeMASTER

Communication Driver User Guide

Introduction

What is FreeMASTER? FreeMASTER is a PC-based application developed by NXP for NXP customers. It is a versatile tool usable as a real-time monitor, visualization tool, and a graphical control panel of embedded applications based on the NXP processing units.

This document describes the embedded-side software driver which implements an interface between the application and the host PC. The interface covers the following communication:

- **Serial** UART communication either over plain RS232 interface or more typically over a USB-to-Serial either external or built in a debugger probe.
- **USB** direct connection to target microcontroller
- **CAN bus**
- **TCP/IP network** wired or WiFi
- **Segger J-Link RTT**
- **JTAG** debug port communication
- ...and all of the above also using a **Zephyr** generic drivers.

The driver also supports so-called "packet-driven BDM" interface which enables a protocol-based communication over a debugging port. The BDM stands for Background Debugging Module and its physical implementation is different on each platform. Some platforms leverage a semi-standard JTAG interface, other platforms provide a custom implementation called BDM. Regardless of the name, this debugging interface enables non-intrusive access to the memory space while the target CPU is running. For basic memory read and write operations, there is no communication driver required on the target when communicating with the host PC. Use this driver to get more advanced FreeMASTER protocol features over the BDM interface. The driver must be configured for the packet-driven BDM mode, in which the host PC uses the debugging interface to write serial command frames directly to the target memory buffer. The same method is then used to read response frames from that memory buffer.

Similar to "packet-driven BDM", the FreeMASTER also supports a communication over [J-Link RTT]((https://www.segger.com/products/debug-probes/j-link/technology/about-real-time-transfer/) interface defined by SEGGER Microcontroller GmbH for ARM CortexM-based microcontrollers. This method also uses JTAG physical interface and enables high-speed real time communication to run over the same channel as used for application debugging.

Driver version 3 This document describes version 3 of the FreeMASTER Communication Driver. This version features the implementation of the new Serial Protocol, which significantly extends the features and security of its predecessor. The new protocol internal number is v4 and its specification is available in the documentation accompanying the driver code.

Driver V3 is deployed to modern 32-bit MCU platforms first, so the portfolio of supported platforms is smaller than for the previous V2 versions. It is recommended to keep using the V2 driver for legacy platforms, such as S08, S12, ColdFire, or Power Architecture. Reach out to FreeMASTER community or to the local NXP representative with requests for more information or to port the V3 driver to legacy MCU devices.

Thanks to a layered approach, the new driver simplifies the porting of the driver to new UART, CAN or networking communication interfaces significantly. Users are encouraged to port the driver to more NXP MCU platforms and contribute the code back to NXP for integration into future releases. Existing code and low-level driver layers may be used as an example when porting to new targets.

Note: Using the FreeMASTER tool and FreeMASTER Communication Driver is only allowed in systems based on NXP microcontroller or microprocessor unit. Use with non-NXP MCU platforms is **not permitted** by the license terms.

Target platforms The driver implementation uses the following abstraction mechanisms which simplify driver porting and supporting new communication modules:

- **General CPU Platform** (see source code in the `src/platforms` directory). The code in this layer is only specific to native data type sizes and CPU architectures (for example; alignment-aware memory copy routines). This driver version brings two generic implementations of 32-bit platforms supporting both little-endian and big-endian architectures. There are also implementations customized for the 56F800E family of digital signal controllers and S12Z MCUs. **Zephyr** is treated as a specific CPU platform as it brings unified user configuration (Kconfig) and generic hardware device drivers. With Zephyr, the transport layer and low-level communication layers described below are configured automatically using Kconfig and Device Tree technologies.
- **Transport Communication Layer** - The Serial, CAN, Networking, PD-BDM, and other methods of transport logic are implemented as a driver layer called FMSTR_TRANSPORT with a uniform API. A support of the Network transport also extends single-client modes of operation which are native for Serial, USB and CAN by a concept of multiple client sessions.
- **Low-level Communication Driver** - Each type of transport further defines a low-level API used to access the physical communication module. For example, the Serial transport defines a character-oriented API implemented by different serial communication modules like UART, LPUART, USART, and also USB-CDC. Similarly, the CAN transport defines a message-oriented API implemented by the FlexCAN or MCAN modules. Moreover, there are multiple different implementations for the same kind of communication peripherals. The difference between the implementation is in the way the low-level hardware registers are accessed. The *mcuxsdk* folder contains implementations which use MCUXpresso SDK drivers. These drivers should be used in applications based on the NXP MCUXpresso SDK. The "ampsdk" drivers target automotive-specific MCUs and their respective SDKs. The "dreg" implementations use a plain C-language access to hardware register addresses which makes it a universal and the most portable solution. In this case, users are encouraged to add more drivers for other communication modules or other respective SDKs and contribute the code back to NXP for integration.

The low-level drivers defined for the Networking transport enable datagram-oriented UDP and stream TCP communication. This implementation is demonstrated using the lwIP software stack but shall be portable to other TCP/IP stacks. It may sound surprisingly, but also the Segger J-Link RTT communication driver is linked to the Networking transport (RTT is stream oriented communication handled similarly to TCP).

Replacing existing drivers For all supported platforms, the driver described in this document replaces the V2 implementation and also older driver implementations that were available separately for individual platforms (PC Master SCI drivers).

Clocks, pins, and peripheral initialization The FreeMASTER communication driver is only responsible for runtime processing of the communication and must be integrated with an user application code to function properly. The user application code is responsible for general initialization of clock sources, pin multiplexers, and peripheral registers related to the communication speed. Such initialization should be done before calling the `FMSTR_Init` function.

It is recommended to develop the user application using one of the Software Development Kits (SDKs) available from third parties or directly from NXP, such as MCUXpresso SDK, MCUXpresso IDE, and related tools. This approach simplifies the general configuration process significantly.

MCUXpresso SDK The MCUXpresso SDK is a software package provided by NXP which contains the device initialization code, linker files, and software drivers with example applications for the NXP family of MCUs. The MCUXpresso Config Tools may be used to generate the clock-setup and pin-multiplexer setup code suitable for the selected processor.

The MCUXpresso SDK also contains this FreeMASTER communication driver as a "middleware" component which may be downloaded along with the example applications from https://mcuxpresso.nxp.com/en/welcome.

MCUXpresso SDK on GitHub

The FreeMASTER communication driver is also released as one of the middleware components of the MCUXpresso SDK on the GitHub. This release enables direct integration of the FreeMASTER source code Git repository into a target applications including Zephyr applications.

Related links:

- The official FreeMASTER middleware repository.
- Online version of this document

FreeMASTER in Zephyr

The FreeMASTER middleware repository can be used with MCUXpresso SDK as well as a Zephyr module. Zephyr-specific samples which include examples of Kconfig and Device Tree configurations for Serial, USB and Network communications are available in separate repository. West manifest in this sample repository fetches the full Zephyr package including the FreeMASTER middleware repository used as a Zephyr module.

Example applications **MCUX SDK Example applications**

There are several example applications available for each supported MCU platform.

- **fmstr_uart** demonstrates a plain serial transmission, typically connecting to a computer's physical or virtual COM port. The typical transmission speed is 115200 bps.

- **fmstr_can** demonstrates CAN bus communication. This requires a suitable CAN interface connected to the computer and interconnected with the target MCU using a properly terminated CAN bus. The typical transmission speed is 500 kbps. A FreeMASTER-over-CAN communication plug-in must be used.
- **fmstr_usb_cdc** uses an on-chip USB controller to implement a CDC communication class. It is connected directly to a computer's USB port and creates a virtual COM port device. The typical transmission speed is above 1 Mbps.
- **fmstr_net** demonstrates the Network communication over UDP or TCP protocol. Existing examples use lwIP stack to implement the communication, but in general, it shall be possible to use any other TCP/IP stack to achieve the same functionality.
- **fmstr_wifi** is the fmstr_net application modified to use a WiFi network interface instead of a wired Ethernet connection.
- **fmstr_rtt** demonstrates the communication over SEGGER J-Link RTT interface. Both fmstr_net and fmstr_rtt examples require the FreeMASTER TCP/UDP communication plug-in to be used on the PC host side.
- **fmstr_eonce** uses the real-time data unit on the JTAG EOnCE module of the 56F800E family to implement pseudo-serial communication over the JTAG port. The typical transmission speed is around 10 kbps. This communication requires FreeMASTER JTAG/EOnCE communication plug-in.
- **fmstr_pdbdm** uses JTAG or BDM debugging interface to access the target RAM directly while the CPU is running. Note that such approach can be used with any MCU application, even without any special driver code. The computer reads from and writes into the RAM directly without CPU intervention. The Packet-Driven BDM (PD-BDM) communication uses the same memory access to exchange command and response frames. With PD-BDM, the FreeMASTER tool is able to go beyond basic memory read/write operations and accesses also advanced features like Recorder, TSA, or Pipes. The typical transmission speed is around 10 kbps. A PD-BDM communication plug-in must be used in FreeMASTER and configured properly for the selected debugging interface. Note that this communication cannot be used while a debugging interface is used by a debugger session.
- **fmstr_any** is a special example application which demonstrates how the NXP MCUXpresso Config Tools can be used to configure pins, clocks, peripherals, interrupts, and even the FreeMASTER "middleware" driver features in a graphical and user friendly way. The user can switch between the Serial, CAN, and other ways of communication and generate the required initialization code automatically.

Zephyr sample spplications

Zephyr sample applications demonstrate Kconfig and Device Tree configuration which configure the FreeMASTER middleware module for a selected communication option (Serial, CAN, Network or RTT).

Refer to *readme.md* files in each sample directory for description of configuration options required to implement FreeMASTER connectivity.

Description

This section shows how to add the FreeMASTER Communication Driver into application and how to configure the connection to the FreeMASTER visualization tool.

Features The FreeMASTER driver implements the FreeMASTER protocol V4 and provides the following features which may be accessed using the FreeMASTER visualization tool:

- Read/write access to any memory location on the target.

- Optional password protection of the read, read/write, and read/write/flash access levels.
- Atomic bit manipulation on the target memory (bit-wise write access).
- Optimal size-aligned access to memory which is also suitable to access the peripheral register space.
- Oscilloscope access—real-time access to target variables. The sample rate may be limited by the communication speed.
- Recorder— access to the fast transient recorder running on the board as a part of the FreeMASTER driver. The sample rate is only limited by the MCU CPU speed. The length of the data recorded depends on the amount of available memory.
- Multiple instances of Oscilloscopes and Recorders without the limitation of maximum number of variables.
- Application commands—high-level message delivery from the PC to the application.
- TSA tables—describing the data types, variables, files, or hyperlinks exported by the target application. The TSA newly supports also non-memory mapped resources like external EEPROM or SD Card files.
- Pipes—enabling the buffered stream-oriented data exchange for a general-purpose terminal-like communication, diagnostic data streaming, or other data exchange.

The FreeMASTER driver features:

- Full FreeMASTER protocol V4 implementation with a new V4 style of CRC used.
- Layered approach supporting Serial, CAN, Network, PD-BDM, and other transports.
- Layered low-level Serial transport driver architecture enabling to select UART, LPUART, USART, and other physical implementations of serial interfaces, including USB-CDC.
- Layered low-level CAN transport driver architecture enabling to select FlexCAN, msCAN, MCAN, and other physical implementations of the CAN interface.
- Layered low-level Networking transport enabling to select TCP, UDP or J-Link RTT communication.
- TSA support to write-protect memory regions or individual variables and to deny the access to the unsafe memory.
- The pipe callback handlers are invoked whenever new data is available for reading from the pipe.
- Two Serial Single-Wire modes of operation are enabled. The “external” mode has the RX and TX shorted on-board. The “true” single-wire mode interconnects internally when the MCU or UART modules support it.

The following sections briefly describe all FreeMASTER features implemented by the driver. See the PC-based FreeMASTER User Manual for more details on how to use the features to monitor, tune, or control an embedded application.

Board Detection

The FreeMASTER protocol V4 defines the standard set of configuration values which the host PC tool reads to identify the target and to access other target resources properly. The configuration includes the following parameters:

- Version of the driver and the version of the protocol implemented.
- MTU as the Maximum size of the Transmission Unit (for example; communication buffer size).
- Application name, description, and version strings.
- Application build date and time as a string.

- Target processor byte ordering (little/big endian).
- Protection level that requires password authentication.
- Number of the Recorder and Oscilloscope instances.
- RAM Base Address for optimized memory access commands.

Memory Read

This basic feature enables the host PC to read any data memory location by specifying the address and size of the required memory area. The device response frame must be shorter than the MTU to fit into the outgoing communication buffer. To read a device memory of any size, the host uses the information retrieved during the Board Detection and splits the large-block request to multiple partial requests.

The driver uses size-aligned operations to read the target memory (for example; uses proper read-word instruction when an address is aligned to 4 bytes).

Memory Write

Similarly to the Memory Read operation, the Memory Write feature enables to write to any RAM memory location on the target device. A single write command frame must be shorter than the MTU to fit into the target communication buffer. Larger requests must be split into smaller ones.

The driver uses size-aligned operations to write to the target memory (for example; uses proper write-word instruction when an address is aligned to 4 bytes).

Masked Memory Write

To implement the write access to a single bit or a group of bits of target variables, the Masked Memory Write feature is available in the FreeMASTER protocol and it is supported by the driver using the Read-Modify-Write approach.

Be careful when writing to bit fields of volatile variables that are also modified in an application interrupt. The interrupt may be serviced in the middle of a read-modify-write operation and it may cause data corruption.

Oscilloscope

The protocol and driver enables any number of variables to be read at once with a single request from the host. This feature is called Oscilloscope and the FreeMASTER tool uses it to display a real-time graph of variable values.

The driver can be configured to support any number of Oscilloscope instances and enable simultaneously running graphs to be displayed on the host computer screen.

Recorder

The protocol enables the host to select target variables whose values are then periodically recorded into a dedicated on-board memory buffer. After such data sampling stops (either on a host request or by evaluating a threshold-crossing condition), the data buffer is downloaded to the host and displayed as a graph. The data sampling rate is not limited by the speed of the communication line, so it enables displaying the variable transitions in a very high resolution.

The driver can be configured to support multiple Recorder instances and enable multiple recorder graphs to be displayed on the host screen. Having multiple recorders also enables setting the recording point differently for each instance. For example; one instance may be recording data in a general timer interrupt while another instance may record at a specific control algorithm time in the PWM interrupt.

TSA

With the TSA feature, data types and variables can be described directly in the application source code. Such information is later provided to the FreeMASTER tool which may use it instead of reading symbol data from the application ELF executable file.

The information is encoded as so-called TSA tables which become direct part of the application code. The TSA tables contain descriptors of variables that shall be visible to the host tool. The descriptors can describe the memory areas by specifying the address and size of the memory block or more conveniently using the C variable names directly. Different set of TSA descriptors can be used to encode information about the structure types, unions, enumerations, or arrays.

The driver also supports special types of TSA table entries to describe user resources like external EEPROM and SD Card files, memory-mapped files, virtual directories, web URL hyperlinks, and constant enumerations.

TSA Safety

When the TSA is enabled in the application, the TSA Safety can be enabled and validate the memory accesses directly by the embedded-side driver. When the TSA Safety is turned on, any memory request received from the host is validated and accepted only if it belongs to a TSA-described object. The TSA entries can be declared as Read-Write or Read-Only so that the driver can actively deny the write access to the Read-Only objects.

Application commands

The Application Commands are high-level messages that can be delivered from the PC Host to the embedded application for further processing. The embedded application can either poll the status, or be called back when a new Application Command arrives to be processed. After the embedded application acknowledges that the command is handled, the host receives the Result Code and reads the other return data from memory. Both the Application Commands and the Result Codes are specific to a given application and it is user's responsibility to define them. The FreeMASTER protocol and the FreeMASTER driver only implement the delivery channel and a set of API calls to enable the Application Command processing in general.

Pipes

The Pipes enable buffered and stream-oriented data exchange between the PC Host and the target application. Any pipe can be written to and read from at both ends (either on the PC or the MCU). The data transmission is acknowledged using the special FreeMASTER protocol commands. It is guaranteed that the data bytes are delivered from the writer to the reader in a proper order and without losses.

Serial single-wire operation

The MCU Serial Communication Driver natively supports normal dual-wire operation. Because the protocol is half-duplex only, the driver can also operate in two single-wire modes:

- "External" single-wire operation where the Receiver and Transmitter pins are shorted on the board. This mode is supported by default in the MCU driver because the Receiver and Transmitter units are enabled or disabled whenever needed. It is also easy to extend this operation for the RS485 communication.
- "True" single-wire mode which uses only a single pin and the direction switching is made by the UART module. This mode of operation must be enabled by defining the FMSTR_SERIAL_SINGLEWIRE configuration option.

Multi-session support

With networking interface it is possible for multiple clients to access the target MCU simultaneously. Reading and writing of target memory is processed atomically so there is no risk of data corruption. The state-full resources such as Recorders or Oscilloscopes are locked to a client session upon first use and access is denied to other clients until lock is released..

Zephyr-specific

Dedicated communication task

FreeMASTER communication may run isolated in a dedicated task. The task automates the FMSTR_Init and FMSTR_Poll calls together with periodic activities enabling the FreeMASTER UI to fetch information about tasks and CPU utilization. The task can be started automatically or manually, and it must be assigned a priority to be able to react on interrupts and other communication events. Refer to Zephyr FreeMASTER sample applications which all use this communication task.

Zephyr shell and logging over FreeMASTER pipe

FreeMASTER implements a shell backend which may use FreeMASTER pipe as a I/O terminal and logging output. Refer to Zephyr FreeMASTER sample applications which all use this feature.

Automatic TSA tables

TSA tables can be declared as "automatic" in Zephyr which make them automatically registered in the table list. This may be very useful when there are many TSA tables or when the tables are defined in different (often unrelated) libraries linked together. In this case user does not need to build a list of all tables manually.

Driver files The driver source files can be found in a top-level src folder, further divided into the sub-folders:

- ***src/platforms*** platform-specific folder—one folder exists for each supported processor platform (for example; 32-bit Little Endian platform). Each such folder contains a platform header file with data types and a code which implements the potentially platform-specific operations, such as aligned memory access.
- ***src/common*** folder—contains the common driver source files shared by the driver for all supported platforms. All the *.c* files must be added to the project, compiled, and linked together with the application.
  - *freemaster.h* - master driver header file, which declares the common data types, macros, and prototypes of the FreeMASTER driver API functions.
  - *freemaster_cfg.h.example* - this file can serve as an example of the FreeMASTER driver configuration file. Save this file into a project source code folder and rename it to *freemaster_cfg.h*. The FreeMASTER driver code includes this file to get the project-specific configuration options and to optimize the compilation of the driver.
  - *freemaster_defcfg.h* - defines the default values for each FreeMASTER configuration option if the option is not set in the *freemaster_cfg.h* file.
  - *freemaster_protocol.h* - defines the FreeMASTER protocol constants used internally by the driver.
  - *freemaster_protocol.c* - implements the FreeMASTER protocol decoder and handles the basic Get Configuration Value, Memory Read, and Memory Write commands.
  - *freemaster_rec.c* - handles the Recorder-specific commands and implements the Recorder sampling and triggering routines. When the Recorder is disabled by the FreeMASTER driver configuration file, this file only compiles to empty API functions.
  - *freemaster_scope.c* - handles the Oscilloscope-specific commands. If the Oscilloscope is disabled by the FreeMASTER driver configuration file, this file compiles as void.
  - *freemaster_pipes.c* - implements the Pipes functionality when the Pipes feature is enabled.

  - *freemaster_appcmd.c* - handles the communication commands used to deliver and execute the Application Commands within the context of the embedded application. When the Application Commands are disabled by the FreeMASTER driver configuration file, this file only compiles to empty API functions.
  - *freemaster_tsa.c* - handles the commands specific to the TSA feature. This feature enables the FreeMASTER host tool to obtain the TSA memory descriptors declared in the embedded application. If the TSA is disabled by the FreeMASTER driver configuration file, this file compiles as void.
  - *freemaster_tsa.h* - contains the declaration of the macros used to define the TSA memory descriptors. This file is indirectly included into the user application code (via *freemaster.h*).
  - *freemaster_sha.c* - implements the SHA-1 hash code used in the password authentication algorithm.
  - *freemaster_private.h* - contains the declarations of functions and data types used internally in the driver. It also contains the C pre-processor statements to perform the compile-time verification of the user configuration provided in the *freemaster_cfg.h* file.
  - *freemaster_serial.c* - implements the serial protocol logic including the CRC, FIFO queuing, and other communication-related operations. This code calls the functions of the low-level communication driver indirectly via a character-oriented API exported by the specific low-level driver.
  - *freemaster_serial.h* - defines the low-level character-oriented Serial API.
  - *freemaster_can.c* - implements the CAN protocol logic including the CAN message preparation, signalling using the first data byte in the CAN frame, and other communication-related operations. This code calls the functions of the low-level communication driver indirectly via a message-oriented API exported by the specific low-level driver.
  - *freemaster_can.h* - defines the low-level message-oriented CAN API.
  - *freemaster_net.c* - implements the Network protocol transport logic including multiple session management code.
  - *freemaster_net.h* - definitions related to the Network transport.
  - *freemaster_pdbdm.c* - implements the packet-driven BDM communication buffer and other communication-related operations.
  - *freemaster_utils.c* - aligned memory copy routines, circular buffer management and other utility functions
  - *freemaster_utils.h* - definitions related to utility code.
- ***src/drivers/[sdk]/serial*** - contains the code related to the serial communication implemented using one of the supported SDK frameworks.
  - *freemaster_serial_XXX.c* and *.h* - implement low-level access to the communication peripheral registers. Different files exist for the UART, LPUART, USART, and other kinds of Serial communication modules.
- ***src/drivers/[sdk]/can*** - contains the code related to the serial communication implemented using one of the supported SDK frameworks.
  - *freemaster_XXX.c* and *.h* - implement low-level access to the communication peripheral registers. Different files exist for the FlexCAN, msCAN, MCAN, and other kinds of CAN communication modules.
- ***src/drivers/[sdk]/network*** - contains low-level code adapting the FreeMASTER Network transport to an underlying TCP/IP or RTT stack.

- *freemaster_net_lwip_tcp.c* and *_udp.c* - default networking implementation of TCP and UDP transports using lwIP stack.
- *freemaster_net_segger_rtt.c* - implementation of network transport using Segger J-Link RTT interface

Driver configuration The driver is configured using a single header file (*freemaster_cfg.h*). Create this file and save it together with other project source files before compiling the driver code. All FreeMASTER driver source files include the *freemaster_cfg.h* file and use the macros defined here for the conditional and parameterized compilation. The C compiler must locate the configuration file when compiling the driver files. Typically, it can be achieved by putting this file into a folder where the other project-specific included files are stored.

As a starting point to create the configuration file, get the *freemaster_cfg.h.example* file, rename it to *freemaster_cfg.h*, and save it into the project area.

Note: It is NOT recommended to leave the *freemaster_cfg.h* file in the FreeMASTER driver source code folder. The configuration file must be placed at a project-specific location, so that it does not affect the other applications that use the same driver.

Configurable items This section describes the configuration options which can be defined in *freemaster_cfg.h*.

Interrupt modes

```
#define FMSTR_LONG_INTR    [0|1]
#define FMSTR_SHORT_INTR   [0|1]
#define FMSTR_POLL_DRIVEN  [0|1]
```

Value Type

boolean (0 or 1)

Description

Exactly one of the three macros must be defined to non-zero. The others must be defined to zero or left undefined. The non-zero-defined constant selects the interrupt mode of the driver. See *Driver interrupt modes*.

- FMSTR_LONG_INTR — long interrupt mode
- FMSTR_SHORT_INTR — short interrupt mode
- FMSTR_POLL_DRIVEN — poll-driven mode

Note: Some options may not be supported by all communication interfaces. For example, the FMSTR_SHORT_INTR option is not supported by the USB_CDC interface.

Protocol transport

```
#define FMSTR_TRANSPORT [identifier]
```

Value Type

Driver identifiers are structure instance names defined in FreeMASTER source code. Specify one of existing instances to make use of the protocol transport.

Description

Use one of the pre-defined constants, as implemented by the FreeMASTER code. The current driver supports the following transports:

- **FMSTR_SERIAL** - serial communication protocol
- **FMSTR_CAN** - using CAN communication
- **FMSTR_PDBDM** - using packet-driven BDM communication
- **FMSTR_NET** - network communication using TCP or UDP protocol

Serial transport

This section describes configuration parameters used when serial transport is used:

```
#define FMSTR_TRANSPORT FMSTR_SERIAL
```

FMSTR_SERIAL_DRV

Select what low-level driver interface will be used when implementing the Serial communication.

```
#define FMSTR_SERIAL_DRV [identifier]
```

Value Type

Driver identifiers are structure instance names defined in FreeMASTER drivers code. Specify one of existing serial driver instances.

Description

When using MCUXpresso SDK, use one of the following constants (see */drivers/mcuxsdk/serial* implementation):

- **FMSTR_SERIAL_MCUX_UART** - UART driver
- **FMSTR_SERIAL_MCUX_LPUART** - LPUART driver
- **FMSTR_SERIAL_MCUX_USART** - USART driver
- **FMSTR_SERIAL_MCUX_MINIUSART** - miniUSART driver
- **FMSTR_SERIAL_MCUX_QSCI** - DSC QSCI driver
- **FMSTR_SERIAL_MCUX_USB** - USB/CDC class driver (also see code in the */support/mcuxsdk_usb* folder)
- **FMSTR_SERIAL_56F800E_EONCE** - DSC JTAG EOnCE driver

Other SDKs or BSPs may define custom low-level driver interface structure which may be used as FMSTR_SERIAL_DRV. For example:

- **FMSTR_SERIAL_DREG_UART** - demonstrates the low-level interface implemented without the MCUXpresso SDK and using direct access to peripheral registers.

FMSTR_SERIAL_BASE

```
#define FMSTR_SERIAL_BASE [address|symbol]
```

Value Type

Optional address value (numeric or symbolic)

Description

Specify the base address of the UART, LPUART, USART, or other serial peripheral module to be used for the communication. This value is not defined by default. User application should call FMSTR_SetSerialBaseAddress() to select the peripheral module.

FMSTR_COMM_BUFFER_SIZE

```
#define FMSTR_COMM_BUFFER_SIZE [number]
```

Value Type

0 or a value in range 32...255

Description

Specify the size of the communication buffer to be allocated by the driver.
Default value, which suits all driver features, is used when this option is defined as 0.

FMSTR_COMM_RQUEUE_SIZE

```
#define FMSTR_COMM_RQUEUE_SIZE [number]
```

Value Type

Value in range 0...255

Description

Specify the size of the FIFO receiver queue used to quickly receive and store characters in the FMSTR_SHORT_INTR interrupt mode.
The default value is 32 B.

FMSTR_SERIAL_SINGLEWIRE

```
#define FMSTR_SERIAL_SINGLEWIRE [0|1]
```

Value Type

Boolean 0 or 1.

Description

Set to non-zero to enable the "True" single-wire mode which uses a single MCU pin to communicate. The low-level driver enables the pin direction switching when the MCU peripheral supports it.

CAN Bus transport

This section describes configuration parameters used when CAN transport is used:

```
#define FMSTR_TRANSPORT FMSTR_CAN
```

FMSTR_CAN_DRV

Select what low-level driver interface will be used when implementing the CAN communication.

```
#define FMSTR_CAN_DRV [identifier]
```

Value Type

Driver identifiers are structure instance names defined in FreeMASTER drivers code. Specify one of existing CAN driver instances.

Description

When using MCUXpresso SDK, use one of the following constants (see */drivers/mcuxsdk/can implementation*):

- **FMSTR_CAN_MCUX_FLEXCAN** - FlexCAN driver
- **FMSTR_CAN_MCUX_MCAN** - MCAN driver
- **FMSTR_CAN_MCUX_MSCAN** - msCAN driver
- **FMSTR_CAN_MCUX_DSCFLEXCAN** - DSC FlexCAN driver
- **FMSTR_CAN_MCUX_DSCMSCAN** - DSC msCAN driver

Other SDKs or BSPs may define the custom low-level driver interface structure which may be used as FMSTR_CAN_DRV.

FMSTR_CAN_BASE

```
#define FMSTR_CAN_BASE [address|symbol]
```

Value Type

Optional address value (numeric or symbolic)

Description

Specify the base address of the FlexCAN, msCAN, or other CAN peripheral module to be used for the communication. This value is not defined by default. User application should call FMSTR_SetCanBaseAddress() to select the peripheral module.

FMSTR_CAN_CMDID

```
#define FMSTR_CAN_CMDID [number]
```

Value Type

CAN identifier (11-bit or 29-bit number)

Description

CAN message identifier used for FreeMASTER commands (direction from PC Host tool to target application). When declaring 29-bit identifier, combine the numeric value with FMSTR_CAN_EXTID bit. Default value is 0x7AA.

FMSTR_CAN_RSPID

```
#define FMSTR_CAN_RSPID [number]
```

Value Type

CAN identifier (11-bit or 29-bit number)

Description

CAN message identifier used for responding messages (direction from target application to PC Host tool). When declaring 29-bit identifier, combine the numeric value with FMSTR_CAN_EXTID bit. Note that both *CMDID* and *RSPID* values may be the same. Default value is 0x7AA.

FMSTR_FLEXCAN_TXMB

```
#define FMSTR_FLEXCAN_TXMB [number]
```

Value Type

Number in range of 0..N where N is number of CAN message-buffers supported by HW module.

Description

Only used when the FlexCAN low-level driver is used. Define the FlexCAN message buffer for CAN frame transmission. Default value is 0.

FMSTR_FLEXCAN_RXMB

```
#define FMSTR_FLEXCAN_RXMB [number]
```

Value Type

Number in range of 0..N where N is number of CAN message-buffers supported by HW module.

Description

Only used when the FlexCAN low-level driver is used. Define the FlexCAN message buffer for CAN frame reception. Note that the FreeMASTER driver may also operate with a common message buffer used by both TX and RX directions. Default value is 1.

Network transport

This section describes configuration parameters used when Network transport is used:

```
#define FMSTR_TRANSPORT FMSTR_NET
```

FMSTR_NET_DRV

Select network interface implementation.

```
#define FMSTR_NET_DRV [identifier]
```

Value Type

Identifiers are structure instance names defined in FreeMASTER drivers code. Specify one of existing NET driver instances.

Description

When using MCUXpresso SDK, use one of the following constants (see */drivers/mcuxsdk/network implementation*):

- **FMSTR_NET_LWIP_TCP** - TCP communication using lwIP stack
- **FMSTR_NET_LWIP_UDP** - UDP communication using lwIP stack
- **FMSTR_NET_SEGGER_RTT** - Communication using SEGGER J-Link RTT interface

Other SDKs or BSPs may define the custom networking interface which may be used as FMSTR_CAN_DRV.

Add another row below:

FMSTR_NET_PORT

```
#define FMSTR_NET_PORT [number]
```

Value Type

TCP or UDP port number (short integer)

Description

Specifies the server port number used by TCP or UDP protocols.

FMSTR_NET_BLOCKING_TIMEOUT

```
#define FMSTR_NET_BLOCKING_TIMEOUT [number]
```

Value Type

Timeout as number of milliseconds

Description

This value specifies a timeout in milliseconds for which the network socket operations may block the execution inside *FMSTR_Poll*. This may be set high (e.g. 250) when a dedicated RTOS task is used to handle FreeMASTER protocol polling. Set to a lower value when the polling task is also responsible for other operations. Set to 0 to attempt to use non-blocking socket operations.

FMSTR_NET_AUTODISCOVERY

```
#define FMSTR_NET_AUTODISCOVERY [0|1]
```

Value Type

Boolean 0 or 1.

Description

This option enables the FreeMASTER driver to use a separate UDP socket to broadcast auto-discovery messages to network. This helps the FreeMASTER tool to discover the target device address, port and protocol options.

Debugging options

FMSTR_DISABLE

```
#define FMSTR_DISABLE [0|1]
```

Value Type

boolean (0 or 1)

Description

Define as non-zero to disable all FreeMASTER features, exclude the driver code from build, and compile all its API functions empty. This may be useful to remove FreeMASTER without modifying any application source code. Default value is 0 (false).

FMSTR_DEBUG_TX

```
#define FMSTR_DEBUG_TX [0|1]
```

Value Type

Boolean 0 or 1.

Description

Define as non-zero to enable the driver to periodically transmit test frames out on the selected communication interface (SCI or CAN). With the debug transmission enabled, it is simpler to detect problems in the baudrate or other communication configuration settings.

The test frames are transmitted until the first valid command frame is received from the PC Host tool. The test frame is a valid error status frame, as defined by the protocol format. On the serial line, the test frame consists of three printable characters (+©W) which are easy to capture using the serial terminal tools.

This feature requires the FMSTR_Poll() function to be called periodically. Default value is 0 (false).

FMSTR_APPLICATION_STR

```
#define FMSTR_APPLICATION_STR
```

Value Type

String.

Description

Name of the application visible in FreeMASTER host application.

Memory access

FMSTR_USE_READMEM

```
#define FMSTR_USE_READMEM [0|1]
```

Value Type

Boolean 0 or 1.

Description

Define as non-zero to implement the Memory Read command and enable FreeMASTER to have read access to memory and variables. The access can be further restricted by using a TSA feature. Default value is 1 (true).

FMSTR_USE_WRITEMEM

```
#define FMSTR_USE_WRITEMEM [0|1]
```

Value Type

Boolean 0 or 1.

Description

Define as non-zero to implement the Memory Write command.
The default value is 1 (true).

Oscilloscope options

FMSTR_USE_SCOPE

```
#define FMSTR_USE_SCOPE [number]
```

Value Type

Integer number.

Description

Number of Oscilloscope instances to be supported. Set to 0 to disable the Oscilloscope feature.
Default value is 0.

FMSTR_MAX_SCOPE_VARS

```
#define FMSTR_MAX_SCOPE_VARS [number]
```

Value Type

Integer number larger than 2.

Description

Number of variables to be supported by each Oscilloscope instance.
Default value is 8.

Recorder options

FMSTR_USE_RECORDER

```
#define FMSTR_USE_RECORDER [number]
```

Value Type

Integer number.

Description

Number of Recorder instances to be supported. Set to 0 to disable the Recorder feature.
Default value is 0.

FMSTR_REC_BUFF_SIZE

```
#define FMSTR_REC_BUFF_SIZE [number]
```

Value Type

Integer number larger than 2.

Description

Defines the size of the memory buffer used by the Recorder instance #0.
Default: not defined, user shall call 'FMSTR_RecorderCreate()" API function to specify this parameter in run time.

FMSTR_REC_TIMEBASE

```
#define FMSTR_REC_TIMEBASE [time specification]
```

Value Type

Number (nanoseconds time).

Description

Defines the base sampling rate in nanoseconds (sampling speed) Recorder instance #0.

Use one of the following macros:

- FMSTR_REC_BASE_SECONDS(x)
- FMSTR_REC_BASE_MILLISEC(x)
- FMSTR_REC_BASE_MICROSEC(x)
- FMSTR_REC_BASE_NANOSEC(x)

Default: not defined, user shall call 'FMSTR_RecorderCreate()" API function to specify this parameter in run time.

FMSTR_REC_FLOAT_TRIG

```
#define FMSTR_REC_FLOAT_TRIG [0|1]
```

Value Type

Boolean 0 or 1.

Description

Define as non-zero to implement the floating-point triggering. Be aware that floating-point triggering may grow the code size by linking the floating-point standard library.

Default value is 0 (false).

Application Commands options

FMSTR_USE_APPCMD

```
#define FMSTR_USE_APPCMD [0|1]
```

Value Type

Boolean 0 or 1.

Description

Define as non-zero to implement the Application Commands feature.
Default value is 0 (false).

FMSTR_APPCMD_BUFF_SIZE

```
#define FMSTR_APPCMD_BUFF_SIZE [size]
```

Value Type

Numeric buffer size in range 1..255

Description

The size of the Application Command data buffer allocated by the driver. The buffer stores the (optional) parameters of the Application Command which waits to be processed.

FMSTR_MAX_APPCMD_CALLS

```
#define FMSTR_MAX_APPCMD_CALLS [number]
```

Value Type

Number in range 0..255

Description

The number of different Application Commands that can be assigned a callback handler function using FMSTR_RegisterAppCmdCall(). Default value is 0.

TSA options

FMSTR_USE_TSA

```
#define FMSTR_USE_TSA [0|1]
```

Value Type

Boolean 0 or 1.

Description

Enable the FreeMASTER TSA feature to be used. With this option enabled, the TSA tables defined in the applications are made available to the FreeMASTER host tool.
Default value is 0 (false).

FMSTR_USE_TSA_SAFETY

```
#define FMSTR_USE_TSA_SAFETY [0|1]
```

Value Type

Boolean 0 or 1.

Description

Enable the memory access validation in the FreeMASTER driver. With this option, the host tool is not able to access the memory which is not described by at least one TSA descriptor. Also a write access is denied for objects defined as read-only in TSA tables.
Default value is 0 (false).

FMSTR_USE_TSA_INROM

```
#define FMSTR_USE_TSA_INROM [0|1]
```

Value Type

Boolean 0 or 1.

Description

Declare all TSA descriptors as *const*, which enables the linker to put the data into the flash memory. The actual result depends on linker settings or the linker commands used in the project.
Default value is 0 (false).

FMSTR_USE_TSA_DYNAMIC

```
#define FMSTR_USE_TSA_DYNAMIC [0|1]
```

Value Type

Boolean 0 or 1.

Description

Enable runtime-defined TSA entries to be added to the TSA table by the FMSTR_SetUpTsaBuff() and FMSTR_TsaAddVar() functions.
Default value is 0 (false).

Pipes options

FMSTR_USE_PIPES

```
#define FMSTR_USE_PIPES [0|1]
```

Value Type

Boolean 0 or 1.

Description

Enable the FreeMASTER Pipes feature to be used.
Default value is 0 (false).

FMSTR_MAX_PIPES_COUNT

```
#define FMSTR_MAX_PIPES_COUNT [number]
```

Value Type

Number in range 1..63.

Description

The number of simultaneous pipe connections to support.
The default value is 1.

Driver interrupt modes To implement the communication, the FreeMASTER driver handles the Serial or CAN module's receive and transmit requests. Use the *freemaster_cfg.h* configuration file to select whether the driver processes the communication automatically in the interrupt service routine handler or if it only polls the status of the module (typically during the application idle time).

This section describes each of the interrupt mode in more details.

Completely Interrupt-Driven operation

Activated using:

```
#define FMSTR_LONG_INTR 1
```

In this mode, both the communication and the FreeMASTER protocol decoding is done in the *FMSTR_SerialIsr*, *FMSTR_CanIsr*, or other interrupt service routine. Because the protocol execution may be a lengthy task (especially with the TSA-Safety enabled) it is recommended to use this mode only if the interrupt prioritization scheme is possible in the application and the FreeMASTER interrupt is assigned to a lower (the lowest) priority.

In this mode, the application code must register its own interrupt handler for all interrupt vectors related to the selected communication interface and call the FMSTR_SerialIsr or FMSTR_CanIsr functions from that handler.

Mixed Interrupt and Polling Modes

Activated using:

```
#define FMSTR_SHORT_INTR 1
```

In this mode, the communication processing time is split between the interrupt routine and the main application loop or task. The raw communication is handled by the *FMSTR_SerialIsr, FMSTR_CanIsr*, or other interrupt service routine, while the protocol decoding and execution is handled by the *FMSTR_Poll* routine. Call *FMSTR_Poll* during the idle time in the application main loop.

The interrupt processing in this mode is relatively fast and deterministic. Upon a serial-receive event, the received character is only placed into a FIFO-like queue and it is not further processed. Upon a CAN receive event, the received frame is stored into a receive buffer. When transmitting, the characters are fetched from the prepared transmit buffer.

In this mode, the application code must register its own interrupt handler for all interrupt vectors related to the selected communication interface and call the *FMSTR_SerialIsr* or *FMSTR_CanIsr* functions from that handler.

When the serial interface is used as the serial communication interface, ensure that the *FMSTR_Poll* function is called at least once per *N* character time periods. *N* is the length of the FreeMASTER FIFO queue (*FMSTR_COMM_RQUEUE_SIZE*) and the character time is the time needed to transmit or receive a single byte over the SCI line.

Completely Poll-driven

```
#define FMSTR_POLL_DRIVEN 1
```

In this mode, both the communication and the FreeMASTER protocol decoding are done in the *FMSTR_Poll* routine. No interrupts are needed and the *FMSTR_SerialIsr*, *FMSTR_CanIsr*, and similar handlers compile to an empty code.

When using this mode, ensure that the *FMSTR_Poll* function is called by the application at least once per the serial "character time" which is the time needed to transmit or receive a single character.

In the latter two modes (*FMSTR_SHORT_INTR* and *FMSTR_POLL_DRIVEN*), the protocol handling takes place in the FMSTR_Poll routine. An application interrupt can occur in the middle of the Read Memory or Write Memory commands' execution and corrupt the variable being accessed by the FreeMASTER driver. In these two modes, some issues or glitches may occur when using FreeMASTER to visualize or monitor volatile variables modified in interrupt servicing code.

The same issue may appear even in the full interrupt mode (FMSTR_LONG_INTR), if volatile variables are modified in the interrupt code with a priority higher than the priority of the communication interrupt.

Data types Simple portability was one of the main requirements when writing the FreeMASTER driver. This is why the driver code uses the privately-declared data types and the vast majority of the platform-dependent code is separated in the platform-dependent source files. The data types used in the driver API are all defined in the platform-specific header file.

To prevent name conflicts with the symbols used in the application, all data types, macros, and functions have the FMSTR_ prefix. The only global variables used in the driver are the transport and low-level API structures exported from the driver-implementation layer to upper layers. Other than that, all private variables are declared as static and named using the fmstr_ prefix.

Communication interface initialization The FreeMASTER driver does not perform neither the initialization nor the configuration of the peripheral module that it uses to communicate. It is the application startup code responsibility to configure the communication module before the FreeMASTER driver is initialized by the FMSTR_Init call.

When the Serial communication module is used as the FreeMASTER communication interface, configure the UART receive and transmit pins, the serial communication baud rate, parity (no-parity), the character length (eight bits), and the number of stop bits (one) before initializing the FreeMASTER driver. For either the long or the short interrupt modes of the driver (see *Driver interrupt modes*), configure the interrupt controller and register an application-specific interrupt handler for all interrupt sources related to the selected serial peripheral module. Call the FMSTR_SerialIsr function from the application handler.

When a CAN module is used as the FreeMASTER communication interface, configure the CAN receive and transmit pins and the CAN module bit rate before initializing the FreeMASTER driver. For either the long or the short interrupt modes of the driver (see *Driver interrupt modes*), configure the interrupt controller and register an application-specific interrupt handler for all interrupt sources related to the selected CAN peripheral module. Call the FMSTR_CanIsr function from the application handler.

Note: It is not necessary to enable or unmask the serial nor the CAN interrupts before initializing the FreeMASTER driver. The driver enables or disables the interrupts and communication lines, as required during runtime.

FreeMASTER Recorder calls When using the FreeMASTER Recorder in the application (FMSTR_USE_RECORDER > 0), call the FMSTR_RecorderCreate function early after FMSTR_Init to set up each recorder instance to be used in the application. Then call the FMSTR_Recorder function periodically in the code where the data recording should occur. A typical place to call the Recorder routine is at the timer or PWM interrupts, but it can be anywhere else. The example applications provided together with the driver code call the FMSTR_Recorder in the main application loop.

In applications where FMSTR_Recorder is called periodically with a constant period, specify the period in the Recorder configuration structure before calling FMSTR_RecorderCreate. This setting enables the PC Host FreeMASTER tool to display the X-axis of the Recorder graph properly scaled for the time domain.

Driver usage Start using or evaluating FreeMASTER by opening some of the example applications available in the driver setup package.

Follow these steps to enable the basic FreeMASTER connectivity in the application:

- Make sure that all **.c* files of the FreeMASTER driver from the *src/common/platforms/[your_platform]* folder are a part of the project. See *Driver files* for more details.
- Configure the FreeMASTER driver by creating or editing the *freemaster_cfg.h* file and by saving it into the application project directory. See *Driver configuration* for more details.
- Include the *freemaster.h* file into any application source file that makes the FreeMASTER API calls.
- Initialize the Serial or CAN modules. Set the baud rate, parity, and other parameters of the communication. Do not enable the communication interrupts in the interrupt mask registers.
- For the FMSTR_LONG_INTR and FMSTR_SHORT_INTR modes, install the application-specific interrupt routine and call the FMSTR_SerialIsr or FMSTR_CanIsr functions from this handler.
- Call the FMSTR_Init function early on in the application initialization code.
- Call the FMSTR_RecorderCreate functions for each Recorder instance to enable the Recorder feature.
- In the main application loop, call the FMSTR_Poll API function periodically when the application is idle.
- For the FMSTR_SHORT_INTR and FMSTR_LONG_INTR modes, enable the interrupts globally so that the interrupts can be handled by the CPU.

Communication troubleshooting The most common problem that causes communication issues is a wrong baud rate setting or a wrong pin multiplexer setting of the target MCU. When a communication between the PC Host running FreeMASTER and the target MCU cannot be established, try enabling the FMSTR_DEBUG_TX option in the *freemaster_cfg.h* file and call the FMSTR_Poll function periodically in the main application task loop.

With this feature enabled, the FreeMASTER driver periodically transmits a test frame through the Serial or CAN lines. Use a logic analyzer or an oscilloscope to monitor the signals at the communication pins of the CPU device to examine whether the bit rate and signal polarity are configured properly.

Driver API

This section describes the driver Application Programmers' Interface (API) needed to initialize and use the FreeMASTER serial communication driver.

Control API There are three key functions to initialize and use the driver.

FMSTR_Init

Prototype

```
FMSTR_BOOL FMSTR_Init(void);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_protocol.c*

Description

This function initializes the internal variables of the FreeMASTER driver and enables the communication interface. This function does not change the configuration of the selected communication module. The hardware module must be initialized before the *FMSTR_Init* function is called.

A call to this function must occur before calling any other FreeMASTER driver API functions.

FMSTR_Poll

Prototype

```
void FMSTR_Poll(void);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_protocol.c*

Description

In the poll-driven or short interrupt modes, this function handles the protocol decoding and execution (see *Driver interrupt modes*). In the poll-driven mode, this function also handles the communication interface with the PC. Typically, the *FMSTR_Poll* function is called during the "idle" time in the main application task loop.

To prevent the receive data overflow (loss) on a serial interface, make sure that the FMSTR_Poll function is called at least once per the time calculated as:

*N * Tchar*

where:

- *N* is equal to the length of the receive FIFO queue (configured by the FMSTR_COMM_RQUEUE_SIZE macro). *N* is 1 for the poll-driven mode.
- *Tchar* is the character time, which is the time needed to transmit or receive a single byte over the SCI line.

Note: In the long interrupt mode, this function typically compiles as an empty function and can still be called. It is worthwhile to call this function regardless of the interrupt mode used in the application. This approach enables a convenient switching between the different interrupt modes only by changing the configuration macros in the *freemaster_cfg.h* file.

FMSTR_SerialIsr / FMSTR_CanIsr

Prototype

```
void FMSTR_SerialIsr(void);
void FMSTR_CanIsr(void);
```

- Declaration: *freemaster.h*
- Implementation: *hw-specific low-level driver C file*

Description

This function contains the interrupt-processing code of the FreeMASTER driver. In long or short interrupt modes (see *Driver interrupt modes*), this function must be called from the application interrupt service routine registered for the communication interrupt vector. On platforms where the communication module uses multiple interrupt vectors, the application should register a handler for all vectors and call this function at each interrupt.

Note: In a poll-driven mode, this function is compiled as an empty function and does not have to be used.

Recorder API FMSTR_RecorderCreate

Prototype

```
FMSTR_BOOL FMSTR_RecorderCreate(FMSTR_INDEX recIndex, FMSTR_REC_BUFF* buffCfg);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_rec.c*

Description

This function registers a recorder instance and enables it to be used by the PC Host tool. Call this function for all recorder instances from 0 to the maximum number defined by the FMSTR_USE_RECORDER configuration option (minus one). An exception to this requirement is the recorder of instance *0* which may be automatically configured by FMSTR_Init when the *freemaster_cfg.h* configuration file defines the *FMSTR_REC_BUFF_SIZE* and *FMSTR_REC_TIMEBASE* options.

For more information, see *Configurable items*.

FMSTR_Recorder

Prototype

```
void FMSTR_Recorder(FMSTR_INDEX recIndex);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_rec.c*

Description

This function takes a sample of the variables being recorded using the FreeMASTER Recorder instance *recIndex*. If the selected Recorder is not active when the *FMSTR_Recorder* function is being called, the function returns immediately. When the Recorder is active, the values of the variables being recorded are copied into the recorder buffer and the trigger conditions are evaluated.

If a trigger condition is satisfied, the Recorder enters the post-trigger mode, where it counts down the follow-up samples (number of *FMSTR_Recorder* function calls) and de-activates the Recorder when the required post-trigger samples are finished.

The *FMSTR_Recorder* function is typically called in the timer or PWM interrupt service routines. This function can also be called in the application main loop (for testing purposes).

FMSTR_RecorderTrigger

Prototype

```
void FMSTR_RecorderTrigger(FMSTR_INDEX recIndex);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_rec.c*

Description

This function forces the Recorder trigger condition to happen, which causes the Recorder to be automatically deactivated after the post-trigger samples are sampled. Use this function in the application code for programmatic control over the Recorder triggering. This can be useful when a more complex triggering conditions need to be used.

Fast Recorder API The Fast Recorder feature is not available in the FreeMASTER driver version 3. This feature was heavily dependent on the target platform and it was only available for the 56F8xxxx DSCs.

TSA Tables When the TSA is enabled in the FreeMASTER driver configuration file (by setting the FMSTR_USE_TSA macro to a non-zero value), it defines the so-called TSA tables in the application. This section describes the macros that must to be used to define the TSA tables.

There can be any number of TSA tables spread across the application source files. There must be always exactly one TSA Table List defined, which informs the FreeMASTER driver about the active TSA tables.

When there is at least one TSA table and one TSA Table List defined in the application, the TSA information automatically appears in the FreeMASTER symbols list. The symbols can then be used to create FreeMASTER variables for visualization or control.

TSA table definition

The TSA table describes the static or global variables together with their address, size, type, and access-protection information. If the TSA-described variables are of a structure type, the TSA table may also describe this type and provide an access to the individual structure members of the variable.

The TSA table definition begins with the FMSTR_TSA_TABLE_BEGIN macro with a *table_id* identifying the table. The *table_id* shall be a valid C-langiage symbol.

```
FMSTR_TSA_TABLE_BEGIN(table_id)
```

After this opening macro, the TSA descriptors are placed using these macros:

```
/* Adding variable descriptors */
FMSTR_TSA_RW_VAR(name, type)  /* read/write variable entry */
FMSTR_TSA_RO_VAR(name, type)  /* read-only variable entry */

/* Description of complex data types */
FMSTR_TSA_STRUCT(struct_name) /* structure or union type entry */
FMSTR_TSA_MEMBER(struct_name, member_name, type)  /* structure member entry */

/* Memory blocks */
FMSTR_TSA_RW_MEM(name, type, address, size) /* read/write memory block */
FMSTR_TSA_RO_MEM(name, type, address, size) /* read-only memory block */
```

The table is closed using the FMSTR_TSA_TABLE_END macro:

```
FMSTR_TSA_TABLE_END()
```

TSA descriptor parameters

The TSA descriptor macros accept these parameters:

- *name* — variable name. The variable must be defined before the TSA descriptor references it.
- *type* — variable or member type. Only one of the pre-defined type constants may be used (see below).
- *struct_name* — structure type name. The type must be defined (typedef) before the TSA descriptor references it.
- *member_name* — structure member name.

Note: The structure member descriptors (FMSTR_TSA_MEMBER) must immediately follow the parent structure descriptor (FMSTR_TSA_STRUCT) in the table.

Note: To write-protect the variables in the FreeMASTER driver (FMSTR_TSA_RO_VAR), enable the TSA-Safety feature in the configuration file.

TSA variable types

The table lists *type* identifiers which can be used in TSA descriptors:

Constant	Description
FMSTR_TSA_UINT*n*	Unsigned integer type of size *n* bits (n=8,16,32,64)
FMSTR_TSA_SINT*n*	Signed integer type of size *n* bits (n=8,16,32,64)
FMSTR_TSA_FRAC*n*	Fractional number of size *n* bits (n=16,32,64).
FMSTR_TSA_FRAC_Q(*m*,*n*)	Signed fractional number in general Q form (m+n+1 total bits)
FMSTR_TSA_FRAC_UQ(*m*,*n*)	Unsigned fractional number in general UQ form (m+n total bits)
FMSTR_TSA_FLOAT	4-byte standard IEEE floating-point type
FMSTR_TSA_DOUBLE	8-byte standard IEEE floating-point type
FMSTR_TSA_POINTER	Generic pointer type defined (platform-specific 16 or 32 bit)
FMSTR_TSA_USERTYPE(*name*)	Structure or union type declared with FMSTR_TSA_STRUCT record

TSA table list

There shall be exactly one TSA Table List in the application. The list contains one entry for each TSA table defined anywhere in the application.

The TSA Table List begins with the FMSTR_TSA_TABLE_LIST_BEGIN macro and continues with the TSA table entries for each table.

```
FMSTR_TSA_TABLE_LIST_BEGIN()

FMSTR_TSA_TABLE(table_id)
FMSTR_TSA_TABLE(table_id2)
FMSTR_TSA_TABLE(table_id3)
...
```

The list is closed with the FMSTR_TSA_TABLE_LIST_END macro:

```
FMSTR_TSA_TABLE_LIST_END()
```

TSA Active Content entries

FreeMASTER v2.0 and higher supports TSA Active Content, enabling the TSA tables to describe the memory-mapped files, virtual directories, and URL hyperlinks. FreeMASTER can access such objects similarly to accessing the files and folders on the local hard drive.

With this set of TSA entries, the FreeMASTER pages can be embedded directly into the target MCU flash and accessed by FreeMASTER directly over the communication line. The HTML-coded pages rendered inside the FreeMASTER window can access the TSA Active Content resources using a special URL referencing the *fmstr:* protocol.

This example provides an overview of the supported TSA Active Content entries:

```
FMSTR_TSA_TABLE_BEGIN(files_and_links)

/* Directory entry applies to all subsequent MEMFILE entries */
FMSTR_TSA_DIRECTORY("/text_files")     /* entering a new virtual directory */

/* The readme.txt file will be accessible at the fmstr://text_files/readme.txt URL */
FMSTR_TSA_MEMFILE("readme.txt", readme_txt, sizeof(readme_txt)) /* memory-mapped file */

/* Files can also be specified with a full path so the DIRECTORY entry does not apply */
FMSTR_TSA_MEMFILE("/index.htm", index, sizeof(index))          /* memory-mapped file */
FMSTR_TSA_MEMFILE("/prj/demo.pmp", demo_pmp, sizeof(demo_pmp)) /* memory-mapped file */

/* Hyperlinks can point to a local MEMFILE object or to the Internet */
FMSTR_TSA_HREF("Board's Built-in Welcome Page", "/index.htm")
FMSTR_TSA_HREF("FreeMASTER Home Page", "http://www.nxp.com/freemaster")

/* Project file links simplify opening the projects from any URLs */
FMSTR_TSA_PROJECT("Demonstration Project (embedded)", "/prj/demo.pmp")
FMSTR_TSA_PROJECT("Full Project (online)", "http://mycompany.com/prj/demo.pmp")

FMSTR_TSA_TABLE_END()
```

TSA API FMSTR_SetUpTsaBuff

Prototype

```
FMSTR_BOOL FMSTR_SetUpTsaBuff(FMSTR_ADDR buffAddr, FMSTR_SIZE buffSize);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_tsa.c*

Arguments

- *buffAddr* [in] - address of the memory buffer for the dynamic TSA table
- *buffSize* [in] - size of the memory buffer which determines the maximum number of TSA entries to be added in the runtime

Description

This function must be used to assign the RAM memory buffer to the TSA subsystem when FMSTR_USE_TSA_DYNAMIC is enabled. The memory buffer is then used to store the TSA entries added dynamically to the runtime TSA table using the FMSTR_TsaAddVar function call. The runtime TSA table is processed by the FreeMASTER PC Host tool along with all static tables as soon as the communication port is open.

The size of the memory buffer determines the number of TSA entries that can be added dynamically. Depending on the MCU platform, one TSA entry takes either 8 or 16 bytes.

FMSTR_TsaAddVar

Prototype

```
FMSTR_BOOL FMSTR_TsaAddVar(FMSTR_TSATBL_STRPTR tsaName, FMSTR_TSATBL_STRPTR␣
↪tsaType,
    FMSTR_TSATBL_VOIDPTR varAddr, FMSTR_SIZE32 varSize,
    FMSTR_SIZE flags);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_tsa.c*

Arguments

- *tsaName* [in] - name of the object
- *tsaType* [in] - name of the object type
- *varAddr* [in] - address of the object
- *varSize* [in] - size of the object
- *flags* [in] - access flags; a combination of these values:
  - *FMSTR_TSA_INFO_RO_VAR* — read-only memory-mapped object (typically a variable)
  - *FMSTR_TSA_INFO_RW_VAR* — read/write memory-mapped object
  - *FMSTR_TSA_INFO_NON_VAR* — other entry, describing structure types, structure members, enumerations, and other types

Description

This function can be called only when the dynamic TSA table is enabled by the FMSTR_USE_TSA_DYNAMIC configuration option and when the FMSTR_SetUpTsaBuff function call is made to assign the dynamic TSA table memory. This function adds an entry into the dynamic TSA table. It can be used to register a read-only or read/write memory object or describe an item of the user-defined type.

See *TSA table definition* for more details about the TSA table entries.

Application Commands API FMSTR_GetAppCmd

Prototype

```
FMSTR_APPCMD_CODE FMSTR_GetAppCmd(void);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_appcmd.c*

Description

This function can be used to detect if there is an Application Command waiting to be processed by the application. If no command is pending, this function returns the FMSTR_APPCMDRESULT_NOCMD constant. Otherwise, this function returns the code of the Application Command that must be processed. Use the FMSTR_AppCmdAck call to acknowledge the Application Command after it is processed and to return the appropriate result code to the host.

The FMSTR_GetAppCmd function does not report the commands for which a callback handler function exists. If the FMSTR_GetAppCmd function is called when a callback-registered command is pending (and before it is actually processed by the callback function), this function returns FMSTR_APPCMDRESULT_NOCMD.

FMSTR_GetAppCmdData

Prototype

```
FMSTR_APPCMD_PDATA FMSTR_GetAppCmdData(FMSTR_SIZE* dataLen);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_appcmd.c*

Arguments

- *dataLen* [out] - pointer to the variable that receives the length of the data available in the buffer. It can be NULL when this information is not needed.

Description

This function can be used to retrieve the Application Command data when the application determines that an Application Command is pending (see *FMSTR_GetAppCmd*).

There is just a single buffer to hold the Application Command data (the buffer length is FMSTR_APPCMD_BUFF_SIZE bytes). If the data are to be used in the application after the command is processed by the FMSTR_AppCmdAck call, copy the data out to a private buffer.

FMSTR_AppCmdAck

Prototype

```
void FMSTR_AppCmdAck(FMSTR_APPCMD_RESULT resultCode);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_appcmd.c*

Arguments

- *resultCode* [in] - the result code which is to be returned to FreeMASTER

Description

This function is used when the Application Command processing finishes in the application. The resultCode passed to this function is returned back to the host and the driver is re-initialized to expect the next Application Command.

After this function is called and before the next Application Command arrives, the return value of the FMSTR_GetAppCmd function is FMSTR_APPCMDRESULT_NOCMD.

FMSTR_AppCmdSetResponseData

Prototype

```
void FMSTR_AppCmdSetResponseData(FMSTR_ADDR resultDataAddr, FMSTR_SIZE resultDataLen);
```

- Declaration: *freemaster.h*

- Implementation: *freemaster_appcmd.c*

Arguments

- *resultDataAddr* [in] - pointer to the data buffer that is to be copied to the Application Command data buffer
- *resultDataLen* [in] - length of the data to be copied. It must not exceed the FMSTR_APPCMD_BUFF_SIZE value.

Description

This function can be used before the Application Command processing finishes, when there are data to be returned back to the PC.

The response data buffer is copied into the Application Command data buffer, from where it is accessed when the host requires it. Do not use FMSTR_GetAppCmdData and the data buffer after FMSTR_AppCmdSetResponseData is called.

Note: The current version of FreeMASTER does not support the Application Command response data.

FMSTR_RegisterAppCmdCall

Prototype

```
FMSTR_BOOL FMSTR_RegisterAppCmdCall(FMSTR_APPCMD_CODE appCmdCode, FMSTR_
↪PAPPCMDFUNC callbackFunc);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_appcmd.c*

Arguments

- *appCmdCode* [in] - the Application Command code for which the callback is to be registered
- *callbackFunc* [in] - pointer to the callback function that is to be registered. Use NULL to unregister a callback registered previously with this Application Command.

Return value

This function returns a non-zero value when the callback function was successfully registered or unregistered. It can return zero when trying to register a callback function for more than FMSTR_MAX_APPCMD_CALLS different Application Commands.

Description

This function can be used to register the given function as a callback handler for the Application Command. The Application Command is identified using single-byte code. The callback function is invoked automatically by the FreeMASTER driver when the protocol decoder obtains a request to get the application command result code.

The prototype of the callback function is

```
FMSTR_APPCMD_RESULT HandlerFunction(FMSTR_APPCMD_CODE nAppcmd,
    FMSTR_APPCMD_PDATA pData, FMSTR_SIZE nDataLen);
```

Where:

- *nAppcmd* -Application Command code
- *pData* —points to the Application Command data received (if any)

- *nDataLen* —information about the Application Command data length

The return value of the callback function is used as the Application Command Result Code and returned to FreeMASTER.

Note: The FMSTR_MAX_APPCMD_CALLS configuration macro defines how many different Application Commands may be handled by a callback function. When FMSTR_MAX_APPCMD_CALLS is undefined or defined as zero, the FMSTR_RegisterAppCmdCall function always fails.

Pipes API FMSTR_PipeOpen

Prototype

```
FMSTR_HPIPE FMSTR_PipeOpen(FMSTR_PIPE_PORT pipePort, FMSTR_PPIPEFUNC pipeCallback,
↪
    FMSTR_ADDR pipeRxBuff, FMSTR_PIPE_SIZE pipeRxSize,
    FMSTR_ADDR pipeTxBuff, FMSTR_PIPE_SIZE pipeTxSize,
    FMSTR_U8 type, const FMSTR_CHAR *name);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_pipes.c*

Arguments

- *pipePort* [in] - port number that identifies the pipe for the client
- *pipeCallback* [in] - pointer to the callback function that is called whenever a pipe data status changes
- *pipeRxBuff* [in] - address of the receive memory buffer
- *pipeRxSize* [in] - size of the receive memory buffer
- *pipeTxBuff* [in] - address of the transmit memory buffer
- *pipeTxSize* [in] - size of the transmit memory buffer
- *type* [in] - a combination of FMSTR_PIPE_MODE_xxx and FMSTR_PIPE_SIZE_xxx constants describing primary pipe data format and usage. This type helps FreeMASTER decide how to access the pipe by default. Optional, use 0 when undetermined.
- *name* [in] - user name of the pipe port. This name is visible to the FreeMASTER user when creating the graphical pipe interface.

Description

This function initializes a new pipe and makes it ready to accept or send the data to the PC Host client. The receive memory buffer is used to store the received data before they are read out by the FMSTR_PipeRead call. When this buffer gets full, the PC Host client denies the data transmission into this pipe until there is enough free space again. The transmit memory buffer is used to store the data transmitted by the application to the PC Host client using the FMSTR_PipeWrite call. The transmit buffer can get full when the PC Host is disconnected or when it is slow in receiving and reading out the pipe data.

The function returns the pipe handle which must be stored and used in the subsequent calls to manage the pipe object.

The callback function (if specified) is called whenever new data are received through the pipe and available for reading. This callback is also called when the data waiting in the transmit buffer are successfully pushed to the PC Host and the transmit buffer free space increases. The prototype of the callback function provided by the user application must be as follows. The *PipeHandler* name is only a placeholder and must be defined by the application.

```
void PipeHandler(FMSTR_HPIPE pipeHandle);
```

FMSTR_PipeClose

Prototype

```
void FMSTR_PipeClose(FMSTR_HPIPE pipeHandle);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_pipes.c*

Arguments

- *pipeHandle* [in] - pipe handle returned from the FMSTR_PipeOpen function call

Description

This function de-initializes the pipe object. No data can be received or sent on the pipe after this call.

FMSTR_PipeWrite

Prototype

```
FMSTR_PIPE_SIZE FMSTR_PipeWrite(FMSTR_HPIPE pipeHandle, FMSTR_ADDR pipeData,
    FMSTR_PIPE_SIZE pipeDataLen, FMSTR_PIPE_SIZE writeGranularity);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_pipes.c*

Arguments

- *pipeHandle* [in] - pipe handle returned from the FMSTR_PipeOpen function call
- *pipeData* [in] - address of the data to be written
- *pipeDataLen* [in] - length of the data to be written
- *writeGranularity* [in] - size of the minimum unit of data which is to be written

Description

This function puts the user-specified data into the pipe's transmit memory buffer and schedules it for transmission. This function returns the number of bytes that were successfully written into the buffer. This number may be smaller than the number of the requested bytes if there is not enough free space in the transmit buffer.

The *writeGranularity* argument can be used to split the data into smaller chunks, each of the size given by the *writeGranularity* value. The FMSTR_PipeWrite function writes as many data chunks as possible into the transmit buffer and does not attempt to write an incomplete chunk. This feature can prove to be useful to avoid the intermediate caching when writing an array of integer values or other multi-byte data items. When making the nGranularity value equal to the nLength value, all data are considered as one chunk which is either written successfully as a whole or not at all. The nGranularity value of 0 or 1 disables the data-chunk approach.

FMSTR_PipeRead

Prototype

```
FMSTR_PIPE_SIZE FMSTR_PipeRead(FMSTR_HPIPE pipeHandle, FMSTR_ADDR pipeData,
    FMSTR_PIPE_SIZE pipeDataLen, FMSTR_PIPE_SIZE readGranularity);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_pipes.c*

Arguments

- *pipeHandle* [in] - pipe handle returned from the FMSTR_PipeOpen function call
- *pipeData* [in] - address of the data buffer to be filled with the received data
- *pipeDataLen* [in] - length of the data to be read
- *readGranularity* [in] - size of the minimum unit of data which is to be read

Description

This function copies the data received from the pipe from its receive buffer to the user buffer for further processing. The function returns the number of bytes that were successfully copied to the buffer. This number may be smaller than the number of the requested bytes if there is not enough data bytes available in the receive buffer.

The readGranularity argument can be used to copy the data in larger chunks in the same way as described in the FMSTR_PipeWrite function.

API data types This section describes the data types used in the FreeMASTER driver. The information provided here can be useful when modifying or porting the FreeMASTER Communication Driver to new NXP platforms.

Note: The licensing conditions prohibit use of FreeMASTER and the FreeMASTER Communication Driver with non-NXP MPU or MCU products.

Public common types

The table below describes the public data types used in the FreeMASTER driver API calls. The data types are declared in the *freemaster.h* header file.

Type name	Description
FMSTR_ADDR	Data type used to hold the memory address. On most platforms, this is normally a C-pointer, but it may also be a pure integer type.
For example, this type is defined as long integer on the 56F8xxx platform where the 24-bit addresses must be supported, but the C-pointer may be only 16 bits wide in some compiler configurations.	
FMSTR_SIZE	Data type used to hold the memory block size.
It is required that this type is unsigned and at least 16 bits wide integer.	
FMSTR_BOOL	Data type used as a general boolean type.
This type is used only in zero/non-zero conditions in the driver code.	
FMSTR_APPCM	Data type used to hold the Application Command code.
Generally, this is an unsigned 8-bit value.	
FMSTR_APPCM	Data type used to create the Application Command data buffer.
Generally, this is an unsigned 8-bit value.	
FMSTR_APPCM	Data type used to hold the Application Command result code.
Generally, this is an unsigned 8-bit value.	

Public TSA types

The table describes the TSA-specific public data types. These types are declared in the *freemaster_tsa.h* header file, which is included in the user application indirectly by the *freemaster.h* file.

FMSTR_TSA_TI	Data type used to hold a descriptor index in the TSA table or a table index in the list of TSA tables.
By default, this is defined as FMSTR_SIZE.	
FMSTR_TSA_TS	Data type used to hold a memory block size, as used in the TSA descriptors.
By default, this is defined as FMSTR_SIZE.	

Public Pipes types

The table describes the data types used by the FreeMASTER Pipes API:

FMSTR_HPIPE	Pipe handle that identifies the open-pipe object.
Generally, this is a pointer to a void type.	
FMSTR_PIPE_P	Integer type required to hold at least 7 bits of data.
Generally, this is an unsigned 8-bit or 16-bit type.	
FMSTR_PIPE_SI	Integer type required to hold at least 16 bits of data.
This is used to store the data buffer sizes.	
FMSTR_PPIPEF	Pointer to the pipe handler function.
See *FMSTR_PipeOpen* for more details.	

Internal types

The table describes the data types used internally by the FreeMASTER driver. The data types are declared in the platform-specific header file and they are not available in the application code.

FMSTR_U8	The smallest memory entity.
On the vast majority of platforms, this is an unsigned 8-bit integer.	
On the 56F8xx DSP platform, this is defined as an unsigned 16-bit integer.	
FMSTR_U16	Unsigned 16-bit integer.
FMSTR_U32	Unsigned 32-bit integer.
FMSTR_S8	Signed 8-bit integer.
FMSTR_S16	Signed 16-bit integer.
FMSTR_S32	Signed 32-bit integer.
FMSTR_FLOAT	4-byte standard IEEE floating-point type.
FMSTR_FLAGS	Data type forming a union with a structure of flag bit-fields.
FMSTR_SIZE8	Data type holding a general size value, at least 8 bits wide.
FMSTR_INDEX	General for-loop index. Must be signed, at least 16 bits wide.
FMSTR_BCHR	A single character in the communication buffer.
Typically, this is an 8-bit unsigned integer, except for the DSP platforms where it is a 16-bit integer.	
FMSTR_BPTR	A pointer to the communication buffer (an array of FMSTR_BCHR).

Document references

Links

- This document online: https://mcuxpresso.nxp.com/mcuxsdk/latest/html/middleware/freemaster/doc/index.html

- FreeMASTER tool home: www.nxp.com/freemaster
- FreeMASTER community area: community.nxp.com/community/freemaster
- FreeMASTER GitHub code repo: https://github.com/nxp-mcuxpresso/mcux-freemaster
- MCUXpresso SDK home: www.nxp.com/mcuxpresso
- MCUXpresso SDK builder: mcuxpresso.nxp.com/en

Documents

- *FreeMASTER Usage Serial Driver Implementation* (document AN4752)
- *Integrating FreeMASTER Time Debugging Tool With CodeWarrior For Microcontrollers v10.X Project* (document AN4771)
- *Flash Driver Library For MC56F847xx And MC56F827xx DSC Family* (document AN4860)

Revision history This Table summarizes the changes done to this document since the initial release.

Revision	Date	Description
1.0	03/2006	Limited initial release
2.0	09/2007	Updated for FreeMASTER version. New Freescale document template used.
2.1	12/2007	Added description of the new Fast Recorder feature and its API.
2.2	04/2010	Added support for MPC56xx platform, Added new API for use CAN interface.
2.3	04/2011	Added support for Kxx Kinetis platform and MQX operating system.
2.4	06/2011	Serial driver update, adds support for USB CDC interface.
2.5	08/2011	Added Packet Driven BDM interface.
2.7	12/2013	Added FLEXCAN32 interface, byte access and isr callback configuration option.
2.8	06/2014	Removed obsolete license text, see the software package content for up-to-date license.
2.9	03/2015	Update for driver version 1.8.2 and 1.9: FreeMASTER Pipes, TSA Active Content, LIN Transport Layer support, DEBUG-TX communication troubleshooting, Kinetis SDK support.
3.0	08/2016	Update for driver version 2.0: Added support for MPC56xx, MPC57xx, KEAxx and S32Kxx platforms. New NXP document template as well as new license agreement used. added MCAN interface. Folders structure at the installation destination was rearranged.
4.0	04/2019	Update for driver released as part of FreeMASTER v3.0 and MCUXpresso SDK 2.6. Updated to match new V4 serial communication protocol and new configuration options. This version of the document removes substantial portion of outdated information related to S08, S12, ColdFire, Power and other legacy platforms.
4.1	04/2020	Minor update for FreeMASTER driver included in MCUXpresso SDK 2.8.
4.2	09/2020	Added example applications description and information about the MCUXpresso Config Tools. Fixed the pipe-related API description.
4.3	10/2024	Added description of Network and Segger J-Link RTT interface configuration. Accompanying the MCUXpresso SDK version 24.12.00.
4.4	04/2025	Added Zephyr-specific information. Accompanying the MCUXpresso SDK version 25.06.00.

3.2 Multicore SDK

Multicore Software Development Kit (MCSDK) provides comprehensive software support for NXP multicore devices. The MCSDK is combined with the MCUXpresso SDK to form a framework for easy development of multicore applications.

3.2.1 Quick links

- *Release Notes*

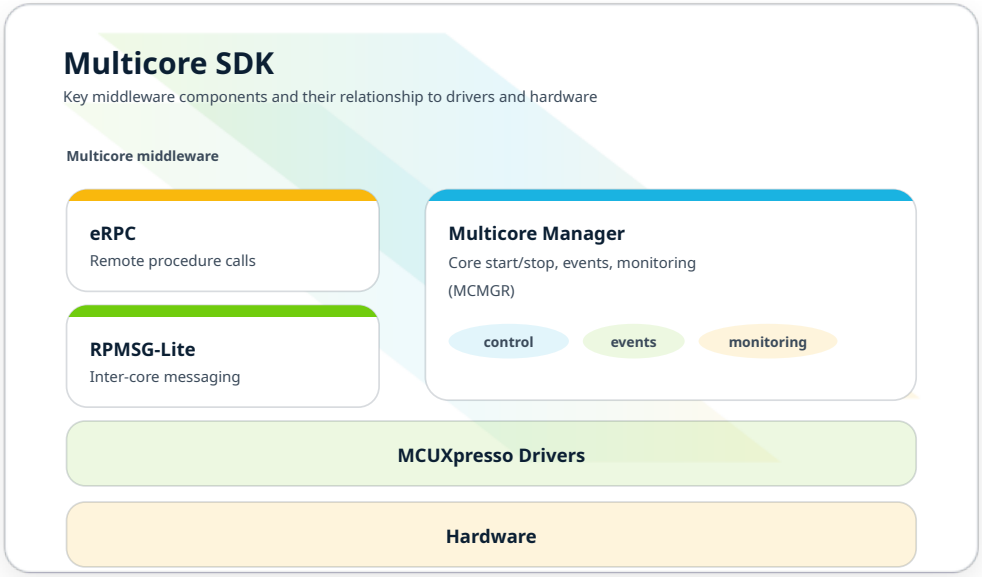


Fig. 1: Multicore SDK software stack (simplified).

- *Getting Started*
- *Multicore SDK Changelog*

3.2.2 Components

Component	Description
RPMSG-Lite	Lightweight implementation of the Remote Processor Messaging (RPMsg) protocol for inter-core messaging.
Multicore Manager	Services for multicore systems (core start/stop, events, and monitoring).
eRPC	Embedded RPC system suitable for multicore and multiprocessor designs.

MCSDK Release Notes

Multicore SDK (MCSDK) Release Notes Overview

These are the release notes for the NXP **Multicore Software Development Kit (MCSDK)**.

Item	Value
MCSDK version	**26.09.00**
Scope	Multicore middleware components and examples
Location in MCUXpresso SDK	<MCUXpressoSDK_install_dir>/middleware/multicore/

The MCSDK is combined with the MCUXpresso SDK to provide a framework for developing multicore applications on NXP devices.

What's new

Component updates

Review detailed changes per component:

- eRPC: *CHANGELOG*
- RPMSG-Lite: *CHANGELOG*
- Multicore Manager (MCMGR): *CHANGELOG*

Supported evaluation boards

Boards with enabled multicore example coverage in this release:

Multicore examples
LPCXpresso55S69FRDM-K32L3A6MIMXRT1170-EVKBMIMXRT1160-EVKMIMXRT1180-EVKMCX-N5XX-EVKMCX-N9XX-EVKFRDM-MCXN947MIMXRT700-EVKKW47-EVKKW47-LOCFRDM-MCXW72MCX-W72-EVKFRDM-IMXRT1186

Development tools

The MCSDK is compiled and tested with the toolchains described in the MCUXpresso SDK documentation:

- https://mcuxpresso.nxp.com/mcuxsdk/latest/html/gsd/installation.html#toolchain

Release contents

This table describes the release contents. Not all MCUXpresso SDK packages contain the full set of these components.

Deliverable	Location
Multicore SDK location (<MCSDK_dir>)	<MCUXpressoSDK_install_dir>/middleware/multicore/
Documentation	<MCSDK_dir>/mcuxsdk-doc/
Embedded Remote Procedure Call (eRPC)	<MCSDK_dir>/erpc/
Multicore Manager (MCMGR)	<MCSDK_dir>/mcmgr/
RPMSG-Lite	<MCSDK_dir>/rpmsg-lite/
Multicore demo applications	<MCUXpressoSDK_install_dir>/examples/multicore_examples/

MCSDK component overview

Together, the MCSDK and the MCUXpresso SDK form a framework for developing software for NXP multicore devices. The MCSDK release includes:

- Embedded Remote Procedure Call (eRPC)
- Multicore Manager (MCMGR)
- Remote Processor Messaging – Lite (RPMSG-Lite)

The MCSDK also includes documentation and demo applications for multicore communication.

Demo applications

The demo applications demonstrate MCSDK component usage on supported development boards.

Examples are located under:

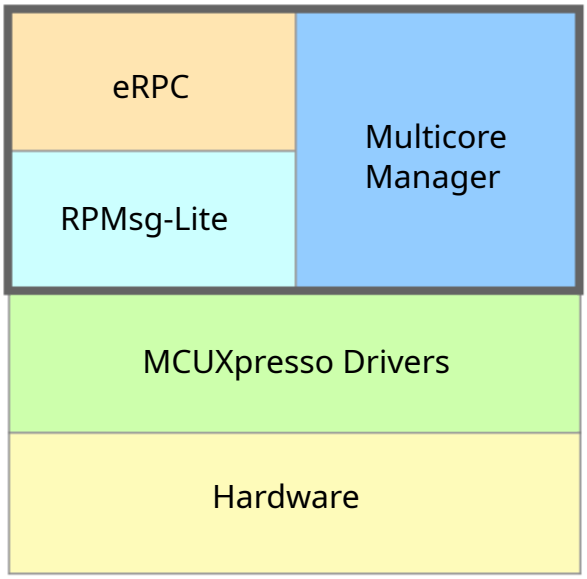
- Multicore: <MCUXpressoSDK_install_dir>/examples/multicore_examples/

Multicore examples
freertos_message_buffershello_worldmulticore_managerrpmsg_lite_pingpongrpmsg_lite_pingpong_rtosrpmsg_lite_

MCSDK Getting Started

Getting Started with Multicore SDK (MCSDK) Overview

The **Multicore Software Development Kit (MCSDK)** provides software components for NXP dual/multicore devices. The MCSDK is combined with the MCUXpresso SDK to provide a framework for developing multicore applications.



All MCSDK-related files are located in:

- <MCUXpressoSDK_install_dir>/middleware/multicore/

For supported toolchain versions, see the release notes.

Quick links

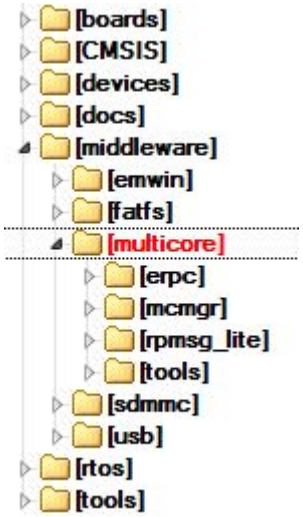
- MCSDK Release Notes
- *MCSDK Changelog*
- Component repositories (within MCSDK):
  - RPMSG-Lite
  - Multicore Manager (MCMGR)
  - eRPC
- eRPC detailed Getting Started guide:

– eRPC Getting Started

MCSDK components

The MCSDK consists of the following components:

- **Embedded Remote Procedure Call (eRPC):** Library + code generator implementing transparent function calls to services running on another core.
- **Multicore Manager (MCMGR):** Core management services (start/stop secondary cores, events, monitoring).
- **Remote Processor Messaging – Lite (RPMSG-Lite):** Lightweight inter-processor messaging library based on RPMsg.



Where things live

Item	Path
MCSDK root	<MCUXpressoSDK_install_dir>/middleware/multicore/
eRPC	<MCUXpressoSDK_install_dir>/middleware/multicore/erpc/
MCMGR	<MCUXpressoSDK_install_dir>/middleware/multicore/mcmgr/
RPMSG-Lite	<MCUXpressoSDK_install_dir>/middleware/multicore/rpmsg-lite/
MCSDK documentation	<MCUXpressoSDK_install_dir>/middleware/multicore/mcuxsdk-doc/
Multicore examples	<MCUXpressoSDK_install_dir>/examples/multicore_examples/

MCSDK demo applications

Multicore example applications are stored together with other MCUXpresso SDK examples.

Location	Folder
Multicore example projects	<MCUXpressoSDK_install_dir>/examples/multicore_examples/<application_name>/

Each example application contains a README describing setup, required images, and runtime behavior.

Inter-Processor Communication (IPC) levels

The MCSDK provides several mechanisms for Inter-Processor Communication (IPC). Pick the lightest one that matches your requirements.

IPC using low-level drivers

NXP multicore SoCs commonly provide dedicated IPC peripherals (Mailbox for some LPC parts, Messaging Unit (MU) for some Kinetis and i.MX parts).

This is the most lightweight approach: using the Mailbox/MU driver API functions, it is possible to pass a value from core to core via registers (scalar or pointer to shared memory) and trigger inter-core interrupts for notifications.

For details about individual driver API functions, see the MCUXpresso SDK API Reference Manual of the specific device.

Messaging mechanism (RPMSG-Lite)

On top of Mailbox/MU drivers, a messaging system can be implemented, allowing messages to be sent between endpoints created on each CPU.

RPMSG-Lite provides this capability and is the preferred MCUXpresso SDK messaging library. It implements ring buffers in shared memory for message exchange.

This costs additional code and data memory, but provides a unified messaging API and helps portability across devices/RTOSes.

Remote procedure calls (eRPC)

To facilitate IPC further, eRPC allows remote function invocation via a simple local function call.

Using different transport layers, it can communicate:

- between individual cores of multicore SoCs (commonly via RPMSG-Lite)
- between separate processors (SPI, UART, TCP/IP)

If the communicating peer is a Linux user-space application, the generated code can be in C/C++ or Python.

The IPC levels demonstrate the scalability of the MCSDK. Based on application needs, different IPC techniques can be used depending on complexity, speed, memory, and system design.

Changelog Multicore SDK

All notable changes to this project will be documented in this file.

The format is based on Keep a Changelog, and this project adheres to Semantic Versioning.

[26.09.00]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.14.0
  - eRPC generator (erpcgen) v1.14.0
  - Multicore Manager (MCMgr) v5.3.0
  - RPMsg-Lite v5.5.0

[26.06.00]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.14.0
  - eRPC generator (erpcgen) v1.14.0

  - Multicore Manager (MCMgr) v5.2.0
  - RPMsg-Lite v5.4.1

[26.03.00]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.14.0
  - eRPC generator (erpcgen) v1.14.0
  - Multicore Manager (MCMgr) v5.1.0
  - RPMsg-Lite v5.4.0

[25.12.00]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.14.0
  - eRPC generator (erpcgen) v1.14.0
  - Multicore Manager (MCMgr) v5.0.2
  - RPMsg-Lite v5.3.0

[25.09.00]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.14.0
  - eRPC generator (erpcgen) v1.14.0
  - Multicore Manager (MCMgr) v5.0.1
  - RPMsg-Lite v5.2.1

[25.06.00]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.14.0
  - eRPC generator (erpcgen) v1.14.0
  - Multicore Manager (MCMgr) v5.0.0
  - RPMsg-Lite v5.2.0

[25.03.00]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.13.0
  - eRPC generator (erpcgen) v1.13.0
  - Multicore Manager (MCMgr) v4.1.7
  - RPMsg-Lite v5.1.4

[24.12.00]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.13.0
  - eRPC generator (erpcgen) v1.13.0
  - Multicore Manager (MCMgr) v4.1.6
  - RPMsg-Lite v5.1.3

[2.16.0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.13.0
  - eRPC generator (erpcgen) v1.13.0
  - Multicore Manager (MCMgr) v4.1.5
  - RPMsg-Lite v5.1.2

[2.15.0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.12.0
  - eRPC generator (erpcgen) v1.12.0
  - Multicore Manager (MCMgr) v4.1.5
  - RPMsg-Lite v5.1.1

[2.14.0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.11.0
  - eRPC generator (erpcgen) v1.11.0
  - Multicore Manager (MCMgr) v4.1.4
  - RPMsg-Lite v5.1.0

[2.13.0_imxrt1180a0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.10.0
  - eRPC generator (erpcgen) v1.10.0
  - Multicore Manager (MCMgr) v4.1.3
  - RPMsg-Lite v5.0.0

[2.13.0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.10.0
  - eRPC generator (erpcgen) v1.10.0
  - Multicore Manager (MCMgr) v4.1.3
  - RPMsg-Lite v5.0.0

[2.12.0_imx93]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.9.1
  - eRPC generator (erpcgen) v1.9.1
  - Multicore Manager (MCMgr) v4.1.2
  - RPMsg-Lite v4.0.1

[2.12.0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.9.1
  - eRPC generator (erpcgen) v1.9.1
  - Multicore Manager (MCMgr) v4.1.2
  - RPMsg-Lite v4.0.0

[2.11.1]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.9.0
  - eRPC generator (erpcgen) v1.9.0
  - Multicore Manager (MCMgr) v4.1.1
  - RPMsg-Lite v3.2.1

[2.11.0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.9.0
  - eRPC generator (erpcgen) v1.9.0
  - Multicore Manager (MCMgr) v4.1.1
  - RPMsg-Lite v3.2.0

[2.10.0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.8.1
  - eRPC generator (erpcgen) v1.8.1
  - Multicore Manager (MCMgr) v4.1.1
  - RPMsg-Lite v3.1.2

[2.9.0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.8.0
  - eRPC generator (erpcgen) v1.8.0
  - Multicore Manager (MCMgr) v4.1.1
  - RPMsg-Lite v3.1.1

[2.8.0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.7.4
  - eRPC generator (erpcgen) v1.7.4
  - Multicore Manager (MCMgr) v4.1.0
  - RPMsg-Lite v3.1.0

[2.7.0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.7.3
  - eRPC generator (erpcgen) v1.7.3
  - Multicore Manager (MCMgr) v4.1.0
  - RPMsg-Lite v3.0.0

[2.6.0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.7.2
  - eRPC generator (erpcgen) v1.7.2
  - Multicore Manager (MCMgr) v4.0.3
  - RPMsg-Lite v2.2.0

[2.5.0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.7.1
  - eRPC generator (erpcgen) v1.7.1
  - Multicore Manager (MCMgr) v4.0.2
  - RPMsg-Lite v2.0.2

[2.4.0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.7.0
  - eRPC generator (erpcgen) v1.7.0
  - Multicore Manager (MCMgr) v4.0.1
  - RPMsg-Lite v2.0.1

[2.3.1]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.6.0
  - eRPC generator (erpcgen) v1.6.0
  - Multicore Manager (MCMgr) v4.0.0
  - RPMsg-Lite v1.2.0

[2.3.0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.5.0
  - eRPC generator (erpcgen) v1.5.0
  - Multicore Manager (MCMgr) v3.0.0
  - RPMsg-Lite v1.2.0

[2.2.0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.4.0
  - eRPC generator (erpcgen) v1.4.0
  - Multicore Manager (MCMgr) v2.0.1
  - RPMsg-Lite v1.1.0

[2.1.0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.3.0
  - eRPC generator (erpcgen) v1.3.0

[2.0.0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.2.0
  - eRPC generator (erpcgen) v1.2.0
  - Multicore Manager (MCMgr) v2.0.0
  - RPMsg-Lite v1.0.0

[1.1.0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.1.0
  - Multicore Manager (MCMgr) v1.1.0
  - Open-AMP / RPMsg based on SHA1 ID 44b5f3c0a6458f3cf80 rev01

[1.0.0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.0.0
  - Multicore Manager (MCMgr) v1.0.0
  - Open-AMP / RPMsg based on SHA1 ID 44b5f3c0a6458f3cf80 rev00

RPMSG-Lite

Quick links

- *Changelog*
- API documentation (Doxygen): https://nxp-mcuxpresso.github.io/rpmsg-lite/

This documentation describes the RPMsg-Lite component, which is a lightweight implementation of the Remote Processor Messaging (RPMsg) protocol. The RPMsg protocol defines a standardized binary interface used to communicate between multiple cores in a heterogeneous multicore system.

Compared to the RPMsg implementation of the Open Asymmetric Multi Processing (OpenAMP) framework (https://github.com/OpenAMP/open-amp), the RPMsg-Lite offers a code size reduction, API simplification, and improved modularity. On smaller Cortex-M0+ based systems, it is recommended to use RPMsg-Lite.

The RPMsg-Lite is an open-source component developed by NXP Semiconductors and released under the BSD-compatible license.

For overview please read RPMSG-Lite VirtIO Overview.

For RPMSG-Lite Design Considerations please read RPMSG-Lite Design Considerations.

Motivation to create RPMsg-Lite

There are multiple reasons why RPMsg-Lite was developed. One reason is the need for the small footprint of the RPMsg protocol-compatible communication component, another reason is the simplification of extensive API of OpenAMP RPMsg implementation.

RPMsg protocol was not documented, and its only definition was given by the Linux Kernel and legacy OpenAMP implementations. This has changed with [1] which is a standardization protocol allowing multiple different implementations to coexist and still be mutually compatible.

Small MCU-based systems often do not implement dynamic memory allocation. The creation of static API in RPMsg-Lite enables another reduction of resource usage. Not only does the dynamic allocation adds another 5 KB of code size, but also communication is slower and less deterministic, which is a property introduced by dynamic memory. The following table shows some rough comparison data between the OpenAMP RPMsg implementation and new RPMsg-Lite implementation:

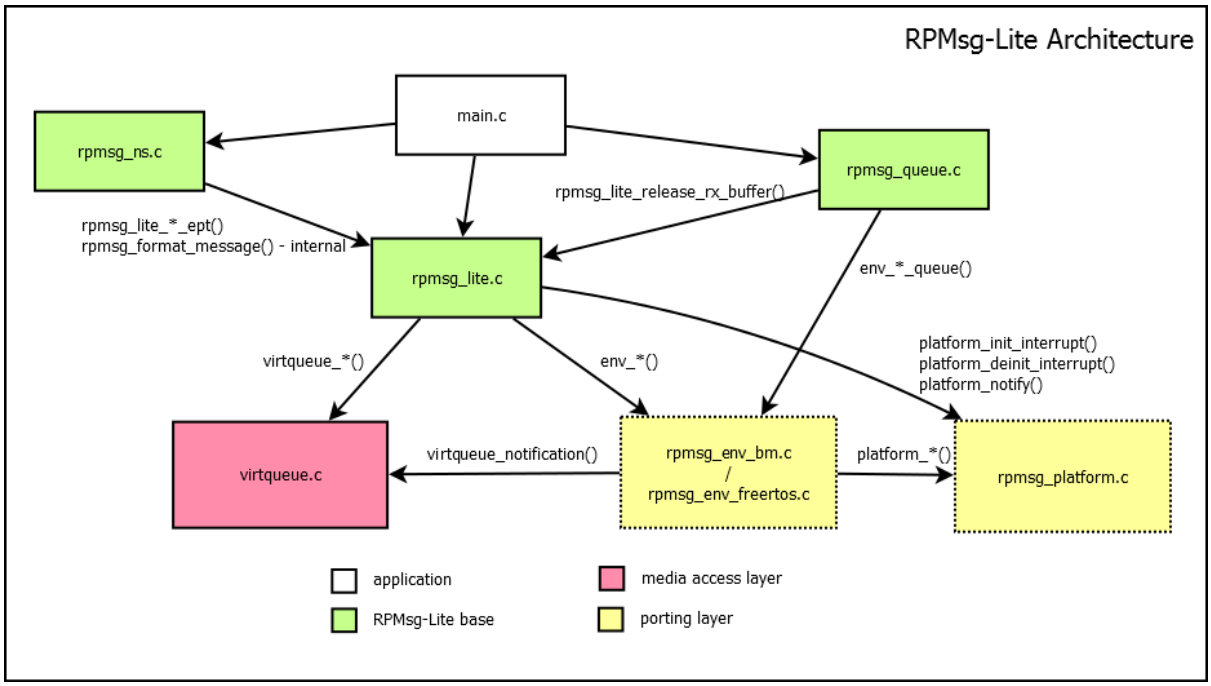
Component / Configuration	Flash [B]	RAM [B]
OpenAMP RPMsg / Release (reference)	5547	456 + dynamic
RPMsg-Lite / Dynamic API, Release	3462	56 + dynamic
Relative Difference [%]	~62.4%	~12.3%
RPMsg-Lite / Static API (no malloc), Release	2926	352
Relative Difference [%]	~52.7%	~77.2%

Implementation

The implementation of RPMsg-Lite can be divided into three sub-components, from which two are optional. The core component is situated in rpmsg_lite.c. Two optional components are used to implement a blocking receive API (in rpmsg_queue.c) and dynamic "named" endpoint creation and deletion announcement service (in rpmsg_ns.c).

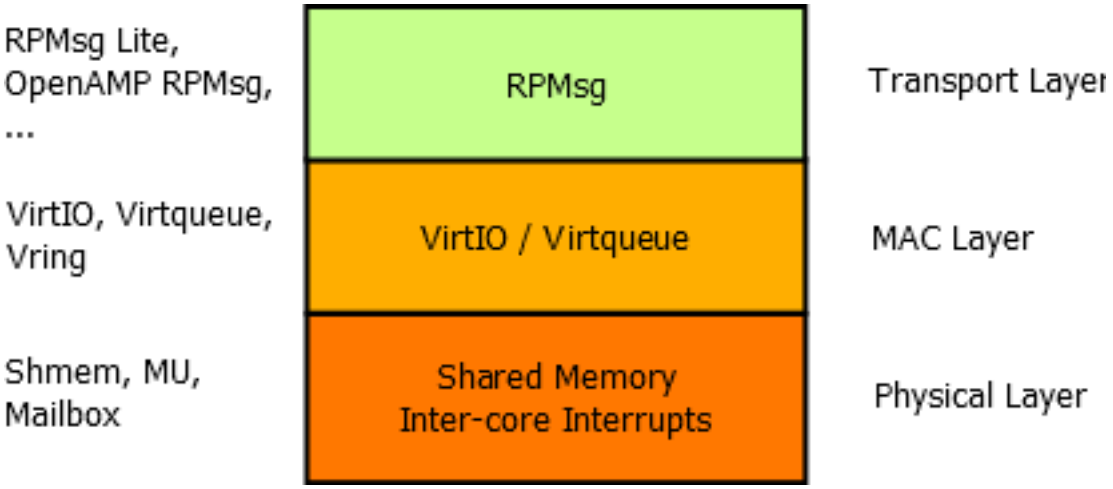
The actual "media access" layer is implemented in virtqueue.c, which is one of the few files shared with the OpenAMP implementation. This layer mainly defines the shared memory model, and internally defines used components such as vring or virtqueue.

The porting layer is split into two sub-layers: the environment layer and the platform layer. The first sublayer is to be implemented separately for each environment. (The bare metal environment already exists and is implemented in rpmsg_env_bm.c, and the FreeRTOS environment is implemented in rpmsg_env_freertos.c etc.) Only the source file, which matches the used environment, is included in the target application project. The second sublayer is implemented in rpmsg_platform.c and defines low-level functions for interrupt enabling, disabling, and triggering mainly. The situation is described in the following figure:



RPMsg-Lite core sub-component

This subcomponent implements a blocking send API and callback-based receive API. The RPMsg protocol is part of the transport layer. This is realized by using so-called endpoints. Each endpoint can be assigned a different receive callback function. However, it is important to notice that the callback is executed in an interrupt environment in current design. Therefore, certain actions like memory allocation are discouraged to execute in the callback. The following figure shows the role of RPMsg in an ISO/OSI-like layered model:



Queue sub-component (optional)

This subcomponent is optional and requires implementation of the env_*_queue() functions in the environment porting layer. It uses a blocking receive API, which is common in RTOS-environments. It supports both copy and nocopy blocking receive functions.

Name Service sub-component (optional)

This subcomponent is a minimum implementation of the name service which is present in the Linux Kernel implementation of RPMsg. It allows the communicating node both to send announcements about "named" endpoint (in other words, channel) creation or deletion and to receive these announcement taking any user-defined action in an application callback. The endpoint address used to receive name service announcements is arbitrarily fixed to be 53 (0x35).

Usage

The application should put the /rpmsg_lite/lib/include directory to the include path and in the application, include either the rpmsg_lite.h header file, or optionally also include the rpmsg_queue.h and/or rpmsg_ns.h files. Both porting sublayers should be provided for you by NXP, but if you plan to use your own RTOS, all you need to do is to implement your own environment layer (in other words, rpmsg_env_myrtos.c) and to include it in the project build.

The initialization of the stack is done by calling the rpmsg_lite_master_init() on the master side and the rpmsg_lite_remote_init() on the remote side. This initialization function must be called prior to any RPMsg-Lite API call. After the init, it is wise to create a communication endpoint, otherwise communication is not possible. This can be done by calling the rpmsg_lite_create_ept() function. It optionally accepts a last argument, where an internal context of the endpoint is created, just in case the RL_USE_STATIC_API option is set to 1. If not, the stack internally calls env_alloc() to allocate dynamic memory for it. In case a callback-based receiving is to be used, an ISR-callback is registered to each new endpoint with user-defined callback data pointer. If a blocking receive is desired (in case of RTOS environment), the rpmsg_queue_create() function must be called before calling rpmsg_lite_create_ept(). The queue handle is passed to the endpoint creation function as a callback data argument and the callback function is set to rpmsg_queue_rx_cb(). Then, it is possible to use rpmsg_queue_receive() function to listen on a queue object for incoming messages. The rpmsg_lite_send() function is used to send messages to the other side.

The RPMsg-Lite also implements no-copy mechanisms for both sending and receiving operations. These methods require specifics that have to be considered when used in an application.

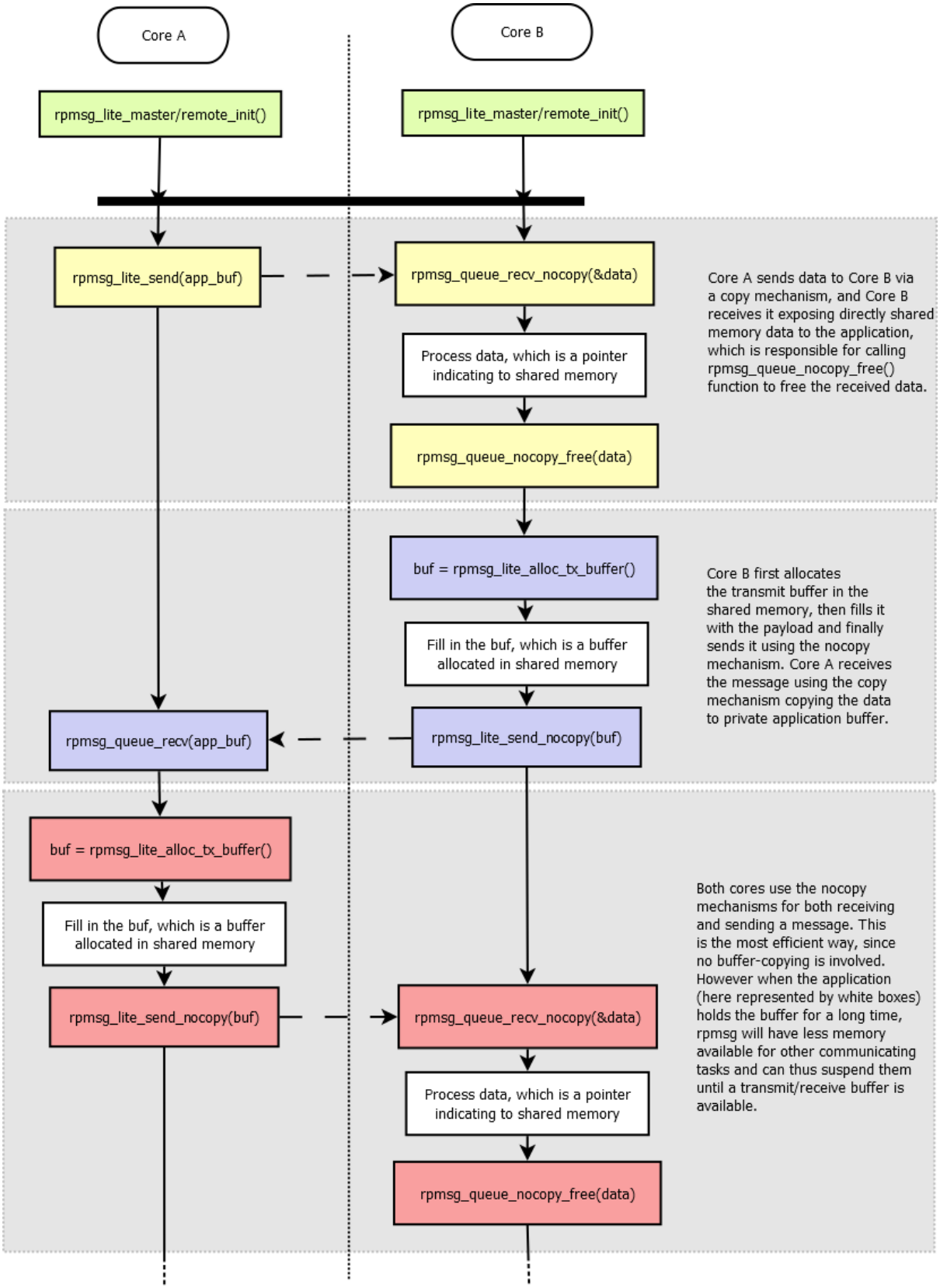
no-copy-send mechanism: This mechanism allows sending messages without the cost for copying data from the application buffer to the RPMsg/virtio buffer in the shared memory. The sequence of no-copy sending steps to be performed is as follows:

- Call the rpmsg_lite_alloc_tx_buffer() function to get the virtio buffer and provide the buffer pointer to the application.
- Fill the data to be sent into the pre-allocated virtio buffer. Ensure that the filled data does not exceed the buffer size (provided as the rpmsg_lite_alloc_tx_buffer() size output parameter).
- Call the rpmsg_lite_send_nocopy() function to send the message to the destination endpoint. Consider the cache functionality and the virtio buffer alignment. See the rpmsg_lite_send_nocopy() function description below.

no-copy-receive mechanism: This mechanism allows reading messages without the cost for copying data from the virtio buffer in the shared memory to the application buffer. The sequence of no-copy receiving steps to be performed is as follows:

- Call the rpmsg_queue_recv_nocopy() function to get the virtio buffer pointer to the received data.
- Read received data directly from the shared memory.
- Call the rpmsg_queue_nocopy_free() function to release the virtio buffer and to make it available for the next data transfer.

The user is responsible for destroying any RPMsg-Lite objects he has created in case of deinitialization. In order to do this, the function rpmsg_queue_destroy() is used to destroy a queue, rpmsg_lite_destroy_ept() is used to destroy an endpoint and finally, rpmsg_lite_deinit() is used to deinitialize the RPMsg-Lite intercore communication stack. Deinitialize all endpoints using a queue before deinitializing the queue. Otherwise, you are actively invalidating the used queue handle, which is not allowed. RPMsg-Lite does not check this internally, since its main aim is to be lightweight.



Examples

RPMsg_Lite multicore examples are part of NXP MCUXpressoSDK packages. Visit https://mcuxpresso.nxp.com to configure, build and download these packages. To get the board list with multicore support (RPMsg_Lite included) use filtering based on Middleware and search for

'multicore' string. Once the selected package with the multicore middleware is downloaded, see

<MCUXpressoSDK_install_dir>/boards/<board_name>/multicore_examples for RPMsg_Lite multicore examples with 'rpmsg_lite_' name prefix.

Another way of getting NXP MCUXpressoSDK RPMsg_Lite multicore examples is using the mcuxsdk-manifests Github repo. Follow the description how to use the West tool to clone and update the mcuxsdk-manifests repo in readme section. Once done the armgcc rpmsg_lite examples can be found in

mcuxsdk/examples/_<board_name>/multicore_examples

You can use the evkmimxrt1170 as the board_name for instance. Similar to MCUXpressoSDK packages the RPMsg_Lite examples use the 'rpmsg_lite_' name prefix.

Notes

Environment layers implementation

Several environment layers are provided in lib/rpmsg_lite/porting/environment folder. Not all of them are fully tested however. Here is the list of environment layers that passed testing:

- rpmsg_env_bm.c
- rpmsg_env_freertos.c
- rpmsg_env_xos.c
- rpmsg_env_threadx.c

The rest of environment layers has been created and used in some experimental projects, it has been running well at the time of creation but due to the lack of unit testing there is no guarantee it is still fully functional.

Shared memory configuration

It is important to correctly initialize/configure the shared memory for data exchange in the application. The shared memory must be accessible from both the master and the remote core and it needs to be configured as Non-Cacheable memory. Dedicated shared memory section in liker file is also a good practise, it is recommended to use linker files from MCUXpressSDK packages for NXP devices based applications. It needs to be ensured no other application part/component is unintentionally accessing this part of memory.

Configuration options

The RPMsg-Lite can be configured at the compile time. The default configuration is defined in the rpmsg_default_config.h header file. This configuration can be customized by the user by including rpmsg_config.h file with custom settings. The following table summarizes all possible RPMsg-Lite configuration options.

Configuration option	Default value	Usage
RL_MS_PE	(1)	Delay in milliseconds used in non-blocking API functions for polling.
RL_BUFFE	(496)	Size of the buffer payload, it must be more than 1 byte, and has to be word align (including rpmsg header size 16 bytes), if not it will be aligned up
RL_BUFFE	(2)	Number of the buffers, it must be power of two (2, 4, ...)
RL_API_H	(1)	Zero-copy API functions enabled/disabled.
RL_USE_S	(0)	Static API functions (no dynamic allocation) enabled/disabled.
RL_USE_D	(0)	Memory cache management of shared memory. Use in case of data cache is enabled for shared memory.
RL_CLEAF	(0)	Clearing used buffers before returning back to the pool of free buffers enabled/disabled.
RL_USE_M	(0)	When enabled IPC interrupts are managed by the Multicore Manager (IPC interrupts router), when disabled RPMsg-Lite manages IPC interrupts by itself.
RL_USE_E	(0)	When enabled the environment layer uses its own context. Required for some environments (QNX). The default value is 0 (no context, saves some RAM).
RL_DEBU(	(0)	When enabled buffer pointers passed to rpmsg_lite_send_nocopy() and rpmsg_lite_release_rx_buffer() functions (enabled by RL_API_HAS_ZEROCOPY config) are checked to avoid passing invalid buffer pointer. The default value is 0 (disabled). Do not use in RPMsg-Lite to Linux configuration.
RL_ALLO\	(0)	When enabled the opposite side is notified each time received buffers are consumed and put into the queue of available buffers. Enable this option in RPMsg-Lite to Linux configuration to allow unblocking of the Linux blocking send. The default value is 0 (RPMsg-Lite to RPMsg-Lite communication).
RL_ALLO\	(0)	It allows to define custom shared memory configuration and replacing the shared memory related global settings from rpmsg_config.h This is useful when multiple instances are running in parallel but different shared memory arrangement (vring size & alignment, buffers size & count) is required. The default value is 0 (all RPMsg_Lite instances use the same shared memory arrangement as defined by common config macros).
RL_ASSER	see rpmsg	Assert implementation.

How to format rpmsg-lite code

To format code, use the application developed by Google, named *clang-format*. This tool is part of the llvm project. Currently, the clang-format 10.0.0 version is used for rpmsg-lite. The set of style settings used for clang-format is defined in the `.clang-format` file, placed in a root of the rpmsg-lite directory where Python script `run_clang_format.py` can be executed. This script executes the application named *clang-format.exe*. You need to have the path of this application in the OS's environment path, or you need to change the script.

References

[1] M. Novak, M. Cingel, Lockless Shared Memory Based Multicore Communication Protocol

Copyright © 2016 Freescale Semiconductor, Inc. Copyright © 2016-2025 NXP

Changelog RPMSG-Lite

All notable changes to this project will be documented in this file.

The format is based on Keep a Changelog, and this project adheres to Semantic Versioning.

v5.5.0

Added

- Added MCXE32B dual Cortex-M7 parts support.
- Added rpmsg_lite_are_all_buffers_consumed() API to report when all locally allocated TX buffers were consumed by the peer and returned to the local free buffer pool, with test coverage in tests/03_send_receive for both sides.
- Added i.MX937 parts support
- Removed obsolete unused k32w1 platform layer

v5.4.1

Fixed

- Corrected isr_counter3 handling in RT700 porting layer platform_deinit_interrupt implementation.

v5.4.0

Added

- Added i.MX93W parts support
- Added i.MX952 parts support
- Added mcxl14x parts support
- Added mcxw70 parts support
- Zephyr rpmsg-lite samples add support for RT600 and RT700 DSP cores

Changed

- Port platform_get_custom_shmem_config change return value from uint32_t to int32_t for error handling consistency.

Fixed

- Updated unit tests code to allow correct GCOV code coverage measurement.
- Added missing dsb instructions into all ISRs for cm4/cm7 based platform layers.
- Fixed CERT EXP34-C by adding __attribute__((noreturn)) to RL_HANG(void) function.
- **MISRA C-2012 Rule 14.3 compliance**: Simplified RL_WORD_ALIGN_UP and RL_WORD_ALIGN_DOWN macros to eliminate invariant controlling expression violations (CID 5011148)
  - Replaced ternary operator implementations with standard bitwise alignment idioms
  - Fixed hardcoded 4U constant in RL_WORD_ALIGN_UP to use RL_WORD_SIZE for better portability
- Fixed several MISRA compliance violations

v5.3.0

Added

- RT700 porting layer added support to send rpmsg messages between CM33_0 <-> Hifi1 and CM33_1 <-> Hifi4 cores.
- Add new platform macro RL_PLATFORM_MAX_ISR_COUNT this will set number of IRQ count per platform. This macro is then used in environment layers to set isr_table size where irq handles are registered. It size should match the bit length of VQ_ID so all combinations can fit into table.
- Unit tests updated to improve code coverage, new unit tests added covering static allocations in rtos environment layers.

Fixed

- virtio.h removed typedef uint8_t boolean and in its place use standard C99 bool type to avoid potential type conflicts.
- env_acquire_sync_lock() and env_release_sync_lock() synchronization primitives removed
- Kconfig consolidation, when RL_ALLOW_CUSTOM_SHMEM_CONFIG enabled the platform_get_custom_shmem_config() function needs to be implemented in platform layer to provide custom shared memory configuration for RPMsg-Lite instance.

v5.2.1

Added

- Doc added RPMSG-Lite VirtIO Overview
- Doc added RPSMG-Lite Design Considerations
- Added frdmimxrt1186 unit testing

Changed

- Remove limitation that RL_BUFFER_SIZE needs to be power of 2. It just has to be more than 16 bytes, e.g. 16 bytes of rpmsg header and payload size at least 1 byte and word aligned, if not it will be aligned up.

Fixed

- Fixed CERT-C INT31-C violation in platform_notify function in rpmsg_platform.c for imxrt700_m33, imxrt700_hifi4, imxrt700_hifi1 platforms

v5.2.0

Added

- Add MCXL20 porting layer and unit testing
- New utility macro RL_CALCULATE_BUFFER_COUNT_DOWN_SAFE to safely determine maximum buffer count within shared memory while preventing integer underflow.
- RT700 platform add support for MCMGR in DSPs

Changed

- Change rpmsg_platform.c to support new MCMGR API
- Improved input validation in initialization functions to properly handle insufficient memory size conditions.
- Refactored repeated buffer count calculation pattern for better code maintainability.

- To make sure that remote has already registered IRQ there is required App level IPC mechanism to notify master about it

Fixed

- Fixed env_wait_for_link_up function to handle timeout in link state checks for baremetal and qnx environment, RL_BLOCK mode can be used to wait indefinitely.
- Fixed CERT-C INT31-C violation by adding compile-time check to ensure RL_PLATFORM_HIGHEST_LINK_ID remains within safe range for 16-bit casting in virtqueue ID creation.
- Fixed CERT-C INT30-C violations by adding protection against unsigned integer underflow in shared memory calculations, specifically in shmem_length - (uint32_t)RL_VRING_OVERHEAD and shmem_length - 2U * shmem_config.vring_size expressions.
- Fixed CERT INT31-C violation in platform_interrupt_disable() and similar functions by replacing unsafe cast from uint32_t to int32_t with a return of 0 constant.
- Fixed unsigned integer underflow in rpmsg_lite_alloc_tx_buffer() where subtracting header size from buffer size could wrap around if buffer was too small, potentially leading to incorrect buffer sizing.
- Fixed CERT-C INT31-C violation in rpmsg_lite.c where size parameter was cast from uint32_t to uint16_t without proper validation.
  - Applied consistent masking approach to both size and flags parameters: (uint16_t)(value & 0xFFFFU).
  - This fix prevents potential data loss when size values exceed 65535.
- Fixed CERT INT31-C violation in env_memset functions by explicitly converting int32_t values to unsigned char using bit masking. This prevents potential data loss or misinterpretation when passing values outside the unsigned char range (0-255) to the standard memset() function.
- Fixed CERT-C INT31-C violations in RPMsg-Lite environment porting: Added validation checks for signed-to-unsigned integer conversions to prevent data loss and misinterpretation.
  - rpmsg_env_freertos.c: Added validation before converting int32_t to UBaseType_t.
  - rpmsg_env_qnx.c: Fixed format string and added validation before assigning to mqstat fields.
  - rpmsg_env_threadx.c: Added validation to prevent integer overflow and negative values.
  - rpmsg_env_xos.c: Added range checking before casting to uint16_t.
  - rpmsg_env_zephyr.c: Added validation before passing values to k_msgq_init.
- Fixed a CERT INT31-C compliance issue in env_get_current_queue_size() function where an unsigned queue count was cast to a signed int32_t without proper validation, which could lead to lost or misinterpreted data if queue size exceeded INT32_MAX.
- Fixed CERT INT31-C violation in rpmsg_platform.c where memcmp() return value (signed int) was compared with unsigned constant without proper type handling.
- Fixed CERT INT31-C violation in rpmsg_platform.c where casting from uint32_t to uint16_t could potentially result in data loss. Changed length variable type from uint16_t to uint32_t to properly handle memory address differences without truncation.
- Fixed potential integer overflow in env_sleep_msec() function in ThreadX environment implementation by rearranging calculation order in the sleep duration formula.

- Fixed CERT-C INT31-C violation in RPMsg-Lite where bitwise NOT operations on integer constants were performed in signed integer context before being cast to unsigned. This could potentially lead to misinterpreted data on imx943 platform.
- Added RL_MAX_BUFFER_COUNT (32768U) and RL_MAX_VRING_ALIGN (65536U) limit to ensure alignment values cannot contribute to integer overflow
- Fixed CERT INT31-C violation in vring_need_event(), added cast to uint16_t for each operand.

v5.1.4 - 27-Mar-2025

Added

- Add KW43B43 porting layer

Changed

- Doxygen bump to version 1.9.6

v5.1.3 - 13-Jan-2025

Added

- Memory cache management of shared memory. Enable with #define RL_USE_DCACHE (1) in rpmsg_config.h in case of data cache is used.
- Cmake/Kconfig support added.
- Porting layers for imx95, imxrt700, mcmxw71x, mcmxw72x, kw47b42 added.

v5.1.2 - 08-Jul-2024

Changed

- Zephyr-related changes.
- Minor Misra corrections.

v5.1.1 - 19-Jan-2024

Added

- Test suite provided.
- Zephyr support added.

Changed

- Minor changes in platform and env. layers, minor test code updates.

v5.1.0 - 02-Aug-2023

Added

- RPMsg-Lite: Added aarch64 support.

Changed

- RPMsg-Lite: Increased the queue size to (2 * RL_BUFFER_COUNT) to cover zero copy cases.

- Code formatting using LLVM16.

Fixed

- Resolved issues in ThreadX env. layer implementation.

v5.0.0 - 19-Jan-2023

Added

- Timeout parameter added to rpmsg_lite_wait_for_link_up API function.

Changed

- Improved debug check buffers implementation - instead of checking the pointer fits into shared memory check the presence in the VirtIO ring descriptors list.
- VRING_SIZE is set based on number of used buffers now (as calculated in vring_init) - updated for all platforms that are not communicating to Linux rpmsg counterpart.

Fixed

- Fixed wrong RL_VRING_OVERHEAD macro comment in platform.h files
- Misra corrections.

v4.0.0 - 20-Jun-2022

Added

- Added support for custom shared memory arrangement per the RPMsg_Lite instance.
- Introduced new rpmsg_lite_wait_for_link_up() API function - this allows to avoid using busy loops in rtos environments, GitHub PR #21.

Changed

- Adjusted rpmsg_lite_is_link_up() to return RL_TRUE/RL_FALSE.

v3.2.0 - 17-Jan-2022

Added

- Added support for i.MX8 MP multicore platform.

Changed

- Improved static allocations - allow OS-specific objects being allocated statically, GitHub PR #14.
- Aligned rpmsg_env_xos.c and some platform layers to latest static allocation support.

Fixed

- Minor Misra and typo corrections, GitHub PR #19, #20.

v3.1.2 - 16-Jul-2021

Added

- Addressed MISRA 21.6 rule violation in rpmsg_env.h (use SDK's PRINTF in MCUXpressoSDK examples, otherwise stdio printf is used).
- Added environment layers for XOS.
- Added support for i.MX RT500, i.MX RT1160 and i.MX RT1170 multicore platforms.

Fixed

- Fixed incorrect description of the rpmsg_lite_get_endpoint_from_addr function.

Changed

- Updated RL_BUFFER_COUNT documentation (issue #10).
- Updated imxrt600_hifi4 platform layer.

v3.1.1 - 15-Jan-2021

Added

- Introduced RL_ALLOW_CONSUMED_BUFFERS_NOTIFICATION config option to allow opposite side notification sending each time received buffers are consumed and put into the queue of available buffers.
- Added environment layers for Threadx.
- Added support for i.MX8QM multicore platform.

Changed

- Several MISRA C-2012 violations addressed.

v3.1.0 - 22-Jul-2020

Added

- Added support for several new multicore platforms.

Fixed

- MISRA C-2012 violations fixed (7.4).
- Fixed missing lock in rpmsg_lite_rx_callback() for QNX env.
- Correction of rpmsg_lite_instance structure members description.
- Address -Waddress-of-packed-member warnings in GCC9.

Changed

- Clang update to v10.0.0, code re-formatted.

v3.0.0 - 20-Dec-2019

Added

- Added support for several new multicore platforms.

Fixed

- MISRA C-2012 violations fixed, incl. data types consolidation.

- Code formatted.

v2.2.0 - 20-Mar-2019

Added

- Added configuration macro RL_DEBUG_CHECK_BUFFERS.
- Several MISRA violations fixed.
- Added environment layers for QNX and Zephyr.
- Allow environment context required for some environment (controlled by the RL_USE_ENVIRONMENT_CONTEXT configuration macro).
- Data types consolidation.

v1.1.0 - 28-Apr-2017

Added

- Supporting i.MX6SX and i.MX7D MPU platforms.
- Supporting LPC5411x MCU platform.
- Baremental and FreeRTOS support.
- Support of copy and zero-copy transfer.
- Support of static API (without dynamic allocations).

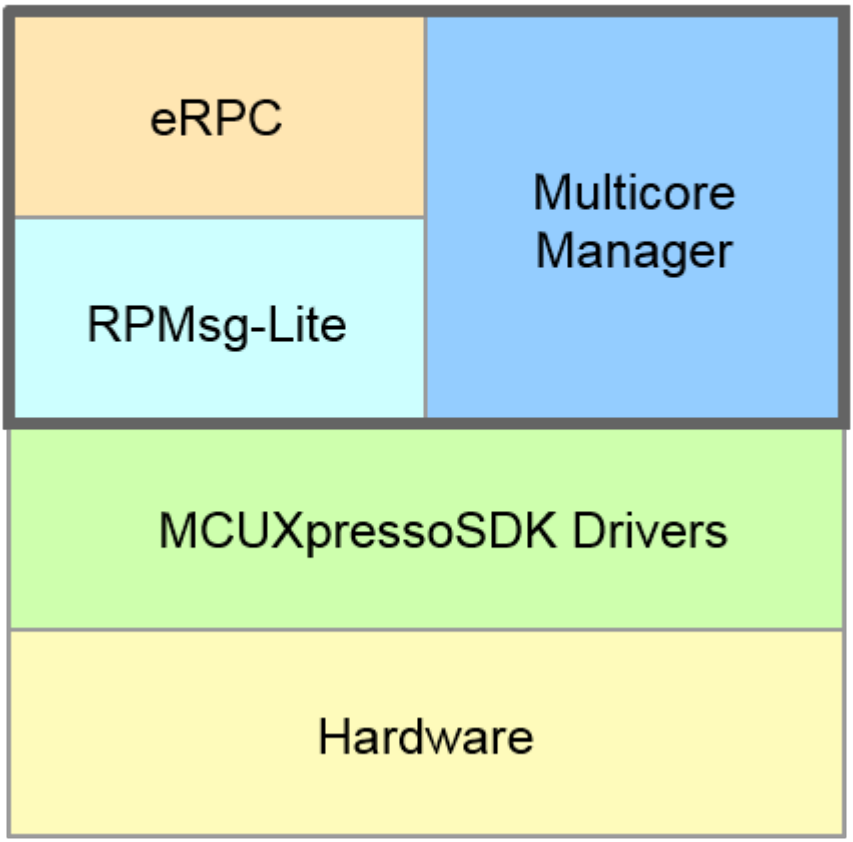
Multicore Manager (MCMGR)

Quick links

- *Changelog*
- API documentation (Doxygen): https://nxp-mcuxpresso.github.io/mcux-mcmgr/

The Multicore Manager (MCMGR) software library provides a number of services for multicore systems. This library is distributed as a part of the Multicore SDK (MCSDK). Together, the MCSDK and the MCUXpresso SDK (SDK) form a framework for development of software for NXP multicore devices.

The MCMGR component is located in the <MCUXpressoSDK_install_dir>/middleware/multicore/mcmgr directory.



The Multicore Manager provides the following major functions:

- Maintains information about all cores in system.
- Secondary/auxiliary core(s) startup and shutdown.
- Remote core monitoring and event handling.

Usage of the MCMGR software component

The main use case of MCMGR is the secondary/auxiliary core start. This functionality is performed by the public API function.

Example of MCMGR usage to start secondary core:

```
#include "mcmgr.h"

void main()
{
    /* Initialize MCMGR, low level multicore management library */
    MCMGR_Init();

    /* Boot secondary core application from the CORE1_BOOT_ADDRESS, pass "1" as startup data,␣
↪starting synchronously. */
    MCMGR_StartCore(kMCMGR_Core1, CORE1_BOOT_ADDRESS, 1, kMCMGR_Start_Synchronous);
```

(continues on next page)

(continued from previous page)

```
.
.
.
    /* Stop secondary core execution. */
    MCMGR_StopCore(kMCMGR_Core1);
}
```

Some platforms allow stopping and re-starting the secondary core application again, using the MCMGR_StopCore / MCMGR_StartCore API calls. It is necessary to ensure the initially loaded image is not corrupted before re-starting, especially if it deals with the RAM target. Cache coherence has to be considered/ensured as well.

It could also happen that the secondary core application stops running correctly and the primary core application does not know about that situation. Therefore, it is beneficial to implement a mechanism for core health monitoring. The test_heartbeat unit test can serve as an example how to ensure that: secondary core could periodically send heartbeat signals to the primary core using MCMGR_TriggerEvent() API to indicate that it is alive and functioning properly.

Another important MCMGR feature is the ability for remote core monitoring and handling of events such as reset, exception, and application events. Application-specific callback functions for events are registered by the MCMGR_RegisterEvent() API. Triggering these events is done using the MCMGR_TriggerEvent() API. mcmgr_event_type_t enums all possible event types.

An example of MCMGR usage for remote core monitoring and event handling. Code for the primary side:

```
#include "mcmgr.h"

#define APP_RPMSG_READY_EVENT_DATA  (1)
#define APP_NUMBER_OF_CORES (2)
#define APP_SECONDARY_CORE kMCMGR_Core1

/* Callback function registered via the MCMGR_RegisterEvent() and triggered by MCMGR_TriggerEvent()␣
↪called on the secondary core side */
void RPMsgRemoteReadyEventHandler(mcmgr_core_t coreNum, uint16_t eventData, void *context)
{
    uint16_t *data = &((uint16_t *)context)[coreNum];

    *data = eventData;
}

void main()
{
    uint16_t RPMsgRemoteReadyEventData[NUMBER_OF_CORES] = {0};

    /* Initialize MCMGR, low level multicore management library */
    MCMGR_Init();

    /* Register the application event before starting the secondary core */
    MCMGR_RegisterEvent(kMCMGR_RemoteApplicationEvent, RPMsgRemoteReadyEventHandler, (void␣
↪*)RPMsgRemoteReadyEventData);

    /* Boot secondary core application from the CORE1_BOOT_ADDRESS, pass rpmsg_lite_base address␣
↪as startup data, starting synchronously. */
    MCMGR_StartCore(APP_SECONDARY_CORE, CORE1_BOOT_ADDRESS, (uint32_t)rpmsg_lite_
↪base, kMCMGR_Start_Synchronous);

    /* Wait until the secondary core application signals the rpmsg remote has been initialized and is ready to␣
↪communicate. */
    while(APP_RPMSG_READY_EVENT_DATA != RPMsgRemoteReadyEventData[APP_SECONDARY_
↪CORE]) {};
```

(continues on next page)

(continued from previous page)

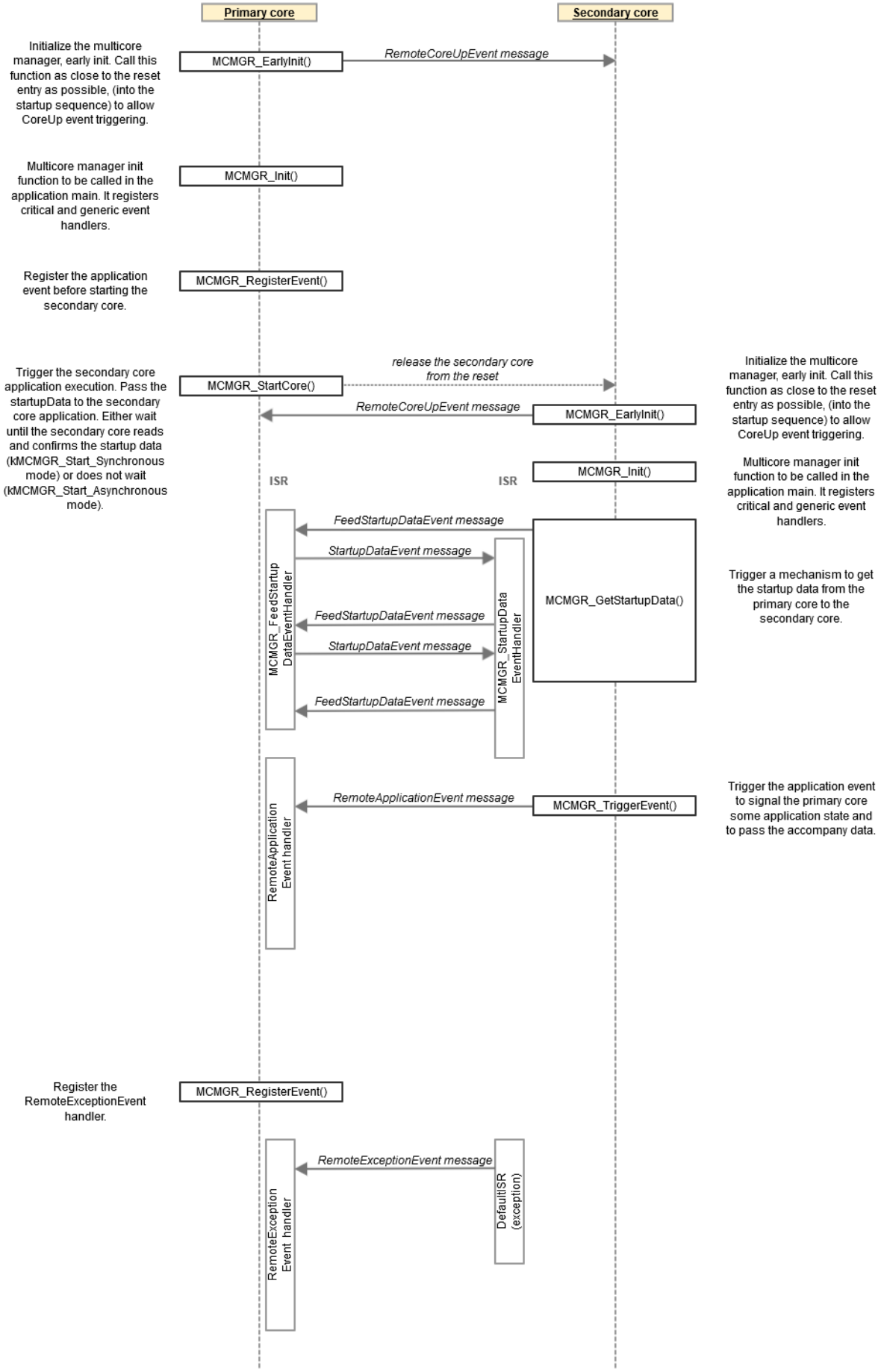
```
.
.
.
}
```

Code for the secondary side:

```
#include "mcmgr.h"

#define APP_RPMSG_READY_EVENT_DATA  (1)

void main()
{
    /* Initialize MCMGR, low level multicore management library */
    MCMGR_Init();
.
.
.

    /* Signal the to other core that we are ready by triggering the event and passing the APP_RPMSG_
↪READY_EVENT_DATA */
    MCMGR_TriggerEvent(kMCMGR_Core0, kMCMGR_RemoteApplicationEvent, APP_RPMSG_
↪READY_EVENT_DATA);
.
.
.
}
```

MCMGR Data Exchange Diagram

The following picture shows how the handshakes are supposed to work between the two cores in the MCMGR software.


Primary core
Secondary core
Initialize the multicore manager, early init. Call this function as close to the reset entry as possible, (into the startup sequence) to allow CoreUp event triggering.
MCMGR_EarlyInit()
RemoteCoreUpEvent message
Multicore manager init function to be called in the application main. It registers critical and generic event handlers.
MCMGR_Init()
Register the application event before starting the secondary core.
MCMGR_RegisterEvent()
Trigger the secondary core application execution. Pass the startupData to the secondary core application. Either wait until the secondary core reads and confirms the startup data (kMCMGR_Start_Synchronous mode) or does not wait (kMCMGR_Start_Asynchronous mode).
MCMGR_StartCore()
release the secondary core from the reset
RemoteCoreUpEvent message
MCMGR_EarlyInit()
Initialize the multicore manager, early init. Call this function as close to the reset entry as possible, (into the startup sequence) to allow CoreUp event triggering.
MCMGR_Init()
Multicore manager init function to be called in the application main. It registers critical and generic event handlers.
ISR
ISR
FeedStartupDataEvent message
StartupDataEvent message
MCMGR_FeedStartup DataEventHandler
MCMGR_StartupData EventHandler
MCMGR_GetStartupData()
Trigger a mechanism to get the startup data from the primary core to the secondary core.
FeedStartupDataEvent message
StartupDataEvent message
FeedStartupDataEvent message
RemoteApplication Event handler
RemoteApplicationEvent message
MCMGR_TriggerEvent()
Trigger the application event to signal the primary core some application state and to pass the accompany data.
Register the RemoteExceptionEvent handler.
MCMGR_RegisterEvent()
RemoteException Event handler
RemoteExceptionEvent message
DefaultISR (exception)

Changelog Multicore Manager

All notable changes to this project will be documented in this file.

The format is based on Keep a Changelog, and this project adheres to Semantic Versioning.

v5.3.0

Added

- Added MCXE32B (dual Cortex-M7) porting layer.
- Remove obsolete unused platform layers
- Added initialization guard to public MCMGR API functions (RPSDK-1183): guarded functions return kStatus_MCMGR_NotReady before MCMGR_Init() completes.

Fixed

- MCUX-88924: Fix KW43 ICS init-handshake hang (RPSDK-1183 regression): s_mcmgrInitialized is now set before mcmgr_platform_init_internal() enables the MU RX IRQ, so re-entrant ISR dispatch during MCMGR_Init() is no longer silently dropped by MCMGR_CHECK_INIT().
- MCUX-89876: Fix Coverity DEADCODE and UNUSED_VALUE findings in mcmgr_internal_core_api_imxrt700.c: align mcmgr_mu_channel_handler event-range guard with positive-logic form (matching imxrt1170), and aggregate mcmgr_trigger_event_internal return values in mcmgr_platform_init_internal_early so earlier errors are not silently overwritten.

v5.2.0

Changed

- Merged internal platform initialization into a single $\mathrm{mcmgr_platform_init_internal()}$ function.
- Simplified initialization API: $\mathrm{MCMGR_Init()}$ now handles both early and standard initialization internally.

Deprecated

- $\mathrm{MCMGR_EarlyInit()}$ is now deprecated and a no-op. It remains for backward compatibility and may be removed in a future major version. Replace calls to $\mathrm{MCMGR_EarlyInit()}$ + $\mathrm{MCMGR_Init()}$ with a single $\mathrm{MCMGR_Init()}$ call.

Migration Guide

Before:

```
MCMGR_EarlyInit();
// ...
MCMGR_Init();
```

After:

```
MCMGR_Init();
```

v5.1.0

Added

- Added mcxl14x parts support
- Added mcxw70 parts support
- Add new config MCMGR_REMOTE_APP_EVENT_COUNT for configurable remote application event count. This will expand number of application events available for MCMGR users. kMCMGR_RemoteApplicationEvent, kMCMGR_RemoteApplicationEvent1 .. kMCMGR_RemoteApplicationEvent4.
- Align Kinetis parts porting to changed register map

Fixed

- Fix MISRA C-2012 Rule 20.8 violations.

v5.0.2

Added

- Added gcov options and configs to support mcmgr code coverage
- Added new test_weak_mu_isr testcase for devices with MU peripheral
- Added new test_heartbeat testcase showing heartbeat mechanism between primary and secondary cores using the MCMGR

v5.0.1

Added

- Added frdmimxrt1186 unit testing

Changed

- [KW43] Rename core#1 reset control register

Fixed

- Added CX flag into CMakeLists.txt to allow c++ build compatibility.
- Fix path to mcmgr headers directory in doxyfile

v5.0.0

Added

- Added MCMGR_BUSY_POLL_COUNT macro to prevent infinite polling loops in MCMGR operations.
- Implemented timeout mechanism for all polling loops in MCMGR code.
- Added support to handle more then two cores. Breaking API change by adding parameter coreNum specifying core number in functions bellow.
  - MCMGR_GetStartupData(uint32_t *startupData, mcmgr_core_t coreNum)
  - MCMGR_TriggerEvent(mcmgr_event_type_t type, uint16_t eventData, mcmgr_core_t coreNum)
  - MCMGR_TriggerEventForce(mcmgr_event_type_t type, uint16_t eventData, mcmgr_core_t coreNum)
  - typedef void (*mcmgr_event_callback_t)(uint16_t data, void *context, mcmgr_core_t coreNum);

When registering the event with function MCMGR_RegisterEvent() user now needs to provide callbackData pointer to array of elements per every core in system (see README.md for example).In case of systems with only two cores the coreNum in callback can be ignored as events can arrive only from one core. Please see Porting guide for more details: PortingGuideTo_v5.md

- Updated all porting files to support new MCMGR API.
- Added new platform specific include file mcmgr_platform.h. It will contain common platform specific macros that can be then used in mcmgr and application. e.g. platform core count MCMGR_CORECOUNT 4.
- Move all header files to new inc directory.
- Added new platform-specific include files inc/platform/<platform_name>/mcmgr_platform.h.

Added

- Add MCXL20 porting layer and unit testing

v4.1.7

Fixed

- mcmgr_stop_core_internal() function now returns kStatus_MCMGR_NotImplemented status code instead of kStatus_MCMGR_Success when device does not support stop of secondary core. Ports affected: kw32w1, kw45b41, kw45b42, mcxw716, mcxw727.

[v4.1.6]

Added

- Multicore Manager moved to standalone repository.
- Add porting layers for imxrt700, mcmxw727, kw47b42.
- New MCMGR_ProcessDeferredRxIsr() API added.

[v4.1.5]

Added

- Add notification into MCMGR_EarlyInit and mcmgr_early_init_internal functions to avoid using uninitialized data in their implementations.

[v4.1.4]

Fixed

- Avoid calling tx isr callbacks when respective Messaging Unit Transmit Interrupt Enable flag is not set in the CR/TCR register.
- Messaging Unit RX and status registers are cleared after the initialization.

[v4.1.3]

Added

- Add porting layers for imxrt1180.

Fixed

- mu_isr() updated to avoid calling tx isr callbacks when respective Transmit Interrupt Enable flag is not set in the CR/TCR register.
- mcmgr_mu_internal.c code adaptation to new supported SoCs.

[v4.1.2]

Fixed

- Update mcmgr_stop_core_internal() implementations to set core state to kMCMGR_ResetCoreState.

[v4.1.0]

Fixed

- Code adjustments to address MISRA C-2012 Rules

[v4.0.3]

Fixed

- Documentation updated to describe handshaking in a graphic form.
- Minor code adjustments based on static analysis tool findings

[v4.0.2]

Fixed

- Align porting layers to the updated MCUXpressoSDK feature files.

[v4.0.1]

Fixed

- Code formatting, removed unused code

[v4.0.0]

Added

- Add new MCMGR_TriggerEventForce() API.

[v3.0.0]

Removed

- Removed MCMGR_LoadApp(), MCMGR_MapAddress() and MCMGR_SignalReady()

Modified

- Modified MCMGR_GetStartupData()

Added

- Added MCMGR_EarlyInit(), MCMGR_RegisterEvent() and MCMGR_TriggerEvent()
- Added the ability for remote core monitoring and event handling

[v2.0.1]

Fixed

- Updated to be Misra compliant.

[v2.0.0]

Added

- Support for lpcxpresso54114 board.

[v1.1.0]

Fixed

- Ported to KSDK 2.0.0.

[v1.0.0]

Added

- Initial release.

eRPC

eRPC (Embedded RPC) is an open-source remote procedure call (RPC) system suitable for multi-core and multiprocessor embedded systems.

Quick links

- *Getting Started*
- *Changelog*
- Upstream project: https://github.com/EmbeddedRPC/erpc

eRPC Getting Started

Overview

This *Getting Started User Guide* shows software developers how to use Remote Procedure Calls (RPC) in embedded multicore microcontrollers (eRPC).

The eRPC documentation is located in the *<MCUXpressoSDK_install_dir>/ middleware/multicore/erpc/doc* folder.

Create an eRPC application

This section describes a generic way to create a client/server eRPC application:

1. **Design the eRPC application:** Decide which data types are sent between applications, and define functions that send/receive this data.

2. **Create the IDL file:** The IDL file contains information about data types and functions used in an eRPC application, and is written in the IDL language.
3. **Use the eRPC generator tool:** This tool takes an IDL file and generates the shim code for the client and the server-side applications.
4. **Create an eRPC application:**
    1. Create two projects, where one project is for the client side (primary core) and the other project is for the server side (secondary core).
    2. Add generated files for the client application to the client project, and add generated files for the server application to the server project.
    3. Add infrastructure files.
    4. Add user code for client and server applications.
    5. Set the client and server project options.
5. **Run the eRPC application:** Run both the server and the client applications. Make sure that the server has been run before the client request was sent.

A specific example follows in the next section.

Multicore server application

The "Matrix multiply" eRPC server project is located in the following folder:

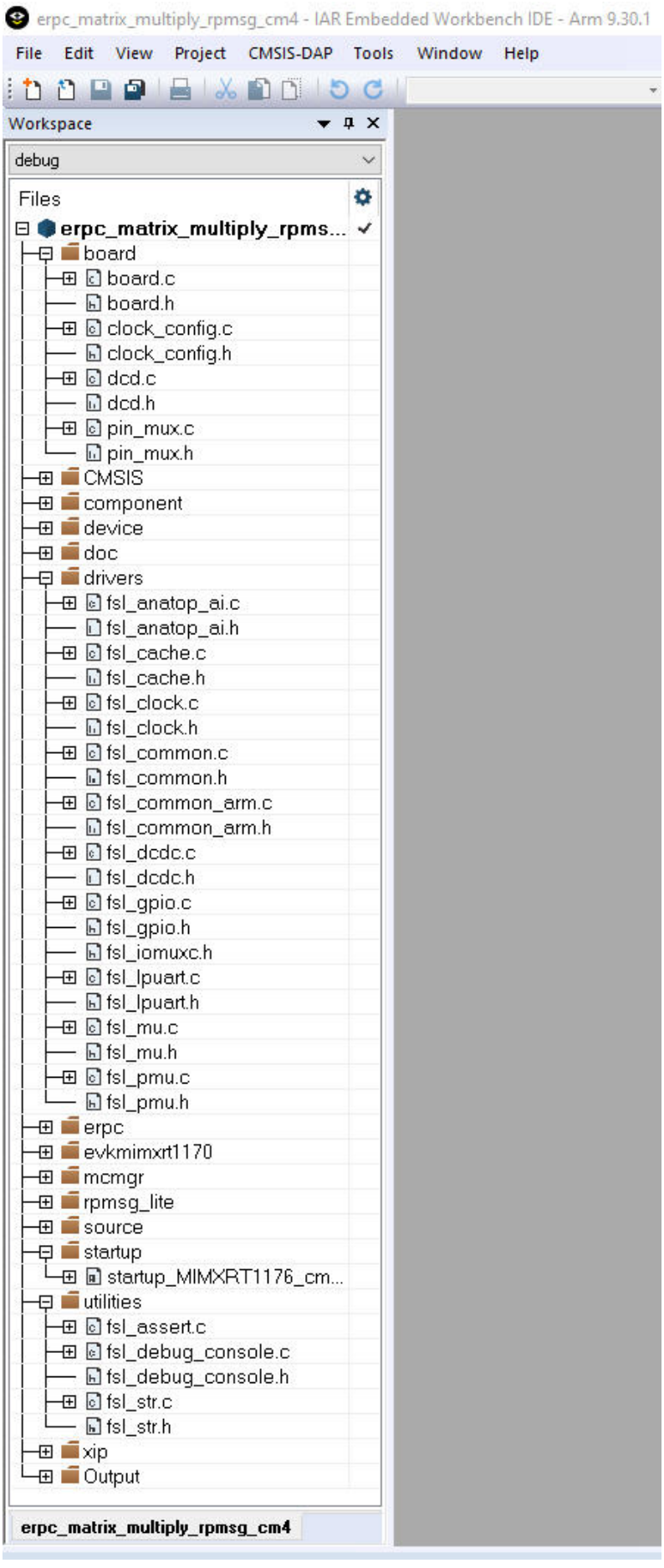
<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_matrix_multiply_rpmsg/cm4/iar

The project files for the eRPC server have the `_cm4` suffix.

Server project basic source files

The startup files, board-related settings, peripheral drivers, and utilities belong to the basic project source files and form the skeleton of all MCUXpresso SDK applications. These source files are located in:

- *<MCUXpressoSDK_install_dir>/devices/<device>*
- *<MCUXpressoSDK_install_dir>/boards/<board_name>/multicore_examples/<example_name>/*


erpc_matrix_multiply_rpmsg_cm4 - IAR Embedded Workbench IDE - Arm 9.30.1
File Edit View Project CMSIS-DAP Tools Window Help
Workspace
debug
Files
erpc_matrix_multiply_rpms...
board
board.c
board.h
clock_config.c
clock_config.h
dcd.c
dcd.h
pin_mux.c
pin_mux.h
CMSIS
component
device
doc
drivers
fsl_anatop_ai.c
fsl_anatop_ai.h
fsl_cache.c
fsl_cache.h
fsl_clock.c
fsl_clock.h
fsl_common.c
fsl_common.h
fsl_common_arm.c
fsl_common_arm.h
fsl_dcdc.c
fsl_dcdc.h
fsl_gpio.c
fsl_gpio.h
fsl_iomuxc.h
fsl_lpuart.c
fsl_lpuart.h
fsl_mu.c
fsl_mu.h
fsl_pmu.c
fsl_pmu.h
erpc
evkmimxrt1170
mcmgr
rpmsg_lite
source
startup
startup_MIMXRT1176_cm...
utilities
fsl_assert.c
fsl_debug_console.c
fsl_debug_console.h
fsl_str.c
fsl_str.h
xip
Output
erpc_matrix_multiply_rpmsg_cm4

|

Parent topic: Multicore server application

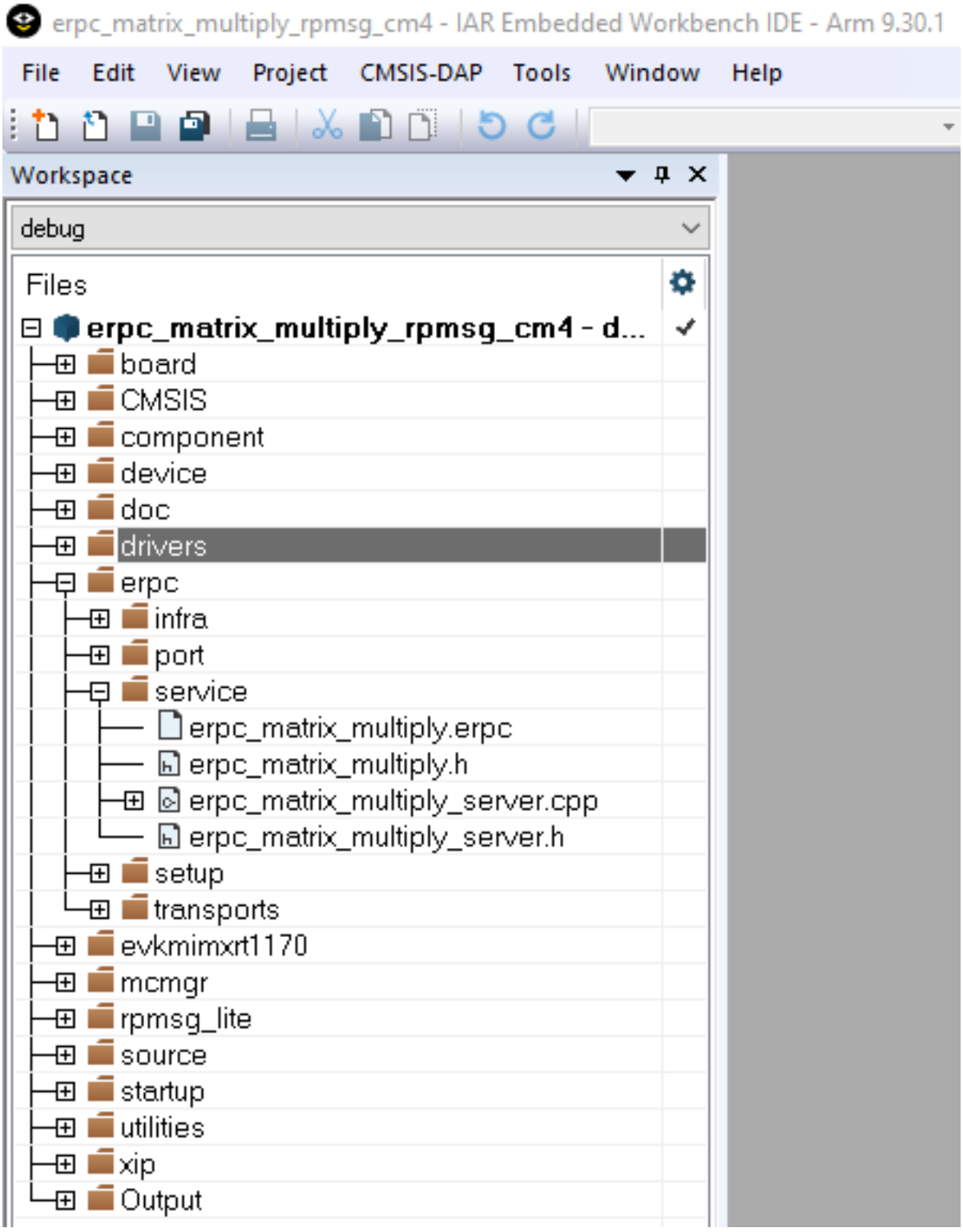
Server related generated files

The server-related generated files are:

- erpc_matric_multiply.h
- erpc_matrix_multiply_server.h
- erpc_matrix_multiply_server.cpp

The server-related generated files contain the shim code for functions and data types declared in the IDL file. These files also contain functions for the identification of client requested functions, data deserialization, calling requested function's implementations, and data serialization and return, if requested by the client. These shim code files can be found in the following folder:

<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_common/erpc_matrix_multiply/s



Parent topic: Multicore server application

Server infrastructure files

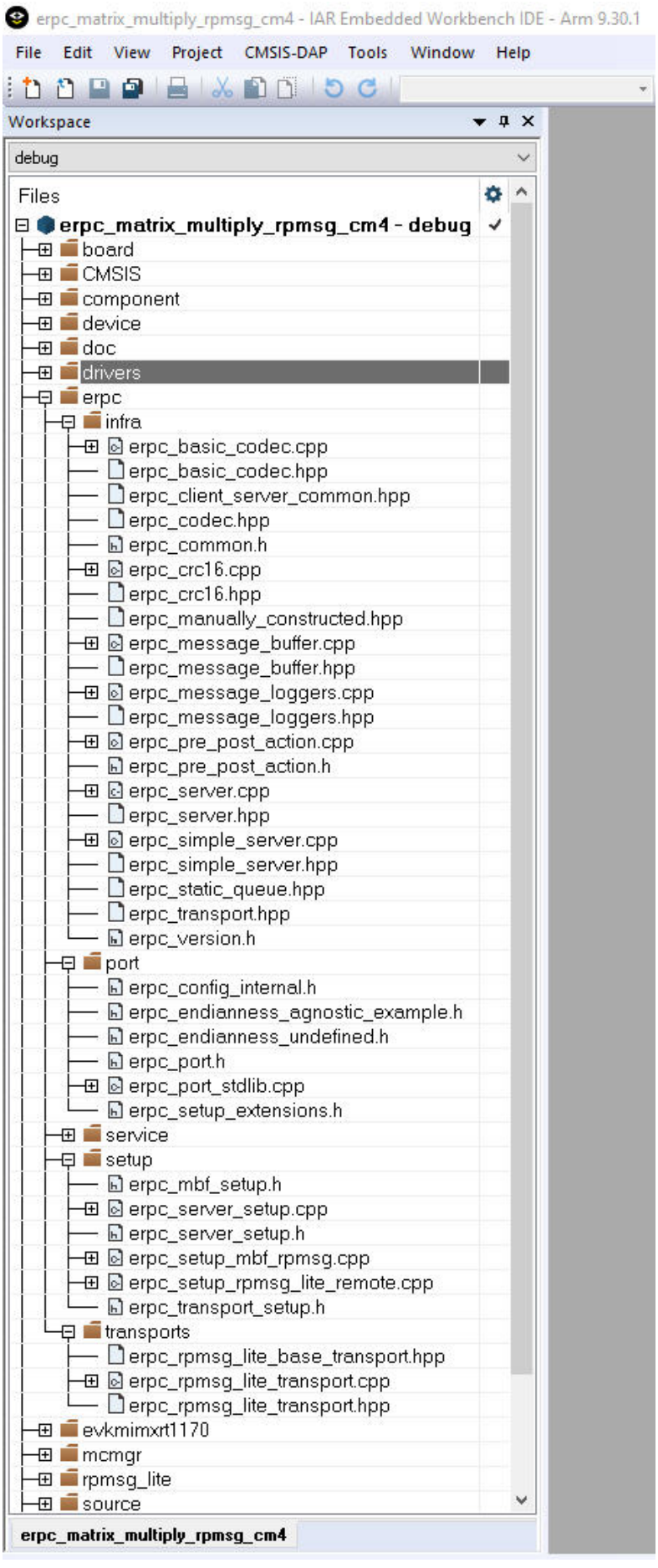
The eRPC infrastructure files are located in the following folder:

<MCUXpressoSDK_install_dir>/middleware/multicore/erpc/erpc_c

The **erpc_c** folder contains files for creating eRPC client and server applications in the *C/C++*

language. These files are distributed into subfolders.

- The **infra** subfolder contains C++ infrastructure code used to build server and client applications.
  - Four files, erpc_server.hpp, erpc_server.cpp, erpc_simple_server.hpp, and erpc_simple_server.cpp, are used for running the eRPC server on the server-side applications. The simple server is currently the only implementation of the server, and its role is to catch client requests, identify and call requested functions, and send data back when requested.
  - Three files (erpc_codec.hpp, erpc_basic_codec.hpp, and erpc_basic_codec.cpp) are used for codecs. Currently, the basic codec is the initial and only implementation of the codecs.
  - The erpc_common.hpp file is used for common eRPC definitions, typedefs, and enums.
  - The erpc_manually_constructed.hpp file is used for allocating static storage for the used objects.
  - Message buffer files are used for storing serialized data: erpc_message_buffer.h and erpc_message_buffer.cpp.
  - The erpc_transport.h file defines the abstract interface for transport layer.
- The **port** subfolder contains the eRPC porting layer to adapt to different environments.
  - erpc_port.h file contains definition of erpc_malloc() and erpc_free() functions.
  - erpc_port_stdlib.cpp file ensures adaptation to stdlib.
  - erpc_config_internal.h internal erpc configuration file.
- The **setup** subfolder contains a set of plain C APIs that wrap the C++ infrastructure, providing client and server init and deinit routines that greatly simplify eRPC usage in C-based projects. No knowledge of C++ is required to use these APIs.
  - The erpc_server_setup.h and erpc_server_setup.cpp files needs to be added into the "Matrix multiply" example project to demonstrate the use of C-wrapped functions in this example.
  - The erpc_transport_setup.h and erpc_setup_rpmsg_lite_remote.cpp files needs to be added into the project in order to allow the C-wrapped function for transport layer setup.
  - The erpc_mbf_setup.h and erpc_setup_mbf_rpmsg.cpp files needs to be added into the project in order to allow message buffer factory usage.
- The **transports** subfolder contains transport classes for the different methods of communication supported by eRPC. Some transports are applicable only to host PCs, while others are applicable only to embedded or multicore systems. Most transports have corresponding client and server setup functions in the setup folder.
  - RPMsg-Lite is used as the transport layer for the communication between cores, erpc_rpmsg_lite_base_transport.hpp, erpc_rpmsg_lite_transport.hpp, and erpc_rpmsg_lite_transport.cpp files need to be added into the server project.


erpc_matrix_multiply_rpmsg_cm4 - IAR Embedded Workbench IDE - Arm 9.30.1
File Edit View Project CMSIS-DAP Tools Window Help
Workspace
debug
Files
erpc_matrix_multiply_rpmsg_cm4 - debug
board
CMSIS
component
device
doc
drivers
erpc
infra
erpc_basic_codec.cpp
erpc_basic_codec.hpp
erpc_client_server_common.hpp
erpc_codec.hpp
erpc_common.h
erpc_crc16.cpp
erpc_crc16.hpp
erpc_manually_constructed.hpp
erpc_message_buffer.cpp
erpc_message_buffer.hpp
erpc_message_loggers.cpp
erpc_message_loggers.hpp
erpc_pre_post_action.cpp
erpc_pre_post_action.h
erpc_server.cpp
erpc_server.hpp
erpc_simple_server.cpp
erpc_simple_server.hpp
erpc_static_queue.hpp
erpc_transport.hpp
erpc_version.h
port
erpc_config_internal.h
erpc_endianness_agnostic_example.h
erpc_endianness_undefined.h
erpc_port.h
erpc_port_stdlib.cpp
erpc_setup_extensions.h
service
setup
erpc_mbf_setup.h
erpc_server_setup.cpp
erpc_server_setup.h
erpc_setup_mbf_rpmsg.cpp
erpc_setup_rpmsg_lite_remote.cpp
erpc_transport_setup.h
transports
erpc_rpmsg_lite_base_transport.hpp
erpc_rpmsg_lite_transport.cpp
erpc_rpmsg_lite_transport.hpp
evkmimxrt1170
mcmgr
rpmsg_lite
source
erpc_matrix_multiply_rpmsg_cm4

|

Parent topic: Multicore server application

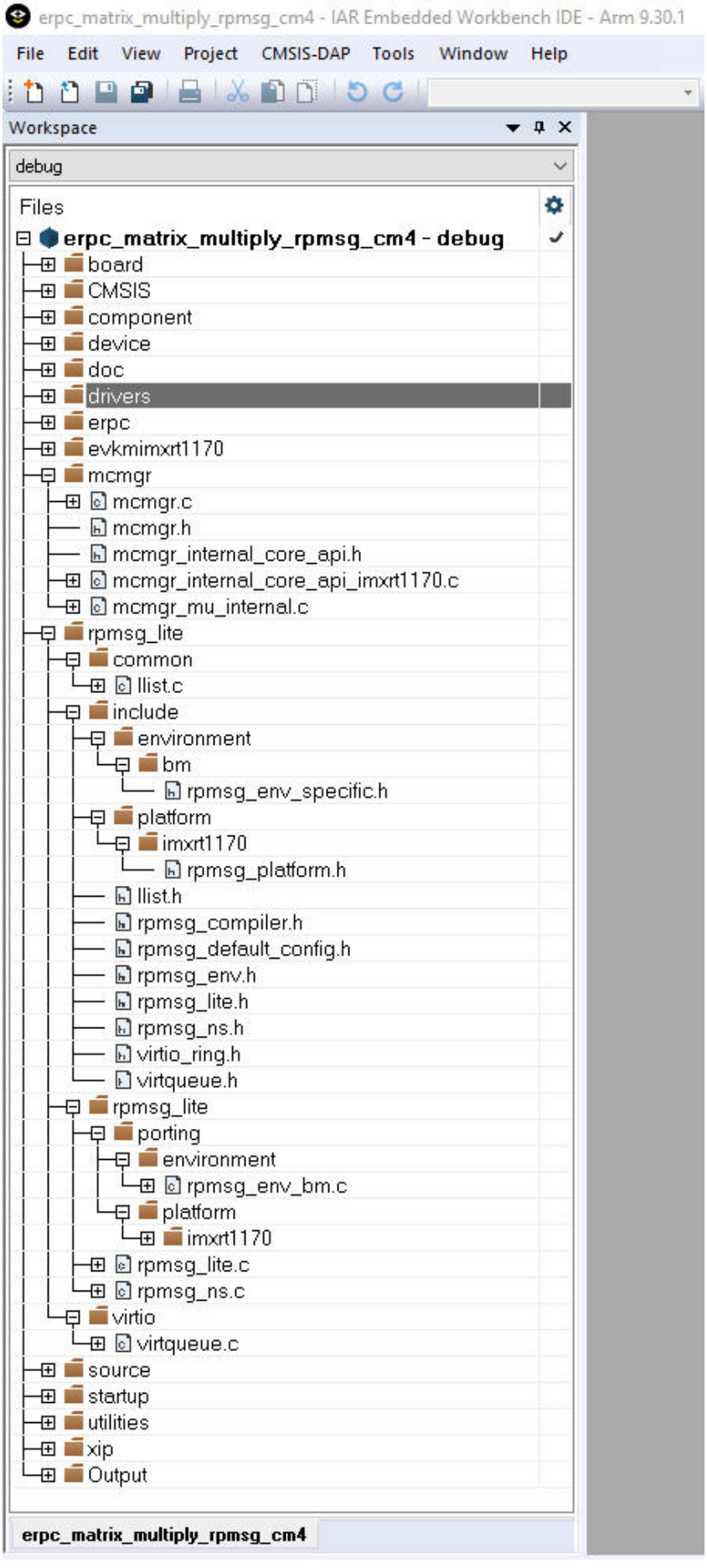
Server multicore infrastructure files

Because of the RPMsg-Lite (transport layer), it is also necessary to include RPMsg-Lite related files, which are in the following folder:

<MCUXpressoSDK_install_dir>/middleware/multicore/rpmsg_lite/

The multicore example applications also use the Multicore Manager software library to control the secondary core startup and shutdown. These source files are located in the following folder:

<MCUXpressoSDK_install_dir>/middleware/multicore/mcmgr/


erpc_matrix_multiply_rpmsg_cm4 - IAR Embedded Workbench IDE - Arm 9.30.1
File Edit View Project CMSIS-DAP Tools Window Help
Workspace
debug
Files
erpc_matrix_multiply_rpmsg_cm4 - debug
board
CMSIS
component
device
doc
drivers
erpc
evkmimxrt1170
mcmgr
mcmgr.c
mcmgr.h
mcmgr_internal_core_api.h
mcmgr_internal_core_api_imxrt1170.c
mcmgr_mu_internal.c
rpmsg_lite
common
llist.c
include
environment
bm
rpmsg_env_specific.h
platform
imxrt1170
rpmsg_platform.h
llist.h
rpmsg_compiler.h
rpmsg_default_config.h
rpmsg_env.h
rpmsg_lite.h
rpmsg_ns.h
virtio_ring.h
virtqueue.h
rpmsg_lite
porting
environment
rpmsg_env_bm.c
platform
imxrt1170
rpmsg_lite.c
rpmsg_ns.c
virtio
virtqueue.c
source
startup
utilities
xip
Output
erpc_matrix_multiply_rpmsg_cm4

|

Parent topic: Multicore server application

Server user code

The server's user code is stored in the main_core1.c file, located in the following folder:

<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_matrix_multiply_rpmsg/cm4

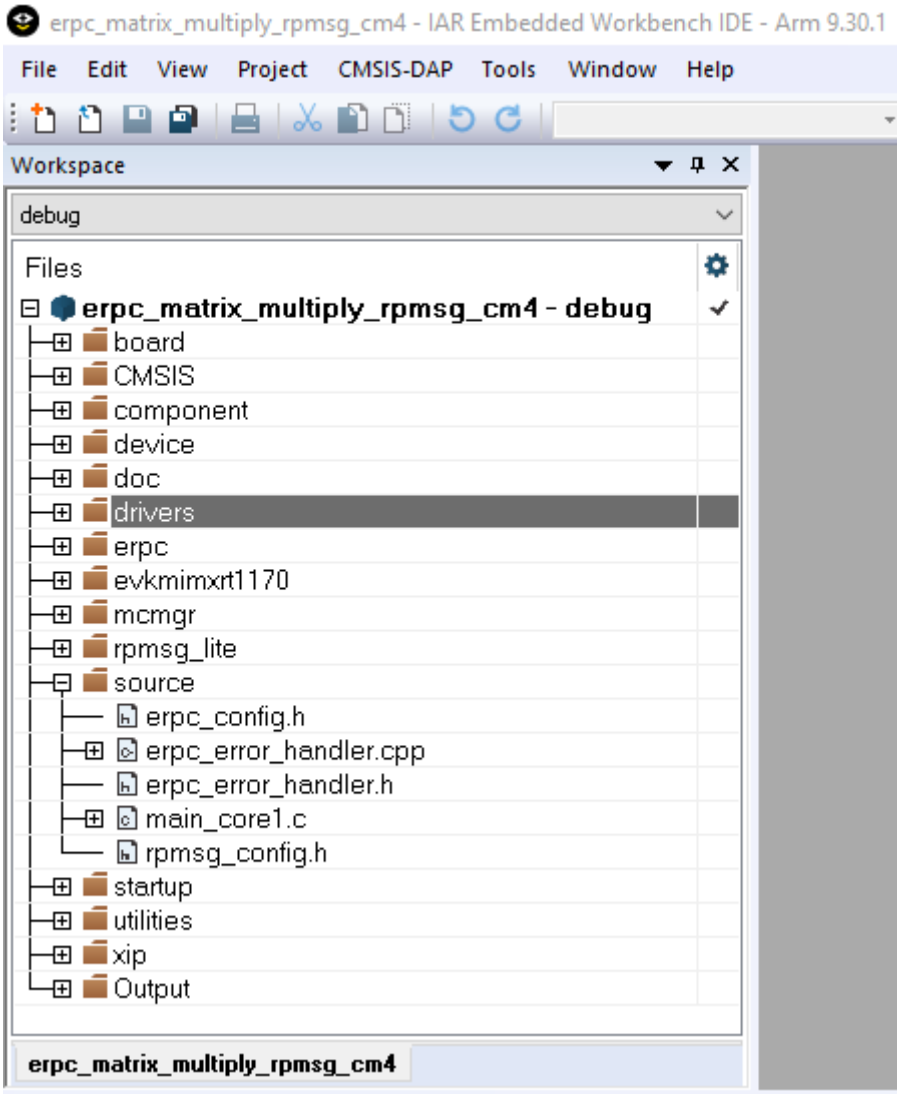
The main_core1.c file contains two functions:

- The **main()** function contains the code for the target board and eRPC server initialization. After the initialization, the matrix multiply service is added and the eRPC server waits for client's requests in the while loop.
- The **erpcMatrixMultiply()** function is the user implementation of the eRPC function defined in the IDL file.
- There is the possibility to write the application-specific eRPC error handler. The eRPC error handler of the matrix multiply application is implemented in the erpc_error_handler.h and erpc_error_handler.cpp files.

The eRPC-relevant code is captured in the following code snippet:

```
/* erpcMatrixMultiply function user implementation */
void erpcMatrixMultiply(const Matrix *matrix1, const Matrix *matrix2, Matrix *result_matrix)
{
  ...
}
int main()
{
  ...
  /* RPMsg-Lite transport layer initialization */
  erpc_transport_t transport;
  transport = erpc_transport_rpmsg_lite_remote_init(src, dst, (void*)startupData,
  ERPC_TRANSPORT_RPMSG_LITE_LINK_ID, SignalReady, NULL);
  ...
  /* MessageBufferFactory initialization */
  erpc_mbf_t message_buffer_factory;
  message_buffer_factory = erpc_mbf_rpmsg_init(transport);
  ...
  /* eRPC server side initialization */
  erpc_server_t server;
  server = erpc_server_init(transport, message_buffer_factory);
  ...
  /* Adding the service to the server */
  erpc_service_t service = create_MatrixMultiplyService_service();
  erpc_add_service_to_server(server, service);
  ...
  while (1)
  {
    /* Process eRPC requests */
    erpc_status_t status = erpc_server_poll(server);
    /* handle error status */
    if (status != kErpcStatus_Success)
    {
      /* print error description */
      erpc_error_handler(status, 0);
      ...
    }
    ...
  }
}
```

Except for the application main file, there are configuration files for the RPMsg-Lite (rpmsg_config.h) and eRPC (erpc_config.h), located in the *<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/ erpc_matrix_multiply_rpmsg* folder.



Parent topic: Multicore server application

Parent topic:*Create an eRPC application*

Multicore client application

The "Matrix multiply" eRPC client project is located in the following folder:

<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_matrix_multiply_rpmsg/cm7/iar

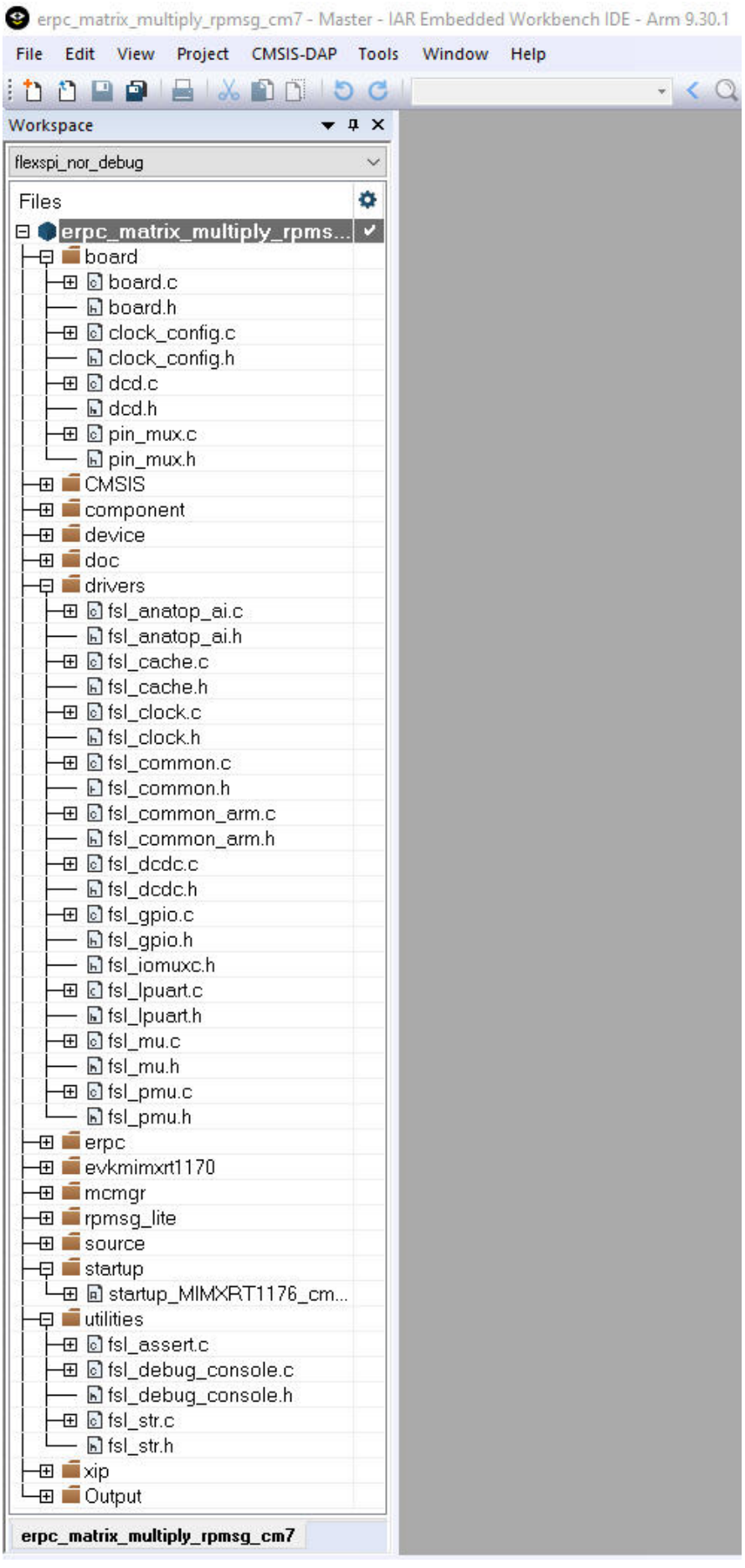
Project files for the eRPC client have the _cm7 suffix.

Client project basic source files

The startup files, board-related settings, peripheral drivers, and utilities belong to the basic project source files and form the skeleton of all MCUXpresso SDK applications. These source files are located in the following folders:

- *<MCUXpressoSDK_install_dir>/devices/<device>*

- *<MCUXpressoSDK_install_dir>/boards/<board_name>/multicore_examples/<example_name>/*


erpc_matrix_multiply_rpmsg_cm7 - Master - IAR Embedded Workbench IDE - Arm 9.30.1
File
Edit
View
Project
CMSIS-DAP
Tools
Window
Help
Workspace
flexspi_nor_debug
Files
erpc_matrix_multiply_rpms...
board
board.c
board.h
clock_config.c
clock_config.h
dcd.c
dcd.h
pin_mux.c
pin_mux.h
CMSIS
component
device
doc
drivers
fsl_anatop_ai.c
fsl_anatop_ai.h
fsl_cache.c
fsl_cache.h
fsl_clock.c
fsl_clock.h
fsl_common.c
fsl_common.h
fsl_common_arm.c
fsl_common_arm.h
fsl_dcdc.c
fsl_dcdc.h
fsl_gpio.c
fsl_gpio.h
fsl_iomuxc.h
fsl_lpuart.c
fsl_lpuart.h
fsl_mu.c
fsl_mu.h
fsl_pmu.c
fsl_pmu.h
erpc
evkmimxrt1170
mcmgr
rpmsg_lite
source
startup
startup_MIMXRT1176_cm...
utilities
fsl_assert.c
fsl_debug_console.c
fsl_debug_console.h
fsl_str.c
fsl_str.h
xip
Output
erpc_matrix_multiply_rpmsg_cm7

|

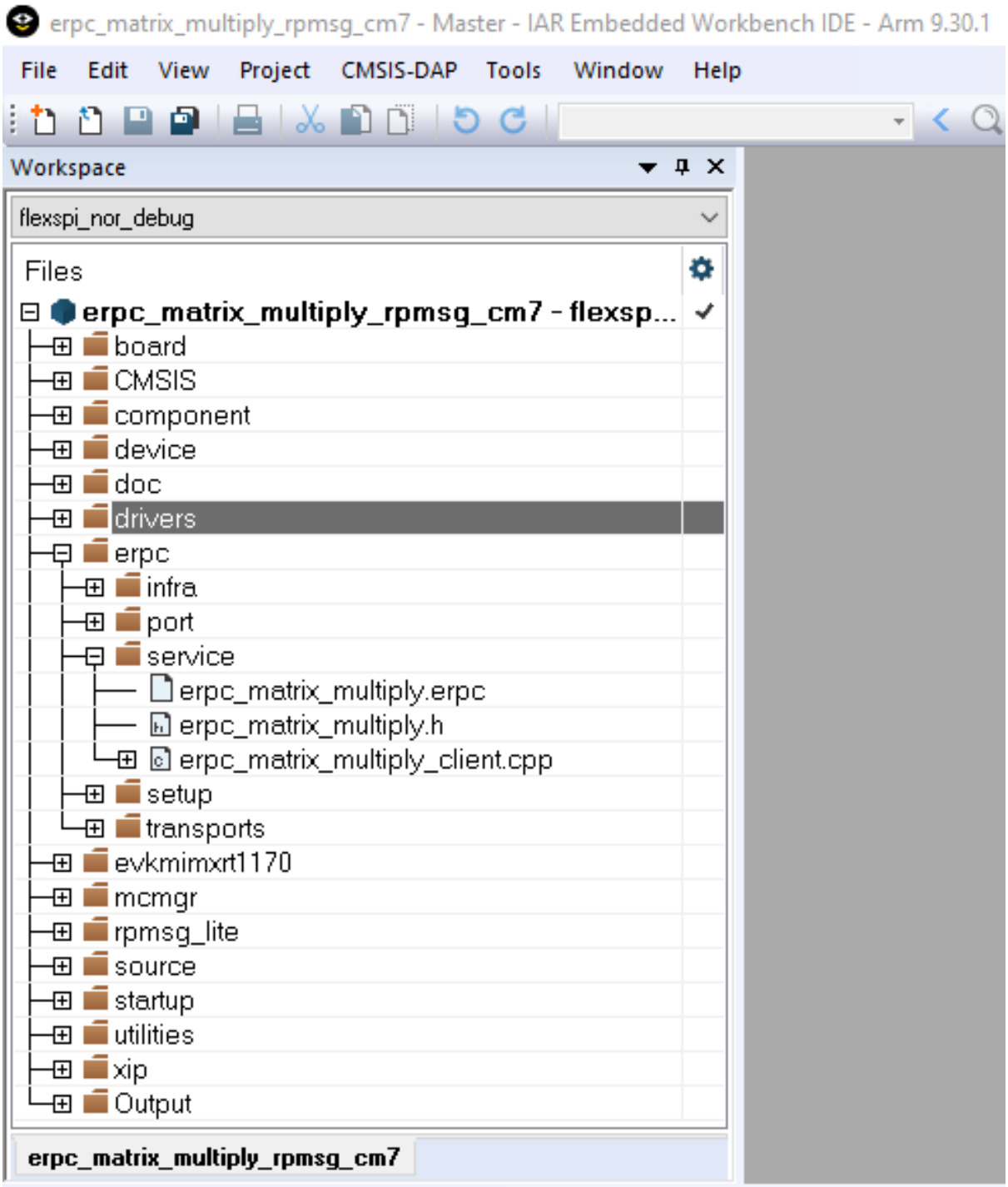
Parent topic: Multicore client application

Client-related generated files

The client-related generated files are:

- erpc_matric_multiply.h
- erpc_matrix_multiply_client.cpp

These files contain the shim code for the functions and data types declared in the IDL file. These functions also call methods for codec initialization, data serialization, performing eRPC requests, and de-serializing outputs into expected data structures (if return values are expected). These shim code files can be found in the *<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_common/erpc_matrix_multiply/service/* folder.



Parent topic: Multicore client application

Client infrastructure files

The eRPC infrastructure files are located in the following folder:

<MCUXpressoSDK_install_dir>/middleware/multicore/erpc/erpc_c

The **erpc_c** folder contains files for creating eRPC client and server applications in the C/C++ language. These files are distributed into subfolders.

- The **infra** subfolder contains C++ infrastructure code used to build server and client applications.

- Two files, erpc_client_manager.h and erpc_client_manager.cpp, are used for managing the client-side application. The main purpose of the client files is to create, perform, and release eRPC requests.
- Three files (erpc_codec.hpp, erpc_basic_codec.hpp, and erpc_basic_codec.cpp) are used for codecs. Currently, the basic codec is the initial and only implementation of the codecs.
- erpc_common.h file is used for common eRPC definitions, typedefs, and enums.
- erpc_manually_constructed.hpp file is used for allocating static storage for the used objects.
- Message buffer files are used for storing serialized data: erpc_message_buffer.hpp and erpc_message_buffer.cpp.
- erpc_transport.hpp file defines the abstract interface for transport layer.

The **port** subfolder contains the eRPC porting layer to adapt to different environments.

- erpc_port.h file contains definition of erpc_malloc() and erpc_free() functions.
- erpc_port_stdlib.cpp file ensures adaptation to stdlib.
- erpc_config_internal.h internal eRPC configuration file.

The **setup** subfolder contains a set of plain C APIs that wrap the C++ infrastructure, providing client and server init and deinit routines that greatly simplify eRPC usage in C-based projects. No knowledge of C++ is required to use these APIs.

- erpc_client_setup.h and erpc_client_setup.cpp files needs to be added into the "Matrix multiply" example project to demonstrate the use of C-wrapped functions in this example.
- erpc_transport_setup.h and erpc_setup_rpmsg_lite_master.cpp files needs to be added into the project in order to allow C-wrapped function for transport layer setup.
- erpc_mbf_setup.h and erpc_setup_mbf_rpmsg.cpp files needs to be added into the project in order to allow message buffer factory usage.

The **transports** subfolder contains transport classes for the different methods of communication supported by eRPC. Some transports are applicable only to host PCs, while others are applicable only to embedded or multicore systems. Most transports have corresponding client and server setup functions, in the setup folder.

- RPMsg-Lite is used as the transport layer for the communication between cores, erpc_rpmsg_lite_base_transport.hpp, erpc_rpmsg_lite_transport.hpp, and erpc_rpmsg_lite_transport.cpp files needs to be added into the client project.

erpc_matrix_multiply_rpmsg_cm7 - Master - IAR Embedded Workbench IDE - Arm 9.30.1
File Edit View Project CMSIS-DAP Tools Window Help
Workspace
flexspi_nor_debug
Files
erpc_matrix_multiply_rpmsg_cm7 - flexspi_n...
board
CMSIS
component
device
doc
drivers
erpc
infra
erpc_basic_codec.cpp
erpc_basic_codec.hpp
erpc_client_manager.cpp
erpc_client_manager.h
erpc_client_server_common.hpp
erpc_codec.hpp
erpc_common.h
erpc_crc16.cpp
erpc_crc16.hpp
erpc_manually_constructed.hpp
erpc_message_buffer.cpp
erpc_message_buffer.hpp
erpc_message_loggers.cpp
erpc_message_loggers.hpp
erpc_pre_post_action.cpp
erpc_pre_post_action.h
erpc_static_queue.hpp
erpc_transport.hpp
erpc_version.h
port
erpc_config_internal.h
erpc_endianness_agnostic_example.h
erpc_endianness_undefined.h
erpc_port.h
erpc_port_stdlib.cpp
erpc_setup_extensions.h
service
setup
erpc_client_setup.cpp
erpc_client_setup.h
erpc_mbf_setup.h
erpc_setup_mbf_rpmsg.cpp
erpc_setup_rpmsg_lite_master.cpp
erpc_transport_setup.h
transports
erpc_rpmsg_lite_base_transport.hpp
erpc_rpmsg_lite_transport.cpp
erpc_rpmsg_lite_transport.hpp
evkmimxrt1170
mcmgr
rpmsg_lite
source
startup
utilities
erpc_matrix_multiply_rpmsg_cm7

|

Parent topic: Multicore client application

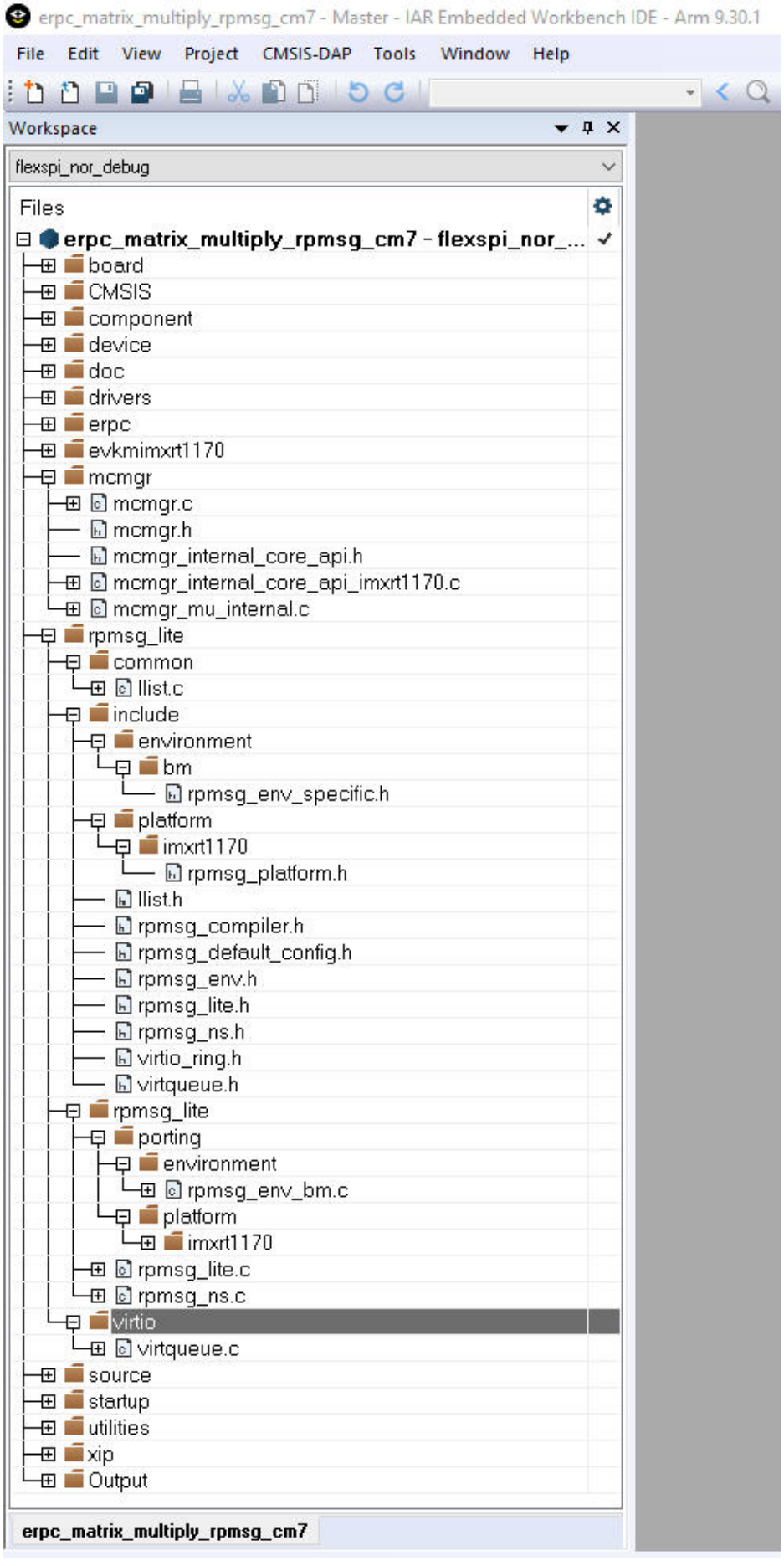
Client multicore infrastructure files

Because of the RPMsg-Lite (transport layer), it is also necessary to include RPMsg-Lite related files, which are in the following folder:

<MCUXpressoSDK_install_dir>/middleware/multicore/rpmsg_lite/

The multicore example applications also use the Multicore Manager software library to control the secondary core startup and shutdown. These source files are located in the following folder:

<MCUXpressoSDK_install_dir>/middleware/multicore/mcmgr/


erpc_matrix_multiply_rpmsg_cm7 - Master - IAR Embedded Workbench IDE - Arm 9.30.1
File
Edit
View
Project
CMSIS-DAP
Tools
Window
Help
Workspace
flexspi_nor_debug
Files
erpc_matrix_multiply_rpmsg_cm7 - flexspi_nor_...
board
CMSIS
component
device
doc
drivers
erpc
evkmimxrt1170
mcmgr
mcmgr.c
mcmgr.h
mcmgr_internal_core_api.h
mcmgr_internal_core_api_imxrt1170.c
mcmgr_mu_internal.c
rpmsg_lite
common
llist.c
include
environment
bm
rpmsg_env_specific.h
platform
imxrt1170
rpmsg_platform.h
llist.h
rpmsg_compiler.h
rpmsg_default_config.h
rpmsg_env.h
rpmsg_lite.h
rpmsg_ns.h
virtio_ring.h
virtqueue.h
rpmsg_lite
porting
environment
rpmsg_env_bm.c
platform
imxrt1170
rpmsg_lite.c
rpmsg_ns.c
virtio
virtqueue.c
source
startup
utilities
xip
Output
erpc_matrix_multiply_rpmsg_cm7

|

Parent topic: Multicore client application

Client user code

The client's user code is stored in the main_core0.c file, located in the following folder:

<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_example/erpc_matrix_multiply_rpmsg/cm7

The main_core0.c file contains the code for target board and eRPC initialization.

- After initialization, the secondary core is released from reset.
- When the secondary core is ready, the primary core initializes two matrix variables.
- The erpcMatrixMultiply eRPC function is called to issue the eRPC request and get the result.

It is possible to write the application-specific eRPC error handler. The eRPC error handler of the matrix multiply application is implemented in erpc_error_handler.h and erpc_error_handler.cpp files.

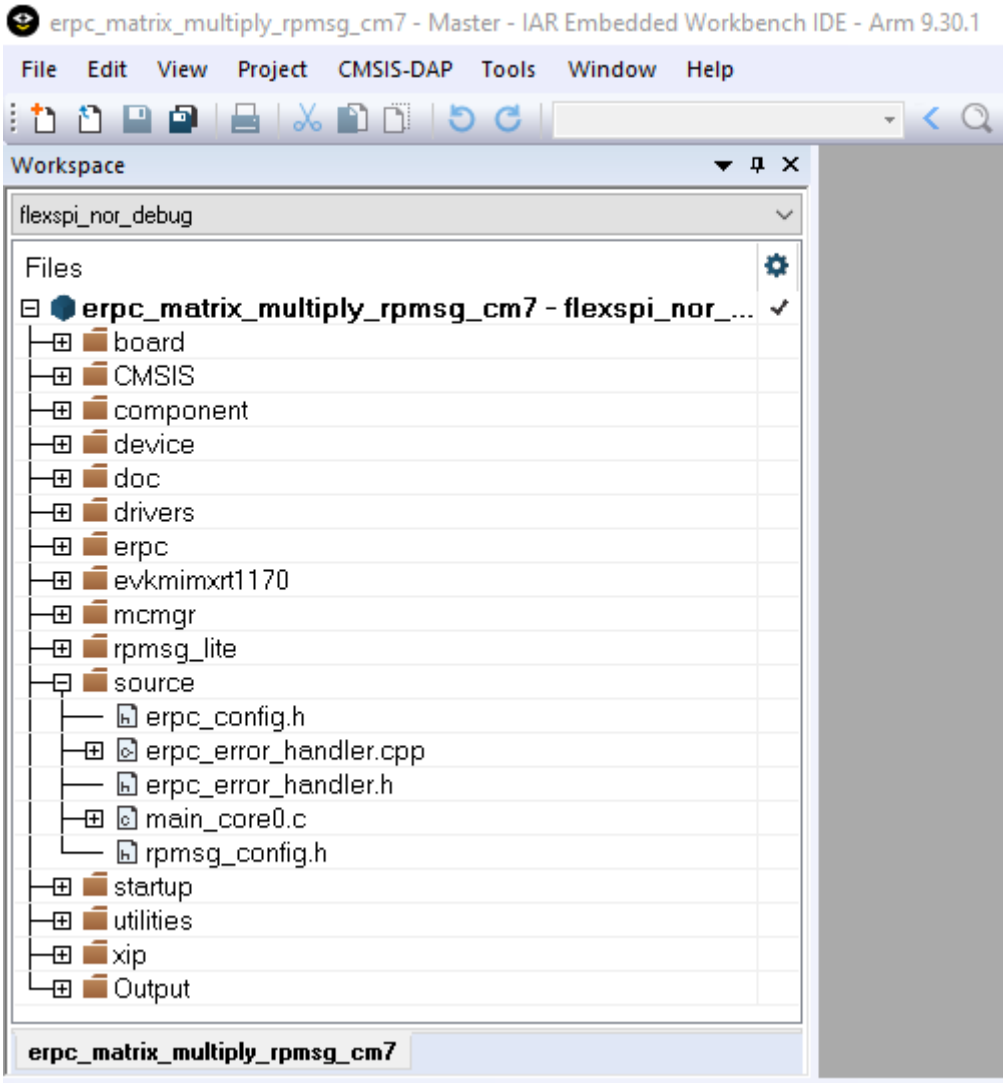
The matrix multiplication can be issued repeatedly, when pressing a software board button.

The eRPC-relevant code is captured in the following code snippet:

```
...
extern bool g_erpc_error_occurred;
...
/* Declare matrix arrays */
Matrix matrix1 = {0}, matrix2 = {0}, result_matrix = {0};
...
/* RPMsg-Lite transport layer initialization */
erpc_transport_t transport;
transport = erpc_transport_rpmsg_lite_master_init(src, dst,
ERPC_TRANSPORT_RPMSG_LITE_LINK_ID);
...
/* MessageBufferFactory initialization */
erpc_mbf_t message_buffer_factory;
message_buffer_factory = erpc_mbf_rpmsg_init(transport);
...
/* eRPC client side initialization */
erpc_client_t client;
client = erpc_client_init(transport, message_buffer_factory);
...
/* Set default error handler */
erpc_client_set_error_handler(client, erpc_error_handler);
...
while (1)
{
  /* Invoke the erpcMatrixMultiply function */
  erpcMatrixMultiply(matrix1, matrix2, result_matrix);
  ...
  /* Check if some error occured in eRPC */
  if (g_erpc_error_occurred)
  {
    /* Exit program loop */
    break;
  }
  ...
}
```

Except for the application main file, there are configuration files for the RPMsg-Lite (rpmsg_config.h) and eRPC (erpc_config.h), located in the following folder:

<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_matrix_multiply_rpmsg



|

|

Parent topic: Multicore client application

Parent topic:*Create an eRPC application*

Multiprocessor server application

The "Matrix multiply" eRPC server project for multiprocessor applications is located in the *<MCUXpressoSDK_install_dir»/boards/<board_name>/multiprocessor_examples/erpc_server_matrix_multiply_<transport_layer>* folder.

Most of the multiprocessor application setup is the same as for the multicore application. The multiprocessor server application requires server-related generated files (server shim code), server infrastructure files, and the server user code. There is no need for server multicore infrastructure files (MCMGR and RPMsg-Lite). The RPMsg-Lite transport layer is replaced either by SPI or UART transports. The following table shows the required transport-related files per each transport type.

|SPI|<eRPC base directory>/erpc_c/setup/erpc_setup_(d)spi_slave.cpp

<eRPC base directory>/erpc_c/transports/erpc_(d)spi_slave_transport.hpp

<eRPC base directory>/erpc_c/transports/erpc_(d)spi_slave_transport.cpp

| |UART|<eRPC base directory>/erpc_c/setup/erpc_setup_uart_cmsis.cpp

<eRPC base directory>/erpc_c/transports/erpc_uart_cmsis_transport.hpp

<eRPC base directory>/erpc_c/transports/erpc_uart_cmsis_transport.cpp

|

Server user code

The server's user code is stored in the main_server.c file, located in the *<MCUXpressoSDK_install_dir>/boards/<board_name>/multiprocessor_examples/erpc_server_matrix_multiply_<transport_lay* folder.

The eRPC-relevant code with UART as a transport is captured in the following code snippet:

```
/* erpcMatrixMultiply function user implementation */
void erpcMatrixMultiply(Matrix matrix1, Matrix matrix2, Matrix result_matrix)
{
  ...
}
int main()
{
  ...
  /* UART transport layer initialization, ERPC_DEMO_UART is the structure of CMSIS UART driver␣
↪operations */
  erpc_transport_t transport;
  transport = erpc_transport_cmsis_uart_init((void *)&ERPC_DEMO_UART);
  ...
  /* MessageBufferFactory initialization */
  erpc_mbf_t message_buffer_factory;
  message_buffer_factory = erpc_mbf_dynamic_init();
  ...
  /* eRPC server side initialization */
  erpc_server_t server;
  server = erpc_server_init(transport, message_buffer_factory);
  ...
  /* Adding the service to the server */
  erpc_service_t service = create_MatrixMultiplyService_service();
  erpc_add_service_to_server(server, service);
  ...
  while (1)
  {
    /* Process eRPC requests */
    erpc_status_t status = erpc_server_poll(server)
    /* handle error status */
    if (status != kErpcStatus_Success)
    {
      /* print error description */
      erpc_error_handler(status, 0);
      ...
    }
    ...
  }
}
```

Parent topic: Multiprocessor server application

Multiprocessor client application

The "Matrix multiply" eRPC client project for multiprocessor applications is located in the *<MCUXpressoSDK_install_dir>/boards/<board_name>/multiprocessor_examples/erpc_client_matrix_multiply_<tra* folder.

Most of the multiprocessor application setup is the same as for the multicore application. The multiprocessor server application requires client-related generated files (server shim code),

client infrastructure files, and the client user code. There is no need for client multicore infrastructure files (MCMGR and RPMsg-Lite). The RPMsg-Lite transport layer is replaced either by SPI or UART transports. The following table shows the required transport-related files per each transport type.

|SPI|<eRPC base directory>/erpc_c/setup/erpc_setup_(d)spi_master.cpp

<eRPC base directory>/erpc_c/transports/ erpc_(d)spi_master_transport.hpp

<eRPC base directory>/erpc_c/transports/ erpc_(d)spi_master_transport.cpp

| |UART|<eRPC base directory>/erpc_c/setup/erpc_setup_uart_cmsis.cpp

<eRPC base directory>/erpc_c/transports/erpc_uart_cmsis_transport.hpp

<eRPC base directory>/erpc_c/transports/erpc_uart_cmsis_transport.cpp

|

Client user code

The client's user code is stored in the main_client.c file, located in the *<MCUXpressoSDK_install_dir>/boards/<board_name>/multiprocessor_examples/ erpc_client_matrix_multiply_<transport_layer>/* folder.

The eRPC-relevant code with UART as a transport is captured in the following code snippet:

```
...
extern bool g_erpc_error_occurred;
...
/* Declare matrix arrays */
Matrix matrix1 = {0}, matrix2 = {0}, result_matrix = {0};
...
/* UART transport layer initialization, ERPC_DEMO_UART is the structure of CMSIS UART driver␣
↪operations */
erpc_transport_t transport;
transport = erpc_transport_cmsis_uart_init((void *)&ERPC_DEMO_UART);
...
/* MessageBufferFactory initialization */
erpc_mbf_t message_buffer_factory;
message_buffer_factory = erpc_mbf_dynamic_init();
...
/* eRPC client side initialization */
erpc_client_t client;
client = erpc_client_init(transport,message_buffer_factory);
...
/* Set default error handler */
erpc_client_set_error_handler(client, erpc_error_handler);
...
while (1)
{
  /* Invoke the erpcMatrixMultiply function */
  erpcMatrixMultiply(matrix1, matrix2, result_matrix);
  ...
  /* Check if some error occured in eRPC */
  if (g_erpc_error_occurred)
  {
    /* Exit program loop */
    break;
  }
  ...
}
```

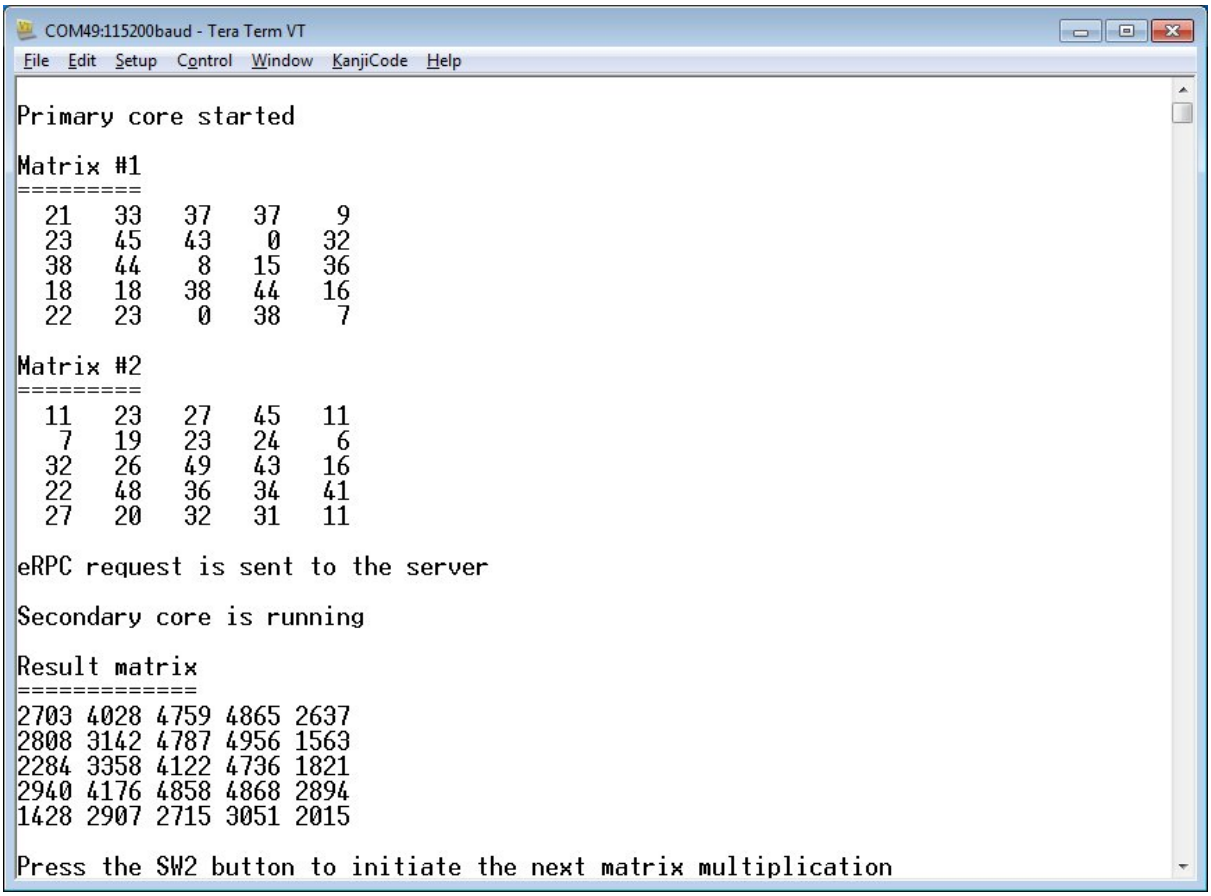
Parent topic: Multiprocessor client application

Parent topic: Multiprocessor server application

Parent topic:*Create an eRPC application*

Running the eRPC application

Follow the instructions in *Getting Started with MCUXpresso SDK* (document MCUXSDKGSUG) (located in the *<MCUXpressoSDK_install_dir>/docs* folder), to load both the primary and the secondary core images into the on-chip memory, and then effectively debug the dual-core application. After the application is running, the serial console should look like:



For multiprocessor applications that are running between PC and the target evaluation board or between two boards, follow the instructions in the accompanied example readme files that provide details about the proper board setup and the PC side setup (Python).

Parent topic:*Create an eRPC application*

Parent topic:*eRPC example*

eRPC example

This section shows how to create an example eRPC application called “Matrix multiply”, which implements one eRPC function (matrix multiply) with two function parameters (two matrices). The client-side application calls this eRPC function, and the server side performs the multiplication of received matrices. The server side then returns the result.

For example, use the NXP MIMXRT1170-EVK board as the target dual-core platform, and the IAR Embedded Workbench for ARM (EWARM) as the target IDE for developing the eRPC example.

- The primary core (CM7) runs the eRPC client.
- The secondary core (CM4) runs the eRPC server.
- RPMsg-Lite (Remote Processor Messaging Lite) is used as the eRPC transport layer.

The "Matrix multiply" application can be also run in the multi-processor setup. In other words, the eRPC client running on one SoC comunicates with the eRPC server that runs on anothe SoC, utilizing different transport channels. It is possible to run the board-to-PC example (PC as the eRPC server and a board as the eRPC client, and vice versa) and also the board-to-board example. These multiprocessor examples are prepared for selected boards only.

|Multicore application source and project files|*<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_*
|Multiprocessor application source and project files|*<MCUXpressoSDK_install_dir>/boards/<board_name>/multi*

<MCUXpressoSDK_install_dir>/boards/<board_name>/multiprocessor_examples/erpc_server_matrix_multiply_<tr

| |eRPC source files|*<MCUXpressoSDK_install_dir>/middleware/multicore/erpc/*| |RPMsg-Lite source files|*<MCUXpressoSDK_install_dir>/middleware/multicore/rpmsg_lite/*|

Designing the eRPC application

The matrix multiply application is based on calling single eRPC function that takes 2 two-dimensional arrays as input and returns matrix multiplication results as another 2 two-dimensional array. The IDL file syntax supports arrays with the dimension length set by the number only (in the current eRPC implementation). Because of this, a variable is declared in the IDL dedicated to store information about matrix dimension length, and to allow easy maintenance of the user and server code.

For a simple use of the two-dimensional array, the alias name (new type definition) for this data type has is declared in the IDL. Declaring this alias name ensures that the same data type can be used across the client and server applications.

Parent topic:*eRPC example*

Creating the IDL file

The created IDL file is located in the following folder:

<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_common/erpc_matrix_multiply/s

The created IDL file contains the following code:

```
program erpc_matrix_multiply
/*! This const defines the matrix size. The value has to be the same as the
Matrix array dimension. Do not forget to re-generate the erpc code once the
matrix size is changed in the erpc file */
const int32 matrix_size = 5;
/*! This is the matrix array type. The dimension has to be the same as the
matrix size const. Do not forget to re-generate the erpc code once the
matrix size is changed in the erpc file */
type Matrix = int32[matrix_size][matrix_size];
interface MatrixMultiplyService {
erpcMatrixMultiply(in Matrix matrix1, in Matrix matrix2, out Matrix result_matrix) ->
void
}
```

Details:

- The IDL file starts with the program name (*erpc_matrix_multiply*), and this program name is used in the naming of all generated outputs.
- The declaration and definition of the constant variable named *matrix_size* follows next. The *matrix_size* variable is used for passing information about the length of matrix dimensions to the client/server user code.
- The alias name for the two-dimensional array type (*Matrix*) is declared.
- The interface group *MatrixMultiplyService* is located at the end of the IDL file. This interface group contains only one function declaration *erpcMatrixMultiply*.

- As shown above, the function's declaration contains three parameters of Matrix type: *matrix1* and *matrix2* are input parameters, while *result_matrix* is the output parameter. Additionally, the returned data type is declared as void.

When writing the IDL file, the following order of items is recommended:

1. Program name at the top of the IDL file.
2. New data types and constants declarations.
3. Declarations of interfaces and functions at the end of the IDL file.

Parent topic:*eRPC example*

Using the eRPC generator tool

|Windows OS|<MCUXpressoSDK_install_dir>/middleware/multicore/tools/erpcgen/Windows| |Linux OS|<MCUXpressoSDK_install_dir>/middleware/multicore/tools/erpcgen/Linux_x64

<MCUXpressoSDK_install_dir>/middleware/multicore/tools/erpcgen/Linux_x86

| |Mac OS|<MCUXpressoSDK_install_dir>/middleware/multicore/tools/erpcgen/Mac|

The files for the "Matrix multiply" example are pre-generated and already a part of the application projects. The following section describes how they have been created.

- The easiest way to create the shim code is to copy the erpcgen application to the same folder where the IDL file (*.erpc) is located; then run the following command:

  erpcgen <IDL_file>.erpc

- In the "Matrix multiply" example, the command should look like:

  erpcgen erpc_matrix_multiply.erpc

Additionally, another method to create the shim code is to execute the eRPC application using input commands:

- "-?"/"—help" – Shows supported commands.
- "-o <filePath>"/"—output<filePath>" – Sets the output directory.

For example,

```
<path_to_erpcgen>/erpcgen –o <path_to_output>
<path_to_IDL>/<IDL_file_name>.erpc
```

For the "Matrix multiply" example, when the command is executed from the default erpcgen location, it looks like:

erpcgen –o

../../../../../boards/evkmimxrt1170/multicore_examples/erpc_common/erpc_matrix_multiply/service
../../../../../boards/evkmimxrt1170/multicore_examples/erpc_common/erpc_matrix_multiply/service/erpc_matrix_mu

In both cases, the following four files are generated into the *<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_common/erpc_matrix_multiply/service folder.*

- erpc_matrix_multiply.h
- erpc_matrix_multiply_client.cpp
- erpc_matrix_multiply_server.h
- erpc_matrix_multiply_server.cpp

For multiprocessor examples, the eRPC file and pre-generated files can be found in the *<MCUXpressoSDK_install_dir>/boards/<board_name>/multiprocessor_examples/erpc_common/erpc_matrix_multiply/serv* folder.

For Linux OS users:

- Do not forget to set the permissions for the eRPC generator application.
- Run the application as `./erpcgen`... instead of as erpcgen ....

Parent topic:*eRPC example*

Create an eRPC application

This section describes a generic way to create a client/server eRPC application:

1. **Design the eRPC application:** Decide which data types are sent between applications, and define functions that send/receive this data.
2. **Create the IDL file:** The IDL file contains information about data types and functions used in an eRPC application, and is written in the IDL language.
3. **Use the eRPC generator tool:** This tool takes an IDL file and generates the shim code for the client and the server-side applications.
4. **Create an eRPC application:**
    1. Create two projects, where one project is for the client side (primary core) and the other project is for the server side (secondary core).
    2. Add generated files for the client application to the client project, and add generated files for the server application to the server project.
    3. Add infrastructure files.
    4. Add user code for client and server applications.
    5. Set the client and server project options.
5. **Run the eRPC application:** Run both the server and the client applications. Make sure that the server has been run before the client request was sent.

A specific example follows in the next section.

Multicore server application

The "Matrix multiply" eRPC server project is located in the following folder:

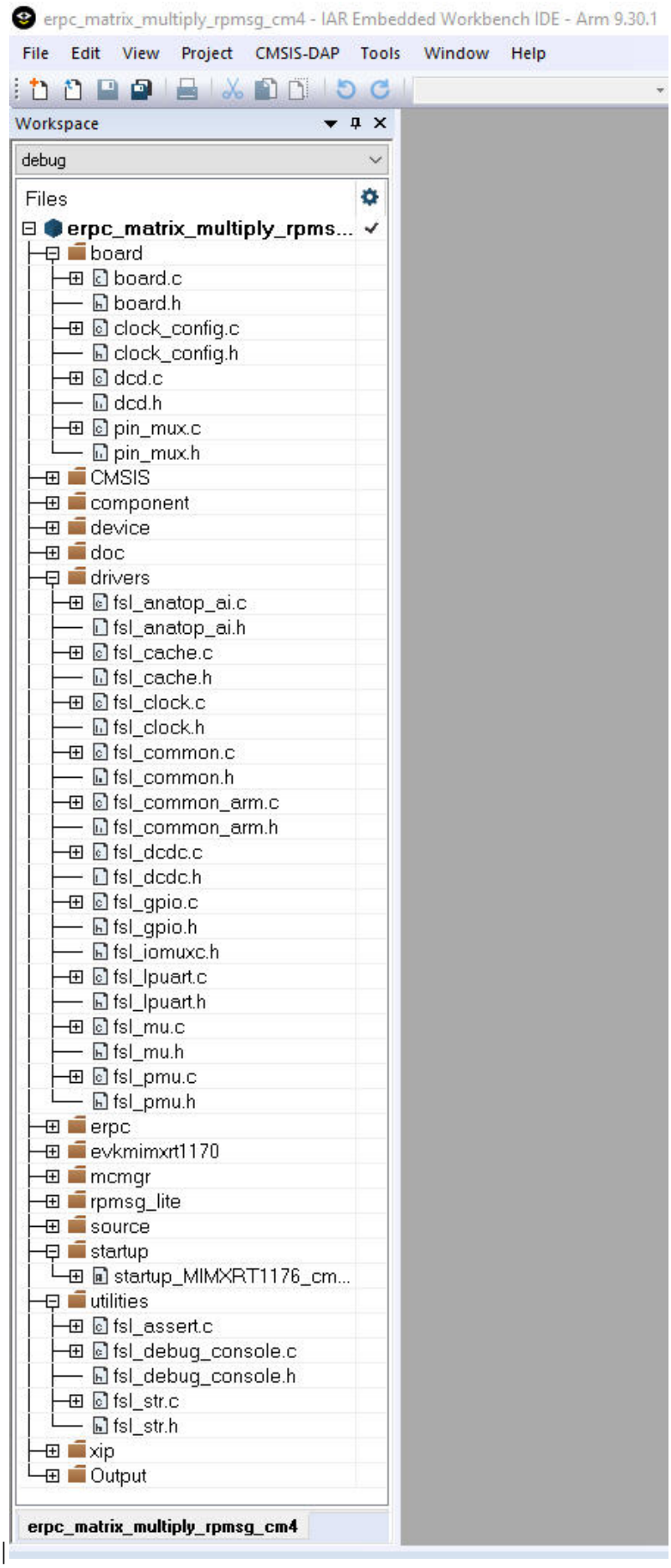
<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_matrix_multiply_rpmsg/cm4/iar

The project files for the eRPC server have the `_cm4` suffix.

Server project basic source files

The startup files, board-related settings, peripheral drivers, and utilities belong to the basic project source files and form the skeleton of all MCUXpresso SDK applications. These source files are located in:

- *<MCUXpressoSDK_install_dir>/devices/<device>*
- *<MCUXpressoSDK_install_dir>/boards/<board_name>/multicore_examples/<example_name>/*


erpc_matrix_multiply_rpmsg_cm4 - IAR Embedded Workbench IDE - Arm 9.30.1
File
Edit
View
Project
CMSIS-DAP
Tools
Window
Help
Workspace
debug
Files
erpc_matrix_multiply_rpms...
board
board.c
board.h
clock_config.c
clock_config.h
dcd.c
dcd.h
pin_mux.c
pin_mux.h
CMSIS
component
device
doc
drivers
fsl_anatop_ai.c
fsl_anatop_ai.h
fsl_cache.c
fsl_cache.h
fsl_clock.c
fsl_clock.h
fsl_common.c
fsl_common.h
fsl_common_arm.c
fsl_common_arm.h
fsl_dcdc.c
fsl_dcdc.h
fsl_gpio.c
fsl_gpio.h
fsl_iomuxc.h
fsl_lpuart.c
fsl_lpuart.h
fsl_mu.c
fsl_mu.h
fsl_pmu.c
fsl_pmu.h
erpc
evkmimxrt1170
mcmgr
rpmsg_lite
source
startup
startup_MIMXRT1176_cm...
utilities
fsl_assert.c
fsl_debug_console.c
fsl_debug_console.h
fsl_str.c
fsl_str.h
xip
Output
erpc_matrix_multiply_rpmsg_cm4

|

Parent topic: Multicore server application

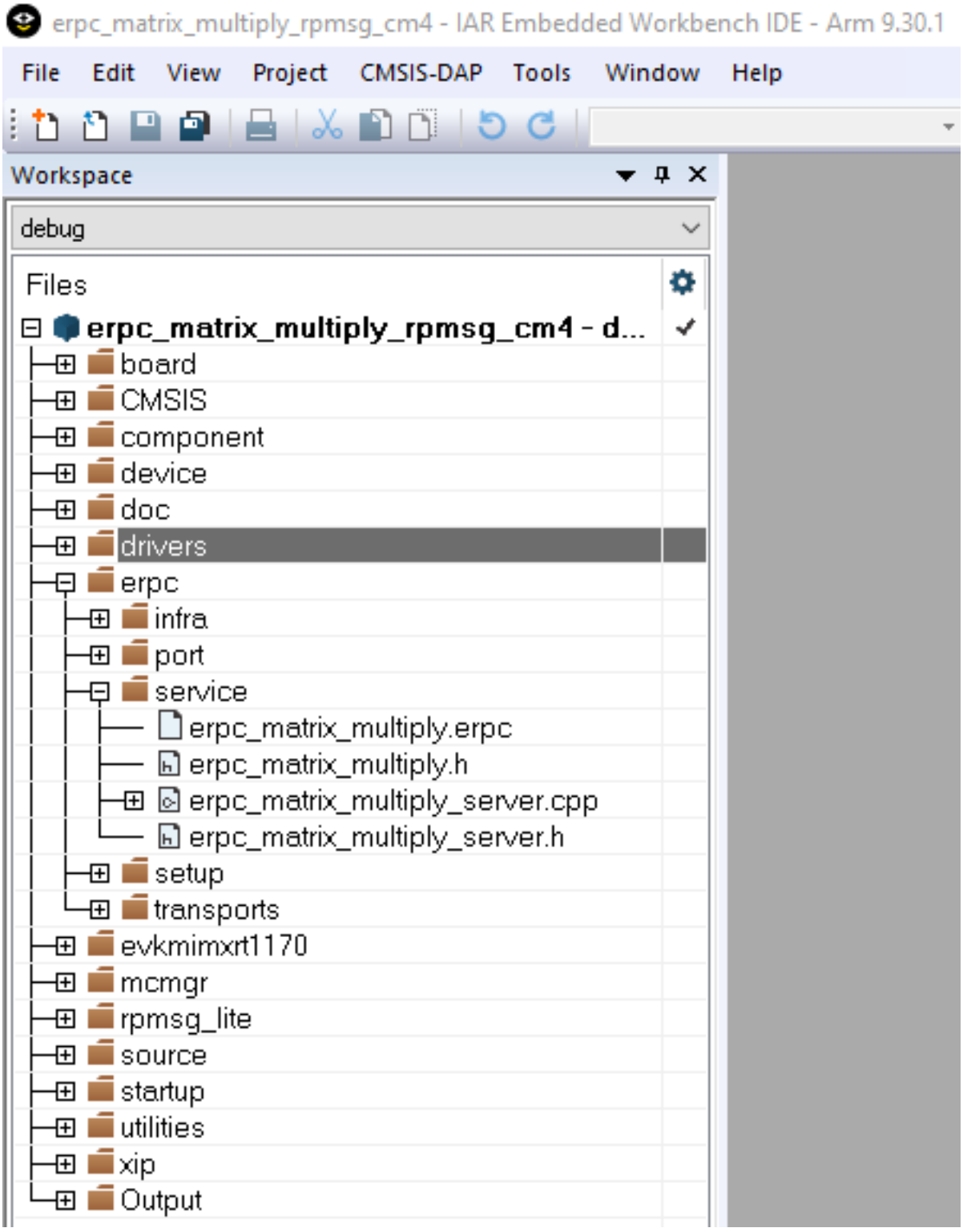
Server related generated files

The server-related generated files are:

- erpc_matric_multiply.h
- erpc_matrix_multiply_server.h
- erpc_matrix_multiply_server.cpp

The server-related generated files contain the shim code for functions and data types declared in the IDL file. These files also contain functions for the identification of client requested functions, data deserialization, calling requested function's implementations, and data serialization and return, if requested by the client. These shim code files can be found in the following folder:

<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_common/erpc_matrix_multiply/s



Parent topic: Multicore server application

Server infrastructure files

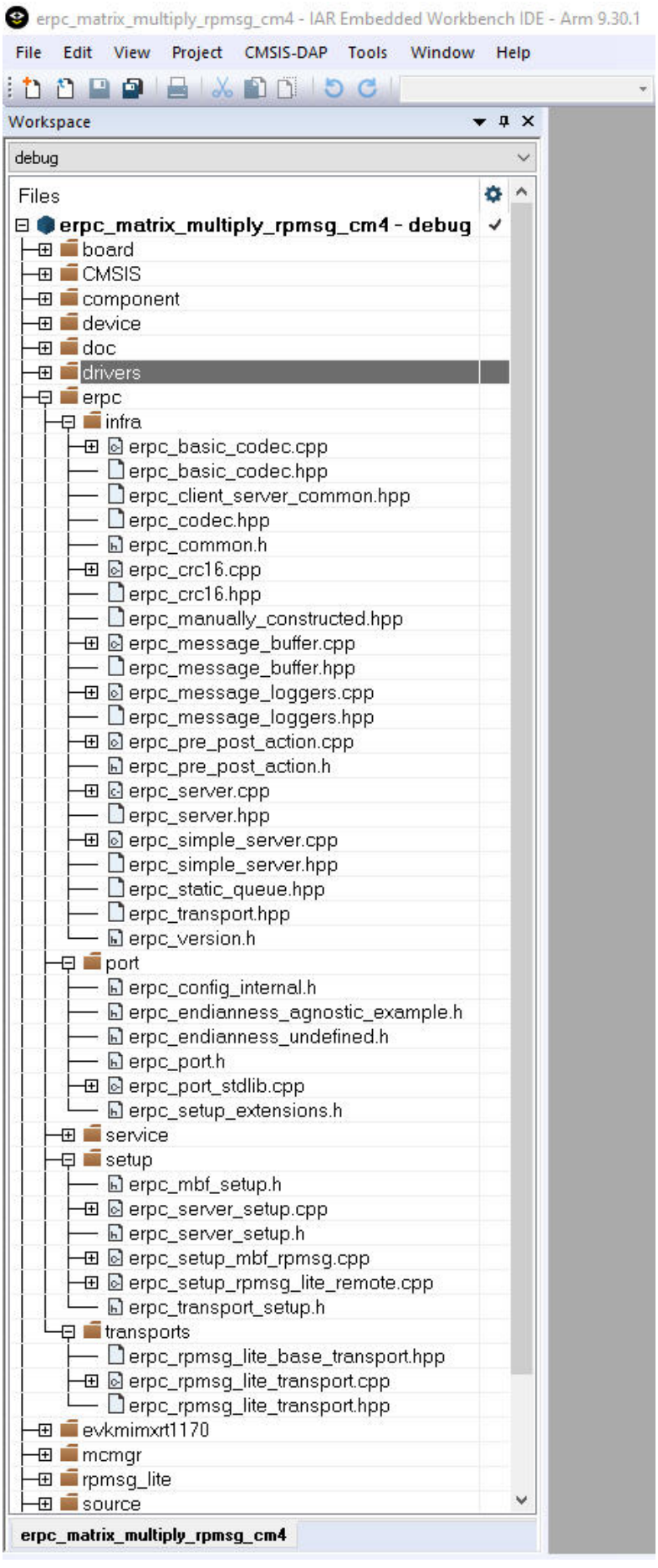
The eRPC infrastructure files are located in the following folder:

<MCUXpressoSDK_install_dir>/middleware/multicore/erpc/erpc_c

The **erpc_c** folder contains files for creating eRPC client and server applications in the *C/C++*

language. These files are distributed into subfolders.

- The **infra** subfolder contains C++ infrastructure code used to build server and client applications.
  - Four files, erpc_server.hpp, erpc_server.cpp, erpc_simple_server.hpp, and erpc_simple_server.cpp, are used for running the eRPC server on the server-side applications. The simple server is currently the only implementation of the server, and its role is to catch client requests, identify and call requested functions, and send data back when requested.
  - Three files (erpc_codec.hpp, erpc_basic_codec.hpp, and erpc_basic_codec.cpp) are used for codecs. Currently, the basic codec is the initial and only implementation of the codecs.
  - The erpc_common.hpp file is used for common eRPC definitions, typedefs, and enums.
  - The erpc_manually_constructed.hpp file is used for allocating static storage for the used objects.
  - Message buffer files are used for storing serialized data: erpc_message_buffer.h and erpc_message_buffer.cpp.
  - The erpc_transport.h file defines the abstract interface for transport layer.
- The **port** subfolder contains the eRPC porting layer to adapt to different environments.
  - erpc_port.h file contains definition of erpc_malloc() and erpc_free() functions.
  - erpc_port_stdlib.cpp file ensures adaptation to stdlib.
  - erpc_config_internal.h internal erpc configuration file.
- The **setup** subfolder contains a set of plain C APIs that wrap the C++ infrastructure, providing client and server init and deinit routines that greatly simplify eRPC usage in C-based projects. No knowledge of C++ is required to use these APIs.
  - The erpc_server_setup.h and erpc_server_setup.cpp files needs to be added into the "Matrix multiply" example project to demonstrate the use of C-wrapped functions in this example.
  - The erpc_transport_setup.h and erpc_setup_rpmsg_lite_remote.cpp files needs to be added into the project in order to allow the C-wrapped function for transport layer setup.
  - The erpc_mbf_setup.h and erpc_setup_mbf_rpmsg.cpp files needs to be added into the project in order to allow message buffer factory usage.
- The **transports** subfolder contains transport classes for the different methods of communication supported by eRPC. Some transports are applicable only to host PCs, while others are applicable only to embedded or multicore systems. Most transports have corresponding client and server setup functions in the setup folder.
  - RPMsg-Lite is used as the transport layer for the communication between cores, erpc_rpmsg_lite_base_transport.hpp, erpc_rpmsg_lite_transport.hpp, and erpc_rpmsg_lite_transport.cpp files need to be added into the server project.


erpc_matrix_multiply_rpmsg_cm4 - IAR Embedded Workbench IDE - Arm 9.30.1
File Edit View Project CMSIS-DAP Tools Window Help
Workspace
debug
Files
erpc_matrix_multiply_rpmsg_cm4 - debug
board
CMSIS
component
device
doc
drivers
erpc
infra
erpc_basic_codec.cpp
erpc_basic_codec.hpp
erpc_client_server_common.hpp
erpc_codec.hpp
erpc_common.h
erpc_crc16.cpp
erpc_crc16.hpp
erpc_manually_constructed.hpp
erpc_message_buffer.cpp
erpc_message_buffer.hpp
erpc_message_loggers.cpp
erpc_message_loggers.hpp
erpc_pre_post_action.cpp
erpc_pre_post_action.h
erpc_server.cpp
erpc_server.hpp
erpc_simple_server.cpp
erpc_simple_server.hpp
erpc_static_queue.hpp
erpc_transport.hpp
erpc_version.h
port
erpc_config_internal.h
erpc_endianness_agnostic_example.h
erpc_endianness_undefined.h
erpc_port.h
erpc_port_stdlib.cpp
erpc_setup_extensions.h
service
setup
erpc_mbf_setup.h
erpc_server_setup.cpp
erpc_server_setup.h
erpc_setup_mbf_rpmsg.cpp
erpc_setup_rpmsg_lite_remote.cpp
erpc_transport_setup.h
transports
erpc_rpmsg_lite_base_transport.hpp
erpc_rpmsg_lite_transport.cpp
erpc_rpmsg_lite_transport.hpp
evkmimxrt1170
mcmgr
rpmsg_lite
source
erpc_matrix_multiply_rpmsg_cm4

|

Parent topic: Multicore server application

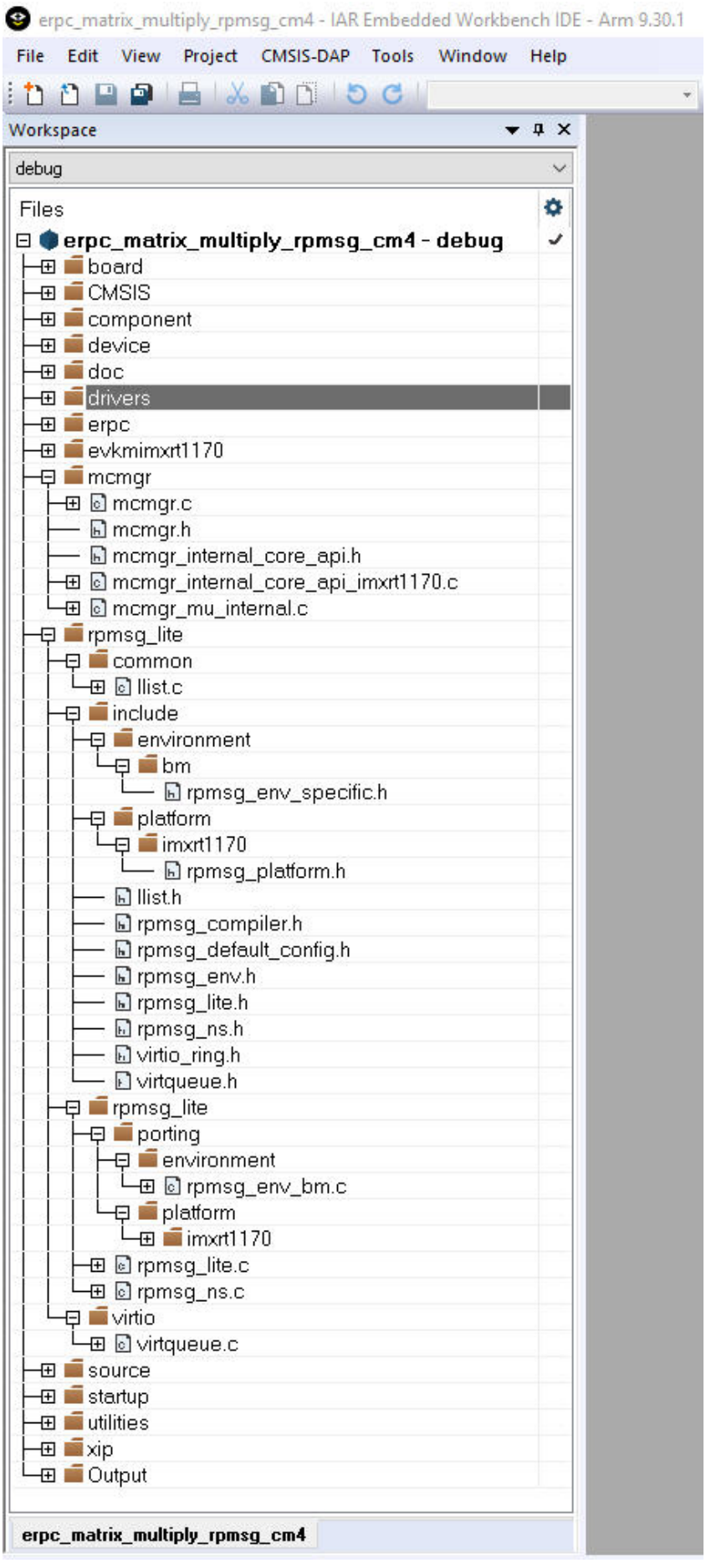
Server multicore infrastructure files

Because of the RPMsg-Lite (transport layer), it is also necessary to include RPMsg-Lite related files, which are in the following folder:

<MCUXpressoSDK_install_dir>/middleware/multicore/rpmsg_lite/

The multicore example applications also use the Multicore Manager software library to control the secondary core startup and shutdown. These source files are located in the following folder:

<MCUXpressoSDK_install_dir>/middleware/multicore/mcmgr/


erpc_matrix_multiply_rpmsg_cm4 - IAR Embedded Workbench IDE - Arm 9.30.1
File
Edit
View
Project
CMSIS-DAP
Tools
Window
Help
Workspace
debug
Files
erpc_matrix_multiply_rpmsg_cm4 - debug
board
CMSIS
component
device
doc
drivers
erpc
evkmimxrt1170
mcmgr
mcmgr.c
mcmgr.h
mcmgr_internal_core_api.h
mcmgr_internal_core_api_imxrt1170.c
mcmgr_mu_internal.c
rpmsg_lite
common
llist.c
include
environment
bm
rpmsg_env_specific.h
platform
imxrt1170
rpmsg_platform.h
llist.h
rpmsg_compiler.h
rpmsg_default_config.h
rpmsg_env.h
rpmsg_lite.h
rpmsg_ns.h
virtio_ring.h
virtqueue.h
rpmsg_lite
porting
environment
rpmsg_env_bm.c
platform
imxrt1170
rpmsg_lite.c
rpmsg_ns.c
virtio
virtqueue.c
source
startup
utilities
xip
Output
erpc_matrix_multiply_rpmsg_cm4

|

Parent topic: Multicore server application

Server user code

The server's user code is stored in the main_core1.c file, located in the following folder:

<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_matrix_multiply_rpmsg/cm4

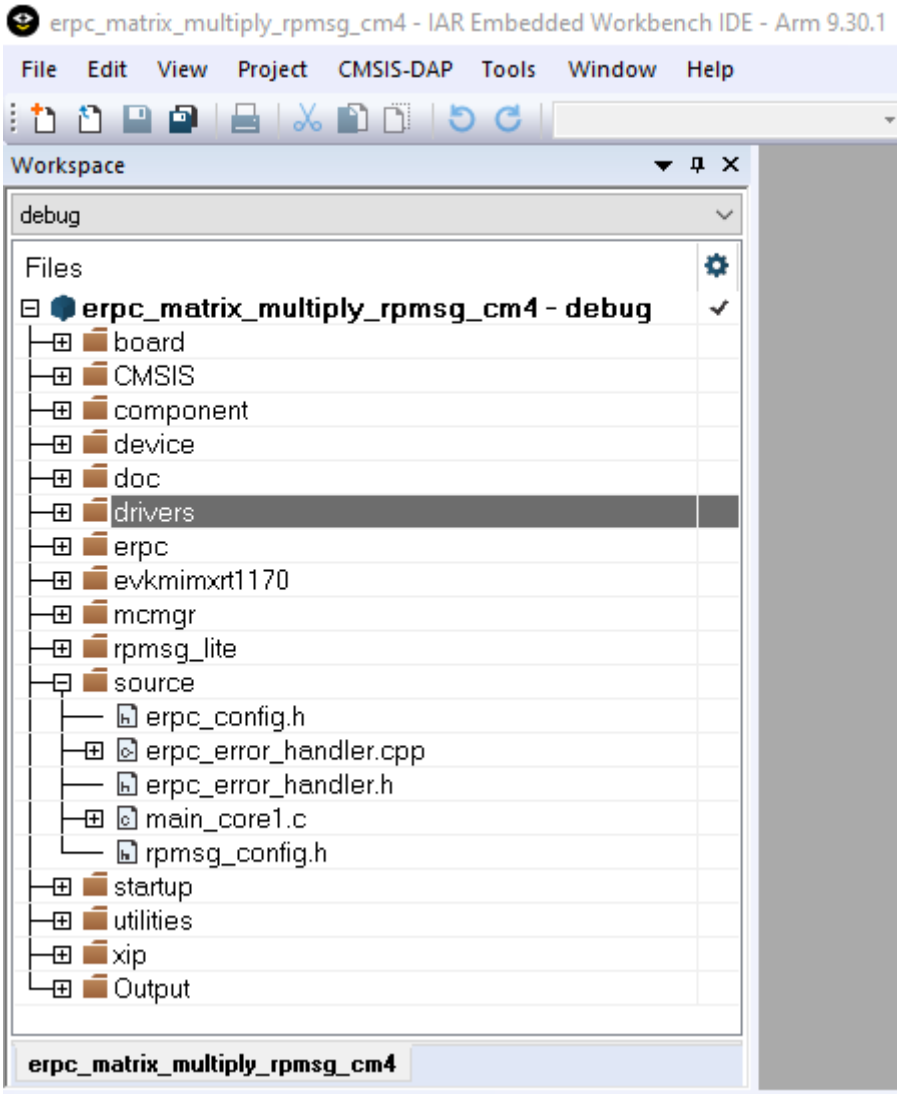
The main_core1.c file contains two functions:

- The **main()** function contains the code for the target board and eRPC server initialization. After the initialization, the matrix multiply service is added and the eRPC server waits for client's requests in the while loop.
- The **erpcMatrixMultiply()** function is the user implementation of the eRPC function defined in the IDL file.
- There is the possibility to write the application-specific eRPC error handler. The eRPC error handler of the matrix multiply application is implemented in the erpc_error_handler.h and erpc_error_handler.cpp files.

The eRPC-relevant code is captured in the following code snippet:

```
/* erpcMatrixMultiply function user implementation */
void erpcMatrixMultiply(const Matrix *matrix1, const Matrix *matrix2, Matrix *result_matrix)
{
  ...
}
int main()
{
  ...
  /* RPMsg-Lite transport layer initialization */
  erpc_transport_t transport;
  transport = erpc_transport_rpmsg_lite_remote_init(src, dst, (void*)startupData,
  ERPC_TRANSPORT_RPMSG_LITE_LINK_ID, SignalReady, NULL);
  ...
  /* MessageBufferFactory initialization */
  erpc_mbf_t message_buffer_factory;
  message_buffer_factory = erpc_mbf_rpmsg_init(transport);
  ...
  /* eRPC server side initialization */
  erpc_server_t server;
  server = erpc_server_init(transport, message_buffer_factory);
  ...
  /* Adding the service to the server */
  erpc_service_t service = create_MatrixMultiplyService_service();
  erpc_add_service_to_server(server, service);
  ...
  while (1)
  {
    /* Process eRPC requests */
    erpc_status_t status = erpc_server_poll(server);
    /* handle error status */
    if (status != kErpcStatus_Success)
    {
      /* print error description */
      erpc_error_handler(status, 0);
      ...
    }
    ...
  }
}
```

Except for the application main file, there are configuration files for the RPMsg-Lite (rpmsg_config.h) and eRPC (erpc_config.h), located in the *<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/ erpc_matrix_multiply_rpmsg* folder.



Parent topic: Multicore server application

Parent topic:*Create an eRPC application*

Multicore client application

The "Matrix multiply" eRPC client project is located in the following folder:

<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_matrix_multiply_rpmsg/cm7/iar

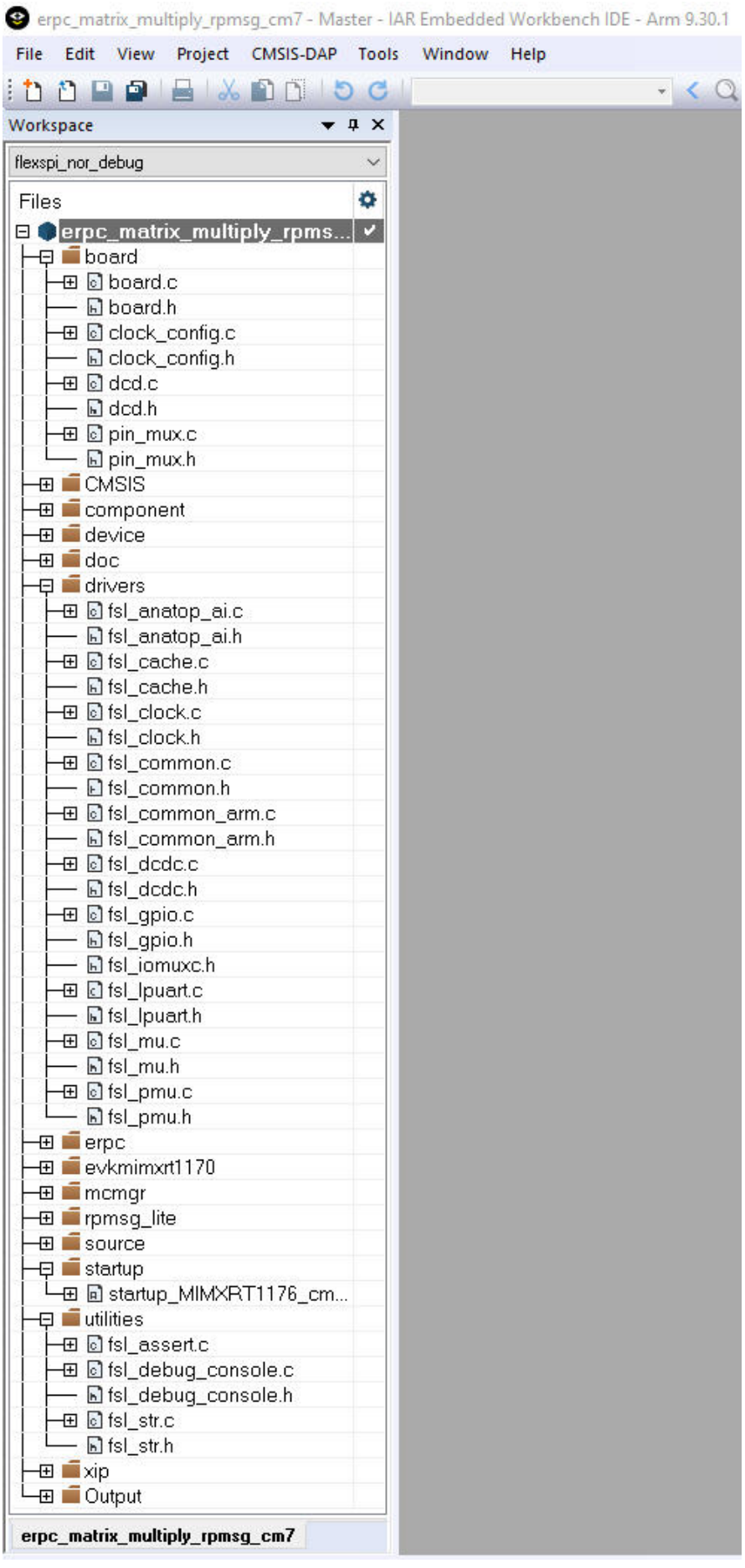
Project files for the eRPC client have the _cm7 suffix.

Client project basic source files

The startup files, board-related settings, peripheral drivers, and utilities belong to the basic project source files and form the skeleton of all MCUXpresso SDK applications. These source files are located in the following folders:

- *<MCUXpressoSDK_install_dir>/devices/<device>*

- *<MCUXpressoSDK_install_dir>/boards/<board_name>/multicore_examples/<example_name>/*


erpc_matrix_multiply_rpmsg_cm7 - Master - IAR Embedded Workbench IDE - Arm 9.30.1
File
Edit
View
Project
CMSIS-DAP
Tools
Window
Help
Workspace
flexspi_nor_debug
Files
erpc_matrix_multiply_rpms...
board
board.c
board.h
clock_config.c
clock_config.h
dcd.c
dcd.h
pin_mux.c
pin_mux.h
CMSIS
component
device
doc
drivers
fsl_anatop_ai.c
fsl_anatop_ai.h
fsl_cache.c
fsl_cache.h
fsl_clock.c
fsl_clock.h
fsl_common.c
fsl_common.h
fsl_common_arm.c
fsl_common_arm.h
fsl_dcdc.c
fsl_dcdc.h
fsl_gpio.c
fsl_gpio.h
fsl_iomuxc.h
fsl_lpuart.c
fsl_lpuart.h
fsl_mu.c
fsl_mu.h
fsl_pmu.c
fsl_pmu.h
erpc
evkmimxrt1170
mcmgr
rpmsg_lite
source
startup
startup_MIMXRT1176_cm...
utilities
fsl_assert.c
fsl_debug_console.c
fsl_debug_console.h
fsl_str.c
fsl_str.h
xip
Output
erpc_matrix_multiply_rpmsg_cm7

|

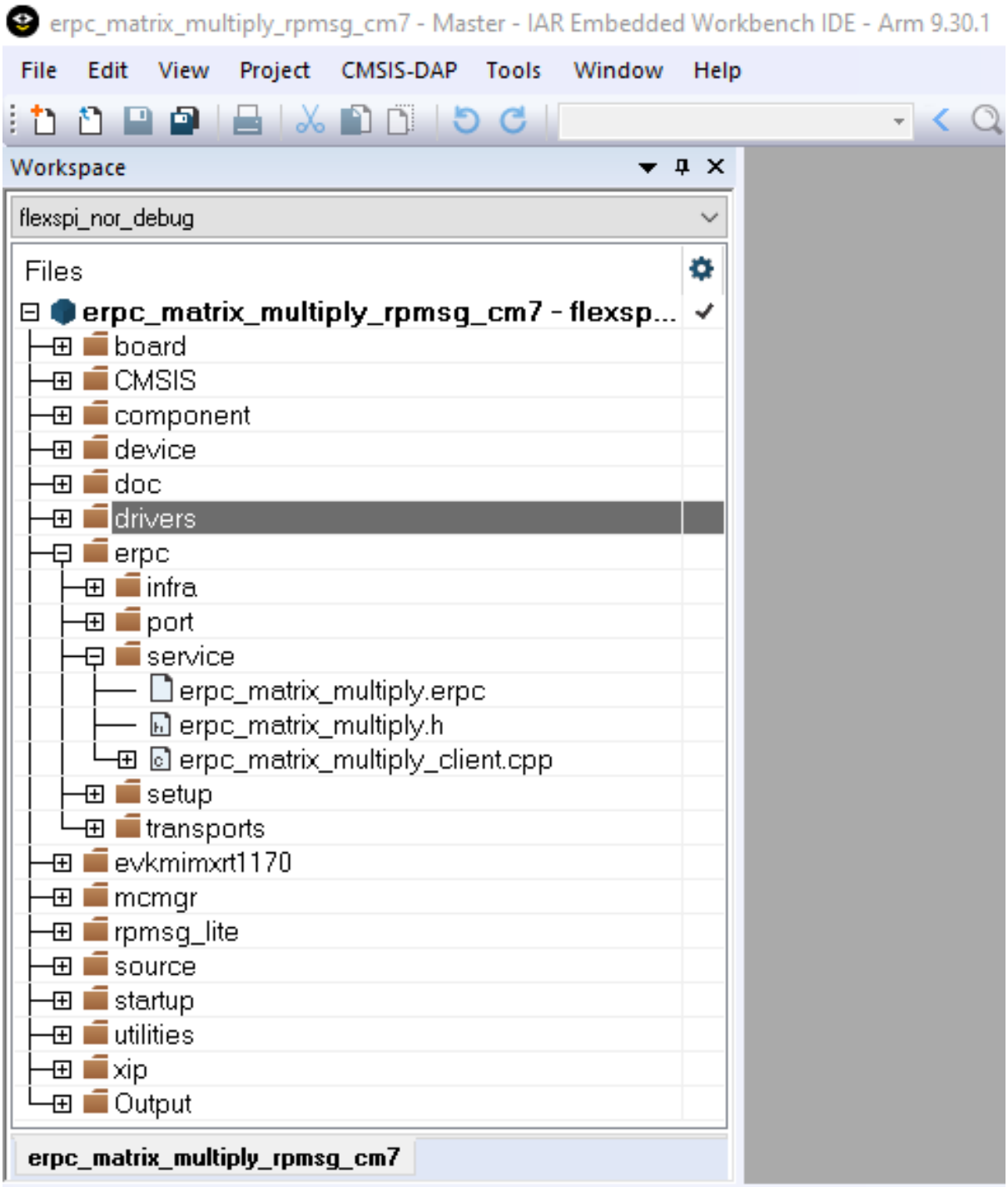
Parent topic: Multicore client application

Client-related generated files

The client-related generated files are:

- erpc_matric_multiply.h
- erpc_matrix_multiply_client.cpp

These files contain the shim code for the functions and data types declared in the IDL file. These functions also call methods for codec initialization, data serialization, performing eRPC requests, and de-serializing outputs into expected data structures (if return values are expected). These shim code files can be found in the *<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_common/erpc_matrix_multiply/service/* folder.



Parent topic: Multicore client application

Client infrastructure files

The eRPC infrastructure files are located in the following folder:

<MCUXpressoSDK_install_dir>/middleware/multicore/erpc/erpc_c

The **erpc_c** folder contains files for creating eRPC client and server applications in the C/C++ language. These files are distributed into subfolders.

- The **infra** subfolder contains C++ infrastructure code used to build server and client applications.

- Two files, erpc_client_manager.h and erpc_client_manager.cpp, are used for managing the client-side application. The main purpose of the client files is to create, perform, and release eRPC requests.
- Three files (erpc_codec.hpp, erpc_basic_codec.hpp, and erpc_basic_codec.cpp) are used for codecs. Currently, the basic codec is the initial and only implementation of the codecs.
- erpc_common.h file is used for common eRPC definitions, typedefs, and enums.
- erpc_manually_constructed.hpp file is used for allocating static storage for the used objects.
- Message buffer files are used for storing serialized data: erpc_message_buffer.hpp and erpc_message_buffer.cpp.
- erpc_transport.hpp file defines the abstract interface for transport layer.

The **port** subfolder contains the eRPC porting layer to adapt to different environments.

- erpc_port.h file contains definition of erpc_malloc() and erpc_free() functions.
- erpc_port_stdlib.cpp file ensures adaptation to stdlib.
- erpc_config_internal.h internal eRPC configuration file.

The **setup** subfolder contains a set of plain C APIs that wrap the C++ infrastructure, providing client and server init and deinit routines that greatly simplify eRPC usage in C-based projects. No knowledge of C++ is required to use these APIs.

- erpc_client_setup.h and erpc_client_setup.cpp files needs to be added into the "Matrix multiply" example project to demonstrate the use of C-wrapped functions in this example.
- erpc_transport_setup.h and erpc_setup_rpmsg_lite_master.cpp files needs to be added into the project in order to allow C-wrapped function for transport layer setup.
- erpc_mbf_setup.h and erpc_setup_mbf_rpmsg.cpp files needs to be added into the project in order to allow message buffer factory usage.

The **transports** subfolder contains transport classes for the different methods of communication supported by eRPC. Some transports are applicable only to host PCs, while others are applicable only to embedded or multicore systems. Most transports have corresponding client and server setup functions, in the setup folder.

- RPMsg-Lite is used as the transport layer for the communication between cores, erpc_rpmsg_lite_base_transport.hpp, erpc_rpmsg_lite_transport.hpp, and erpc_rpmsg_lite_transport.cpp files needs to be added into the client project.

erpc_matrix_multiply_rpmsg_cm7 - Master - IAR Embedded Workbench IDE - Arm 9.30.1
File Edit View Project CMSIS-DAP Tools Window Help
Workspace
flexspi_nor_debug
Files
erpc_matrix_multiply_rpmsg_cm7 - flexspi_n...
board
CMSIS
component
device
doc
drivers
erpc
infra
erpc_basic_codec.cpp
erpc_basic_codec.hpp
erpc_client_manager.cpp
erpc_client_manager.h
erpc_client_server_common.hpp
erpc_codec.hpp
erpc_common.h
erpc_crc16.cpp
erpc_crc16.hpp
erpc_manually_constructed.hpp
erpc_message_buffer.cpp
erpc_message_buffer.hpp
erpc_message_loggers.cpp
erpc_message_loggers.hpp
erpc_pre_post_action.cpp
erpc_pre_post_action.h
erpc_static_queue.hpp
erpc_transport.hpp
erpc_version.h
port
erpc_config_internal.h
erpc_endianness_agnostic_example.h
erpc_endianness_undefined.h
erpc_port.h
erpc_port_stdlib.cpp
erpc_setup_extensions.h
service
setup
erpc_client_setup.cpp
erpc_client_setup.h
erpc_mbf_setup.h
erpc_setup_mbf_rpmsg.cpp
erpc_setup_rpmsg_lite_master.cpp
erpc_transport_setup.h
transports
erpc_rpmsg_lite_base_transport.hpp
erpc_rpmsg_lite_transport.cpp
erpc_rpmsg_lite_transport.hpp
evkmimxrt1170
mcmgr
rpmsg_lite
source
startup
utilities
erpc_matrix_multiply_rpmsg_cm7

|

Parent topic: Multicore client application

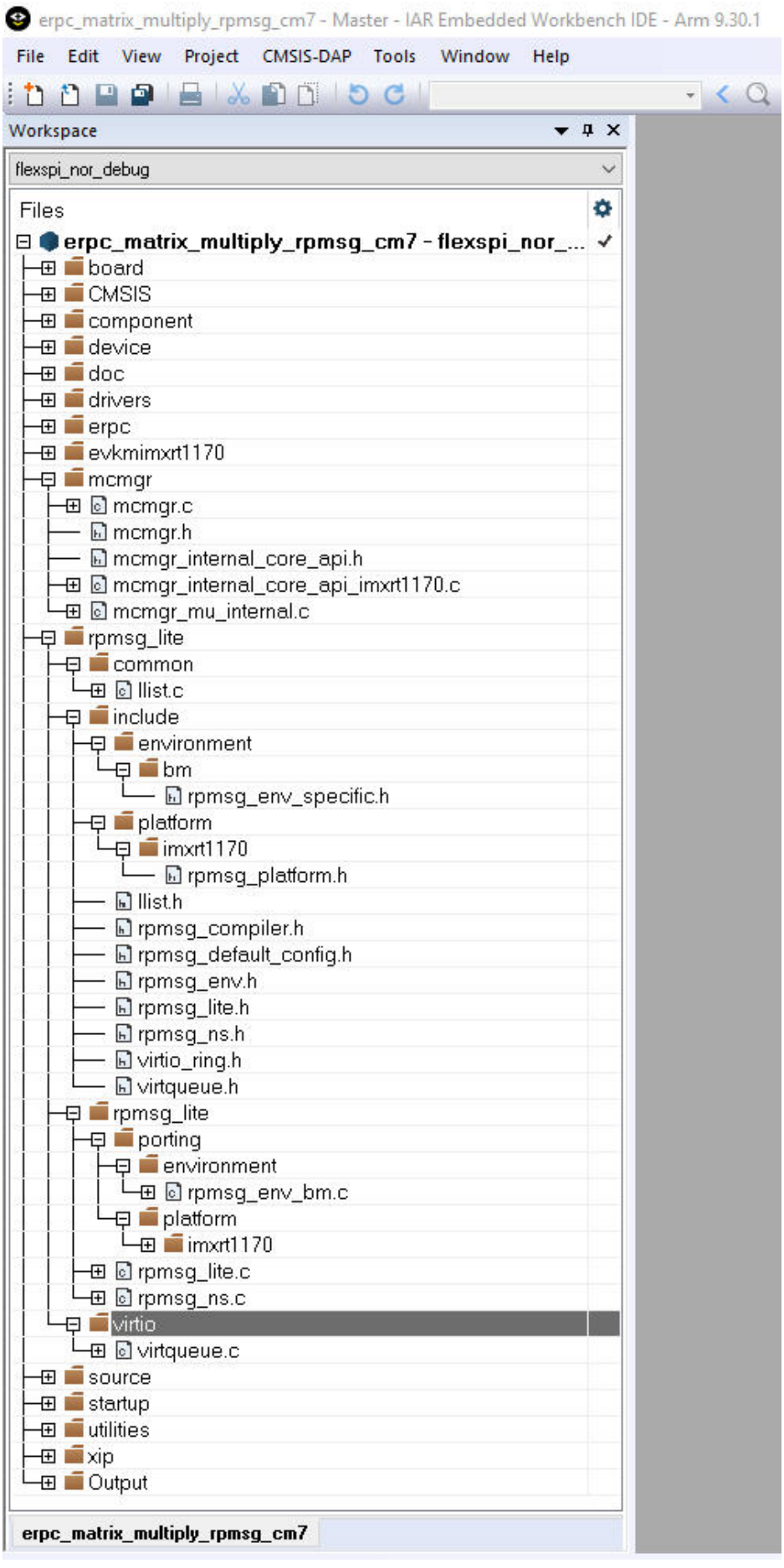
Client multicore infrastructure files

Because of the RPMsg-Lite (transport layer), it is also necessary to include RPMsg-Lite related files, which are in the following folder:

<MCUXpressoSDK_install_dir>/middleware/multicore/rpmsg_lite/

The multicore example applications also use the Multicore Manager software library to control the secondary core startup and shutdown. These source files are located in the following folder:

<MCUXpressoSDK_install_dir>/middleware/multicore/mcmgr/


erpc_matrix_multiply_rpmsg_cm7 - Master - IAR Embedded Workbench IDE - Arm 9.30.1
File
Edit
View
Project
CMSIS-DAP
Tools
Window
Help
Workspace
flexspi_nor_debug
Files
erpc_matrix_multiply_rpmsg_cm7 - flexspi_nor_...
board
CMSIS
component
device
doc
drivers
erpc
evkmimxrt1170
mcmgr
mcmgr.c
mcmgr.h
mcmgr_internal_core_api.h
mcmgr_internal_core_api_imxrt1170.c
mcmgr_mu_internal.c
rpmsg_lite
common
llist.c
include
environment
bm
rpmsg_env_specific.h
platform
imxrt1170
rpmsg_platform.h
llist.h
rpmsg_compiler.h
rpmsg_default_config.h
rpmsg_env.h
rpmsg_lite.h
rpmsg_ns.h
virtio_ring.h
virtqueue.h
rpmsg_lite
porting
environment
rpmsg_env_bm.c
platform
imxrt1170
rpmsg_lite.c
rpmsg_ns.c
virtio
virtqueue.c
source
startup
utilities
xip
Output
erpc_matrix_multiply_rpmsg_cm7

|

Parent topic: Multicore client application

Client user code

The client's user code is stored in the main_core0.c file, located in the following folder:

<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_example/erpc_matrix_multiply_rpmsg/cm7

The main_core0.c file contains the code for target board and eRPC initialization.

- After initialization, the secondary core is released from reset.
- When the secondary core is ready, the primary core initializes two matrix variables.
- The erpcMatrixMultiply eRPC function is called to issue the eRPC request and get the result.

It is possible to write the application-specific eRPC error handler. The eRPC error handler of the matrix multiply application is implemented in erpc_error_handler.h and erpc_error_handler.cpp files.

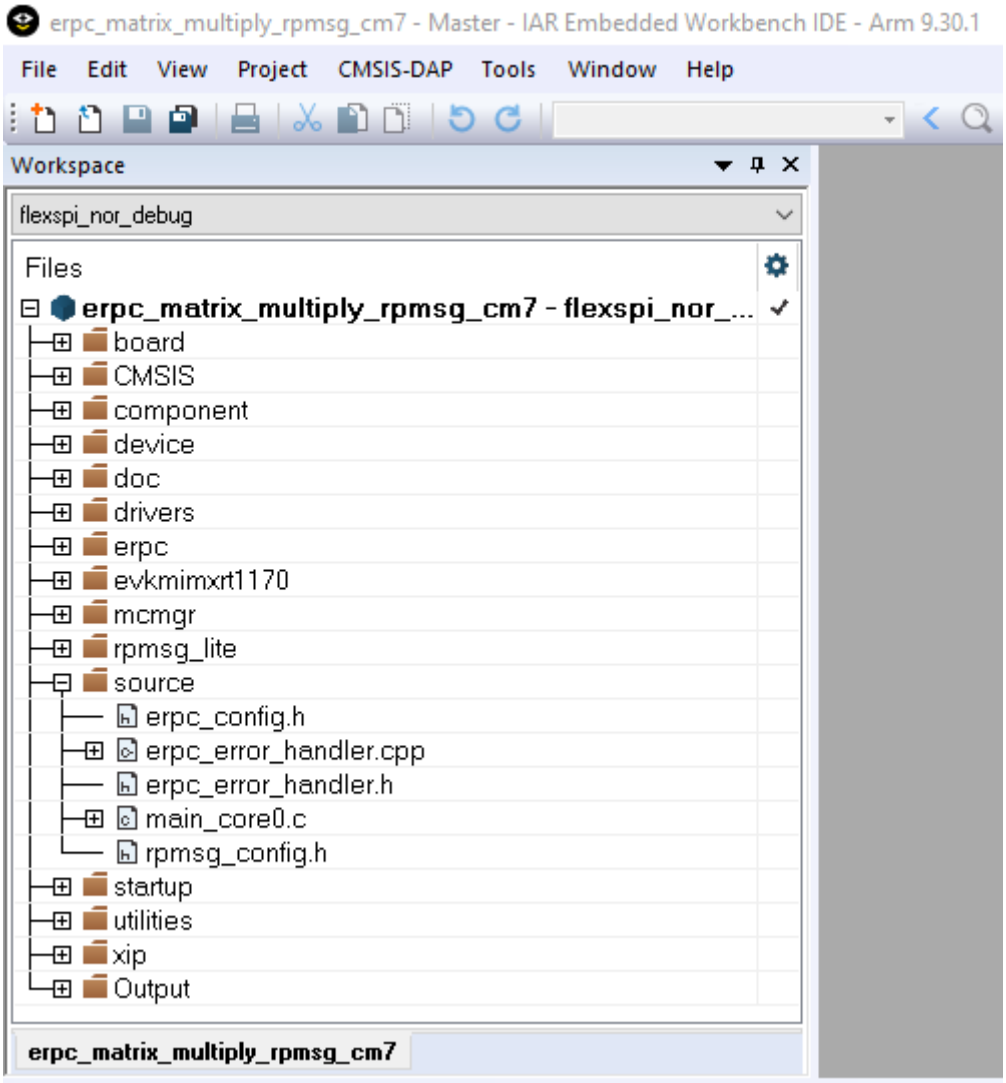
The matrix multiplication can be issued repeatedly, when pressing a software board button.

The eRPC-relevant code is captured in the following code snippet:

```
...
extern bool g_erpc_error_occurred;
...
/* Declare matrix arrays */
Matrix matrix1 = {0}, matrix2 = {0}, result_matrix = {0};
...
/* RPMsg-Lite transport layer initialization */
erpc_transport_t transport;
transport = erpc_transport_rpmsg_lite_master_init(src, dst,
ERPC_TRANSPORT_RPMSG_LITE_LINK_ID);
...
/* MessageBufferFactory initialization */
erpc_mbf_t message_buffer_factory;
message_buffer_factory = erpc_mbf_rpmsg_init(transport);
...
/* eRPC client side initialization */
erpc_client_t client;
client = erpc_client_init(transport, message_buffer_factory);
...
/* Set default error handler */
erpc_client_set_error_handler(client, erpc_error_handler);
...
while (1)
{
  /* Invoke the erpcMatrixMultiply function */
  erpcMatrixMultiply(matrix1, matrix2, result_matrix);
  ...
  /* Check if some error occured in eRPC */
  if (g_erpc_error_occurred)
  {
    /* Exit program loop */
    break;
  }
  ...
}
```

Except for the application main file, there are configuration files for the RPMsg-Lite (rpmsg_config.h) and eRPC (erpc_config.h), located in the following folder:

<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_matrix_multiply_rpmsg



|

|

Parent topic: Multicore client application

Parent topic:*Create an eRPC application*

Multiprocessor server application

The "Matrix multiply" eRPC server project for multiprocessor applications is located in the *<MCUXpressoSDK_install_dir»/boards/<board_name>/multiprocessor_examples/erpc_server_matrix_multiply_<transport_layer>* folder.

Most of the multiprocessor application setup is the same as for the multicore application. The multiprocessor server application requires server-related generated files (server shim code), server infrastructure files, and the server user code. There is no need for server multicore infrastructure files (MCMGR and RPMsg-Lite). The RPMsg-Lite transport layer is replaced either by SPI or UART transports. The following table shows the required transport-related files per each transport type.

|SPI|<eRPC base directory>/erpc_c/setup/erpc_setup_(d)spi_slave.cpp

<eRPC base directory>/erpc_c/transports/erpc_(d)spi_slave_transport.hpp

<eRPC base directory>/erpc_c/transports/erpc_(d)spi_slave_transport.cpp

| |UART|<eRPC base directory>/erpc_c/setup/erpc_setup_uart_cmsis.cpp

<eRPC base directory>/erpc_c/transports/erpc_uart_cmsis_transport.hpp

<eRPC base directory>/erpc_c/transports/erpc_uart_cmsis_transport.cpp

|

Server user code

The server's user code is stored in the main_server.c file, located in the *<MCUXpressoSDK_install_dir>/boards/<board_name>/multiprocessor_examples/erpc_server_matrix_multiply_<transport_lay* folder.

The eRPC-relevant code with UART as a transport is captured in the following code snippet:

```
/* erpcMatrixMultiply function user implementation */
void erpcMatrixMultiply(Matrix matrix1, Matrix matrix2, Matrix result_matrix)
{
  ...
}
int main()
{
  ...
  /* UART transport layer initialization, ERPC_DEMO_UART is the structure of CMSIS UART driver␣
↪operations */
  erpc_transport_t transport;
  transport = erpc_transport_cmsis_uart_init((void *)&ERPC_DEMO_UART);
  ...
  /* MessageBufferFactory initialization */
  erpc_mbf_t message_buffer_factory;
  message_buffer_factory = erpc_mbf_dynamic_init();
  ...
  /* eRPC server side initialization */
  erpc_server_t server;
  server = erpc_server_init(transport, message_buffer_factory);
  ...
  /* Adding the service to the server */
  erpc_service_t service = create_MatrixMultiplyService_service();
  erpc_add_service_to_server(server, service);
  ...
  while (1)
  {
    /* Process eRPC requests */
    erpc_status_t status = erpc_server_poll(server)
    /* handle error status */
    if (status != kErpcStatus_Success)
    {
      /* print error description */
      erpc_error_handler(status, 0);
      ...
    }
    ...
  }
}
```

Parent topic: Multiprocessor server application

Multiprocessor client application

The "Matrix multiply" eRPC client project for multiprocessor applications is located in the *<MCUXpressoSDK_install_dir>/boards/<board_name>/multiprocessor_examples/erpc_client_matrix_multiply_<tra* folder.

Most of the multiprocessor application setup is the same as for the multicore application. The multiprocessor server application requires client-related generated files (server shim code),

client infrastructure files, and the client user code. There is no need for client multicore infrastructure files (MCMGR and RPMsg-Lite). The RPMsg-Lite transport layer is replaced either by SPI or UART transports. The following table shows the required transport-related files per each transport type.

|SPI|<eRPC base directory>/erpc_c/setup/erpc_setup_(d)spi_master.cpp

<eRPC base directory>/erpc_c/transports/ erpc_(d)spi_master_transport.hpp

<eRPC base directory>/erpc_c/transports/ erpc_(d)spi_master_transport.cpp

| |UART|<eRPC base directory>/erpc_c/setup/erpc_setup_uart_cmsis.cpp

<eRPC base directory>/erpc_c/transports/erpc_uart_cmsis_transport.hpp

<eRPC base directory>/erpc_c/transports/erpc_uart_cmsis_transport.cpp

|

Client user code

The client's user code is stored in the main_client.c file, located in the *<MCUXpressoSDK_install_dir>/boards/<board_name>/multiprocessor_examples/ erpc_client_matrix_multiply_<transport_layer>/* folder.

The eRPC-relevant code with UART as a transport is captured in the following code snippet:

```
...
extern bool g_erpc_error_occurred;
...
/* Declare matrix arrays */
Matrix matrix1 = {0}, matrix2 = {0}, result_matrix = {0};
...
/* UART transport layer initialization, ERPC_DEMO_UART is the structure of CMSIS UART driver␣
↪operations */
erpc_transport_t transport;
transport = erpc_transport_cmsis_uart_init((void *)&ERPC_DEMO_UART);
...
/* MessageBufferFactory initialization */
erpc_mbf_t message_buffer_factory;
message_buffer_factory = erpc_mbf_dynamic_init();
...
/* eRPC client side initialization */
erpc_client_t client;
client = erpc_client_init(transport,message_buffer_factory);
...
/* Set default error handler */
erpc_client_set_error_handler(client, erpc_error_handler);
...
while (1)
{
  /* Invoke the erpcMatrixMultiply function */
  erpcMatrixMultiply(matrix1, matrix2, result_matrix);
  ...
  /* Check if some error occured in eRPC */
  if (g_erpc_error_occurred)
  {
    /* Exit program loop */
    break;
  }
  ...
}
```

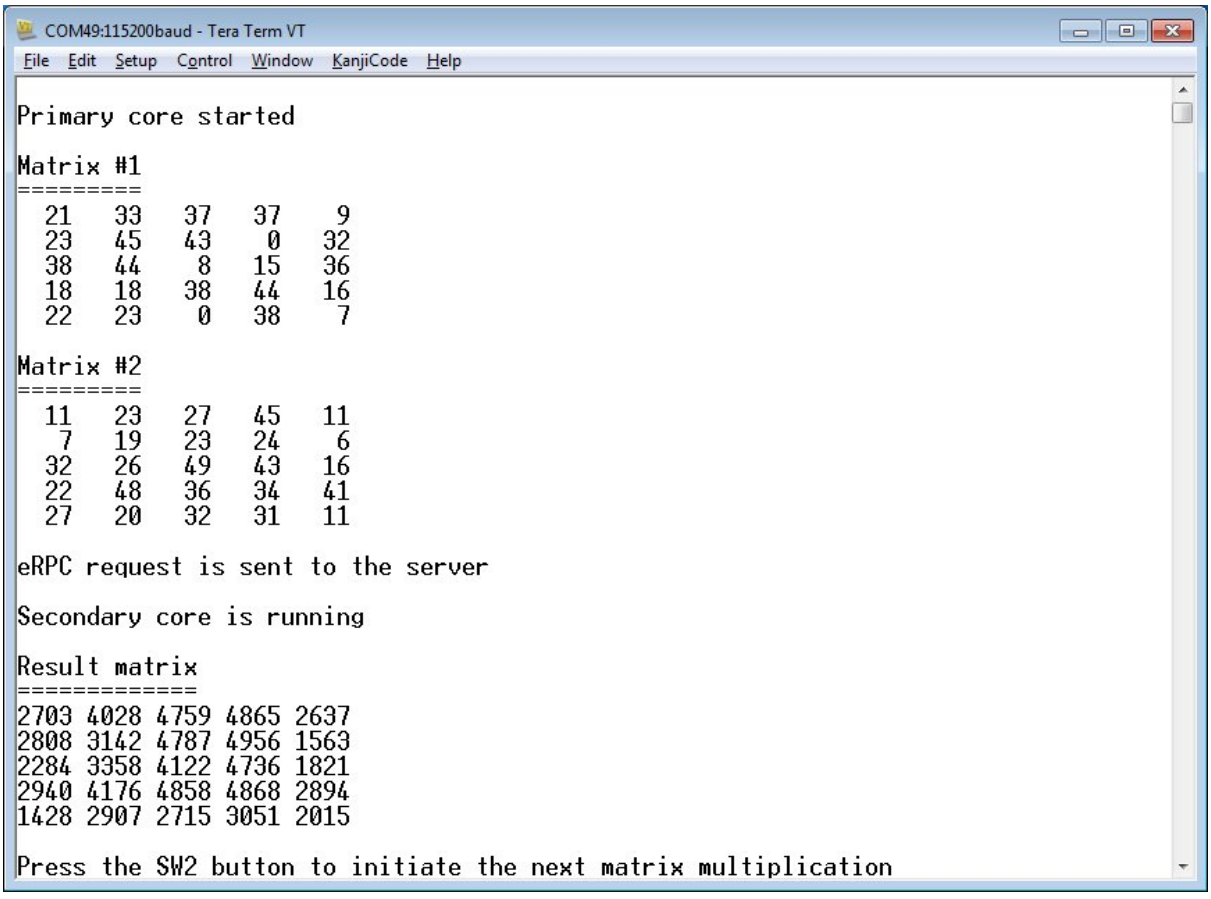
Parent topic: Multiprocessor client application

Parent topic: Multiprocessor server application

Parent topic:*Create an eRPC application*

Running the eRPC application

Follow the instructions in *Getting Started with MCUXpresso SDK* (document MCUXSDKGSUG) (located in the *<MCUXpressoSDK_install_dir>/docs* folder), to load both the primary and the secondary core images into the on-chip memory, and then effectively debug the dual-core application. After the application is running, the serial console should look like:



For multiprocessor applications that are running between PC and the target evaluation board or between two boards, follow the instructions in the accompanied example readme files that provide details about the proper board setup and the PC side setup (Python).

Parent topic:*Create an eRPC application*

Parent topic:*eRPC example*

Other uses for an eRPC implementation

The eRPC implementation is generic, and its use is not limited to just embedded applications. When creating an eRPC application outside the embedded world, the same principles apply. For example, this manual can be used to create an eRPC application for a PC running the Linux operating system. Based on the used type of transport medium, existing transport layers can be used, or new transport layers can be implemented.

For more information and erpc updates see the github.com/EmbeddedRPC.

Note about the source code in the document

Example code shown in this document has the following copyright and BSD-3-Clause license:

Copyright 2024 NXP Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Changelog eRPC

All notable changes to this project will be documented in this file.

The format is based on Keep a Changelog, and this project adheres to Semantic Versioning.

Unreleased

Added

- Added Zephyr IPC Service transport layer support.
- Zephyr examples added support for RT600 and RT700.

Updated

- eRPC Zephyr module port updated for Zephyr version 4.4

Fixed

- Python code of the eRPC infrastructure was updated to match the proper python code style, add type annotations and improve readability.
- eRPC: Several MISRA violations addressed.

1.14.0

Added

- Added Cmake/Kconfig support.
- Made java code jdk11 compliant, GitHub PR #432.
- Added imxrt1186 support into mu transport layer.
- erpcgen: Added assert for listType before usage, GitHub PR #406.

Fixed

- eRPC: Sources reformatted.
- erpc: Fixed typo in semaphore get (mutex -> semaphore), and write it can fail in case of timeout, GitHub PR #446.
- erpc: Free the arbitrated client token from client manager, GitHub PR #444.
- erpc: Fixed Makefile, install the erpc_simple_server header, GitHub PR #447.
- erpc_python: Fixed possible AttributeError and OSError on calling TCPTransport.close(), GitHub PR #438.
- Examples and tests consolidated.

1.13.0

Added

- erpc: Add BSD-3 license to endianness agnostic files, GitHub PR #417.
- eRPC: Add new Zephyr-related transports (zephyr_uart, zephyr_mbox).
- eRPC: Add new Zephyr-related examples.

Fixed

- eRPC,erpcgen: Fixing/improving markdown files, GitHub PR #395.
- eRPC: Fix Python client TCPTransports not being able to close, GitHub PR #390.
- eRPC,erpcgen: Align switch brackets, GitHub PR #396.
- erpc: Fix zephyr uart transport, GitHub PR #410.
- erpc: UART ZEPHYR Transport stop to work after a few transactions when using USB-CDC resolved, GitHub PR #420.

Removed

- eRPC,erpcgen: Remove cstbool library, GitHub PR #403.

1.12.0

Added

- eRPC: Add dynamic/static option for transport init, GitHub PR #361.
- eRPC,erpcgen: Winsock2 support, GitHub PR #365.
- eRPC,erpcgen: Feature/support multiple clients, GitHub PR #271.
- eRPC,erpcgen: Feature/buffer head - Framed transport header data stored in MessageBuffer, GitHub PR #378.
- eRPC,erpcgen: Add experimental Java support.

Fixed

- eRPC: Fix receive error value for spidev, GitHub PR #363.
- eRPC: UartTransport::init adaptation to changed driver.
- eRPC: Fix typo in assert, GitHub PR #371.
- eRPC,erpcgen: Move enums to enum classes, GitHub PR #379.
- eRPC: Fixed rpmsg tty transport to work with serial transport, GitHub PR #373.

1.11.0

Fixed

- eRPC: Makefiles update, GitHub PR #301.
- eRPC: Resolving warnings in Python, GitHub PR #325.
- eRPC: Python3.8 is not ready for usage of typing.Any type, GitHub PR #325.
- eRPC: Improved codec function to use reference instead of address, GitHub PR #324.
- eRPC: Fix NULL check for pending client creation, GitHub PR #341.
- eRPC: Replace sprintf with snprintf, GitHub PR #343.
- eRPC: Use MU_SendMsg blocking call in MU transport.
- eRPC: New LPSPI and LPI2C transport layers.
- eRPC: Freeing static objects, GitHub PR #353.
- eRPC: Fixed casting in deinit functions, GitHub PR #354.
- eRPC: Align LIBUSBSIO.GetNumPorts API use with libusbsio python module v. 2.1.11.
- erpcgen: Renamed temp variable to more generic one, GitHub PR #321.
- erpcgen: Add check that string read is not more than max length, GitHub PR #328.
- erpcgen: Move to g++ in pytest, GitHub PR #335.
- erpcgen: Use build=release for make, GitHub PR #334.
- erpcgen: Removed boost dependency, GitHub PR #346.
- erpcgen: Mingw support, GitHub PR #344.
- erpcgen: VS build update, GitHub PR #347.
- erpcgen: Modified name for common types macro scope, GitHub PR #337.
- erpcgen: Fixed memcpy for template, GitHub PR #352.
- eRPC,erpcgen: Change default build target to release + adding artefacts, GitHub PR #334.
- eRPC,erpcgen: Remove redundant includes, GitHub PR #338.
- eRPC,erpcgen: Many minor code improvements, GitHub PR #323.

1.10.0

Fixed

- eRPC: MU transport layer switched to blocking MU_SendMsg() API use.

1.10.0

Added

- eRPC: Add TCP_NODELAY option to python, GitHub PR #298.

Fixed

- eRPC: MUTransport adaptation to new supported SoCs.
- eRPC: Simplifying CI with installing dependencies using shell script, GitHub PR #267.
- eRPC: Using event for waiting for sock connection in TCP python server, formatting python code, C specific includes, GitHub PR #269.

- eRPC: Endianness agnostic update, GitHub PR #276.
- eRPC: Assertion added for functions which are returning status on freeing memory, GitHub PR #277.
- eRPC: Fixed closing arbitrator server in unit tests, GitHub PR #293.
- eRPC: Makefile updated to reflect the correct header names, GitHub PR #295.
- eRPC: Compare value length to used length() in reading data from message buffer, GitHub PR #297.
- eRPC: Replace EXPECT_TRUE with EXPECT_EQ in unit tests, GitHub PR #318.
- eRPC: Adapt rpmsg_lite based transports to changed rpmsg_lite_wait_for_link_up() API parameters.
- eRPC, erpcgen: Better distuingish which file can and cannot by linked by C linker, GitHub PR #266.
- eRPC, erpcgen: Stop checking if pointer is NULL before sending it to the erpc_free function, GitHub PR #275.
- eRPC, erpcgen: Changed api to count with more interfaces, GitHub PR #304.
- erpcgen: Check before reading from heap the buffer boundaries, GitHub PR #287.
- erpcgen: Several fixes for tests and CI, GitHub PR #289.
- erpcgen: Refactoring erpcgen code, GitHub PR #302.
- erpcgen: Fixed assigning const value to enum, GitHub PR #309.
- erpcgen: Enable runTesttest_enumErrorCode_allDirection, serialize enums as int32 instead of uint32.

1.9.1

Fixed

- eRPC: Construct the USB CDC transport, rather than a client, GitHub PR #220.
- eRPC: Fix premature import of package, causing failure when attempting installation of Python library in a clean environment, GitHub PR #38, #226.
- eRPC: Improve python detection in make, GitHub PR #225.
- eRPC: Fix several warnings with deprecated call in pytest, GitHub PR #227.
- eRPC: Fix freeing union members when only default need be freed, GitHub PR #228.
- eRPC: Fix making test under Linux, GitHub PR #229.
- eRPC: Assert costumizing, GitHub PR #148.
- eRPC: Fix corrupt clientList bug in TransportArbitrator, GitHub PR #199.
- eRPC: Fix build issue when invoking g++ with -Wno-error=free-nonheap-object, GitHub PR #233.
- eRPC: Fix inout cases, GitHub PR #237.
- eRPC: Remove ERPC_PRE_POST_ACTION dependency on return type, GitHub PR #238.
- eRPC: Adding NULL to ptr when codec function failed, fixing memcpy when fail is present during deserialization, GitHub PR #253.
- eRPC: MessageBuffer usage improvement, GitHub PR #258.
- eRPC: Get rid for serial and enum34 dependency (enum34 is in python3 since 3.4 (from 2014)), GitHub PR #247.

- eRPC: Several MISRA violations addressed.
- eRPC: Fix timeout for Freertos semaphore, GitHub PR #251.
- eRPC: Use of rpmsg_lite_wait_for_link_up() in rpmsg_lite based transports, GitHub PR #223.
- eRPC: Fix codec nullptr dereferencing, GitHub PR #264.
- erpcgen: Fix two syntax errors in erpcgen Python output related to non-encapsulated unions, improved test for union, GitHub PR #206, #224.
- erpcgen: Fix serialization of list/binary types, GitHub PR #240.
- erpcgen: Fix empty list parsing, GitHub PR #72.
- erpcgen: Fix templates for malloc errors, GitHub PR #110.
- erpcgen: Get rid of encapsulated union declarations in global scale, improve enum usage in unions, GitHub PR #249, #250.
- erpcgen: Fix compile error:UniqueIdChecker.cpp:156:104:'sort' was not declared, GitHub PR #265.

1.9.0

Added

- eRPC: Allow used LIBUSBSIO device index being specified from the Python command line argument.

Fixed

- eRPC: Improving template usage, GitHub PR #153.
- eRPC: run_clang_format.py cleanup, GitHub PR #177.
- eRPC: Build TCP transport setup code into liberpc, GitHub PR #179.
- eRPC: Fix multiple definitions of g_client error, GitHub PR #180.
- eRPC: Fix memset past end of buffer in erpc_setup_mbf_static.cpp, GitHub PR #184.
- eRPC: Fix deprecated error with newer pytest version, GitHub PR #203.
- eRPC, erpcgen: Static allocation support and usage of rpmsg static FreeRTOSs related APi, GitHub PR #168, #169.
- erpcgen: Remove redundant module imports in erpcgen, GitHub PR #196.

1.8.1

Added

- eRPC: New i2c_slave_transport trasnport introduced.

Fixed

- eRPC: Fix misra erpc c, GitHub PR #158.
- eRPC: Allow conditional compilation of message_loggers and pre_post_action.
- eRPC: (D)SPI slave transports updated to avoid busy loops in rtos environments.
- erpcgen: Re-implement EnumMember::hasValue(), GitHub PR #159.
- erpcgen: Fixing several misra issues in shim code, erpcgen and unit tests updated, GitHub PR #156.

- erpcgen: Fix bison file, GitHub PR #156.

1.8.0

Added

- eRPC: Support win32 thread, GitHub PR #108.
- eRPC: Add mbed support for malloc() and free(), GitHub PR #92.
- eRPC: Introduced pre and post callbacks for eRPC call, GitHub PR #131.
- eRPC: Introduced new USB CDC transport.
- eRPC: Introduced new Linux spidev-based transport.
- eRPC: Added formatting extension for VSC, GitHub PR #134.
- erpcgen: Introduce ustring type for unsigned char and force cast to char*, GitHub PR #125.

Fixed

- eRPC: Update makefile.
- eRPC: Fixed warnings and error with using MessageLoggers, GitHub PR #127.
- eRPC: Extend error msg for python server service handle function, GitHub PR #132.
- eRPC: Update CMSIS UART transport layer to avoid busy loops in rtos environments, introduce semaphores.
- eRPC: SPI transport update to allow usage without handshaking GPIO.
- eRPC: Native _WIN32 erpc serial transport and threading.
- eRPC: Arbitrator deadlock fix, TCP transport updated, TCP setup functions introduced, GitHub PR #121.
- eRPC: Update of matrix_multiply.py example: Add –serial and –baud argument, GitHub PR #137.
- eRPC: Update of .clang-format, GitHub PR #140.
- eRPC: Update of erpc_framed_transport.cpp: return error if received message has zero length, GitHub PR #141.
- eRPC, erpcgen: Fixed error messages produced by -Wall -Wextra -Wshadow -pedantic-errors compiler flags, GitHub PR #136, #139.
- eRPC, erpcgen: Core re-formatted using Clang version 10.
- erpcgen: Enable deallocation in server shim code when callback/function pointer used as out parameter in IDL.
- erpcgen: Removed '$' character from generated symbol name in '_$union' suffix, GitHub PR #103.
- erpcgen: Resolved mismatch between C++ and Python for callback index type, GitHub PR #111.
- erpcgen: Python generator improvements, GitHub PR #100, #118.
- erpcgen: Fixed error messages produced by -Wall -Wextra -Wshadow -pedantic-errors compiler flags, GitHub PR #136.

1.7.4

Added

- eRPC: Support MU transport unit testing.
- eRPC: Adding mbed os support.

Fixed

- eRPC: Unit test code updated to handle service add and remove operations.
- eRPC: Several MISRA issues in rpmsg-based transports addressed.
- eRPC: Fixed Linux/TCP acceptance tests in release target.
- eRPC: Minor documentation updates, code formatting.
- erpcgen: Whitespace removed from C common header template.

1.7.3

Fixed

- eRPC: Improved the test_callbacks logic to be more understandable and to allow requested callback execution on the server side.
- eRPC: TransportArbitrator::prepareClientReceive modified to avoid incorrect return value type.
- eRPC: The ClientManager and the ArbitratedClientManager updated to avoid performing client requests when the previous serialization phase fails.
- erpcgen: Generate the shim code for destroy of statically allocated services.

1.7.2

Added

- eRPC: Add missing doxygen comments for transports.

Fixed

- eRPC: Improved support of const types.
- eRPC: Fixed Mac build.
- eRPC: Fixed serializing python list.
- eRPC: Documentation update.

1.7.1

Fixed

- eRPC: Fixed semaphore in static message buffer factory.
- erpcgen: Fixed MU received error flag.
- erpcgen: Fixed tcp transport.

1.7.0

Added

- eRPC: List names are based on their types. Names are more deterministic.
- eRPC: Service objects are as a default created as global static objects.

- eRPC: Added missing doxygen comments.
- eRPC: Added support for 64bit numbers.
- eRPC: Added support of program language specific annotations.

Fixed

- eRPC: Improved code size of generated code.
- eRPC: Generating crc value is optional.
- eRPC: Fixed CMSIS Uart driver. Removed dependency on KSDK.
- eRPC: Forbid users use reserved words.
- eRPC: Removed outByref for function parameters.
- eRPC: Optimized code style of callback functions.

1.6.0

Added

- eRPC: Added @nullable support for scalar types.

Fixed

- eRPC: Improved code size of generated code.
- eRPC: Improved eRPC nested calls.
- eRPC: Improved eRPC list length variable serialization.

1.5.0

Added

- eRPC: Added support for unions type non-wrapped by structure.
- eRPC: Added callbacks support.
- eRPC: Added support @external annotation for functions.
- eRPC: Added support @name annotation.
- eRPC: Added Messaging Unit transport layer.
- eRPC: Added RPMSG Lite RTOS TTY transport layer.
- eRPC: Added version verification and IDL version verification between eRPC code and eRPC generated shim code.
- eRPC: Added support of shared memory pointer.
- eRPC: Added annotation to forbid generating const keyword for function parameters.
- eRPC: Added python matrix multiply example.
- eRPC: Added nested call support.
- eRPC: Added struct member “byref” option support.
- eRPC: Added support of forward declarations of structures
- eRPC: Added Python RPMsg Multiendpoint kernel module support
- eRPC: Added eRPC sniffer tool

1.4.0

Added

- eRPC: New RPMsg-Lite Zero Copy (RPMsgZC) transport layer.

Fixed

- eRPC: win_flex_bison.zip for windows updated.
- eRPC: Use one codec (instead of inCodec outCodec).

[1.3.0]

Added

- eRPC: New annotation types introduced (@length, @max_length, ...).
- eRPC: Support for running both erpc client and erpc server on one side.
- eRPC: New transport layers for (LP)UART, (D)SPI.
- eRPC: Error handling support.

[1.2.0]

Added

- eRPC source directory organization changed.
- Many eRPC improvements.

[1.1.0]

Added

- Multicore SDK 1.1.0 ported to KSDK 2.0.0.

[1.0.0]

Added

- Initial Release

3.3 Wireless

3.3.1 NXP Wireless Framework and Stacks

NXP Bluetooth LE Host and Sample Applications

Changelog All notable changes to NXP Bluetooth LE Host will be documented in this file.

NXP Bluetooth LE Host Stack is certified **Bluetooth 6.3**

[1.10.23] - mcux v2026-09-00

Note: Version 1.10.23 is the first NXP Bluetooth LE Host release officially certified according to the Bluetooth® Core Specification Version 6.3

Added

- Bluetooth Sample Applications support for the KW43-LOC board
- Support in Wireless Uart for NBU core-dump packet reception (gEnableCoredumpPackets)

Improved

- Optimised stack sizes for the PSA module (frdmmcxw70/frdmkw43)
- Added MinimalHeapSize_c to all localization applications, scaled for RADE2 and number of connections
- Optimized memory buffer values in the loc_reader Channel Sounding application
- Reworked setcsconfig and setcsproc shell commands for per-parameter updates in localization applications
- Updated wireless ranging Python host application setup documentation (Python 3.12+ supported)
- Updated CCC Digital Key R3 Application Note connection handover features

Fixed

- Fixed ExtendedFeatures mismatch between GAPInit and HCI LE Read All Remote Features Complete event
- Fixed EATT out-of-bounds read in IsEnhncdChanReconfInProgress()
- Fixed CCC demo CS ranging procedure counter wrapping past 0xFFFF
- Fixed GattDbDynamic_AddCharServiceChanged using Notify instead of Indicate property
- Fixed ShellGap_ConnectFromPawr setting the success status incorrectly
- Miscellaneous Coverity fixes
- Miscellaneous CERT-C fixes
- Miscellaneous MISRA fixes

Changed

- Bluetooth 6.3 Compliance: removed Data Signing (LE Security Mode 2)

Supported Platforms

- KW45, KW47, MCXW71, MCXW72, MCXW23, KW43, MCXW70

[1.10.22] - mcux v2026-09-00-pvw2

Added

- Added gSecEvt_SameConfirmValue_c IDS event to detect Legacy Pairing Confirm Value replay attacks

Improved

- Replaced mode assert with graceful drop and cleanup in localization applications

- RAS/RAP: handle case where MTU size reduction forces use of more segments than predefined gRASMaxNoOfSegments_c
- Advertising-set-related BLE Host storage is now allocated by the application and scales with the configurable gMaxAdvSets_c

Fixed

- Fixed CS measurement not starting after handover in digital_key_car_anchor_cs
- Fixed Insufficient Encryption returned instead of Insufficient Authentication
- Fixed RSSI filter not excluding invalid RSSI values in CCC monitoring
- Fixed missing CCC characteristics in digital_key_car_anchor_cs example
- Fixed mismatch between ExtendedFeatures from GAPInit and HCI LE Read All Remote Features Complete event
- Miscellaneous Sample Applications fixes
- Miscellaneous Coverity fixes
- Miscellaneous MISRA fixes

Supported Platforms

- KW45, KW47, MCXW71, MCXW72, MCXW23, KW43, MCXW70

[1.10.21] - mcux v2026-09-00-pvw1

Added

- RSSI-based adaptive CS procedure interval for localization applications (experimental; disabled by default)
- Added BTCS timer mechanism to ensure CS procedure loop restarts are not blocked by failed L2CAP transfers

Improved

- CS procedure parameters are now updated on connection interval change to avoid CS request rejection
- Aligned wireless_uart preinclude settings across all platforms for consistency
- Updated ble_shell periodic advertising default interval set to 1 second for improved discoverability
- Increased host task stack size across multiple applications
- Extended ble_shell periodic advertising handling to support gPeriodicDeviceScannedV2_c event
- RADE component naming updated in documentation
- Unified CS results complete event handling; real-time transfer moved to AppLocalization_HandleCompleteResults()

Fixed

- Fixed out-of-bounds access in gap.c
- HCI LE Read All Remote Features command is now issued if supported by the controller
- Power Control APIs now return gBleFeatureNotSupported_c if the controller does not indicate support

- Fixed array index overflow in application ranging_client.c
- Fixed local RSSI parsing: correct byte is now collected before advancing parse pointer
- Fixed bounds check for gaAntPermNAp in processMode2Data and processMode3Data
- Fixed localization index overflow; added sanity check
- Corrected bounds check for deviceId
- Fixed wireless_uart mAdvPending flag to prevent multiple advertising starts during multi-peer disconnect
- Fixed missing RSSI variables causing compilation issues in app_localization_utils.c
- TPMS sensor Signing Key and sequence number now saved in NVM to prevent MAC check failures
- CS IPT mode now configured with only 1 antenna path to fix Samsung phone CS request rejection
- FSCI Extended NBU: Added LL_API_NotifyWakeUp call in idle_task to detect NBU stuck condition; added leExtendedFeatures to NCP GAP init complete event
- Fixed systick drift when PWR_DisallowDeviceToSleep() is active
- CS: Added check for valid numAntennaPaths in processCsResultsEvent
- Miscellaneous MISRA fixes
- Miscellaneous Coverity fixes

Changed

- Reverted incorrect connection handle handling for non-connection enhanced notification events

Supported Platforms

- KW45, KW47, MCXW71, MCXW72, MCXW23, KW43, MCXW70

[1.10.20] - mcux v2026-06-00

Added

- Tire Pressure Monitoring System (TPMS) sample demo applications (KW4x)
- Tire Pressure Monitoring System (TPMS) documentation added to DAUG
- Localization accelerator LCE support for KW43/MCXW70
- Added LE Power Control support in BLE Shell application
- LE Read All Remote Features (LLEFS) support in HOST-GAP
- LE Channel Assessment (CHAS) Config HCI command support in HOST-GAP
- Sensor support enabled on KW43/MCXW70

Improved

- CS procedure auto-loop shell command; improved RAS data drop mechanism
- Allow ‘setnumprocs’ 0 for infinite CS procedure repeats
- Synchronize code between loc_reader and ncp_loc_reader application files
- Subevent interval added to RADE algoritm API

- Updated connection handover setup info for FRDM-KW43 in CS application documentations
- Documentation updates

Fixed

- L2CAP data fragmentation fix causing inccorect fragmentation over the air
- Fix for no valid subevents in localization
- Allow sub_mode_type 255 in CS config params
- Incorrect connection handle (0 instead of 0xFFFF) for non-connection enhanced notification events
- Channel Sounding subevent abort (No CS_SYNC mode0) after rebond
- Set 'preferredPeerAntenna' based on peer capabilities
- Miscellaneous minor application bug fixes

Changed

- Reduced cyclomatic complexity (CCM) in multiple localization functions
- Decoupled buttons from LEDs in Bluetooth applications

Supported Platforms

- KW45, KW47, MCXW71, MCXW72, MCXW23, KW43, MCXW70

[1.10.19] - mcux v2026-06-00-pvw2

Added

- Support for LE Read All Remote Features after connection is established
- Experimental Inline PCT Transfer support for localization sample applications
- Support for configurable limits for concurrent Channel Sounding connections/procedures

Improved

- Optimized memory allocation in isp_mciq_ranging_compute for localization applications
- Updated RAS loc_reader application for multiple CS connections
- Updated handover documentation with broadcast time synchronization details
- Documentation updates

Fixed

- Fixed double free issue in L2ca_SendAclDataWithSegmentation
- Fixed consecutive CS procedures sustainability issue
- Miscellaneous minor application bug fixes

Changed

- CS procedure state management: do not allow new procedure to start before previous one completes

Supported Platforms

- KW45, KW47, MCXW71, MCXW72, MCXW23, KW43

[1.10.18] - mcux v2026-06-00-pvw1

Added

- Support for RTT 0 m calibration in applications.
- Support for CS Enhancements at Host level:
  - Inline PCT Transfer
  - RTT 2M PHY
- New connection event 'gConnEvtRemoteFeaturesRead_c', providing the peer feature bitmask read via the Read Remote Features procedure
- Support for 5 advertising sets in the Bluetooth LE Host
- Method to specify GATT handles on the client, avoiding repeated service discovery
- Slope calibration algorithm for CS (experimental, disabled by default). Computes distance and quality indicators per antenna path
- Handover broadcast time synchronization, allowing one connected anchor to synchronize multiple target anchors simultaneously
- Added common configuration header 'app_localization_config.h', overridable by the user

Improved

- Updated handling of 'Procedure_Results_Start' to correctly process multiple subevents in a single message
- Improved CS temperature polling mechanism
- BTCS Server now explicitly notifies the PSM channel via 'AppLocalization_SetPsmChannelId' upon handover connection
- RAS/BTCS clients can disable algorithm execution via 'gRunAlgo_d'
- Updated and cleaned 'app_preinclude.h' for sample applications
- Documentation and configuration updates for LCE enable/disable
- Documentation updates
- Miscellaneous minor application bug fixes

Fixed

- If the LE Set Periodic Adv Subevent Data command finishes with an error, use 'gInternalError_c' with 'gLeSetPeriodicAdvSubeventData_c' as the source and the command complete status as the error code
- Correct handling of AddrType values in MonAdvReport generated from XML

Changed

- Removed the Bluetooth LE Host library from the wireless_uart_host project
- Disabled use of Random Static Address for all applications except: digital_key_car_anchor and digital_key_car_anchor_cs
- Set maximum CS procedure duration to: (procedure interval x connection interval x 2 - 1) slots

Supported Platforms

- KW45, KW47, MCXW71, MCXW72, MCXW23

[1.10.17] - mcux v2026-03-00

Added

- Test mode functionality for the Intrusion Detection System
- GAP API for HCI LE_Set_Local_Slave_Latency_Enable vendor command
- gRasUseNotifOrInd_c to allow configuration of Indication or Notification on RAS
- cs_sync_phy argument to setcsconfig shell command
- snr_control_init and snr_control_refl arguments to setcsproc shell command
- Detailed description for setcsconfig and setcsproc command arguments to the CCC w/ CS and Localization applications notes
- Mode0 step information parsing for both RAS and BTCS, made available for the export module

Improved

- ATT Write Commands can now be sent at any time, as per specification
- Enhanced CCC and RAS Channel Sounding by incorporating real-time local and remote data decompression
- Localization applications now record the PHY used for establishing the connection and incorporate it into the CS procedure parameters configuration
- Localization applications keep the subevtStopIdx for the remote data and provide it to the algorithm
- Localization applications now pass remote mode map to algorithm
- Eliminated the dependency on hci_interface.h in localization files
- Documentation updates
- Miscellaneous minor application bug fixes

Fixed

- Wrong data field type in GAP2GenericEventVendorSpecificDebugFatalError
- Localization application now processes CS results only when local data is available
- Digital key car anchor application issue related to setting the GAP role prior to any use
- Error in the definition of a CCC PSM VDBT Version GATT characteristic
- Issue with PACKET_NADM byte not being packed correctly into the TOF buffer for mode-3 steps
- MISRA fixes

Changed

- Modified linker section names, comments, and RAM layout in the connectivity.icf and connectivity_ble_loc.ld files for every KW47 and MCXW72 board
- Set the default CS submode and default rtt_type to 3
- Set the default radio core speed for the Channel Sounding application to 48 MHz

- Localization applications configured to use the extended heap as the final segment of DATA1
- Deleted the gcGapMaximumActiveConnections_c definition from applications
- Eliminated unused fields related to CS Test Mode
- Deleted initializations of range settings that were subsequently overwritten

Supported Platforms

- KW45, KW47, MCXW71, MCXW72, MCXW23

[1.10.16] - mcux v2026-03-00-pvw2

Added

- Implemented Bluetooth LE Host Transient Application Key feature (experimental). in ble_shell, loc_reader and loc_user_device
- Added Bluetooth LE Host Transient Application Key in ble_shell, loc_reader and loc_user_device
- Added Channel Sounding phase calibration, with 0.5m compensation for all boards
- Added Vendor Specific Command to enable Link Layer experimental features
- Added gAppCsTimeInfo_d define in app_preinclude.h for digital_key_car_anchor_cs on KW47EVK
- Added Channel Sounding **phase calibration** logic and updates in RAS CS Procedure Timing Measurement

Improved

- Updated RADE algorithm source files license description with NXP proprietary license
- Improved data logging for digital key application
- Updated CCC enumeration values to align with CCC v4.0.0 specification
- Updated Intrusion Detection System documentation with event details
- Documentation miscellaneous updates

Fixed

- Fixed ranging data body parser on the client side — aborted steps are now counted correctly to prevent subevent misalignment
- Fixed handling of CS Procedure Disable at Host and application level
- Fixed app_localization timer blocking when sensors were disabled
- Fixed setting of gCsTimeInfo.transferStart for BTCS Client L2CAP transfer
- Removed unused gAppRealTimeDataTransfer_c define from digital_key_car_anchor_cs
- Removed pin_mux files from core1 application projects
- Fixed issues in data transfer and event updates related to timing measurement

Changed

- Updated (SHELL_MAX_ARGS = 20) in app_preinclude.h for localization applications.

Supported Platforms

- KW45, KW47, MCXW71, MCXW72, MCXW23

[1.10.15] – mcux v2026-03-00-pvw1

Added

- Added an API to update local synchronization parameters after PAwR is already established on the scanner side
- Added Connection Subrating feature (experimental) in the Bluetooth LE Host
- Added Connection Subrating feature in ble_shell
- CCC v4.1.0: renamed DK_VERSION to VDBT_VERSION as per updated specification

Improved

- Improved localization applications timer mechanism to reduce the number of wakeups
- Updated localization timeout values to be overridable by the application
- Retain the subevent_done_status local and remote values for all subevents and pass them to the algorithm
- Documentation miscellaneous updates

Fixed

- Fixed L2CAP credit-based channel disconnection where the channel's timer ID would be set to 0 instead of gTmrInvalidTimerID_c
- Fixed Channel Sounding applications issue: clear the Indication Pending flag for a peer device on disconnect
- ble_shell: Set maximum arguments in command (SHELL_MAX_ARGS = 20) in app_preinclude.h

Supported Platforms

- KW45, KW47, MCXW71, MCXW72, MCXW23

[1.10.14] - mcux v2025-12-00

Added

- KW47/MCXW72 CCC and IOT Channel Sounding Localization applications on GitHub repository
- Monitoring Advertisers support in fsci_black_box and BleShell applications
- Local average and remote average RSSI values to the CS measurement report
- The "-Os" optimization flag to the ARMGCC release configuration for NCP applications

Improved

- Enabled low power support in loc_reader_host application

Fixed

- loc_reader_host application event set issue
- Missing handler for Version2 of the Set RPA Timeout command

Supported Platforms

- KW45, KW47, MCXW71, MCXW72, MCXW23

[1.10.13] - mcux v2025-12-00-pvw2

Added

- Threshold for the invalid number of Anchor Monitor events received by the target anchor
- **Experimental Monitoring Advertisers** feature in Bluetooth LE Host
- **Experimental Randomized RPA** feature in Bluetooth LE Host
- Application defines for default connection and default advertising tx power

Improved

- Miscellaneous applications updates
- Central applications now wait for status of Encrypt procedure in case of bonded device
- Logging data on localization applications
- NBU Low Power Mode enabled by default for Localization Applications
- PCT rotation calibration added to localization apps
- Configured CS Reflector to start the CS procedure with the tdm command and updated the documentation
- Populated the optionalSubfeaturesSupported field correctly
- Prevented CORE 0 from entering deep sleep while LCE is computing by setting the low power mode constraint to PWR_WFI during LCE computation and releasing it afterwards
- Implemented in CCC_CS, Channel Sounding data transfer from the anchor to the device
- Documentation miscellaneous updates
- Updated all kw47 and kw45 armgcc ld linker files to take gUseInternalStorageLink_d flag value into consideration

Fixed

- Memory issue when setting scan response data would return an error status from the LL
- Set advertises with the public address, overwritten by a previously used random address on receiving the Advertising Set Terminated event

Changed

- Updated memory configuration: replaced the extended heap area in the available SMU2 memory with a 24KB array in the data1
- Removed redundant cached remote capabilities write on reflectors, as the initiator will always trigger a capabilities exchange and trying to write cached capabilities afterwards results in an HCI error

Supported Platforms

- KW45, KW47, MCXW71, MCXW72, MCXW23

[1.10.12] - mcux v2025-12-00-pvw1

Added

- **Gap_LoadCustomBondedDeviceInformation API** to retrieve custom peer information using NVM index

Improved

- Miscellaneous application updates

Fixed

- **Updated privacy timeout** mechanism affected by LE Set Extended Advertising Enable Command

Changed

- Merged **Gap_SetPeriodicAdvParameters** and Gap_SetPeriodicAdvParametersV2 into Gap_SetPeriodicAdvParameters

Supported Platforms

- KW45, KW47, MCXW71, MCXW72, MCXW23

[1.10.11] - mcux v2025-09-00

Added

- CCC sample applications updated to **CCC Digital Key v4.0.0** Specification
- **RAS/RAP PTS** 8.7.4 test support added in Localization Sample applications
- Support for CS start procedure while the previous procedure is not completed; old procedure replaced with the new one
- Support for arm gcc for ncp_loc_reader__cm33_core1

Improved

- Localization Sample Applications Ram partition
- **RAS/RAP** profile and service
- Various sample applications have been updated

Fixed

- Privacy setting issue on ncp_loc_reader
- Extended NBU FSCI message handling issue
- Ble_shell updated to set the Random Static Address properly
- Always set the Advertising Legacy Set handle if the legacy API was used
- fsci_bridge and w_uart_host memory leak
- PAWR parameters in PeriodicSyncTransferReceived are now parsed correctly
- Ensure an RPA/NRPA is properly set from the application to enable a central using Controller Privacy to connect to unbonded peripherals
- CS algorithm buffer overwrite issue during Connection Handover application
- Various sample applications bug fixes applied
- Pass correct Codded PHY (S2) to Channel Sounding Set Procedure Parameters

Changed

- Merged Gap_SetExtAdvertisingParameters and Gap_SetExtAdvertisingParametersV2 into **Gap_SetExtAdvertisingParameters**
- ce_status_buffer type changed to int32_t
- BLE_Shell prints Random Static address as identity address instead of the Public Device Address

Supported Platforms

- KW45, KW47, MCXW71, MCXW72, MCXW23

[1.10.10] - mcux v2025-09-00-pvw2

Improved

- **CS Event Handling**: CS (Channel Sounding) events are now sent to the application task for processing, rather than being handled directly in the Host task
- Various sample applications have been updated

Fixed

- Bluetooth Advertising Sets: Now supports **4 advertising** sets in the Bluetooth host libraries
- Various sample applications bug fixes applied

Changed

- Bluetooth Address Type: The default address type has been changed from **Public to Random Static**

Supported Platforms

- KW45, KW47, MCXW71, MCXW72, MCXW23

[1.10.9] - mcux v2025-09-00-pvw1

Added

- **IDS event** trigger when incoming ATT packets exceed agreed MTU
- **IDS event** trigger when Unexpected SMP Messages received in idle states (before pairing starts)
- Average RSSI reporting in Anchor Monitor event
- Support for gAppDeferAlgoRun_d in btcs_client.c
- **Multi-connection monitoring** in Handover/Monitor Mode

Improved

- Updated **NBU channel sounding** applications to support **64MHz** clock speed
- Cleanup of commComplete structures that only contain status from hci_interface.h

Fixed

- CCC Application handover monitoring RSSI issue
- Intrusion Detection System not reporting event type

- Extended **NBU armgcc projects** stability
- Advertising Extended Applications when Gap_PeriodicAdvCreateSync fails

Changed

- Updated digital_key_car_anchor applications to configure coding scheme via Host API
- Enhanced RAS handling of ACK Ranging Data in invalid conditions

Supported Platforms

- KW45, KW47, MCXW71, MCXW72, MCXW23

[1.10.8] - mcux v2025-06-00

Added

- **Gap_SetBondedDeviceName()** to set device name using NVM index
- **RAS** queue for GATT indications sent
- **gHciStatusBase_c** to **csError** status
- Option to use statically allocated **memory** for dynamic **GATT database** (prevents heap fragmentation)
- Checks for **controller** supported features and setting **PAST bits** accordingly
- **Anchor** support to **export** device data via RAS using gAppHciDataLogExport_d = 2
- **Anchor** support to **export** device local HCI data using gAppHciDataLogExport_d = 1
- **Shell commands** to list peer devices and trigger connection handover
- Define for **enabling** optional **CCC** LE Coded **PHY** advertising
- **cs_sync_phy** parameter to mDefaultRangeSettings (**renamed** from outdated RTTPhy)

Improved

- **Stack Host** now saves the most recently set **random address** after successful controller response
- Miscellaneous **minor** application **updates**

Fixed

- Compilation issue in **loc_reader app** with real-time RAS transfer
- **CCC** application **handover state machine** race condition
- **CCC** resets **gCurrentAdvHandle** upon connection
- **RAS** uses correct bit for data overwrite preference

Changed

- Updated **Bluetooth LE Host Documentation**.
- **BLE_Shell** Tx timer interval adjusted for **max throughput** on 1M PHY
- **CS_ConfigVendorCommand** updated with **Inline Phase Return** field
- Renamed **tx_pwr_phy** to **phy** and removed obsolete rtt_phy field
- Updated **documentation** to clarify **Controller Privacy** restrictions

Supported Platforms

- KW45, KW47, MCXW71, MCXW72, MCXW23

[1.10.7] - mcux v2025-06-00-pvw2

Added

- Support for **Bluetooth LE Debug Keys**
- Support for **pairing failure** reason 0x10 (Busy)
- Extended NBU **Wireless_uart_host battery service**
- **Channel Sounding RSSI Information** at application level
- Application support for **4-antenna configuration**

Improved

- **Clean** include directives
- Allow central-only devices to use **Gap_EncryptAdvertisingData**
- **CSTACK** size optimization for **RADE** using baremetal apps

Fixed

- Extended NBU **memory leak** issues
- Issues when **gAppOtaASyncFlashTransactions_c** is set to 0
- Issue with **CS procedure** affected by the CS data export
- Extended NBU **Wireless_uart_host privacy**

Changed

- Updated **Bluetooth LE Host Documentation**.

Supported Platforms

- KW45, KW47, MCXW71, MCXW72, K32W1

[1.10.6] - mcux v2025-06-00-pvw1

Added

- **Encypted Advertising Data** support in Extendded Advertising Applications.
- Support for **disable UART** for **CS applications** for **low power measurements**.
- Support for **LCE (DSPV) non-blocking API** integration to **RADE**.
- **Intrusion Detection System** as **Experimental**.

Improved

- **L2CAP command length validation** to cover all signaling commands.
- **Extended NBU Wireless_uart_host multiple connections** support.

Supported Platforms

- KW45, KW47, MCXW71, MCXW72, K32W1

[1.10.5] - mcux v2025-03-00

Added

- MCXW72 **Extended NBU** support and **w_uart_host**, **fsci_bridge** and **ncp_fscibb** sample applications.
- **L2CAP support** for Channel Sounding **IQ Sample Transfer** in CCC CS sample applications.
- Bluetooth LE Sample applications for **MCX-W71-EVK** board.

Changed

- Updated **FSCI XML file**.
- Updated **Bluetooth LE Host Documentation**.

Fixed

- Cleared the **mpRemoteCachedCaps** entry when the peer disconnects (CS sample applications).
- Transfer **RAS subscription** data during connection handover (CCC CS sample applications).
- **EAD** - Updated advertising data length check to ensure encrypted data fits inside one AD.
- Updated **digital_key_car_anchor** and **digital_key_car_anchor_cs** applications to manage **Random Static address** from the application layer.

Supported Platforms

- KW45, KW47, MCXW71, MCXW72, K32W1

[1.10.4] - mcux v2025-03-00-pvw2

Added

- PAWR support in **BLE Shell** sample application.
- PAWR support in **adv_ext_peripheral** and **adv_ext_central** sample applications.
- New sample applications for **FRDM-MCXW72**.
- **Gap_SetScanningCallback** API.
- Support for handover connection interval update command.

Changed

- Updated HID Device for **Windows 11** compatibility.
- Updated CCC demos to **Digital Key R4 spec version 1.0.0**.
- Improved RPA resolution at the **Host level**, now performed synchronously.
- Enhanced parsing of the **CS procedure** in Ranging Service.

Fixed

- Corrected parsing of the **PAST command** in FSCI GAP.
- Fixed **scan event reporting** in PAST scenario.
- Added an error case for `Gap_SetChannelMap` in the generic event handler.

Supported Platforms

- KW45, KW47, MCXW71, MCXW72, K32W1

Bluetooth Low Energy Application Developer's Guide Introduction

This document explains how to integrate the NXP Bluetooth Low Energy Host Stack in an application and provides detailed explanation of the most commonly used APIs and code examples.

- *Introduction*: This section outlines the document structure.
- *Prerequisites*: The document sets out the prerequisites.
- *Bluetooth LE Host Stack Initialization and APIs*: This section describes the Bluetooth Low Energy Host Stack initialization. It also presents the APIs categorized according to the layer and by application role.
- *Generic Access Profile (GAP) Layer*: The Generic Access Profile (GAP) layer is divided into two sections according to the GAP role of the device: Central and Peripheral. The basic setup of two such devices is explained with code examples, such as how to prepare the devices for connections, how to connect them together, and pairing and bonding processes.
- *Generic Attribute Profile (GATT) Layer*: This section describes the Generic Attribute Profile (GATT) layer and introduces the APIs required for data transfer between the two connected devices. This section is divided into two subsections according to the GATT role of the device: Client and Server.
- *GATT database application interface*: The document further describes the usage of the GATT database APIs that allow the application to manipulate data stored in the GATT Server database.
- *Creating GATT database*: This section describes a user-friendly method to build a GATT database statically. The method involves the use of a predefined set of macros that the application can include to build the database at application compile time.
- *Creating a Custom Profile*: This section contains instructions on how to build a custom profile.
- *Application Structure*: The section describes the structure of the typical application.
- *Low-Power Management*: This section describes low-power management and how an application can use the low-power modes of the hardware and software.
- *Over the Air Programming (OTAP)*: This section describes the Over The Air Programming (OTAP) capabilities that the Host Stack offers via a dedicated Service/Profile. The section also describes how to use the OTAP capabilities in an application and also contains a detailed description of the SDK components involved in the OTAP process.
- *Creating a Bluetooth LE application when the Host Stack runs on another processor*: This section describes how to build a Bluetooth Low Energy application when the Host Stack is running on a separate processor.
- *Intrusion Detection System*: This section describes the usage of the Intrusion Detection System (IDS).
- *References*: This section lists the documents that can be referred to for more information.
- *Acronyms and abbreviations*: This section lists the acronyms used in this document.

Prerequisites

The Bluetooth Low Energy Host Stack library contains several external references that the application must define to enable full functionality.

Attention: Application developers must ensure to define these references to prevent linkage errors when trying to build the application binary.

Task and event queues

The task queues are declared in the *ble_host_tasks.h* as follows:

```
/*! App to Host message queue for the Host Task */
    extern messaging_t   gApp2Host_TaskQueue;
/*! HCI to Host message queue for the Host Task */
    extern messaging_t   gHci2Host_TaskQueue;
/*! Event for the Host Task Queue */
    extern OSA_EVENT_HANDLE_DEFINE(gHost_TaskEvent);
```

See Initialization for more details about the RTOS tasks required by the Bluetooth LE Host Stack.

Parent topic:*Prerequisites*

GATT database

The application must define and populate the database according to its requirements and constraints either statically, at application compile time, or dynamically.

Regardless of how the application creates the GATT database, the following two external references from *gatt_database.h* must be defined:

```
/*! The number of attributes in the GATT Database. */
extern uint16_t gGattDbAttributeCount_c;
/*! Reference to the GATT database */
extern gattDbAttribute_t* gattDatabase;
```

The attribute template is defined as shown here:

```
typedef struct{
    uint16_t handle ;
/*!< Attribute handle - cannot be 0x0000; attribute handles need not be consecutive, but must be strictly␣
↪increasing. */
    uint16_t permissions ;
/*!< Attribute permissions as defined by ATT. */
    uint32_t uuid ;
/*!< The UUID should be read according to the gattDbAttribute_t.uuidType member: for 2-byte and 4-byte␣
↪UUIDs, this contains the value of the UUID; for 16-byte UUIDs, this is a pointer to the allocated 16-byte␣
↪array containing the UUID. */
    uint8_t * pValue ;
/*!< Pointer to allocated value array. */
    uint16_t valueLength ;
/*!< Size of the value array. */
    uint16_t uuidType : 2;
/*!< Identifies the length of the UUID; the 2-bit values are interpreted according to the bleUuidType_t␣
↪enumeration. */
    uint16_t maxVariableValueLength : 10;
/*!< Maximum length of the attribute value array; if this is set to 0, then the attribute's length␣
↪(valueLength) is fixed and cannot be changed. */
} gattDbAttribute_t ;
```

Parent topic:*Prerequisites*

Non-Volatile Memory (NVM) access

The Bluetooth LE Host Stack implements an internal module responsible for managing device information. This module relies on accessing a Non-Volatile Memory module for storing and loading bonded devices data.

The application developers determine the NVM access mechanism through the definition of three functions and one variable. The functions must first pre-process the information and then perform standard NVM operations (erase, write, read). The declarations are as follows:

```
bleResult_t App_NvmErase
(
    uint8_t mEntryIdx
);
bleResult_t App_NvmRead
(
    uint8_t  mEntryIdx,
    void*    pBondHeader,
    void*    pBondDataDynamic,
    void*    pBondDataStatic,
    void*    pBondDataLegacy,
    void*    pBondDataDeviceInfo,
    void*    pBondDataDescriptor,
    uint8_t  mDescriptorIndex
);
bleResult_t App_NvmWrite
(
    uint8_t  mEntryIdx,
    void*    pBondHeader,
    void*    pBondDataDynamic,
    void*    pBondDataStatic,
    void*    pBondDataLegacy,
    void*    pBondDataDeviceInfo,
    void*    pBondDataDescriptor,
    uint8_t  mDescriptorIndex
);
```

The device information is divided into several components to ensure that even software wear leveling mechanisms can be used optimally. The components sizes are fixed (defined in *ble_constants.h*) and have the following meaning:

API pointer to bond component	Component size (*ble_constants.h*)	Description
pBondHeader: points to a bleBondIdentityHeaderBlob_t element	gBleBondIdentityHeaderSize_c	Bonding information which is sufficient to identify a bonded device.
pBondDataDynamic: points to a bleBondDataDynamicBlob_t element	gBleBondDataDynamicSize_c	Bonding information that might change frequently.
pBondDataStatic: points to a bleBondDataStaticBlob_t element	gBleBondDataStaticSize_c	Bonding information that is unlikely to change frequently.
pBondDataLegacy: points to a bleBondDataLegacyBlob_t element	gBleBondDataLegacySize_c	Stores legacy pairing bond information.
pBondDataDeviceInfo: points to a bleBondDataDeviceInfoBlob_t element	gBleBondDataDeviceInfoSize_c	Additional bonding information that can be accessed using the host stack API.
pBondDataDescriptor: points to a bleBondDataDescriptorBlob_t element	gBleBondDataDescriptorSize_c	Bonding information used to store one Client Characteristic Configuration Descriptor (CCCD).

The Bluetooth LE Host Stack handles the format of the bonding information. Therefore, application developers need not to take care of this aspect.

Each bonding data slot must contain one bonding header blob, one dynamic data blob, one static data blob, one data legacy blob, one device information blob, and an array of descriptor blobs equal to *gcGapMaximumSavedCccds_c.*

Note: *The application must define the* gcGapMaximumSavedCccds_c. *macro according to its requirement. The default value can be found in the* ble_constants.h *file.*)

A slot is uniquely identified by the *mEntryIdx* parameter.

A descriptor is uniquely identified by the pair *mEntryIdx - mDescriptorIndex*.

If one or more pointers passed as parameters are NULL, the read from or write to the corresponding blob of the bonding slot must be ignored. The erase function must clear the entire bonding data slot specified by the entry index.

Note:

When Advanced Secure Mode is chosen (gAppSecureMode_d is defined as 1 in app_preinclude.h), two additional application NVM functions are defined to handle local keys encrypted blob storage. Their declaration is:

```
bleResult_t App_NvmWriteLocalKeys
(
    uint8_t mEntryIdx,
    void* pLocalKey
)
bleResult_t App_NvmReadLocalKeys
(
    uint8_t mEntryIdx,
    void* pLocalKey
)
```

The functions write/read a structure of type bleLocalKeysBlob_t into/from NVM using a dedicated data set. The parameter mEntryIdx is 0 (local IRK is handled).

The format of the local keys blob nor about the generation and storage of the local keys is automatically handled in **BLE Connection Manager** (*BleConnManager_GenericEvent*). Therefore the application developer need not manage this aspect.

The current implementation of the aforementioned functions uses either the framework NVM module or a RAM buffer. Additional details about the NVM configuration and functionality can be found in the *Connectivity Framework Reference Manual.* See *References*.

To enable the NVM mechanism, ensure the following points:

- gAppUseNvm_d (in *app_preinclude.h*) is set to 1 and
- gUseNVMLink_d is set to 1 in the linker options of the toolchain.

Note:

- If gAppUseNvm_d is set to 0, then all bonding data is stored in the RAM and is accessible until reset or power cycle.
- If gAppUseNvm_d is set to 1, the default NVM module configurations are applied in the *app_preinclude.h* file.

Parent topic:*Prerequisites*

Bluetooth LE Host Stack Initialization and APIs

Initialization

The application developer is required to configure the Host Task as part of the Host Stack requirement. The task is the context for running all the Host layers (GAP, GATT, ATT, L2CAP, SM, GATTDB)

The prototype of the task function is located in the *ble_host_tasks.h* file:

```
void Host_TaskHandler(void * args);
```

It should be called with *NULL* as an argument in the task code from the application.

Application developers are required to define task events and queues as explained in RTOS Task Queues and Events.

If the Controller software runs on the same chip as the Host, the Controller task always has a higher priority than the Host task. The priority value of the Host Task can be configured through the *gHost_TaskPriority_c define* (by default set in *ble_host_task_config.h*). Note that changing this value can have a significant impact on the Bluetooth Low Energy stack.

Parent topic:*Bluetooth LE Host Stack Initialization and APIs*

Main function to initialize the Bluetooth LE Host Stack

This Figure provides an overview of Bluetooth Low Energy Host Stack. When using the existing application common files, the startup task uses *BluetoothLEHost_AppInit()*, which is defined in *app_conn.h*. The function initializes all components related to the Bluetooth Low Energy application. It has the following prototype:

```
void BluetoothLEHost_AppInit(void);
```

The *BluetoothLEHost_AppInit()* function must be implemented by each application. It should register its generic event callback using *BluetoothLEHost_SetGenericCallback()* and initialize the Bluetooth LE Host Stack layer by calling *BluetoothLEHost_Init()*. The prototype for the *BluetoothLEHost_Init()* function is found in *app_conn.h* and is implemented in *app_conn.c*.

```
void BluetoothLEHost_Init
(
    appBluetoothLEInitCompleteCallback_t pCallback
);
```

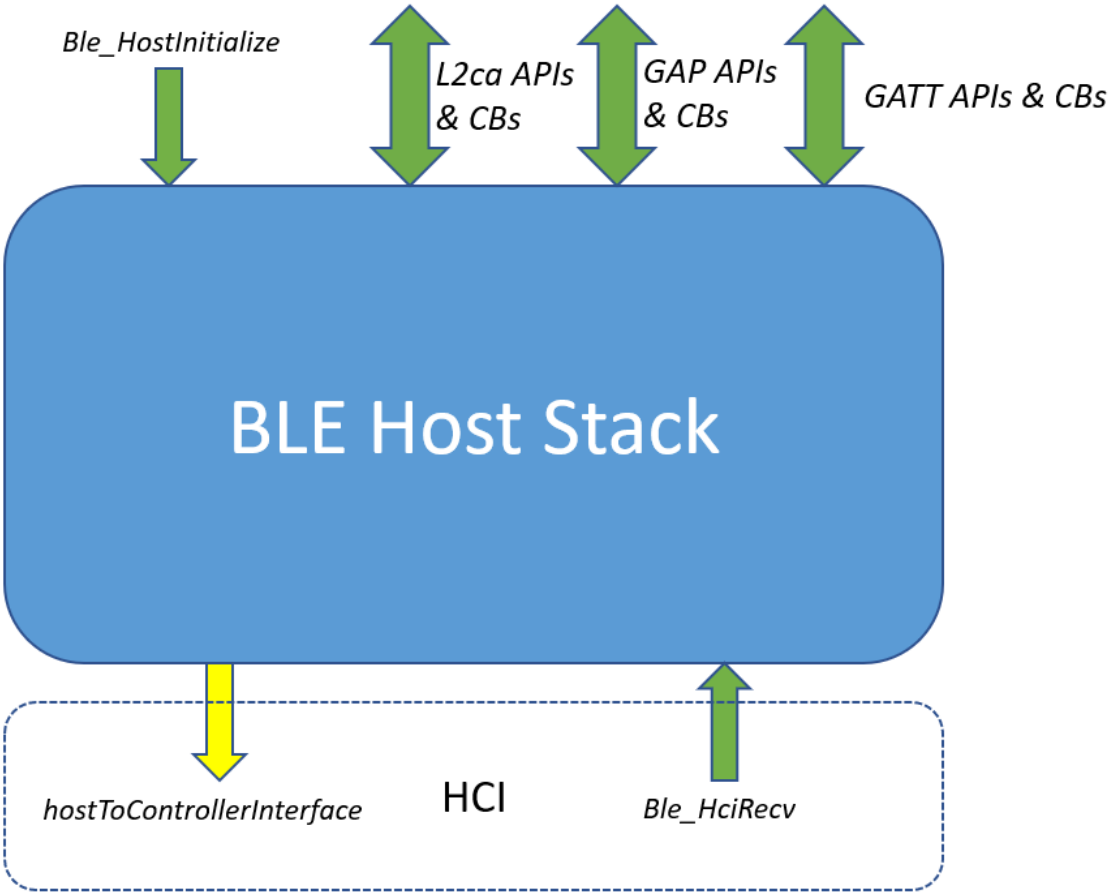
BluetoothLEHost_Init takes as parameter a function to be called at the end of Bluetooth LE Host Stack initialization. In this callback, the application can register its callbacks with the Host layer, allocate timers, start services, and perform similar tasks. The callback should have a prototype as follows:

```
static void BluetoothLEHost_Initialized(void);
```

BluetoothLEHost_Init() is responsible for initializing the Host.

Initialize the Bluetooth LE Host Stack after platform setup is complete and all RTOS tasks have been started. The function that should be called for this purpose is located in the *ble_general.h* file and has the following prototype:

```
bleResult_t Ble_HostInitialize
(
    gapGenericCallback_t              genericCallback,
    hciHostToControllerInterface_t    hostToControllerInterface
);
```



Parent topic:*Bluetooth LE Host Stack Initialization and APIs*

HCI entry and exit points

The HCI entry point of the Host Stack is the second function located in the *ble_general.h* file:

```
void Ble_HciRecv
(
    hciPacketType_t packetType,
    void* pHciPacket,
    uint16_t packetSize
);
```

This is the function that the application must call to insert an HCI message into the Host.

An equivalent exists, to be used in ISR context:

```
bleResult_t Ble_HciRecvFromIsr
(
    hciPacketType_t      packetType,
    void*                pHciPacket,
    uint16_t             packetSize
);
```

Therefore, the *Ble_HciRecv* function and the *hostToControllerInterface* parameter of the *Ble_HostInitialize* function represent the two points that need to be connected to the LE Controller (see Bluetooth Low Energy Host Stack overview), either directly (if the Controller software runs on the same chip as the Host) or through a physical interface (for example, UART).

Parent topic:*Bluetooth LE Host Stack Initialization and APIs*

Bluetooth LE Host Stack libraries and API availability

All the APIs referenced in this document are available in the Central and Peripheral libraries. The support for Bluetooth 5.3 optional features such as Advertising, Advertising Extensions, GATT Caching, and EATT are provided in separate host libraries. They are distributed in a similar process as the legacy ones using GAP/GATT role support, which is described as follows.

For example, below are listed the full-featured libraries with complete support for both Central and Peripheral APIs, at GAP level.

- *lib_ble_OPT_host_cm33_iar.a* (for IAR projects)
- *lib_ble_OPT_host_cm33_gcc.a* (for MCUX projects)

These libraries include optional features implemented by the Bluetooth LE Host. For applications that need to use only the mandatory 5.3 Bluetooth LE and below features, the *lib_ble_host_cm33_iar.a* or *lib_ble_host_cm33_gcc.a* libraries can be used instead.

However, some applications may be targeted to memory-constrained devices and do not need the full support. In the interest of reducing code size and RAM utilization, optimized libraries are provided:

- *lib_ble_host_peripheral_cm33_iar.a/ lib_ble_host_peripheral_cm33_gcc.a* and
- *lib_ble_OPT_host_peripheral_cm33_iar.a / lib_ble_OPT_host_peripheral_cm33_gcc.a*.
  - Support only APIs for the GAP Peripheral and GAP Broadcaster roles
  - Support only APIs for the GATT Server role
- *lib_ble_host_central_cm33_iar.a* and *lib_ble_OPT_host_central_cm33_iar.a*
- *lib_ble_host_central_cm33_gcc.a* and *lib_ble_OPT_host_central_cm33_gcc.a*
  - Support only APIs for the GAP Central and GAP Observer roles
  - Support only APIs for the GATT Client role

If one attempts to use an API that is not supported (for instance, calling *Gap_Connect* with the *lib_ble_host_peripheral_cm33_iar.a* and *lib_ble_host_peripheral_cm33_gcc.a*), then the API returns the *gBleFeatureNotSupported_c* error code.

Similarly, if the API for OPT is used with a host library that does not have support for optional features, then *gBleFeatureNotSupported_c* is returned. For instance, calling *Gap_SetExtAdvertising* parameters with the *lib_ble_host_peripheral_cm33_iar.a* and *lib_ble_host_peripheral_cm33_gcc.a*) returns the exit code *gBleFeatureNotSupported_c*.

Note: See the *Bluetooth Low Energy Host Stack API Reference Manual* for explicit information regarding API support. Each function documentation contains this information in the Remarks section.

Parent topic:*Bluetooth LE Host Stack Initialization and APIs*

Synchronous and asynchronous functions

The vast majority of the GAP and GATT APIs are executed **asynchronously**. Calling these functions generates a message and places it in the Host Task message queue.

Therefore, the actual result of these APIs is signaled in **events** triggered by specific callbacks installed by the application. See the *Bluetooth Low Energy Host Stack API Reference Manual* for specific information about the events that are triggered by each API.

However, there are a few APIs which are executed immediately (**synchronously**). This is explicitly mentioned in the *Bluetooth Low Energy Host Stack API Reference Manual* in the *Remarks* section of each function documentation.

If nothing is mentioned, then the API is asynchronous.

Parent topic:*Bluetooth LE Host Stack Initialization and APIs*

Radio TX Power level

The controller interface includes APIs that can be used to set the Radio TX Power to a different level than the default one.

The power level can be set differently for advertising and connection channels by calling the function *Controller_SetTxPowerLevelDbm()* with the channel parameter set to *gAdvTxChannel_c* or *gConnTxChannel_c*.

Parent topic:*Bluetooth LE Host Stack Initialization and APIs*

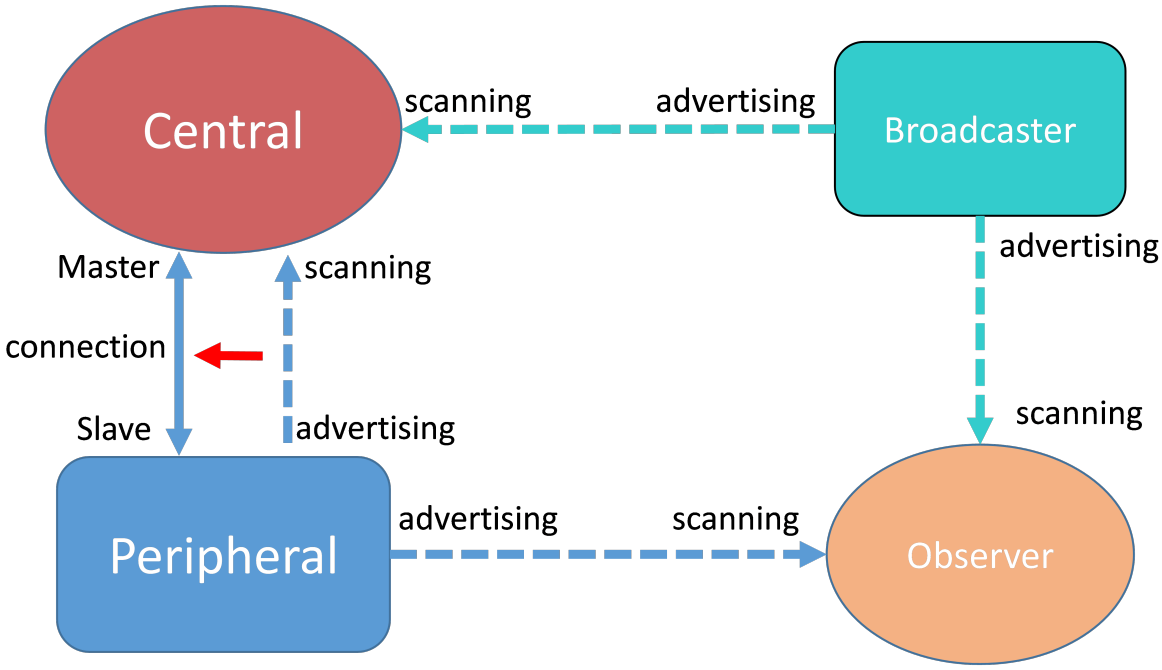
Generic Access Profile (GAP) Layer

The GAP layer manages connections, security, and bonded devices.

The GAP layer APIs are built on top of the Host-Controller Interface (HCI), the Security Manager Protocol (SMP), and the Device database.

GAP defines four possible roles that a Bluetooth Low Energy device may have in a Bluetooth Low Energy system:

- **Central**
  - Scans for advertisers (Peripherals and Broadcasters)
  - Initiates connection to Peripherals; Central at Link Layer (LL) level
  - Usually acts as a GATT Client, but can also contain a GATT Database itself
- **Peripheral**
  - Advertises and accepts connection requests from Central devices; LL Peripheral
  - Usually contains a GATT Database and acts as a GATT Server, but may also be a Client
- **Observer**
  - Scans for advertisers, but does not initiate connections; Transmit is optional
- **Broadcaster**
  - Advertises, but does not accept connection requests from Central devices; Receive is optional



The Figure illustrates the generic GAP topology.

Peripheral setup

The Peripheral starts advertising and waits for scan and connection requests from other Central devices.

Advertising

Before starting advertising, the advertising parameters should be configured. Otherwise, the following defaults are used.

```
#define gGapDefaultAdvertisingParameters_d \
{
  /* minInterval */      gGapAdvertisingIntervalDefault_c, \
  /* maxInterval */      gGapAdvertisingIntervalDefault_c, \
  /* advertisingType */ gConnectableUndirectedAdv_c, \
  /* addressType */      gBleAddrTypePublic_c, \
  /* peerAddressType */  gBleAddrTypePublic_c, \
  /* peerAddress */      {0U, 0U, 0U, 0U, 0U, 0U}, \
  /* channelMap */       (gapAdvertisingChannelMapFlags_t)gGapAdvertisingChannelMapDefault_c,  \
  /* filterPolicy */    gProcessAll_c \
}
```

To set different advertising parameters, a *gapAdvertisingParameters_t* structure should be allocated and initialized with defaults. Then, the necessary fields may be modified.

After that, the following function should be called:

```
bleResult_t Gap_SetAdvertisingParameters
(
   const gapAdvertisingParameters_t *      pAdvertisingParameters
);
```

The application should listen to the *gAdvertisingParametersSetupComplete_c* generic event.

Next, the advertising data should be configured and, if the advertising type supports active scanning, the scan response data should also be configured. If either of these is not configured, they are defaulted to empty data.

The function used to configure the advertising and/or scan response data is shown here:

```
bleResult_t Gap_SetAdvertisingData
(
  const gapAdvertisingData_t *      pAdvertisingData,
  const gapScanResponseData_t *     pScanResponseData
);
```

Either of the two pointers may be *NULL*, in which case they are ignored (the corresponding data is left as it was previously configured, or empty if it has never been set), but not both at the same time.

The application should listen to the *gAdvertisingDataSetupComplete_c* generic event.

After all the necessary setup is done, advertising may be started with this function:

```
bleResult_t Gap_StartAdvertising
(
   gapAdvertisingCallback_t advertisingCallback,
   gapConnectionCallback_t connectionCallback
);
```

The advertising callback is used to receive advertising events (advertising state changed or advertising command failed), while the connection callback is only used if a connection is established during advertising.

The connection callback is the same as the callback used by the Central when calling the *Gap_Connect* function.

When using the common application structure, the application can use the following API defined in *app_conn.h*:

```
bleResult_t BluetoothLEHost_StartAdvertising
(
    appAdvertisingParams_t    *pAdvParams,
    gapAdvertisingCallback_t pfAdvertisingCallback,
    gapConnectionCallback_t   pfConnectionCallback
);
```

The API goes through the steps of setting the advertising data and parameters. Events from the Host task are treated in the *App_AdvertiserHandler()* function, implemented in *app_advertiser.c*. To set the advertising parameters and data *BluetoothLEHost_StartAdvertising* requires a parameter of the following type:

```
typedef struct
appAdvertisingParams_tag
{
 gapAdvertisingParameters_t *pGapAdvParams; /*!< Pointer to the GAP advertising parameters */
 const gapAdvertisingData_t *pGapAdvData; /*!< Pointer to the GAP advertising data */
 const gapScanResponseData_t *pScanResponseData;
 /*!< Pointer to the scan response data */ } appAdvertisingParams_t;
```

If a Central initiates a connection to this Peripheral, the *gConnEvtConnected_c* connection event is triggered.

To stop advertising while the Peripheral has not yet received any connection requests, use this function:

```
bleResult_t Gap_StopAdvertising (void);
```

This function should not be called after the Peripheral enters a connection, as the advertising automatically stops in this case.

Parent topic: Peripheral setup

Pairing and bonding (peripheral)

After a connection has been established to a Central, the Peripheral's role regarding security is a passive one. It is the responsibility of the Central device to start the pairing process. In case, the devices have already bonded in the past, the Central encrypts the link using the shared LTK.

The Peripheral sends error responses (at ATT level) with proper error code if the Central attempts to access sensitive data without authenticating. Examples of such error responses are: Insufficient Authentication, Insufficient Encryption, Insufficient Authorization, and so on. Therefore, it indicates to the Central that it needs to perform security procedures.

All security checks are performed internally by the GAP module and the security error responses are sent automatically. All the application developer needs to do is register the security requirements.

First, when building the GATT Database (see *Creating GATT database*), the sensitive attributes should have the security built into their access permissions (for example, read-only / read with authentication / write with authentication / write with authorization, and so on.).

Second, if the GATT Database requires additional security besides that already specified in attribute permissions (for example, certain services require higher security in certain situations), the following function must be called:

```
bleResult_t Gap_RegisterDeviceSecurityRequirements
(
    const gapDeviceSecurityRequirements_t * pSecurity
);
```

The parameter is a pointer to a structure which contains a "device security setting" and service-specific security settings. All these security requirements are pointers to gapSecurityRequirements_t structures. The pointers that are to be ignored should be set to NULL.

Although the Peripheral does not initiate any kind of security procedure, it can inform the Central about its security requirements.

The informing is performed through the Peripheral Security Request packet at SMP level. To use it, the following GAP API is provided:

```
bleResult_t Gap_SendPeripheralSecurityRequest
(
    deviceId_t                    deviceId,
    const gapPairingParameters_t* pPairingParameters
);
```

The gapPairingParameters_t structure includes two important fields. The withBonding field indicates to the Central whether this Peripheral can bond and the securityModeAndLevel field informs about the required security mode and level that the Central should pair for. See Pairing and bonding (Central) for an explanation about security modes and levels, as defined by the GAP module.

This request expects no reply, nor any immediate action from the Central. The Central may easily choose to ignore the Peripheral Security Request.

If the two devices have bonded in the past, the Central proceeds directly to encrypting the link. If the bond was not made using LE Secure Connections, the Peripheral expects to receive a gConnEvtLongTermKeyRequest_c connection event. If the bond was made using LE Secure Connections, the Host provides the LTK automatically to the LE Controller.

When the devices have been previously pairing without using LE Secure Connections, along with the Peripheral's LTK, the EDIV (2 bytes) and RAND (8 bytes) values were also sent (their meaning is defined by the SMP). Therefore, before providing the key to the Controller, the application should check that the two values match with those received in the gConnEvtLongTermKeyRequest_c event. If they do, the application should reply with:

```
bleResult_t Gap_ProvideLongTermKey
(
    deviceId_t          deviceId,
    const uint8_t*      aLtk,
    uint8_t             ltkSize
);
```

The LTK size cannot exceed the maximum value of 16.

If the EDIV and RAND values do not match, or if the Peripheral does not recognize the bond, it can reject the encryption request with:

```
bleResult_t Gap_DenyLongTermKey
(
    deviceId_t deviceId
);
```

If LE SC Pairing was used then the LTK is generated internally by the Bluetooth LE Host Stack and it is not requested from the application during post-bonding link encryption. In this scenario, the application is only notified of the link encryption through the gConnEvtEncryptionChanged_c connection event.

If the devices are not bonded, the Peripheral should expect to receive the gConnEvtPairingRequest_c, indicating that the Central has initiated pairing.

If the application agrees with the pairing parameters (see Pairing and bonding (Central) for detailed explanations), it can reply with:

```
bleResult_t Gap_AcceptPairingRequest
(
    deviceId_t                      deviceId,
    const gapPairingParameters_t *   pPairingParameters
);
```

This time, the Peripheral sends its own pairing parameters, as defined by the SMP.

After sending this response, the application should expect to receive the same pairing events as the Central (see Pairing and bonding (Central)), with one exception: the gConnEvtPasskeyRequest_c event is not called if the application sets the Passkey (PIN) for pairing before the connection by calling the API:

```
bleResult_t Gap_SetLocalPasskey
(
    uint32_t passkey
);
```

This is done because, usually, the Peripheral has a static secret PIN that it distributes only to trusted devices. If, for any reason, the Peripheral must dynamically change the PIN, it can call the aforementioned function every time it wants to, before the pairing starts (for example, right before sending the pairing response with Gap_AcceptPairingRequest).

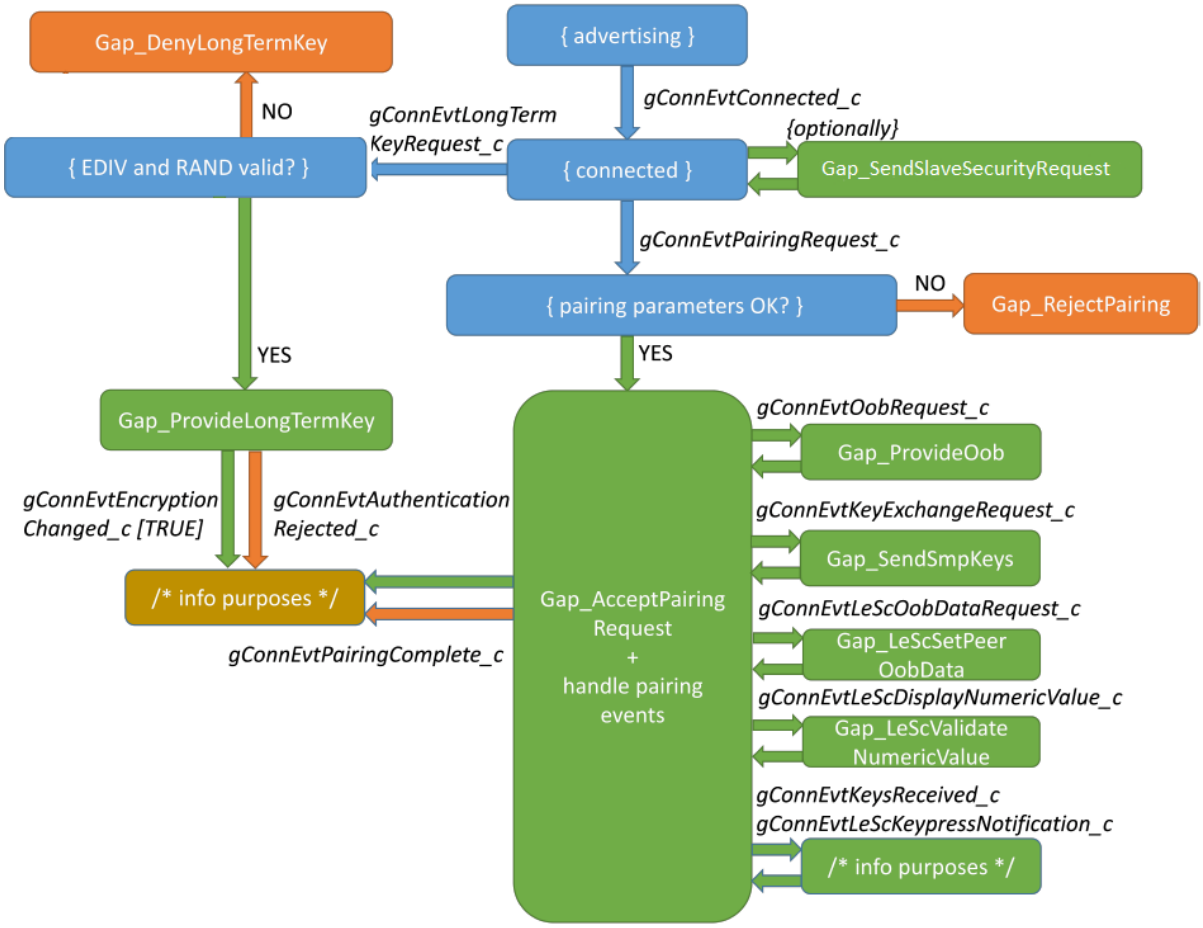
If the Peripheral application never calls Gap_SetLocalPasskey, then the gConnEvtPasskeyRequest_c event is sent to the application as usual.

The Peripheral can use the following API to reject the pairing process:

```
bleResult_t Gap_RejectPairing
(
deviceId_t                      deviceId,
gapAuthenticationRejectReason_t     reason
);
```

The reason should indicate why the application rejects the pairing. The value gLinkEncryptionFailed_c is reserved for the gConnEvtAuthenticationRejected_c connection event to indicate the link encryption failure rather than pairing failures. Therefore, it is not meant as a pairing reject reason.

The Gap_RejectPairing function may be called not only after the Pairing Request was received, but also during the pairing process. For example, when handling pairing events or asynchronously, if for any reason the Peripheral decides to abort the pairing, this function can be called. This also holds true for the Central. Figure illustrates the Peripheral pairing flow and lists the main APIs and events. Gap_RejectPairing can be called on any pairing event.



For both the Central and the Peripheral, bonding is performed internally and is not the application's concern. The gConnEvtPairingComplete_c event parameters inform the application if bonding has occurred.

Parent topic: Peripheral setup

Parent topic:*Generic Access Profile (GAP) Layer*

Central setup

Usually, a Central must start scanning to find Peripherals. When the Central has scanned a Peripheral it wants to connect to, it stops scanning and initiates a connection to that Peripheral. After the connection has been established, it may start pairing, if the Peripheral requires it, or directly encrypt the link, if the two devices have already bonded in the past.

Scanning

The most basic setup for a Central device begins with scanning, which is performed by the following function from *gap_interface.h*:

```
bleResult_t Gap_StartScanning
(
  const gapScanningParameters_t* pScanningParameters,
  gapScanningCallback_t scanningCallback,
  gapFilterDuplicates_t enableFilterDuplicates,
  uint16_t duration,
  uint16_t period
);
```

If the *pScanningParameters* pointer is NULL, the currently set parameters are used. If no parameters have been set after a device power-up, the standard default values are used:

```
#define gGapDefaultScanningParameters_d \
{ \
    /* type */              gScanTypePassive_c, \
    /* interval */          gGapScanIntervalDefault_d, \
    /* window */            gGapScanWindowDefault_d, \
    /* ownAddressType */    gBleAddrTypePublic_c, \
    /* filterPolicy */      gScanAll_c \
    /* scanning PHY */      gLePhy1MFlag_c\
}
```

The easiest way to define non-default scanning parameters is to initialize a *gapScanningParameters_t* structure with the above default and change only the required fields.

For example, to perform active scanning and only scan for devices in the Filter Accept List, the following code can be used:

```
gapScanningParameters_t scanningParameters = gGapDefaultScanningParameters_d;
scanningParameters.type = gScanTypeActive_c;
scanningParameters.filterPolicy = gScanWithFilterAcceptList_c;
Gap_StartScanning(&scanningParamters, scanningCallback, enableFilterDuplicates, duration, period);
```

When using the common application structure, the application can use the following API defined in *app_conn.h*:

```
bleResult_t BluetoothLEHost_StartScanning
(
    appScanningParams_t    *pAppScanParams,
    gapScanningCallback_t pfCallback
);
```

The API uses the *appScanningParams_t* structure, which is defined as follows:

```
typedef struct appScanningParams_tag
{
    gapScanningParameters_t *pHostScanParams;          /*!< Pointer to host scan structure */
    gapFilterDuplicates_t enableDuplicateFiltering;    /*!< Duplicate filtering mode */
    uint16_t duration;                                 /*!< scan duration  */
    uint16_t period;                                   /*!< scan period  */
} appScanningParams_t;
```

The *scanningCallback* is triggered by the GAP layer to signal events related to scanning.

The most important event is the *gDeviceScanned_c* event, which is triggered each time an advertising device is scanned. This event data contains information about the advertiser:

```
typedef struct
{
    bleAddressType_t                addressType ;
    bleDeviceAddress_t              aAddress ;
    int8_t                          rssi ;
    uint8_t                         dataLength ;
    uint8_t*                        data ;
    bleAdvertisingReportEventType_t   advEventType ;
    bool_t                          directRpaUsed;
    bleDeviceAddress_t              directRpa;
    bool_t                          advertisingAddressResolved;
} gapScannedDevice_t;
```

If this information signals a known Peripheral that the Central wants to connect to, the latter must stop scanning and connect to the Peripheral.

To stop scanning, call this function:

```
bleResult_t Gap_StopScanning (void);
```

By default, the GAP layer is configured to report all scanned devices to the application using the *gDeviceScanned_c* event type. However, some use cases might require to perform specific GAP Discovery Procedures. In such use cases the advertising reports might require the filtering of Flags AD value from the advertising data. Other use cases require the Bluetooth LE Host Stack to automatically initiate a connection when a specific device has been scanned.

To enable filtering based on the Flags AD value or to set device addresses for automatic connections, the following function must be called before the scanning is started:

```
bleResult_t Gap_SetScanMode
(
    gapScanMode_t            scanMode,
    gapAutoConnectParams_t* pAutoConnectParams,
    gapConnectionCallback_t connCallback
);
```

The default value for the scan mode is *gDefaultScan_c,* which reports all packets regardless of their content and does not perform any automatic connection.

To enable Limited Discovery, the *gLimitedDiscovery_c* value must be used, while the *gGeneralDiscovery_c* value activates General Discovery.

To enable automatic connection when specific devices are scanned, the *gAutoConnect_c* value must be set, in which case the *pAutoConnectParams* parameter must point to the structure that holds the target device addresses and the connection parameters to be used by the Host for these devices.

If scanMode is set to *gAutoConnect_c*, connCallback must be set and is triggered by GAP to send the events related to the connection.

Parent topic: Central setup

Initiating and closing a connection

To connect to a scanned Peripheral, extract its address and address type from the *gDeviceScanned_c* event data, stop scanning, and call the following function:

```
bleResult_t Gap_Connect
(
const gapConnectionRequestParameters_t * pParameters,
gapConnectionCallback_t connCallback
);
```

When using the common application structure, the application can also use the following API defined in *app_conn.h*:

```
bleResult_t BluetoothLEHost_Connect
(
    gapConnectionRequestParameters_t*    pParameters,
    gapConnectionCallback_t              connCallback
);
```

An easy way to create the connection parameter structure is to initialize it with the defaults, then change only the necessary fields. The default structure is defined as shown here:

```
#define gGapDefaultConnectionRequestParameters_d \
{ \
    /* scanInterval */        gGapScanIntervalDefault_d, \
    /* scanWindow */          gGapScanWindowDefault_d, \
```

(continues on next page)

(continued from previous page)

```
    /* filterPolicy */        gUseDeviceAddress_c, \
    /* ownAddressType */      gBleAddrTypePublic_c, \
    /* peerAddressType */     gBleAddrTypePublic_c, \
    /* peerAddress */         { 0, 0, 0, 0, 0, 0 }, \
    /* connIntervalMin */     gGapDefaultMinConnectionInterval_d, \
    /* connIntervalMax */     gGapDefaultMaxConnectionInterval_d, \
    /* connLatency */         gGapDefaultConnectionLatency_d, \
    /* supervisionTimeout */ gGapDefaultSupervisionTimeout_d, \
    /* connEventLengthMin */ gGapConnEventLengthMin_d, \
    /* connEventLengthMax */ gGapConnEventLengthMax_d \
    /* initiatingPHYs      */ gLePhylMFlag_c \
}
```

In the following example, Central scans for a specific Heart Rate Sensor with a known address. When it finds it, it immediately connects to it.

```
static void BleApp_ScanningCallback
(
    gapScanningEvent_t *pScanningEvent
)
{
    switch (pScanningEvent->eventType)
    {
        case gDeviceScanned_c:
        {
            if (BleApp_CheckScanEvent(&pScanningEvent->eventData.scannedDevice))
            {
                gConnReqParams.peerAddressType = pScanningEvent->eventData.scannedDevice.addressType;
                FLib_MemCpy(gConnReqParams.peerAddress,
                        pScanningEvent->eventData.scannedDevice.aAddress,
                        sizeof(bleDeviceAddress_t));
                (void)Gap_StopScanning();
#if gAppUsePrivacy_d
                gConnReqParams.usePeerIdentityAddress =  pScanningEvent->eventData.scannedDevice.
↪advertisingAddressResolved;
#endif
                (void)BluetoothLEHost_Connect(&gConnReqParams, BleApp_ConnectionCallback);
            }
        }
        break;
}
```

The *connCallback* is triggered by GAP to send all events related to the active connection. It has the following prototype:

```
typedef void (* gapConnectionCallback_t )
(
    deviceId_t              deviceId,
    gapConnectionEvent_t * pConnectionEvent
);
```

The very first event that should be listened inside this callback is the *gConnEvtConnected_c* event. If the application decides to drop the connection establishment before this event is generated, it should call the following macro:

```
#define Gap_CancelInitiatingConnection()\
    Gap_Disconnect(gCancelOngoingInitiatingConnection_d)
```

This is useful, for instance, when the application chooses to use an expiration timer for the connection request.

Upon receiving the *gConnEvtConnected_c* event, the application may proceed to extract the nec-

essary parameters from the event data (*pConnectionEvent->event.connectedEvent*). The most important parameter to be saved is the *deviceId*.

The *deviceId* is a unique 8-bit, unsigned integer, used to identify an active connection for subsequent GAP and GATT API calls. All functions related to a certain connection require a *deviceId* parameter. For example, to disconnect, call this function:

```
bleResult_t Gap_Disconnect
(
    deviceId_t deviceId
);
```

Parent topic: Central setup

Pairing and bonding (Central)

After the user has connected to a Peripheral, use the following function to check whether this device has bonded in the past:

```
bleResult_t Gap_CheckIfBonded
(
    deviceId_t    deviceId,
    bool_t *      pOutIsBonded,
    uint8_t*      pOutNvmIndex
);
```

If it has, link encryption can be requested with:

```
bleResult_t Gap_EncryptLink
(
    deviceId_t      deviceId
);
```

If the link encryption is successful, the *gConnEvtEncryptionChanged_c* connection event is triggered. Otherwise, a *gConnEvtAuthenticationRejected_c*event is received with the *rejectReason* event data parameter set to *gLinkEncryptionFailed_c*.

On the other hand, if this is a new device (not bonded), pairing may be started as shown here:

```
bleResult_t Gap_Pair
(
    deviceId_t                      deviceId,
    const gapPairingParameters_t *    pPairingParameters
);
```

The pairing parameters are shown here:

```
typedef struct gapPairingParameters_tag {
    bool_t                      withBonding ;
    gapSecurityModeAndLevel_t      securityModeAndLevel ;
    uint8_t                     maxEncryptionKeySize ;
    gapIoCapabilities_t            localIoCapabilities ;
    bool_t                      oobAvailable ;
    gapSmpKeyFlags_t               centralKeys ;
    gapSmpKeyFlags_t               peripheralKeys ;
    bool_t                      leSecureConnectionSupported ;
    bool_t                      useKeypressNotifications ;
} gapPairingParameters_t;
```

The names of the parameters are self-explanatory. The *withBonding* flag should be set to *TRUE* if the Central must/wants to bond.

When Advanced Secure Mode is enabled, (*gAppSecureMode_d* id defined as *1* in *app_preinclude.h*), the security mode and level for pairing is automatically enforced as Mode 1

Level 4, and LE Secure Connection Supported is automatically enforced TRUE. Legacy pairing is not supported in this mode.

For the Security Mode and Level, the GAP layer defines them as follows:

- Security Mode 1 Level 1 stands for no security requirements.
- Except for Level 1 (which is only used with Mode 1), Security Mode 1 requires encryption, while Security Mode 2 requires data signing.
- Mode 1 Level 2 and Mode 2 Level 1 do not require authentication (in other words, they allow Just Works pairing, which has no MITM protection). Mode 1 Level 3 and Mode 2 Level 2 require authentication (must pair with PIN or OOB data, which provide MITM protection).
- Starting with Bluetooth specification 4.2, OOB pairing offers MITM protection only in certain conditions. The application must inform the stack if the OOB data exchange capabilities offer MITM protection via a dedicated API.
- Security Mode 1 Level 4 is reserved for authenticated pairing (with MITM protection) using a LE Secure Connections pairing method.
- If a pairing method is used but it does not offer MITM protection, then the pairing parameters must use Security Mode 1 level 2. If the requested pairing parameters are incompatible (for example, Security Mode 1 Level 4 without LE Secure Connections enabled), a gBleInvalidParameter_c status is returned by the security API functions: Gap_SetDefaultPairingParameters, Gap_SendPeripheralSecurityRequest, Gap_Pair and Gap_AcceptPairingRequest.

—	No security	No MITM protection	Legacy MITM protection	LE secure connections with MITM protection
Mode 1 (encryption) distributed LTK (EDIV+RAND) or generated LTK	**Level 1** no security	**Level 2** unauthenticated encryption	**Level 3** authenticated encryption	**Level 4** LE SC authenticated encryption

The *centralKeys* should have the flags set for all the keys that are available in the application. The IRK is mandatory if the Central is using a Private Resolvable Address. The LTK is provided by the Peripheral and should only be included if the Central intends on becoming a Peripheral in future reconnections (GAP role change).

The *peripheralKeys* should follow the same guidelines. The LTK is mandatory if encryption is to be performed, while the peer's IRK should be requested if the Peripheral is using Private Resolvable Addresses.

See table below for detailed guidelines regarding key distribution.

The first three rows are both guidelines for Pairing Parameters (*centralKeys* and *peripheralKeys*) and for distribution of keys with *Gap_SendSmpKeys*.

If LE Secure Connections Pairing is performed (Bluetooth Low Energy 4.2 and above), then the LTK is generated internally, so the corresponding bits in the key distribution fields from the pairing parameters are ignored by the devices.

The Identity Address is distributed if the IRK is also distributed (its flag has been set in the Pairing Parameters). Therefore, it can be "asked" only by asking for IRK (it does not have a separate flag in a *gapSmpKeyFlags_t* structure). Therefore, it is N/A.

The negotiation of the distributed keys is as follows:

- In the SMP Pairing Request (started by *Gap_Pair*), the Central sets the flags for the keys it wants to distribute (*centralKeys*) and receive (*peripheralKeys*).

	Central keys (Central)	Peripheral keys (Central)	Peripheral () keys	Central keys (peripheral)
Long Term Key (LTK)+EDIV +RAND	If it wants to be a peripheral in a future reconnection	If it wants encryption	If it wants encryption	If it wants to become a central in a future reconnection
Identity Resolving Key (IRK)	If it uses or intends to use private resolvable addresses	If a peripheral is using a private resolvable address	If it uses or intends to use private resolvable addresses	If a central is using a private resolvable address
Identity address	If it distributes the IRK	N/A	If it distributes the IRK	N/A

- The Peripheral examines the two distributions and must send an SMP Pairing Response (started by the *Gap_AcceptPairingRequest*) after performing any changes it deems necessary. The Peripheral is only allowed to set to 0 some flags that are set to 1 by the Central, but not the other way around. For example, it cannot request/distribute keys that were not offered/requested by the Central. If the Peripheral is adverse to the Central's distributions, it can reject the pairing by using the *Gap_RejectPairing* function.
- The Central examines the updated distributions from the Pairing Response. If it is adverse to the changes made by the Peripheral, it can reject the pairing (*Gap_RejectPairing*). Otherwise, the pairing continues and, during the key distribution phase (the *gConnEvtKeyExchangeRequest_c* event) only the final negotiated keys are included in the key structure sent with *Gap_SendSmpKeys*.
- For LE Secure Connections (both devices set the SC bit in the AuthReq field of the Pairing Request and Pairing Response packets), the LTK is not distributed. It is generated and the corresponding bit in the Initiator Key Distribution and Responder Key Distribution fields of the Pairing Response packet are set to 0.

If LE Secure Connections Pairing (Bluetooth LE 4.2 and above) is used, and OOB data needs to be exchanged, the application must obtain the local LE SC OOB Data from the Bluetooth LE Host Stack by calling the *Gap_LeScGetLocalOobData* function. The data is contained by the generic *gLeScLocalOobData_c* event.

The local LE SC OOB Data is refreshed in the following situations:

- The *Gap_LeScRegeneratePublicKey* function is called (the *gLeScPublicKeyRegenerated_c* generic event is also generated as a result of this API).
- The device is reset (which also causes the Public Key to be regenerated).

If the pairing continues, the following connection events may occur:

- **Request events**
  - *gConnEvtPasskeyRequest_c*: a PIN is required for pairing; the application must respond with the *Gap_EnterPasskey(deviceId, passkey).*
  - *gConnEvtOobRequest_c*: if the pairing started with the *oobAvailable* set to *TRUE* by both sides; the application must respond with the *Gap_ProvideOob(deviceId, oob).*
  - *gConnEvtKeyExchangeRequest_c*: the pairing has reached the key exchange phase; the application must respond with the *Gap_SendSmpKeys(deviceId, smpKeys).*
  - *gConnEvtLeScOobDataRequest_c*: the stack requests the LE SC OOB Data received from the peer (r, Cr and Addr); the application must respond with *Gap_LeScSetPeerOobData(deviceId, leScOobData).*
  - *gConnEvtLeScDisplayNumericValue_c*: the stack requests the display and confirmation of the LE SC Numeric Comparison Value; the application must respond with *Gap_LeScValidateNumericValue(deviceId, ncvValidated).*

- **Informational events**
  - *gConnEvtKeysReceived_c*: the key exchange phase is complete; keys are automatically saved in the internal device database and are also provided to the application for immediate inspection; application does not have to save the keys in NVM storage because this is done internally if *withBonding*was set to TRUE by both sides.
  - *gConnEvtAuthenticationRejected_c*: the peer device rejected the pairing; the *rejectReason* parameter of the event data indicates the reason that the Peripheral does not agree with the pairing parameters (it cannot be *gLinkEncryptionFailed_c* because that reason is reserved for the link encryption failure).
  - *gConnEvtPairingComplete_c*: the pairing process is complete, either successfully, or an error may have occurred during the SMP packet exchanges; note that this is different from the *gConnEvtKeyExchangeRequest_c* event; the latter signals that the pairing was rejected by the peer, while the former is used for failures due to the SMP packet exchanges.
  - *gConnEvtLeScKeypressNotification_c*: the stack informs the application that a remote SMP Keypress Notification has been received during Passkey Entry Pairing Method.

After the link encryption or pairing is completed successfully, the Central may immediately start exchanging data using the GATT APIs. Gap_RejectPairing may be called on any pairing event.

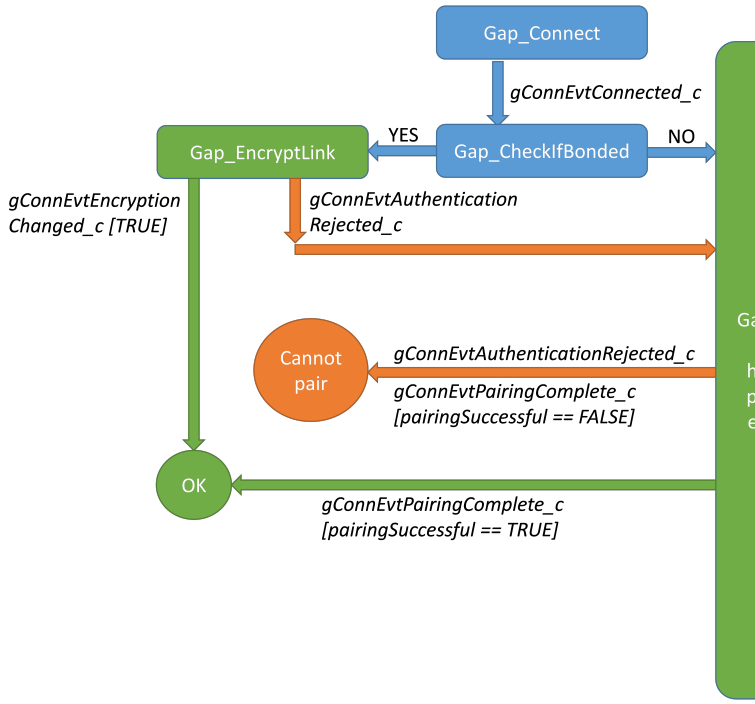


Figure 4. Central pairing flow – APIs and events.

Parent topic: Central setup

Parent topic:*Generic Access Profile (GAP) Layer*

LE data packet length extension

This new feature extends the maximum data channel payload length from 27 to 251 octets.

The length management is done automatically by the link layer immediately after the connection is established. The stack passes the default values for maximum transmission number of payload octets and maximum packet transmission time that the application configures at compilation time in *ble_config.h*:

```
#ifndef gBleDefaultTxOctets_c
```

(continues on next page)

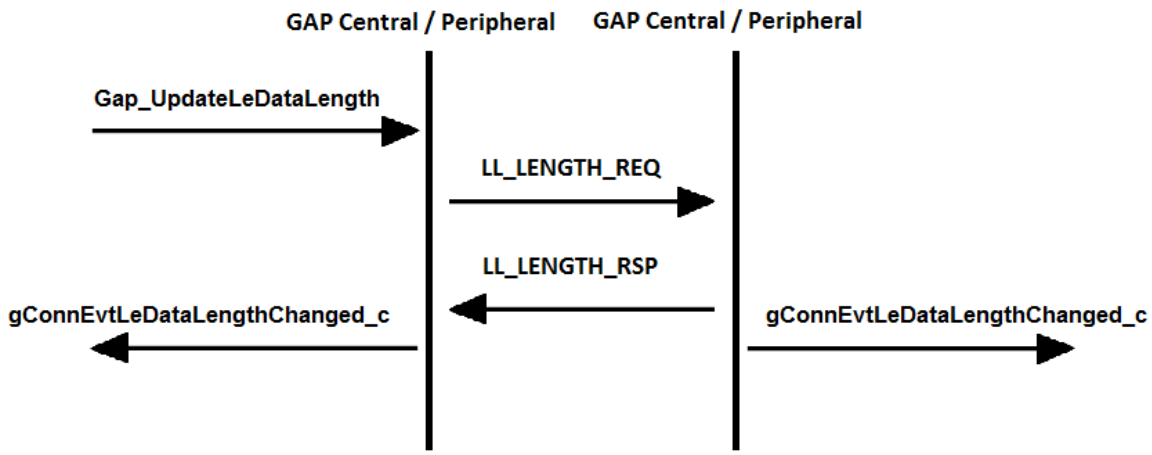
(continued from previous page)

```
#define gBleDefaultTxOctets_c        0x00FB
#endif

#ifndef gBleDefaultTxTime_c
#define gBleDefaultTxTime_c          0x0848
#endif
```

The device can update the data length anytime, while in connection. The function that triggers this mechanism is the following:

```
bleResult_t Gap_UpdateLeDataLength
(
    deviceId_t          deviceId,
    uint16_t     txOctets,
    uint16_t     txTime
);
```

After the procedure executes, a *gConnEvtLeDataLengthChanged_c* connection event is triggered with the maximum values for number of payload octets and time to transmit and receive a link layer data channel PDU. The event is send event if the remote device initiates the procedure. This procedure is shown in Figure. **Figure 5. Data Length Update Procedure**



Parent topic:*Generic Access Profile (GAP) Layer*

Privacy feature

Introduction

Starting with Bluetooth 4.2, Privacy can be enabled either in the Host or in the Controller:

- **Host Privacy** consists of two use cases that are described in detail in the following sections. These are:
  - Random address generation - Periodically regenerating a random address (Resolvable or Non-Resolvable Private Address) inside the Host and then applying it into the Controller.
  - Random address resolution - Trying to resolve incoming RPAs using the IRKs stored in the bonded devices list. The address resolution is performed when a connection is established with a device or for the autoconnect scan. The advertising packets that have an RPA are not resolved automatically due to the high MCU processing that is required.

The random address resolution is performed by default by the Host whenever the Controller is not able to resolve an RPA. The Host performs random address generation only when Host Privacy is requested to be enabled. During random address generation, the advertising and scan operations, if active, are stopped and restarted. If errors occur during this process and the scan or advertising cannot be started, the application is notified through the corresponding event (gAdvertisingStateChanged_c, gExtAdvertisingStateChanged_c or gScanStateChanged_c)

- **Controller Privacy**, introduced by Bluetooth 4.2, consists of writing the local IRK in the Controller, together with all known peer IRKs, and letting the Controller perform hardware, fully automatic RPA generation and resolution. The Controller uses a Resolving List to store these entries. The size of the list is platform dependent and determined by gMaxResolvingListSize_c. For RPA resolution, the entries that do not fit in this list are processed by the Host to be resolved using the IRKs from Bonded Devices list.

Host Privacy can be enabled at any time. Controller Privacy should only be enabled while the device is not in the advertising, scanning or connection initiating state - otherwise the HCI LE Set Address Resolution Enable command will fail and privacy will not be considered as successfully enabled. Trying to enable one while the other is in progress generates a *gBleInvalidState_c* error. The same error is returned when trying to enable the same privacy type twice, or when trying to disable privacy when it is not enabled.

The recommended way of using Privacy is the Controller Privacy. However, enabling Controller Privacy requires at least a pair of local IRK and peer IRK, so this can only be enabled only after a pairing is performed with a peer and the IRKs are exchanged during the Key Distribution phase. When a device starts, if Privacy is required, the workflow is the following:

1. Enable Host Privacy using the local IRK.
2. Connect to a peer and perform pairing and bonding to exchange IRKs.
3. Disable Host Privacy.
4. Enable Controller Privacy using the local IRK and the peer IRK and peer identity address.

After enabling Host Privacy or Controller Privacy, the application must wait for the gHostPrivacyStateChanged_c or gControllerPrivacyStateChanged_c generic event and verify that privacy has been successfully enabled. Only then it is safe to proceed with setting advertising parameters (via the Gap_SetAdvertisingParameters or Gap_SetExtAdvertisingParameters APIs) or starting scanning (via the Gap_StartScanning API). Failure to do so could result in unwanted behavior, such as the device advertising or scanning with a public address.

Resolvable private addresses

A Resolvable Private Address (RPA) is a random address generated using an Identity Resolving Key (IRK). This address appears completely random to an outside observer, so a device may periodically regenerate its RPA to maintain privacy, as there is no correlation between any two different RPAs generated using the same IRK.

On the other hand, an IRK can also be used to resolve an RPA, in other words, to check if this RPA has been generated with this IRK. This process is called “resolving the identity of a device”. Whoever has the IRK of a device can always try to resolve its identity against an RPA.

For example, assume device A frequently changes its RPA using IRKA. At some point, A bonds with B. A must give B a way to recognize it in a subsequent connection when it (A) has a different address. To achieve this purpose, A distributes the IRKA during the Key Distribution phase of the pairing process. B stores the IRKA it received from A.

Later, B connects to a device X that uses RPAX. This address appears completely random, but B can try to resolve RPAX using IRKA. If the resolving operation is successful, it means that IRKA was used to generate RPAX, and since IRKA belongs to device A, it means that X is A. So B was able to recognize the identity of device X, but nobody else can do that since they do not have IRKA.

Parent topic: Introduction

Non-resolvable private addresses

A Non-Resolvable Private Address (NRPA) is a completely random address that has no generation pattern and therefore cannot be resolved by a peer.

A device that uses an NRPA that is changed frequently is impossible to track because each new address appears to belong to a new device.

Parent topic: Introduction

Multiple identity resolving keys

If a device bonds with multiple peers, all of which are using RPAs, it needs to store the IRK of each in order to be able to recognize them later (see previous section).

This means that whenever the device connects to a peer that uses an unknown RPA, it needs to try and resolve the RPA with each of the stored IRKs. If the number of IRKs is large, then this introduces a lot of computation.

Performing all these resolving operations in the Host can be costly. It is much more efficient to take advantage of hardware acceleration and enable the Controller Privacy.

Parent topic: Introduction

Parent topic: Privacy feature

Host privacy

To enable or disable Host Privacy, the following API may be used:

```
bleResult_t Gap_EnableHostPrivacy
(
    bool_t              enable,
    const uint8_t *     aIrk
);
```

When *enable* is set to TRUE, the *aIrk*parameter defines which type of Private Address to generate. If *aIrk* is NULL, then a new NRPA is generated periodically and written into the Controller. Otherwise, an IRK is copied internally from the *aIrk* address and it is used to periodically generate a new RPA.

The lifetime of the Private Address (NRPA or RPA) is a number of seconds contained by the *gGapHostPrivacyTimeout*external constant, which is defined in the *ble_config.c* source file. The default value for this is 900 (15 minutes).

When Host Privacy is enabled, the Host ignores the ownAddressType value for the advertising, scanning or connect parameters. It will always use the random address type in order to use the RPA configured in the Controller in the packets sent over the air.

As mentioned in the Introduction section, call this API for random address generation. For random address resolution there is no need to do so, it is performed by default against the bonded devices list.

Parent topic: Privacy feature

Controller privacy

To enable or disable Controller Privacy, the following API may be used:

```
bleResult_t Gap_EnableControllerPrivacy
(
    bool_t                          enable,
    const uint8_t *                   aOwnIrk,
    uint8_t                         peerIdCount,
    const gapIdentityInformation_t*      aPeerIdentities
);
```

When *enable* is set to TRUE, *aOwnIrk*parameter shall not be NULL, *peerIdCount* shall not be zero or greater than *gMaxResolvingListSize_c*, and *aPeerIdentities* shall not be NULL.

The IRK defined by *aOwnIrk* is used by the Controller to periodically generate a new Resolvable Private Address (RPA). The lifetime of the RPA is a number of seconds contained by the *gGapControllerPrivacyTimeout* external constant, which is defined in the *ble_globals.c* source file. The default value for this is 900 (15 minutes). The *gGapControllerPrivacyTimeoutMin* and *gGapControllerPrivacyTimeoutMax* external constants, also defined in *ble_globals.c*, contain the minimum and maximum values for the RPA lifetime if the Controller supports the Randomized RPA Timeout feature. The default value for both is 900 - same behavior as the non-randomized timeout.

The *aPeerIdentities* is an array of identity information for each bonded device. The identity information contains the device's identity address (public or random static address) and the device's IRK. This array can be obtained from the Host with the *Gap_GetBondedDevicesIdentityInformation*API.

Enabling Controller Privacy involves a quick sequence of commands to the Controller. When the sequence is complete, the *gControllerPrivacyStateChanged_c* generic event is triggered.

Privacy mode

In Bluetooth LE 5.0, the privacy mode has been introduced as an optional feature and is part of the GAP identity structure together with the address and address type. There are two modes: Network Privacy Mode (default) and Device Privacy Mode. These are valid only for Controller Privacy.

A device in network privacy mode only accepts packets from peers using private addresses.

A device in device privacy mode also accepts packets from peers using identity addresses, even if the peer had previously distributed the IRK. Private addresses are also accepted.

The privacy mode of a device is stored in NVM together with the IRK with a default value of Network. If the application wants to change this value it can extract the peer identities, modify the privacy mode from network to device and then enable Controller Privacy with the value.

To change the privacy mode of a device and make the change persistent, the user must call the following API:

```
bleResult_t Gap_SetPrivacyMode
(
    uint8_t nvmIndex,
    blePrivacyMode_t privacyMode
);
```

Parent topic: Controller privacy

Scanning and initiating

When a Central device is scanning while Controller Privacy is enabled, the Controller actively tries to resolve any RPA contained in the Advertising Address field of advertising packets. If any match is found against the peer IRK list, then the *advertisingAddressResolved* parameter from the scanned device structure is set to TRUE.

In this case, the *addressType* and *aAddress* fields no longer contain the actual Advertising Address as seen over the air, but instead they contain the identity address of the device whose

IRK was able to resolve the Advertising Address. In order to connect to this device, these fields shall be used to complete the *peerAddressType* and *peerAddress* fields of the connection request parameter structure, and the *usePeerIdentityAddress* field shall be set to TRUE.

If *advertisingAddressResolved* is equal to FALSE, then the advertiser is using a Public or Random Static Address, an NRPA, or a RPA that could not be resolved. Therefore, the connection to this device is initiated as if Controller Privacy was not enabled, by setting *usePeerIdentityAddress* to FALSE.

Parent topic: Controller privacy

Advertising

When a Peripheral starts advertising while Controller Privacy is enabled, the *ownAddressType* field of the advertising parameter structure is unused. Instead, the Controller always generates an RPA and advertises with it as Advertising Address.

If directed advertising is used, the Host only allows advertising to a device in the resolving list in order to be able to generate RPAs.

Parent topic: Controller privacy

Connected

When a device connects while Controller Privacy is enabled, the *gConnEvtConnected_c* connection event parameter structure contains more relevant fields than without Controller Privacy.

The *peerRpaResolved* field equals TRUE if the peer was using an RPA that was resolved using an IRK from the list. In that case, the *peerAddressType* and *peerAddress*fields contain the identity address of the resolved device, and the actual RPA used to create the connection (the RPA that a Central used when initiating the connection, or the RPA that the Peripheral advertised with) is contained by the *peerRpa* field.

The *localRpaUsed*field equals TRUE if the local Controller was automatically generating an RPA when the connection was created, and the actual RPA is contained by the *localRpa* field.

Parent topic: Controller privacy

Parent topic: Privacy feature

Parent topic:*Generic Access Profile (GAP) Layer*

Setting PHY mode in a connection

In Bluetooth LE 5.0, the user is able to change the PHY mode in a connection through the Link Layer PHY Update Procedure and choose between default 1 Mbit/s, 2 Mbit/s high data rate or the coded S2 or S8 PHYs with 500 Kbps or 125 Kbps for longer range.

To set the PHY, the user can call:

```
bleResult_t Gap_LeSetPhy
(
    bool_t        defaultMode,
    deviceId_t    deviceId,
    uint8_t       allPhys,
    uint8_t       txPhys,
    uint8_t       rxPhys,
    uint16_t      phyOptions
);
```

There are two modes to use this API:

1. If defaultMode is set to TRUE, the user can call this function without being in a connection, i.e. provide a device ID. The PHY option is used by the Link Layer in the PHY response when a connection is created and the peer device initiates the PHY Update Procedure. The

application should listen for gLePhyEvent_c with the gPhySetDefaultComplete_c sub event type for the confirmation of the operation.

2. If defaultMode is set to FALSE, the user must also provide a valid device ID. The Host asks the Link Layer to initiate the PHY Update Procedure with the peer device using the provided parameters.

The application should listen for gLePhyEvent_c with the gPhyUpdateComplete_c sub event type for the confirmation of the update procedure to have ended. The result of the operation populates in the txPhy and rxPhy of the event. The result is from the negotiation of the local parameters and the peer PHY preferences.

To read the current PHY on a connection, call the following API:

```
bleResult_t Gap_LeReadPhy
(
    deviceId_t deviceId
);
```

The application should listen for gLePhyEvent_c with the gPhyRead_c sub event type for the confirmation of the operation. The txPhy and rxPhy indicate the current modes used in the connection.

Parent topic:*Generic Access Profile (GAP) Layer*

Data management of bonded devices

The Host handles the management of the bonding data without requiring application intervention. The application must provide the NVM write, read, and erase functions presented in Non-Volatile Memory (NVM) access. The Host creates bonds if bonding is required after the pairing.

The bonded data structure is presented below, together with the GAP APIs that access it, for most APIs require a connection to be established with the device in the bonded list, the others can be accessed any time using the NVM index.

1. Bond Header – identity address and address type that uniquely identify a device together with the IRK and privacy mode.
   - Gap_GetBondedDevicesIdentityInformation – for all bonds
2. Bond Data Dynamic - security counters for signed operations – managed by the stack
3. Bond Data Static – LTK, Rand, EDIV, security information for read and write authorizations
   - Gap_SaveKeys– NVM index
   - Gap_LoadKeys– NVM index
   - Gap_LoadEncryptionInformation - deviceId
   - Gap_Authorize – deviceId - GATT Server only
4. Bond Data Legacy - Legacy pair information
   - Gap_LoadEncryptionInformation - deviceId
5. Bond Data Device Info - custom peer information (service discovery data) and device name
   - Gap_SaveCustomPeerInformation - deviceId
   - Gap_LoadCustomPeerInformation - deviceId
   - Gap_SaveDeviceName - deviceId
   - Gap_GetBondedDeviceName – NVM index
6. Bond Data Descriptor List - configuration of indications and notifications for CCCD handles – GATT Server only
   - Gap_CheckNotificationStatus - deviceId

- Gap_CheckIndicationStatus- deviceId

However, there may be some cases when an application wants to manage this data to read data from a bonded device created by the Host, create a bond obtained out-of-band or update an existing bond. For this use case, two GAP APIs and a GAP event have been added.

1. **Load the Keys of a bonded device**.

   The user can call the following function to read the keys exchanged during pairing and stored by the Bluetooth LE Host Stack in the bond area when the pairing is complete.

   The application is informed of the NVM index through the gBondCreatedEvent_c sent by the stack immediately after the bond creation. The application is responsible for passing the memory in the pOutKeys OUT parameter to fill in the keys, if any of the keys are set to NULL, the stack does not fill that information. The pOutKeyFlags OUT parameter indicates to the application which of the keys were stored by the stack as not all of them may have been distributed during pairing.

   The pOutLeSc indicates if Bluetooth LE 4.2 LE Secure Connections Pairing was used, while the pOutAuth indicates if the peer device is authenticated for MITM protection. All these OUT parameters are recommended to be retrieved from the bond and added if later passed as input parameters for the save keys API.

   This function executes synchronously.

```
bleResult_t Gap_LoadKeys
(
    uint8_t             nvmIndex,
    gapSmpKeys_t*       pOutKeys,
    gapSmpKeyFlags_t*   pOutKeyFlags,
    bool_t*             pOutLeSc,
    bool_t*             pOutAuth
);
```

   The gapSmpKeys_t is the structure used during the key distribution phase, as well as in the gConnEvtKeysReceived_c event and is as follows. The difference is that the Bluetooth LE device address cannot be set to NULL neither when loading a bond or when creating one as it identifies the bonded device together with the NVM index.
   Table 3. 'gapSmpKeys_t' structure

Event Data	Data type	Data Description
cLtkSize	uint8_t	Encryption Key Size filled by the stack. If aLtk is NULL, this is ignored.In Advanced Secure Mode, this should be the size of the LTK encrypted blob of 40 bytes.
aLtk	uint8_t*	Long Term (Encryption) Key or LTK encrypted blob if Advanced Secure Mode is enabled. NULL if LTK isnot distributed, else size is given by cLtkSize
aIrk	uint8_t*	Identity Resolving Key. NULL if aIrk is not distributed.
cRandSize	uint8_t	Size of RAND filled by the stack; usually equal to gcSmpMaxRandSize_c. IfaLtk is NULL, this is ignored.
aRand	uint8_t*	RAND value used to identify the LTK. If aLtk is NULL, this is ignored.
ediv	uint16_t	EDIV value used to identify the LTK. If aLtk is NULL, this is ignored.
addressTy	bleAddressType_t	Public or Random address.
aAddress	uint8_t*	Device Address. It cannot be NULL.

The structure for the GAP SMP Key Flags is the following:

Table 4. GAP SMP Key Flags

Flag Type	Description
gNoKeys_c	No key is available.
gLtk_c	Long-Term Key is available.
gIrk_c	Identity Resolving Key is available.

2. Save the Keys to create a bond or update an existing bonded device.

   The user can call the following function to create a bond on a device based on information obtained Out of Band. For instance, one can use the output of Gap_LoadKeys from the previous section. This can be useful in transferring a bond created by the stack after a pairing procedure or if the application wants to manipulate bonding data. The behavior of the stack remains the same, if the bonding is required after a pairing, the stack stores the bonding information if possible. In this case, the NVM index is passed to the application through gBondCreatedEvent_c.

   This function executes asynchronously, as the stack can create a bond during the execution. The application should listen for the previous mentioned event gBondCreatedEvent_c. The result of the function call is passed synchronously. However, if an asynchronous error has occurred during the actual save, it is passed to the application through the gInternalError_c event with a gSaveKeys_c error source.

   The stack creates a bond if the NVM index is free or update the keys from an NVM index if it stores a valid entry.

   The address from the GAP SMP Keys structure must not be NULL. If other members of the structure are NULL, they are ignored.

   LE SC flag indicates if Bluetooth LE 4.2 Secure Connections was used during pairing and Auth specifies if the peer is authenticated for MITM protection.

```
bleResult_t Gap_SaveKeys
(
    uint8_t                 nvmIndex,
    const gapSmpKeys_t*     pKeys,
    bool_t                  leSc,
    bool_t                  auth
);
```

3. Bond created event.

   A GAP event is added to the Bluetooth LE Generic Callback to inform the application of the NVM index whenever the stack creates a bond or when a Gap_SaveKeys request succeeds. The event is also generated if the NVM index was a valid occupied entry and only some of the keys in the bonded information have been updated.

   The NVM index is then used in the GAP APIs to save or load information from the bond.

Event Data	Data type	Data Description
nvmIndex	uint8_t	NVM index for the new created bond
addressType	bleAddressType_t	Public or Random (static) address of the bond
address	bleDeviceAddress_t	Address of the bond

Application removal of bonded devices data

The application can remove a bonded device from NVM. The bonded device cannot be deleted if it is in an active connection. The application can remove one or all bonds by calling the following synchronous GAP APIs:

- Gap_RemoveBond(uint8_t nvmIndex) – nvmIndex can be obtained via the Gap_CheckIfBonded API.
- Gap_RemoveAllBonds() - no connections should be active otherwise the call fails.

Removing a bonded device does not affect the controller address resolution state nor the contents of either the Controller Filter Accept List or the Controller Resolving List. If Controller Privacy is enabled, it remains so until it is disabled or the device is reset.

In a scenario where the user wants to remove a bonded device and all its effects on device behavior (Controller Filter Accept List, Controller Resolving List), the following operations should be executed:

- Gap_ClearFilterAcceptList or Gap_RemoveDeviceFromFilterAcceptList
  - Clear Controller Filter Accept List or clear a device from Filter Accept List.
- Gap_RemoveAllBonds or Gap_RemoveBond
  - All bonded devices are removed or one bonded device is removed from NVM.
- BleConnManager_DisablePrivacy
  - Controller Privacy is disabled, Controller Resolving List is cleared and address resolution is disabled. The device should not be advertising or scanning, otherwise this call fails.
- BleConnManager_EnablePrivacy
  - Called after the gControllerPrivacyStateChanged_c event is received, confirming Controller Privacy has been disabled. If not all bonds have been deleted, Controller Privacy is reenabled. In the absence of bonds, Host Privacy is enabled.

Parent topic: Data management of bonded devices

Parent topic:*Generic Access Profile (GAP) Layer*

Controller enhanced notifications

This section describes how the application can configure and monitor the notifications generated by the Bluetooth Controller when advertising, scan, or connection events occur. This feature is proprietary to NXP that is available on selected Controllers.

The user can choose between two options:

1. Enable notifications from the GAP layer and monitor GAP events in the GAP Generic Callback. The controller issues HCI vendor-specific events processed by the Bluetooth LE Host and presented to the application in the GAP Generic Callback.
2. Enable notifications from the Controller interface and monitor controller events in a user-defined Application Callback.
3. Combination of the above two options: configure feature at GAP layer and install an Application Callback through the Controller interface. After setting the callback, the HCI vendor-specific events are not issued and implicitly the GAP events. Instead, the user receives the notifications in the installed callback until setting the callback to NULL again if it wants to revert to GAP events.

- **GAP configuration:**

  The user should call the following function to enable various events from the mask or use Event None to disable the feature. The Device ID is valid only for connection events.

```
bleResult_t Gap_ControllerEnhancedNotification
(    uint16_t eventType,
    deviceId_t deviceId
);
```

The event type is a bitmask having the following options:
Table 5. Event types and their description

Event Type	Event Description
gNotifEventNone_c	No enhanced notification event enabled
gNotifConnEventOver_c	Connection event over
gNotifConnRxPdu_c	Connection RX PDU
gNotifAdvEventOver_c	Advertising event over
gNotifAdvTx_c	Advertising ADV transmitted
gNotifAdvScanReqRx_c	Advertising SCAN REQ RX
gNotifAdvConnReqRx_c	Advertising CONN REQ RX
gNotifScanEventOver_c	Scanning event over
gNotifScanAdvPktRx_c	Scanning ADV PKT RX
gNotifScanRspRx_c	Scanning SCAN RSP RX
gNotifScanReqTx_c	Scanning SCAN REQ TX
gNotifConnCreated_c	Connection created
gNotifChannelMatrix_c	Enable channel status monitoring
gNotifPhyReq_c	Phy Req Pdu ack received
gNotifConnChannelMapUpdate_c	Channel map update
gNotifConnInd_c	Connect indication
gNotifPhyUpdateInd_c	Phy update indication

After enabling events, the user should wait for a gControllerNotificationEvent_c GAP Generic Event in the GAP Generic Callback. The first event received should have the event type set to gNotifEventNone_c with a status of success confirming the selected event mask has been enabled. The same event types apply for both the GAP command and the GAP event. The structure for the Controller Notification event is the following:
Table 6. Controller Notification Event structure

Event Data	Data type	Data Description
eventType	bleNotificationEvent_t	Enhanced notification event type
deviceId	deviceId_t	Device id of the peer, valid for connection events
rssi	int8_t	RSSI, valid for RX event types
channel	uint8_t	Channel, valid for connection event over or Rx/Tx events
ce_counte	uint16_t	Connection event counter, valid for connection events only
status	bleResult_t	Status of the request to select which events to be enabled/disabled
timestamp	uint16_t	Timestamp in 625 µs slots, valid for Conn RX event and Conn Created event
adv_hand	uint8_t	Advertising Handle, valid for advertising events, if multiple ADV sets supported

- **Controller configuration:**

  The user should call the following function to enable various events from the mask or use Event None to disable the feature. The same event types apply as the GAP layer types. The connection handle is valid only for connection events.

```
bleResult_t Controller_ConfigureEnhancedNotification
(
    uint16_t eventType,
    uint16_t conn_handle
);
```

The event monitoring is done in a user-installed callback by calling:

```
bleResult_t Controller_RegisterEnhancedEventCallback
(
    bleCtrlNotificationCallback_t notificationCallback
);
```

Where the types are the following:

```
typedef struct bleCtrlNotificationEvent_tag
{
    uint16_t event_type; /*! bleNotificationEventType_t */
    uint16_t conn_handle;
    uint8_t  rssi;
    uint8_t  channel_index;
    uint16_t conn_ev_counter;
    uint16_t timestamp;
    uint8_t  adv_handle;
} bleCtrlNotificationEvent_t;
typedef void (*bleCtrlNotificationCallback_t)
(
    bleCtrlNotificationEvent_t    *pNotificationEvent
);
```

The event structure is nearly identical as the GAP one, except there is no status as the function call executes synchronously.

Table 7. 'bleCtrlNotificationEvent_tag' Event structure

Event Data	Data type	Data Description
event_type	bleNotificationEvent_t	Enhanced notification event type
conn_handle	uint16_t	Connection handle of the peer, valid for connection events
rssi	int8_t	RSSI, valid for RX event types
channel_index	uint8_t	Channel, valid for connection event over or Rx/Tx events
conn_ev_cou	uint16_t	Connection event counter, valid for connection events only
timestamp	uint16_t	Timestamp in 625 µs slots, valid for Conn RX event and Conn Created event
adv_handle	uint8_t	Advertising Handle, valid for advertising events, if multiple ADV sets supported

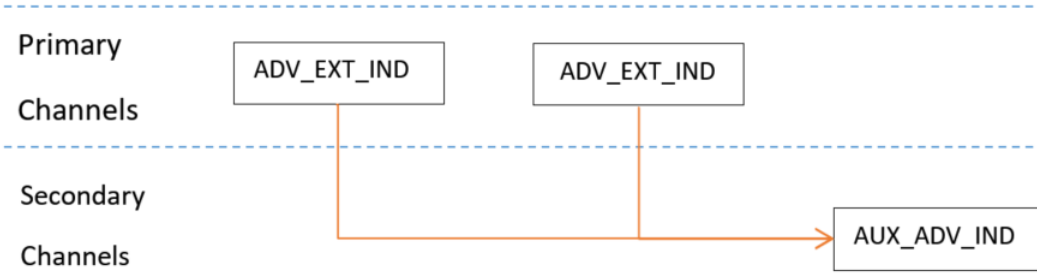
Parent topic:*Generic Access Profile (GAP) Layer*

Extended advertising

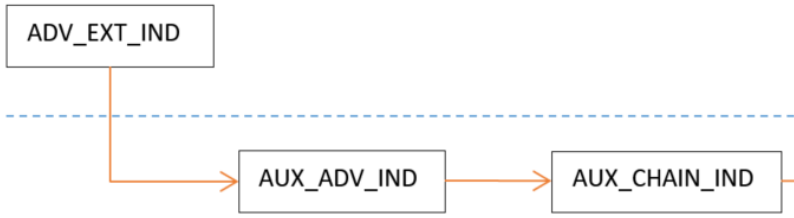
Starting with Bluetooth 5, the advertising channels are separated in primary advertising channels and secondary advertising channels:

1. Primary advertising channels
    - Use 3 legacy advertising channels 37, 38, and 39.
    - Can use either legacy 1M PHY or new LE Coded PHY.
    - PHY payload can vary from 6 to 37 bytes.
    - Packets on these channels are part of the advertising events.
2. Secondary advertising channels

- Use 37 channels, with the same channel index as the data channels.
- Can use any LE PHY, but the same PHY during an Extended Advertising Event.
- PHY payload can vary from 0 to 255 bytes.
- Auxiliary packets on these channels are part of the Extended Advertising Event that begins at the same time with the advertising event on primary channel and ends with the last packet on the secondary channel.



Extended advertising



Extended advertising – Multiple chains

An advertising data set is represented by advertising PDUs belonging together in an advertising event. Each set has different advertising parameters: PDU type, advertising interval, and PHY mode. The advertising data sets are identified by the Advertising SID (Set ID) field from the ADI – Advertising Data Info. Advertising data or Scan response data can be changed for each adverting data set and the random value of DID (Data ID) field is updated to differentiate between them.

Refer to Figure 1 and Figure 2.

Peripheral setup

This section describes the extended advertising GAP API. The application should not use both the extended and legacy API Refer to Scanning. If this requirement cannot be met, the application should at least wait for the generated events in the Advertising Callback prior to using the other API. That is, it is advisable to call legacy functions only after the event pertaining to an extended API is received, and vice versa. This GAP constraint can be considered an extension of the HCI constraint from the Bluetooth 5 specification: "A Host should not issue legacy commands to a Controller that supports the LE Feature (Extended Advertising)".

The application configures extended advertising by going through the following states:

1. Set the extended advertising parameters by calling:

```
bleResult_t Gap_SetExtAdvertisingParameters
(
gapExtAdvertisingParameters_t* pAdvertisingParameters
);
```

   It may use the default set of parameters *gGapDefaultExtAdvertisingParameters_d*. The application should wait for a *gExtAdvertisingParametersSetupComplete_c event* in the Generic Callback. Only one advertising set can be configured at a time. Comparing with the legacy *Gap_SetAdvertisingParameters* command, the new set of parameters is as follows.

Parameter	Description
SID	Value of the Advertising SID subfield in the ADI field of the PDU.
handle	Used to identify an advertising set. Possible values are 0x00 or 0x01 since the current implementation supports two advertising sets.
extAdvProperties	BIT0 - Connectable advertising
extAdvProperties	BIT1 - Scannable advertising
extAdvProperties	BIT2 - Directed advertising
extAdvProperties	BIT3 - High Duty Cycle Directed Connectable advertising (□3.75 ms Advertising Interval)
extAdvProperties	BIT4 - Use legacy advertising PDUs
extAdvProperties	BIT5 - Omit advertiser's address from all PDUs ("anonymous advertising")
extAdvProperties	BIT6 - Include TxPower in the extended header of the advertising PDU. If legacy advertising PDU types are being used (BIT4 = 1), permitted properties values are presented in the next table. If the advertising set already contains data, the type shall be one that supports advertising data and the amount of data shall not exceed 31 octets. If extended advertising PDU types are being used (BIT4 =0), then the advertisement shall not be both connectable and scannable. While high duty cycle directed connectable advertising (□ 3.75 ms advertising interval) shall not be used (BIT3 = 0).
txPowe	Maximum power level at which the advertising packets are to be transmitted, the Controller can choose any power level <= txPower. Value 127 to be used if Host has no preference.
primaryPHY	PHY for ADV_EXT_IND: LE 1 M or LE Coded
secondaPHY	PHY for AUX_ADV_IND and periodic advertising: LE 1 M, LE 2 M or LE Coded. Ignored for legacy advertising
secondav-MaxS	Maximum advertising events that the Controller can skip before sending the AUX_ADV_IND packets on the secondary advertising channel. Higher values may result in lower power consumption. Ignored for legacy advertising
enableSca-nReqNotifi-	Whether to enable notifications when scanning PDUs (SCAN_REQ, AUX_SCAN_REQ) are received. If enabled, the application is notified upon scan requests by gExtScanNotification_c events in the Advertising Callback

When using LE Coded PHY for advertising, the default coding scheme chosen by link layer is S=8 (125 kb/s data rate). To change the default coding scheme, the user has two options:

- At compile time by defining *mLongRangeAdvCodingScheme_c*, or
- At run time by calling the API *Controller_ConfigureAdvCodingScheme()*. In both cases, the value of the define or the parameter of the API has to be an appropriate value for primary and secondary PHYs as defined by the enumeration *advCodingScheme_tag* found in *controller_interface.h*. **API Controller_ConfigureAdvCodingScheme()**

EventType	PDU Type	Advertising Event Properties
Connectable and scannable undirected	ADV_IND	00010011b
Connectable directed (low duty cycle)	ADV_DIRECT_IND	00010101b
Connectable directed (high duty cycle)	ADV_DIRECT_IND	00011101b
Scannable undirected	ADV_SCAN_IND	00010010b
Non-connectable and Nonscannable undirected	ADV_NONCONN_INI	00010000b

2. Set the advertising data and/or scan response data by calling:

```
bleResult_t Gap_SetExtAdvertisingData
(
uint8_t handle,
gapAdvertisingData_t* pAdvertisingData,
gapScanResponseData_t* pScanResponseData
);
```

Note: Either of the pAdvertisingData or pScanResponseData parameters can be NULL, but not both. For extended advertising (BIT4 = 0) only one must be different than NULL – the scannable advertising bit (BIT1) indicates whether pAdvertisingData (BIT1 = 0) orpScanResponseData (BIT1 = 1) is accepted. The total amount of Advertising Data shall not exceed 1650 bytes. Application should wait for a *gExtAdvertisingDataSetupComplete_c* event in the Generic Callback.

3. Enable extended advertising by calling:

```
bleResult_t Gap_StartExtAdvertising
(
gapAdvertisingCallback_t advertisingCallback,
gapConnectionCallback_t connectionCallback,
uint8_t handle,
uint16_t duration,
uint8_t maxExtAdvEvents
);
```

When using the common application structure, the application can use the following API defined in *app_conn.h*:

```
bleResult_t BluetoothLEHost_StartExtAdvertising
(
    appExtAdvertisingParams_t *pExtAdvParams,
    gapAdvertisingCallback_t  pfAdvertisingCallback,
    gapConnectionCallback_t   pfConnectionCallback
);
```

The API goes through the steps of setting the advertising data and parameters. Events from the Host task are treated in the *App_AdvertiserHandler()* function, implemented in *app_advertiser.c*. To set the extended advertising parameters and data *BluetoothLEHost_StartExtAdvertising* a parameter of the following type:

```
typedef struct appExtAdvertisingParams_tag
{
    gapExtAdvertisingParameters_t *pGapExtAdvParams;
    gapAdvertisingData_t *pGapAdvData;
    gapScanResponseData_t *pScanResponseData;
    uint8_t                    handle;
    uint16_t                   duration;
    uint8_t                    maxExtAdvEvents;
} appExtAdvertisingParams_t;
```

Advertising may be enabled for each previously configured advertising set, identified by the handle parameter. If duration is set to 0, advertising continues until the Host disables it, otherwise advertising is only enabled for this period (multiple of 10 ms). *maxExtAdvEvents* represent the maximum number of extended advertising events the Controller shall attempt to send prior to terminating the extended advertising, ignored if set to 0. Application should wait for a *gExtAdvertisingStateChanged_c* or a *gAdvertisingCommandFailed_c* event in the Advertising Callback.

4. Disable advertising by calling:

```
bleResult_t Gap_StopExtAdvertising
(
uint8_t handle
);
```

Application should wait for a *gExtAdvertisingStateChanged_c* or a *gAdvertisingCommandFailed_c* event in the Advertising Callback.

5. Remove the advertising set by calling:

```
bleResult_t Gap_RemoveAdvSet
(
uint8_t handle
);
```

Application should wait for a *gExtAdvertisingSetRemoveComplete_c* event in the Generic Callback.

Parent topic: Extended advertising

Central setup

The application configures the extended scanning by going through the following states:

1. Start scanning by calling:

```
bleResult_t Gap_StartScanning
(
const gapScanningParameters_t* pScanningParameters,
gapScanningCallback_t scanningCallback,
gapFilterDuplicates_t enableFilterDuplicates,
uint16_t duration,
uint16_t period
)
```

When using the common application structure, the application can use the following API defined in *app_conn.h*:

```
bleResult_t BluetoothLEHost_StartScanning
(
    appScanningParams_t   *pAppScanParams,
    gapScanningCallback_t pfCallback
);
```

The API starts scanning using the given parameters, which must have the following structure:

```
typedef struct appScanningParams_tag
{
    gapScanningParameters_t *pHostScanParams;          /*!< Pointer to host scan structure */
    gapFilterDuplicates_t enableDuplicateFiltering;     /*!< Duplicate filtering mode */
    uint16_t duration;                                          /*!< scan duration  */
    uint16_t period;                                             /*!< scan period  */
} appScanningParams_t;
```

Application may use the default set of parameters *gGapDefaultExtScanningParameters_d*. If the *pScanningParameters* pointer is NULL, the latest set of parameters are used. The *scanningPHYs* parameter indicates the PHYs on which the advertising packets should be received on the primary advertising channel. As a result, permitted values for the parameter are 0x01 (scan LE 1M), 0x04 (scan LE Coded) and 0x05 (scan both LE 1M and LE Coded). There are no strict timing rules for scanning, yet if both PHYs are enabled for scanning, the scan interval value must be large enough to accommodate two scan windows (interval >= 2 * window).

If the advertiser uses legacy advertising PDUs, the device may actively scan by sending a SCAN_REQ PDU to the advertiser on the LE 1M primary advertising channel (no secondary channel in legacy advertising). Respectively, if the advertiser uses extended advertising PDUs, the active scan operation takes place on the secondary advertising channel. After the device receives a scannable ADV_EXT_IND PDU on the primary advertising channel (PHY LE 1M or Coded), it starts listening for the AUX_ADV_IND PDU on the secondary advertising channel (PHY 1M, 2M or Coded). Once received, the device sends an AUX_SCAN_REQ to the advertiser. Next, an AUX_SCAN_RSP PDU should be received, containing the scan response data. Application should wait for a *gScanStateChanged_c* or a *gScanCommandFailed_c*in the Scanning Callback.

2. Collect information by waiting for *gDeviceScanned_c* (legacy advertising PDUs) or *gExtDeviceScanned_c* (extended advertising PDUs) event in the Scanning Callback. The *gExtDeviceScanned_c* event contains additional information pertaining to the extended received PDU, such as: primary PHY, secondary PHY, advertising SID, interval of the periodic advertising if enabled in the set.

   When using the common application structure, the application can use the following API defined in *app_conn.h*, to search the contents from *pData* in an advertising element:

```
bool_t BluetoothLEHost_MatchDataInAdvElementList
(
    gapAdStructure_t *pElement,
    void             *pData,
    uint8_t           iDataLen
);
```

3. Stop scanning by calling the function below:

```
bleResult_t Gap_StopScanning(void);
```

   Application should wait for a *gScanStateChanged_c* or a *gScanCommandFailed_c* in the Scanning Callback.

4. Connect to a device by calling the function below:

```
bleResult_t Gap_Connect
(
const gapConnectionRequestParameters_t* pParameters,
gapConnectionCallback_t connCallback
);
```

   When using the common application structure, the following API can be used:

```
bleResult_t BluetoothLEHost_Connect
(
    gapConnectionRequestParameters_t*    pParameters,
    gapConnectionCallback_t              connCallback
);
```

The *initiatingPHYs* parameter indicates the PHYs on which the advertising packets should be received on the primary advertising channel and the PHYs for which connection parameters have been specified. The parameter is a bitmask of PHYs: BIT0 = LE 1M, BIT1 = LE 2M and BIT2 = LE Coded. The Host may enable one or more initiating PHYs, but it must at least set one bit for a PHY allowed for scanning on the primary advertising channel, i.e., BIT0 for LE 1M PHY or BIT2 for LE Coded PHY.

If the advertiser uses legacy advertising PDUs, the device may connect by sending a CONNECT_IND PDU to the advertiser on the LE 1M primary advertising channel (no secondary channel in legacy advertising). On the other hand, if the advertiser uses extended advertising PDUs, the extended connect operation takes place on the secondary advertising channel. After the device receives a connectable ADV_EXT_IND PDU on the primary advertising channel (PHY LE 1M or Coded), it starts listening for the connectable AUX_ADV_IND PDU on the secondary advertising channel (PHY 1M, 2M or Coded). Once received, the device sends an AUX_CONNECT_REQ to the advertiser. Next, if AUX_CONNECT_RSP PDU is received, the device enters the Connection State in the Central role on the secondary advertising channel PHY.

Application should wait for a *gConnEvtConnected_c* event in the Connection Callback. If the channel selection algorithm #2 is used for this connection, then a *gConnEvtChanSelectionAlgorithm2_c* event is also generated.

After the connection is successfully established, the application may choose to read the connection PHY by calling the *Gap_LeReadPhy* API. It may also opt to change the PHY of the connection by triggering a PHY Update Procedure using the *Gap_LeSetPhy API*. However, the Controller might not be able to perform the change if, in case the peer does not support the new requested PHY.

Parent topic: Extended advertising

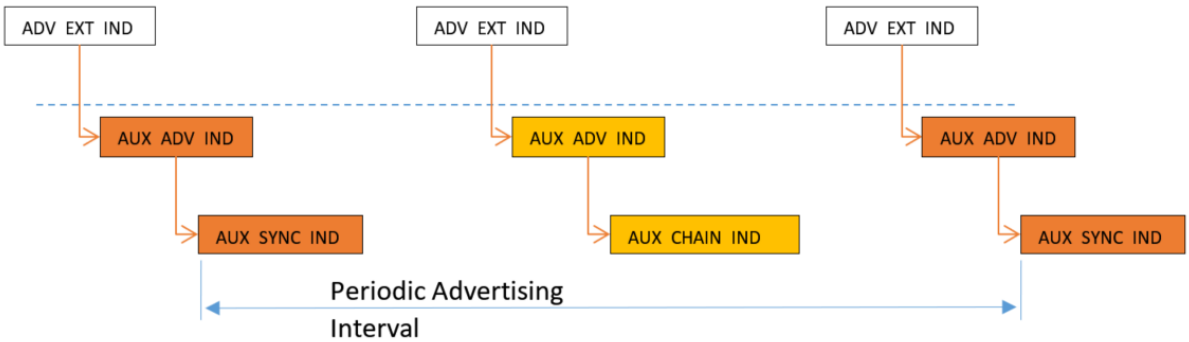
Parent topic:*Generic Access Profile (GAP) Layer*

Periodic Advertising

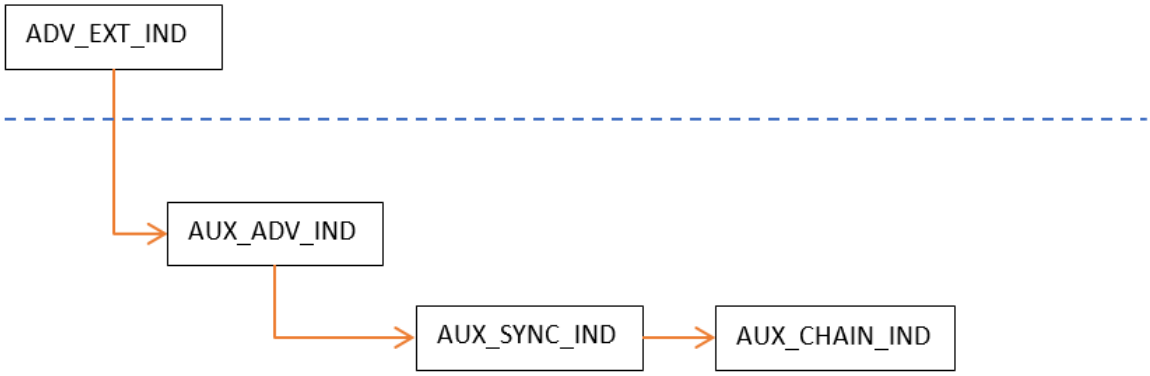
Periodic channels are used for periodic broadcast between unconnected devices. A periodic channel is represented by a channel map and a set of hopping and timing parameters.

The set of channels is represented by the 37 data channels. A packet sent by an advertiser can also have a payload of up to 255 bytes and it can be sent on any LE PHY. Figure 1 and Figure 2.

Extended Advertising and Periodic Advertising combined



Periodic Advertising – Multiple chains



Peripheral Setup

1. First set the extended advertising parameters using Gap_SetExtAdvertisingParameters. The extended advertising type must be set to non-connectable and non-scannable.
2. Set the periodic advertising parameters using the same handle as in the previous command.

```
bleResult_t Gap_SetPeriodicAdvParameters
(
    gapPeriodicAdvParameters_t*     pAdvertisingParameters
);
```

   Wait for a *gPeriodicAdvParamSetupComplete_c*event in the generic callback.

3. Next, set the periodic advertising data by calling:

```
bleResult_t Gap_SetPeriodicAdvertisingData
(
    uint8_t                     handle,
    gapAdvertisingData_t*   pAdvertisingData,
    bool_t bUpdateDID
);
```

   pAdvertisingData cannot be NULL. If periodic advertising data must be empty, set *cNumAdStructures* to 0. Wait for a *gPeriodicAdvDataSetupComplete_c*event in the generic callback.

4. Start extended advertising using Gap_StartExtAdvertising.
5. Last, enable Periodic Advertising. Periodic advertising starts only after extended advertising is started.

```
bleResult_t Gap_StartPeriodicAdvertising
(
    uint8_t handle,
    bool_t  bIncludeADI
);
```

   Wait for a *gPeriodicAdvertisingStateChanged_c*event in the advertising callback.

Parent topic: Periodic Advertising

Central Setup

The application may decide to listen to periodic advertising by going through the following states:

1. [Optional] Add a known periodic advertiser to the periodic advertiser list held in the Controller by calling:

```
bleResult_t Gap_UpdatePeriodicAdvList
(
    gapPeriodicAdvListOperation_t operation,
    bleAddressType_t              addrType,
    uint8_t*                      pAddr,
    uint8_t                       SID
);
```

Wait for the *gPeriodicAdvListUpdateComplete_c* event in the Generic Callback.

2. Synchronize with a periodic advertiser by calling:

```
bleResult_t Gap_PeriodicAdvCreateSync
(
    gapPeriodicAdvSyncReq_t*   pReq,
);
```

pReq parameter *filterPolicy*can be set to *gUseCommandParameters_c* to synchronize with the given peer, or to *gUsePeriodicAdvList_c* to start synchronizing with all the devices in the previously populated periodic advertiser list.

Wait for the *gPeriodicAdvSyncEstablished_c* event and check the status. If scanning is not enabled at the time this command is sent, synchronization occurs after scanning is started. Synchronization remains pending until *gPeriodicAdvSyncEstablished_c*event is received. If synchronization was successful, the *syncHandle* is returned in this event.

3. Terminate the synchronization with the periodic advertiser by calling:

```
bleResult_t Gap_PeriodicAdvTerminateSync
(
    uint16_t syncHandle
);
```

To cancel a pending synchronization, the application should call *Gap_PeriodicAdvTerminateSync* with *syncHandle* set to the reserved value *gBlePeriodicAdvOngoingSyncCancelHandle* and wait for *gPeriodicAdvCreateSyncCancelled_c* event.

Otherwise, to terminate an already established sync with an advertiser, use the *syncHandle* value from the *gPeriodicAdvSyncEstablished_c* event and wait for a *gPeriodicAdvSyncTerminated_c* event.

Parent topic: Periodic Advertising

Parent topic:*Generic Access Profile (GAP) Layer*

Periodic Advertising with Responses (PAwR)

This section describes the Central and Peripheral setup for Periodic Advertising with Responses (PAwR).

Central Setup

1. Start scanning using Gap_StartScanning. Wait for gPeriodicDeviceScannedV2_c events in the scanning callback.
2. Synchronize with a periodic advertiser by calling Gap_PeriodicAdvCreateSync. Wait for the gPeriodicAdvSyncEstablished_c event in the scanning callback. When PAwR is involved, this event includes additional information such as number of subevents, subevent interval, response slot delay and spacing,
3. Synchronize to a PAwR subevent by calling Gap_SetPeriodicSyncSubevent. This API instructs the Controller to sync with a subset of the subevents within a PAwR train identified by syncHandle (obtained after synchronizing with the PAwR train in the previous step).

```
bleResult_t Gap_SetPeriodicSyncSubevent ( uint16_t syncHandle, const␣
↪gapPeriodicSyncSubeventParameters_t* pParams );
```

Wait for the gPeriodicSyncSubeventComplete_c event.

4. Use Gap_SetPeriodicAdvResponseData to set data in the AD format which would be sent as a Periodic Advertising Response to the broadcaster.

```
bleResult_t Gap_SetPeriodicAdvResponseData ( uint16_t syncHandle, const␣
↪gapPeriodicAdvertisingResponseData_t* pData );
```

5. Optionally, the periodic advertiser may initiate a connection. If no connection callback was set on the scanner via APIs such as Gap_Connect or Gap_StartAdvertising/Gap_StartExtAdvertising, one must be explicitly set. This is achieved by calling BluetoothLEHost_SetConnectionCallback (defined in app_conn.h), which in turn calls Gap_SetConnectionCallback.

```
void Gap_SetConnectionCallback ( gapConnectionCallback_t pfConnectionCallback );
```

Parent topic: Periodic Advertising with Responses (PAwR)

Peripheral Setup

1. First set the extended advertising parameters using Gap_SetExtAdvertisingParameters. The extended advertising type must be set to non-connectable and non-scannable.
2. Set the periodic advertising parameters with the same handle as in the previous command. Use the Gap_SetPeriodicAdvParametersV2 command. Compared to Gap_SetPeriodicAdvParameters, this command also configures parameters relevant to PAwR, such as the number of subevents and response slots as well as timing information.

```
bleResult_t Gap_SetPeriodicAdvParametersV2
  (gapPeriodicAdvParametersV2_t* pAdvertisingParameters);
```

Wait for a gPeriodicAdvParamSetupComplete_c event in the generic callback.

3. Start extended advertising using Gap_StartExtAdvertising.
4. Start periodic advertising using Gap_StartPeriodicAdvertising.
5. Wait for gPerAdvSubeventDataRequest_c events. These events are used by the Controller to indicate that it is ready to transmit one or more subevents and it is requesting the advertising data for these subevents. Upon receiving an event, use Gap_SetPeriodicAdvSubeventData to set the advertising data for specific subevents.

```
bleResult_t Gap_SetPeriodicAdvSubeventData
    (uint8_t advHandle, const gapPeriodicAdvertisingSubeventData_t* pData);
```

Wait for the gPeriodicAdvSetSubeventDataComplete_c event in the generic callback.

6. Wait for gPerAdvResponse_c events. These events contain responses sent by devices who are synchronized to the periodic advertising. They include data in the AD format.
7. Optionally, PAwR allows the advertising device to initiate a connection to one of the synchronized scanners. The connection can be initiated by calling Gap_ConnectFromPawr.

```
bleResult_t Gap_ConnectFromPawr
    (const gapConnectionFromPawrParameters_t* pParameters, gapConnectionCallback_t connCallback);
```

Parent topic: Periodic Advertising with Responses (PAwR)

Parent topic:*Generic Access Profile (GAP) Layer*

Encrypted Advertising Data

This section describes the Central and Peripheral setup for encrypted advertising data.

Central Setup

Use the Gap_DecryptAdvertisingData API to decrypt the contents of "Encrypted Advertising Data" (0x31) AD types included in scanned data.

```
bleResult_t Gap_DecryptAdvertisingData
    (uint8_t *pData, uint16_t dataLength, const uint8_t *pKey, const uint8_t *pIV, uint8_t *pOutput)
```

Parent topic: Encrypted Advertising Data

Peripheral Setup

Use the Gap_EncryptAdvertisingData API to obtain the encrypted advertising data, which can then be placed inside the "Encrypted Advertising Data" (0x31) AD type.

```
bleResult_t Gap_EncryptAdvertisingData
  ( const gapAdvertisingData_t *pAdvertisingData, const uint8_t *pKey, const uint8_t *pIV, uint8_t␣
↪*pOutput )
```

Parent topic: Encrypted Advertising Data

Parent topic:*Generic Access Profile (GAP) Layer*

L2CAP credit-based channels

The L2CAP layer, which is responsible for protocol multiplexing, segmentation, and reassembly operations, allows devices to communicate via connection-oriented channels. These channels use credit-based flow control, in which a device grants each peer a number of credits which the peer can use to send packets. The number of credits is decremented with every sent packet. A device can grant more credits to its peers over the duration of the connection.

Unlike the fixed L2CAP CIDs used by protocols such as ATT and SMP, credit-based channels use dynamically allocated CIDS (in the 0x0040-0xFFFF range). The CIDs are automatically allocated by the Bluetooth LE Host Stack.

The Bluetooth LE Host Stack supports both the Credit-based Flow Control Mode and the Enhanced Credit-based Flow Control Mode. In the Enhanced Credit-based Flow Control Mode, devices can open up to five channels in a single connect request/response exchange. Additionally, these channels can be later reconfigured with new MTU and MPS values. In the Credit-based Flow Control Mode, reconfiguration is not possible.

The first thing an application must do is register the control and data callbacks:

```
bleResult_t    L2ca_RegisterLeCbCallbacks
(
    l2caLeCbDataCallback_t      pCallback,
    l2caLeCbControlCallback_t   pCtrlCallback
);
```

The control callback receives events related to channel management such as connection, disconnection, received credits, reconfiguration, and so on.

The data callback receives the data which is being exchanged on the channel.

To use L2CAP credit-based channels, the application must register a PSM. The PSM is analogous to a TCP/UDP port. It is an identifier used to determine the upper layer protocol which is making use of the L2CAP channel. The dynamic PSM range is 0x0080-0x00FF. The number of PSMs supported by an application can be configured at compile time via the gL2caMaxLePsmSupported_c define. The following API must be called in order to register a PSM:

```
bleResult_t  L2ca_RegisterLePsm
(
  uint16_t     lePsm,
  uint16_t     lePsmMtu
);
```

The MTU configured via this API is used by every channel opened under the PSM, if the Credit-based Flow Control Mode is used. The minimum MTU is 23 and the maximum MTU is 65535.

When the Enhanced Credit-based Flow Control Mode is used, the MTU is specified at each connection request. In this mode, the minimum MTU is 64 and the maximum MTU is 65535.

The local MPS is not configurable by the application. It is set automatically by the Host Stack based on Controller capabilities. Usually, it will be 247.

A previously registered PSM can be deregistered:

```
bleResult_t L2ca_DeregisterLePsm
(
 uint16_t     lePsm
);
```

The number of credit-based channels that can be opened is configurable by the application via the gL2caMaxLeCbChannels_c define. This is the total number for all peers. To open a channel, the following API must be called:

```
bleResult_t L2ca_ConnectLePsm
(
  uint16_t     lePsm,
  deviceId_t   deviceId,
  uint16_t     initialCredits
);
```

To open up to five channels using Enhanced Credit-based Flow Control Mode, use this API:

```
bleResult_t  L2ca_EnhancedConnectLePsm
(
 uint16_t     lePsm,
 deviceId_t   deviceId,
 uint16_t     mtu,
 uint16_t     initialCredits,
 uint8_t      noOfChannels,
 uint16_t     *aCids
);
```

The connect APIs must be called by both the initiator and the responder (upon receiving the gL2ca_LePsmConnectRequest_c or gL2ca_LePsmEnhancedConnectRequest_c events in the application).

If the responder does not wish to accept the connection request, it can use the following APIs:

```
bleResult_t  L2ca_CancelConnection
   (
     uint16_t      lePsm,
     deviceId_t    deviceId,
    l2caLeCbConnectionRequestResult_t refuseReason
   );
bleResult_t L2ca_EnhancedCancelConnection
   (
   uint16_t lePsm,
   deviceId_t deviceId,
   l2caLeCbConnectionRequestResult_t refuseReason,
```

(continues on next page)

(continued from previous page)

```
    uint8_t noOfChannels,
    uint16_t *aCids
    );
```

When a channel has been successfully established, the gL2ca_LePsmConnectionComplete_c or gL2ca_LePsmEnhancedConnectionComplete_c events are received in the application.

To send data on a channel:

```
bleResult_t L2ca_SendLeCbData
(
  deviceId_t       deviceId,
  uint16_t         channelId,
  const uint8_t*   pPacket,
  uint16_t         packetLength
);
```

The Host Stack keeps track of the credits granted to peers for each channel and decrements them accordingly. When a peer's credit count reaches zero, the application is notified through the gL2ca_NoPeerCredits_c event and it can decide to send more credits to the peer for that channel:

```
bleResult_t L2ca_SendLeCredit
(
 deviceId_t      deviceId,
 uint16_t        channelId,
 uint16_t        credits
);
```

The application can also choose to be notified when the number of credits allocated to a peer for a certain channel is nearing 0, by setting the gL2caLowPeerCreditsThreshold_c macro to a non-zero value. When this limit is reached, the gL2ca_LowPeerCredits_c event is received and the application can choose to send more credits.

Similarly, when a device receives credits from a peer, the application is notified through the gL2ca_LocalCreditsNotification_c event. When the local device has used its last credit, it receives the same gL2ca_LocalCreditsNotification_c event with the localCredits field set to 0. The packet that could not be sent due to exhausting the credits remains queued in the Host Stack and it is sent automatically when the local device receives credits from the peer.

To improve application flow control, two notification-type events are implemented by the Host Stack:

- gL2ca_ChannelStatusChannelBusy_c
- gL2ca_ChannelStatusChannelIdle_c

When the application sends a packet using L2ca_SendLeCbData, it receives a gL2ca_ChannelStatusChannelBusy_c event in the L2CAP control callback when the Host Stack begins sending the packet. When the Host Stack has sent the packet, a gL2ca_ChannelStatusChannelIdle_c event is received. The application can choose to use this event as a signal that it is safe to send the next packet.

To disconnect a channel:

```
bleResult_t L2ca_DisconnectLeCbChannel
(
    deviceId_t   deviceId,
    uint16_t     channelId
);
```

As mentioned previously, channels which use the Enhanced Credit-based Flow Control Mode can be reconfigured. This is achieved via the API:

```
bleResult_t L2ca_EnhancedChannelReconfigure
(
    deviceId_t      deviceId,
    uint16_t        newMtu,
    uint16_t        newMps,
    uint8_t         noOfChannels,
    uint16_t        *aCids
);
```

The reconfiguration request is automatically accepted by the Host Stack if parameters are valid (as per the Bluetooth Core Spec v5.3, MTU cannot be lowered and MPS cannot be lowered for more than one channel). On the responder, the gL2ca_EnhancedReconfigureRequest_c is received by the application in case of a successful reconfiguration, informing it of the new channel parameters. On the initiator, the gL2ca_EnhancedReconfigureResponse_c is received, informing the application about the received response or a timeout.

Parent topic:*Generic Access Profile (GAP) Layer*

Enhanced ATT

The Enhanced ATT protocol allows concurrent transactions to be handled by the stack. The sequential transaction rule still exists when EATT is used, but its scope is now defined as being per instance of the Enhanced ATT Bearer. EATT transactions might execute in parallel if they are supported by distinct L2CAP channels, which use the Enhanced Credit Based Flow Control Mode (that is, distinct Enhanced ATT Bearers).

When using an Enhanced ATT Bearer, ATT MTU and L2CAP MTU are independently configurable and may be reconfigured during a connection. An increase to the MTU is allowed but reducing its size is not. Allowing MTU to be increased without needing to reestablish the connection has an advantage. It eliminates the risk of a second application using the stack, being unable to continue, due to the previously negotiated MTU being too small.

Enhanced ATT bearers are identified through Bearer Ids. Enhanced ATT Bearer Ids are assigned internally and have a valid range between 1 and 251. The Unenhanced ATT bearer is always available for a connected peer device and has the *BearerId 0*.

EATT Credits management

Credits for the L2CAP channels used by Enhanced ATT bearers may be managed internally if the autoCreditsMgmt parameter is set to TRUE in the Gap_EattConnectionRequest or Gap_EattConnectionAccept function call. Otherwise, the application is responsible for credits management.

If the application chooses to manage the credits of the L2CAP channels used as Enhanced ATT bearers, it should use the following function to send credits for a specified bearer to a peer device:

```
bleResult_t Gap_EattSendCredits
(
    deviceId_t   deviceId,
    bearerId_t   bearerId,
    uint16_t     credits
);
```

If the local credits or peer credits of the L2CAP channel used by an Enhanced ATT bearer reaches 0, agConnEvtEattBearerStatusNotification_c connection event is updated with a status value ofgEnhancedBearerSuspendedNoLocalCredits_c, or gEnhancedBearerNoPeerCredits_c respectively.

Parent topic: Enhanced ATT

EATT Connection establishment

In order to take advantage of the Enhanced ATT features, first a number of Enhanced Bearers should be opened for a connected peer device. For this, the function below may be used to create up to five Enhanced ATT bearers at a time:

```
bleResult_t Gap_EattConnectionRequest
(
    deviceId_t  deviceId,
    uint16_t    mtu,
    uint8_t     cBearers,
    uint16_t    initialCredits,
    bool_t      autoCreditsMgmt
);
```

The mtu parameter specifies the MTU for all the bearers to be established.

The cBearers parameter is used to specify the number of Enhanced ATT bearers to be opened, and should have a value between 1 and 5. The initialCredits parameter specifies the initial number of credits of the L2CAP credit based channels used as Enhanced ATT bearers.

The autoCreditsMgmt parameter is used to tell the Bluetooth LE Host Stack if it should manage L2CAP channel credits automatically. If set to TRUE the Bluetooth LE Host Stack automatically sends credits to a peer device when exhausted in chunks of initialCredits.

For example, to establish two Enhanced ATT bearers with a peer device the application may call the Gap_EattConnectionRequest as shown below:

```
bleResult_t result = Gap_EattConnectionRequest(peerDeviceId,
                                               64U,
                                               2U,
                                               3U,
                                               TRUE);
if (gBleSuccess_c != result)
{
    /* Treat error */
}
```

If an EATT Connection Request is received from a peer device it would be signaled through the gConnEvtEattConnectionRequest_c connection event of type gapEattConnectionRequest_t sent to the connection callback. The application should handle this event by calling Gap_EattConnectionAccept. The example below shows how an application may accept an incoming EATT Connection Request with the same MTU as requested by the peer device.

```
case gConnEvtEattConnectionRequest_c:
{
    gapEattConnectionRequest_t *pEattConnectionReq = &pConnectionEvent->eventData.
↪eattConnectionRequest;

    bleResult_t result = Gap_EattConnectionAccept(peerDeviceId,
                                                  TRUE,
                                                  pEattConnectionReq->mtu,
                                                  3U,
                                                  TRUE);

    if (gBleSuccess_c != result)
    {
        /* Treat error */
    }
}
break;
```

In case the localMtu specified when accepting a connection differs from the MTU requested by the peer device, the minimum of the two would become the MTU of the Enhanced Bearers.

After the Gap_EattConnectionRequest or Gap_EattConnectionAccept is called, for the result the

application should wait for the gConnEvtEattConnectionComplete_c connection event of type gapEattConnectionComplete_t shown below:

```
typedef struct {
    l2caLeCbConnectionRequestResult_t          status;
    uint16_t                                   mtu;
    uint8_t                                    cBearers;
    bearerId_t                                 aBearerIds[gGapEattMaxBearers];
} gapEattConnectionComplete_t;
```

If successful, the aBearerIds array contains the bearer ids, for the Enhanced ATT bearers established. These ids may be used with the GATT Enhanced APIs in order to trigger GATT procedures over Enhanced ATT bearers.

Parent topic: Enhanced ATT

EATT Bearer reconfiguration

One of the advantages of Enhanced ATT bearers over the Unenhanced ATT bearer is the ability to increase the MTU multiple times. To reconfigure the MTU and/or MPS of existing Enhanced ATT bearers, the Gap_EattReconfigureRequest should be used. If a mps value of 0 is given, the maximum available MPS value for that channel is used.

For example, in order to reconfigure the MTU of two bearers from 64 to 128 the application may call Gap_EattReconfigureRequest as shown below:

```
bleResult_t result = gBleSuccess_c;
bearerId_t aBearerIds[2] = {1U, 2U};
result = Gap_EattReconfigureRequest(peerDeviceId,
                                    128U,
                                    0U,
                                    2U,
                                    aBearerIds);
if (gBleSuccess_c != result)
{
    /* Treat error */
}
```

The application should monitor the gConnEvtEattChannelReconfigureResponse_c connection event of type gapEattReconfigureResponse_t for the result.

The procedure triggered by Gap_EattReconfigureRequest updates only the local MTU. The ATT_MTU for Enhanced ATT bearers is the minimum of the MTU values of the two devices.

Parent topic: Enhanced ATT

EATT Bearer disconnection

Individual Enhanced ATT bearers can be disconnected by calling the Gap_EattDisconnect API as shown below:

```
bleResult_t result = gBleSuccess_c;
result = Gap_EattDisconnect(peerDeviceId, bearerId);
if (gBleSuccess_c != result)
{
    /* Treat error */
}
```

The application should look for a connection event of type gEnhancedBearerDisconnected_c in the connection callback.

Parent topic: Enhanced ATT

Parent topic:*Generic Access Profile (GAP) Layer*

HCI Logging and Debug Keys

To facilitate debugging, the Bluetooth LE Host Stack can make use of HCI packet logging and security debug keys. This section describes how to configure and use these features.

Debug Keys

Debug keys are used during the pairing process to allow developers to decrypt and analyze Bluetooth LE traffic. When debug keys are enabled, the stack uses predefined key material instead of generating random keys, making it possible to monitor and inspect encrypted communications for debugging purposes. To use the Bluetooth LE Core Specification-defined debug keys for pairing, the application must set the gSecLibUseBleDebugKeys_d configuration macro to 1 in app_preinclude.h. It is sufficient to configure this setting on only one of the two connected devices. Bluetooth LE packet analyzers should be able to correctly decrypt communication.

HCI Packet Logging (BTSNOOP)

To configure HCI packet logging via the BTSNOOP protocol, the following steps must be performed:

- Enable the BTSNOOP debug framework module in the project configuration file (prj.conf) via west:

```
CONFIG_MCUX_COMPONENT_middleware.wireless.framework.dbg.sbtsnoop=y
CONFIG_MCUX_COMPONENT_middleware.wireless.framework.dbg.sbtsnoop.port.nxp_ble=y
```

- Generate the project.
- Set the SERIAL_BTSNOOP configuration macro to 1 in app_preinclude.h.
- Refer to BTSNOOP Module Documentation for details on how to retrieve logs.

Generic Attribute Profile (GATT) Layer

The GATT layer contains the APIs for discovering services and characteristics and transferring data between devices and is built on top of the Attribute Protocol (ATT).

The Attribute Protocol (ATT) transfers data between Bluetooth Low Energy devices on a dedicated L2CAP channel (channel ID 0x04).

As soon as a connection is established between devices, the GATT APIs are readily available. No initialization is required because the L2CAP channel is automatically created.

To identify the GATT peer instance, the same *deviceId* value from the GAP layer (obtained in the *gConnEvtConnected_c*connection event) is used.

There are two GATT roles that define the two devices exchanging data over ATT:

- GATT Server – the device that contains a GATT Database, which is a collection of services and characteristics exposing meaningful data. Usually, the Server responds to *requests* and *commands* sent by the Client. However, it can be configured to send data on its own through *notifications*and *indications*.
- GATT Client – the "active" device that usually sends *requests* and *commands* to the Server to *discover* Services and Characteristics on the Server's Database and to exchange data.

There is no fixed rule deciding which device is the Client and which one is the Server. Any device may initiate a request at any moment. Therefore, it temporarily acts as a Client, at which the peer device may respond, provided it has the Server support and a GATT Database.

Often, a GAP Central acts as a GATT Client to discover Services and Characteristics and obtain data from the GAP Peripheral, which usually has a GATT database. Many standard Bluetooth Low

Energy profiles assume that the Peripheral has a database and must act as a Server. However, this is by no means a general rule.

Client APIs

A Client can configure the ATT MTU, discover Services and Characteristics, and initiate data exchanges.

All the functions have the same first parameter: a *deviceId* which identifies the connected device whose GATT Server is targeted in the GATT procedure. This is necessary because a Client may be connected to multiple Servers at the same time.

First, however, the application must install the necessary callbacks.

Installing client callbacks

There are three callbacks that the Client application must install.

Client procedure callback

All the procedures initiated by a Client are asynchronous. They rely on exchanging ATT packets over the air.

To be informed of the procedure completion, the application must install a callback with the following signature:

```
typedef void (* gattClientProcedureCallback_t )
(
    deviceId_t              deviceId,
    gattProcedureType_t     procedureType,
    gattProcedureResult_t   procedureResult,
    bleResult_t             error
);
```

For EATT, the following signature should be used:

```
typedef void (*gattClientEnhancedProcedureCallback_t)
(
   deviceId_t deviceId,
   bearerId_t bearerId,
   gattProcedureType_t procedureType,
   gattProcedureResult_t procedureResult,
   bleResult_t error
);
```

To install this callback, the following function must be called:

```
bleResult_t GattClient_RegisterProcedureCallback
(
    gattClientProcedureCallback_t callback
);
```

The EATT procedure callback should be installed using the following API:

```
bleResult_t GattClient_RegisterEnhancedProcedureCallback
(
 gattClientEnhancedProcedureCallback_t callback
);
```

The *procedureType* parameter can be used to identify the procedure that was started and has reached completion. Only one procedure would be active at a given moment. Trying to start another procedure while a procedure is already in progress returns the error *gGattAnotherProcedureInProgress_c*.

The *procedureResult* parameter indicates whether the procedure completes successfully or an error occurs. In the latter case, the *error* parameter contains the error code.

```
void gatt ClientProcedureCallback
(
    deviceId_t              deviceId,
    gattProcedureType_t     procedureType,
    gattProcedureResult_t   procedureResult,
    bleResult_t             error
)
{
    switch (procedureType)
    {
        /* ... */
    }
}
GattClient_RegisterProcedureCallback(gattClientProcedureCallback);
```

Parent topic: Installing client callbacks

Notification and indication callbacks

When the Client receives a notification from the Server, it triggers a callback with the following prototype:

```
typedef void (* gattClientNotificationCallback_t )
(
    deviceId_t      deviceId,
    uint16_t        characteristicValueHandle,
    uint8_t *       aValue,
    uint16_t        valueLength
);
```

The *deviceId* identifies the Server connection (for multiple connections at the same time). The *characteristicValueHandle* is the attribute handle of the Characteristic Value declaration in the GATT Database. The Client must have discovered it previously to be able recognize it.

For EATT, the following signature should be used:

```
typedef void (*gattClientEnhancedNotificationCallback_t)
 ( deviceId_t deviceId,
   bearerId_t bearerId,
   uint16_t characteristicValueHandle,
   uint8_t* aValue,
   uint16_t valueLength );
```

The callback must be installed with:

```
bleResult_t GattClient_RegisterNotificationCallback
(
    gattClientNotificationCallback_t callback
);
```

Very similar definitions exist for indications.

The EATT notification callback should be installed using the following API:

```
bleResult_t GattClient_RegisterEnhancedNotificationCallback
(
 gattClientEnhancedNotificationCallback_t callback
)
```

When receiving a notification or indication, the Client uses the *characteristicValueHandle* to identify which Characteristic was notified. The Client must be aware of the possible Characteristic

Value handles that can be notified/indicated at any time, because it has previously activated them by writing its CCCD (see Reading and writing characteristic descriptors).

When the Client receives a multiple value notification from the Server, it triggers a callback with the following prototype:

```
typedef void (*gattClientMultipleValueNotificationCallback_t)
(
    deviceId_t      deviceId,
    /*!< Device ID identifying the active connection. */
    uint8_t* aHandleLenValue,
    /*!< The array of handle, value length, value tuples. */
    uint32_t totalLength
    /*!< Value array size. */
);
```

The callback must be installed with:

```
bleResult_t GattClient_RegisterMultipleValueNotificationCallback
(
    gattClientMultipleValueNotificationCallback_t callback
);
```

When using EATT, the following callback prototype and registration APIs should be used:

```
typedef void (*gattClientEnhancedMultipleValueNotificationCallback_t)
( deviceId_t deviceId,
    /*!< Device ID identifying the active connection. */
  bearerId_t bearerId,
   /*!< Bearer ID identifing the Enhanced ATT bearer used. */
  uint8_t* aHandleLenValue,
   /*!< The array of handle, value length, value tuples. */
  uint32_t totalLength      /*!< Value array size. */
 );
```

Parent topic: Installing client callbacks

Parent topic: Client APIs

MTU exchange

A radio packet sent over the Bluetooth Low Energy contains a maximum of 27 bytes of data for the L2CAP layer. The L2CAP header is 4 bytes long, including the Channel ID. Therefore, all layers above L2CAP, including ATT and GATT, can only send 23 bytes of data in a radio packet (as per *Bluetooth 4.1 Specification for Bluetooth Low Energy*). This specification also sets the default length of an ATT packet (also called *ATT_MTU*) to 23. The ATT packet length is set to this value to maintain a logical mapping between radio packets and ATT packets.

Note: This number is fixed and cannot be increased in Bluetooth Low Energy 4.1.

Therefore, any ATT request fits in a single radio packet. If the layer above ATT wishes to send more than 23 bytes of data, the data must be fragmented into smaller packets and multiple ATT requests issued.

Despite this setting, the ATT protocol allows devices to increase the ATT_MTU, only if both can support it. Increasing the ATT_MTU has only one effect: the application does not have to fragment long data. However, it can send more than 23 bytes in a single transaction. The fragmentation is moved on to the L2CAP layer. Over the air though, there would still be more than one radio packet sent.

If the GATT Client supports a larger than default MTU, it must start an MTU exchange as soon as it connects to any server. During the MTU exchange, both devices would send their maximum MTU to the other, and the minimum of the two is chosen as the new MTU.

Consider an example where the Client supports a maximum ATT_MTU of 250, and the server supports a maximum value of 120 for the same attribute. For this case, after MTU exchange, both devices must set the new ATT_MTU value equal to 120.

To initiate the MTU exchange, call the following function from *gatt_client_interface.h*:

```
bleResult_t result = GattClient_ExchangeMtu(deviceId, mtu);
if (gBleSuccess_c != result)
{
    /* Treat error */
}
```

When having the role of a GATT Client, the value of the maximum supported *ATT_MTU* of the local device is given as a parameter to the GattClient_ExchangeMtu API. On the GATT Server side, the application configures this value via the gcGattServerMtu_c variable that exists in the file ble_globals.c. The minimum of these two values is chosen as the new MTU for the connection.

When the exchange is complete, the *gGattProcExchangeMtu_c* procedure type triggers the Client callback.

```
void gattClientProcedureCallback
(
    deviceId_t deviceId,
    gattProcedureType_t procedureType,
    gattProcedureResult_t procedureResult,
    bleResult_t error
)
{
    switch (procedureType)
    {
        /* ... */
        case gGattProcExchangeMtu_c:
            if (gGattProcSuccess_c == procedureResult)
            {
                /* To obtain the new MTU */
                uint16_t newMtu;
                bleResult_t result = Gatt_GetMtu(deviceId, &newMtu);
                if (gBleSuccess_c == result)
                {
                    /* Use the value of the new MTU */
                    (void) newMtu;
                }
            }
            else
            {
                /* Handle error */
            }
            break;
        /* ... */
    }
}
```

Note: The Exchange MTU procedure is only available for the unenhanced/legacy bearer. For procedures sent on enhanced bearers, the upper layer must use provided **L2CAP APIs** to create dedicated L2CAP channels. Each channel has its own MTU value specified upon creation, which can also be reconfigured later.

Parent topic: Client APIs

Service and characteristic discovery

There are multiple APIs that can be used for Discovery. The application may use any of them, according to its necessities.

All of the following APIs have an enhanced counterpart of the form *GattClient_Enhanced[procedure]*. A *bearerId*parameter was added to specify on which bearer the transaction should take place. A value of *0* for the bearer Id identifies the Unenhanced ATT bearer. Values higher than *0* are used to identify the Enhanced ATT bearer used for the ATT procedure.

Discover all primary services

The following API can be used to discover all the Primary Services in a Server's database:

```
bleResult_t GattClient_DiscoverAllPrimaryServices
(
    deviceId_t          deviceId,
    gattService_t *     aOutPrimaryServices,
    uint8_t             maxServiceCount,
    uint8_t *           pOutDiscoveredCount
);
```

The *aOutPrimaryServices* parameter must point to an allocated array of services. The size of the array must be equal to the value of the *maxServiceCount* parameter, which is passed to make sure the GATT module does not attempt to write past the end of the array if more Services are discovered than expected.

The *pOutDiscoveredCount* parameter must point to a static variable because the GATT module uses it to write the number of Services discovered at the end of the procedure. This number is less than or equal to the *maxServiceCount*.

If there is equality, it is possible that the Server contains more than *maxServiceCount* Services, but they could not be discovered as a result of the array size limitation. It is the application developer's responsibility to allocate a large enough number according to the expected contents of the Server's database.

In the following example, the application expects to find no more than 10 Services on the Server.

```
#define mcMaxPrimaryServices_c 10
static gattService_t primaryServices[mcMaxPrimaryServices_c];
uint8_t mcPrimaryServices;
bleResult_t result = GattClient_DiscoverAllPrimaryServices
(
    deviceId,
    primaryServices,
    mcMaxPrimaryServices_c,
    &mcPrimaryServices
);
if (gBleSuccess_c != result)
{
    /* Treat error */
}
```

The operation triggers the Client Procedure Callback when complete. The application may read the number of discovered services and each service's handle range and UUID.

```
void gattClientProcedureCallback
(
    deviceId_t              deviceId,
    gattProcedureType_t     procedureType,
    gattProcedureResult_t   procedureResult,
    bleResult_t             error
)
{
    switch (procedureType)
    {
```

(continues on next page)

(continued from previous page)

```
        /* ... */
        case gGattProcDiscoverAllPrimaryServices_c:
            if (gGattProcSuccess_c == procedureResult)
            {
                /* Read number of discovered services */
                PRINT( mcPrimaryServices );
                /* Read each service's handle range and UUID */
                for (int j = 0; j < mcPrimaryServices; j++)
                {
                    PRINT( primaryServices[j]. startHandle );
                    PRINT( primaryServices[j]. endHandle );
                    PRINT( primaryServices[j]. uuidType );
                    PRINT( primaryServices[j]. uuid );
                }
            }
            else
            {
                /* Handle error */
                PRINT( error );
            }
            break;
        /* ... */
    }
}
```

Parent topic: Service and characteristic discovery

Discover primary services by UUID

To discover only Primary Services of a known type (Service UUID), the following API can be used:

```
bleResult_t GattClient_DiscoverPrimaryServicesByUuid
(
    deviceId_t          deviceId,
    bleUuidType_t       uuidType,
    const bleUuid_t *   pUuid,
    gattService_t *     aOutPrimaryServices,
    uint8_t             maxServiceCount,
    uint8_t *           pOutDiscoveredCount
);
```

The procedure is very similar to the one described in Discover all primary services. The only difference is this time we are filtering the search according to a Service UUID described by two extra parameters: *pUuid* and *uuidType*.

This procedure is useful when the Client is only interested in a specific type of Services. Usually, it is performed on Servers that are known to contain a certain Service, which is specific to a certain profile. Therefore, most of the times the search is expected to find a single Service of the given type. As a result, only one structure is usually allocated.

For example, when two devices implement the Heart Rate (HR) Profile, an HR Collector connects to an HR Sensor and may only be interested in discovering the Heart Rate Service (HRS) to work with its Characteristics. The following code example shows how to achieve this. Standard values for Service and Characteristic UUIDs, as defined by the Bluetooth SIG, are located in the *ble_sig_defines.h* file.

```
static gattService_t heartRateService;
static uint8_t mcHrs;
bleResult_t result = GattClient_DiscoverPrimaryServicesByUuid
(
    deviceId,
```

(continues on next page)

(continued from previous page)

```
    gBleUuidType16_c,            /* Service UUID type */
    gBleSig_HeartRateService_d,   /* Service UUID */
    &heartRateService,           /* Only one HRS is expected to be found */
    1,
    &mcHrs
   /* Will be equal to 1 at the end of the procedure
    if the HRS is found, 0 otherwise */

);
if (gBleSuccess_c != result)
{
    /* Treat error */
}
```

In the Client Procedure Callback, the application should check if any Service with the given UUID was found and read its handle range (also perhaps proceed with Characteristic Discovery within that service range).

```
void gattClientProcedureCallback
(
    deviceId_t deviceId,
    gattProcedureType_t procedureType,
    gattProcedureResult_t procedureResult,
    bleResult_t error
)
{
    switch (procedureType)
    {
        /* ... */
        case gGattProcDiscoverPrimaryServicesByUuid_c:
            if (gGattProcSuccess_c == procedureResult)
            {
                if (1 == mcHrs)
                {
                    /* HRS found, read the handle range */
                    PRINT( heartRateService. startHandle );
                    PRINT( heartRateService. endHandle );
                }
                else
                {
                    /* HRS not found! */
                }
            }
            else
            {
                /* Handle error */
                PRINT( error );
            }
            break;
        /* ... */
    }
}
```

Parent topic: Service and characteristic discovery

Discover included services

Discover all primary services shows how to discover Primary Services. However, a Server may also contain Secondary Services, which are not meant to be used standalone and are usually included in the Primary Services. The inclusion means that all the Secondary Service's Characteristics may be used by the profile that requires the Primary Service.

Therefore, after a Primary Service has been discovered, the following procedure may be used to discover services (usually Secondary Services) included in it:

```
bleResult_t GattClient_FindIncludedServices
(
    deviceId_t          deviceId,
    gattService_t *     pIoService,
    uint8_t             maxServiceCount
);
```

The service structure that *pIoService* points to must have the *aIncludedServices* field linked to an allocated array of services, of size *maxServiceCount*, chosen according to the expected number of included services to be found. This is the application's choice, usually following profile specifications.

Also, the service's range must be set (the *startHandle* and *endHandle* fields), which may have already been done by the previous Service Discovery procedure (as described in Discover all primary services and Discover primary services by UUID).

The number of discovered included services is written by the GATT module in the *cNumIncludedServices* field of the structure from *pIoService*. Obviously, a maximum of *maxServiceCount* included services is discovered.

The following example assumes the Heart Rate Service was discovered using the code provided in Discover primary services by UUID.

```
/* Finding services included in the Heart Rate Primary Service */
gattService_t * pPrimaryService = &heartRateService;
#define mxMaxIncludedServices_c 3
static gattService_t includedServices[mxMaxIncludedServices_c];
/* Linking the array */
pPrimaryService-> aIncludedServices = includedServices;
bleResult_t result = GattClient_FindIncludedServices
(
    deviceId,
    pPrimaryService,
    mxMaxIncludedServices_c
);
if (gBleSuccess_c != result)
{
    /* Treat error */
}
```

When the Client Procedure Callback is triggered, if any included services are found, the application can read their handle range and their UUIDs.

```
void gattClientProcedureCallback
(
    deviceId_t deviceId,
    gattProcedureType_t procedureType,
    gattProcedureResult_t procedureResult,
    bleResult_t error
)
{
    switch (procedureType)
    {
        /* ... */
        case gGattProcFindIncludedServices_c:
            if (gGattProcSuccess_c == procedureResult)
            {
                /* Read included services data */
                PRINT( pPrimaryService-> cNumIncludedServices );
                for (int j = 0; j < pPrimaryService-> cNumIncludedServices ; j++)
```

(continues on next page)

(continued from previous page)

```
            {
                PRINT( pPrimaryService-> aIncludedServices [j]. startHandle );
                PRINT( pPrimaryService-> aIncludedServices [j]. endHandle );
                PRINT( pPrimaryService-> aIncludedServices [j]. uuidType );
                PRINT( pPrimaryService-> aIncludedServices [j]. uuid );
            }
        }
        else
        {
            /* Handle error */
            PRINT( error );
        }
        break;
    /* ... */
  }
}
```

Parent topic: Service and characteristic discovery

Discover all characteristics of a service

The main API for Characteristic Discovery has the following prototype:

```
bleResult_t GattClient_DiscoverAllCharacteristicsOfService
(
    deviceId_t          deviceId,
    gattService_t *     pIoService,
    uint8_t             maxCharacteristicCount
);
```

All required information is contained in the service structure pointed to by *pIoService*, most importantly being the service range (*startHandle* and *endHandle*) which is usually already filled out by a Service Discovery procedure. If not, they need to be written manually.

Also, the service structure's *aCharacteristics* field must be linked to an allocated characteristic array.

The following example discovers all Characteristics contained in the Heart Rate Service discovered in Section Discover primary services by UUID.

```
gattService_t* pService = &heartRateService
#define mcMaxCharacteristics_c 10
static gattCharacteristic_t hrsCharacteristics[mcMaxCharacteristics_c];
pService->aCharacteristics = hrsCharacteristics;
bleResult_t result = GattClient_DiscoverAllCharacteristicsOfService
(
    deviceId,
    pService,
    mcMaxCharacteristics_c
);
```

The Client Procedure Callback is triggered when the procedure completes.

```
void gattClientProcedureCallback
(
    deviceId_t                  deviceId,
    gattProcedureType_t         procedureType,
    gattProcedureResult_t       procedureResult,
    bleResult_t                 error
)
{
```

(continues on next page)

(continued from previous page)

```
    switch (procedureType)
    {
      /* ... */
      case gGattProcDiscoverAllCharacteristics_c:
          if (gGattProcSuccess_c == procedureResult)
          {
            /* Read number of discovered Characteristics */
              PRINT(pService-> cNumCharacteristics );
            /* Read discovered Characteristics data */
              for ( uint8_t j = 0; j < pService-> cNumCharacteristics ; j++)
                {
            /* Characteristic UUID is found inside the value field
               to avoid duplication */
                    PRINT(pService-> aCharacteristics [j]. value . uuidType );
                    PRINT(pService-> aCharacteristics [j]. value . uuid );
            /* Characteristic Properties indicating the supported operations:
                     * - Read
                     * - Write
                     * - Write Without Response
                     * - Notify
                     * - Indicate
            */
                    PRINT(pService-> aCharacteristics [j]. properties );
           /* Characteristic Value Handle is used to identify the
              Characteristic in future operations */
                    PRINT(pService-> aCharacteristics [j]. value . handle );
                }
            }
            else
            {
                /* Handle error */
                PRINT( error );
            }
            break;
        /* ... */
    }
}
```

Parent topic: Service and characteristic discovery

Discover characteristics by UUID

This procedure is useful when the Client intends to discover a specific Characteristic in a specific Service. The API allows for multiple Characteristics of the same type to be discovered, but most often it is used when a single Characteristic of the given type is expected to be found.

Continuing the example from Discover primary services by UUID, assume the Client wants to discover the Heart Rate Control Point Characteristic inside the Heart Rate Service, as shown in the following code.

```
gattService_t * pService = &heartRateService;
static gattCharacteristic_t hrcpCharacteristic;
static uint8_t mcHrcpChar;
bleUuid_t hrcpUuid;
hrcpUuid.uuid16 = gBleSig_HrControlPoint_d;
bleResult_t result = GattClient_DiscoverCharacteristicOfServiceByUuid
(
    deviceId,
    gBleUuidType16_c,
    &hrcpUuid,
    pService,
```

(continues on next page)

(continued from previous page)

```
    &hrcpCharacteristic,
    1,
    &mcHrcpChar
);
```

This API can be used as in the previous examples, following a Service Discovery procedure. However, the user may want to perform a Characteristic search with UUID over the entire database, skipping the Service Discovery entirely. To do so, a dummy service structure must be defined and its range must be set to maximum, as shown in the following example:

```
gatt Service_t dummyService;
dummyService. startHandle = 0x0001;
dummyService. endHandle = 0xFFFF;
static gattCharacteristic_t hrcpCharacteristic;
static uint8_t mcHrcpChar;
bleResult_t result = GattClient_DiscoverCharacteristicOfServiceByUuid
(
    deviceId,
    gBleUuidType16_c,
    gBleSig_HrControlPoint_d,
    &dummyService,
    &hrcpCharacteristic,
    1,
    &mcHrcpChar
);
```

.

```
void gattClientProcedureCallback
(
    deviceId_t              deviceId,
    gattProcedureType_t     procedureType,
    gattProcedureResult_t   procedureResult,
    bleResult_t             error
)
{
    switch (procedureType)
    {
        /* ... */
        case gGattProcDiscoverCharacteristicByUuid_c:
            if (gGattProcSuccess_c == procedureResult)
            {
                if (1 == mcHrcpChar)
                {
                    /* HRCP found, read discovered data */
                    PRINT(hrcpCharacteristic. properties );
                    PRINT(hrcpCharacteristic. value . handle );
                }
                else
                {
                    /* HRCP not found! */
                }
            }
            else
            {
                /* Handle error */
                PRINT(error);
            }
            break;
        /* ... */
    }
```

(continues on next page)

(continued from previous page)

```
}
```

Parent topic: Service and characteristic discovery

Discover characteristic descriptors

To discover all descriptors of a Characteristic, the following API is provided:

```
bleResult_t GattClient_DiscoverAllCharacteristicDescriptors
(
    deviceId_t                  deviceId,
    gattCharacteristic_t *      pIoCharacteristic,
    uint16_t                    endingHandle,
    uint8_t                     maxDescriptorCount
);
```

The *pIoCharacteristic* pointer must point to a Characteristic structure with the *value.handle* field set (either by a discovery operation or by the application) and the *aDescriptors* field pointed to an allocated array of Descriptor structures.

The *endingHandle* should be set to the handle of the next Characteristic or Service declaration in the database to indicate when the search for descriptors must stop. The GATT Client module uses ATT Find Information Requests to discover the descriptors, and it does so until it discovers a Characteristic or Service declaration or until *endingHandle* is reached. Thus, by providing a correct ending handle, the search for descriptors is optimized and the number of packets sent over the air is reduced.

If, however, the application does not know where the next declaration lies and cannot provide this optimization hint, the *endingHandle* should be set to *0xFFFF*.

Continuing the example from Discover characteristics by UUID, the following code assumes that the Heart Rate Control Point Characteristic has no more than 5 descriptors and performs Descriptor Discovery.

```
#define mcMaxDescriptors_c 5
static gattAttribute_t aDescriptors[mcMaxDescriptors_c];
hrcpCharacteristic. aDescriptors = aDescriptors;
bleResult_t result = GattClient_DiscoverAllCharacteristicDescriptors
(
    deviceId,
    &hrcpCharacteristic,
    0xFFFF, /* We do not know where the next Characterstic Service begins */
    mcMaxDescriptors_c
);
if (gBleSuccess_c != result)
{
    /* Handle error */
}
```

The Client Procedure Callback is triggered at the end of the procedure.

```
void gattClientProcedureCallback
(
    deviceId_t              deviceId,
    gattProcedureType_t     procedureType,
    gattProcedureResult_t   procedureResult,
    bleResult_t             error
)
{
    switch (procedureType)
    {
```

(continues on next page)

(continued from previous page)

```
        /* ... */
        case gGattProcDiscoverAllCharacteristicDescriptors_c:
            if (gGattProcSuccess_c == procedureResult)
            {
                /* Read number of discovered descriptors */
                PRINT(hrcpCharacteristic. cNumDescriptors );
                /* Read descriptor data */
                for ( uint8_t j = 0; j < hrcpCharacteristic. cNumDescriptors ; j++)
                {
                    PRINT(hrcpCharacteristic. aDescriptors [j]. handle );
                    PRINT(hrcpCharacteristic. aDescriptors [j]. uuidType );
                    PRINT(hrcpCharacteristic. aDescriptors [j]. uuid );
                }
            }
            else
            {
                /* Handle error */
                PRINT(error);
            }
            break;
        /* ... */
    }
}
```

Parent topic: Service and characteristic discovery

Parent topic: Client APIs

Reading and writing characteristics

All the APIs described in the following sections have an enhanced counterpart of the form *GattClient_Enhanced[procedure]*. A *bearer id* parameter was added to specify on which bearer the transaction should take place. A value of *0* for the bearer id identifies the Unenhanced ATT bearer. Values higher than *0* are used to identify the Enhanced ATT bearer used for the ATT procedure.

Characteristic value read procedure

The main API for reading a Characteristic Value is shown here:

```
bleResult_t GattClient_ReadCharacteristicValue
(
    deviceId_t                  deviceId,
    gattCharacteristic_t *      pIoCharacteristic,
    uint16_t                    maxReadBytes
);
```

This procedure assumes that the application knows the Characteristic Value Handle, usually from a previous Characteristic Discovery procedure. Therefore, the *value.handle* field of the structure pointed by *pIoCharacteristic* must be completed.

Also, the application must allocate a large enough array of bytes where the received value (from the ATT packet exchange) is written. The *maxReadBytes* parameter is set to the size of this allocated array.

The GATT Client module takes care of long characteristics, whose values have a greater length than can fit in a single ATT packet, by issuing repeated ATT Read Blob Requests when needed.

The following examples assume that the application knows the Characteristic Value Handle and that the value length is variable, but limited to 50 bytes.

```
gattCharacteristic_t myCharacteristic;
myCharacteristic. value . handle = 0x10AB;
#define mcMaxValueLength_c 50
static uint8_t aValue[mcMaxValueLength_c];
myCharacteristic. value . paValue = aValue;
bleResult_t result = GattClient_ReadCharacteristicValue
(
    deviceId,
    &myCharacteristic,
    mcMaxValueLength_c
);
if (gBleSuccess_c != result)
{
    /* Handle error */
}
```

Regardless of the value length, the Client Procedure Callback is triggered when the reading is complete. The received value length is also filled in the *value* structure.

```
void gattClientProcedureCallback
(
    deviceId_t              deviceId,
    gattProcedureType_t     procedureType,
    gattProcedureResult_t   procedureResult,
    bleResult_t             error
)
{
    switch (procedureType)
    {
        /* ... */
        case gGattProcReadCharacteristicValue_c:
            if (*gGattProcSuccess_c == procedureResult)
            {
                /* Read value length */
                PRINT(myCharacteristic. value . valueLength );
                /* Read data */
                for ( uint16_t j = 0; j < myCharacteristic. value . valueLength ; j++)
                {
                    PRINT(myCharacteristic. value . paValue [j]);
                }
            }
            else
            {
                /* Handle error */
                PRINT(error);
            }
            break;
        /* ... */
    }
}
```

Parent topic: Reading and writing characteristics

Characteristic read by UUID procedure

This API for this procedure is shown here:

```
bleResult_t GattClient_ReadUsingCharacteristicUuid
(
    deviceId_t              deviceId,
    bleUuidType_t           uuidType,
    const bleUuid_t*        pUuid,
```

(continues on next page)

(continued from previous page)

```
    const gattHandleRange_t* pHandleRange,
    uint8_t*                aOutBuffer,
    uint16_t                maxReadBytes,
    uint16_t*               pOutActualReadBytes
);
```

This provides support for an important optimization, which involves reading a Characteristic Value without performing any Service or Characteristic Discovery.

For example, the following is the process to write an application that connects to any Server and wants to read the device name.

The device name is contained in the Device Name Characteristic from the GAP Service. Therefore, the necessary steps involve discovering all primary services, identifying the GAP Service by its UUID, discovering all Characteristics of the GAP Service and identifying the Device Name Characteristic (alternatively, discovering Characteristic by UUID inside GAP Service), and, finally, reading the device name by using the Characteristic Read Procedure.

Instead, the Characteristic Read by UUID Procedure allows reading a Characteristic with a specified UUID, assuming one exists on the Server, without knowing the Characteristic Value Handle.

The described example is implemented as follows:

```
#define mcMaxValueLength_c 10 /* Sample value length , adjust as needed */
/* First byte is for handle-value pair length. Next 2 bytes are the handle */
static uint8_t aValue[1 + 2 + mcMaxValueLength_c];
static uint16_t deviceNameLength;
bleUuid_t uuid = {
        .uuid16 = gBleSig_GapDeviceName_d
};
bleResult_t result = GattClient_ReadUsingCharacteristicUuid
(
    deviceId,
    gBleUuidType16_c,
    &uuid,
    &pHandleRange,
    aValue,
    1 + 2 + mcMaxValueLength_c,
    &deviceNameLength
);
if (gBleSuccess_c != result)
{
    /* Handle error */
}
```

The Client Procedure Callback is triggered when the reading is complete. Because only one air packet is exchanged during this procedure, it can only be used as a quick reading of Characteristic Values with length no greater than *ATT_MTU – 1*.

```
void gattClientProcedureCallback
(
    deviceId_t                  deviceId,
    gattProcedureType_t         procedureType,
    gattProcedureResult_t       procedureResult,
    bleResult_t                 error
)
{
    switch (procedureType)
    {
        /* ... */
        case gGattProcReadUsingCharacteristicUuid_c:
            if (gGattProcSuccess_c == procedureResult)
```

(continues on next page)

(continued from previous page)

```
        {
          /* Read handle-value pair length */
           PRINT(aValue[0]);
           deviceNameLength -= 1;
         /* Read characteristic value handle */
           PRINT(aValue[1] | (aValue[2] << 8));
           deviceNameLength -= 2;
           /* Read value length */
           PRINT(deviceNameLength);
           /* Read data */
           for ( uint8_t j = 0; j < deviceNameLength; j++)
           {
              PRINT(aValue[3 + j]);
           }
        }
         else
        {
           /* Handle error */
           PRINT(error);
        }
        break;
      /* ... */
   }
}
```

Parent topic: Reading and writing characteristics

Characteristic read multiple procedure

The API for this procedure is shown here:

```
bleResult_t GattClient_ReadMultipleCharacteristicValues
(
   deviceId_t                  deviceId,
   uint8_t                     cNumCharacteristics,
   gattCharacteristic_t *      aIoCharacteristics
);
```

This procedure also allows an optimization for a specific situation, which occurs when multiple Characteristics, whose values are of known, fixed-length, can be all read in one single ATT transaction (usually one single over-the-air packet).

The application must know the value handle and value length of each Characteristic. It must also write the *value.handle*and *value.maxValueLength* with the aforementioned values, respectively, and then link the *value.paValue* field with an allocated array of size *maxValueLength*.

The following example involves reading three characteristics in a single packet.

```
#define mcNumCharacteristics_c 3
#define mcChar1Length_c 4
#define mcChar2Length_c 5
#define mcChar3Length_c 6
static uint8_t aValue1[mcChar1Length_c];
static uint8_t aValue2[mcChar2Length_c];
static uint8_t aValue3[mcChar3Length_c];
static gattCharacteristic_t myChars[mcNumCharacteristics_c];
myChars[0]. value . handle = 0x0015;
myChars[1]. value . handle = 0x0025;
myChars[2]. value . handle = 0x0035;
myChars[0]. value . maxValueLength = mcChar1Length_c;
myChars[1]. value . maxValueLength = mcChar2Length_c;
```

(continues on next page)

(continued from previous page)

```
myChars[2]. value . maxValueLength = mcChar3Length_c;
myChars[0]. value . paValue = aValue1;
myChars[1]. value . paValue = aValue2;
myChars[2]. value . paValue = aValue3;
bleResult_t result = GattClient_ReadMultipleCharacteristicValues
(
    deviceId,
    mcNumCharacteristics_c,
    myChars
);
if (gBleSuccess_c != result)
{
    /* Handle error */
}
```

When the Client Procedure Callback is triggered, if no error occurs, each Characteristic's value length should be equal to the requested lengths.

```
void gattClientProcedureCallback
(
    deviceId_t              deviceId,
    gattProcedureType_t p  rocedureType,
    gattProcedureResult_t  procedureResult,
    bleResult_t             error
)
{
    switch (procedureType)
    {
        /* ... */
       case gGattProcReadMultipleCharacteristicValues_c:
            if (gGattProcSuccess_c == procedureResult)
            {
                for ( uint8_t i = 0; i < mcNumCharacteristics_c; i++)
                {
                    /* Read value length */
                    PRINT(myChars[i]. value . valueLength );
                    /* Read data */
                    for ( uint8_t j = 0; j < myChars[i]. value . valueLength ; j++)
                    {
                        PRINT(myChars[i]. value . paValue [j]);
                    }
                }
            }
            else
            {
                /* Handle error */
                PRINT(error);
            }
            break;
        /* ... */
    }
}
```

If the server does not know the length of the characteristic values, then the Read Multiple Variable Characteristic Values procedure can be used. This sub-procedure is used to read multiple characteristic values of variable length from a server when the client knows the characteristic value handles. The response returns the characteristic values and their corresponding lengths in the Length Value Tuple List parameter.

```
bleResult_t GattClient_ReadMultipleVariableCharacteristicValues
(
```

(continues on next page)

(continued from previous page)

```
  deviceId_t              deviceId,
  uint8_t                 cNumCharacteristics,
  gattCharacteristic_t*   pIoCharacteristics
);
```

The following example involves reading three characteristics of variable length in a single packet.

```
#define mcNumCharacteristics_c 3
#define mcCharLengthMax_c 10
static uint8_t aValue1[mcCharLengthMax_c];
static uint8_t aValue2[mcCharLengthMax_c];
static uint8_t aValue3[mcCharLengthMax_c];
static gattCharacteristic_t myChars[mcNumCharacteristics_c];
myChars[0].value.handle = 0x0015;
myChars[1].value.handle = 0x0025;
myChars[2].value.handle = 0x0035;
myChars[0].value.paValue = aValue1;
myChars[1].value.paValue = aValue2;
myChars[2].value.paValue = aValue3;
bleResult_t result = GattClient_ReadMultipleVariableCharacteristicValues
(
    deviceId,
    mcNumCharacteristics_c,
    pIoCharacteristics
);
if (gBleSuccess_c != result)
{
    /* Handle error */
}
```

The result of this procedure is sent to the application via the GATT procedure callback. The response includes the characteristic value together with a handle, length pair corresponding to each characteristic.

```
static void BleApp_GattClientCallback
(
    deviceId_t              serverDeviceId,
    gattProcedureType_t     procedureType,
    gattProcedureResult_t   procedureResult,
    bleResult_t             error
)
{
    switch (procedureResult)
    {
        /* ... */
        case gGattProcReadMultipleVarLengthCharValues_c:
            if (gGattProcSuccess_c == procedureResult)
            {
                for (uint8_t i = 0; i < mcNumCharacteristics_c; i++)
                {
                    /* Print characteristic handle and length */
                    PRINT(myChars[i].value.handle);
                    PRINT(myChars[i].value.valueLength);
                    for (uint8_t j = 0; j < myChars[i].value.maxValueLength; j++)
                    {
                        /* Print characteristic value */
                        PRINT(myChars[i].value.paValue[j]);
                    }
                }
            }
```

(continues on next page)

(continued from previous page)

```
        else
        {
            /* Handle error */
        }
        break;
}
```

Parent topic: Reading and writing characteristics

Characteristic write procedure

There is a general API that may be used for writing Characteristic Values:

```
bleResult_t GattClient_WriteCharacteristicValue
(
    deviceId_t                      deviceId,
    const gattCharacteristic_t *        pCharacteristic,
    uint16_t                        valueLength,
    const uint8_t *                   aValue,
    bool_t                          withoutResponse,
    bool_t                          signedWrite,
    bool_t                          doReliableLongCharWrites,
    const uint8_t *                   aCsrk
);
```

It has many parameters to support different combinations of Characteristic Write Procedures.

The structure pointed to by the *pCharacteristic* is only used for the *value.handle*field which indicates the Characteristic Value Handle. The value to be written is contained in the *aValue* array of size *valueLength*.

The *withoutResponse* parameter can be set to *TRUE* if the application wishes to perform a Write Without Response Procedure, which translates into an ATT Write Command. Both *signedWrite* and *aCsrk* are deprecated and ignored.

Finally, *doReliableLongCharWrites* should be sent to *TRUE* if the application is writing a long Characteristic Value (one that requires multiple air packets due to *ATT_MTU* limitations) and wants the Server to confirm each part of the attribute that is sent over the air.

To simplify the application code, the following macros are defined:

```
#define GattClient_SimpleCharacteristicWrite(deviceId, pChar, valueLength, aValue) \
    GattClient_WriteCharacteristicValue\
        (deviceId, pChar, valueLength, aValue, FALSE, FALSE, FALSE, NULL)
```

This is the simplest usage for writing a Characteristic. It sends an ATT Write Request if the value length does not exceed the maximum space for an over-the-air packet (*ATT_MTU – 3*). Otherwise, it sends ATT Prepare Write Requests with parts of the attribute, without checking the ATT Prepare Write Response data for consistency, and in the end an ATT Execute Write Request.

```
#define GattClient_CharacteristicWriteWithoutResponse(deviceId, pChar, valueLength, aValue) \
    GattClient_WriteCharacteristicValue\
        (deviceId, pChar, valueLength, aValue, TRUE, FALSE, FALSE, NULL)
```

This usage sends an ATT Write Command. Long Characteristic values are not allowed here and trigger a *gBleInvalidParameter_c* error.

This is a short example to write a 3-byte long Characteristic Value.

```
gattCharacteristic_t myChar;
myChar. value . handle = 0x00A0; /* Or maybe it was previously discovered? */
#define mcValueLength_c 3
```

(continues on next page)

(continued from previous page)

```
uint8_t aValue[mcValueLength_c] = { 0x01, 0x02, 0x03 };
bleResult_t result = GattClient_SimpleCharacteristicWrite
(
    deviceId,
    &myChar,
    mcValueLength_c,
    aValue
);
if (gBleSuccess_c != result)
{
    /* Handle error */
}
```

The Client Procedure Callback is triggered when writing is complete.

```
void gattClientProcedureCallback
(
    deviceId_t                  deviceId,
    gattProcedureType_t         procedureType,
    gattProcedureResult_t       procedureResult,
    bleResult_t                 error
)
{
    switch (procedureType)
    {
        /* ... */
        case gGattProcWriteCharacteristicValue_c:
            if (gGattProcSuccess_c == procedureResult)
            {
                /* Continue */
            }
            else
            {
                /* Handle error */
                PRINT(error);
            }
            break;
        /* ... */
    }
}
```

Parent topic: Reading and writing characteristics

Parent topic: Client APIs

Reading and writing characteristic descriptors

Two APIs are provided for these procedures which are very similar to Characteristic Read and Write.

The only difference is that the handle of the attribute to be read/written is provided through a pointer to an *gattAttribute_t* structure (same type as the *gattCharacteristic_t.value* field).

All of the following APIs have an enhanced counterpart of the form *GattClient_Enhanced[procedure]*. A *bearerId*parameter was added to specify on which bearer the transaction should take place. A value of *0* for the bearer Id identifies the Unenhanced ATT bearer. Values higher than *0* are used to identify the Enhanced ATT bearer used for the ATT procedure.

```
bleResult_t GattClient_ReadCharacteristicDescriptor
(
```

(continues on next page)

(continued from previous page)

```
    deviceId_t              deviceId,
    gattAttribute_t *       pIoDescriptor,
    uint16_t                maxReadBytes
);
```

The *pIoDescriptor->handle* is required (it may have been discovered previously by *GattClient_DiscoverAllCharacteristicDescriptors*). The GATT module fills the value that was read in the fields *pIoDescriptor->aValue* (must be linked to an allocated array) and *pIoDescriptor->valueLength* (size of the array).

Writing a descriptor is also performed similarly with this function:

```
bleResult_t GattClient_WriteCharacteristicDescriptor
(
    deviceId_t              deviceId,
    gattAttribute_t *       pDescriptor,
    uint16_t                valueLength,
    uint8_t *               aValue
);
```

Only the *pDescriptor->handle* must be filled before calling the function.

One of the most frequently written descriptors is the Client Characteristic Configuration Descriptor (CCCD). It has a well-defined UUID (*gBleSig_CCCD_d*) and a 2-byte long value that can be written to enable/disable notifications and/or indications.

In the following example, a Characteristic's descriptors are discovered and its CCCD written to activate notifications.

```
static gattCharacteristic_t myChar;
myChar.value.handle = 0x00A0; /* Or maybe it was previously discovered? */
#define mcMaxDescriptors_c 5
static gattAttribute_t aDescriptors[mcMaxDescriptors_c];
myChar.aDescriptors = aDescriptors;
/* ... */
{
    bleResult_t result = GattClient_DiscoverAllCharacteristicDescriptors
    (
        deviceId,
        &myChar,
        0xFFFF,
        mcMaxDescriptors_c
    );
    if (gBleSuccess_c != result)
    {
        /* Handle error */
    }
}
/* ... */
void gattClientProcedureCallback
(
    deviceId_t                  deviceId,
    gattProcedureType_t         procedureType,
    gattProcedureResult_t       procedureResult,
    bleResult_t                 error
)
{
    switch (procedureType)
    {
      /* ... */
      case gGattProcDiscoverAllCharacteristicDescriptors_c:
        if (gGattProcSuccess_c == procedureResult)
```

(continues on next page)

(continued from previous page)

```
        {
         /* Find CCCD */
          for ( uint8_t j = 0; j < myChar.cNumDescriptors; j++)
            {
             if ((myChar.aDescriptors[j].uuidType == gBleUuidType16_c) &&
                 (gBleSig_CCCD_d == myChar.aDescriptors[j].uuid.uuid16))
              {
                 uint8_t cccdValue[2];
                 packTwoByteValue(gCccdNotification_c, cccdValue);
                 bleResult_t result = GattClient_WriteCharacteristicDescriptor
                  (
                    deviceId,
                    &myChar. aDescriptors [j],
                    2,
                    (uint8_t*)&cccdValue
                  );
                 if (gBleSuccess_c != result)
                    {
                     /* Handle error */
                    }
                    break;
                  }
             }
          }
          else
          {
             /* Handle error */
             PRINT(error);
          }
          break;
      case gGattProcWriteCharacteristicDescriptor_c:
          if (gGattProcSuccess_c == procedureResult)
          {
             /* Notification successfully activated */
          }
          else
          {
             /* Handle error */
             PRINT(error);
          }
      /* ... */
   }
}
```

Parent topic: Client APIs

Resetting procedures

To cancel an ongoing Client Procedure, the following API can be called:

```
bleResult_t GattClient_ResetProcedure (void);
```

It resets the internal state of the GATT Client and new procedure may be started at any time.

Parent topic: Client APIs

Parent topic:*Generic Attribute Profile (GATT) Layer*

Server APIs

Once the GATT Database has been created and the required security settings have been registered with *Gap_RegisterDeviceSecurityRequirements*, all ATT Requests and Commands and at-

tribute access security checks are handled internally by the GATT Server module.

Besides this automatic functionality, the application may use GATT Server APIs to send Notifications and Indication and, optionally, to intercept Clients' attempts to write certain attributes.

Server callback

The first GATT Server call is the installation of the Server Callback, which has the following prototype:

```
typedef void (* gattServerCallback_t )
(
    deviceId_t              deviceId,     /*!< Device ID identifying the active connection. */
    gattServerEvent_t *     pServerEvent /*!< Server event. */
);
```

For EATT, the following signature should be used:

```
typedef void (*gattServerEnhancedCallback_t) ( deviceId_t deviceId, bearerId_t bearerId, gattServerEvent_
↪t* pServerEvent );
```

The callback can be installed with:

```
bleResult_t GattServer_RegisterCallback
(
    gattServerCallback_t callback
);
```

The EATT server callback should be installed using the following API:

```
bleResult_t GattServer_RegisterEnhancedCallback
(
     gattServerEnhancedCallback_t callback
);
```

The first member of the *gattServerEvent_t* structure is the *eventType*, an enumeration type with the following possible values:

- *gEvtMtuChanged_c:* Signals that the Client-initiated MTU Exchange Procedure has completed successfully and the *ATT_MTU* has been increased. The event data contains the new value of the *ATT_MTU*. Is it possible that the application flow depends on the value of the *ATT_MTU*, for example, there may be specific optimizations for different *ATT_MTU* ranges. This event is not triggered if the *ATT_MTU* was not changed during the procedure.
- *gEvtHandleValueConfirmation_c:* A Confirmation was received from the Client after an Indication was sent by the Server.
- *gEvtAttributeWritten_c, gEvtAttributeWrittenWithoutResponse_c:* See Attribute write notifications.
- *gEvtCharacteristicCccdWritten_c:* The Client has written a CCCD. The application should save the CCCD value for bonded devices with *Gap_SaveCccd*.
- *gEvtError_c:* An error occurred during a Server-initiated procedure.
- *gEvtLongCharacteristicWritten_c*: A long characteristic was written.
- *gEvtInvalidPduReceived_c*: An invalid PDU was received from Client. Application decides if disconnection is required.
- *gEvtAttributeRead_c*: An attribute registered with GattServer_RegisterHandlesForReadNotifications is being read.

Parent topic: Server APIs

Sending notifications and indications

The APIs provided for these Server-initiated operations are very similar.

All of the following APIs have an enhanced counterpart of the form *GattServer_Enhanced[procedure]*. A *bearerId* parameter was added to specify on which bearer the transaction should take place. A value of *0* for the bearerId identifies the Unenhanced ATT bearer. Values higher than*0* are used to identify the Enhanced ATT bearer used for the ATT procedure.

```
bleResult_t GattServer_SendNotification
(
    deviceId_t      deviceId,
    uint16_t        handle
);
bleResult_t GattServer_SendIndication
(
    deviceId_t      deviceId,
    uint16_t        handle
);
```

Only the attribute handle needs to be provided to these functions. The attribute value is automatically retrieved from the GATT Database.

Note: It is the application developer's responsibility to check if the Client designated by the *deviceId* has previously activated Notifications/Indications by writing the corresponding CCCD value. To do that, the following GAP APIs should be used:

```
bleResult_t Gap_CheckNotificationStatus
(
    deviceId_t      deviceId,
    uint16_t        handle,
    bool_t *        pOutIsActive
);
bleResult_t Gap_CheckIndicationStatus
(
    deviceId_t      deviceId,
    uint16_t        handle,
    bool_t *        pOutIsActive
    );
```

Note: It is necessary to use these two functions with the *Gap_SaveCccd* only for bonded devices, because the data is saved in NVM and reloaded at reconnection. For devices that do not bond, the application may also use its own bookkeeping mechanism.

There is an important difference between sending **Notifications and Indications**:

- The latter can only be sent one at a time. In addition, the application must wait for the Client Confirmation (signaled by the *gEvtHandleValueConfirmation_c* Server event, or by a *gEvtError_c* event with *gGattClientConfirmationTimeout_c* error code) before sending a new Indication. Otherwise, a *gEvtError_c* event with *gGattIndicationAlreadyInProgress_c* error code is triggered.
- The Notifications can be sent consecutively.

Parent topic: Server APIs

Attribute write notifications

When the GATT Client reads and writes values from/into the Server's GATT Database, it uses ATT Requests.

The GATT Server module implementation manages these requests and, according to the database security settings and the Client's security status (authenticated, authorized, and so on), automatically sends the ATT Responses without notifying the application.

There are however some situations where the application needs to be informed of ATT packet exchanges. For example, a lot of standard profiles define, for certain Services, some, so-called, Control-Point Characteristics. These are Characteristics whose values are only of immediate significance to the application. Writing these Characteristics usually triggers specific actions.

For example, consider a fictitious Smart Lamp. It has Bluetooth Low Energy connectivity in the Peripheral role and it contains a small GATT Database with a Lamp Service (among other Services). The Lamp Service contains two Characteristics: the Lamp State Characteristic (LSC) and the Lamp Action Characteristic (LAC).

LSC is a "normal" Characteristic with Read and Write properties. Its value is either 0, lamp off, or 1, lamp on). Writing the value sets the lamp in the desired state. Reading it provides its current state, which is only useful when passing the information remotely.

The LAC has only one property, which is Write Without Response. The user can use the Write Without Response procedure to write only the value 0x01 (all other values are invalid). Whenever the user writes 0x01 in LAC, the lamp switches its state.

The LAC is a good example of a Control-Point Characteristic for these reasons:

- Writing a certain value (in this case 0x01) triggers an action on the lamp.
- The value the user writes has immediate significance only ("0x01 switches the lamp") and is never used again in the future. For this reason, it does not need to be stored in the database.

Obviously, whenever a Control-Point Characteristic is written, the application must be notified to trigger some application-specific action.

The GATT Server allows the application to register a set of attribute handles as "write-notifiable", in other words, the application wants to receive an event each time any of these attributes is written by the peer Client.

All Control-Point Characteristics in the GATT Database must have their Value handle registered. In fact, the application may register any other handle for write notifications for its own purposes with the following API:

```
bleResult_t GattServer_RegisterHandlesForWriteNotifications
(
    uint8_t          handleCount,
    const uint16_t *      aAttributeHandles
);
```

The *handleCount* is the size of the *aAttributeHandles* array and it cannot exceed *gcGattMaxHandleCountForWriteNotifications_c*.

After an attribute handle has been registered with this function, whenever the Client attempts to write its value, the GATT Server Callback is triggered with one of the following event types:

- *gEvtAttributeWritten_c* is triggered when the attribute is written with a Write procedure (ATT Write Request). In this instance, the application has to decide whether the written value is valid and whether it must be written in the database, and, if so, the application must write the value with the *GattDb_WriteAttribute*, see *GATT database application interface*. At this point, the GATT Server module does not automatically send the ATT Write Response over the air. Instead, it waits for the application to call this function:

```
bleResult_t GattServer_SendAttributeWrittenStatus
(
    deviceId_t      deviceId,
    uint16_t        attributeHandle,
    uint8_t         status
);
```

This API also has an enhanced counterpart, which adds the *bearerId* parameter.

The value of the *status* parameter is interpreted as an ATT Error Code. It must be equal to the *gAttErrCodeNoError_c* (0x00) if the value is valid and it is successfully processed by the appli-

cation. Otherwise, it must be equal to a profile-specific error code (in interval 0xE0-0xFF) or an application-specific error code (in interval 0x80-0x9F).

- *gEvtAttributeWrittenWithoutResponse_c* is triggered when the attribute is written with a Write Without Response procedure (ATT Write Command). Because this procedure expects no response, the application may process it and, if necessary, write it in the database. Regardless of whether the value is valid or not, no response is needed from the application.
- *gEvtLongCharacteristicWritten_c* is triggered when the Client has completed writing a Long Characteristic value; the event data includes the handle of the Characteristic Value attribute and a pointer to its value in the database.

Attributes can also be registered for read notifications using the followng API:

```
bleResult_t GattServer_RegisterHandlesForReadNotifications
(
 uint8_t handleCount,
 const uint16_t* aAttributeHandles
);
```

To unregister one or more handles from the list for either write or read, the following APIs can be used:

```
bleResult_t GattServer_UnregisterHandlesForWriteNotifications
(
 uint8_t handleCount,
 const uint16_t* aAttributeHandles
);

bleResult_t GattServer_UnregisterHandlesForReadNotifications
(
 uint8_t handleCount,
 const uint16_t* aAttributeHandles
);
```

Parent topic: Server APIs

Parent topic:*Generic Attribute Profile (GATT) Layer*

GATT database application interface

For over-the-air packet exchanges between a Client and a Server, the GATT Server module automatically retrieves data from the GATT database and responds to all ATT Requests from the peer Client, provided it passes the security checks. This ensures that the Server application does not have to perform any kind of searches over the database.

However, the application must have access to the database to write meaningful data into its characteristics. For example, a temperature sensor must periodically write the temperature, which is measured by an external thermometer, into the Temperature Characteristic.

For these kinds of situations, a few APIs are provided in the *gatt_db_app_interface.h* file.

Note: All functions provided by this interface are executed synchronously. The result of the operation is saved in the return value and it generates no event.

Writing and reading attributes

These are the two functions to perform basic attribute operations from the application:

```
bleResult_t GattDb_WriteAttribute
(
```

(continues on next page)

(continued from previous page)

```
    uint16_t            handle,
    uint16_t            valueLength,
    const uint8_t *     aValue
);
```

The value length must be valid, as defined when the database is created. Otherwise, a *gGattInvalidValueLength_c* error is returned.

Also, if the database is created statically, as explained in *Creating GATT database*, the *handle* may be referenced through the enumeration member with a friendly name defined in the *gatt_db.h*.

```
bleResult_t GattDb_ReadAttribute
(
    uint16_t      handle,
    uint16_t      maxBytes,
    uint8_t *     aOutValue,
    uint16_t *    pOutValueLength
);
```

The *aOutValue* array must be allocated with the size equal to *maxBytes*.

Parent topic:*GATT database application interface*

Finding attribute handles

Although the application should be fully aware of the contents of the GATT Database, in certain situations it might be useful to perform some dynamic searches of certain attribute handles.

To find the handle value for a Service for which only the UUID is know the following API can be used:

```
bleResult_t GattDb_FindServiceHandle
(
 uint16_t startHandle,
 bleUuidType_t serviceUuidType,
 const bleUuid_t* pServiceUuid,
 uint16_t* pOutServiceHandle
);
```

To find a specific Characteristic Value Handle in a Service whose declaration handle is known, the following API is provided:

```
bleResult_t GattDb_FindCharValueHandleInService
(
    uint16_t                serviceHandle,
    bleUuidType_t           characteristicUuidType,
    const bleUuid_t *       pCharacteristicUuid,
    uint16_t *              pOutCharValueHandle
);
```

If the return value is *gBleSuccess_c*, the handle is written at *pOutCharValueHandle*. If the *serviceHandle* is invalid or not a valid Service declaration, the *gBleGattDbInvalidHandle_c* is returned. Otherwise, the search is performed starting with the *serviceHandle+1*. If no Characteristic of the given UUID is found, the function returns the *gBleGattDbCharacteristicNotFound_c* value.

To find a Characteristic Descriptor of a given type in a Characteristic, when the Characteristic Value Handle is known, the following API is provided:

```
bleResult_t GattDb_FindDescriptorHandleForCharValueHandle
(
    uint16_t            charValueHandle,
    bleUuidType_t       descriptorUuidType,
```

(continues on next page)

(continued from previous page)

```
    const bleUuid_t *   pDescriptorUuid,
    uint16_t *          pOutDescriptorHandle
);
```

Similarly, the function returns *gBleGattDbInvalidHandle_c* is the handle is invalid. Otherwise, it starts searching from the *charValueHandle+1*. Then, *gBleGattDbDescriptorNotFound_c* is returned if no Descriptor of the specified type is found. Otherwise, its attribute handle is written at the *pOutDescriptorHandle* and the function returns *gBleSuccess_c*.

One of the most commonly used Characteristic Descriptors is the Client Configuration Characteristic Descriptor (CCCD), which has the UUID equal to *gBleSig_CCCD_d*. For this specific type, a special API is used as a shortcut:

```
bleResult_t GattDb_FindCccdHandleForCharValueHandle
(
    uint16_t     charValueHandle,
    uint16_t *   pOutCccdHandle
);
```

Parent topic:*GATT database application interface*

Creating GATT database

The GATT Database contains several *GATT Services* where each Service must contain at least one *GATT Characteristic*.

The Attribute Database contains a collection of *attributes*. Each attribute has four fields:

- The *attribute handle* – a 2-byte database index, which starts from 0x0001 and increases with each new attribute, not necessarily consecutive; maximum value is 0xFFFF.
- The *attribute type* or *UUID* – a 2-byte or 16-byte UUID.
- The *attribute permissions* – 1 byte containing access flags; this defines whether the attribute's value can be read or written and the security requirements for each operation type
- The *attribute value* – an array of maximum 512 bytes.

The ATT does not interpret the UUIDs and values contained in the database. It only deals with data transfer based on the attributes' handles.

The GATT gives meaning to the attributes based on their UUIDs and groups them into Characteristics and Services.

There are two possible ways of defining the GATT database:

- At compile-time (statically) or
- At runtime (dynamically)

Creating static GATT database

To define a GATT Database at compile-time, several macros are provided by the GATT_DB API. These macros expand in many different ways at compilation, generating the corresponding *Attribute Database* on which the Attribute Protocol (ATT) may operate.

This is the default way of defining the database.

The GATT Database definition is written in two files that are required to be added to the application project together with all macro expansion files:

- ***gatt_db.h*** - contains the actual declaration of Services and Characteristics.

- ***gat_uuid128.h*** – contains the declaration of Custom UUIDs (16-byte wide); these UUIDs are given a user-friendly name that is used in *gatt_db.h* file instead of the entire 16-byte sequence.

Declaring custom 128-bit UUIDs

All Custom 128-bit UUIDs are declared in the required file *gatt_uuid128.h*.

Each line in this file contains a single UUID declaration. The declaration uses the following macro:

- *UUID128 (name, byte1, byte2, ..., byte16)*

The*name* parameter is the user-friendly handle that references this UUID in the *gatt_db.h*file.

The 16 bytes are written in the *LSB-first* order each one using the 0xZZ format.

Note: On some occasions, it is desired to reuse an 128-bit UUID declared in *gatt_uuid128.h*. The 16 byte array is available through its friendly name and be accessed by including *gatt_db_handles.h* in the application. It is strongly advised to use it only in read-only operations. For example:

```
(gatt_uuid128.h)
UID128(uuid_service_wireless_uart, 0xE0, 0x1C, 0x4B, 0x5E, 0x1E, 0xEB, 0xA1, 0x5C, 0xEE, 0xF4, 0x5E,␣
↪0xBA, 0x00, 0x01, 0xFF, 0x01)
(app.c)
#include "gatt_db_handles.h"
........
/* Start Service Discovery*/
BleServDisc_FindService(peerDeviceId, gBleUuidType128_c, (bleUuid_t*) &uuid_service_wireless_uart);
```

Parent topic: Creating static GATT database

Declaring a service

There are two types of Services:

- *Primary Services*
- *Secondary Services* - these are only to be included by other Primary or Secondary Services

The Service declaration attribute has one of these UUIDs, as defined by the Bluetooth SIG:

- 0x2800 a.k.a. *«Primary Service»* - for a Primary Service declaration
- 0x2801 a.k.a. *«Secondary Service»* - for a Secondary Service declaration

The Service declaration attribute permissions are read-only and no authentication required. The Service declaration attribute value contains the *Service UUID*. The *Service Range* starts from the Service declaration and ends at the next service declaration. All the Characteristics declared within the Service Range are considered to belong to that Service. For a more comprehensive list of SIG defied UUID values, check `ble_sig_defines.h`.

Service declaration macros

The following macros are to be used for declaring a Service:

- *PRIMARY_SERVICE (name, uuid16)*
  - Most often used.
  - The *name* parameter is common to all macros; it is a universal, user-friendly identifier for the generated attribute.
  - The *uuid16* is a 2-byte SIG-defined UUID, written in 0xZZZZ format.
- *PRIMARY_SERVICE_UUID32 (name, uuid32)*

  - This macro is used for a 4-byte, SIG-defined UUID, written in 0xZZZZZZZZ format.
- *PRIMARY_SERVICE_UUID128 (name, uuid128)*
  - The *uuid128* is the friendly name given to the custom UUID in the *gatt_uuid128.h* file.
- *SECONDARY _SERVICE (name, uuid16)*
- *SECONDARY_SERVICE_UUID32 (name, uuid32)*
- *SECONDARY _SERVICE_UUID128 (name, uuid128)*
  - All three are similar to Primary Service declarations.

Parent topic: Declaring a service

Include declaration macros

Secondary Services are meant to be included by other Services, usually by Primary Services. Primary Services may also be included by other Primary Services. The inclusion is done using the Include declaration macro:

- *INCLUDE (service_name)*
  - The *service_name* parameter is the friendly name used to declare the Secondary Service.
  - This macro is used only for Secondary Services with a SIG-defined, 2-byte, Service UUID.
- *INCLUDE_CUSTOM (service_name)*
  - This macro is used for Secondary Services that have either a 4-byte UUID or a 16-byte UUID.

The effect of the service inclusion is that the *including* Service is considered to contain all the Characteristics of the *included* Service.

Parent topic: Declaring a service

Parent topic: Creating static GATT database

Declaring a characteristic

A Characteristic must only be declared inside a Service. It belongs to the most recently declared Service, so the GATT Database must always begin with a Service declaration.

The Characteristic declaration attribute has the following UUID, as defined by the Bluetooth SIG:

- 0x2803 a.k.a. *«Characteristic»*

The Characteristic declaration attribute value contains:

- the *Characteristic UUID*
- the *Characteristic Value* 's declaration handle
- the *Characteristic Properties* – Read, Write, Notify, and so on. (1 byte of flags)

The *Characteristic Range*starts from the Characteristic declaration and ends before a new Characteristic or a Service declaration.

After the Characteristic declaration these follow:

- A *Characteristic Value* declaration (mandatory; immediately after the Characteristic declaration).
- Zero or more *Characteristic Descriptor* declarations.

Characteristic declaration macros

The following macros are used to declare Characteristics:

- *CHARACTERISTIC (name, uuid16, properties)*
- *CHARACTERISTIC_UUID32 (name, uuid32, properties)*
- *CHARACTERISTIC _UUID128 (name, uuid128, properties)*

See Service declaration for *uuidXXX* parameter explanation.

The *properties* parameter is a bit mask. The flags are defined in the *gattCharacteristicPropertiesBitFields_t*.

Parent topic: Declaring a characteristic

Declaring characteristic values

The Characteristic Value declaration immediately follows the Characteristic declaration and uses one of the following macros:

- *VALUE (name, uuid16, permissions, valueLength, valueByte1, valueByte2, ...)*
- *VALUE_UUID32 (name, uuid32, permissions, valueLength, valueByte1, valueByte2, ...)*
- *VALUE _UUID128(name, uuid128, permissions, valueLength, valueByte1, valueByte2, ...)*
  - See Declaring a service for description of the *uuidXXX* parameter.
  - The *permissions* parameter is a bit mask, whose flags are defined in *gattAttributePermissionsBitFields_t* .
  - The *valueLength* is the number of bytes to be allocated for the Characteristic Value. After this parameter, exactly [*valueLength*] bytes follow in 0xZZ format, representing the initial value of this Characteristic.

These macros are used to declare Characteristic Values of *fixed lengths*.

Some Characteristics have *variable length values*. For those, the following macros are used:

- *VALUE_VARLEN (name, uuid16, permissions, maximumValueLength, initialValueLength, valueByte1, valueByte2, ...)*
- *VALUE_UUID32_VARLEN (name, uuid32, permissions, maximumValueLength, initialValueLength, valueByte1, valueByte2, ...)*
- *VALUE_UUID128_VARLEN (name, uuid128, permissions, maximumValueLength, initialValueLength, valueByte1, valueByte2, ...)*
  - The number of bytes allocated for this Characteristic Value is *maximumValueLength*.
  - The number of *valueByteXXX* parameters shall be equal to *initialValueLength*.

Obviously, *initialValueLength* is, at most, equal to *maximumValueLength*.

Parent topic: Declaring a characteristic

Declaring characteristic descriptors

Characteristic's Descriptors are declared after the Characteristic Value declaration and before the next Characteristic declaration.

The macros used to declare Characteristic Descriptors are very similar to those used to declare fixed-length Characteristic Values:

- *DESCRIPTOR (name, uuid16, permissions, descriptorValueLength, descriptorValueByte1, descriptorValueByte2, ...)*
- *DESCRIPTOR_UUID32 (name, uuid32, permissions, descriptorValueLength, descriptorValueByte1, descriptorValueByte2, ...)*

- *DESCRIPTOR_UUID128(name, uuid128, permissions, descriptorValueLength, descriptorValueByte1, descriptorValueByte2, ...)*

A special Characteristic Descriptor that is used very often is the *Client Characteristic Configuration Descriptor (CCCD)*. This is the descriptor where clients write some of the bits to activate Server notifications and/or indications. It has a reserved, 2-byte, SIG-defined UUID (0x2902), and its attribute value consists of only 1 byte (out of which 2 bits are used for configuration, the other 6 are reserved).

Because the CCCD appears very often in Characteristic definitions for standard Bluetooth Low Energy profiles, a special macro is used for CCCD declaration:

- *CCCD (name)*

This simple macro is basically equivalent to the following Descriptor declaration:

```
* DESCRIPTOR \(name, *
  *0x2902, *
  (gGattAttPermAccessReadable_c
  | gGattAttPermAccessWritable_c),
  2, 0x00, 0x00)
```

Parent topic: Declaring a characteristic

Parent topic: Creating static GATT database

Static GATT database definition examples

The GAP Service must be present on any GATT Database. It has the Service UUID equal to 0x1800, *«GAP Service»*, and it contains three read-only Characteristics, no authentication required: *Device Name, Appearance,*and *Peripheral Preferred Connection Parameters*. These also have well defined UUIDs in the SIG documents.

Most of the demos also include the optional GATT Security Levels characteristic, which defines the highest security requirements of the GATT server when operating in a LE connection.

The definition for this Service is shown here:

```
PRIMARY_SERVICE(service_gap, 0x1800)
  CHARACTERISTIC(char_device_name, 0x2A00, (gGattCharPropRead_c) )
      VALUE(value_device_name, 0x2A00, (gGattAttPermAccessReadable_c), 6, "Sensor")
  CHARACTERISTIC(char_appearance, 0x2A01, (gGattCharPropRead_c) )
     VALUE(value_appearance, 0x2A01, (gGattAttPermAccessReadable_c), 2,
      0xC2, 0x03)
  CHARACTERISTIC(char_ppcp, 0x2A04, (gGattCharPropRead_c) )
      VALUE(value_ppcp, 0x2A04, (gGattAttPermAccessReadable_c), 8, 0x0A,
      0x00, 0x10, 0x00, 0x64, 0x00, 0xE2, 0x04)
  CHARACTERISTIC(char_security_levels, gBleSig_GattSecurityLevels_d,
     (gGattCharPropRead_c) )
      VALUE(value_security_levels, gBleSig_GattSecurityLevels_d,
      (gPermissionFlagReadable_c), 2, 0x01, 0x01)
```

Another often encountered Service is the Scan Parameters Service:

```
PRIMARY_SERVICE(service_scan_parameters, 0x1813)
    CHARACTERISTIC(char_scan_interval_window, 0x2A4F,          (gGattCharPropWriteWithoutRsp_
↪c) )
      VALUE(value_scan_interval_window, 0x2A4 (gGattAttPermAccessWritable), 4, 0x00, 0x00, 0x00,␣
↪0x00)
     CHARACTERISTIC(char_scan_refresh, 0x2A31, (gGattCharPropRead_c gGattCharPropNotify_c) )
      VALUE(value_scan_refresh, 0x2A31, (gGattAttPermAccessReadable_c), 1, 0x00) CCCD(cccd_scan_
↪refresh)
```

Note: All "user-friendly" names given in declarations are statically defined as *enum* members, numerically equal to the *attribute handle* of the declaration. This means that one of those names

can be used in code wherever an attribute handle is required as a parameter of a function if *gatt_db_handles.h* is included in the application source file. For example, to write the value of the Scan Refresh Characteristic from the application-level code, use these instructions:

```
#include "gatt_db_handles.h"
...
uint8_t scan_refresh_value = 0x12;
GattDb_WriteAttribute(char_scan_refresh, 1, &scan_refresh_value);
```

For static database declarations, the 'attribute handle' is equal to the line number in the gatt_fb.h file, where the attribute is defined.

Parent topic: Creating static GATT database

Parent topic:*Creating GATT database*

Creating a GATT database dynamically

To define a GATT Database at runtime, the *gGattDbDynamic_d* macro must be defined in *app_preinclude.h* with the value equal to 1.

Then, the application must use the APIs provided by the *gatt_db_dynamic.h* interface to add and remove Services and Characteristics as needed.

See Creating static GATT database for a detailed description of Service and Characteristic parameters.

Memory considerations

The GATT Dynamic database module internally manages the memory allocation for the database. There are two ways to specify the memory configuration. The selection is done by setting gGattDynamicAttrSize_c and/or gGattDynamicValSize_c.

- *Statically pre-allocated memory area*
  - gGattDynamicAttrSize_c - the total size of the table of characteristics
  - gGattDynamicValSize_c - the total size of the values cumulated from all characteristics. If any of these macros is greater than 1 the application is responsible to decide upon the best value. In case the selected value is not big enough gBleOutOfMemory_c is returned by the GATT API calls.
- *Dynamic allocated memory area* If the gMemManagerLightExtendHeapAreaUsage define is set to 1 in the desired application, the whole available heap is used. In such as case, the user does not have to allocate space for the dynamic database. If this is not done, the user only needs to make sure that the MinimalHeapSize_c define is set to a high enough value considering all attributes and attribute values they want to add to the database, as well as other memory requirements the application might have. This method is used in case only several attributes are added dynamically. If multiple attributes are needed the method from *Statically pre-allocated memory area* above is recommended. This is because the heap gets fragmented with each adding of a new attribute.

Internally, two buffers are used by the dynamic database module: an attribute buffer and a value buffer. The attribute buffer size increases with the addition of each attribute to the database. The value buffer size increases depending on the UUID type and value lengths required by the application. The two buffers start with a minimum size and are reallocated whenever new requests to add entries are received and there is not enough available memory left. If the user removes these entries from the database, the memory reserved for those entries is not freed, but shifted, leaving room for new entries. Thus, an add operation after a remove operation might not necessarily reallocate the buffer if the new entries fit. The two buffers used by the Dynamic database module will not be available to the application until the user releases the database.

Parent topic: Creating a GATT database dynamically

Initialization and release

Before anything can be added to the database, it must be initialized with an empty collection of attributes.

The *GattDbDynamic_Init()* API is automatically called by the *GattDb_Init()* implementation provided in the *gatt_database.c* source file. Application-specific code does not need to call this API again, unless at some point it destroys the database with *GattDb_ReleaseDatabase()*.

Parent topic: Creating a GATT database dynamically

Adding services

The APIs that can be used to add Services are self-explanatory:

- *GattDbDynamic_AddPrimaryServiceDeclaration*
  - The Service UUID is specified as parameter.

    Memory requirements: one entry in the attribute buffer and UUID size in value buffer.
- *GattDbDynamic_AddSecondaryServiceDeclaration*
  - The Service UUID is specified as parameter.

    Memory requirements: one entry in the attribute buffer and UUID size in value buffer.
- *GattDbDynamic_AddIncludeDeclaration*
  - The Service UUID and handle range are specified as parameters.

    Memory requirements: one entry in the attribute buffer and 6 bytes in value buffer.

The functions have an optional out parameter *pOutHandle*. If its value is not NULL, the execution writes a 16-bit value in the pointed location representing the attribute handle of the added declaration. The application can use this handle as parameter in few *GattDbApp* APIs or in the Service removal functions.

At least one Service must be added before any Characteristic.

Parent topic: Creating a GATT database dynamically

Adding characteristics and descriptors

The APIs for adding Characteristics and Descriptors are enumerated below:

- *GattDbDynamic_AddCharacteristicDeclarationAndValue*
  - The Characteristic UUID, properties, access permissions, and initial value are specified as parameters.
- *GattDbDynamic_AddCharacteristicDeclarationWithUniqueValue*
  - Multiple calls to this API allocate a unique 512-byte value buffer as an optimization for application that deal with large value buffers that do not always need to be stored separately.
- *GattDbDynamic_AddCharacteristicDescriptor*
  - The Descriptor UUID, access permissions and initial value are specified as parameters.
- *GattDbDynamic_AddCccd*
  - Shortcut for a CCCD.

Characteristics and descriptors are automatically added at the end of the database. Thus, a service declaration should be followed by all desired characteristic and descriptor definitions before adding a new service to the database.

Parent topic: Creating a GATT database dynamically

Removing services and characteristics

To remove a Service or a Characteristic, the following APIs may be used, both of which only require the declaration handle as parameter:

- *GattDbDynamic_RemoveService*
- *GattDbDynamic_RemoveCharacteristic*

Parent topic: Creating a GATT database dynamically

Parent topic:*Creating GATT database*

Gatt caching

Service change feature

The **service changed** feature applies to GATT servers and supports the service changed characteristic, dynamic databases, and handle value indications. The GATT clients that require to be notified for structural modifications on the database should write the CCCD of the Service Changed Characteristic on the server. The value of the Service Changed characteristic is represented by 2 handle values for the handle range affected by the modifications.

The changes that trigger a server database modification are represented by the following API calls:

- *GattDbDynamic_AddPrimaryServiceDeclaration*
- *GattDbDynamic_AddSecondaryServiceDeclaration*
- *GattDbDynamic_AddIncludeDeclaration*
- *GattDbDynamic_AddCharacteristicDeclarationAndValue*
- *GattDbDynamic_AddCharDescriptor*
- *GattDbDynamic_AddCccd*
- *GattDbDynamic_RemoveService*
- *GattDbDynamic_RemoveCharacteristic*

Those GATT server API calls update two internal handles to memorize the minimum and maximum range affected by the change.

After the GATT server database update is done, the application must call the GattDbDynamic_EndDatabaseUpdate() API. After this, a Service Changed indication is internally sent to each connected peer that has enabled these indications. The indication contains the handle range affected by the change.

. For bonded devices with whom the server is not currently in an active connection, the changes are buffered on the server and the peers are notified upon reconnection.

Parent topic: Gatt caching

Robust caching

Robust caching is a feature where the server sends an error response to the client if the server does not consider the client to be aware of a database structural change. A server supporting robust caching provides the Client Supported Features, Database Hash, and Service Changed characteristics. To indicate support for robust caching, clients should write the robust caching bit (bit 0) of the Client Supported Features characteristic on the server. An example of the GAP service definition for a server with robust caching support is the following:

- *PRIMARY_SERVICE(service_gatt, gBleSig_GenericAttributeProfile_d)*
- *CHARACTERISTIC(char_service_changed, gBleSig_GattServiceChanged_d, (gGattCharPropIndicate_c))*

- *VALUE(value_service_changed, gBleSig_GattServiceChanged_d, (gPermissionNone_c), 4, 0x01, 0x00, 0xFF, 0xFF)*
- *CCCD(cccd_service_changed)*
- *CHARACTERISTIC(char_client_supported_features, gBleSig_GattClientSupportedFeatures_d, (gGattCharPropRead_c | gGattCharPropWrite_c))*
- *VALUE(value_client_supported_features, gBleSig_GattClientSupportedFeatures_d, (gPermissionFlagReadable_c | gPermissionFlagWritable_c), 8, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00)*
- *CHARACTERISTIC(char_database_hash, gBleSig_GattDatabaseHash_d, (gGattCharPropRead_c))*
- *VALUE(value_database_hash, gBleSig_GattDatabaseHash_d, (gPermissionFlagReadable_c), 16, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00)*

The Client Supported Features characteristic should be declared as an array. The array size is equal to the maximum number of connections, which can be active at one time, so that each possible peer can have its own value. The Server Supported Features characteristic value is automatically written by the Host to indicate robust caching support when appropriate. The Database Hash characteristic value is a 128-bit unsigned integer number where the computed hash value is written.

In order to enable the Robust Caching feature, the *gGattCaching_d* define should be enabled in the file app_preinclude.h. In order to enable automatic Host support for Robust Caching, the *gGattAutomaticRobustCachingSupport_d*define should also be enabled. This includes writing the Client Supported Features characteristic, writing the CCCD of the Service Changed characteristic, and reading the Database Hash after reconnecting with a previously bonded peer to check for new changes. If this define is not enabled, then it is up to the application to read and write all necessary characteristic for Robust Caching support.

The client state is kept on the server using the following enum:

```
typedef enum
{
gGattClientChangeUnaware_c = 0x00U, /*!< Gatt client state */
gGattClientStateChangePending_c = 0x01U, /*!< Gatt client state */
gGattClientChangeAware_c = 0x02U, /*!< Gatt client state */
} gattCachingClientState_c;
```

The initial state of a client without a trusted relationship is change-aware. The state of a client with a trusted relationship remains unchanged from the previous connection. However, in cases where the database has been updated since the last connection, the initial state is change-unaware. When a database update occurs, all connected clients become change unaware.

If a change-unaware client sends an ATT command, the server ignores it. For ATT requests received from a change-unaware client, the server sends an error response with the error code set to *gAttErrCodeDatabaseOutOfSync_c*. The server should also not send indications and notifications to change unaware clients, except for the Service Changed indication. The state of a client is verified by the GATT server before executing each command, request or sending any notifications or indications.

The following PDU types are an exception to this rule and do not generate an gAttErrCodeDatabaseOutOfSync_c error code:

- ATT_FIND_INFORMATION_REQ
- ATT_FIND_BY_TYPE_VALUE_REQ
- ATT_READ_BY_GROUP_TYPE_REQ
- ATT_EXECUTE_WRTIE

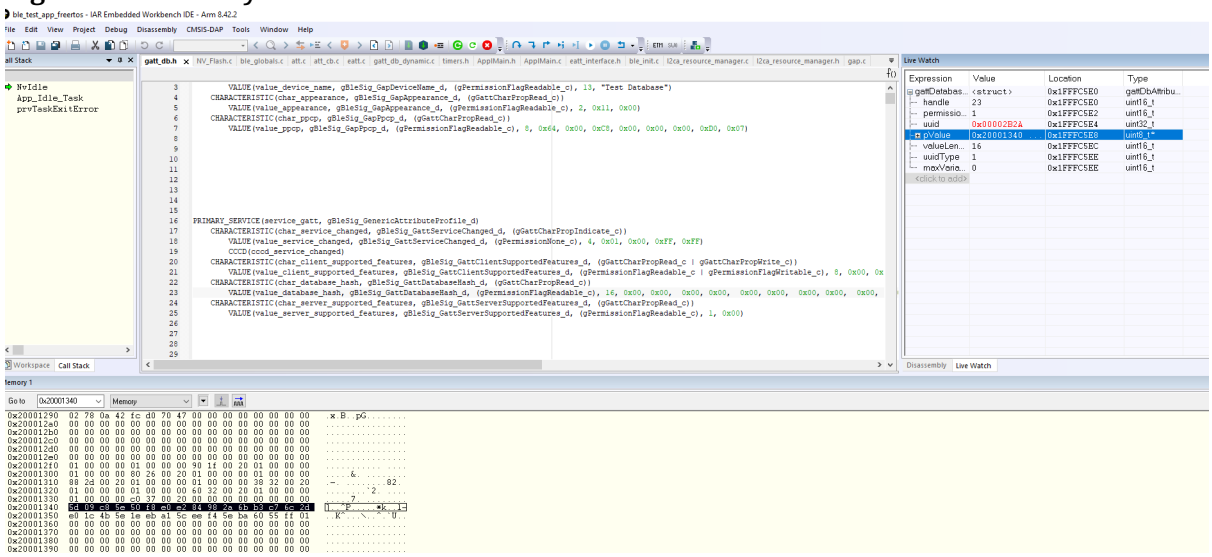
For a change unaware client to become change aware again, one of the following must happen:

- The client receives and confirms a Service Changed Indication.
- The server, upon receiving a request from a change unaware client, sends the client a response with the error code set to Database Out Of Sync and then the server receives another ATT request from the client.
- The change unaware client reads the Database Hash characteristic and then the server receives another ATT request from the client.

The function *GattDb_ComputeDatabaseHash()* is used by the server to compute the hash value and save its value in the database. The computation is done when a read request for the database hash characteristic is first received from a peer GATT client for dynamic databases.

For static databases, hash computation is disabled by default. If you have a static database and want to compute the database hash, then declare the following define to TRUE in app_preinclude.h: *gGattDbComputeHash_d*. By doing this, the hash value is computed during the host initialization. The value is written directly to the database as characteristic and it can be viewed in the memory, as see in the image below. Since static databases do not change in structure over time, this value remains constant, so it can be saved separately and written manually to memory if needed. See Figure 10 below.

Figure 10. Memory view of the Database Hash characteristic



On the client side, the *GattClient_GetDatabaseHash()* function is used to read the hash value from a peer GATT server. If the *gGattAutomaticRobustCachingSupport_c* define is enabled, then the following steps are executed automatically:

- Writing the Client Supported Features characteristic value to indicate robust caching support – set BIT0 to 1.
- Write the CCCD of the Service Changed characteristic.
- Read the initial value of the Database Hash characteristic and store it locally.

Otherwise, just the read request for the Database Hash characteristic is sent to the peer.

The following arrays and variables are used for the implementations of the robust caching and service changed features (declared in ble_globals.c):

```
/* Service changed indication buffer */
gattHandleRange_t gServiceChangedIndicationStorage[gMaxBondedDevices_c];

/* client saved values for service changed characteristic and CCCD handles */
uint16_t mActiveServiceChangedCharHandle[gAppMaxConnections_c] = {gGattDbInvalidHandle_d};
uint16_t mServiceChangedCharHandle[gMaxBondedDevices_c] = {gGattDbInvalidHandle_d};
```

(continues on next page)

(continued from previous page)

```
    uint16_t mActiveServiceChangedCCCDHandle[gAppMaxConnections_c] = {gGattDbInvalidHandle_d};

/* server values for its own service changed characteristic and CCCD handles */
    uint16_t mServerServiceChangedCharHandle;
    uint16_t mServerServiceChangedCCCDHandle;

/* client state information for bonded and active clients */
    gattCachingClientState_c gGattClientState[gMaxBondedDevices_c] = {gGattClientChangeAware_c};
    gattCachingClientState_c gGattActiveClientState[gAppMaxConnections_c] = {gGattClientChangeAware_
↪c};

/* Database hash values - the client needs a hash value for each possible peer */
    uint8_t mGattActiveServerDatabaseHash[gGattDatabaseHashSize_c * gAppMaxConnections_c] = {0};
    uint8_t mGattServerDatabaseHash[gGattDatabaseHashSize_c * gMaxBondedDevices_c] = {0};

/* client supported features handles for active gatt servers */
    uint16_t gGattActiveClientSupportedFeaturesHandles[gAppMaxConnections_c] =
↪{gGattDbInvalidHandle_d};

/* client supported features information for bonded gatt clients */
    uint8_t gGattClientSupportedFeatures[gMaxBondedDevices_c] = {0U};

/* index of the database hash characteristic in the database */
    uint32_t mServerDatabaseHashIndex = gGattDbInvalidHandleIndex_d;

/* index of the client supported features characteristic in the database */
    uint32_t mServerClientSupportedFeatureIndex = gGattDbInvalidHandleIndex_d;
```

It is up to the application to save a local copy of the information from the server's database and to initiate service discovery only on the first connection or when it is informed of a change by the peer using Service Changed and Robust Caching.

If the *gGattAutomaticRobustCachingSupport_c* define is not set, it is up to the application to check the Server Supported Features characteristic value on the peer, to write the Client Supported Features characteristic value and to write the Service Changed CCCD.

Two new GATT procedures are introduced as part of the Robust Caching feature. Both should be treated according to the application needs in the GATT procedure callback of the application.

- *gGattProcSignalServiceDiscoveryComplete_c –*informs the application that the service discovery procedure has finished after reading the Database Hash value using the read using characteristic UUID procedure. The application procedure callback should call *BleServDisc_Finished()*on this event when robust caching is supported.
- *gGattProcUpdateDatabaseCopy_c –*informs the application that its database copy is no longer up to date and service discovery should be reperformed. Used when the client received an error response with the *gAttErrCodeDatabaseOutOfSync_c* opcode, when the local database hash value is found to be out of sync with the one on the server, or when a service changed indication is received from the server.

If service discovery is performed using our ble_service_discovery module, then the application should wait for the *gDiscoveryFinished_c* event before initiating its own GATT procedures. The application should also make sure to not initiate a second GATT procedure which requires a response from the peer before receiving a response to the first request it made.

Parent topic: Gatt caching

Parent topic:*Creating GATT database*

Creating a Custom Profile

This chapter describes how the user can create customizable functionality over the Bluetooth Low Energy Host Stack by defining profiles and services. The Temperature Profile, used by the

Temperature Sensor and Collector applications, is used as a reference to explain the steps of building custom functionality.

Defining custom UUIDs

The first step when defining a new service included in a profile is to define the custom 128-bit UUID for the service and the included characteristics. These values are defined in *gatt_uuid128.h*, which is located in the application folder. For example, the Temperature Profile uses the following UUID for the service:

```
/* Temperature */
UUID128(uuid_service_temperature, 0xfb ,0x34 ,0x9b ,0x5f ,0x80 ,0x00, 0x00, 0x80, 0x00, 0x10, 0x00, 0x02,␣
↪0x00, 0xfe, 0x00, 0x00)
```

The definition of the services and characteristics are made in *gattdb.h*, as explained in *Creating GATT database*. For more details on how to structure the database, see *Application Structure*.

Parent topic:*Creating a Custom Profile*

Creating service functionality

All defined services in the SDK have a common template which helps the application to act accordingly.

The service locally stores the device identification for the connected client. This value is changed on subscription and non-subscription events.

```
/*! Temperature Service - Subscribed Client*/
static deviceId_t mTms_SubscribedClientId;
```

The service is initialized and changed by the application through a service configuration structure. It usually contains the service handle, initialization values for the service (for example, the initial temperature for the Temperature Service) and in some cases user-specific structures that can store saved measurements (for example, the Blood Pressure Service). Below is an example for the custom Temperature Service:

```
/*! Temperature Service - Configuration */
typedef struct tmsConfig_tag
{
    uint16_t serviceHandle ;
    int16_t initialTemperature ;
} tmsConfig_t ;
```

The initialization of the service is made by calling the start procedure. The function requires as input a pointer to the service configuration structure. This function is usually called when the application is initialized. It resets the static device identification for the subscribed client and initializes both dynamic and static characteristic values. An example for the Temperature Service (TMS) is shown below:

```
bleResult_t Tms_Start ( tmsConfig_t *pServiceConfig)
{
    mTms_SubscribedClientId = gInvalidDeviceId_c;
    return Tms_RecordTemperatureMeasurement (pServiceConfig-> serviceHandle ,
                                             pServiceConfig-> initialTemperature );
}
```

The service subscription is triggered when a device connects to the server. It requires the peer device identification as an input parameter to update the local variable. On disconnect, the unsubscribe function is called to reset the device identification. For the Temperature Service:

```
bleResult_t Tms_Subscribe ( deviceId_t deviceId)
{
    mTms_SubscribedClientId = deviceId;
    return gBleSuccess_c;
}
bleResult_t Tms_Unsubscribe (void)
{
    mTms_SubscribedClientId = gInvalidDeviceId_c;
    return gBleSuccess_c;
}
```

Depending on the complexity of the service, the API implements additional functions. For the Temperature Service, there is only a temperature characteristic that is notifiable by the server. The API implements the record measurement function which saves the new measured value in the GATT database and send the notification to the client device if possible. The function needs the service handle and the new temperature value as input parameters:

```
bleResult_t Tms_RecordTemperatureMeasurement ( uint16_t serviceHandle, int16_t temperature)
{
    uint16_t handle;
    bleResult_t result;
    bleUuid_t uuid = Uuid16(gBleSig_Temperature_d);
    /* Get handle of Temperature characteristic */
    result = GattDb_FindCharValueHandleInService(serviceHandle,
        gBleUuidType16_c, &uuid, &handle);
    if (result != gBleSuccess_c)
        return result;
    /* Update characteristic value */
    result = GattDb_WriteAttribute(handle, sizeof( uint16_t ), ( uint8_t *)&temperature);
    if (result != gBleSuccess_c)
        return result;
    Hts_SendTemperatureMeasurementNotification(handle);
    return gBleSuccess_c;
}
```

To accommodate some use cases where the service is reset, the stop function is called. The reset also implies a service unsubscribe. Below is an example for the Temperature Service:

```
bleResult_t Tms_Stop ( tmsConfig_t *pServiceConfig)
{
    return Tms_Unsubscribe();
}
```

Parent topic:*Creating a Custom Profile*

GATT client interactions

The client side of the service, which includes the service discovery, notification configuration, attribute reads and others are left to be handled by the application. The application calls the GATT client APIs and reacts accordingly. The only exception for this rule is that the service interface declares the client configuration structure. This structure usually contains the service handle and the handles of all the characteristic values and descriptors discovered. Additionally it can contain values that the client can use to interact with the server. For the Temperature Service client, the structure is as follows:

```
/*! Temperature Client - Configuration */
typedef struct tmcConfig_tag
{
    uint16_t            hService;
    uint16_t            hTemperature ;
    uint16_t            hTempCccd ;
```

(continues on next page)

(continued from previous page)

```
    uint16_t                hTempDesc ;
    gattDbCharPresFormat_t    tempFormat ;
} tmcConfig_t ;
```

Parent topic:*Creating a Custom Profile*

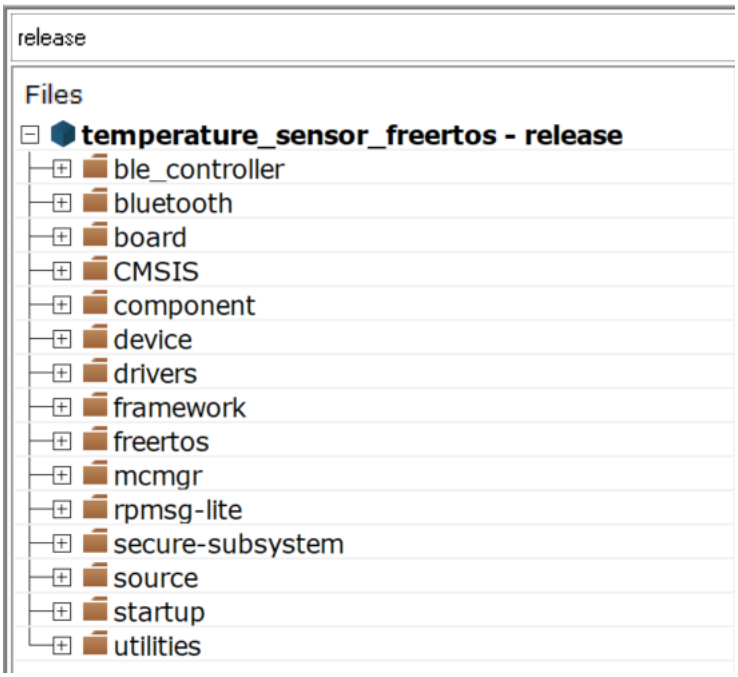
Application Structure

This chapter describes the organization of the Bluetooth Low Energy demo applications that can be found in the SDK. By familiarizing with the application structure, the user is able to quickly adapt its design to an existing demo or create a new application.

The Temperature Sensor application is used as a reference to showcase the architecture.

Folder structure

The Figure shows the application folder structure.

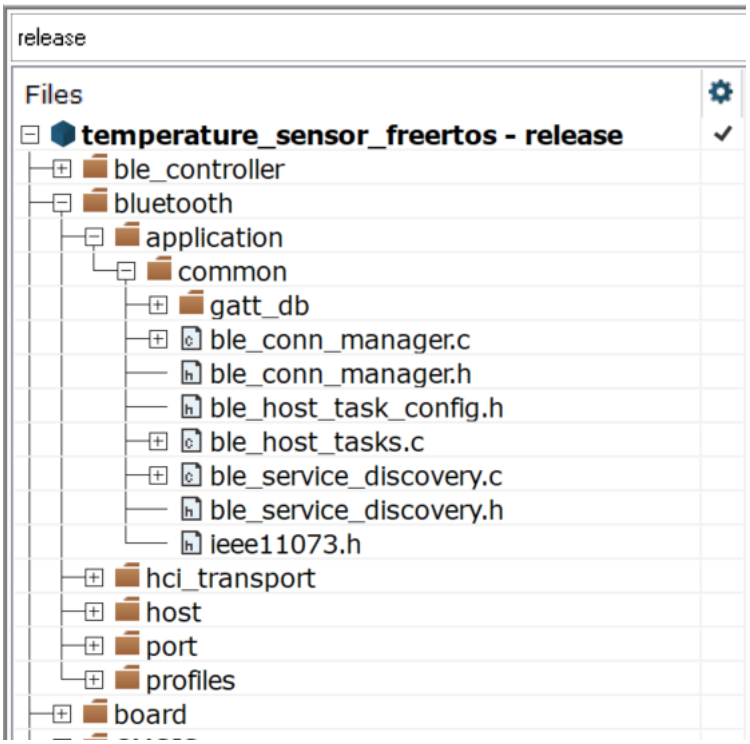


Application Folder structure in workspace

The *app* folder follows a specific structure which is recommended for any application developed using the Bluetooth Low Energy Host Stack:

- The *common* group contains the application framework shared by all profiles and demo applications:
  - Bluetooth Low Energy Connection Manager
  - Bluetooth Service Discovery Manager
  - Bluetooth Low Energy Stack and Task Initialization and Configuration
  - GATT Database

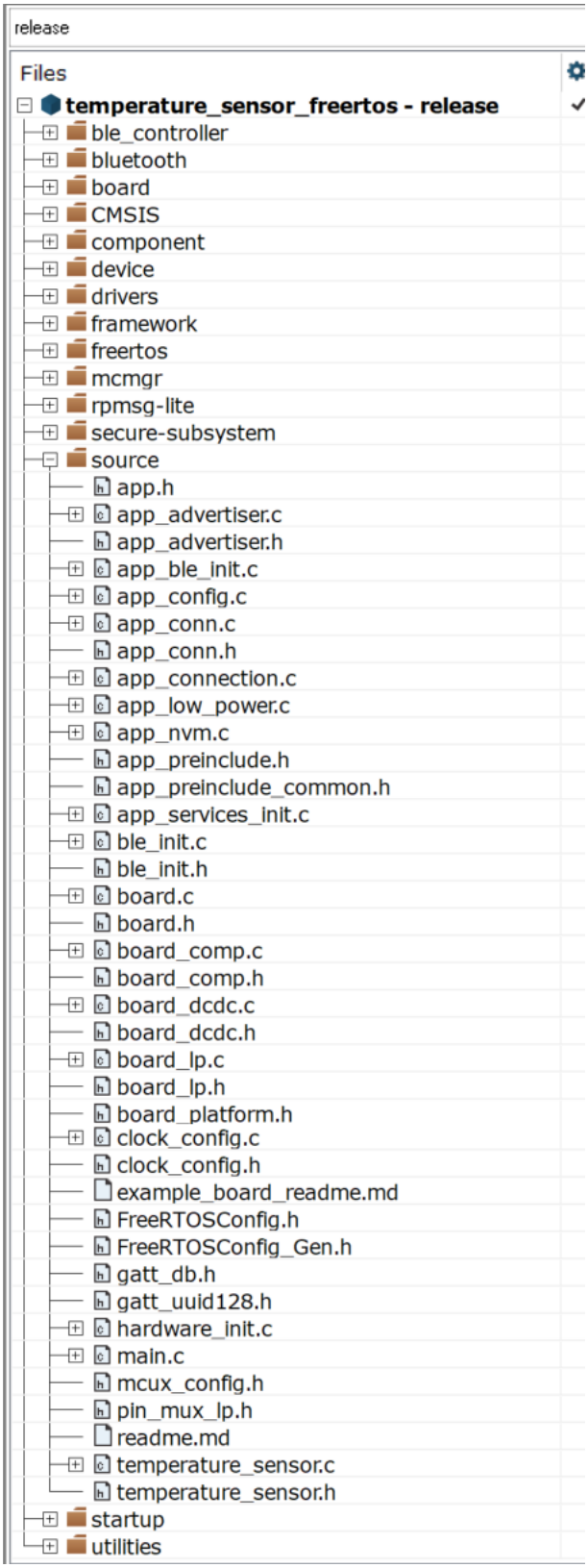
Common Group



- The *source* group contains code specific to the Temperature Sensor application.

  In the following examples, *app.c* is used as a placeholder for the main application source file. In the case of Temperature Sensor, it is *temperature_sensor.c*.

  The source group also contains files which aid with the handling of Host events and framework related functionality. These are: *app_conn.c*/*app_conn.h*, *app_advertiser.c*, *app_connection.c*, *app_scanner.c*, *app_nvm.c*, and *app_low_power.c*. These files do not allow the application to implement its own state machines, unless application-specific functionality is required. For example, the application is restricted from setting advertising parameters and starting advertising, unless certain application-specific functionalities, such as UI related updates are required.



Source Group

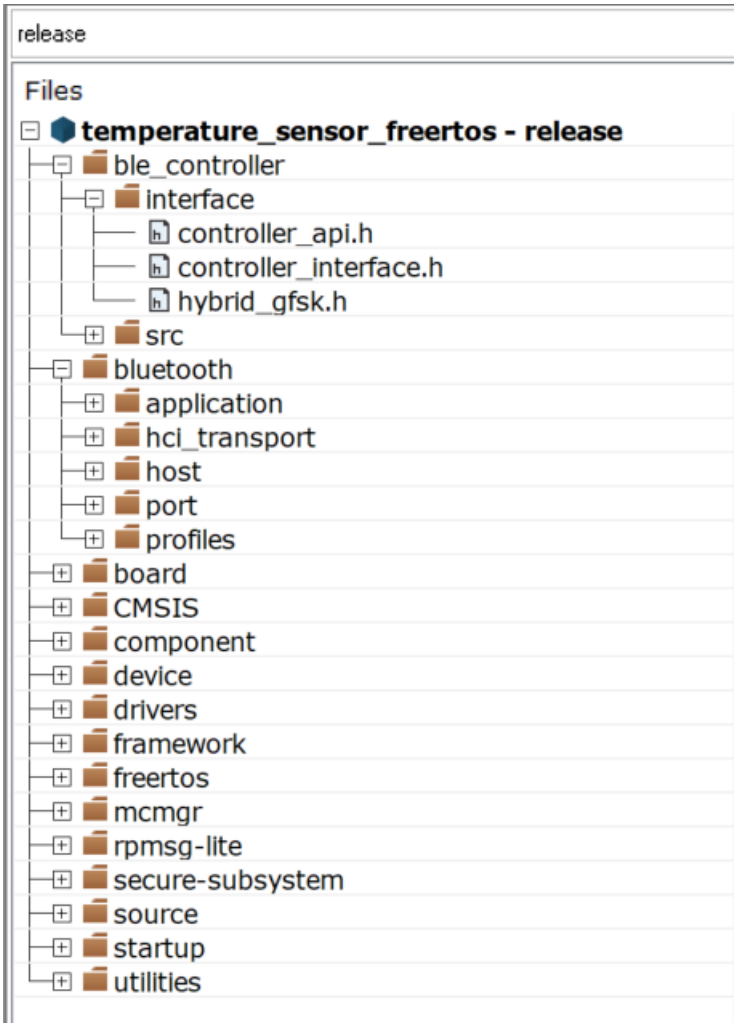
The *bluetooth* folder/group contains:

- The *host/interface*, and *host/config*. These are public interfaces and configuration files for the Host. The functionality is included in the library located in the *host/lib* subfolder. The folder is not shown in the IAR project structure, but added into the toolchain linker settings under the library category.
- *bluetooth/profiles* contains profile-specific code; it is used by each demo application of stan-

dard profiles.

The *ble_controller* folder/group contains:

- The *controller/interface*. These are public interfaces and configuration files for the Controller.



Bluetooth and Controller Groups

The framework and component folders/groups contain framework components used by the demo applications. For additional information, see the *Connectivity Framework Reference Manual*.

The *freertos* folder contains sources for the supported operating system.

Parent topic:*Application Structure*

Application main framework

The Application Main module contains common code used by all the applications, such as:

- The Main Task
- Messaging framework between the Bluetooth LE Host Stack Task and the Application Task

Start task

The Start Task (*start_task*) is the first task created by the operating system and is also the one that initializes the rest of the system. It initializes framework components (Memory Manager, Timers Manager etc.) and it calls *BluetoothLEHost_AppInit* from *app.c*, which is used to initialize the Bluetooth LE Host Stack as well as peripheral drivers specific to the implemented application.

The function calls *BluetoothLEHost_HandleMessages*, which represents the Application Task and is used to process events and messages from the Host Stack.

The stack size and priority of the main task are defined in *fsl_os_abstraction_config.h*:

```
#ifndef gMainThreadStackSize_c
#define gMainThreadStackSize_c 1024
#endif
#ifndef gMainThreadPriority_c
#define gMainThreadPriority_c 7
#endif
```

Parent topic: Application main framework

Application messaging

The module contains a wrapper that is used to create messages for events generated by the Bluetooth LE Host Stack in the Host Task context. The wrapper also sends them to be processed by the application in the context of the Application Task.

For example, connection events generated by the Host are received by *App_ConnectionCallback*. The function creates a message, places it in the Host to Application queue and signals the Application with *gAppEvtMsgFromHostStack_c*. The Application Task de-queues the message and calls *App_HandleHostMessageInput*, which calls the corresponding callback implemented the application-specific code (*app.c*), in this example: *BleApp_ConnectionCallback*.

It is strongly recommended that the application developer uses the *app.c* module to add custom code on this type of callbacks.

Parent topic: Application main framework

Parent topic:*Application Structure*

Bluetooth LE Connection Manager

The connection manager is a helper module that contains common application configurations and interactions with the Bluetooth LE Host Stack. It implements the following events and methods:

- Host Stack GAP Generic Event
- Host Stack Connection Event on both GAP Peripheral and GAP Central configuration
- Host Stack configuration for GAP Peripheral or GAP Central

GAP generic event

The GAP Generic Event is triggered by the Bluetooth LE Host Stack and sent to the application via the generic callback. Before any application-specific interactions, the Connection Manager callback is called to handle common application events, such as device address storage.

```
void BleApp_GenericCallback ( gapGenericEvent_t * pGenericEvent)
{
    /* Call Bluetooth Low Energy Conn Manager */
    BleConnManager_GenericEvent(pGenericEvent);
    switch (pGenericEvent-> eventType )
    {
        ...
    }
}
```

In the **BleConnManager_GenericEvent** function, local keys are generated.

- The local LTK, IRK, as well as the EDIV and RAND are obtained hashing over the board's UID and stored in RAM as plain-text every time the **gInitializationComplete_c** event is received.
- In **Advanced Secure** mode, local IRK is generated using the EdgeLock Secure Enclave and stored into a dedicated NVM data set as ELKE blobs (40 bytes blob encrypted using unique die key) on the first gInitializationComplete_c event received.

The NBU Decryption key for IRK is generated and distributed to the NBU over the private key bus. The EIRK blob (16 bytes blob which can be decrypted only by NBU hardware using NBU Decryption key for IRK) is generated from the IRK ELKE blob and stored in the RAM to be used for controller privacy on every gInitializationComplete_c event received. For the host privacy, the ELKE IRK blob is used instead. For details, refer to the section "Advanced security capabilities".

Parent topic: Bluetooth LE Connection Manager

GAP configuration

The GAP Central or Peripheral Configuration is used to create common configurations (such as setting the public address, registering the security requirements, adding the addresses of bonded devices in the Controller Filter Accept List), which can be customized by the application afterwards. It is called inside the *BluetoothLEHost_Initialized callback* function, before any application-specific configuration, as shown in the example code below.

```
static void BluetoothLEHost_Initialized()
{
    /* Set common GAP configuration */
    BleConnManager_GapCommonConfig();
    ...
}
```

Parent topic: Bluetooth LE Connection Manager

GAP connection event

The GAP Connection Event is triggered by the Host Stack and sent to the application via the connection callback. Before any application-specific interactions, the Connection Manager callback is called to handle common application events, such as device connect, disconnect or pairing-related requests. It is called inside the registered connection such as shown below:

```
static voidBle App_ConnectionCallback ( deviceId_t peerDeviceId, gapConnectionEvent_t *␣
↪pConnectionEvent)
{
    /* Connection Manager to handle Host Stack interactions */
    BleConnManager_GapPeripheralEvent(peerDeviceId, pConnectionEvent);
    switch (pConnectionEvent-> eventType )
    {
        ...
    }
}
```

It is strongly recommended that the application developer uses the app.c module to add custom code.

Parent topic: Bluetooth LE Connection Manager

Privacy

To enable or disable Privacy, the following APIs may be used:

```
bleResult_t
 BleConnManager_EnablePrivacy(void);
```

```
bleResult_t
 BleConnManager_DisablePrivacy(void);
```

The function BleConnManager_EnablePrivacy calls *BleConnManager_ManagePrivacyInternal* after checking if the privacy is enabled.

```
static bleResult_t
BleConnManager_ManagePrivacyInternal
(bool_t bCheckNewBond);
```

If the privacy feature is supported (*gAppUsePrivacy_d = 1*), the Connection Manager activates Controller Privacy or Host Privacy depending on the board capabilities.

The *bCheckNewBond* is a boolean that tells the Manager whether it should check or not if a bond between the devices already exists.

In order to update the identity information after a bond is added or removed privacy should be disabled and enabled. For pairing with bonding this is done automatically in *ble_conn_manager*. In case the application adds or removes a bond through the GAP API, it should also disable and enable privacy.

Parent topic: Bluetooth LE Connection Manager

Parent topic:*Application Structure*

GATT database

The *gatt_db* contains a set of header files grouped in the *macros* subfolder. These macros are used for static code generation for the GATT Database by expanding the contents of the *gatt_db.h* file in different ways. *Creating GATT database* explains how to write the *gatt_db.h* file using user-friendly macros that define the GATT database.

At application compile time, the *gatt_database.c* file is populated with enumerations, structures, and initialization code used to allocate and properly populate the GATT database. In this way, the *gattDatabase*array and the *gGattDbAttributeCount_c* variable (see GATT database) are created and properly initialized.

Note: Do not modify any of the files contained in the *gatt_db* folder and its subfolder.

To complete the GATT database initialization, this demo application includes the required gatt_db.h and gatt_uuid128.h files in its specific application folder, along with other profile-specific configuration and code files.

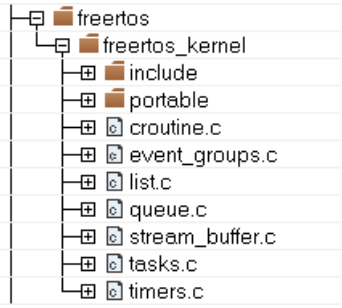
Parent topic:*Application Structure*

RTOS specifics

Operating system selection

The SDK offers different projects for each supported operating system (FreeRTOS OS) and for BareMetal configuration. To switch between systems, the user needs to switch the workspace.

The RTOS source code is found in the SDK package and is linked in the workspace in the *freertos* virtual folder, as shown in Figure:



Location of FreeRTOS source code in workspace

Parent topic:RTOS specifics

Bluetooth LE Host task configuration

Application developers are provided with two files for RTOS task initialization:

- *ble_host_task_config.h*, and *ble_host_tasks.c* for the Host.

Reusing these files is recommended because they perform all the necessary RTOS-related work. The application developer must only modify the macros from **_config.h* files whenever tasks need a bigger stack size or different priority settings. The new values should be overridden in the *app_preinclude.h* file.

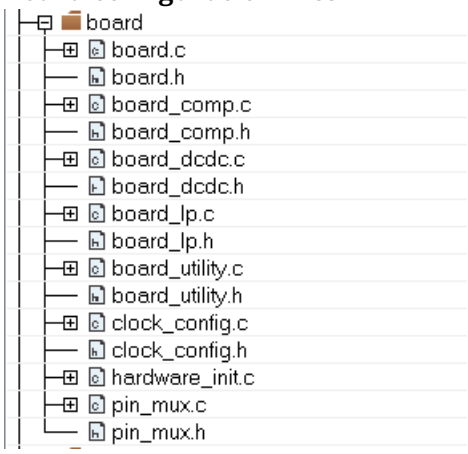
Parent topic:RTOS specifics

Parent topic:*Application Structure*

Board configuration

The configuration files for the supported boards can be found in the *board* folder, as shown in Figure. The files contain clock and pin configurations that are used by the drivers. The user can customize the board files by modifying the configuration of the pins and clock source according to his design.

Board configuration files



Parent topic:*Application Structure*

Bluetooth Low Energy initialization

The *ble_init.h* and *ble_init.c* files contain the declaration and the implementation of the following function:

```
bleResult_t Ble_Initialize
(
    gapGenericCallback_t gapGenericCallback
)
{
```

(continues on next page)

(continued from previous page)

```
#if defined(gUseHciTransportDownward_d) && gUseHciTransportDownward_d
    /* HCI Transport Init */
    if (gHciSuccess_c != Hcit_Init(Ble_HciRecvFromIsr))
    {
        return gHciTransportError_c;
    }

    /*
     * Set BD Address in Controller. Must be done after HCI init
     * and before Host init.
     */
      Ble_SetBDAddr();

    /* Check for available memory storage */
    if (!Ble_CheckMemoryStorage())
    {
        return gBleOutOfMemory_c;
    }
    /* Bluetooth Low Energy Host Tasks Init */
    if (KOSA_StatusSuccess != Ble_HostTaskInit())
    {
        return gBleOsError_c;
    }
    /* Bluetooth Low Energy Host Stack Init */
    return Ble_HostInitialize(gapGenericCallback, Hcit_SendPacket);
#elif defined(gUseHciTransportUpward_d) && gUseHciTransportUpward_d
#else /* gUseHciTransportUpward_d */
#endif /* gUseHciTransportUpward_d */
}
```

Note: This function should be used by your application because it correctly performs all the necessary Bluetooth Low Energy initialization.

Step-by-step analysis is provided below:

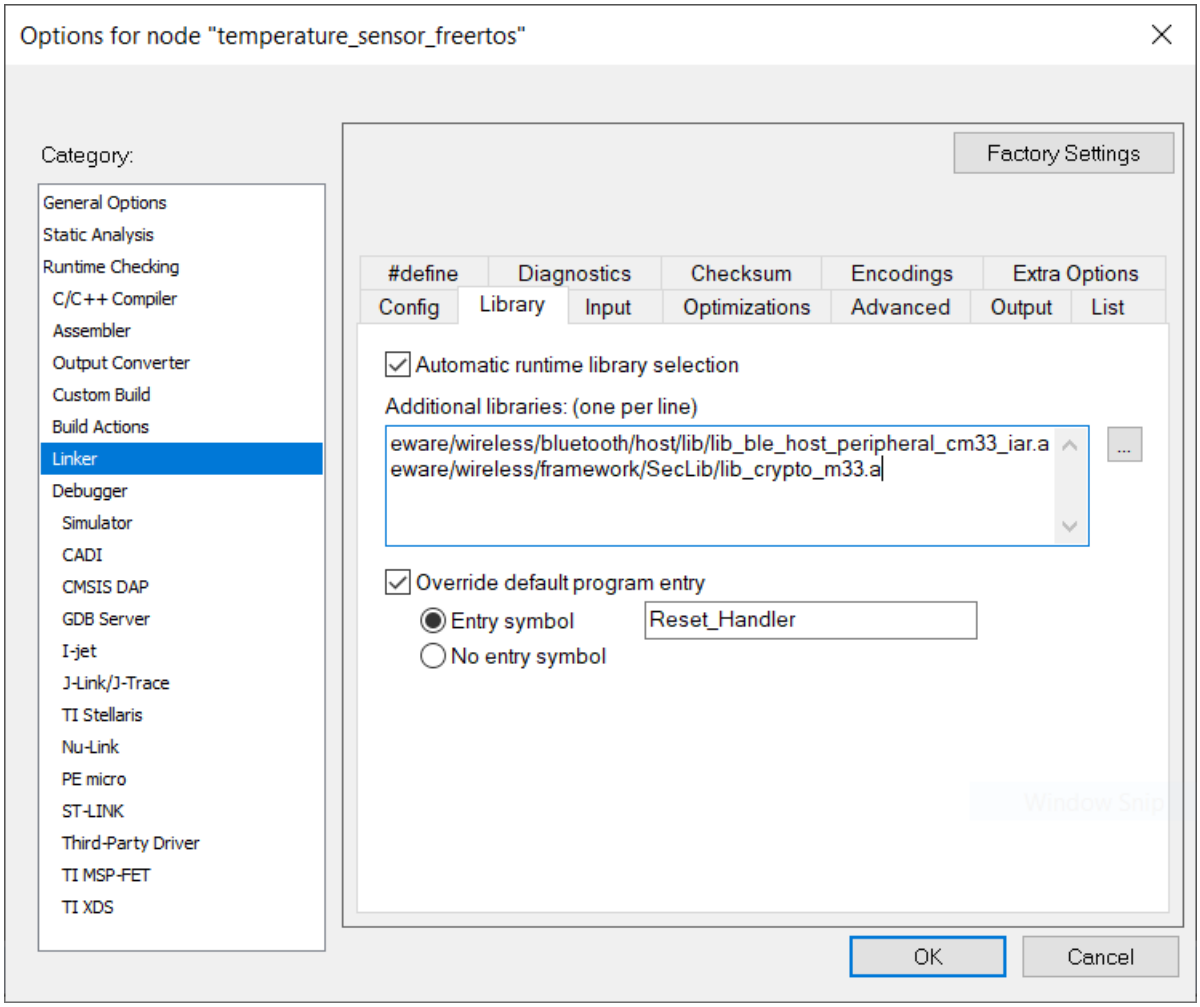
1. First, the HCI interface is initialized by calling *Hcit_Init*. This initializes communication between the Host and the Controller.
2. After setting the BD address into the Controller (Ble_SetBDAddr) and performing memory validation checks (*Ble_CheckMemoryStorage*), the Host task is initialized by calling *Ble_HostTaskInit*.
3. Finally the *Ble_HostInitialize* function initializes the Host with the transport packet transmit function used as the *hciHostToControllerInterface_t parameter*.

Parent topic:*Application Structure*

Bluetooth Low Energy Host Stack configuration

The Bluetooth LE Host Stack libraries are found in the middleware/wireless/bluetooth/host/lib folder. The user should add the best matching library for its use case to the linker options of its project.

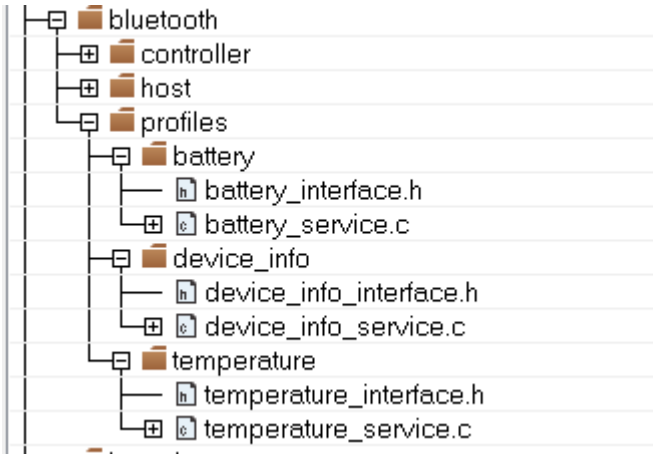
For example, the temperature sensor uses the Peripheral Host Stack library, as shown in Figure below:



Parent topic:*Application Structure*

Profile configuration

The implemented profiles and services are located in *middleware/wireless/bluetooth/profiles* folder. The application links every service source file and interface it needs to implement the profile. For example, for the Temperature Sensor the tree looks as shown Figure:



The Temperature Profile implements the custom Temperature service, the Battery, and Device Information services.

Application code

The application folder contains the following modules:

- *app.c* and *app.h*. This module stores the application-specific functionality (APIs for specific triggers, handling of peripherals, callbacks from the stack, handling of low power, and so on).

Before initializing the Bluetooth LE Host Stack, the start task calls *BluetoothLEHost_AppInit*. This function initializes application specific functionality before initializing the Bluetooth LE Host Stack by calling *BluetoothLEHost_Init*.

After the stack is initialized, the *BluetoothLEHost_Initialized* callback is called. The function contains configurations made to the Bluetooth LE Host Stack after the initialization. This includes registering callbacks, setting security for services, starting services, allocating timers, adding devices to the Filter Accept List, and so on. For example, the Temperature Sensor configures the following:

```
static void BluetoothLEHost_Initialized(void)
{
  /* Common GAP configuration */
    BleConnManager_GapCommonConfig();

  /* Register for callbacks*/
    (void)App_RegisterGattServerCallback(BleApp_GattServerCallback);

    mAdvState.advOn = FALSE;

  /* Start services */
    SENSORS_TriggerTemperatureMeasurement();
    (void)SENSORS_RefreshTemperatureValue();
    /* Multiply temperature value by 10. SENSORS_GetTemperature() reports temperature
    value in tenths of degrees Celsius. Temperature characteristic value is degrees
    Celsius with a resolution of 0.01 degrees Celsius (GATT Specification
    Supplement v6). */
    tmsServiceConfig.initialTemperature = (int16_t)(10 * SENSORS_GetTemperature());
    (void)Tms_Start(&tmsServiceConfig);

    basServiceConfig.batteryLevel = SENSORS_GetBatteryLevel();
    (void)Bas_Start(&basServiceConfig);
    (void)Dis_Start(&disServiceConfig);

  /* Allocate application timer */
    (void)TM_Open(appTimerId);

    AppPrintString("\r\nTemperature sensor -> Press switch to start advertising.\r\n");
}
```

To start the application functionality, BleApp_Start() function is called. This function usually contains code to start advertising for sensor nodes or scanning for central devices. In the example of the Temperature Sensor, the function is the following:

```
static void BleApp_Start(void)
{
    Led1On();

    if (mPeerDeviceId == gInvalidDeviceId_c)
    {
      /* Device is not connected and not advertising */
        if (!mAdvState.advOn)
        {
          /* Set advertising parameters, advertising to start on gAdvertisingParametersSetupComplete_c */
          BleApp_Advertise();
```

(continues on next page)

(continued from previous page)

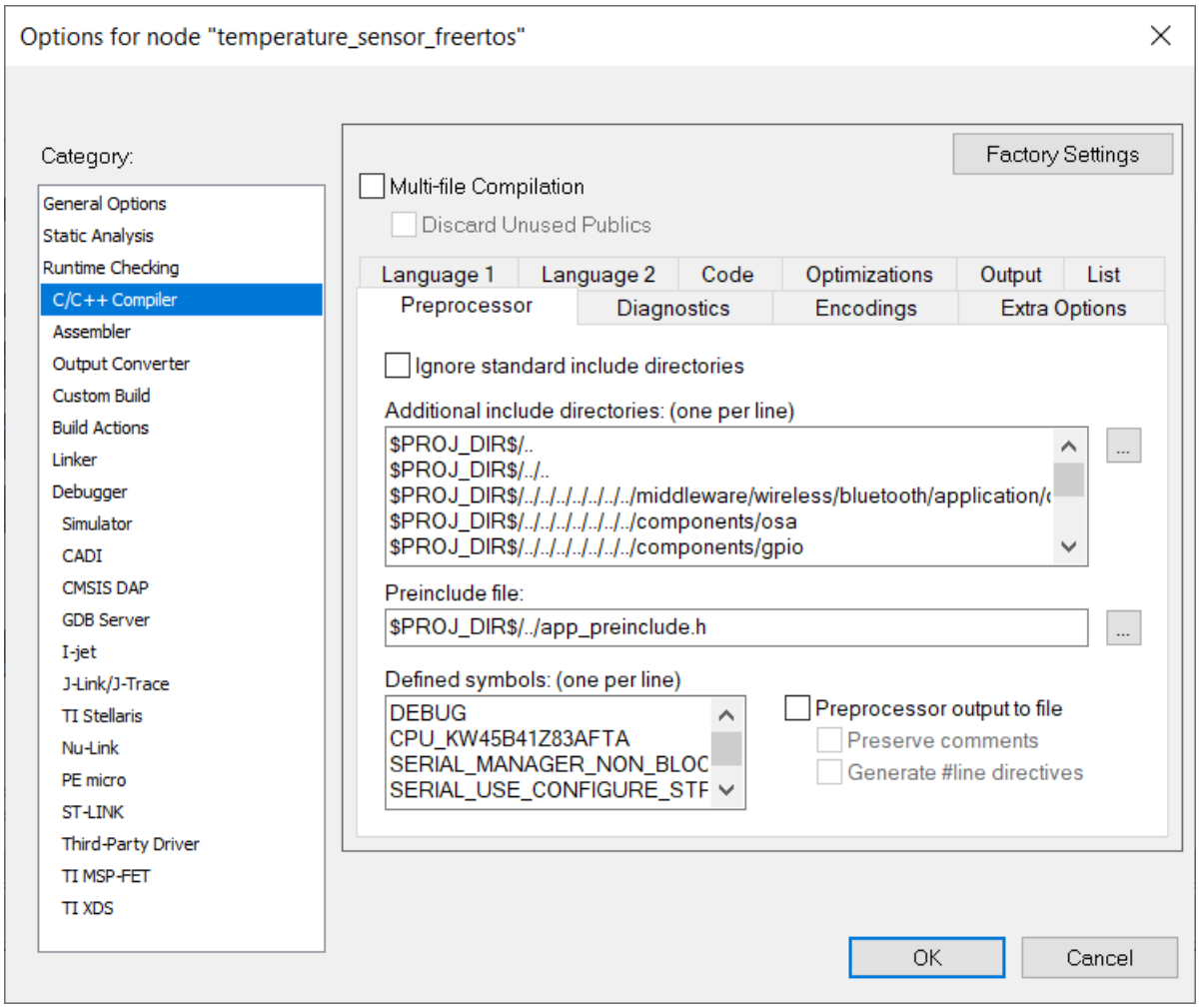
```
        }
    }
    else
    {
        /* Device is connected, send temperature value */
        BleApp_SendTemperature();
    }
}
```

- *app_config.c.* This file contains data structures that are used to configure the stack.

This includes advertising data, scanning data, connection parameters, advertising parameters, SMP keys, security requirements, and so on.

- *app_preinclude.h.*

This header file contains macros to override the default configuration of any module in the application. It is added as a preinclude file in the preprocessor command line in IAR, as shown in Figure:



- *gatt_db.h* and *gatt_uuid128.h.* The two header files contain the definition of the GATT database and the custom UUIDs used by the application. See *Creating GATT database* for more information.

Parent topic: Profile configuration

Parent topic:*Application Structure*

Multiple connections

Applications can be configured to support multiple connections. To allow multiple connections, the *gAppMaxConnections_c* must be set to a value up to the maximum number of connections (*this value is chip-specific*). Refer to the chip documentation for the supported number of connections.

The application can save information about the peer devices it connects to according to the value of *gAppMaxConnections_c*. The Bluetooth Low Energy profile associated with the application use case must be instantiated to support the use of its functionality for each peer device. When handling multiple connections, the applications can behave as either the GAP central, GAP peripheral, or both at the same time. It is up to the application code to decide whether to start the advertising or scanning before creating the next connection. The supported combinations enable a device to connect as a peripheral to multiple centrals, as a central to multiple peripherals, or for it to be a central for some peers and a peripheral to others. The demo applications provide this functionality as an example of exercising multiple connection support. In such applications, the GAP role can be changed from central to peripheral and the information is saved for each peer device.

Parent topic:*Application Structure*

Bluetooth address generation

BD_ADDR is represented by 48 bits that uniquely identify a device and consist of a 24-bit OUI (Organizationally Unique Identifier) and a 24-bit random part that varies between devices. There are multiple options of storing and using the BD_ADDR. Depending on the chip, it may be read from a device specific register (if supported), from the global hardware parameters stored in the flash, or generated randomly based on the processor-unique identifier. The demo applications provide a combination of the last two options. The *Ble_SetBDAddr* function is called during the initialization process, after initializing HCI and before initializing the Bluetooth LE Host Stack. The global hardware parameters are read from the flash. If a useful value is found, it is used as the BD address. If the found value is all 0xFFs, an address is generated by concatenating the OUI configured at compile time with three randomly generated bytes. The result is stored in the hardware parameters for future use and then set into the Controller. The Bluetooth LE Host Stack uses little-endian format to represent all addresses, in compliance with the Bluetooth Core Specification.

Parent topic:*Application Structure*

Repeated attempts

Applications can be configured to enable protection against repeated Pairing Requests/Peripheral Security Requests coming from the same device. This is to prevent an intruder from repeating the pairing process with a large number of different keys in order to extract information about the local device's private key. If this feature is enabled, after a pairing procedure fails, another attempt to pair coming from the same device is allowed only after a specific time period has passed. For each failure, the waiting period doubles up until a maximum period.

The following *app_preinclude.h* macros support this feature:

- gRepeatedAttempts_d
  - Set to 1 to enable the feature. By default, it is disabled (0).
- gRepeatedAttemptsNoOfDevices_c
  - Number of remote devices to keep track of. By default, the value is 4.
  - If a new device needs to be added and the list is full, one of the oldest entries will be replaced.
- gRepeatedAttemptsTimeoutMin_c

  - Minimum waiting period in seconds – default 10.

- gRepeatedAttemptsTimeoutMax_c
  - Maximum waiting period in seconds – default 640. The waiting period doubles after each failed pairing with the same device.

Parent topic:*Application Structure*

Advanced Secure Mode

This section describes the advanced security capabilities of the Bluetooth LE Host Stack which are available on the KW45/KW47/MCXW72/KW43 platform via the EdgeLock Secure Enclave (ELKE).

The security capabilities are enabled at application, Host and Controller level by setting Advanced Secure Mode to active. To do this, the user must set the *gAppSecureMode_d* macro to *1* in the project's *app_preinclude.h* file. This macro is defined by default as *0* in *app_preinclude_common.h*:

```
#if (gAppSecureMode_d == 1U)
#define gHostSecureMode_d              (1U)
#else
#define gHostSecureMode_d              (0U)
#endif
```

At application level, when Advanced Secure Mode is enabled, the security mode and level for pairing is automatically enforced as Mode 1 Level 4, ensuring LE Secure Connections pairing. Legacy pairing is not supported in this mode.

When enabled, the main benefit of Advanced Secure Mode is the secured storage and handling of Bluetooth LE security keys. The EdgeLock Secure Enclave is capable of generating, importing, and exporting security keys as plain text or as encrypted blobs. All encrypted blobs are created by the EdgeLock Secure Enclave using a die unique key, which makes them impossible to decrypt by devices other than the one that created them. The Bluetooth LE security keys are managed in Advanced Secure Mode as follows:

- IRK
  - The peer IRKs received after pairing and bonding are no longer stored into NVM as plaintext, but as encrypted blobs 40 bytes in length (or ELKE blobs). They can still be retrieved and converted to plaintext.
  - The local IRK is no longer generated using the default method of hashing over the board's UID at every startup. It is instead generated once using the EdgeLock Secure Enclave and stored into a new NVM dataset as an ELKE blob.
  - Local and peer IRKs are no longer transmitted through HCI in plaintext but as EIRK (Encrypted IRK) blobs, 16 bytes in length, which can be decrypted by the Controller.
- LTK
  - The LTK is no longer stored into NVM as plaintext, but as an ELKE blob. Furthermore, the plaintext of the LTK is never available to the Host/application. Generating the LTK via the ECDH process and generating the Session Key for individual connections is done by the Host via the EdgeLock Secure Enclave and custom vendor HCI messages which are transparent to the application.

At startup, Advanced Secure Mode for the Controller is enabled dynamically by calling:

```
#if (defined(gAppSecureMode_d) && (gAppSecureMode_d > 0U))
    (void) PLATFORM_EnableBleSecureKeyManagement();
#endif
```

This call can be found in BluetoothLEHost_Init, as part of the initialization sequence.

Additional considerations

- The Gap_LoadKeys API will return the IRK in plaintext, having converted it from the ELKE blob which is stored in the NVM. However, it cannot convert the LTK to plaintext - it will thus return the LTK in ELKE blob form.
- As mentioned in the section above, the plaintext LTK is never available. Generating the Session Key for individual connections is done via the EdgeLock Secure Enclave by the Host, using the LTK in ELKE blob form. The Session Key is passed to the Controller via a custom vendor HCI message. Be aware that the content of the HCI_LE_Long_Term_Key_Request_Reply command is not the actual LTK and cannot be used for debugging.
- The Host works with IRKs in ELKE blob form, therefore APIs such as Gap_EnableHostPrivacy and Gap_CreateRandomAddress take ELKE blobs as arguments. The Controller works with IRKs in EIRK blob form, therefore the Gap_EnableControllerPrivacy takes an EIRK blob as argument. This management is done in *ble_conn_manager.c* when Advanced Secure Mode is enabled.

Parent topic:*Application Structure*

Low-Power Management

System considerations

The KW45 has a dual-core architecture and has two separated power domains:

- The main domain for the Cortex M33
- The Radio domain which comprises the Cortex M3 core and the NBU (Narrow Band Unit).

The two power domains can go into or exit independently different low-power modes, namely Wait for instruction (WFI), Deep-sleep mode, Power-down mode, and Deep Power-down mode.

In Wait For Interrupt (WFI) mode, the CPU core is powered ON but is in an idle mode with the clock turned OFF.

In Deep Sleep mode, the fast clock is turned off, and the CPU along with the main power domain are placed into a retention state, with the voltage being scaled down to support state retention only. As no high frequency clock runs in this mode, the voltage applied on the power domain can be reduced to reduce leakage on the hardware logic. This reduces the overall power consumption in the Deep Sleep mode. When waking up from Deep Sleep mode, the core voltage is increased back to nominal voltage, the fast clock (FRO) is turned back on, and the peripheral in this domain can be reused as normal.

In Power-down mode, both the clock, and power are shut off to the CPU and the main peripheral domain. SRAM is retained, but register values are lost. The SDK power manager handles the restore of the processor registers and dependencies such as interrupt controller and similar ones transparently from the application.

In Deep Power-down mode the SRAM is not retained. This is the lowest power mode available. It is exited through the reset sequence.

Parent topic:*Low-Power Management*

When/how to enter low power

To enable low power at application level, the `gAppLowpowerEnabled_d` define should be set to `1` in `app_preinclude.h` file.

The system should enter low power when the entire system is idle, and all software layers agree on that. The device enters low power by calling the `PWR_EnterLowPower` function.

For FreeRTOS applications, the low-power entry point is placed in the FreeRTOS idle task, which has the lowest priority in the system. From that task, the `vPortSuppressTicksAndSleep` function is called, which at its turn, calls the `PWR_EnterLowPower` to enter low power.

For the BareMetal examples, the application low power entry point is placed in the main function and is called when there are no messages to be processed by other tasks.

The wake-up sources that can be configured for the application are UART or button. Note that Low-power timer wake-up source and wake-up from the Radio domain are directly enabled from the Connectivity framework.

Each software layer/entity running on the system can prevent it from entering low power by calling *PWR_LowPowerEnterCritical* function. The system stays awake until all software layers that called *PWR_LowPowerEnterCritical* call back *PWR_LowPowerExitCritical* and the system reaches the low-power entry point.

When going to low power, the SDK Power Manager selects the best low-power mode that fits all the constraints.

The default low-power mode for each application is Deep-sleep mode. Users can change the behavior by setting a new low-power constraint for the application.

For example, if the low power constraint set from the application is Deep-sleep mode, and no other constraint is set, the**SDK Power Manager** selects Deep-sleep the next time the device enters low power. However, there might be a case when a WFI constraint is set (*PWR_SetLowPowerModeConstraint (PWR_WFI)*) by another component, such as the SecLib module that operates Hardware encryption. In such cases, the **SDK Power Manager** selects this WFI mode until the constraint is released by the SecLib module (*PWR_ReleaseLowPowerModeConstraint(PWR_WFI)*).

If it is required to change the mode from Deep-sleep to Power-down mode, the deep sleep constraint (*PWR_ReleaseLowPowerModeConstraint(PWR_DeepSleep)*) must be released and the power-down constraint (*PWR_SetLowPowerModeConstraints(PWR_PowerDown)*) must be set. The **SDK Power Manager** selects the Power-down mode when the device enters low power.

Parent topic:*Low-Power Management*

Over the Air Programming (OTAP)

This chapter contains a detailed description of the Over The Air Programming capabilities of the Bluetooth Low Energy Host Stack enabled by dedicated GATT Service/Profile and of the support modules needed for OTA programming.

The image transfer is done using a dedicated protocol which is designed to run on both the Bluetooth Low Energy transport and serial transport.

The container for the upgrade image is an image file which has a predefined format which is described in detail. The image file format is independent of the protocol but must contain information specific to the image upgrade infrastructure on an OTAP Client device. Detailed information on how to build an image file starting from a generic format executable generated by an embedded cross-compiling toolchain is shown.

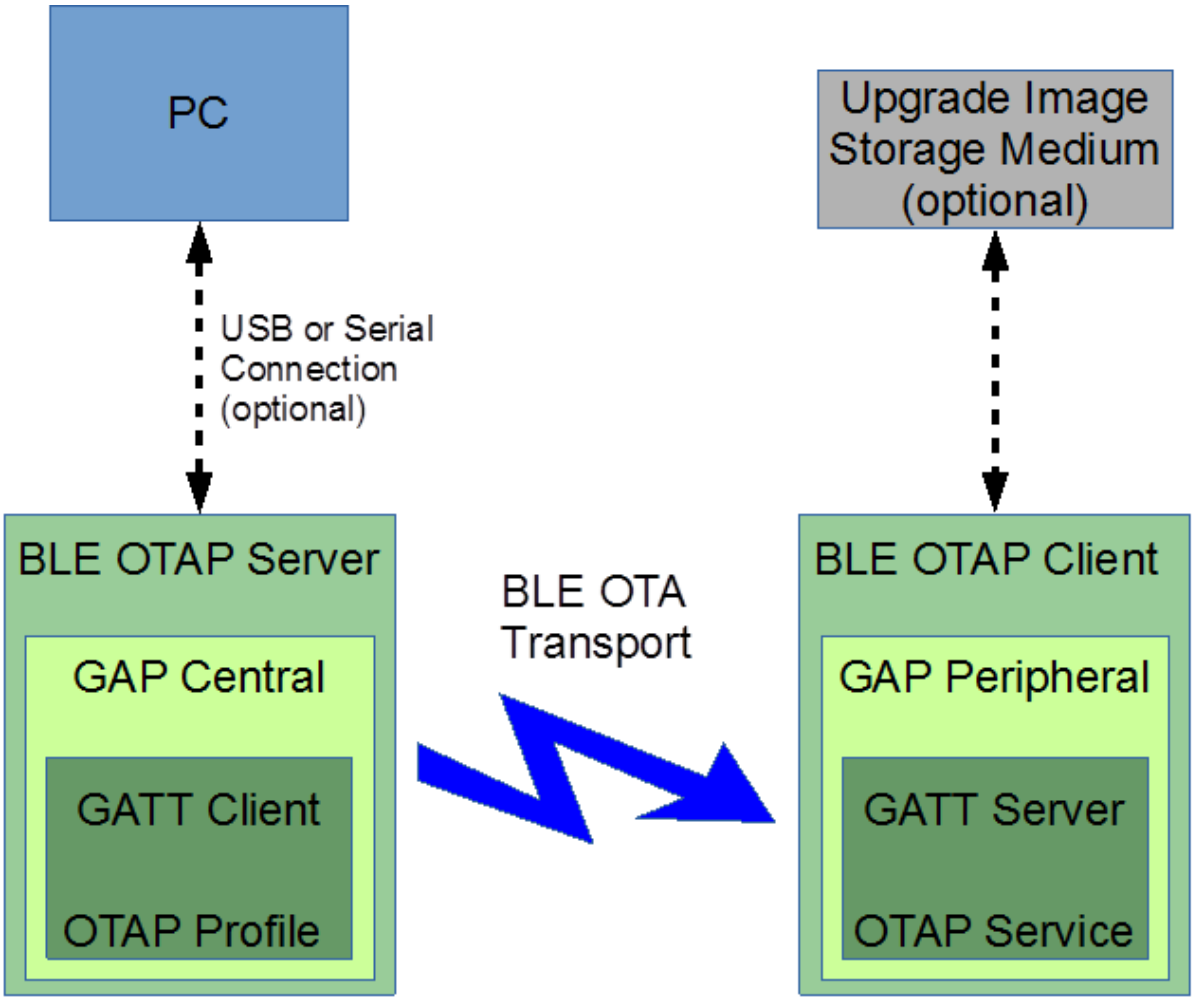
The demo applications implement a typical scenario where a new image is sent from a PC via serial interface to a Bluetooth Low Energy OTAP Server and then over the air to an OTAP Client which is the target of the upgrade image. There are 3 applications involved in the OTAP demo: 1 PC application which builds the image file and serves it to the embedded OTAP Server and 2 embedded applications (OTAP Server and OTAP Client). This chapter contains enough information for building Bluetooth Low Energy OTAP applications which implement different image upgrade scenarios specific to other use cases.

General functionality

A Bluetooth Low Energy OTAP system consists of an OTAP Server and an OTAP Client which exchange an image file over the air using the infrastructure provided by Bluetooth Low Energy (GAP, GATT, SM) via a custom GATT Service and GATT Profile. Additionally, a third application may be used to serve an image to the embedded OTAP Server.

The OTAP Server runs on the GATT Client via the Bluetooth Low Energy OTAP Profile and the OTAP Client runs on the GATT Server via the Bluetooth Low Energy OTAP Service. For the moment the OTAP Server runs on the GAP Central and the OTAP Client runs on the GAP Peripheral.

The Figure shows a typical image upgrade scenario.



Parent topic:*Over the Air Programming (OTAP)*

Bluetooth Low Energy OTAP service-profile

The Bluetooth Low Energy OTAP Service is implemented using the GATT Server which runs on the OTAP Client (GAP Peripheral).

The Bluetooth LE OTAP Service does not require any other Bluetooth LE services because it is a custom service it has a 128-bit UUID. The service has 2 custom characteristics which also have 128-bit UUIDs.

The service must be included in the GATT database of the GATT Server as described in *Creating GATT database* of this document.

OTAP service and characteristics

The OTAP Service has a custom 128-bit UUID which is shown below. The UUID is based on a base 128-bit UUID used for Bluetooth LE custom services and characteristics. These are shown in the tables below.

Service	128-bit UUID
Base Bluetooth Low Energy	00000000 -ba5e-f4ee-5ca1-eb1e5e4b1ce0

The OTAP Service custom 128-bit UUID is built using the base UUID by replacing the most significant 4 bytes which are 0 with a value specific to the OTAP Service which is 01FF5550 in hexadecimal format.

Service	UUID (128-bit)
Bluetooth LE OTAP Service	01ff5550 -ba5e-f4ee-5ca1-eb1e5e4b1ce0

The Bluetooth LE OTAP Service Characteristics UUIDs are built the same as the Bluetooth LE OTAP Service UUID starting from the base 128-bit UUID but using other values for the most significant 4 bytes.

Characteristic	UUID (128-bit)	Properties	Descriptors
Bluetooth LE OTAP Control Point	01ff5551 -ba5e-f4ee-5ca1-eb1e5e4b1ce0	Write, Indicate	CCC
Bluetooth LE OTAP Data	01ff5552 -ba5e-f4ee-5ca1-eb1e5e4b1ce0	Write Without Response	-

Both characteristics are implemented as variable length characteristics.

The Bluetooth LE OTAP Control Point Characteristic is used for exchanging OTAP commands between the OTAP Server and the OTAP Client. The OTAP Client sends commands to the OTAP Server using ATT Notifications for this characteristic and the OTAP Server sends commands to the OTAP Client by making ATT Write Requests to this characteristic. Both ATT Writes and ATT Notifications are acknowledged operations via ATT Write Responses and ATT Confirmations.

The Bluetooth LE OTAP Data characteristic is used by the OTAP Server to send parts of the OTAP image file to the OTAP Client when the ATT transfer method is chosen by the application. The ATT Write Commands (GATT Write Without Response operation) is not an acknowledged operation.

The Bluetooth LE OTAP service and characteristics 128-bit UUIDs are defined in the *gatt_uuid128.h* just as shown below.

```
UUID128(uuid_service_otap, 0xE0, 0x1C, 0x4B, 0x5E, 0x1E, 0xEB, 0xA1, 0x5C, 0xEE, 0xF4, 0x5E, 0xBA,␣
↪0x50, 0x55, 0xFF, 0x01)
UUID128(uuid_char_otap_control_point, 0xE0, 0x1C, 0x4B, 0x5E, 0x1E, 0xEB, 0xA1, 0x5C, 0xEE, 0xF4,␣
↪0x5E, 0xBA, 0x51, 0x55, 0xFF, 0x01)
UUID128(uuid_char_otap_data, 0xE0, 0x1C, 0x4B, 0x5E, 0x1E, 0xEB, 0xA1, 0x5C, 0xEE, 0xF4, 0x5E,␣
↪0xBA, 0x52, 0x55, 0xFF, 0x01)
```

The service is included into the GATT database of the device. It is declared in the *gatt_db.h* file as shown below.

```
PRIMARY_SERVICE_UUID128(service_otap, uuid_service_otap)
CHARACTERISTIC_UUID128(char_otap_control_point, uuid_char_otap_control_point,␣
↪(gGattCharPropWrite_c | gGattCharPropIndicate_c))
VALUE_UUID128_VARLEN(value_otap_control_point, uuid_char_otap_control_point,␣
↪(gPermissionFlagWritable_c), 16, 16, 0x00)
CCCD(cccd_otap_control_point)
CHARACTERISTIC_UUID128(char_otap_data, uuid_char_otap_data, (gGattCharPropWriteWithoutRsp_
↪c))
VALUE_UUID128_VARLEN(value_otap_data, uuid_char_otap_data, (gPermissionFlagWritable_c),␣
↪gAttMaxMtu_c - 3, gAttMaxMtu_c - 3, 0x00)
```

The Bluetooth LE OTAP Control Point characteristic should be large enough for the longest command which can be exchanged between the OTAP Server and The OTAP Client.

The Bluetooth LE OTAP Data characteristic should be large enough for the longest data chunk command the OTAP Client expects from the OTAP Server to be sent via ATT. The maximum length

of the OTAP Data Characteristic value is ATT_MTU- 3. 1 byte is used for the ATT OpCode and 2 bytes are used for the Attribute Handle when performing a Write Without Response, the only operation permitted for this characteristic value.

Parent topic: Bluetooth Low Energy OTAP service-profile

OTAP server and OTAP client interactions

The OTAP Server application scans for devices advertising the OTAP Service. When it finds one it connects to that device and notifies it of the available image files or waits for requests regarding available image files. The behavior is specific to each application which needs the OTAP functionality. The Bluetooth LE OTAP Protocol described below details how to do this.

After an OTAP Server (GAP Central, GATT Client) connects to an OTAP Client (GAP Peripheral, GATT Server) it scans the device database and identifies the handles of the OTAP Control Point and OTAP Data characteristics and their descriptors. Then it writes the CCC Descriptor of the OTAP Control point to allow the OTAP Client to send it commands via ATT Indications. It can send commands to the OTAP Client by using ATT Write Commands to the OTAP Control Point characteristic.

After the connection is established, if the OTAP Client wants to use the L2CAP CoC transfer method it must register a L2CAP PSM with the OTAP Server.

The OTAP Client only starts any image information request or image transfer request procedures only after the OTAP Server writes the OTAP Control Point CCCD to ensure there is bidirectional communication between the devices.

Parent topic: Bluetooth Low Energy OTAP service-profile

Parent topic:*Over the Air Programming (OTAP)*

Bluetooth LE OTAP protocol

The protocol consists of a set of commands (messages) which allow the OTAP Client to request or be notified about the available images on an OTAP Server and to request the transfer of parts of images from the OTAP Server.

All commands with the exception of the image data transfer commands are exchanged through the OTAP Control Point characteristic of the OTAP Service. The data transfer commands are sent only from the OTAP Server to the OTAP Client either via the OTAP Data characteristic of the OTAP Service or via a dedicated Credit Based Channel assigned to a L2CAP PSM.

Protocol design considerations

The OTAP Client is a GAP Peripheral device, and therefore has limited resources. This is why the OTAP Protocol was designed in such a way that it is at the discretion of the OTAP Client if, when, how fast and how much of an available upgrade image is transferred from the OTAP Server. The OTAP Client also decides which is the image transfer method based on its capabilities. Two image transfer methods are supported at this moment: the ATT Transfer Method and the L2CAP PSM CoC Transfer Method.

The ATT Transfer Method is supported by all devices which support Bluetooth LE but it has the disadvantage of a small data payload size and a larger Bluetooth LE stack protocols overhead leading to a lower throughput. This disadvantage has been somewhat reduced by the introduction of the Long Frames feature in the Bluetooth LE specification 4.2 Link Layer which allows for a larger ATT_MTU value. The L2CAP PSM CoC Transfer Method is an optional feature available for devices running a Bluetooth stack version 4.1 and later. The protocol overhead is smaller and the data payload is higher leading to a high throughput. The L2CAP PSM Transfer Method is the preferred transfer method and it is available on all Bluetooth LE Devices if the application requires it.

Based on application requirements and device resources and capabilities the OTAP Clients can request blocks of OTAP images divided into chunks. To minimize the protocol overhead and

maximize throughput an OTAP Client makes a data block request specifying the block size and the chunk size and the OTAP Server sends the requested data chunks (which have a sequence number) without waiting for confirmation. The block size, chunk size and number of chunks per block are limited and suitable values must be used based on application needs.

The OTAP Client can stop or restart an image block transfer at any time if the application requires it or a transfer error occurs. The OTAP Server implementation can be almost completely stateless. The OTAP Server does not need to remember what parts of an image have been transferred, this is the job of the OTAP Client which can request any part of an image at any time. This allows it to download parts of the image whenever and how fast its resources allow it. The OTAP Server simply sends image information and image parts on request.

The Bluetooth LE OTAP Protocol is designed to be used not only on Bluetooth LE transport medium but on any transport medium, for example a serial communication interface or another type of wireless interface. This may be useful when transferring an upgrade image from a PC or a mobile device to the OTAP Server to be sent via Bluetooth LE to the OTAP Clients which require it. In the OTAP Demo Applications the embedded OTAP Server relays OTAP commands between an OTAP Client and a PC via a serial interface and using a FSCI type protocol. Effectively the OTAP Client downloads the upgrade image from the PC and not from the OTAP Server. Other transfer methods may be used based on application needs.

Parent topic: Bluetooth LE OTAP protocol

Bluetooth Low Energy OTAP commands

The Bluetooth LE OTAP Commands general format is shown below. A command consists of two parts, a Command ID, and a Command Payload as shown in the table below.

Field Name	CmdId	CmdPayload
Size (Bytes)	1	variable

Commands are sent over the transport medium starting with the Command ID and continuing with the Command Payload.

All multibyte command parameters in the Command Payload are sent in a least significant octet first order (little endian).

A summary of the commands supported by the Bluetooth LE OTAP Protocol is shown in the table below. Each of the commands is then detailed in its own section.

CmdId	Command Name
0x01	New Image Notification
0x02	New Image Info Request
0x03	New Image Info Response
0x04	Image Block Request
0x05	Image Chunk
0x06	Image Transfer Complete
0x07	Error Notification
0x08	Stop Image Transfer

New image notification command

This command can be sent by an OTAP Server to an OTAP Client, usually immediately after the first connection, to notify the OTAP Client of the available images on the OTAP Server.

Table 14: New Image Notification Command Parameters

CmdId	Name	Dir	Parameters	Param Size (Bytes)	Description	Total Size (CmdId + Payload)
0x01	New Image Notification	S->C	ImageId	2	Short image identifier used for transactions between the OTAP Server and OTAP Client. Should be unique for all images on a server.	15
			ImageVersion	8	Image file version. Contains sufficient information to identify the target hardware, stack version and build version.	
			ImageFileSize	4	Image file size in bytes.	

The *ImageId* parameter should not be '0x0000', which is the reserved value for the current running image or 0xFFFF, which is the reserved value for "no image available".

Parent topic: Bluetooth Low Energy OTAP commands

New image info request command

This command can be sent by an OTAP Client to an OTAP Server to inquire about available upgrade images on the OTAP Server.

CmdId	Name	Dir	Parameters	Param Size (Bytes)	Description	Total Size (CmdId+Payload)
0x02	New Image Info Request	C->S	CurrImageId	2	Id of the currently running image. Should be 0x0000.	11
0x02	New Image Info Request	C->S	CurrImageVer	8	Version of the currently running image. A value of all zeroes signals that the client is looking for all images available on an OTAP Server. A value of all zeroes requests information about all images on the server.	11

- The *CurrImageId* parameter should be set to 0x0000 to signify the current running image.
- The *CurrImageVer* parameter should contain sufficient information about the target device for the OTAP Server to determine if it has an upgrade image available for the requesting OTAP Client.
- A value of all zeroes for the *CurrImageVer* means that an OTAP Client is requesting information about all available images on an OTAP Server and the OTAP Server should send a New Image Info Response for each image.

Parent topic: Bluetooth Low Energy OTAP commands

New image info response command

This command is sent by the OTAP Server to the OTAP Client as a response to a New Image Information Request Command.

CmdId	Name	Dir	Parameters	Param Size (Bytes)	Description	Total Size (CmdId+Payload)
0x03	New Image Info Response	S->C	ImageId	2	Image Id. Value 0xFFFF is reserved as "no image available"	15
0x03	New Image Info Response	S->C	ImageVersion	8	Image file version	15
0x03	New Image Info Response	S->C	4 Image file size.	4	Image file size	15

The *ImageId* parameter with a value of 0xFFFF is reserved for the situation where no upgrade image is available for the requesting device.

Parent topic: Bluetooth Low Energy OTAP commands

Image block request command

This command is sent by the OTAP Client to the OTAP Server to request a part of the upgrade image after it has determined the OTAP Server has an upgrade image available.

When an OTAP Server Receives this command it should stop any image file chunk transfer sequences in progress.

CmdId	Nam	Dir	Parameters	Para Size	Description	Total Size (CmdId+Payload)
0x04	Image Bloc Reques	C->S	StartPositionBlockSizeChunkSizeTransferMethodL2ca	4421	Start position of the image block to be transferred.Requested total block size in bytes.Should be optimized to the Transfer Channel type. The maximum number of chunks per block is 256. Value is inbytes.0x00 - ATT 0x01 – L2CAP PSM Credit based channel0x0004 - ATT Other values – PSM for credit based channels	16

The *ImageId* parameter contains the ID of the upgrade image.

The *StartPosition* parameter specifies the location in the image upgrade file at which the requested block starts.

The *BlockSize* and *ChunkSize* parameters specify the size in bytes of the block to be transferred and the size of the chunks into which a block is separated. The *ChunkSize* value must be chosen in such a way that the total number of chunks can be represented by the *SeqNumber* parameter of the Image Chunk Command. At the moment this parameter is 1 byte in size so there are a maximum of 256 chunks per block. The chunk sequence number goes from 0 to 255 (0x00 to 0xFF). If this condition is not met or the requested block is not entirely into the image file bounds an error is sent to the OTAP Client when the OTAP Server receives this misconfigured Image Block Request Command.

The maximum value of the *ChunkSize* parameter depends on the maximum ATT_MTU and L2CAP_MTU supported by the Bluetooth LE stack version and implementation.

The *TransferMethod* parameter is used to select the transfer method which can be ATT or L2CAP PSM CoC. The *L2capChannelOrPsm* parameter must contain the value 0x0004 for the ATT transfer method and another value representing the chosen PSM for the L2CAP PSM transfer method. The default PSM for the Bluetooth LE OTAP demo applications is 0x004F for both the OTAP Server and the OTAP Client although the specification allows different values at the 2 ends of the L2CAP PSM connection. The PSM must be in the range reserved by the Bluetooth specification which is 0x0040 to 0x007F.

The optimal value of the *ChunkSize* parameter depends on the chosen transfer method and the Link Layer payload size. Ideally it must be chosen in such a way that full packets are sent for every chunk in the block.

The default Link Layer payload is 27 bytes form which we subtract 4 for the L2CAP layer and 3 for the ATT layer (1 for the ATT Cmd Opcode and 2 for the Handle) leaving us with a 20 byte OTAP protocol payload. From these 20 bytes we subtract 1 for the OTAP CmdId and 1 for the chunk sequence number leaving us with an optimum chunk size of 18 for the ATT transfer method – which is the default in the demo applications. For the L2CAP PSM transfer method the chosen default chunk size is 111. This was chosen so as a chunk fits exactly 5 link layer packets. The default L2CAP payload of 23 (27 - 4) multiplied by 5 gives us 115 from which we subtract 2 bytes for the SDU Length (which is only sent in the first packet), 1 byte for the OTAP CmdId and 1 byte for the chunk sequence number which leaves exactly 111 bytes for the actual payload.

If the Link layer supports Long Frames feature then the chunk size should be set according to the negotiated ATT MTU for the ATT transfer method. From the negotiated ATT MTU (*att_mtu*) subtract 3 bytes for the ATT layer (1 for the ATT Cmd Opcode and 2 for the Handle) then subtract 2 bytes for the OTAP protocol (1 for the CmdId and 1 for the chunk sequence number) to determine the optimum chunk size (*optimum_att_chunk_size = att_mtu – 3 – 2).* For the L2CAP PSM transfer method the chunk size can be set based on the maximum L2CAP SDU size (*max_l2cap_sdu_size*) from which 4 bytes should be subtracted, 2 for the SDU Length and 2 for the OTAP protocol (*optimum_l2cap_chunk_size = max_l2cap_sdu_size – 3 – 2*). In some particular cases reducing the L2CAP chunk size could lead to better performance. If the L2CAP chunk size needs to be reduced it should be reduced so it fits exactly a number of link layer packets. An example of how to compute an optimal reduced L2CAP chunk size is given in the previous paragraph.

Parent topic: Bluetooth Low Energy OTAP commands

Image chunk command

One or more Image Chunk Commands are sent from the OTAP Server to the OTAP Client after an Image Block Request is received by the former. The image chunks are sent via the ATT Write Without Response mechanism if the ATT transfer method is chosen and directly via L2CAP if the L2CAP PSM CoC transfer method is chosen.

CmdId	Name	Dir	Parameters	Parai Size	Description	Total Size (CmdId+Payload)
0x05	Image Chunl	S->C	SeqNumberData	1variable	In the range 0 -> BlockSize/ChunkSize - calculated by Server, checked by Client. The command code is present even when ATT is used.Actual data	3 or more

The *SeqNumber* parameter is the chunk sequence number and it has incremental values from 0 to 255 (0x00 to 0x FF) for a maximum of 256 chunks per block.

The *Data* parameter is an array containing the actual image part being transferred starting from the *BlockStartPosition* + *SeqNumber ChunkSize* position in the image file and containing *ChunkSize* or less bytes depending on the position in the block. Only the last chunk in a block can have less than *ChunkSize*bytes in the Image Chunk Command data payload.

Parent topic: Bluetooth Low Energy OTAP commands

Image transfer complete command

This command is sent by the OTAP Client to the OTAP Server when an image file has been completely transferred and its integrity has been checked.

CmdId	Name	Dir	Parameters	Param Size	Description	Total Size (CmdId+Payload)
0x06	Image Transfer Complete	C->S	ImageIdStatus	21	Image IdStatus of the image transfer. 0x00 - Success	4

The *ImageId* parameter contains the ID of the image file that was transferred.

The *Status* parameter is 0x00 (Success) if image integrity and possibly other checks have been successfully made after the image is transferred and another value if integrity or other kind of errors have occurred.

If the status is 0x00 the OTAP Client can trigger the Bootloader to start flashing the new image. The image flashing should take about 15 seconds for a 160 KB flash memory.

Parent topic: Bluetooth Low Energy OTAP commands

Error notification command

This command can be sent by both the OTAP Server and the OTAP Client when an error of any kind occurs. When an OTAP Server receives this command, it should stop any image file chunk transfer sequences in progress.

CmdId	Name	Dir	Parameters	Param Size (Bytes)	Description	Total Size (CmdId+Payload)
0x07	Error Notification	Bidi	CmdIdErrorStatus	11	Id of the command which generated the error.Error Status: Examples: out of image bounds, chunk too small, chunk too large, image verification failure, bad command format, image not available, unknown command	3

The *CmdId* parameter contains the ID of the command which caused the error (if applicable).

The *ErrorStatus* parameter contains the source of the error. All error statuses are defined in the *otapStatus_t* enumerated type in the *otap_interface.h* file.

Parent topic: Bluetooth Low Energy OTAP commands

Stop image transfer command

This command is sent from the OTAP Client to the OTAP Server whenever the former wants to stop the transfer of an image block which is currently in progress, or from OTAP Server to the OTAP Client when the image transfer is stopped from application (Test Tool).

CmdId	Name	Dir	Parameters	Param Size	Description	Total Size (CmdId+Payload)
0x08	Stop Image Transfer	C->S	ImageId	2	Image Id	3

The *ImageId* parameter contains the ID of the image being transferred.

Parent topic: Bluetooth Low Energy OTAP commands

Parent topic: Bluetooth LE OTAP protocol

OTAP client–server interactions

The interactions between the OTAP Server and OTAP Client start immediately after the connection, discovery of the OTAP Service characteristics and writing of the OTAP Control Point CCC Descriptor by the OTAP Server.

The first command sent could be a New Image Notification sent by the OTAP Server to the OTAP Client or a New Image Info Request sent by the OTAP Client. The OTAP Server can respond with a New Image Info response if it has a new image for the device which sent the request (this can be determined from the *ImageVerison* parameter). The best strategy depends on application requirements.

After the OTAP Client has determined that the OTAP Sever has a newer image it can start downloading the image. This is done by Sending Image Block Request commands to retrieve parts of the image file. The OATP Server answers to these requests with one or more Image Chunk Commands via the requested transfer method or with an Error Notification if there are improper parameters in the Image Block Request. The OTAP Client makes as many Image Block Requests as it is necessary to transfer the entire image file.

The OTAP Client decides how often Image Block Request Commands are sent and can even stop a block transfer which is in progress via the Stop Image Transfer Command. The OTAP Client is in complete control of the image download process and can stop it and restart it at any time based on its resources and application requirements.

A typical **Bluetooth LE OTAP Image Transfer** scenario is shown in the message sequence chart Figure.

Parent topic: Bluetooth LE OTAP protocol

Parent topic:*Over the Air Programming (OTAP)*

Bluetooth Low Energy OTAP image file format

The Bluetooth LE OTAP Image file has a binary file format. It is composed of a header followed by a number of sub-elements. The header describes general information about the file. There are some predefined sub-elements of a file but an end manufacturer could add manufacturer-specific sub-elements. The header does not have details of the sub-elements. Each element is described by its type.

The general format of an image file is shown in the table below.

Image File Element	Value Field Length (bytes)	Description
Header	Variable	The header contains general information about the image file.
Upgrade Image Sub-element	Variable	This sub-element contains the actual binary executable image, which is copied into the flash memory of the target device. The maximum size of this sub-element depends on the target hardware.
Image File CRC Sub-element	2	This is a 16-bit CCITT type CRC which is calculated over all elements of the image file with the exception of the Image File CRC sub-element itself. This must be the last sub-element in an image file.

Each sub-element in a Bluetooth LE OTAP Image File has a Type-Length-Value (TLV) format. The type identifier provides forward and backward compatibility as new sub-elements are introduced. Existing devices that do not understand newer sub-elements may ignore the data.

The following table shows the general format of a Bluetooth LE Image File sub-element.

Sub-field	Size (Bytes)	Format	Description
Type	2	uint16	Type Identifier – determines the format of the data contained in the value field
Length	4	uint32	Length of the *Value* field of the sub-element.
Value	variable	uint8[]	Data payload

Some sub-element type identifiers are reserved while others are left for manufacturer-specific use. The table below shows the reserved type identifiers and the manufacturer-specific ranges.

Type Identifiers	Description
0x0000	Upgrade Image
0x0001 – 0xefff	Reserved
0xf000 – 0xffff	Manufacturer-Specific Use

The OTAP Demo applications use two of the manufacturer-specific sub-element type identifiers while the rest remain free to use. The two are shown in the table below along with a short description.

Manufacturer-specific Type Identifiers	Sub-element Name	Notes
0xf000	Sector Bitmap	Bitmap that signals the bootloader the sectors of the internal flash, which should be overwritten and which should remain as is.
0xf100	Image File CRC	16-bit CRC that is computed over the image file with the exception of the CRC sub-element itself.

Bluetooth Low Energy OTAP header

The format and fields of the Bluetooth Low Energy OTAP Header are summarized in the table below.

Octets	Data Types	Field Name	Mandatory/Optional
4	Unsigned 32-bit integer	Upgrade File Identifier	M
2	Unsigned 16-bit integer	Header Version	M
2	Unsigned 16-bit integer	Header Length	M
2	Unsigned 16-bit integer	Header Field Control	M
2	Unsigned 16-bit integer	Company Identifier	M
2	Unsigned 16-bit integer	Image ID	M
8	8 byte array	Image Version	M
32	Character string	Header String	M
4	Unsigned 32-bit integer	Total Image File Size	M

The fields are shown in the order they are placed in memory from the first location to the last.

The total size of the header without the optional fields (if defined by the *Header Field Control*) is 58 bytes.

All the fields in the header have a little endian format with the exception of the *Header String* field which is an ASCII character string.

A packed structure type definition for the contents of the Bluetooth LE OTAP Header can be found in the *otap_interface.h* file.

Upgrade file identifier

Fixed value 4 byte field used to identify the file as being a Bluetooth LE OTAP Image File. The predefined value is "**0x0B1EF11E**".

Parent topic: Bluetooth Low Energy OTAP header

Header version

This 2 byte field contains the major and minor version number. The high byte contains the major version and the low byte contains the minor version. The current value is "**0x0100**" with the major version "**01**" and the minor version **"00"**. A change to the minor version means the OTA upgrade file format is still backward compatible, while a change to the major version suggests incompatibility.

Parent topic: Bluetooth Low Energy OTAP header

Header length

Length of all the fields in the header including the ***Upgrade File Identifier*** field, ***Header Length*** field and all the optional fields. The value insulates existing software against new fields that

may be added to the header. If new header fields added are not compatible with current running software, the implementations should process all fields they understand and then skip over any remaining bytes in the header to process the image or CRC sub-element. The value of the ***Header Length*** field depends on the value of the Header Field Control field, which dictates which optional header fields are included.

Parent topic: Bluetooth Low Energy OTAP header

Header field control

This is a 2-byte bit mask that specifies the optional fields present in the OTAP Header.

In case no optional fields are defined, this whole field is reserved and should be set to "**0x0000**".

Parent topic: Bluetooth Low Energy OTAP header

Company identifier

This is the company identifier assigned by the Bluetooth SIG. The Company Identifier used for the OTAP demo applications is "**0x01FF**".

Parent topic: Bluetooth Low Energy OTAP header

Image ID

This is a unique short identifier for the image file. It is used to request parts of an image file. This number should be unique for all images available on a Bluetooth LE OTAP Server.

- The value 0x0000 is reserved for the current running image.
- The value 0xFFFF is reserved as a "no image available" code for New Image Info Response commands.

This field value must be used in the *ImageID* field in the *New Image Notification* and *New Image Info Response* commands.

Parent topic: Bluetooth Low Energy OTAP header

Image version

This is the full identifier of the image file. It should allow a Bluetooth LE OTAP Client to identify the target hardware, stack version, image file build version, and other parameters if necessary. The recommended format of this field (which is used by the OTAP Demo applications) is shown below but an end device manufacturer could choose different format. The subfields are shown in the order they are placed in memory from the first location to the last. Each subfield has a little-endian format, if applicable.

Subfield	Size (bytes)	Format	Description
Build Version	3	uint8[]	Image build version.
Stack Version	1	uint8	0x41 for example for Bluetooth Low Energy Stack version 4.1.
Hardware ID	3	uint8[]	Unique hardware identifier.
End Manufacturer Id	1	uint8	ID of the hardware–specific to the end manufacturer

This field value must be used in the *ImageVersion* field in the New Image Notification and New Image Info Response commands.

Parent topic: Bluetooth Low Energy OTAP header

Header string

This is a manufacturer-specific string that may be used to store other necessary information as seen fit by each manufacturer. The idea is to have a human readable string that can prove helpful during the development cycle. The string is defined to occupy 32 bytes of space in the OTAP Header. The default string used for the Bluetooth LE OTAP demo application is "BLE OTAP Demo Image File".

Parent topic: Bluetooth Low Energy OTAP header

Total image file size

The value represents the total image size in bytes. This is the total of data in bytes that is transferred over-the-air from the server to the client. In most cases, the total image size of an OTAP upgrade image file is the sum of the sizes of the OTAP Header and all the other sub-elements on the file. If the image contains any integrity and/or source identity verification fields then the Total Image File Size also includes the sizes of these fields.

Parent topic: Bluetooth Low Energy OTAP header

Parent topic: Bluetooth Low Energy OTAP image file format

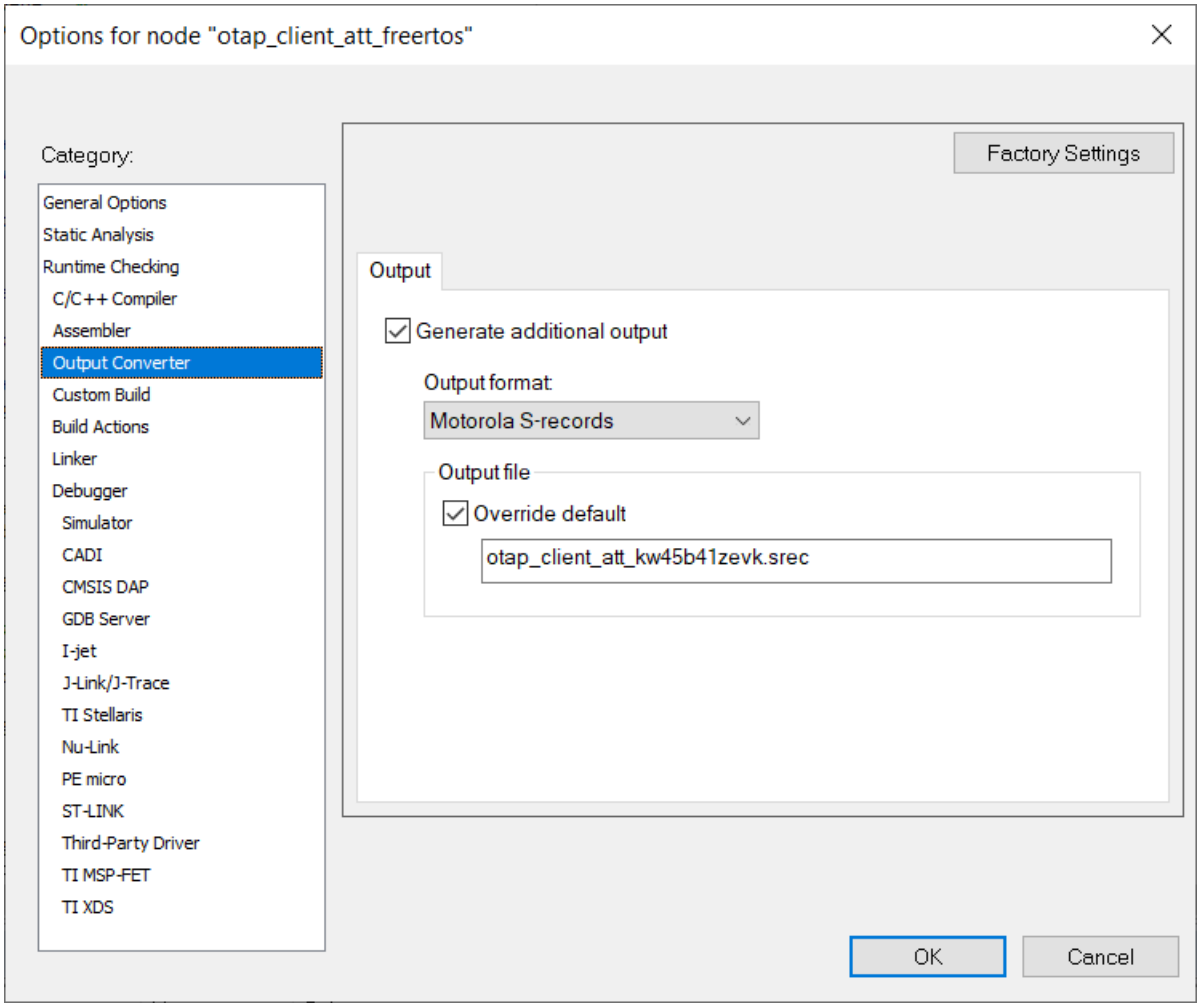
Parent topic:*Over the Air Programming (OTAP)*

Building Bluetooth Low Energy OTAP image file from SREC file

A SREC (Motorola S-record) file is an ASCII format file which contains binary information. Common file extensions are: .srec, .s19, .s28, .s37 and others. Most modern compiler toolchains can generate an SREC format executable.

The steps described in this section enable the creation of a SREC file for your embedded application in IAR Embedded Workbench.

For this, open the target properties and go to the **Output Converter** tab. Activate the **Generate additional output** checkbox and choose the **Motorola** option from the **Output format** drop down menu. From the same pane you can also override the name of the output file. A screenshot of the described configuration is shown in the Figure below.



The format of the SREC file is shown in table below. It contains lines of text called records which have a specific format. An example of the contents of a SREC file is shown below.

```
S02000006F7461705F636C69656E745F6174745F4672656552544F532E73726563A1
S1130000F83F0020EB05000075060000750600 00AF
S1130010750600007506000075060000750600 00F0
S1130020750600007506000075060000750600 00E0
S1130030750600007506000075060000750600 00D0
S1130040000000000000000000000000000000 00AC
S1130050000000000000000000000000000000 009C
.............
S2140117900121380004F05FF8002866D12A003100E4
S2140117A06846008804F022F8A689002E16D0002884
S2140117B014D12569278801A868A11022F7F782FCB1
S2140117C06B4601AA0121380004F045F800284CD1E7
S2140117D02A0031006846008804F008F8A68A002E20
```

All records start with the ASCII letter 'S' followed by an ASCII digit from '0' to '9'. These two characters from the record type identify the format of the data field of the record.

The next 2 ASCII characters are 2 hex digits that indicate the number of bytes (hex digit pairs) which follow the rest of the record (address, data, and checksum).

The address that follows next can have 4, 6, or 8 ASCII hex digits, depending on the record type.

The data field is placed after the address and it contains 2 * n ASCII hex digits for 'n' bytes of actual data.

The last element of the S record is the checksum, which comprises 2 ASCII hex digits. The check-

sum is computed by adding all the bytes of the byte count, address, and data fields. Then the ones complement of the least significant octet of the sum is computed to determine the checksum.

Field	Record Type	Count	Address	Data	Check-sum	Line Ter-minator
Format	"Sn", n=0..9	ASCI-Ihex digits	ASCI-Ihex digits	ASCIIhex digits	ASCI-Ihex digits	"\r\n"
Length (charac-ters)	2	2	4,6,8	*Count –*len*(Address) – *len(*Checksum)*	2	2

More details about the SREC file format can be found at this location: en.wikipedia.org/wiki/SREC_(file_format).

We are only interested in records that contain actual data. These are S1, S2, and S3 records. The other types of records can be ignored.

The S1, S2, and S3 records are used to build the Upgrade Image Sub-element of the image file simply by placing the record data at the location specified by the record address in the *Value* field of the Sub-element. It is recommended to fill all gaps in S record addresses with 0xFF.

To build an OTAP Image File from a SREC file, follow the procedure described below:

- Generate the SREC file by correctly configuring your toolchain to do so.
- Create the image file header.
  - Set the Image ID field of the header to be unique on the OTAP Server.
  - Leave the Total Image File Size Field blank for the moment.
- Create the Upgrade Image Sub-element
  - Read the S1, S2, and S3 records from the SREC file and place the binary record data to the record addresses in the *Value* filed of the sub-element. Fill all address gaps in the S records with 0xFF.
  - Fill in the *Length* field of the sub-element with the length of the written *Value* field.
- Create the Sector Bitmap Sub-element
  - A default working setting would be all byes 0xFF for the *Value* field of this sub-element.
- Create the Image File CRC Sub-element
  - Compute the total image file size as the length of the header + the length of all 3 sub-elements and fill in the appropriate filed in the header with this value.
  - Compute and write the *Value*field of this sub-element using the header and all sub-elements except this one.
  - The *OTA_CrcCompute()* function in the *OtaSupport.c* file can be used to incrementally compute the CRC.

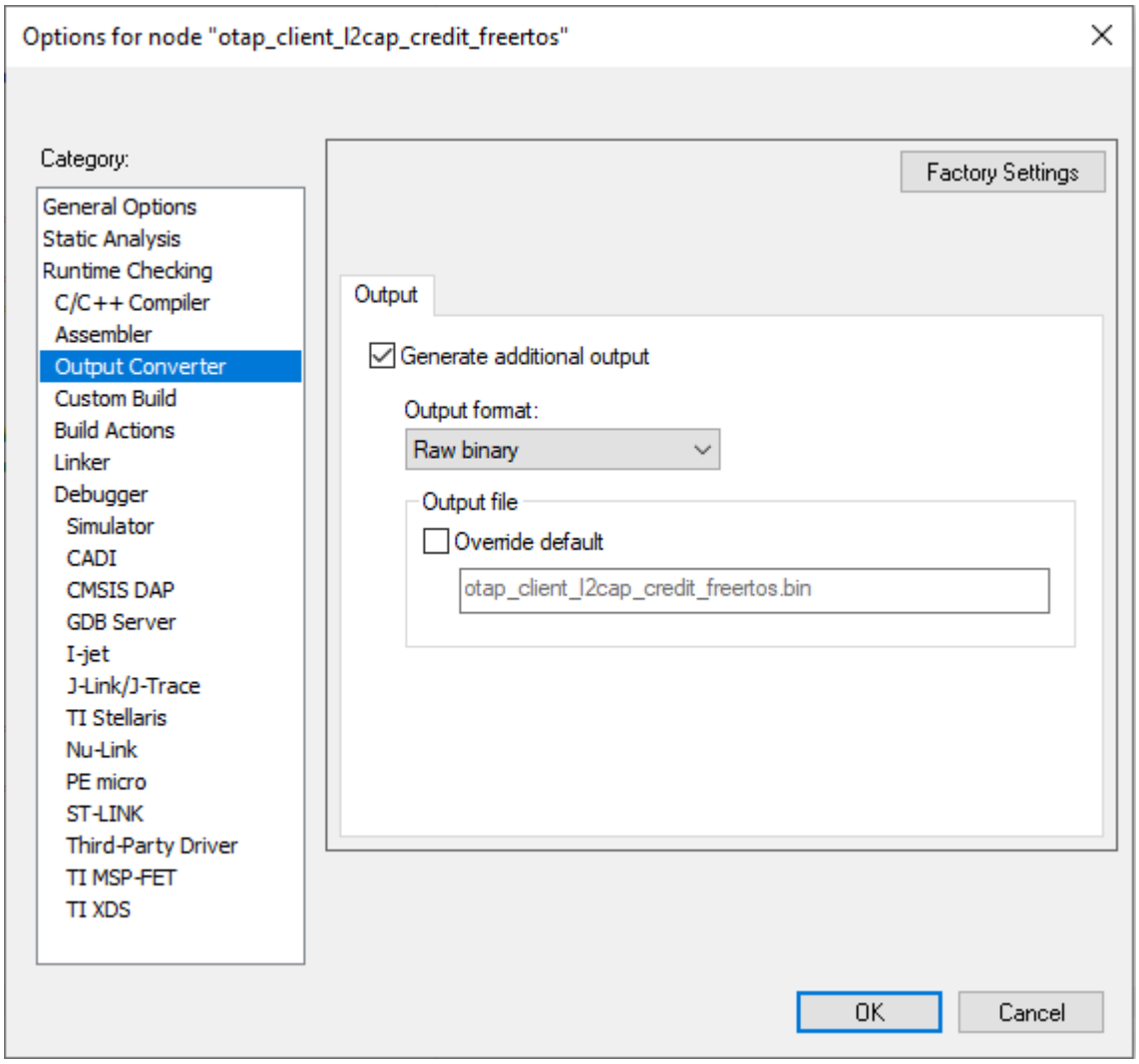
If the Image ID is not available when the image file is created, then the CRC cannot be computed. It can be computed later after the Image ID is established and written in the appropriate field in the header.

Parent topic:*Over the Air Programming (OTAP)*

Building Bluetooth Low Energy OTAP image file from BIN file

A BIN file is an binary file which contains an executable image. The most common extension for this type of file is `.bin`. Most modern compiler toolchains can output a BIN format executable.

To enable the creation of a BIN file for your embedded application in IAR Embedded Workbench open the target properties and go to the *Output Converter* tab. Activate the *"Generate additional output"* checkbox and choose the *binary* option from the *"Output format"* drop down menu. From the same pane you can also override the name of the output file. The Figure shows a screenshot of the described configuration.



The format of the BIN file is very simple. It contains the executable image in binary format as is, starting from address 0 and up to the highest address. This type of file does not have any explicit address information.

To build an OTAP Image File from a BIN file, follow the procedure below:

- Generate the BIN file by correctly configuring your toolchain to do so.
- Create the image file header
  - Set the Image ID field of the header to be unique on the OTAP Server.
  - Leave the Total Image File Size field blank for the moment.
- Create the Upgrade Image Sub-element
  - Copy the entire contents of the BIN file as is into the *Value* filed of the sub-element.
  - Fill in the *Length* field of the sub-element with the length of the written *Value* filed.
- Create the Sector Bitmap Sub-element
  - A default working setting would be all bytes 0xFF for the *Value* field of this sub-element.

- Create the Image File CRC Sub-element
  - Compute the total image file size as the length of the header + the length of all 3 sub-elements and fill in the appropriate filed in the header with this value.
  - Compute and write the *Value*field of this sub-element using the header and all sub-elements except this one.
  - The *OTA_CrcCompute()* function in the *OtaSupport.c* file can be used to incrementally compute the CRC.

If the Image ID is not available when the image file is created, then the CRC cannot be computed. It can be computed later after the Image ID is established and written in the appropriate field in the header.

Parent topic:*Over the Air Programming (OTAP)*

Bluetooth Low Energy OTAP application integration

The Bluetooth Low Energy OTAP demo applications are standalone applications that only run the OTAP Server and the OTAP Client. In practice, however the OTAP Server and OTAP Client are used alongside with other functions. The OTAP functionality is used as a tool along with the main application on a device.

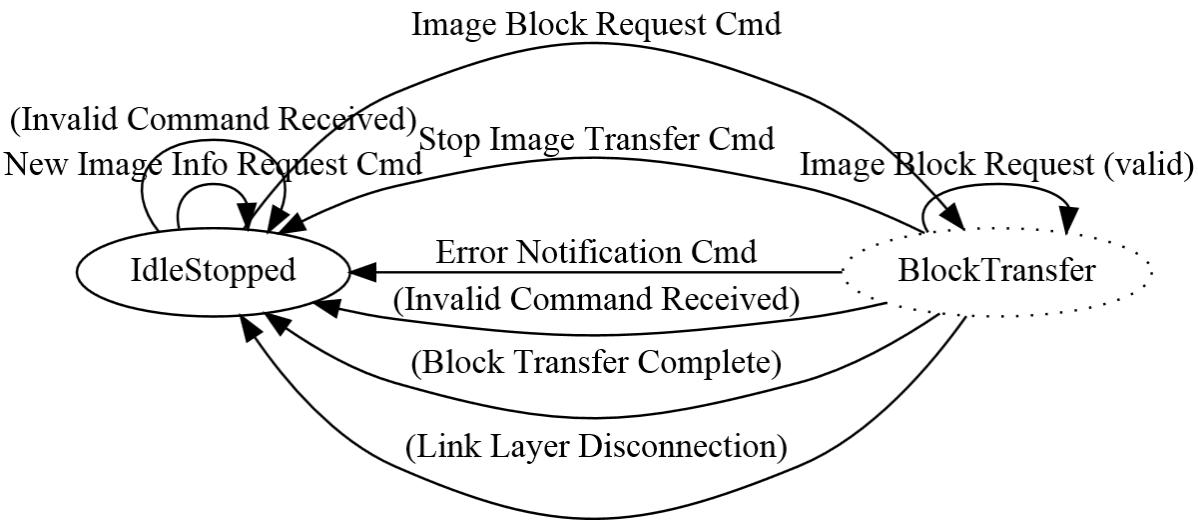
This section contains some guidelines on how to integrate OTAP functionality into other Bluetooth Low Energy applications.

OTAP server

Before any OTAP transactions can be done the application which acts as an OTAP Server must connect to a peer device and perform ATT service and characteristic discovery. Once the handles of the OTAP Service, OTAP Control Point and OTAP Data characteristics and their descriptors are found then OTAP communication can begin.

A good starting point for OTAP transactions for both the OTAP Server and The OTAP client is the moment the Server writes the OTAP Control Point CCCD to receive ATT Indications from the OTAP Client. At that point the Server can send a New Image Notification to the Client if it finds out what kind of device the client is through other means than the OTAP server. How this can be done is entirely application-specific. If the OTAP Server does not know exactly what kind of device is the OTAP Client it can wait for the Client to send a New Image Info Request. Again, the best behavior depends on application requirements.

Once OTAP communication begins then the OTAP Server just has to wait for commands from the OTAP Client and answer them. This behavior is almost completely stateless. An example state diagram for the OTAP Server application is shown in Figure.



The OTAP Server waits in an idle state until a valid Image Block Request command is received and then moves to a pseudo-state and starts sending the requested block. The transfer can be interrupted by some commands (Error Notification, Stop Image Transfer, and so on) or other events (disconnection, user interruption, and so on).

The *otap_interface.h* file contains infrastructure for sending and receiving OTAP Commands and parsing OTAP image files. Packed structure types are defined for all OTAP commands and type enumerations are defined for command parameter values and some configuration values like the data payloads for the different transfer methods.

To receive ATT Indications and ATT Write Confirmations from the OTAP Client the OTAP Server application registers a set of callbacks in the stack. This is done in the *BluetoothLEHost_Initialized* function.

```
App_RegisterGattClientProcedureCallback (BleApp_GattClientCallback);
App_RegisterGattClientIndicationCallback (BleApp_GattIndicationCallback);
```

This *BleApp_GattIndicationCallback()* function is called when any attribute is indicated so the handle of the indicated attribute must be checked against a list of expected handles. In our case, we are looking for the OTAP Control Point handle that was obtained during the discovery procedure.

The *BleApp_GattIndicationCallback()* function from the demo calls an application-specific function called *BleApp_AttributeIndicated()* in which the OTAP Commands are handled.

```
static void BleApp_AttributeIndicated
(
    deviceId_t      deviceId,
    uint16_t        handle,
    uint8_t*        pValue,
    uint16_t        length
)
{
    if (handle == mPeerInformation.customInfo.otapServerConfig.hControlPoint)
    {
       otapCommandVars.pValueTemp = pValue;
        otapCommand_t*  pOtaCmd = otapCommandVars.otapCommandTemp;
        /* ... Missing code here ... */
        /* If the OTAP Server does not have internal storage then all commands must be forwarded
         *  via the serial interface. */
           FsciBleOtap_SendPkt (&(pOtaCmd->cmdId),
                    (uint8_t*)(&(pOtaCmd->cmd)),
                    length - gOtap_CmdIdFieldSize_c);
    }
    elseif (handle == otherHandle)
    {
        /* Handle other attribute indications here */
        /* ... Missing code here ... */
    }
    else
    {
        /*! A GATT Client is trying to GATT Indicate an unknown attribute value.
         * This should not happen. Disconnect the link. */
        Gap_Disconnect (deviceId);
    }
}
```

OTAP Server demo does not have internal storage, so all commands are forwarded via the serial interface.

To send OTAP Commands to the OTAP Client the application running the OTAP Server calls the *OtapServer_SendCommandToOtapClient()* function, which performs an ATT Write operation on the OTAP Control Point attribute.

```
static void OtapServer_SendCommandToOtapClient
        (deviceId_t  otapClientDevId,
         void*       pCommand,
         uint16_t    cmdLength)
{
    /* GATT Characteristic to be written - OTAP Client Control Point */
    gattCharacteristic_t    otapCtrlPointChar;
    bleResult_t             bleResult;

    /* Only the value handle element of this structure is relevant for this operation. */
    otapCtrlPointChar.value.handle = mPeerInformation.customInfo.otapServerConfig.hControlPoint;
    otapCtrlPointChar.value.valueLength = 0;
    otapCtrlPointChar.cNumDescriptors = 0;
    otapCtrlPointChar.aDescriptors = NULL;

    bleResult = GattClient_SimpleCharacteristicWrite (mPeerInformation.deviceId,
                                                      &otapCtrlPointChar,
                                                      cmdLength,
                                                      pCommand);

    if (gBleSuccess_c == bleResult)
    {
        otapServerData.lastCmdSentToOtapClient = (otapCmdIdt_t)(((otapCommand_t*)pCommand)->
↪cmdId);
    }
    else
    {
        /*! A Bluetooth Low Energy error has occurred - Disconnect */
        (void)Gap_Disconnect (otapClientDevId);
    }
}
```

The ATT Confirmation for the ATT Write is received in the *BleApp_GattClientCallback()* set up earlier which receives a GATT procedure success message for a *gGattProcWriteCharacteristicValue_c* procedure type.

```
static void BleApp_GattClientCallback(
    deviceId_t              serverDeviceId,
    gattProcedureType_t     procedureType,
    gattProcedureResult_t   procedureResult,
    bleResult_t             error
)
{
   union
    {
        uint8_t                 errorTemp;
        attErrorCode_t              attErrorCodeTemp;
    }attErrorCodeVars;

    if (procedureResult == gGattProcError_c)
    {
        attErrorCodeVars.errorTemp = (uint8_t)error & 0xFFU;
        attErrorCode_t attError = attErrorCodeVars.attErrorCodeTemp;
        if (attError == gAttErrCodeInsufficientEncryption_c     ||
            attError == gAttErrCodeInsufficientAuthorization_c  ||
            attError == gAttErrCodeInsufficientAuthentication_c)
        {
    #if gAppUsePairing_d
            /* Start Pairing Procedure */
            (void)Gap_Pair (serverDeviceId, &gPairingParameters);
    #endif
```

(continues on next page)

(continued from previous page)

```
        }

        BleApp_StateMachineHandler (serverDeviceId, mAppEvt_GattProcError_c);
    }
    else if (procedureResult == gGattProcSuccess_c)
    {
        switch(procedureType)
        {
            /* ... Missing code here... */
            case gGattProcWriteCharacteristicValue_c:
            {
                BleApp_HandleValueWriteConfirmations (serverDeviceId);
            }
            break;

            default:
                ; /* For MISRA compliance */
           break;
        }

        BleApp_StateMachineHandler(serverDeviceId, mAppEvt_GattProcComplete_c);
    }
    else
    {
        ; /* For MISRA compliance */
    }
}
```

The *BleApp_HandleValueWriteConfirmations()* function deals with ATT Write Confirmations based on the requirements of the application.

There are two possible transfer methods for Image Chunks, the ATT transfer method and the L2CAP transfer method. The OTAP server is prepared to handle both, as requested by the OTAP Client.

To be able to use the L2CAP transfer method, the OTAP Server application must register a L2CAP LE PSM and 2 callbacks: a data callback and a control callback. This is done by using the *BluetoothLEHost_Initialized()* function.

```
/* Register OTAP L2CAP PSM */
  L2ca_RegisterLePsm (gOtap_L2capLePsm_c,
                      gOtapCmdImageChunkCocMaxLength_c); /*!< The negotiated MTU must be higher than␣
↪the biggest data chunk that is sent fragmented */
...
  App_RegisterLeCbCallbacks(BleApp_L2capPsmDataCallback, BleApp_L2capPsmControlCallback);
```

The data callback *BleApp_L2capPsmDataCallback()* is not used by the OTAP Server.

The control callback is used to handle L2CAP LE PSM connection requests from the OTAP Client and other events: PSM disconnections, No peer credits, and so on. The OTAP Client must initiate the L2CAP PSM connection if it wants to use the L2CAP transfer method.

```
static void BleApp_L2capPsmControlCallback(l2capControlMessage_t* pMessage)
{
    switch (pMessage->messageType)
    {
        case gL2ca_LePsmConnectRequest_c:
        {
            l2caLeCbConnectionRequest_t *pConnReq = &pMessage->messageData.connectionRequest;

            /* Respond to the peer L2CAP CB Connection request - send a connection response. */
            (void)L2ca_ConnectLePsm (gOtap_L2capLePsm_c,
```

(continues on next page)

(continued from previous page)

```
                        pConnReq->deviceId,
                        mAppLeCbInitialCredits_c);
            break;
        }
        case gL2ca_LePsmConnectionComplete_c:
        {
            l2caLeCbConnectionComplete_t *pConnComplete = &pMessage->messageData.connectionComplete;

            if (pConnComplete->result == gSuccessful_c)
            {
                /* Set the application L2CAP PSM Connection flag to TRUE because there is no gL2ca_
↪LePsmConnectionComplete_c
                 * event on the responder of the PSM connection. */
                otapServerData.l2capPsmConnected = TRUE;
                otapServerData.l2capPsmChannelId = pConnComplete->cId;

                if (pConnComplete->peerMtu > gOtap_l2capCmdMtuDataChunkOverhead_c)
                {
                    otapServerData.negotiatedMaxL2CapChunkSize = pConnComplete->peerMtu - gOtap_
↪l2capCmdMtuDataChunkOverhead_c;
                }
            }
            break;
        }
        case gL2ca_LePsmDisconnectNotification_c:
        {
            l2caLeCbDisconnection_t *pCbDisconnect = &pMessage->messageData.disconnection;

            /* Call App State Machine */
            BleApp_StateMachineHandler (pCbDisconnect->deviceId, mAppEvt_CbDisconnected_c);

            otapServerData.l2capPsmConnected = FALSE;
            break;
        }
        case gL2ca_NoPeerCredits_c:
        {
            l2caLeCbNoPeerCredits_t *pCbNoPeerCredits = &pMessage->messageData.noPeerCredits;
            (void)L2ca_SendLeCredit (pCbNoPeerCredits->deviceId,
                        otapServerData.l2capPsmChannelId,
                        mAppLeCbInitialCredits_c);
            break;
        }
        case gL2ca_Error_c:
        {
            /* Handle error */
            break;
        }
        default:
            ; /* For MISRA compliance */
            break;
    }
}
```

The ATT transfer method is supported by default but the L2CAP transfer method only works if the OTAP Client opens an L2CAP PSM credit-oriented channel.

To send data chunks to the OTAP Client the OTAP Server application calls the *OtapServer_SendCImgChunkToOtapClient()* function which delivers the chunk via the selected transfer method. For the ATT transfer method the chunk is sent via the *GattClient_CharacteristicWriteWithoutResponse()* function and for the L2CAP transfer method the chunk is sent via the *L2ca_SendLeCbData()* function.

```
static void OtapServer_SendCImgChunkToOtapClient(deviceId_t otapClientDevId,
                                                 void       pChunk,
                                                 uint16_t   chunkCmdLength)
{
    bleResult_t bleResult = gBleSuccess_c;
    if (otapServerData.transferMethod == gOtapTransferMethodAtt_c)
    {
        /* GATT Characteristic to be written without response - OTAP Client Data */
        gattCharacteristic_t otapDataChar;
        /* Only the value handle element of this structure is relevant for this operation. */
        otapDataChar.value.handle = mPeerInformation.customInfo.otapServerConfig.hData;
        bleResult = GattClient_CharacteristicWriteWithoutResponse
                                            (mPeerInformation.deviceId,
                                             &otapDataChar,
                                             chunkCmdLength,
                                             pChunk);
    }
    else if (otapServerData.transferMethod == gOtapTransferMethodL2capCoC_c)
    {
        bleResult = L2ca_SendLeCbData (mPeerInformation.deviceId,
                                       otapServerData.l2capPsmChannelId,
                                       pChunk,
                                       chunkCmdLength);
    }
    if (gBleSuccess_c != bleResult)
    {
        /*! A Bluetooth Low Energy error has occurred - Disconnect */
        Gap_Disconnect (otapClientDevId);
    }
}
```

The OTAP Server demo application relays all commands received from the OTAP Client to a PC through the FSCI type protocol running over a serial interface. It also directly relays all responses from the PC back to the OTAP Client.

Other implementations can bring the image to an external memory through other means of communication and directly respond to the OTAP Client requests.

Parent topic: Bluetooth Low Energy OTAP application integration

OTAP client

An application running an OTAP Client must wait for an OTAP Server to connect and perform service and characteristic discovery before performing any OTAP-related operations. OTAP transactions can begin only after the OTAP Server writes the OTAP Control point CCC Descriptor to receive ATT Notifications. After this is done, bidirectional communication is established between the OTAP Server and Client and OTAP transactions can begin.

The OTAP Client can advertise the OTAP Service via the demo application. Optionally, the OTAP Server may already know the advertising device has an OTAP Service based on application-specific means. In both situations, the OTAP Server must discover the handles of the OTAP Service and its characteristics.

In addition to the OTAP Service instantiated in the GATT Database, the OTAP Client needs to have some storage capabilities for the downloaded image file.

How to put the OTAP Service in the GATT Database is described in The OTAP Service and Characteristics.

The upgrade image storage capabilities in the demo OTAP Client applications are handled by the *OtaSupport* module from the Framework, which contains support modules and drivers. The *OtaSupport* module has support for both internal storage (a part of the internal flash memory is reserved for storing the upgrade image) and external storage (a SPI flash memory chip).

The demo applications use internal storage by default. The internal storage is viable only if there is enough space in the internal flash for the upgrade image – the flash in this case should be at least twice the size of the largest application. The *OtaSupport* module also needs the *Eeprom* module from the Framework to work correctly.

The OtaSupport module also includes functionality for configuring the OTACFG IFR sections after the image is received in order to enable the ROM bootloader to perform the actual image update.

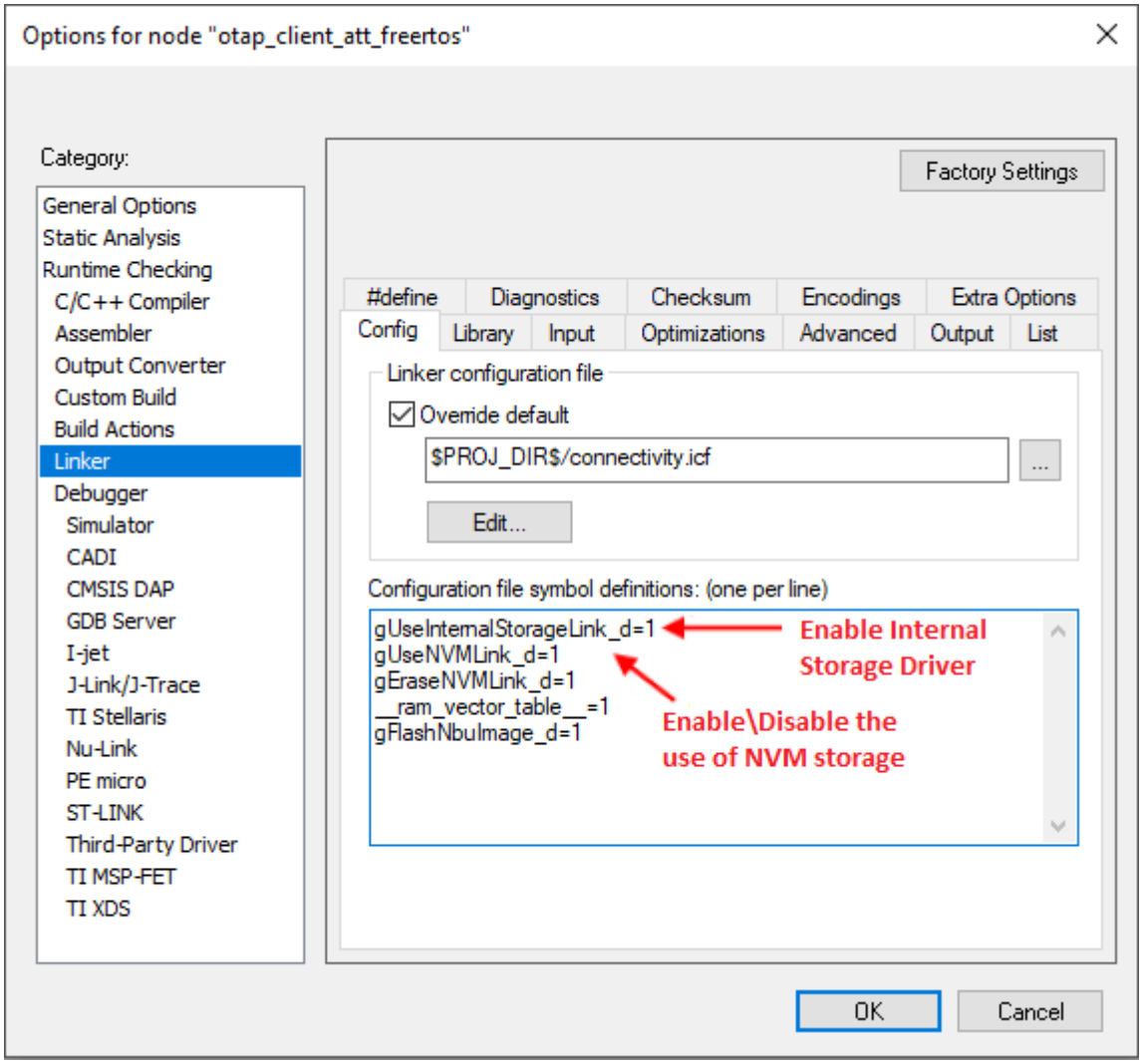
To use the *OtaSupport* module several configuration options must be set up in both the source files and the linker options of the toolchain.

To use internal storage, set up the gUseInternalStorageLink_d=1 symbol in the linker configuration window (**Linker->Config tab** in the IAR project properties) and set the gAppOtaExternalStorage_c value to (0) in the app_preinclude.hfile:

```
/*! Define as 1 to place OTA storage in external flash */
 #define gAppOtaExternalStorage_c (0)
```

The OTAP demo applications for the IAR EW IDE have some settings in the Linker options tab which must be configured to use OtaSupport and the OTAP Bootloader. In the **Project Target Options**->**Linker**->**Config** tab, 3 symbols must be correctly defined. To use NVM storage, thegUseNVMLink_d symbol must be set to 1. The gUseInternalStorageLink_d symbol must be set to 0 when OTAP external storage is used and to 1when the internal storage is used. The gEraseNVMLink_d must be set to 0.

An example linker configuration window for IAR is shown below.



Note: The gEraseNVMLink_d=1 IAR linker flag places some dummy bytes into the NVM region to invalidate the data and force the application to erase the entire NVM region. When generating an image for the OTA upgrade, this flag must be set to 0. This results in a smaller image size being transferred and lower power consumption. If the NVM region must be erased after the upgrade process, the "Preserve NVM" checkbox (from the Over The Air programming tool) should be unchecked.

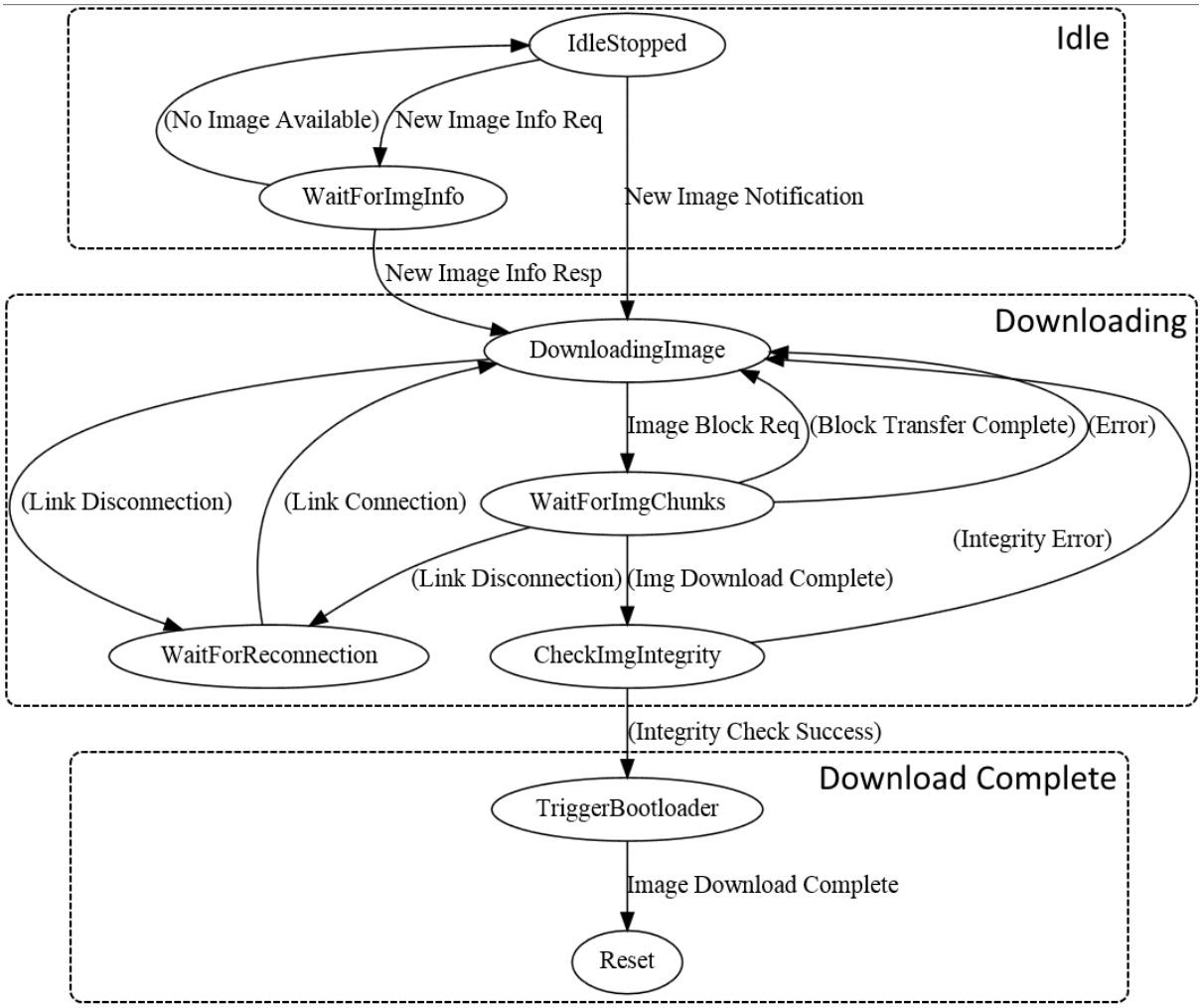
Once the application starts and bidirectional OTAP communication is established via the OTAP Service, then the OTAP Client must determine if the connected OTAP Server has a newer image than the one currently present on the device. This can be done in two ways:

- The OTAP Server knows by some application-specific means that it has a newer image and sends a New Image Notification to the OTAP Client or
- The OTAP Client sends a New Image Info Request to the OTAP Server and waits for a response. This example application uses the second method.

The New Image Info Request contains enough information about the currently running image to allow the OTAP Server to determine if it has a newer image for the requesting device. The New Image Info Response contains enough information for the OTAP Client to determine if the "dead-vertised" image is newer and it wants to download it. The best method is entirely dependent on application requirements.

An example function that checks if an *ImageVerison* field from a New Image Notification or a New Image Info Response corresponds to a newer image (based on the suggested format of this field) is provided in the OTAP Client demo applications. The function is called *OtapClient_IsRemoteImageNewer()*.

The OTAP Client application is a little more complicated than the OTAP Server application because more state information needs to be handled (current image position, current chunk sequence number, image file parsing information, and so on). An example state diagram for the OTAP Client is shown below. The Figure briefly lists the steps of the image download process. Note that some of the states may not be explicitly present in the demo applications.



After the OTAP Client determines that the peer OTAP Server has a suitable upgrade image available, it can start the download process. This is done by sending multiple Image Block Request messages and waiting for the Image Chunks via the selected transfer method.

While receiving the image file blocks, the OTAP Client application parses the image file. In case any parameter of an image file sub-element is invalid or the image file format is invalid, it sends an Error Notification to the OTAP Server and tries to restart the download process from the beginning or a known good position.

When an Image Chunk is received, its sequence number is checked and its content is parsed in the context of the image file format. If the sequence number is not as expected, then the block transfer is restarted from the last known good position. When all chunks of an Image Block are received, the next block is requested, if there are more blocks to download. When the last Image Block in an image file is received, then the image integrity is checked (the received CRC from the Image File CRC sub-element is compared to the computed CRC).

The computed image integrity initialization and intermediary value must be reset to '0' before starting or restarting an image download. If the image integrity check fails then the image download process is restarted from the beginning. If the image integrity check is successful, then the Image Download Complete message is sent to the OTAP Server, the OTACFG IFR is updated and the MCU is restarted. After the restart, the ROM bootloader kicks in and writes the new image to the flash memory, afterwards giving CPU control to the newly installed application.

If at any time during the download process, a Link Layer disconnection occurs, then the image download process is restarted from the last known good position when the link is re-established.

As noted earlier, the OTAP Client application needs to handle a lot of state information. In the demo application, all this information is held in the *otapClientData* structure of the *otapClien-*

tAppData_t type. The type is defined and the structure is initialized in the *otap_client.c* file of the application. This structure is defined and initialized differently for the OTAP Client ATT and L2CAP example applications. Mainly, the *transferMethod* member of the structure is constant and has different values for the two example applications and the L2CAP application structure has an extra member.

To receive write notifications when the OTAP Server writes the OTAP Control Point attribute and ATT Confirmations when it indicates the OTAP Control Point attribute, the OTAP Client application must register a GATT Server callback and enable write notifications for the OTAP Control Point attribute. This is done in the *BluetoothLEHost_Initialized()* function in the *otap_client_att.c/otap_client_l2cap_credit.c* file.

```
static void BluetoothLEHost_Initialized(void)
{
    /* ... Missing code here ... */

    /* Register stack callbacks */
    (void)App_RegisterGattServerCallback (BleApp_GattServerCallback);

    /* ... Missing code here ... */
}
```

The *BleApp_GattServerCallback()* function handles all incoming communication from the OTAP Server.

```
static void BleApp_GattServerCallback (deviceId_t deviceId, gattServerEvent_t* pServerEvent)
{
    switch (pServerEvent->eventType)
    {
        /* ... Missing code here ... */

        case gEvtCharacteristicCccdWritten_c:
        {
            OtapClient_CccdWritten (deviceId,
                                    pServerEvent->eventData.charCccdWrittenEvent.handle,
                                    pServerEvent->eventData.charCccdWrittenEvent.newCccd);
        }
        break;

       case gEvtAttributeWritten_c:
        {
            OtapClient_AttributeWritten (deviceId,
                                         pServerEvent->eventData.attributeWrittenEvent.handle,
                                         pServerEvent->eventData.attributeWrittenEvent.cValueLength,
                                         pServerEvent->eventData.attributeWrittenEvent.aValue);
        }
       break;

        case gEvtAttributeWrittenWithoutResponse_c:
        {
            OtapClient_AttributeWrittenWithoutResponse (deviceId,
                                                        pServerEvent->eventData.attributeWrittenEvent.handle,
                                                        pServerEvent->eventData.attributeWrittenEvent.cValueLength,
                                                        pServerEvent->eventData.attributeWrittenEvent.aValue);
        }
        break;

        case gEvtHandleValueConfirmation_c:
        {
            OtapClient_HandleValueConfirmation (deviceId);
        }
       break;*
```

(continues on next page)

(continued from previous page)

```
        /* ... Missing code here ... */

        default:
            ; /* For MISRA compliance */
        break;
    }
}
```

When the OTAP Server Writes a CCCD the *BleApp_GattServerCallback()* function calls the *OtapClient_CccdWritten()* function which sends a New Image Info Request when the OTAP Control Point CCCD is written it – this is the starting point of OTAP transactions in the demo applications.

When an ATT Write Request is made by the OTAP Server the the *BleApp_GattServerCallback()* function calls the *OtapClient_AttributeWritten()* function which handles the data as an OTAP command. Only writes to the OTAP Control Point are handled as OTAP commands. For each command received from the OTAP Server there is a separate handler function which performs required OTAP operations. These are:

- *OtapClient_HandleNewImageNotification()*
- *OtapClient_HandleNewImageInfoResponse()*
- *OtapClient_HandleErrorNotification()*

When an ATT Write Command (GATT Write Without Response) is sent by the OTAP Server the *BleApp_GattServerCallback()* function calls the *OtapClient_AttributeWrittenWithoutResponse()* function which handles Data Chunks if the selected transfer method is ATT and returns an error if any problems are encountered. Data chunks are handled by the *OtapClient_HandleDataChunk()* function.

```
static void BleApp_AttributeWrittenWithoutResponse (deviceId_t deviceId,
                                                    uint16_t handle,
                                                    uint16_t length,
                                                    uint8_t* pValue)
{
    /* ... Missing code here ... */
    f (handle == value_otap_data)
    {
        /* ... Missing code here ... */
        if (otapClientData.transferMethod == gOtapTransferMethodAtt_c)
        {
            if (((otapCommand_t*)pValue)->cmdId == gOtapCmdIdImageChunk_c)
            {
                OtapClient_HandleDataChunk (deviceId,
                                            length,
                                            pValue);
            }
        }
        /* ... Missing code here ... */
    }
    /* ... Missing code here ... */
}
```

Finally, when an ATT Confirmation is received for a previously sent ATT Indication the *BleApp_GattServerCallback()* function calls the *OtapClient_HandleValueConfirmation()* function, which performs the necessary OTAP operations based on the last sent command to the OTAP Server. This is done using separate confirmation handling functions for each command that is sent to the OTAP Server. These functions are:

- *OtapClient_HandleNewImageInfoRequestConfirmation()*
- *OtapClient_HandleImageBlockRequestConfirmation()*

- *OtapClient_HandleImageTransferCompleteConfirmation()*
- *OtapClient_HandleErrorNotificationConfirmation()*
- *OtapClient_HandleStopImageTransferConfirmation()*

Outgoing communication from the OTAP Client to the OTAP Server is done using the *OtapCS_SendCommandToOtapServer()* function. This function writes the value to be indicated to the OTAP Control Point attribute in the GATT database and then calls the *OtapCS_SendControlPointIndication()*which checks if indications are enabled for the target device and sends the actual ATT Indication. Both functions are implemented in the *otap_service.c* file.

```
bleResult_t OtapCS_SendCommandToOtapServer (uint16_t serviceHandle,
                                            void* pCommand,
                                            uint16_t cmdLength)
{
    union
    {
        uint8_t*                uuid_char_otap_control_pointTemp;
        bleUuid_t*               bleUuidTemp;
    }bleUuidVars;

    uint16_t  handle;
    bleResult_t result;
    bleUuidVars.uuid_char_otap_control_pointTemp = uuid_char_otap_control_point;
    bleUuid_t* pUuid = bleUuidVars.bleUuidTemp;

    /* Get handle of OTAP Control Point characteristic */
    result = GattDb_FindCharValueHandleInService(serviceHandle,
                                                 gBleUuidType128_c, pUuid, &handle);

    if (result == gBleSuccess_c)
    {
        /* Write characteristic value */
        result = GattDb_WriteAttribute(handle,
                                       cmdLength,
                                       (uint8_t*)pCommand);

       if (result == gBleSuccess_c)
        {
            /* Send Command to the OTAP Server via ATT Indication */
            result = OtapCS_SendControlPointIndication (handle);
        }
    }

   return result;
}

static bleResult_t OtapCS_SendControlPointIndication (uint16_t handle)
{
    uint16_t     hCccd;
    bool_t       isIndicationActive;
    /* Get handle of CCCD */
    GattDb_FindCccdHandleForCharValueHandle (handle, &hCccd);
    Gap_CheckIndicationStatus (...);
    return GattServer_SendIndication (...);
}
```

The *otap_interface.h* file contains all the necessary information for parsing and building OTAP commands (packed command structures type definitions, command parameters enumerations, and so on).

For the two possible image transfer methods (ATT and L2CAP) there are two separate demo appli-

cations. To be able to use the L2CAP transfer method the OATP Client application must register a L2CAP LE PSM and 2 callbacks: a data callback and a control callback. This is done in the *OtapClient_Config()* function.

```
/* Register OTAP L2CAP PSM */
L2ca_RegisterLePsm (gOtap_L2capLePsm_c,
gOtapCmdImageChunkCocMaxLength_c); /*!< The negotiated MTU must be higher than the biggest data
↪chunk that is sent fragmented */
...
App_RegisterLeCbCallbacks(BleApp_L2capPsmDataCallback, BleApp_L2capPsmControlCallback);
```

The control callback is used to handle L2CAP LE PSM-related events: PSM disconnections, PSM Connection Complete, No peer credits, and so on.

```
static void BleApp_L2capPsmControlCallback(l2capControlMessage_t* pMessage)
{
    switch (pMessage->messageType)
    {
        case gL2ca_LePsmConnectRequest_c:
        {
            l2caLeCbConnectionRequest_t *pConnReq = &pMessage->messageData.connectionRequest;

            /* This message is unexpected on the OTAP Client, the OTAP Client sends L2CAP PSM connection
             * requests and expects L2CAP PSM connection responses.
             * Disconnect the peer. */
            (void)Gap_Disconnect (pConnReq->deviceId);

            break;
        }
        case gL2ca_LePsmConnectionComplete_c:
        {
            l2caLeCbConnectionComplete_t *pConnComplete = &pMessage->messageData.connectionComplete;

            /* Call the application PSM connection complete handler. */
            OtapClient_HandlePsmConnectionComplete (pConnComplete);

            break;
        }
        case gL2ca_LePsmDisconnectNotification_c:
        {
            l2caLeCbDisconnection_t *pCbDisconnect = &pMessage->messageData.disconnection;

            /* Call the application PSM disconnection handler. */
            OtapClient_HandlePsmDisconnection (pCbDisconnect);

            break;
        }
        case gL2ca_NoPeerCredits_c:
        {
            l2caLeCbNoPeerCredits_t *pCbNoPeerCredits = &pMessage->messageData.noPeerCredits;
            (void)L2ca_SendLeCredit (pCbNoPeerCredits->deviceId,
                                     otapClientData.l2capPsmChannelId,
                                     mAppLeCbInitialCredits_c);
            break;
        }
        case gL2ca_Error_c:
        {
            /* Handle error */
            break;
        }
        default:
            ; /* For MISRA compliance */
```

(continues on next page)

(continued from previous page)

```
        break;
    }
}
```

The OTAP Client must initiate the L2CAP PSM connection if it wants to use the L2CAP transfer method; this can be done using the *L2ca_ConnectLePsm()* function. The *L2ca_ConnectLePsm()* function is called by the *OtapClient_ContinueImageDownload()* if the transfer method is L2CAP and the PSM is found to be disconnected.

```
static void OtapClient_ContinueImageDownload (deviceId_t deviceId)
{
    /* ... Missing code here ... */
    /* Check if the L2CAP OTAP PSM is connected and if not try to connect and exit immediately. */
    if ((otapClientData.l2capPsmConnected == FALSE) &&
            (otapClientData.state != mOtapClientStateImageDownloadComplete_c))
    {
        L2ca_ConnectLePsm (gOtap_L2capLePsm_c,
                           deviceId,
                           mAppLeCbInitialCredits_c);
        bValidState = FALSE;;
    }
    /* ... Missing code here ... */
}
```

The PSM data callback *BleApp_L2capPsmDataCallback()* is used by the OTAP Client to handle incoming image file parts from the OTAP Server.

```
static void BleApp_L2capPsmDataCallback (deviceId_t     deviceId,
                                         uint16_t       lePsm,
                                         uint8_t*       pPacket,
                                         uint16_t       packetLength)
{
    OtapClient_HandleDataChunk (deviceId,
                                packetLength,
                                pPacket);
}
```

All data chunks regardless of their source (ATT or L2CAP) are handled by the *OtapClient_HandleDataChunk()* function. This function checks the validity of Image Chunk messages, parses the image file, requests the continuation or restart of the image download and triggers the bootloader when the image download is complete.

```
static void OtapClient_HandleDataChunk (deviceId_t deviceId, uint16_t length, uint8_t* pData);
```

The Image File CRC Value is computed on the fly as the image chunks are received using the *OTA_CrcCompute()* function from the *OtaSupport* module which is called by the *OtapClient_HandleDataChunk()* function. The *OTA_CrcCompute()* function has a parameter for the intermediary CRC value which must be initialized to 0 every time a new image download is started.

The actual write of the received image parts to the storage medium is also done in the *OtapClient_HandleDataChunk()* function using the *OtaSupport* module. This is achieved using the following functions:

- *OTA_StartImage()* – called before the start of writing a new image to the storage medium.
- *OTA_CancelImage()* – called whenever an error occurs and the image download process needs to be stopped/restarted from the beginning.
- *OTA_PushImageChunk()* – called to write a received image chunk to the storage medium. Note that only the Upgrade Image Sub-element of the image file is actually written to the storage medium.

- *OTA_CommitImage()* - called to set up what parts of the downloaded image are written to flash and other information for the bootloader. The Value field of the Sector Bitmap Sub-element of the Image File is given as a parameter to this function.
- *OTA_SetNewImageFlag()* - called to configure the OTACFG IFR when a new image has been successfully received. When the MCU is reset, the ROM bootloader transfers the new image from the storage medium to the program flash.

To continue the image download process after a block is transferred or to restart it after an error has occurred the *OtapClient_ContinueImageDownload()* function is called. This function is used in multiple situations during the image download process.

To summarize, an outline of the steps required to perform the image download process is shown below:

- Wait for a connection from an OTAP Server
- Wait for the OTAP Server to write the OTAP Control Point CCCD
- Ask or wait for image information from the server
- If a new image is available on the server, start the download process using the *OtapClient_ContinueImageDownload()* function.
  - If the transfer method is L2CAP CoC, then initiate a PSM connection to the OTAP Server
- Repeat while image download is not complete.
  - Wait for image chunks.
  - *Call the OtapClient_HandleDataChunk()* function for all received image chunks regardless of the selected transfer method.
    - Check image file header integrity using the *OtapClient_IsImageFileHeaderValid()* function.
    - Write the Upgrade Image Sub-element to the storage medium using *OtaSupport* module functions.
    - When the download is complete, check image integrity.
      - If the integrity check is successful, commit the image using the Sector Bitmap Sub-element and trigger the bootloader
      - If integrity check fails, restart the image download from the beginning
    - If the download is not complete, ask for a new image chunk.
  - If any error occurs during the processing of the image chunk, restart the download from the last known good position.
- If an image was successfully downloaded and transferred to the storage medium and the bootloader triggered, then reset the MCU to start the flashing process of the new image.

Parent topic: Bluetooth Low Energy OTAP application integration

Parent topic:*Over the Air Programming (OTAP)*

Secured OTAP

The security features of the KW45 devices enable them to use secured OTAP, meaning the new images can be authenticated and encrypted. The decryption/authentication keys are programmed into hardware fuses. For information on how to prepare the board, refer to the accompanying document related to board provisioning. For information on how to obtain the secured image, refer to the *Bluetooth Low Energy Demo Applications User's Guide (BLEDAUG)*.

Parent topic:*Over the Air Programming (OTAP)*

Creating a Bluetooth LE application when the Host Stack runs on another processor

This section describes how to create a Bluetooth Low Energy application (host), when the Bluetooth Low Energy Host Stack is running on another processor (blackbox). The section alsp provides sample code to explain how to achieve this.

The supported serial interfaces between the two chips (application and the Bluetooth Low Energy Host Stack) are UART, SPI, or USB.

Typical applications employing Bluetooth LE Host Stack blackboxes are host systems such as a PC tool or an embedded system that has an application implementation. This chapter describes an embedded application.

For more information, refer to *Bluetooth Low Energy Host Stack FSCI Reference Manual*. This document provides explicit information on exercising the Bluetooth Low Energy Host Stack functionality through a serial communication interface to a host system.

Serial manager and FSCI configuration

For creating an embedded application that communicates with the Bluetooth Low Energy Host Stack using the serial interface, the following steps must be done:

Serial manager initialization

The function that must be called for Serial Manager initialization is located in *SerialManager.h*:

```
/* Init serial manager */
SerialManager_Init();
```

Parent topic: Serial manager and FSCI configuration

FSCI configuration and initialization

By default, the FSCI module is disabled. It must be enabled by setting gFsciIncluded_c to 1. Also, gFsciLenHas2Bytes_c must be set to 1 because Bluetooth Low Energy Host Stack interface commands and events need serial packets bigger than 255 octets.

For more information on the following configuration parameters, refer to the FSCI chapter of the *Connectivity Framework Reference Manual*.

To configure the FSCI module, the following parameters can be set on both the Bluetooth Low Energy Application project and the Bluetooth Low Energy FSCI blackbox:

```
/* Enable/Disable FSCI */
#define gFsciIncluded_c                 1

/* Enable/Disable FSCI Low Power Commands*/
#define gFSCI_IncludeLpmCommands_c      0

/* Defines FSCI length - set this to FALSE is FSCI length has 1 byte */
#define gFsciLenHas2Bytes_c             1

/* Defines FSCI maximum payload length */
#define gFsciMaxPayloadLen_c            1660

/* Enable/Disable Ack transmission */
#define gFsciTxAck_c                    0

/* Enable/Disable Ack reception */
#define gFsciRxAck_c                    0

/* Enable FSCI Rx restart with timeout */
#define gFsciRxTimeout_c                1
#define mFsciRxTimeoutUsePolling_c      1
```

(continues on next page)

(continued from previous page)

```
/* Use Misra Compliant version of FSCI module */
#define gFsciUseDedicatedTask_c          1

/* FSCI task size */
#if defined(DEBUG)
#define gFsciTaskStackSize_c             4600
#else
#define gFsciTaskStackSize_c             2600
#endif
```

To perform the FSCI module initialization, the following code can be used:

```
/* Define fsci serial manager handle */
  #if defined(gFsciIncluded_c) && (gFsciIncluded_c > 0)
    extern serial_handle_t g_fsciHandleList[gFsciIncluded_c];
  #endif /*gFsciIncluded_c > 0*/

void BluetoothLEHost_AppInit(void)
{
    /* Init FSCI */
    FSCI_commInit( g_fsciHandleList );

    /* Register BLE handlers in FSCI */
    fsciBleRegister(0);
    ...
}
```

Parent topic: Serial manager and FSCI configuration

FSCI handlers (GAP, GATT, and GATTDB) registration

For receiving messages from all the Bluetooth Low Energy Host Stack serial interfacing layers (GAP, GATT, and GATTDB), a function handler must be registered in FSCI for each layer:

```
fsciBleRegister(0);
```

Parent topic: Serial manager and FSCI configuration

Parent topic:*Creating a Bluetooth LE application when the Host Stack runs on another processor*

Bluetooth Low Energy Host Stack initialization

The Bluetooth Low Energy Host Stack must be initialized when platform setup is complete and all RTOS tasks have been started. This initialization is done by restarting the black-box using a FSCI CPU Reset Request command. This is performed automatically by the ***Ble_Initialize(App_GenericCallback)*** function.

```
/* Send FSCI CPU reset command to BlackBox */
FSCI_transmitPayload(gFSCI_ReqOpcodeGroup_c, mFsciMsgResetCPUReq_c, NULL, 0, fsciInterface);
```

The completion of the Bluetooth Low Energy Host Stack initialization is signaled by the reception of the *GAP-GenericEventInitializationComplete.Indication* event (over the serial communication interface, in FSCI). The *Bluetooth Low Energy-HostInitialize.Request* command is not required to be sent to the blackbox (the entire initialization is performed by the blackbox, when it resets).

Parent topic:*Creating a Bluetooth LE application when the Host Stack runs on another processor*

GATT database configuration

The GATT database always resides on the same processor as the entire Bluetooth Low Energy Host Stack, so the attributes must be added by the host application using the serial communication interface.

To create a GATT database remotely, *GATTDBDynamic* commands must be used. The GATTDBDynamic API is provided to the user that performs all the required memory allocations and sends the FSCI commands to the blackbox. The result of the operation is returned, including optionally the service, characteristic, and 'cccd' handles returned by the blackbox.

Current supported API for adding services is the following:

```
bleResult_t GattDbDynamic_AddGattService (gattServiceHandles_t* pOutServiceHandles);
bleResult_t GattDbDynamic_AddGapService (gapServiceHandles_t* pOutServiceHandles);
bleResult_t GattDbDynamic_AddIpssService (ipssServiceHandles_t* pOutServiceHandles);
bleResult_t GattDbDynamic_AddHeartRateService (heartRateServiceHandles_t* pOutServiceHandles);
bleResult_t GattDbDynamic_AddBatteryService (batteryServiceHandles_t* pOutServiceHandles);
bleResult_t GattDbDynamic_AddDeviceInformationService (deviceInfoServiceHandles_t*
↪pOutServiceHandles);
```

The service handles are optional.

Also, a generic function is provided, so that the user can add any generic service to the database:

```
bleResult_t GattDbDynamic_AddServiceInDatabase (serviceInfo_t* pServiceInfo);
```

Usually, a Bluetooth Low Energy Application is ported from a single chip solution, where the Bluetooth Low Energy Application and the Bluetooth Low Energy stack reside on the same processor and the GATT database is populated statically. The user should remove all the attribute handles from any structure and replace them with *gGattDbInvalidHandle_d*. The attribute handles should be populated after the services are added dynamically to the database with the handles returned by the previous API.

Parent topic:*Creating a Bluetooth LE application when the Host Stack runs on another processor*

FSCI host layer

The Bluetooth Low Energy GAP, GATT, GATTDB, and L2CAP APIs are included in the Bluetooth Low Energy interface. When these APIs reside on a separate processor than the Bluetooth Low Energy stack, they are implemented as an FSCI Host Layer that should be added to the Bluetooth Low Energy Application project.

This layer is responsible for serializing API to the corresponding FSCI commands. The layer also sends these APIs to the blackbox, receives and deserializes FSCI statuses and events, presents them to the Bluetooth Low Energy Application, and arbitrates access from multiple tasks to the serial interface.

All the GAP, GATT, GATTDB, and L2CAP APIs are executed asynchronously, so the user context blocks waiting for the response from the blackbox. The response can be the status of the request or optionally an FSCI event, which includes the output parameters of a synchronous function.

There are also functions without parameters that are not executed synchronously and they are provided asynchronously through a later FSCI event. It is the responsibility of the FSCI Host layer to keep the application-allocated memory between the time of the request and the completion of the event with the actual values of the output parameters and populate them accordingly.

The Bluetooth Low Energy API execution inside the FSCI Host layer first waits for gaining access to the serial interface through a mutex. Once the access is gained, the FSCI request is sent to the serial interface to the blackbox. Then, by default, the serial interface response is received by polling until the whole FSCI packet is received. The other option available is to block the user task to wait for an OS event that is set by the FSCI module when the status is received. For more information on the FSCI module, see the Connectivity Framework Reference Manual. See *References*.

The API can have output parameters that are to be received immediately after the status of the request. In such as case, if the status of the request is 'success', the polling mechanism continues to receive the whole FSCI packet of the Bluetooth Low Energy event. The output parameters are obtained and the values are filled in the memory space provided by the application. After obtaining the status and optionally the event, the execution of the request is considered completed, the

mutex to the serial interface is unlocked, and the execution flow is returned to the user calling context.

Parent topic:*Creating a Bluetooth LE application when the Host Stack runs on another processor*

Intrusion Detection System

Overview

The Intrusion Detection System (IDS) provides a comprehensive security monitoring framework for Bluetooth Low Energy applications. This interface defines various security events that can be detected across different layers of the Bluetooth LE stack and provides mechanisms to register callbacks and handle security incidents.

Event Types

The idsEventType_t enumeration defines various security events that can be detected:

Pairing-Related Events

gSecEvt_Sweyntooth_MaxEncKeySize_c (BIT0)

- **Attack**: Sweyntooth - Max Encryption Key Size
- **Target**: Pairing process
- **Stack Location**: Host
- **Description**: Attacker sends a Pairing Request with a value for the Encryption Key Size field which exceeds the maximum allowed (16 bytes), hoping to later trigger a buffer overflow on the victim device.
- **Preventive Action**: Sanity check the fields of received SMP messages.

gSecEvt_DhKeyCheckZero_c (BIT1)

- **Attack**: Deviating Behavior - DHKey Check Zero
- **Target**: Pairing process
- **Stack Location**: Host
- **Description**: During the pairing process, the attacker sends a DHKey Check message with all fields set to zero, hoping to bypass this step of the process.
- **Preventive Action**: Sanity check the fields of received SMP messages.

gSecEvt_UnexpectedSmpMessage_c (BIT2)

- **Attack**: Deviating Behavior - Unexpected SMP message
- **Target**: Pairing process
- **Stack Location**: Host
- **Description**: During the pairing process, an SMP message is received out of turn. Attacker may try to bypass one of the steps of the process.
- **Preventive Action**: Ensure the SMP state machine does not allow out of turn messages.

gSecEvt_IdenticalPeerPublicKey_c (BIT3)

- **Attack**: ANSSI - Identical Public Key
- **Target**: Pairing process
- **Stack Location**: Host

- **Description**: During the pairing process, the attacker sends a Public Key whose X coordinate is equal to the X coordinate of the victim device's Public Key. This is a precursor step to a successful attack.
- **Preventive Action**: Check the peer's Public Key X coordinate and do not accept it if it is equal to our own.

gSecEvt_ForcedRepairing_c (BIT13)

- **Attack**: Forced re-pairing
- **Target**: Pairing process
- **Stack Location**: Host
- **Description**: A bonded peer rejects link encryption and, if it is a GAP central, attempts to pair again. This could indicate an attempt to downgrade the security of an existing connection or to force a new pairing process that may be vulnerable to attacks.
- **Preventive action**: Signal the application and allow it to decide whether to accept the new pairing request or terminate the connection.

gSecEvt_KNOB_c (BIT16)

- **Attack**: Key Negotiation over Bluetooth
- **Target**: Pairing process
- **Stack Location**: Host
- **Description**: During the pairing process, an attacker attempts to negotiate a lower encryption key size than the minimum allowed (7 bytes), weakening the security of the connection.
- **Preventive Action**: Validate that the negotiated key size meets minimum security requirements.

gSecEvt_InvalidPublicKey_c (BIT18)

- **Attack**: Invalid public key
- **Target**: Pairing process
- **Stack Location**: Host
- **Description**: During the pairing process, the attacker sends an invalid Public Key that does not conform to the expected elliptic curve parameters, attempting to cause processing errors or bypass key validation checks.
- **Preventive Action**: Validate that the received Public Key conforms to the expected elliptic curve and reject any malformed keys.

gSecEvt_WrongConfirmValue_c (BIT19)

- **Attack**: Wrong confirm value
- **Target**: Pairing process
- **Stack Location**: Host
- **Description**: During the pairing process, the attacker sends a Confirm Value that does not match the expected cryptographic computation, attempting to bypass the confirmation step of the pairing process.
- **Preventive Action**: Verify that the received Confirm Value matches the computed expected value and reject the pairing if it does not.

gSecEvt_DHKeyCheckFailed_c (BIT20)

- **Attack**: DH Key Check failed
- **Target**: Pairing process
- **Stack Location**: Host

- **Description**: During the pairing process, the attacker sends a DHKey Check message that fails cryptographic verification, attempting to bypass the key confirmation step of the pairing process.
- **Preventive Action**: Verify that the received DHKey Check value matches the computed expected value and reject the pairing if it does not.

gSecEvt_SameConfirmValue_c (BIT21)

- **Attack**: Confirm Value Replay Attack (Same Confirm Value)
- **Target**: Pairing process
- **Stack Location**: Host
- **Description**: During Legacy Pairing Phase 2, the attacker (acting as Responder) sends LP_CONFIRM_R equal to LP_CONFIRM_I received from the Initiator, attempting to bypass the confirm value verification step and force the Initiator into an incorrect STK encryption attempt. Per Core Spec 6.3 Section 2.3.5.5, the Initiator SHALL verify that LP_CONFIRM_R is not equal to LP_CONFIRM_I.
- **Preventive Action**: Verify that LP_CONFIRM_R != LP_CONFIRM_I and reject the pairing with Confirm Value Failed (0x04) if equal. The IDS event allows the application to take additional defensive action (e.g., disconnect, block device).

Connection-Related Events

gSecEvt_MicFailureDisconnect_c (BIT5)

- **Attack**: MIC Failure Disconnect
- **Target**: Connection
- **Stack Location**: Host
- **Description**: A packet received from the peer device fails MIC (Message Integrity Check) verification, indicating potential tampering or corruption of the encrypted data. This could indicate an active attack or a corrupted connection.
- **Preventive Action**: Disconnect from the peer device and report the security event for monitoring.

gSecEvt_EncReqNonBonded_c (BIT6)

- **Attack**: Encryption Request from Non-Bonded Device
- **Target**: Connection
- **Stack Location**: Host
- **Description**: An encryption request is received from a device that is not bonded with the local device. This could indicate an attempt to establish an encrypted connection without proper pairing, potentially allowing unauthorized access.
- **Preventive Action**: Reject encryption requests from non-bonded devices and require proper pairing before allowing encrypted connections.

gSecEvt_MalformedL2cap_c (BIT7)

- **Attack**: Malformed L2CAP Packet
- **Target**: Connection
- **Stack Location**: Host
- **Description**: An L2CAP packet is received with malformed structure or invalid parameters, potentially indicating an attempt to exploit protocol parsing vulnerabilities or cause processing errors.

- **Preventive Action**: Validate L2CAP packet structure and parameters, reject malformed packets, and disconnect from the peer device if violations are detected.

gSecEvt_MalformedAtt_c (BIT8)

- **Attack**: Malformed ATT Packet
- **Target**: Connection
- **Stack Location**: Host
- **Description**: An ATT packet is received with malformed structure or invalid parameters, potentially indicating an attempt to exploit protocol parsing vulnerabilities, cause processing errors, or bypass attribute access controls.
- **Preventive Action**: Validate ATT packet structure and parameters, reject malformed packets, and disconnect from the peer device if violations are detected.

gSecEvt_L2capFragmentsExceedMTU_c (BIT14)

- **Attack**: Blue Frag
- **Target**: Connection
- **Stack Location**: Host
- **Description**: An attacker sends L2CAP fragments that exceed the negotiated Maximum Transmission Unit (MTU), attempting to cause buffer overflow vulnerabilities or bypass MTU validation checks during packet reassembly.
- **Preventive Action**: Validate that L2CAP fragment sizes do not exceed the negotiated MTU, reject oversized fragments, and disconnect from the peer device if violations are detected.

Application-Related Events

gSecEvt_UnauthenticatedServer_c (BIT4)

- **Attack**: BLESA
- **Target**: Application
- **Stack Location**: Application
- **Description**: An attacker attempts to access GATT database attributes or services without proper authentication, potentially allowing unauthorized read/write operations on sensitive characteristics.
- **Preventive Action**: Ensure that the peer is properly authenticated before allowing access to protected attributes, enforce attribute-level access controls, and disconnect from the peer device if unauthorized access attempts are detected. This is an application-level operation.

gSecEvt_UnauthorizedDbAccess_c (BIT9)

- **Attack**: Unauthorized GATT Database Access
- **Target**: Application
- **Stack Location**: Application
- **Description**: An attacker attempts to perform unauthorized read or write operations on GATT database attributes without proper authorization, potentially allowing access to sensitive data or modification of critical characteristics.
- **Preventive Action**: Enforce attribute-level authorization checks, validate that the peer has the required permissions before allowing read/write operations, and disconnect from the peer device if unauthorized access attempts are detected. This is an application-level operation.

Usage Example

The user must set the gIntrusionDetectionSystem_d define to 1 in the application's app_preinclude.h header. The IDS_RegisterCallback() API is used to register a callback for events specified via an event mask.

```
/* Register for specific security events */
uint32_t eventMask = gSecEvt_MicFailureDisconnect_c |
                     gSecEvt_UnexpectedSmpMessage_c |
                     gSecEvt_MalformedAtt_c;

IDS_RegisterCallback(securityEventHandler, eventMask);
```

The security events can then be handled in the registered callback.

```
/* Callback function to handle security events */
void securityEventHandler(idsEventData_t *pEventData) {
    /* Handle the security event based on type */
    switch(pEventData->type) {
        case gSecEvt_MicFailureDisconnect_c:
            /* Handle MIC failure */
            break;
        case gSecEvt_UnexpectedSmpMessage_c:
            /* Handle unexpected SMP message */
            break;
        /* ... handle other events */
    }
}
```

The IDS provides comprehensive security monitoring capabilities for Bluetooth LE applications, helping to detect and respond to various known attack vectors and security vulnerabilities.

References

For more information, refer to the following documents:

- Bluetooth Low Energy Demo Applications User's Guide (*KW45_BLEDAUG*)
- Bluetooth Low Energy Host Stack API Reference Manual (*KW45_BLEHSAPIRM*)
- Bluetooth Low Energy Host Stack FSCI (Framework Serial Connectivity Interface) API Reference Manual (*KW45_BLEHSFSCIAPIRM*)
- Connectivity Framework Reference Manual (*KW45_CONNFWRM*)
- Bluetooth Low Energy CCC Digital Key R3 Application Note (*AN12791*)

Acronyms and abbreviations

The following acronyms are used in this document.

Acronym	Description
Bluetooth LE	Bluetooth Low Energy
CCCD	Client Characteristic Configuration Descriptor
ELKE	EdgeLock Secure Enclave
FSCI	Framework Serial Connectivity Interface
GAP	Generic Access Profile
GATT	Generic Attribute Profile
GATTDB	Generic Attribute Profile Database
HCI	Host Controller Interface
IRK	Identity Resolving Key
LTK	Long Term Key
LL	Link Layer
L2CAP	Logical Link Control and Adaptation Protocol
MTU	Maximum Transmission Unit
PDU	Protocol Data Unit
PAwR	Periodic Advertising with Responses
RPA	Resolvable Private Address
RSSI	Received Signal Strength Indicator
RTOS	Real Time Operating System
RX	Receiver
SDK	Software Development Kit
TX	Transmitter
WFI	Wait For Interrupt

Note about the source code in the document

Example code shown in this document has the following copyright and BSD-3-Clause license:

Copyright 2022-2025 NXP Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Revision history

This table summarizes revisions to this document.

Document ID	Release date	Description
UG10184 v.1.1	06 April 2026	Added support for FRDM-KW43 platform. Updated Advanced Secure Mode section
UG10184 v.1.0	26 November 2024	Document is aligned to KW47 EAR 2.1 24.12.00-pvw2 release

Legal information

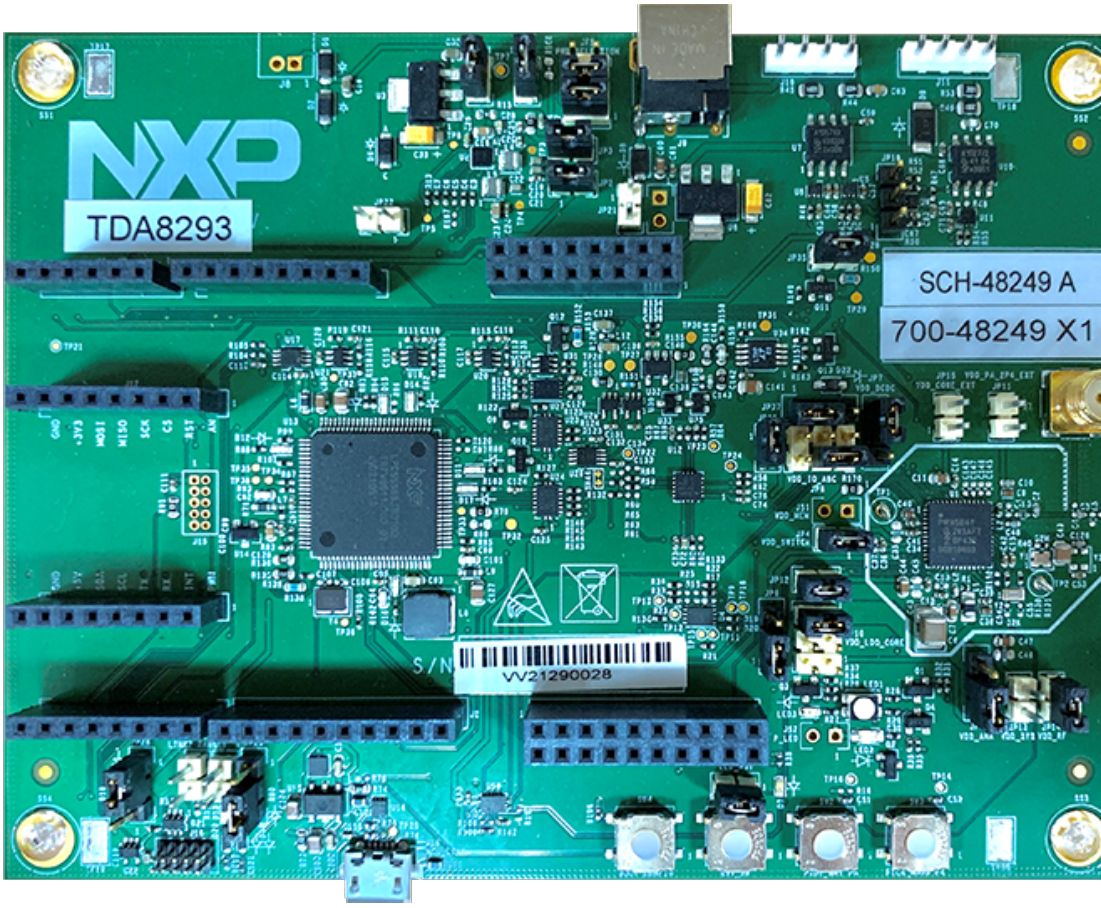
KW45, MCXW71, KW47, MCXW72, KW43 Bluetooth Low Energy Software Quick Start Guide Introduction

This document briefly describes the process of using NXP Bluetooth Low Energy Software for the KW45, MCXW71, KW47, MCXW72 and KW43 wireless microcontroller platforms (version 1.1.0). It lists the hardware setup and steps for building and usage of the provided demo applications.

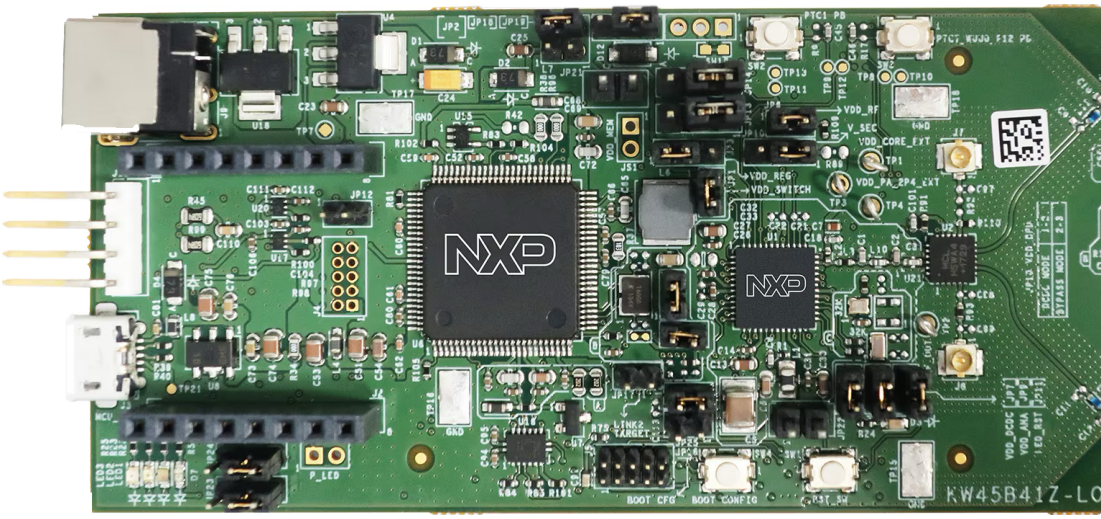
Hardware setup

The examples described in this document use a KW45B41Z-EVK, KW45B41Z-LOC, MCX-W71-EVK, FRDM-MCXW71, KW47-EVK, KW47-LOC, KW43-LOC, MCX-W72-EVK, FRDM-MCXW72 or FRDM-KW43 as the development platform, as shown in the figures below.

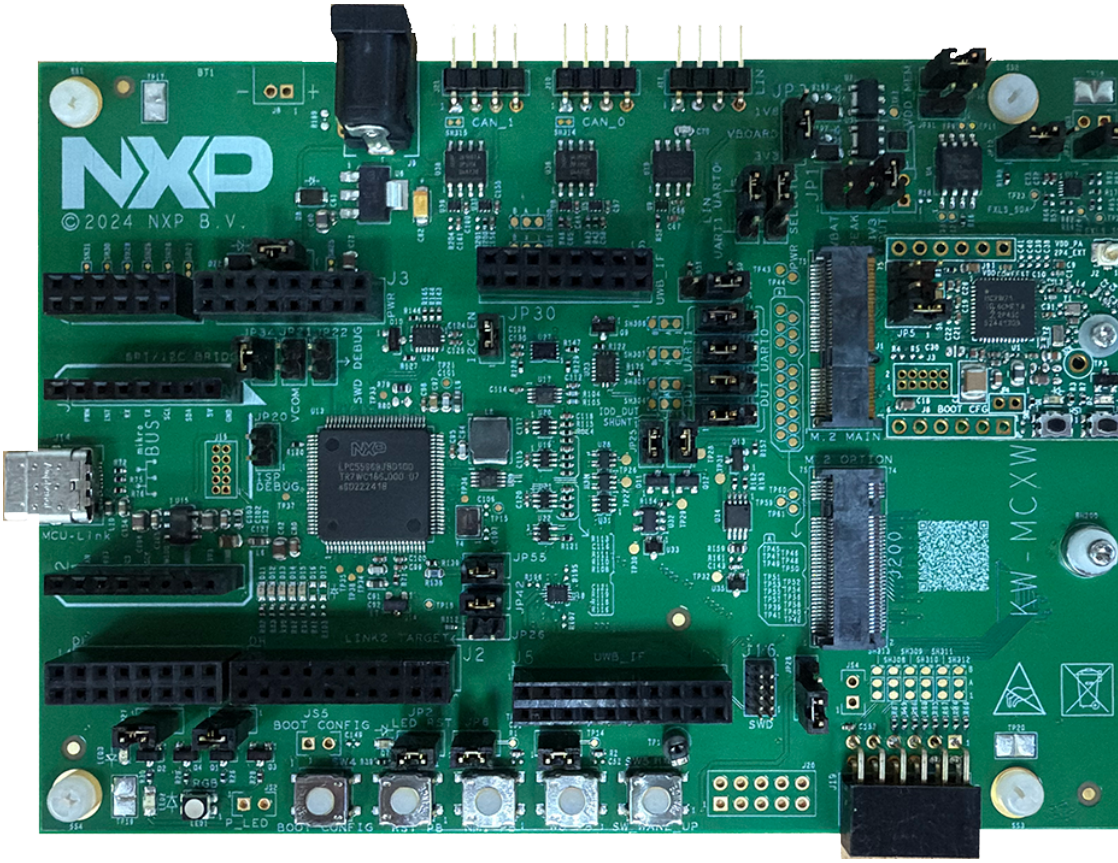
- The default interface selected in the IAR Embedded Workbench for Arm projects included in this release is below:
  - CMSIS-DAP for kw45b41zevk, kw45b41zloc, mcxw71evk, frdmmcxw71, frdm-kw43 platforms.
  - JLink for kw47evk, kw47loc, kw43loc, mcxw72evk, frdmmcxw72 and frdm-kw43 platforms.
- Use jumpers to configure the boards in one of the available power configurations (refer to the specific board documentation).
- On all boards, the OpenSDA USB port is connected to a Windows PC. The OpenSDA chip on the board requires flashing with appropriate firmware with debugging and virtual serial COM port capabilities.



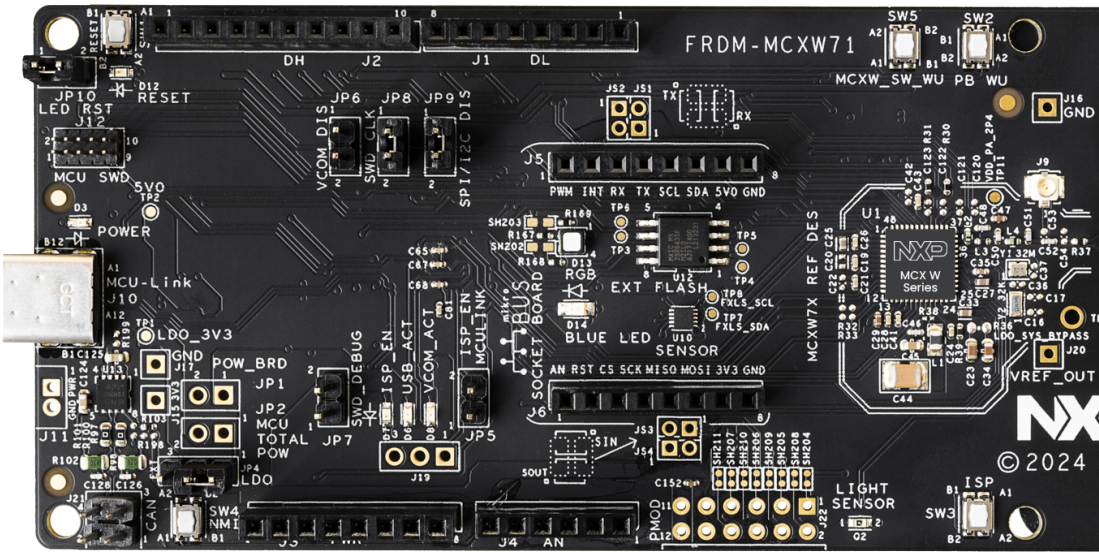
KW45B41Z-EVK board



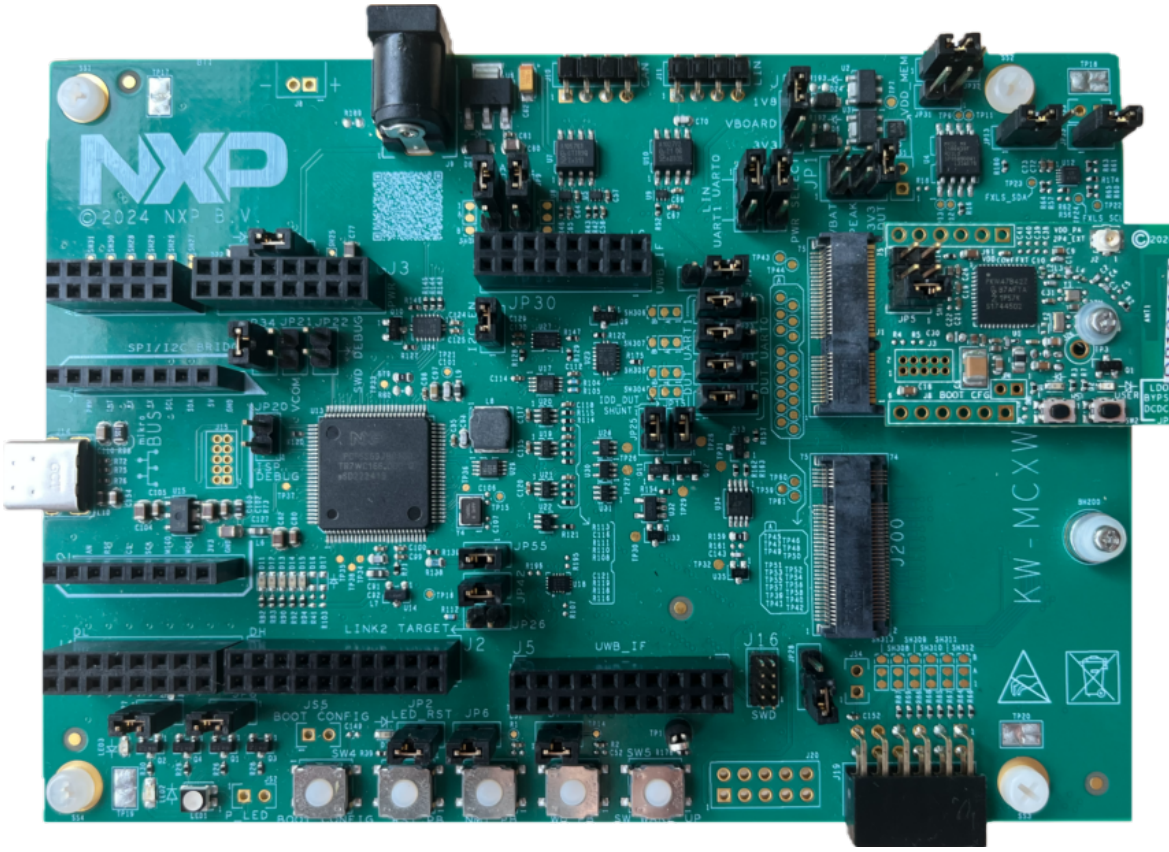
KW45B41Z-LOC board



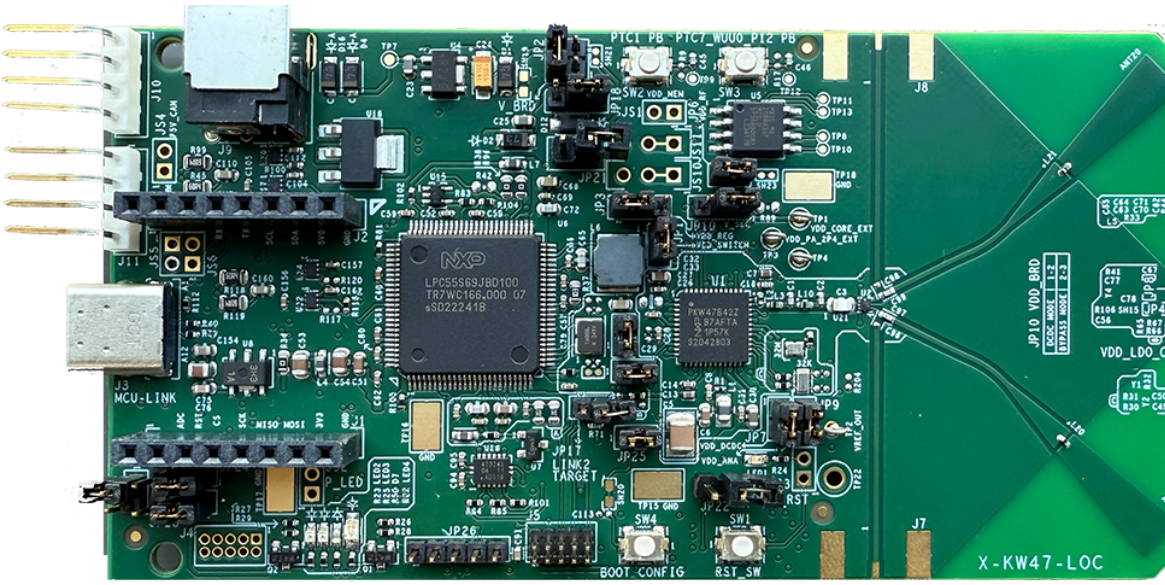
MCX-W71-EVK board



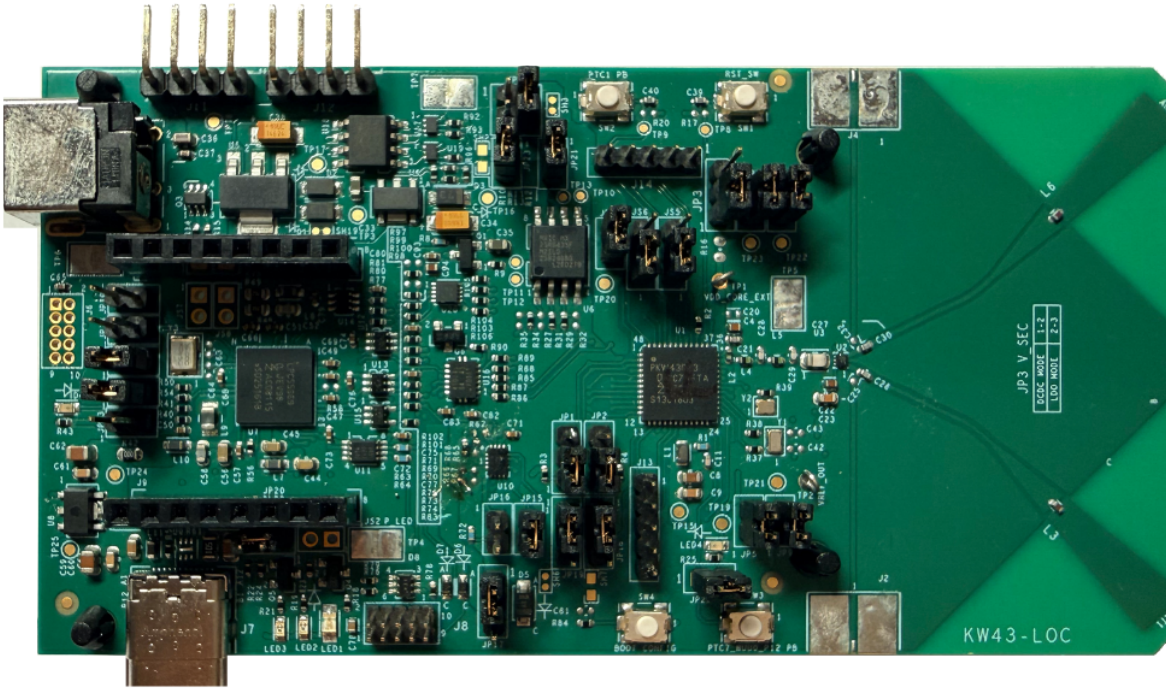
FRDM-MCXW71 board



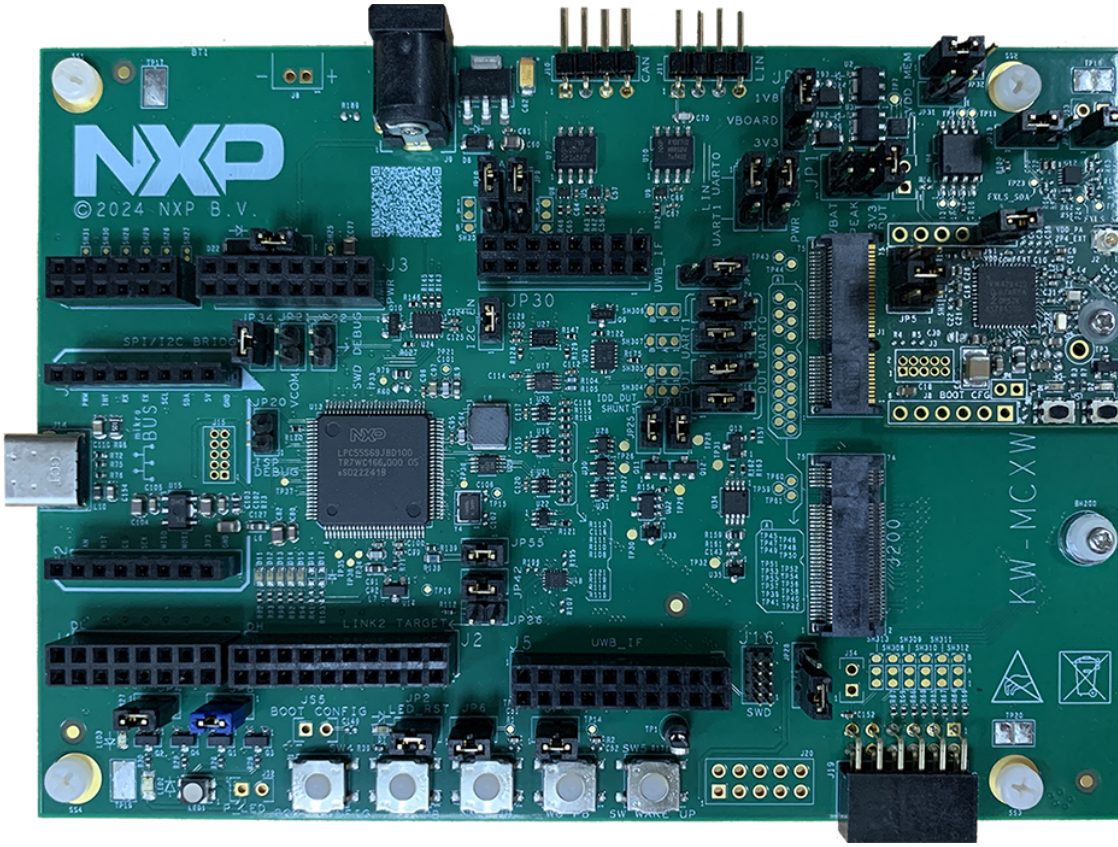
KW47-EVK board



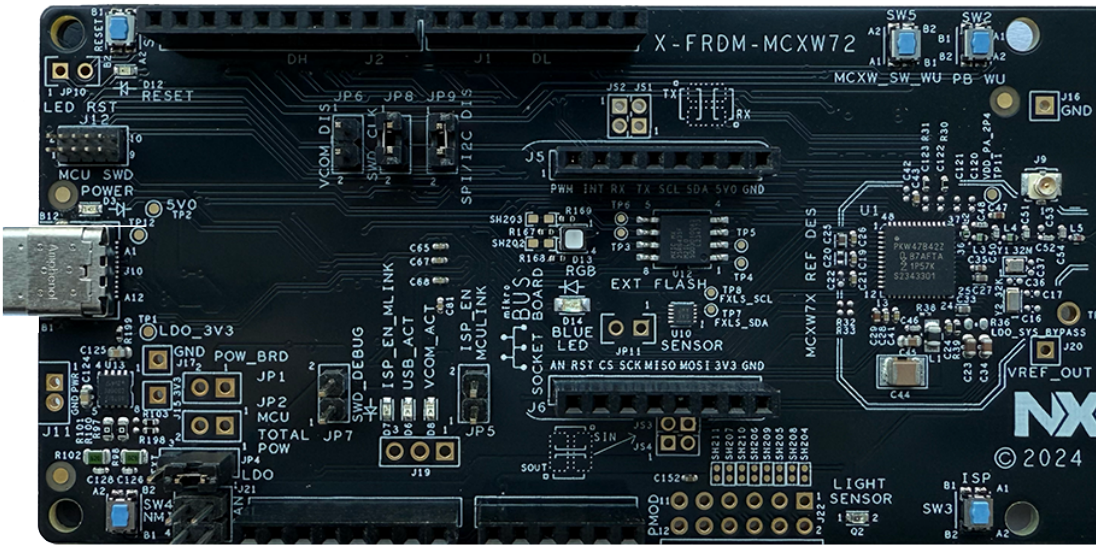
KW47-LOC board



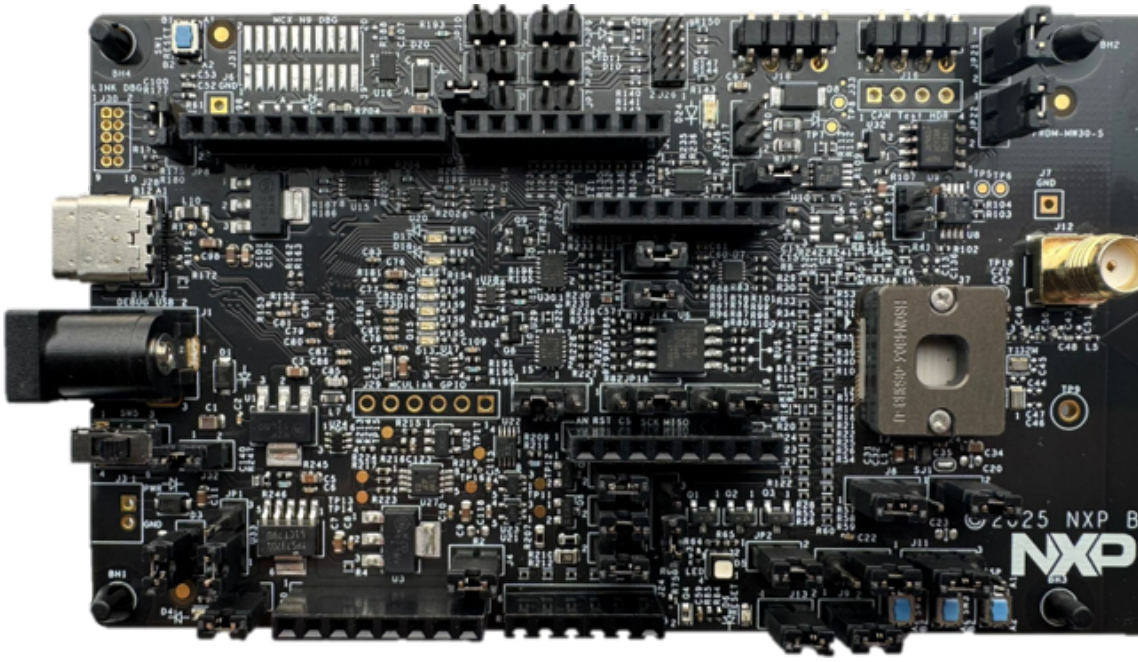
KW43-LOC board



MCX-W72-EVK board



FRDM-MCXW72 board



FRDM-KW43 board

Getting Started with west Build Environment

These steps are intended for **GitHub users** working with Bluetooth Low Energy (BLE) demo applications from the GitHub MCUXpresso SDK repository.

Option 1: Using west

Follow the steps in the order presented in Getting Started with MCUXpresso SDK Repository to install the west environment.

Option 2: Using Repository ZIP Download

1. **Download the Repository ZIP**
   - Navigate to https://mcuxpresso.nxp.com/
   - Search for your board (e.g., KW47-EVK, FRDM-MCXW71, KW45B41Z-EVK)
   - Click on your board from the search results
   - Click **ARMGCC**

- Select the components you need (ensure Bluetooth Low Energy is selected)
- Click **BUILD SDK**
- Extract the downloaded ZIP file to your workspace

2. **Locate Bluetooth Examples** Bluetooth examples
3. **Install west (Required for Building)**

   Follow Getting Started with MCUXpresso SDK Repository to install west.
4. **Build from Repository ZIP**

   Navigate to the SDK root directory and use west build commands as shown below.

Note: The Repository ZIP includes all dependencies pre-configured for your board.

Command Line to Generate/Build Projects

Example: w_uart **on** kw47evk

To build the w_uart example for the kw47evk board using the IAR toolchain and generate an IDE project, use the following command:

```
west build -b kw47evk boards/kw47evk/wireless_examples/bluetooth/w_uart/freertos/ --toolchain=iar -t␣
↪guiproject -Dcore_id=cm33_core0 --pristine
```

Note: If using Repository ZIP download, navigate to the SDK root directory before running west commands.

Argument	Description
-b kw47evk	Specifies the board
--toolchain=iar	Specifies the toolchain (default is armgcc)
-t guiproject	Creates an IDE project
-Dcore_id=cm33_core0	Core ID, if available
--pristine	Cleans the build folder before building

Use the following steps in order to build and flash the BLE software demo applications on all toolchains presented in:

- Building and flashing the BLE software demo applications using IAR Embedded Workbench
- Building and flashing the BLE Software Demo applications using MCUXpresso IDE
- Building and flashing the BLE software demo applications using Visual Studio Code

Installing the Connectivity Package

To install the Connectivity Package, configure and download the package archive from the staging system on the https://mcuxpresso.nxp.com website. You can simply download the precreated package archive if it is available on the same website.

Note: Use the default location for the package (C:\NXP) and create a subfolder there specific to each device and release.

Note: Prior to loading any wireless SDK example, update your NBU image with the provided binaries in the following folder of the SDK: ../middleware/wireless/ble_controller/bin

Building the binaries

This section describes the necessary steps for obtaining the binary files for usage with the boards.

Prerequisites

To build any of the demo applications, you need the following toolchain:

- IAR Embedded Workbench for Arm (details in release note)
- Visual Studio Code with "*MCUXpresso for Visual Studio Code*" extension (details in release note)
- Teraterm (version 4.105 or higher)

The Connectivity Software Package does not include support for any other toolchains. The packages must be built with the debug configuration to enable debugging information. This package includes various sample applications that can be used as a starting point.

Conventions for building the *wireless_UART* application.

The following sections present the steps required for building the *wireless_UART* application. All applications can be found using the following placeholders for text:

- <connectivity_path>: represents the root path for the SDK.
- <board>: represents the target board for the demo app, "kw47evk" in this case.
- <RTOS>: represents the scheduler or RTOS used by the app; it can be either "bm" or "freertos".
- <demo_app>: represents the demo application name.
- <IDE>: represents the integrated development environment used to build projects; "iar" in this case.
- <core_id>: represents the target CPU on which the application will run, "cm33_core0" in this case (applicable only on KW47-EVK, KW47-LOC, KW43-LOC, MCX-W72-EVK, FRDM-MCXW72 and FRDM-KW43 boards).
- The general folder structure of the demo applications is the following:

  <connectivity_path>\boars\<board>\wireless_examples\bluetooth\<demo_app>\<core_id>\<RTOS>\<IDE>

Selected application: w_uart

Board: One of the following boards:

- **kw45b41zevk** (for this guide)
- kw45b41zloc
- frdmmcxw71
- kw47evk
- kw47loc
- kw43loc
- mcxw72evk
- frdmmcxw72
- frdm-kw43

RTOS: FreeRTOS

Resulting location:

<connectivity_path>\boards\<kw45b41zevk / kw45b41zloc / frdmmcxw71 / kw47evk / kw47loc / kw43loc / mcxw72evk / frdmmcxw72/frdm-kw43>\wireless_examples\bluetooth\w_uart\freertos\<IDE>

Building and flashing BLE demo applications using IAR Embedded Workbench

This guide shows how to build and flash Bluetooth Low Energy demo applications using IAR Embedded Workbench and the west build environment.

Prerequisites

- **west build environment** - Follow Getting Started with MCUXpresso SDK Repository
- **IAR Embedded Workbench for Arm** (see release notes for version)
- **GitHub MCUXpresso SDK repository** cloned locally (west init -m https://github.com/nxp-mcuxpresso/mcuxsdk-manifests.git)

Step 1: Generate IAR Project

Use west build to generate the IAR project:

```
west build -b <board> <application_path> --toolchain=iar -t guiproject -Dcore_id=<core_id> --pristine
```

Example for Wireless UART on KW47-EVK:

```
west build -b kw47evk examples/wireless_examples/bluetooth/w_uart/freertos/ --toolchain=iar -t guiproject -
↪Dcore_id=cm33_core0 --pristine
```

Parameters:

Parameter	Description
-b <board>	Target board (kw47evk, kw47loc, kw43loc, frdmkw43, frdmmcxw72, etc.)
--toolchain=iar	Use IAR toolchain
-t guiproject	Generate IAR workspace
-Dcore_id=cm33_core0	CPU core (for KW47, MCXW72)
--pristine	Clean build

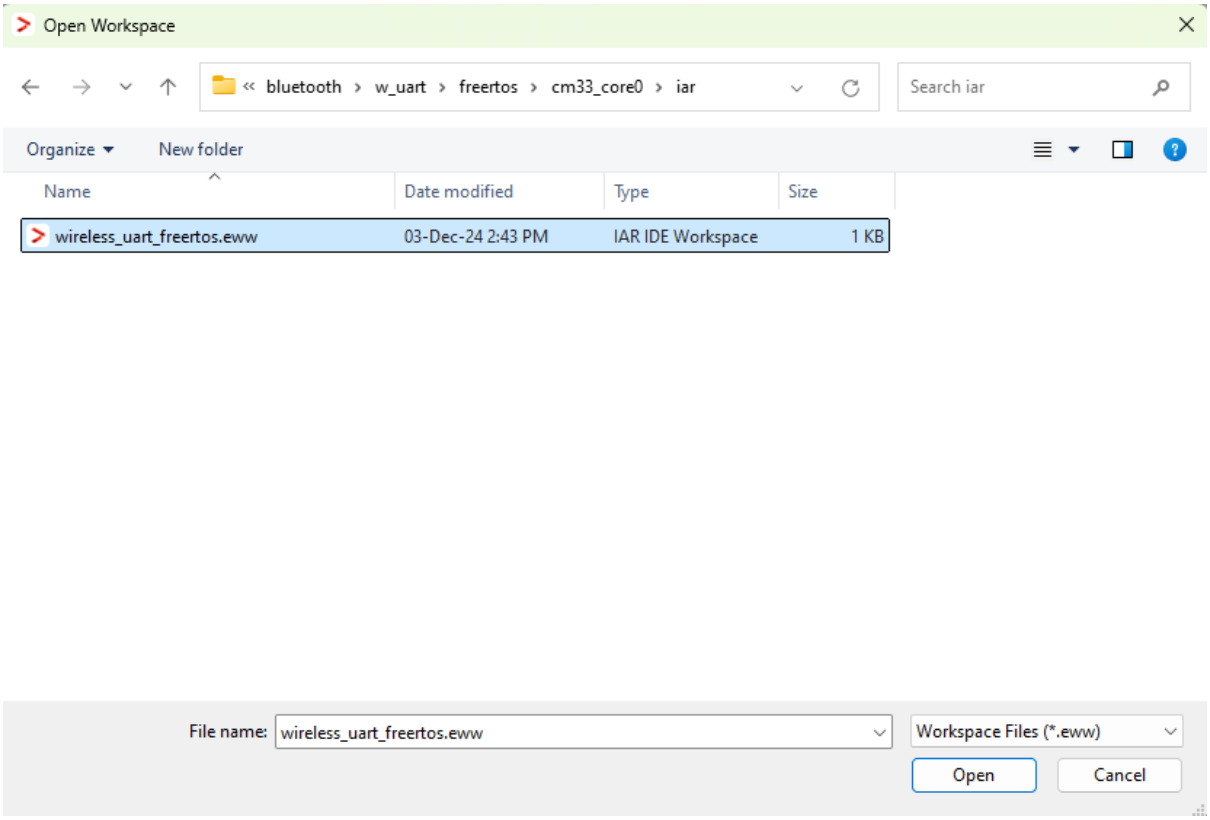
Supported boards: kw47evk, kw47loc, kw43loc, frdmmcxw71, frdmmcxw72, mcxw72evk, frdmkw43.

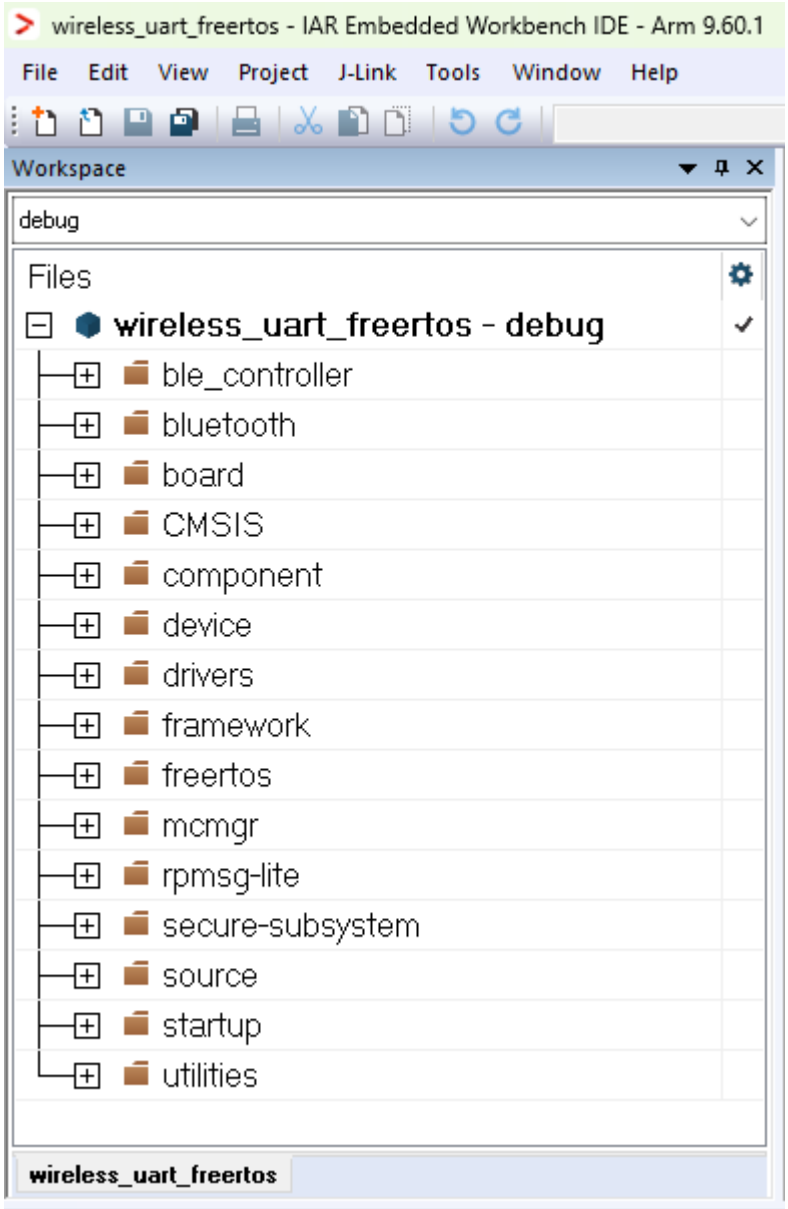
Output: IAR workspace at build/iar/<project_name>.eww

Note: The west build command builds in debug mode by default. To build in release mode, add --config=release to the command.

Step 2: Open IAR Workspace

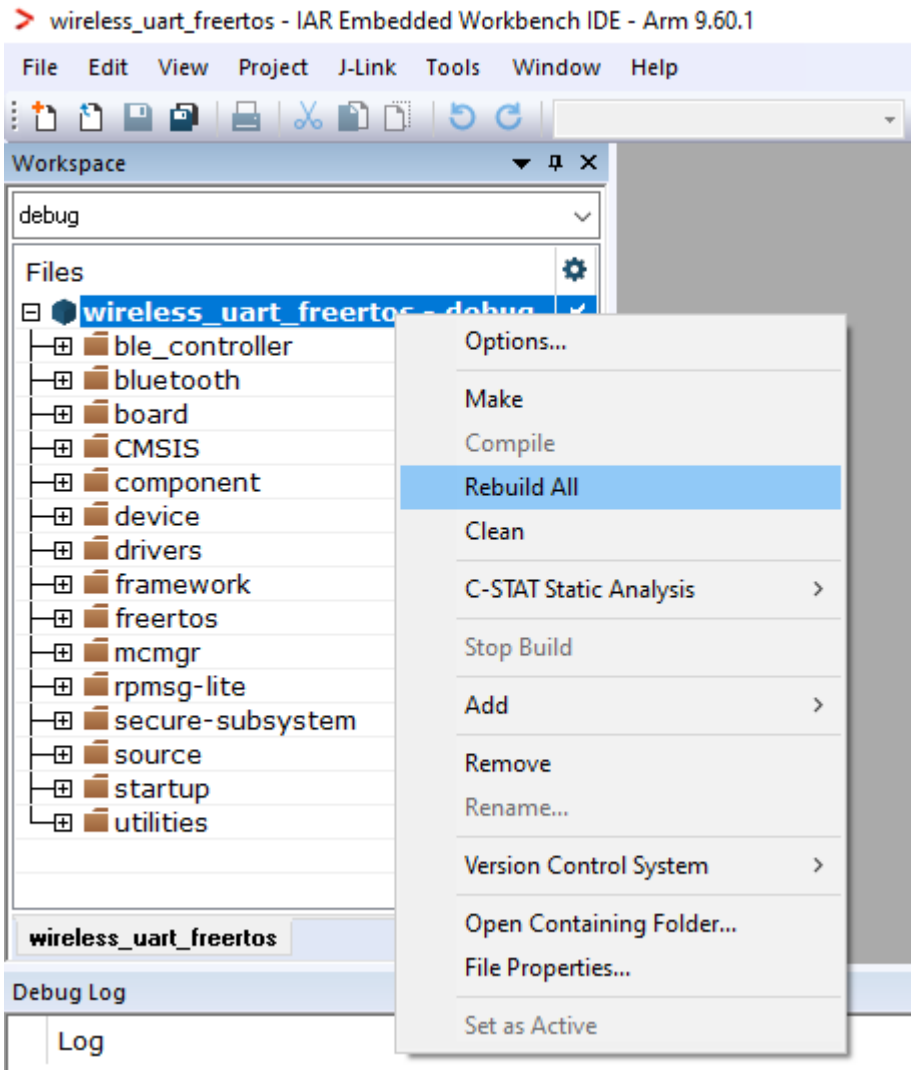
1. Navigate to build/iar/
2. Open the .eww file in IAR Embedded Workbench


Open Workspace
« bluetooth › w_uart › freertos › cm33_core0 › iar
Search iar
Organize
New folder
Name
Date modified
Type
Size
wireless_uart_freertos.eww
03-Dec-24 2:43 PM
IAR IDE Workspace
1 KB
File name:
wireless_uart_freertos.eww
Workspace Files (*.eww)
Open
Cancel



Step 3: Build Project

Build the project using **Project** > **Make** or **F7**.

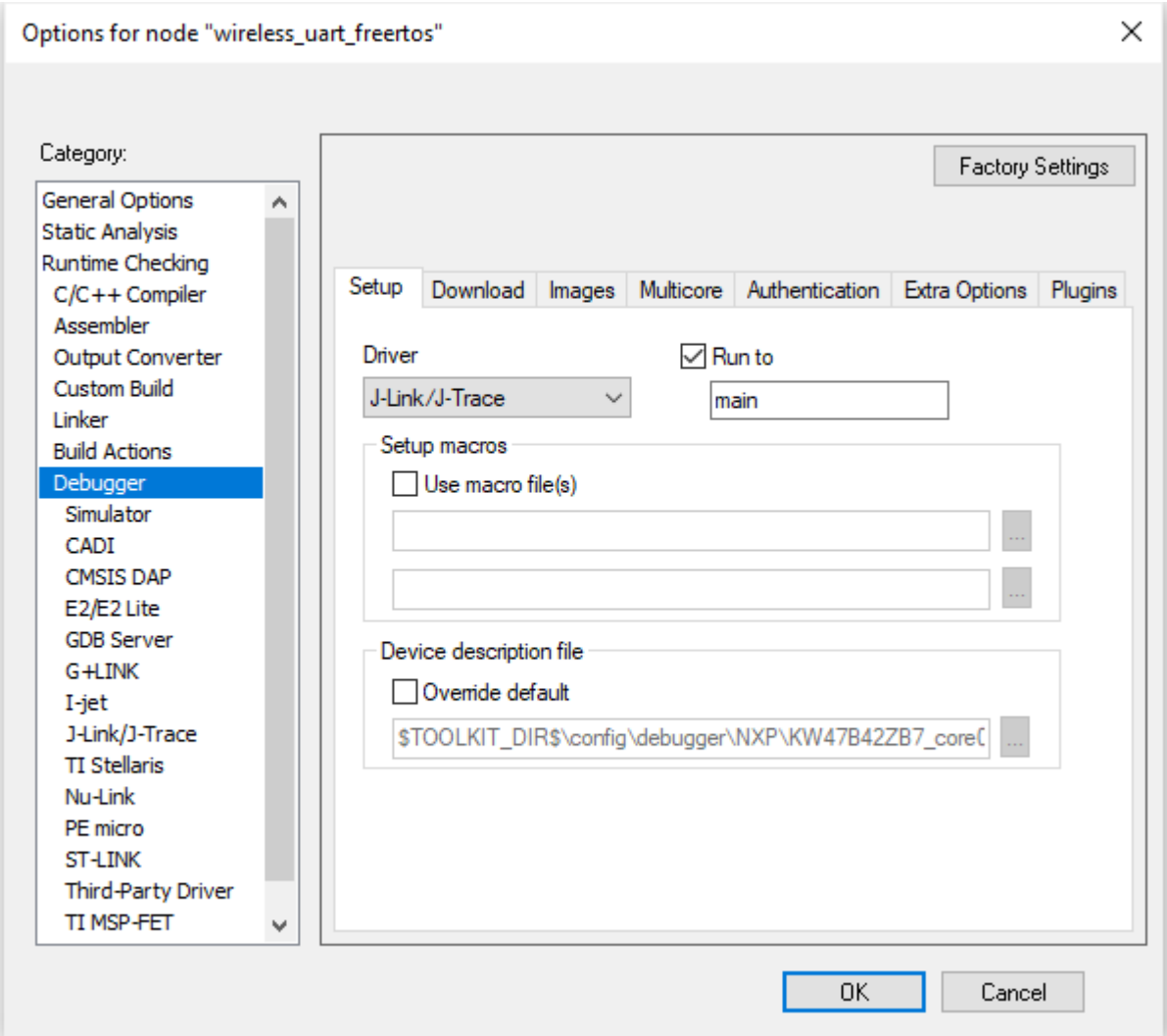


Step 4: Configure Debugger

Go to **Project** > **Options (Alt+F7)** > **Debugger** > **Setup** > **Driver**

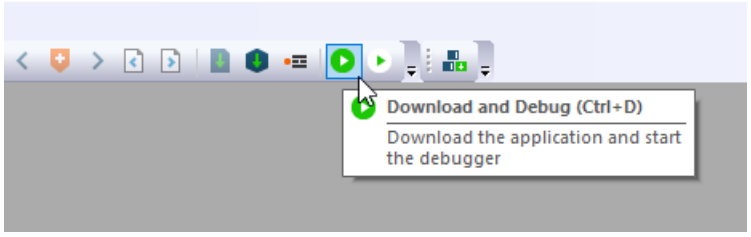
Select debugger:

- **CMSIS-DAP** for kw45b41zevk, mcxw71evk, frdmmcxw71
- **J-Link** for kw47evk, kw47loc, kw43loc, mcxw72evk, frdmmcxw72, frdmkw43



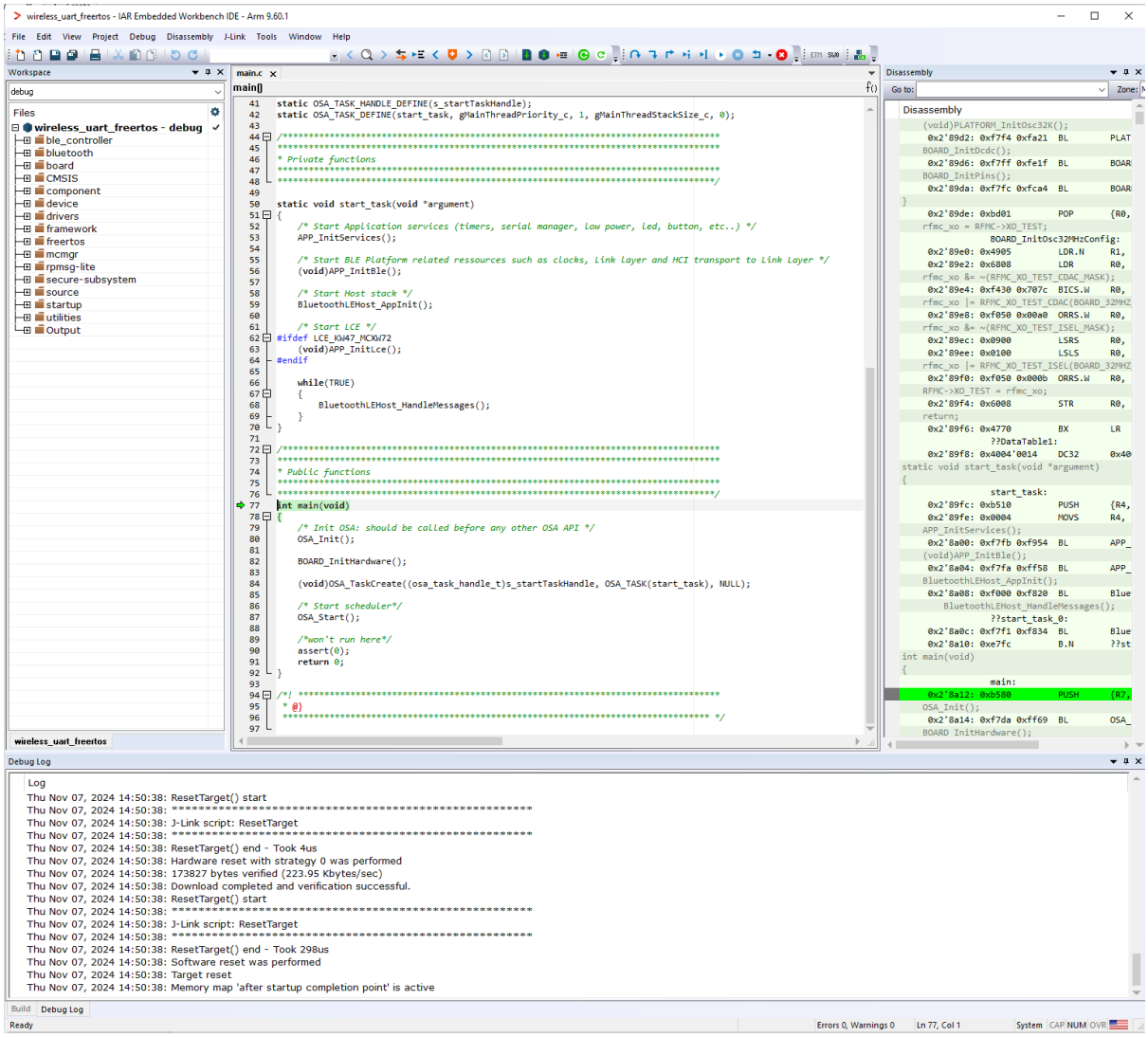
Step 5: Flash and Debug

Click **Download and Debug** or press **Ctrl+D**.



Step 6: Run Application

Press **Go (F5)** to start the application.



Notes

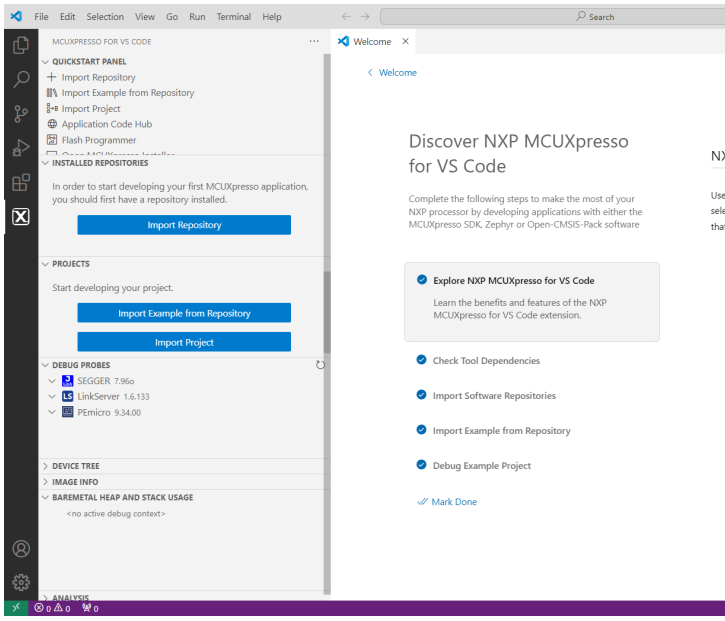
- To rebuild after configuration changes, run west build with `--pristine`
- Build artifacts are in the `build/` directory
- Change board with `-b` parameter and application path as needed

Parent topic: *Building the binaries*

Building and flashing the BLE software demo applications using Visual Studio Code

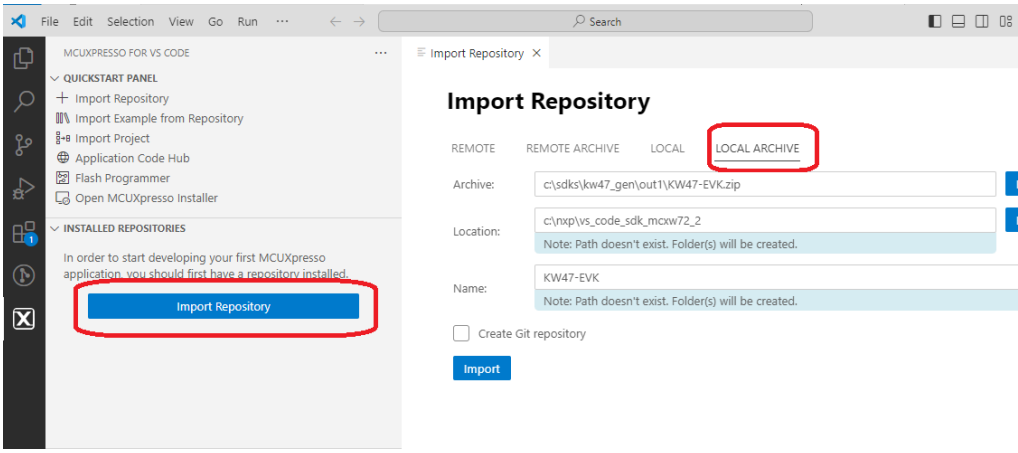
To build and flash the BLE software demo applications using Visual Studio Code, follow the steps listed below:

1. Open Visual Studio Code and open the MCUXpresso for Visual Studio Code extension as shown in the figure below.



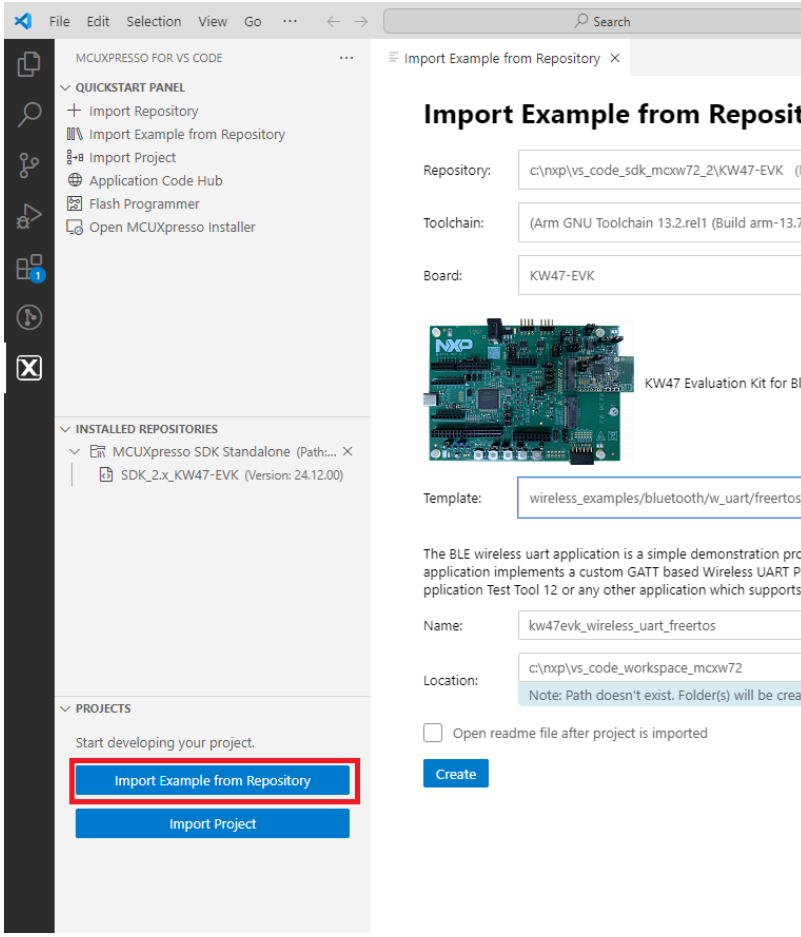
MCUXpresso for Visual Studio Code extension

2. Import the SDK package: click "**Import Repository**". Then, choose the "**Local**" option (if the SDK is archived use "**Local archive**"), browse to the path of the SDK you want, and click "**Import**".



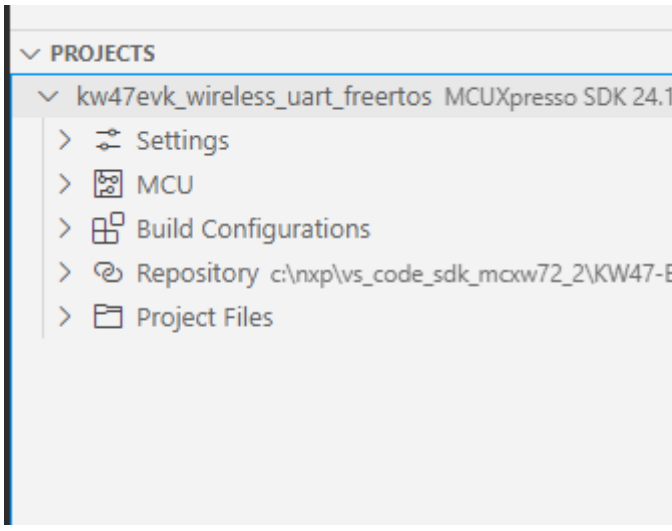
Steps to import the SDK

3. After the SDK is loaded successfully, select the "**Import Example from Repository**" to add an application to your workspace. Choose the repository, toolchain (Arm GNU), board, example you want to add, and the location where the VS Code project would be created. Then click "**Create**".



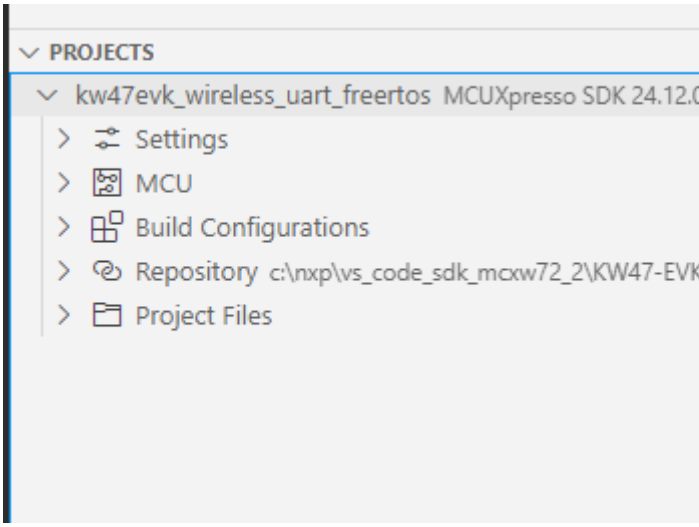
Steps to import the example application

4. The application now appears in the "**Projects**" tab on the left. Build the application by pressing the "**Build selected**" button.



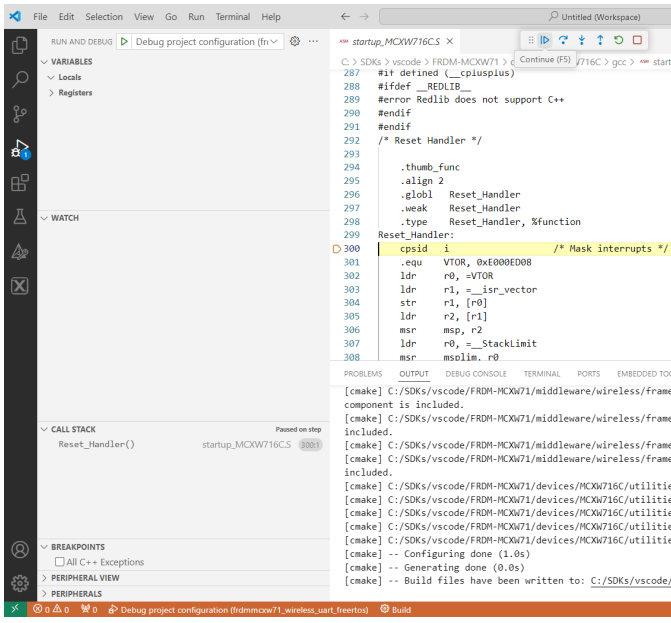
Building the Wireless UART FreeRTOS application

5. After the build is completed successfully, debug the application by clicking the "**Debug**" button.



Debug the Wireless UART FreeRTOS application

6. Press the run (“**Continue**”) button twice to run the application on the board.



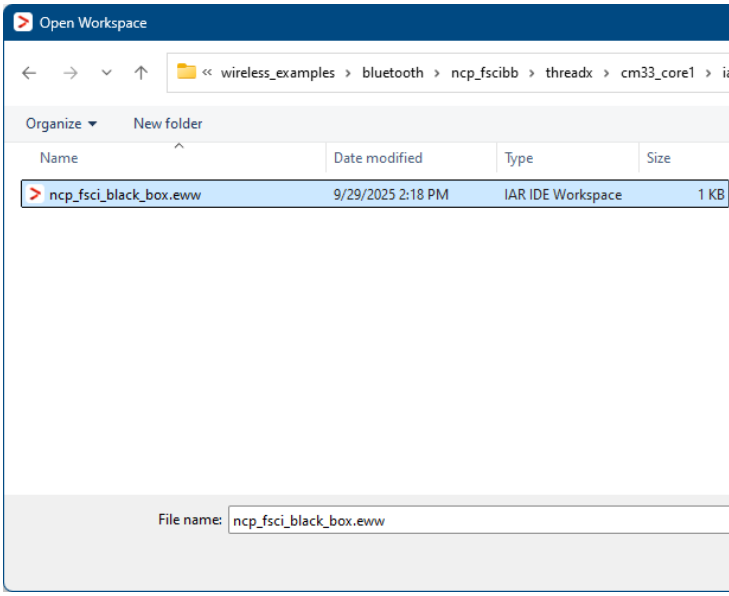
Running the Wireless UART FreeRTOS application

Parent topic:*Building the binaries*

Building and flashing the BLE Extended NBU software demo applications using IAR Embedded Workbench

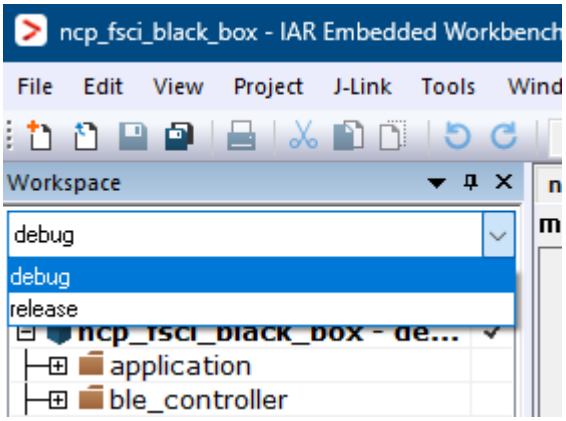
Use the following steps in order to build and flash the BLE software demo applications using the IAR Embedded Workbench:

1. First unpack the contents of the archive to a folder on the local disk. Then, navigate to the resulting location starting from the SDK root directory.
2. Open the IAR workspace file (*.eww file format) highlighted file in the figure below.



NCP FSCI Blackbox IAR demo project location

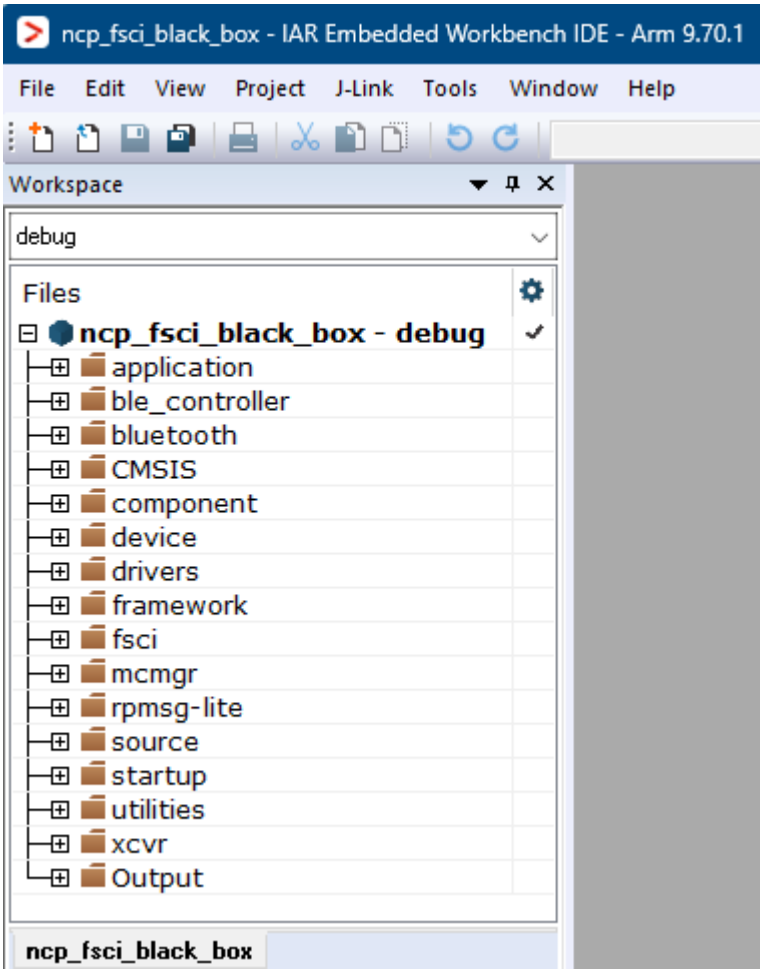
3. Choose between Debug and Release configurations in the drop-down selector above the project tree in the workspace.



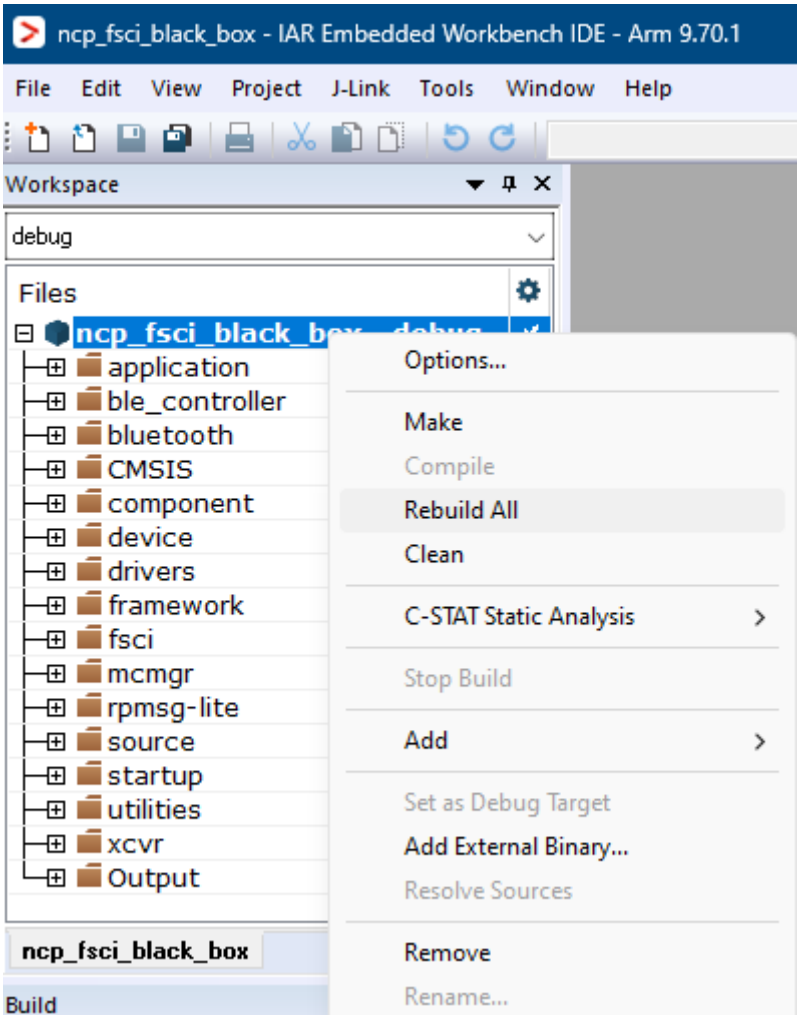
Select the desired configuration (Debug or Release)

The figure below shows the NCP FSCI Blackbox - IAR workspace.

NCP FSCI Blackbox - IAR workspace



4. Build the NCP FSCI Blackbox project using the options shown in the figure.

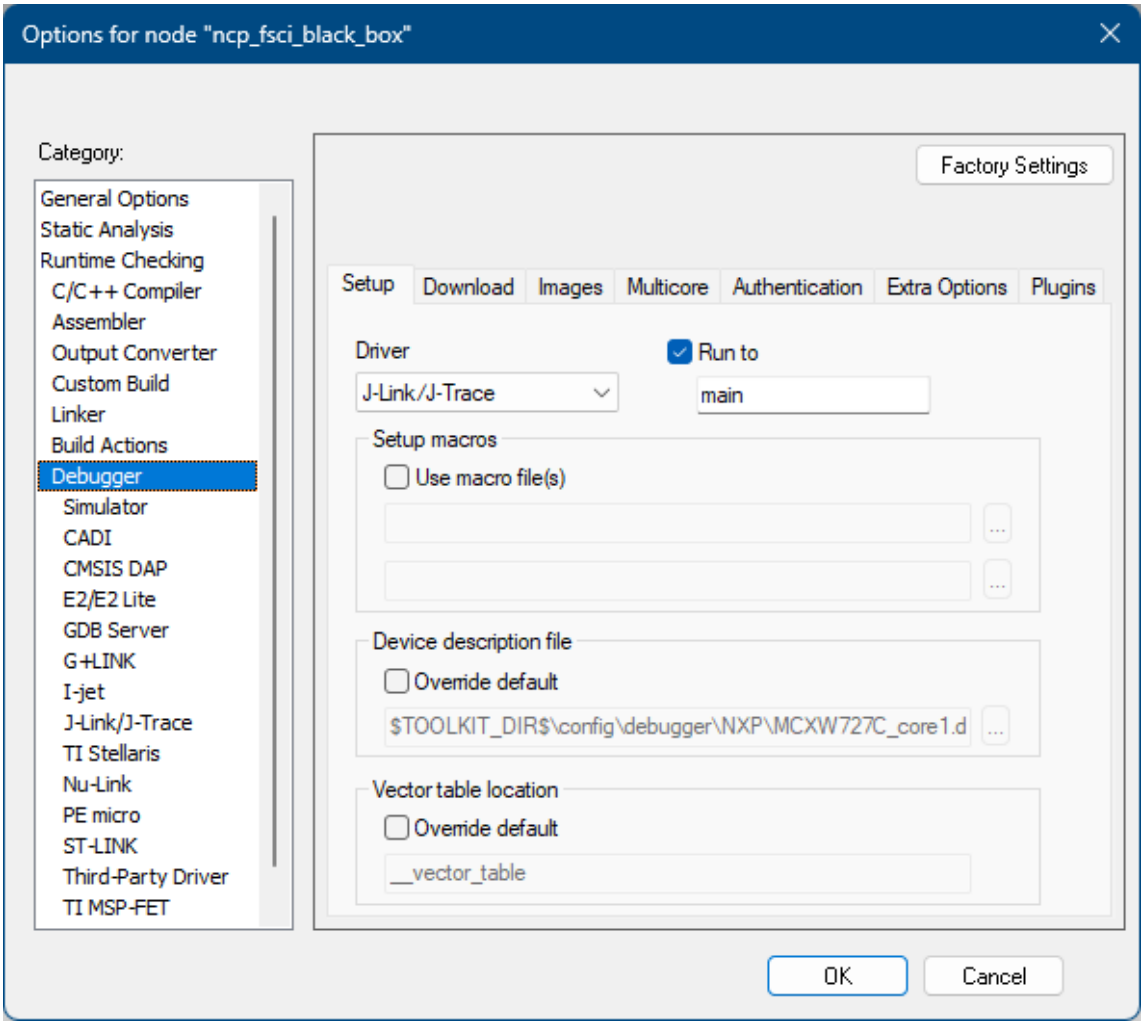


Build NCP FSCI Blackbox application

5. Make the appropriate debugger settings in the project options window, as seen in the next figure.

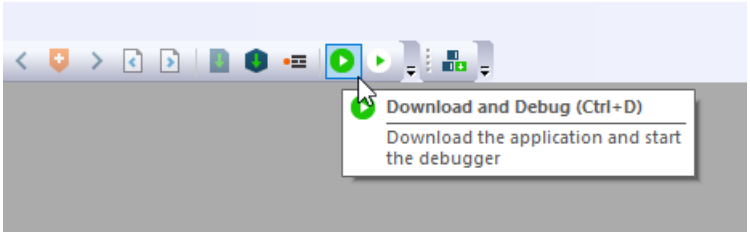
   Go to: **Project** > **Options (Alt+F7)** > **Debugger** > **Setup (tab)** > **Driver** > **J-Link/J-Trace**

   Debugger Settings for the NCP FSCI Blackbox project

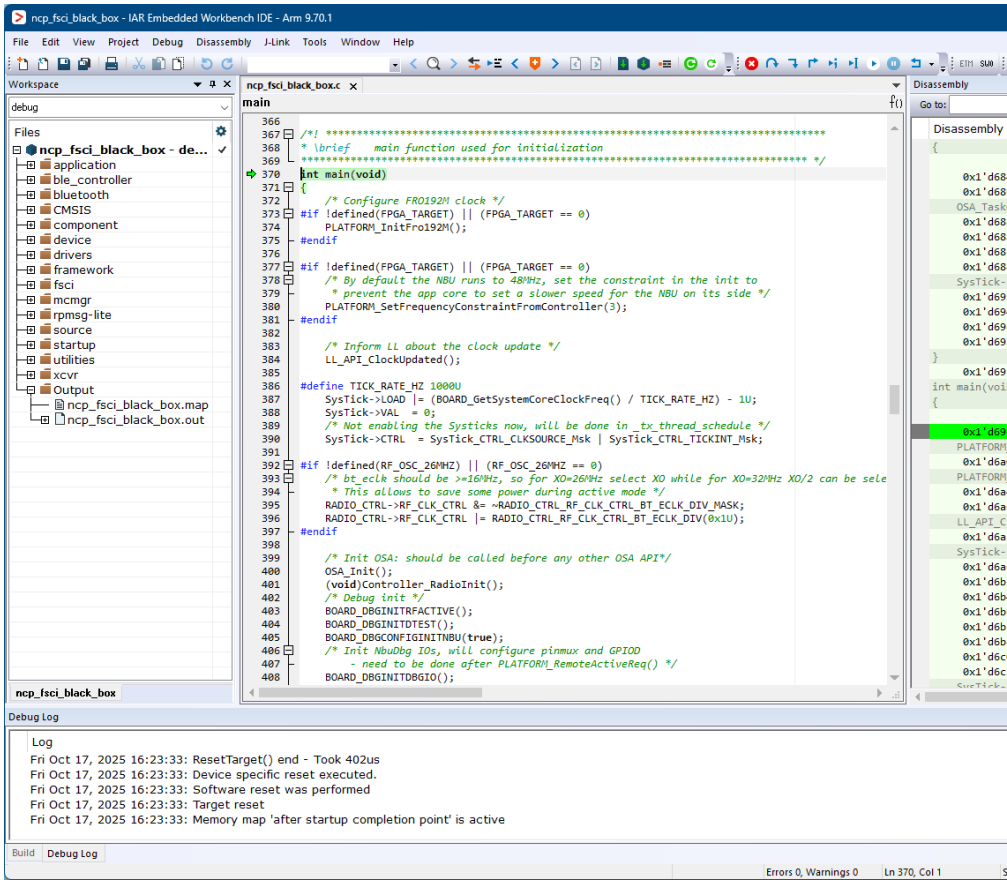


6. Click the "**Download and Debug**" button (or **CTRL+D**) to flash the executable onto the board.

Download and Debug the NCP FSCI Blackbox application

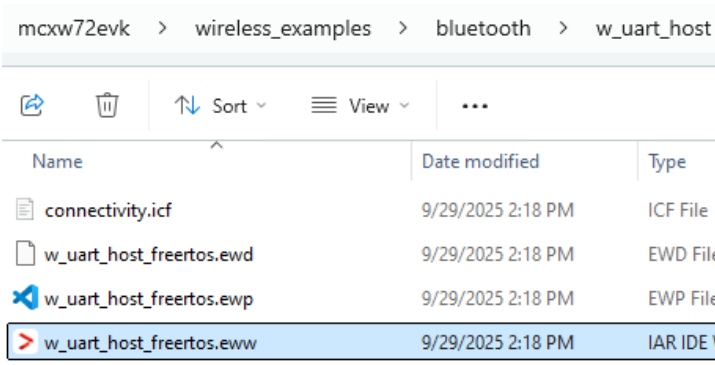


7. Press **Go (F5)**. At this moment, the board starts running the application.



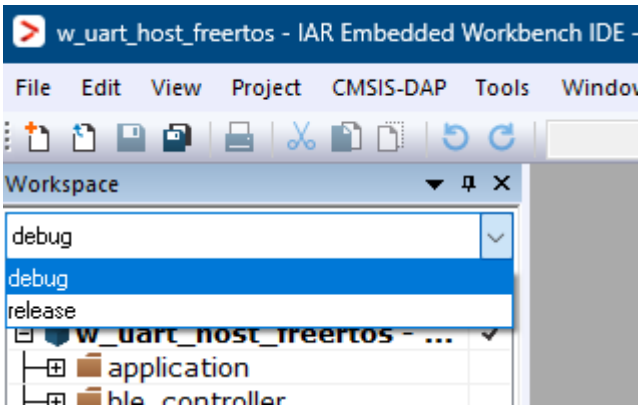
Running the code on IAR

8. Open the Core0 IAR workspace file (*.eww file format) highlighted file in the figure below.



Wireless UART Host IAR demo project location

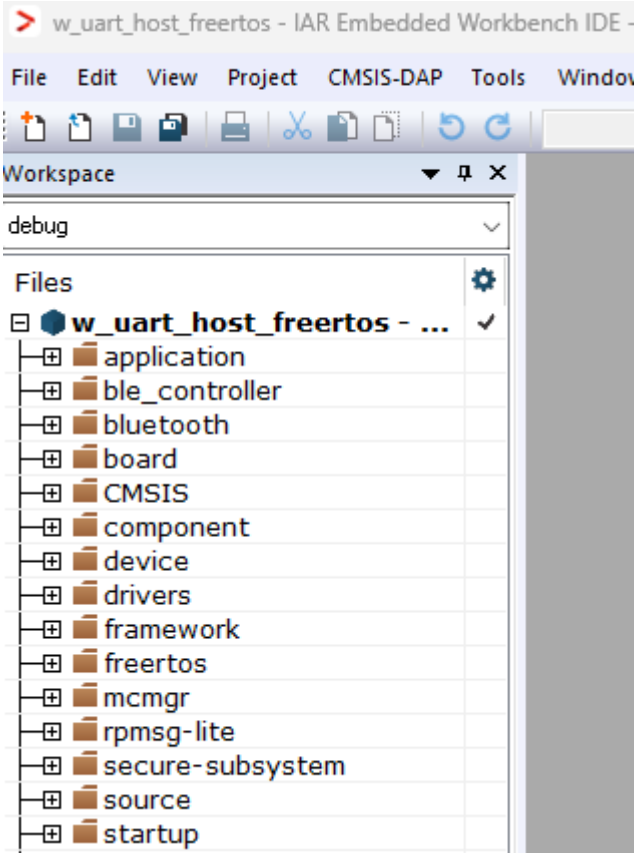
9. Choose between Debug and Release configurations in the drop-down selector above the project tree in the workspace.



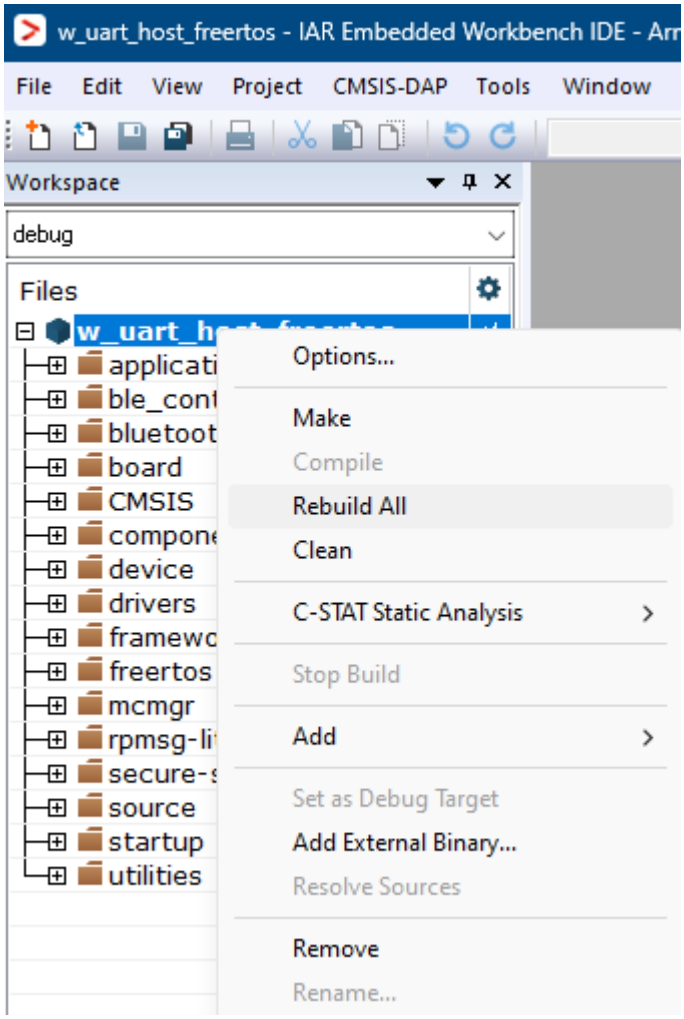
Select the desired configuration (Debug or Release)

The figure below shows the Wireless Uart Host - IAR workspace.

Wireless Uart Host - IAR workspace



10. Build the Wireless Uart Host project using the options shown in the figure.

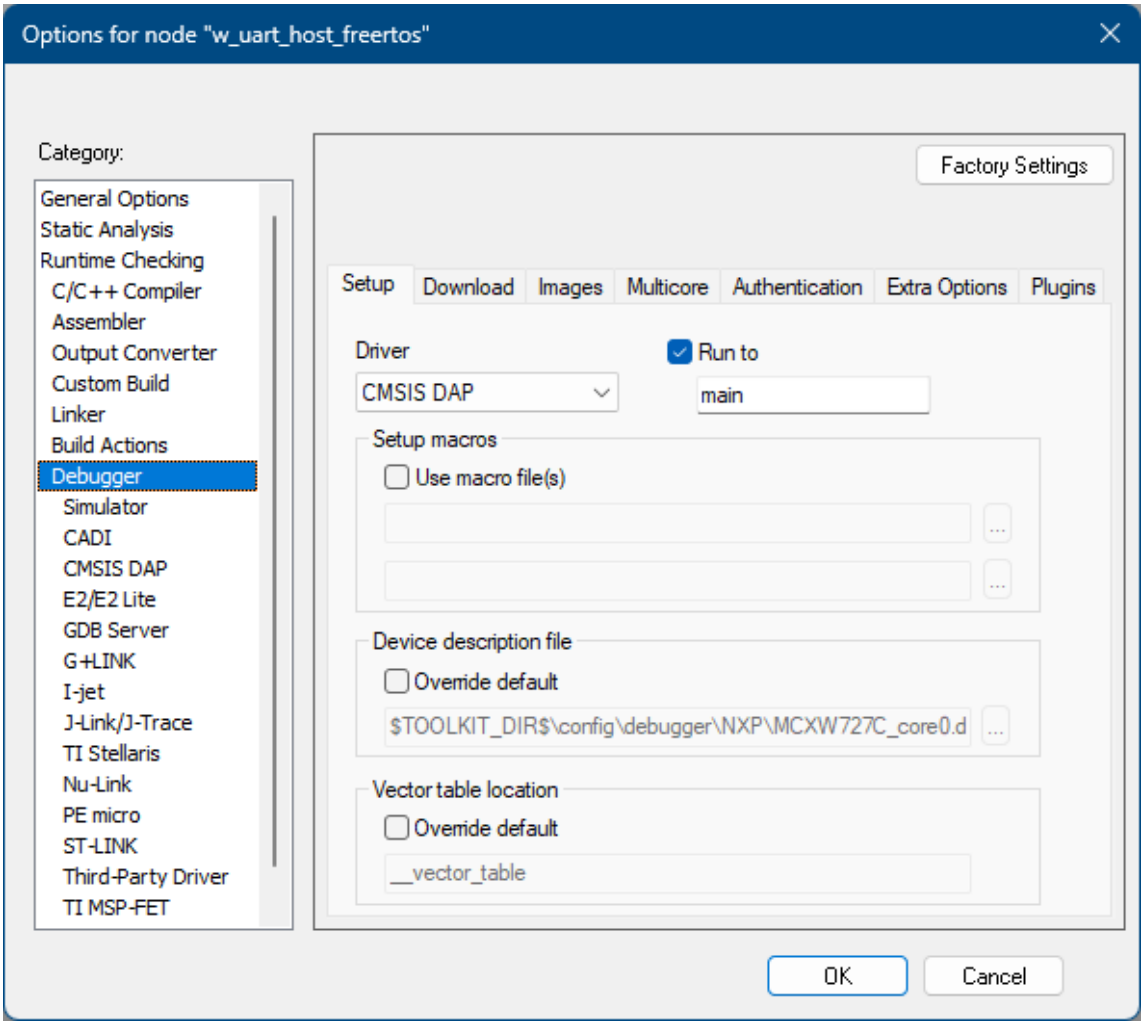


Build Wireless Uart Host application

11. Make the appropriate debugger settings in the project options window, as seen in the next figure.

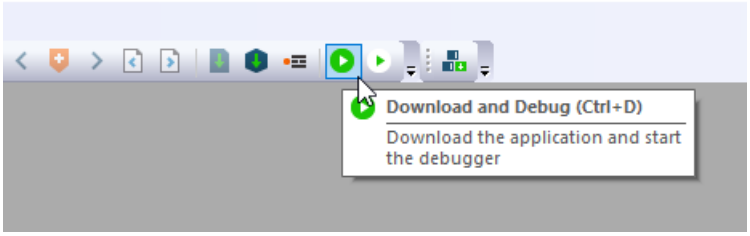
    Go to: **Project** > **Options (Alt+F7)** > **Debugger** > **Setup (tab)** > **Driver** > **J-Link/J-Trace**

    Debugger Settings for the Wireless Uart Host project

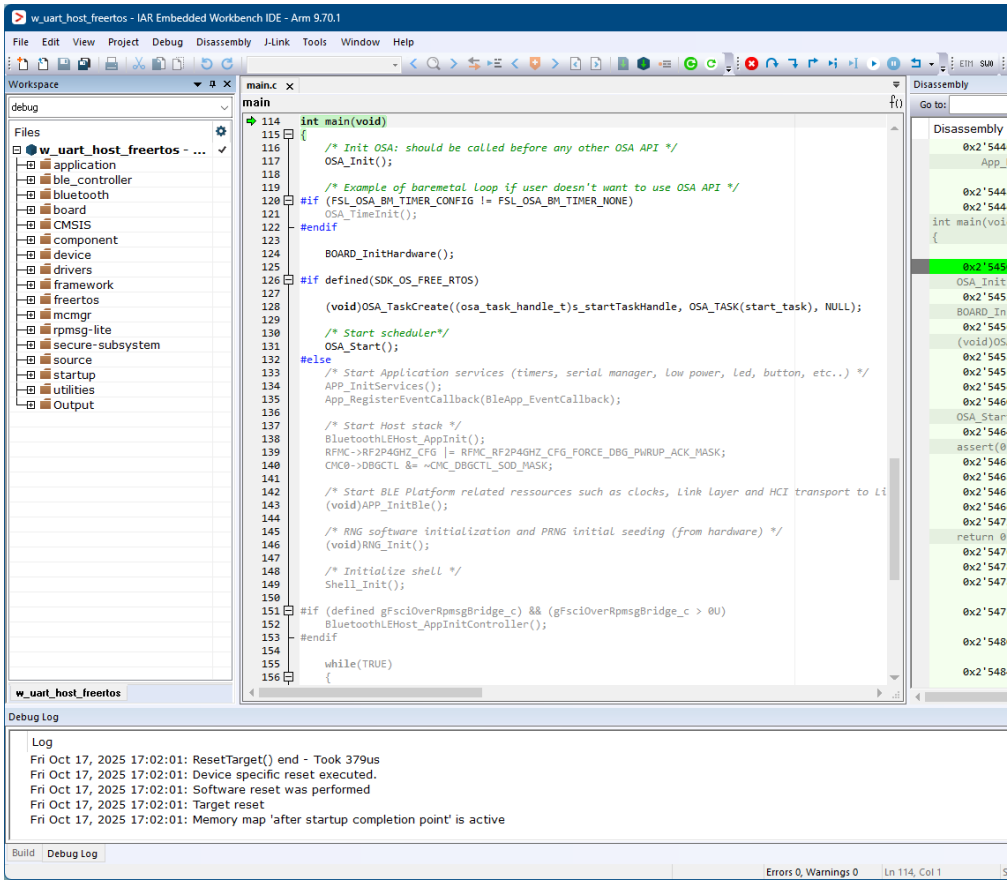


12. Click the "**Download and Debug**" button (or **CTRL+D**) to flash the executable onto the board.

Download and Debug the Wireless Uart Host application



13. Press **Go (F5)**. At this moment, the board starts running the application.



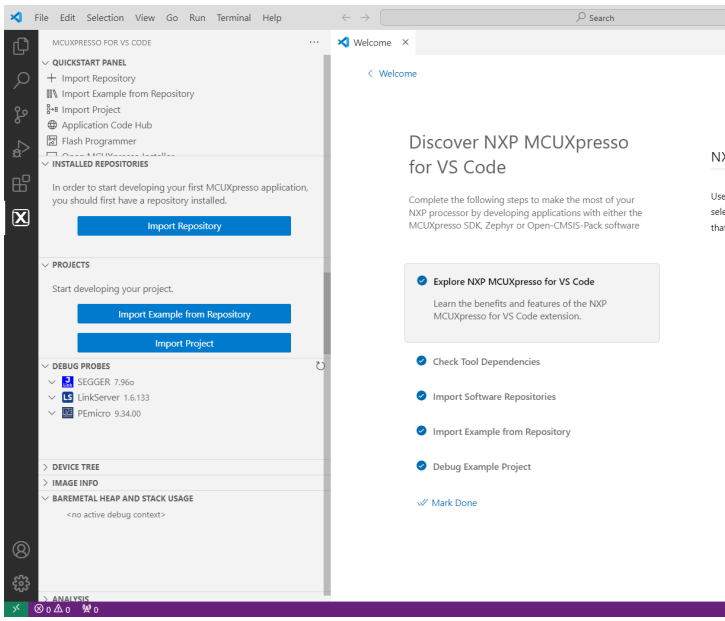
Running the code on IAR

Parent topic:*Building the binaries*

Building and flashing the BLE Extended NBU software demo applications using Visual Studio Code

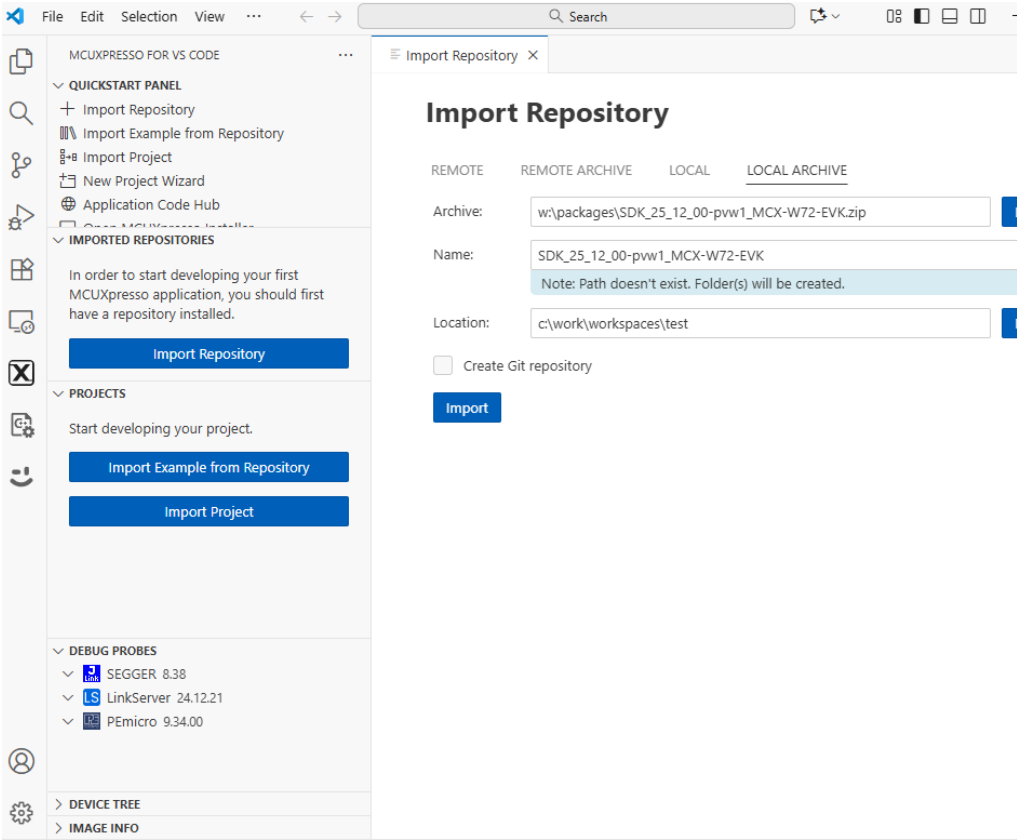
To build and flash the BLE software demo applications using Visual Studio Code, follow the steps listed below:

1. Open Visual Studio Code and open the MCUXpresso for Visual Studio Code extension as shown in the figure below.



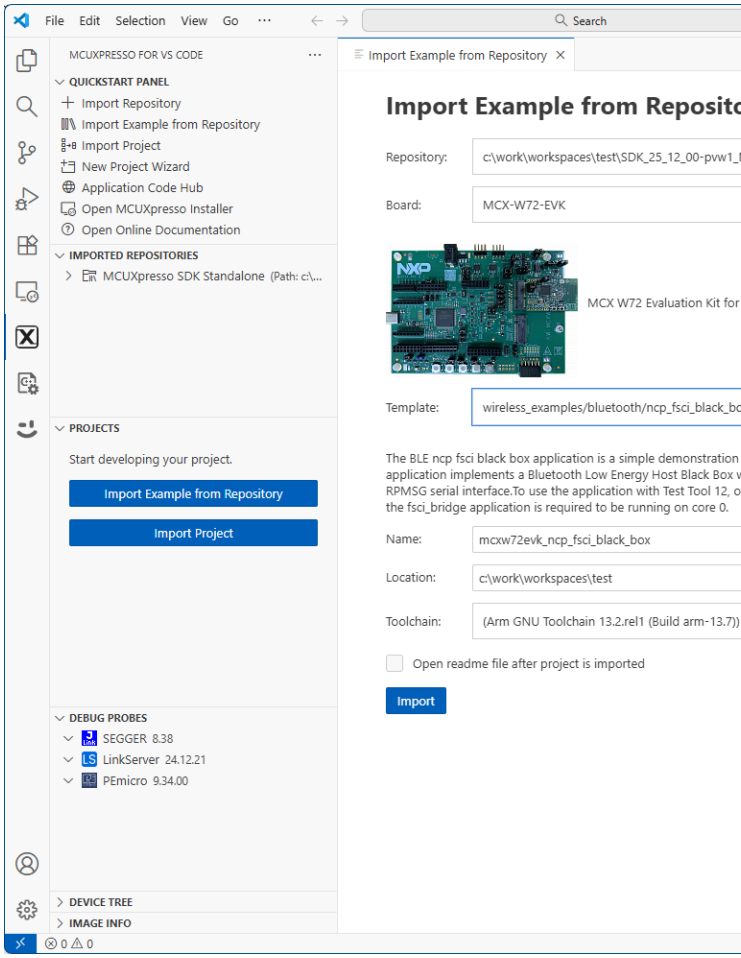
MCUXpresso for Visual Studio Code extension

2. Import the SDK package: click "**Import Repository**". Then, choose the "**Local**" option (if the SDK is archived use "**Local archive**"), browse to the path of the SDK you want, and click "**Import**".



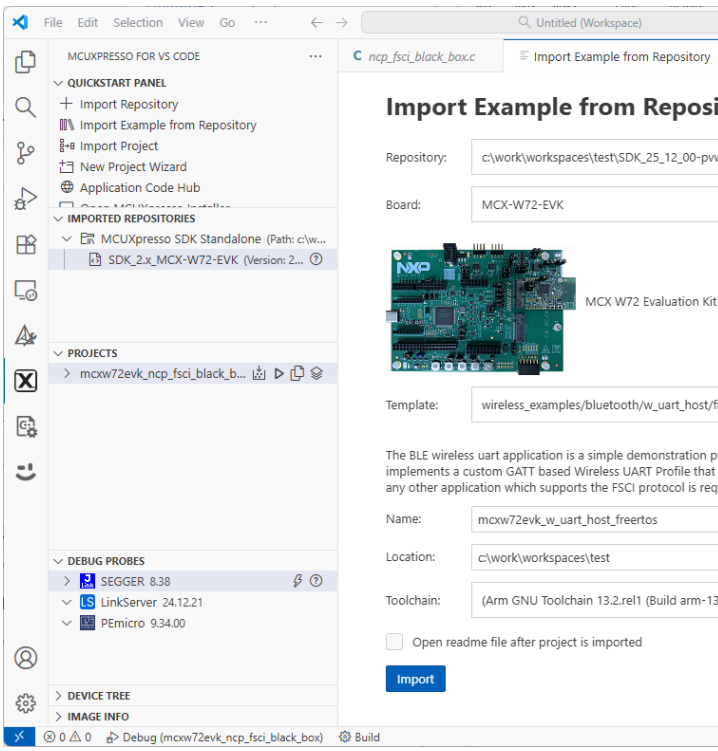
Steps to import the SDK

3. Import NCP(Core1) application: After the SDK is loaded successfully, select the "**Import Example from Repository**" to add an application to your workspace. Choose the repository, toolchain (Arm GNU), board, example you want to add, and the location where the VS Code project would be created. Then click "**Import**".



Steps to import the NCP example application

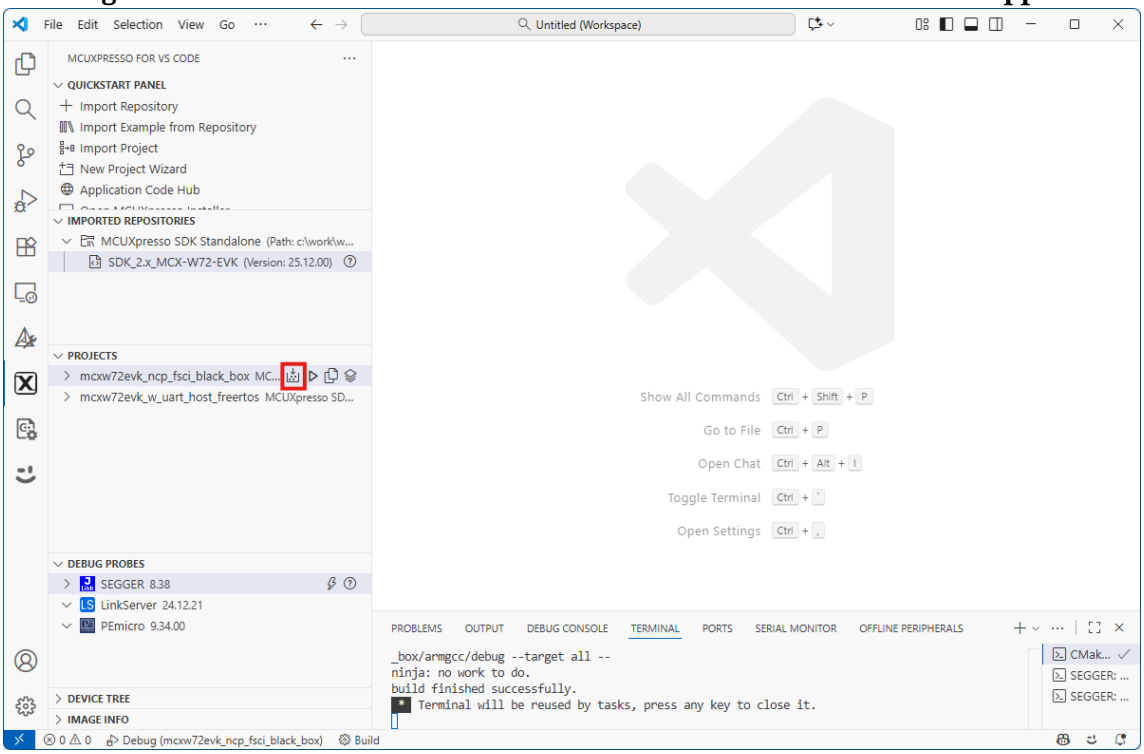
4. Import Core0 application: After the SDK is loaded successfully, select the "**Import Example from Repository**" to add an application to your workspace. Choose the repository, toolchain (Arm GNU), board, example you want to add, and the location where the VS Code project would be created. Then click "**Import**".



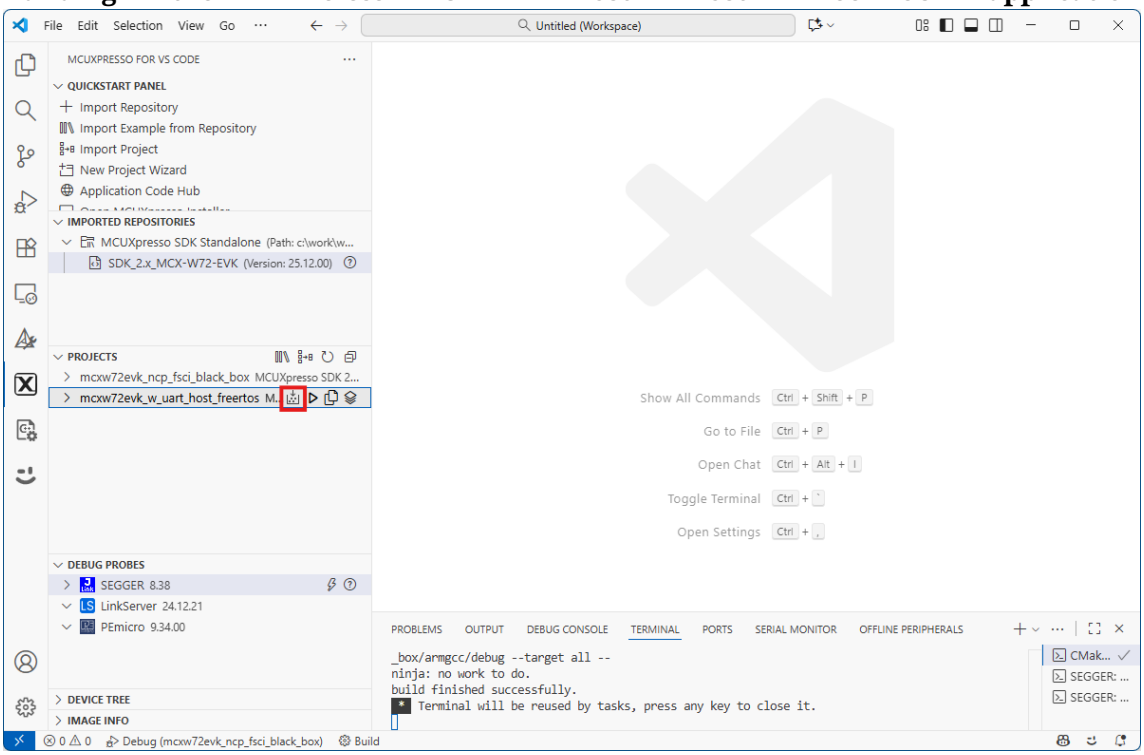
Steps to import the Core0 example application

5. Both Extended NBU applications now appears in the "**Projects**" tab on the left. Build both applications by pressing the "**Build selected**" button on each.

Building the NCP FSCI Black Box ThreadX application

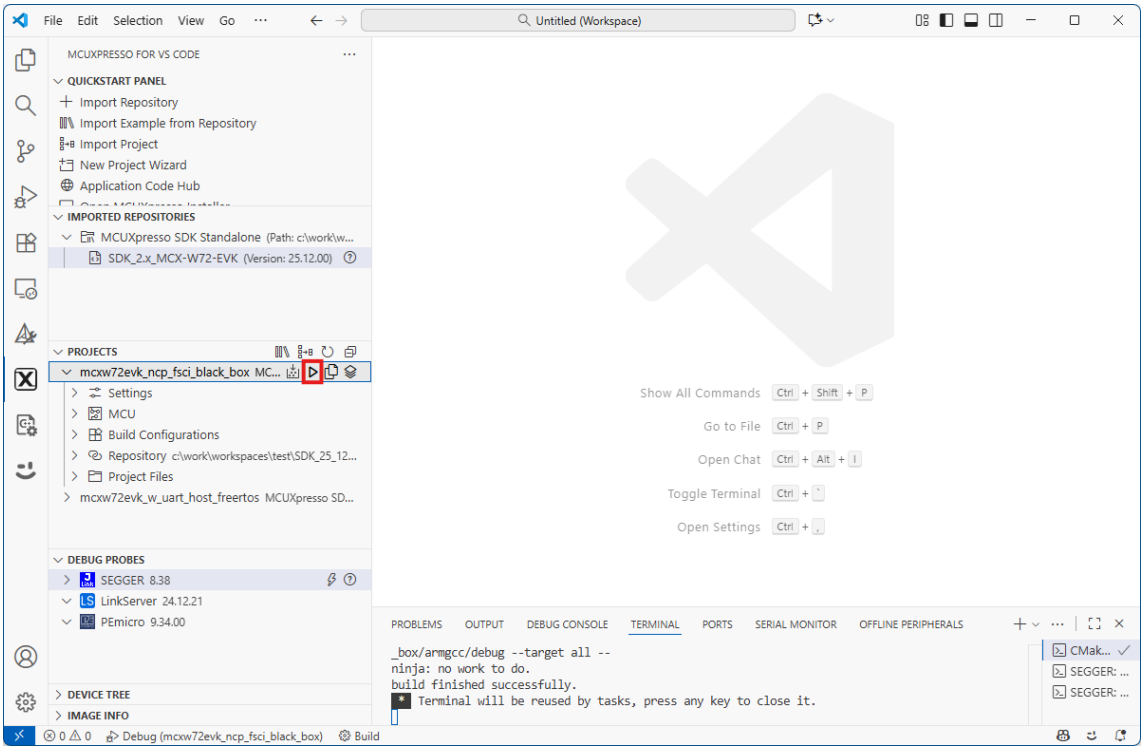


Building the Wireless UART Host Host FreeRTOS application



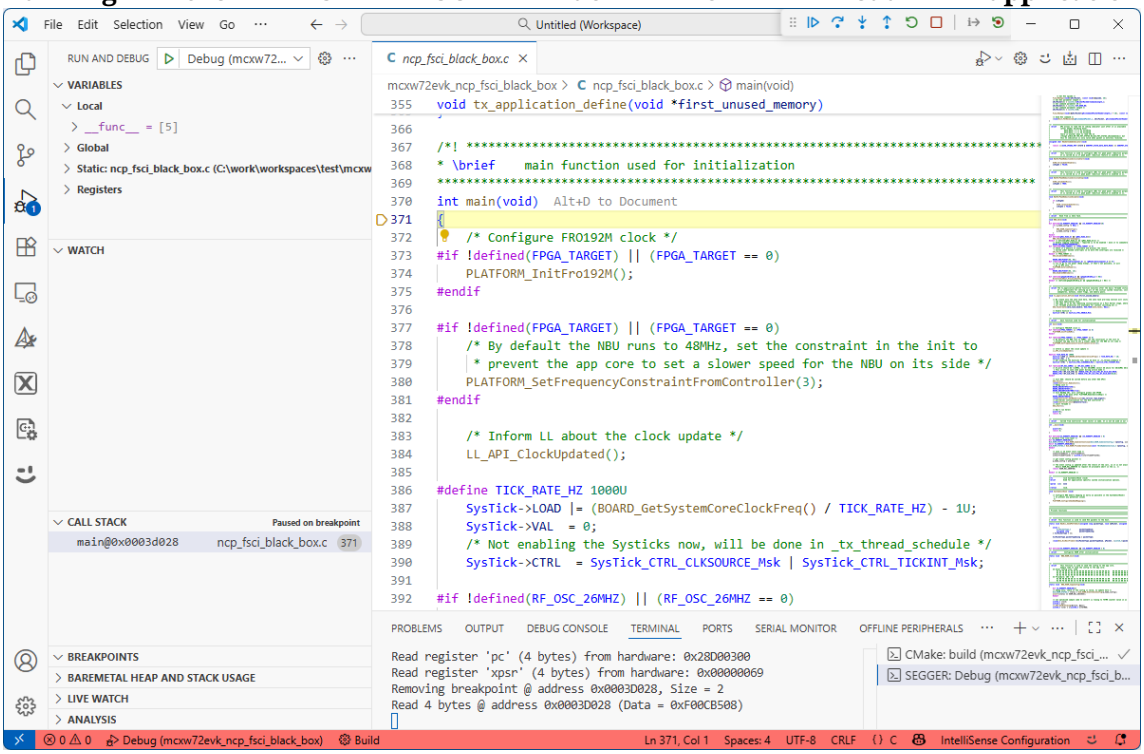
6. After the build is completed successfully, first debug the NBU(Core1) application by clicking the "**Debug**" button.

Debug the NCP FSCI Black Box ThreadX application



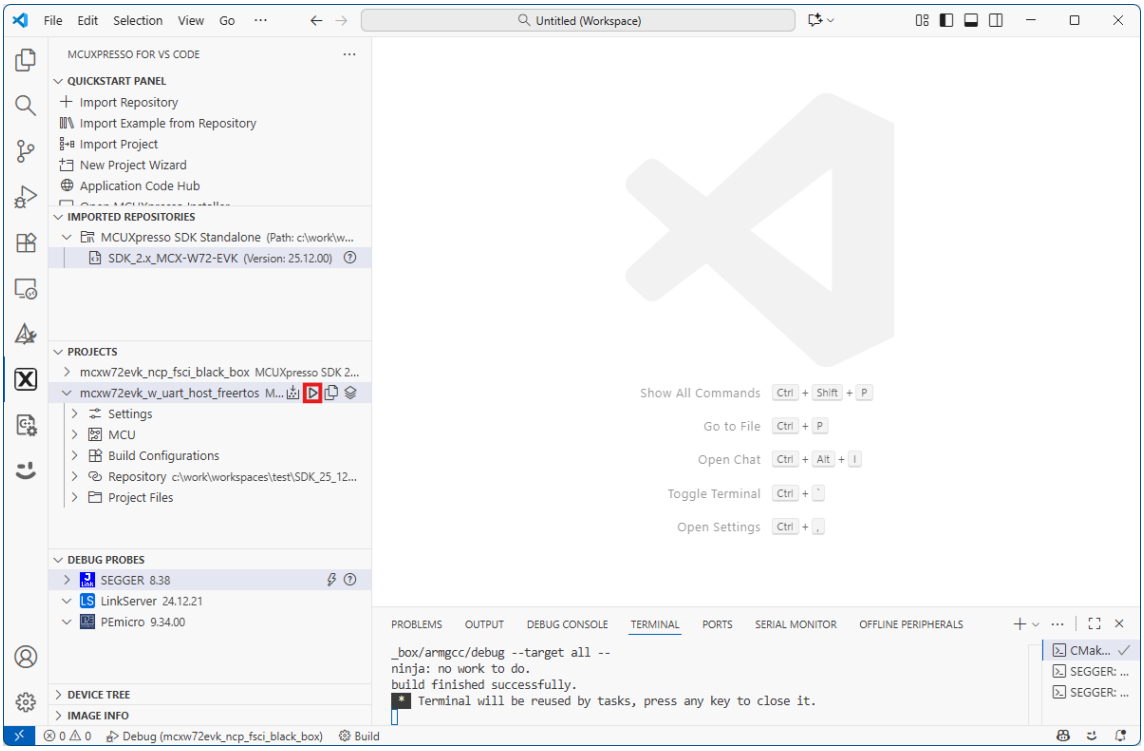
7. Press the run (“**Continue**”) button to run the application on the board.

Running the NCP FSCI Black Box ThreadX application



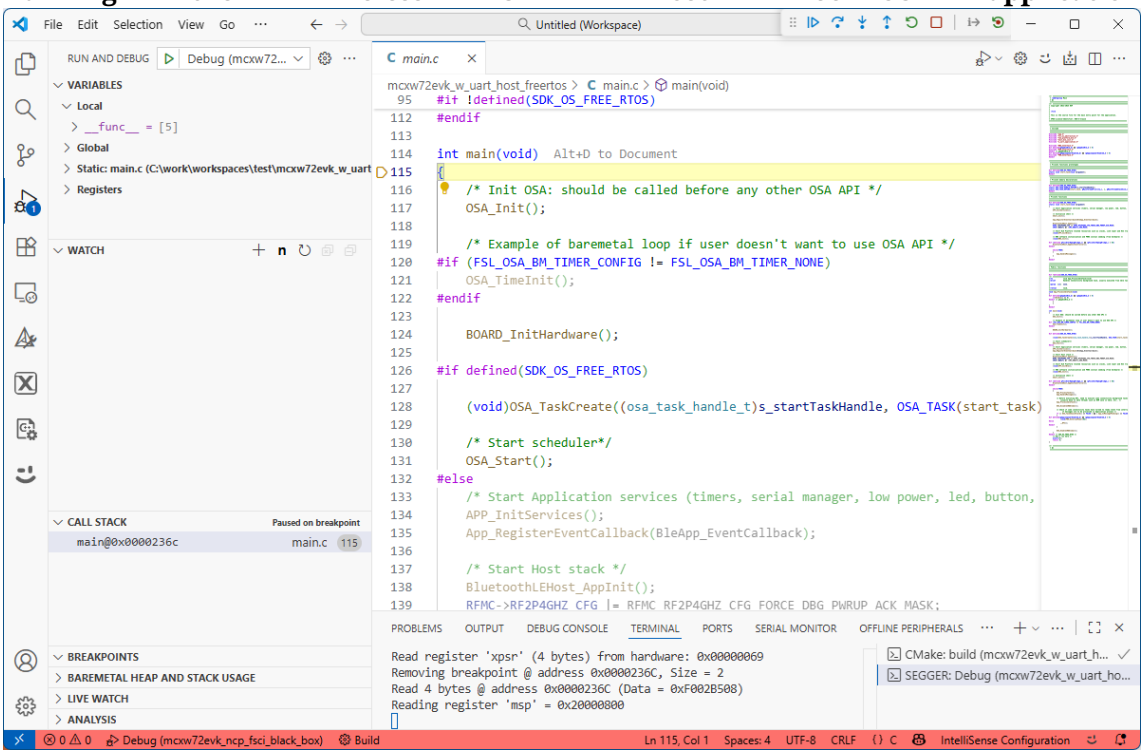
8. After the build is completed successfully and the NBU application was successfully written, debug the Core0 application by clicking the “**Debug**” button.

Debug the Wireless UART Host FreeRTOS application



9. Press the run (“**Continue**”) button to run the application on the board.

Running the Wireless UART Host FreeRTOS application



Parent topic:*Building the binaries*

Running the Wireless UART application using NXP IoT Toolbox mobile application

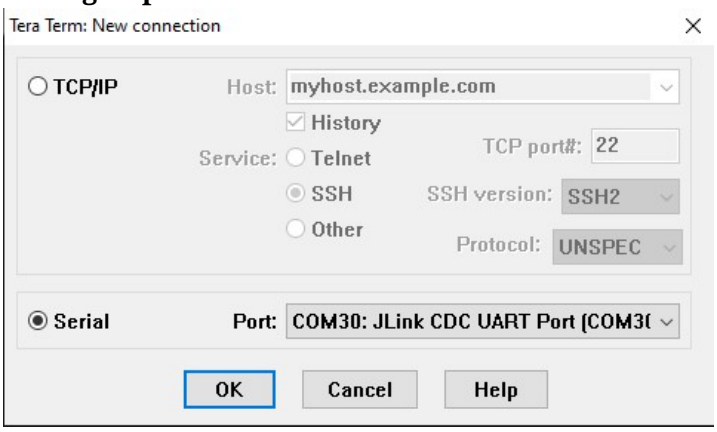
To run the Wireless UART application using NXP IoT Toolbox mobile application, follow the steps as listed below:

Note: Before working on these steps, ensure to install the latest version of the NXP IoT Toolbox

mobile application from the application store.

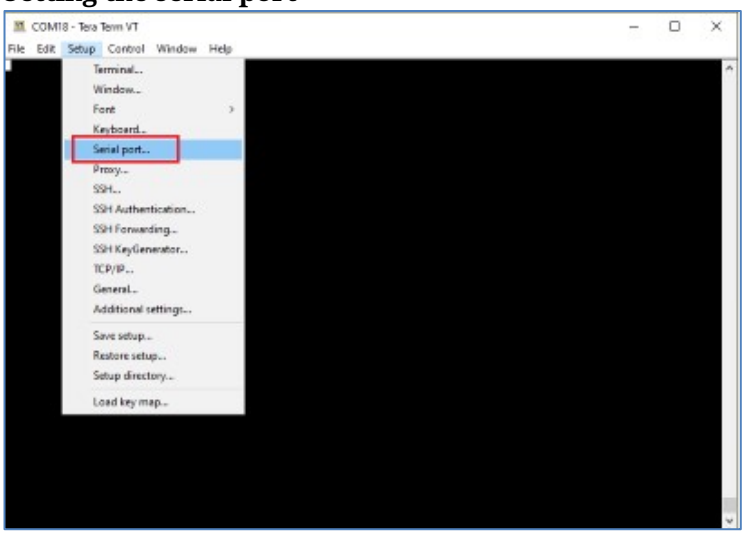
1. Flash the board with the Wireless UART application as previously described.
2. Open Tera Term (or any other Serial Communication software) and choose **Serial**. Then choose the port of your board and click **OK**.

   Setting up a new Tera Term connection for Wireless UART application

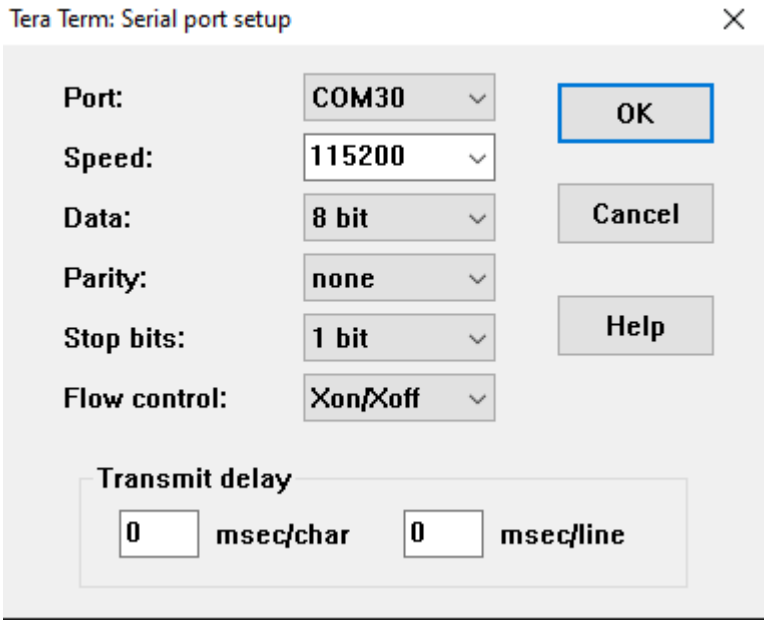
3. Stop running the code on the board. Go to **Setup** >**Serial Port**.

   Setting the serial port

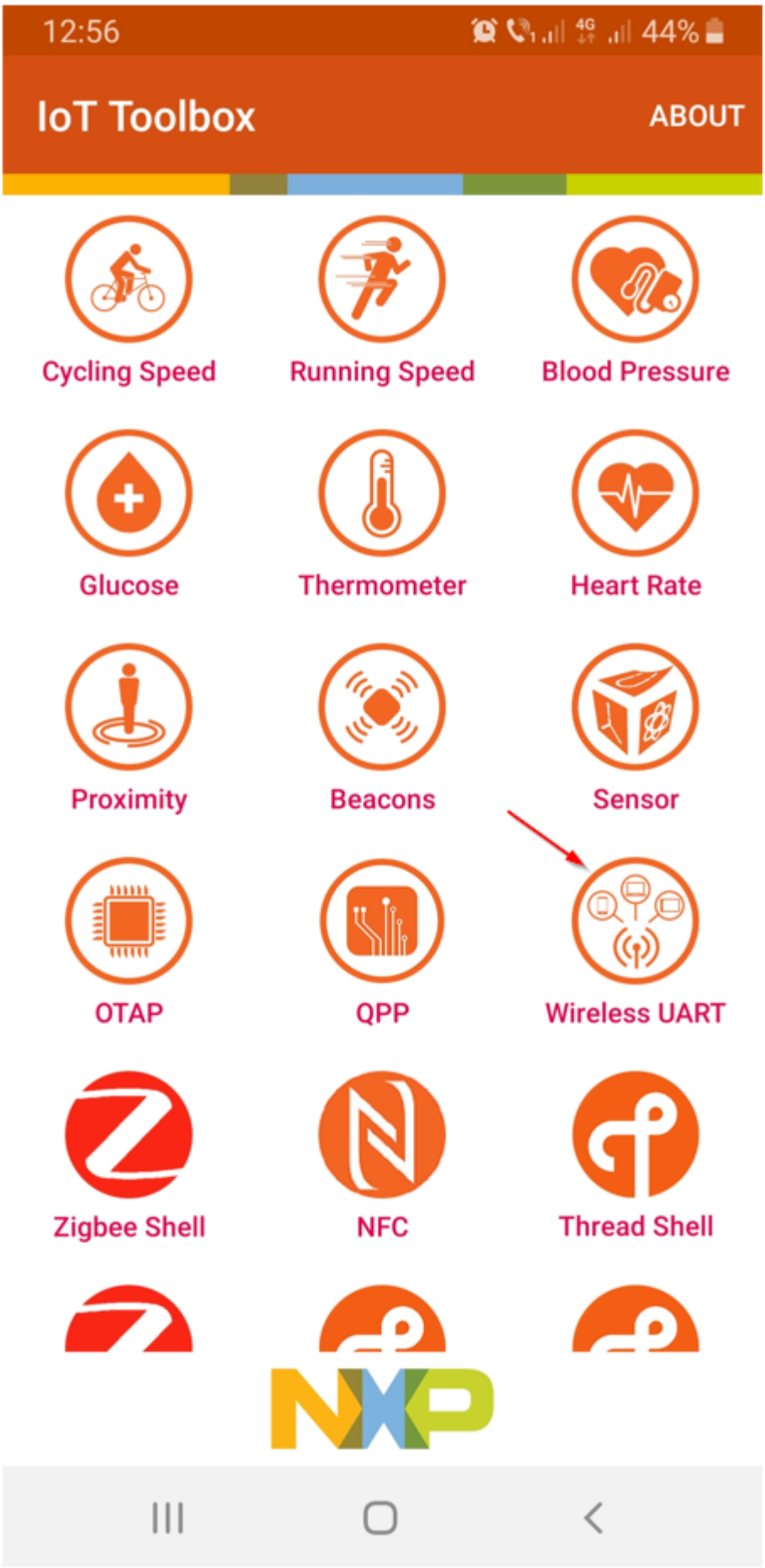
4. Choose the required Speed (baud rate) and Flow Control values. Here, the baud rate should be 115200 and the Flow control mode should be set to **Xon/Xoff**.

   Setting the baud rate



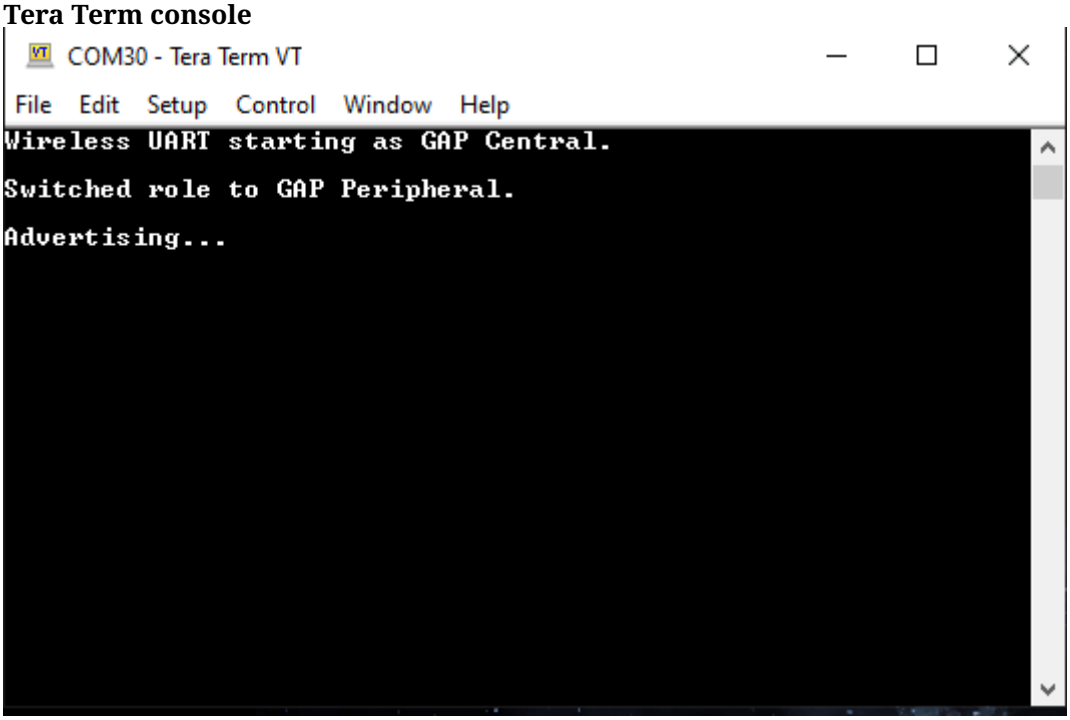
5. Ensure that the Bluetooth device of your phone is enabled.
6. Open the *IoT Toolbox* application and select the ***Wireless UART*** icon.

   NXP IoT Toolbox application

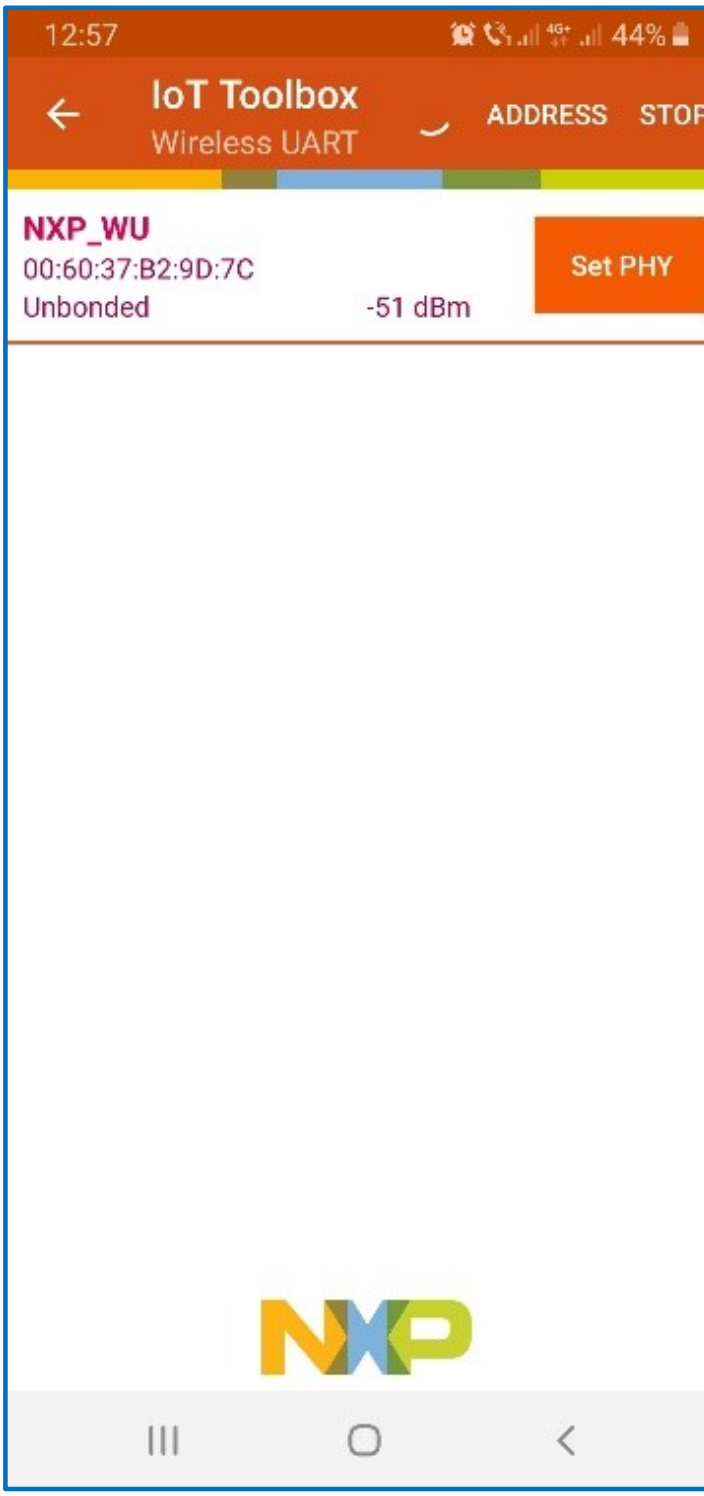

12:56
44%
IoT Toolbox
ABOUT
Cycling Speed
Running Speed
Blood Pressure
Glucose
Thermometer
Heart Rate
Proximity
Beacons
Sensor
OTAP
QPP
Wireless UART
Zigbee Shell
NFC
Thread Shell

7. Now run the application on the board. On this step, the RGB LED blinks white quickly. Press the button **SW3** to switch to Peripheral (Central) mode, then press **SW2** to start Advertising (scanning). The Tera Term terminal shows the message shown in the figure below.

Tera Term console



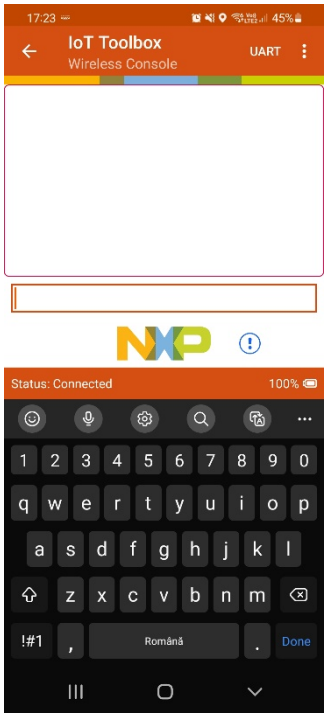
8. The device should become visible for the Wireless UART mobile application.



NXP Wireless UART scanned by IoT toolbox app

9. Select the device that appears in the **Wireless UART** tab to connect to it. After connecting, the mobile application shows the console, as shown below. Users can choose between the Wireless Console and Wireless UART tabs.

   The device should become visible for the Wireless UART mobile application.

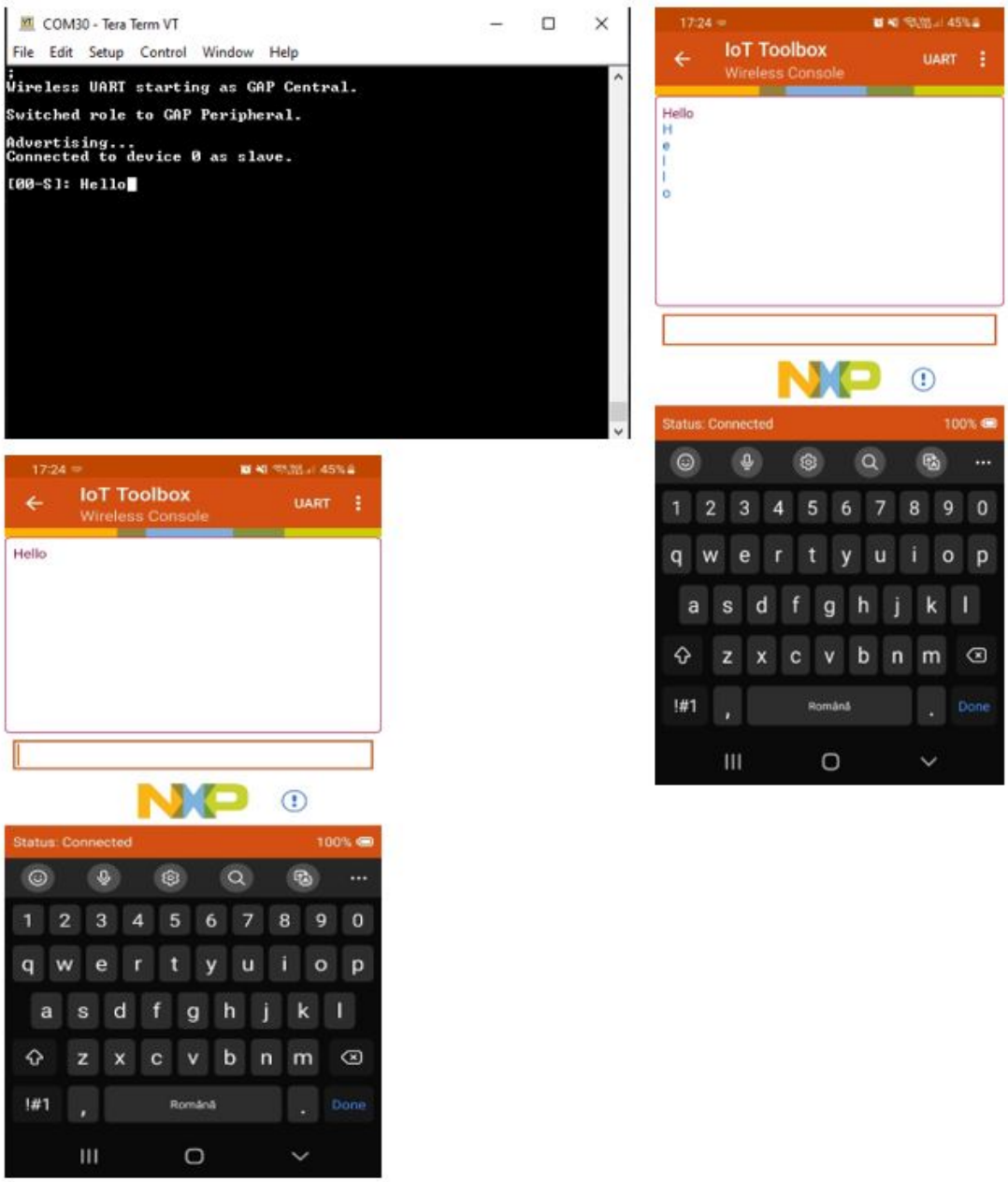


Wireless UART working in Connected mode

10. On the IoT Toolbox Wireless application console, the message is introduced in one line and sent to the peer. If Wireless UART mode is used, communication is done character by character. On Tera Term, the message received is displayed. One character at a time can be sent from Tera Term to the peer. When the mobile application receives the character, it displays it.

    The next figure shows the board acting as a peripheral, connected to a phone using IoT Toolbox in Wireless Console and Wireless UART modes.

    Exchanging messages between the phone and Wireless UART Console



References

For more information, refer to the NXP website or contact your local Field Application Engineer (FAE).

Acronyms and abbreviations

The following acronyms are used in this document.

Acronym	Description
Bluetooth LE	Bluetooth Low Energy
EVK	Evaluation Kit
IDE	Integrated Design Environment
ISP	In-system Programming
IoT	Internet of Things
RTOS	Real-time Operating System
SDK	Software Development Kit
UART	Universal Asynchronous Receiver Transmitter
USB	Universal Serial Bus

Note about the source code in the document

Example code shown in this document has the following copyright and BSD-3-Clause license:

Copyright 2023-2024 NXP Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Bluetooth Low Energy Demo Applications User Guide Introduction

This document describes the Bluetooth Low Energy host stack enablement for NXP development platforms. The document is organized as follows:

- *Bluetooth Low Energy applications* lists the demo applications that can be located in the Software Development Kit (SDK).
- *Hardware configurations* describes the hardware and toolchain requirements.
- *Building and running a Bluetooth LE example application* describes the general requirements for using and testing a Bluetooth Application on a compatible device.
- *Bluetooth LE stack and demo applications* describes the steps and instructions for using the demo applications on your device. It also presents the profiles and services implemented and how to interact with them.
- *References* lists the additional documents that can be referred for more information.
- *Acronyms* lists the acronyms used in this document.

Bluetooth Low Energy applications

The Software Development Package provides a Bluetooth Low Energy v5.3-compliant host stack implementation with a set of GATT-based profiles and services implemented on top. To demonstrate the device functionality, the following demo applications are implemented.

1. ANCS Client
2. Beacon Application
3. Bluetooth LE FSCI Black Box
4. EATT Central
5. EATT Peripheral
6. HCI Black Box
7. HID Host
8. HID Device (Mouse)
9. Low-power Temperature Sensor and Collector
10. Low-power Extended Advertising Central and Peripheral
11. OTAP Clients ATT and L2CAP and OTAP Server
12. Wireless UART demo application
13. Bluetooth LE Shell application
14. Hybrid (Dual-Mode) Bluetooth Low Energy and Generic FSK
15. Wireless UART Host
16. FSCI Bridge
17. NCP FSCI Black Box

Note: Refer to the application notes that are located in the '`documentation`' folder.

Hardware configurations

Hardware requirements

The NXP Bluetooth LE demo applications run on a selection of the following platforms:

- KW45B41Z-EVK
- KW45B41Z-LOC
- FRDM-MCXW71
- MCX-W71-EVK
- KW47-EVK
- KW47-LOC
- KW43-LOC
- FRDM-MCXW72
- MCX-W72-EVK
- MCXW72-LOC
- FRDM-MCXW23
- MCXW23-EVK
- FRDM-KW43

Parent topic:*Hardware configurations*

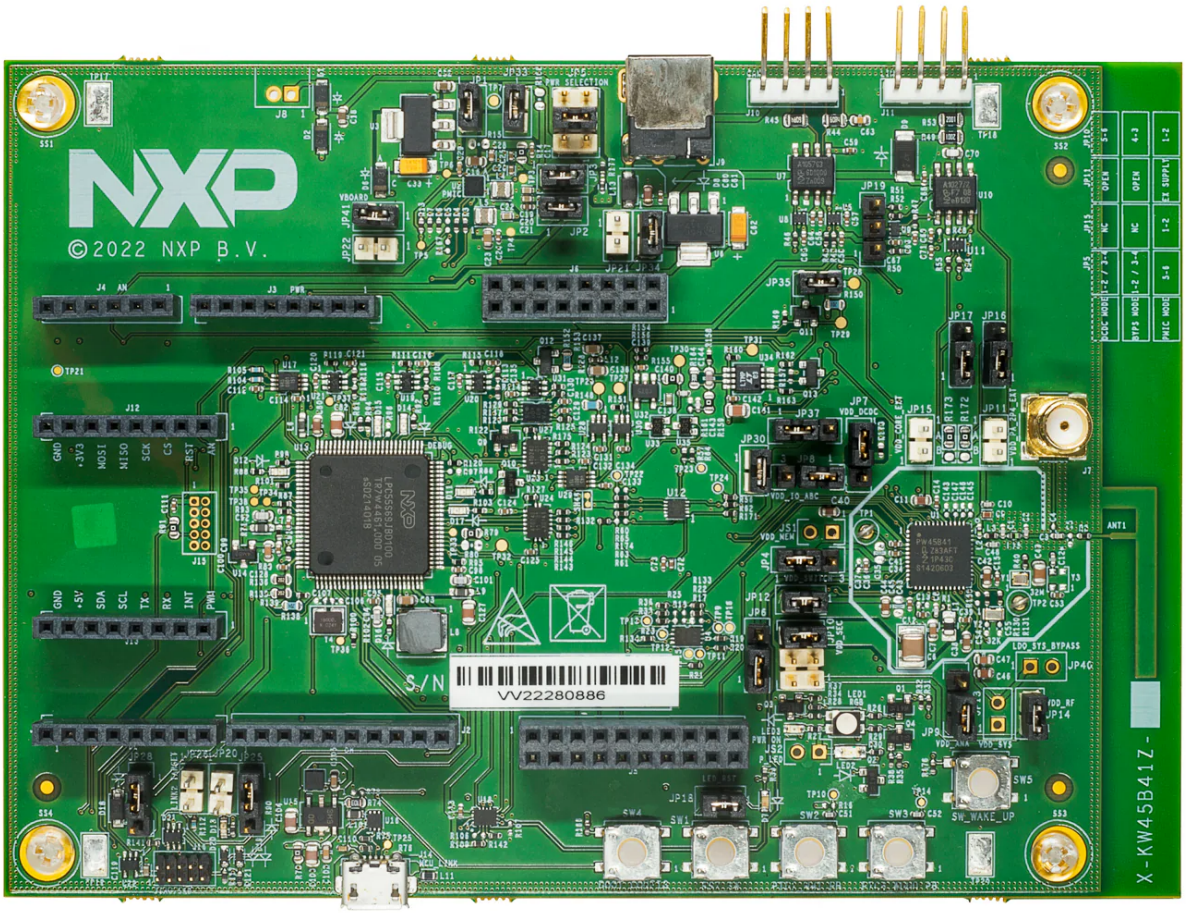
Toolchain requirements

The Bluetooth Low Energy Stack demo applications were compiled and tested with IAR Embedded Workbench for Arm and MCUXpresso. Users must use one of these tools.

Parent topic:*Hardware configurations*

KW45B41Z-EVK platform

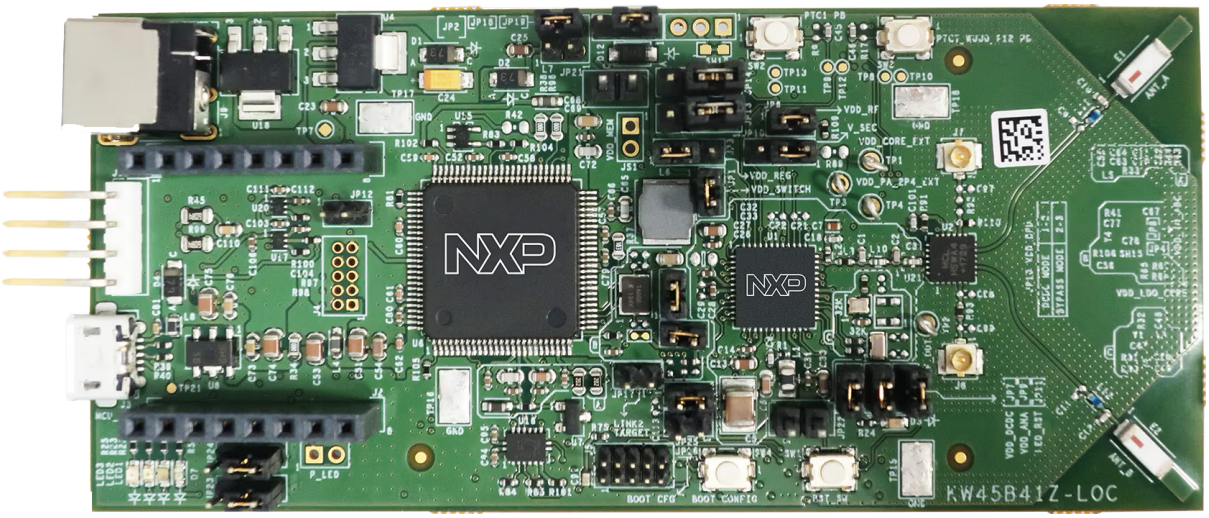
The figure below displays the top view of the KW45B41Z-EVK board.



Parent topic:*Hardware configurations*

KW45B41Z-LOC platform

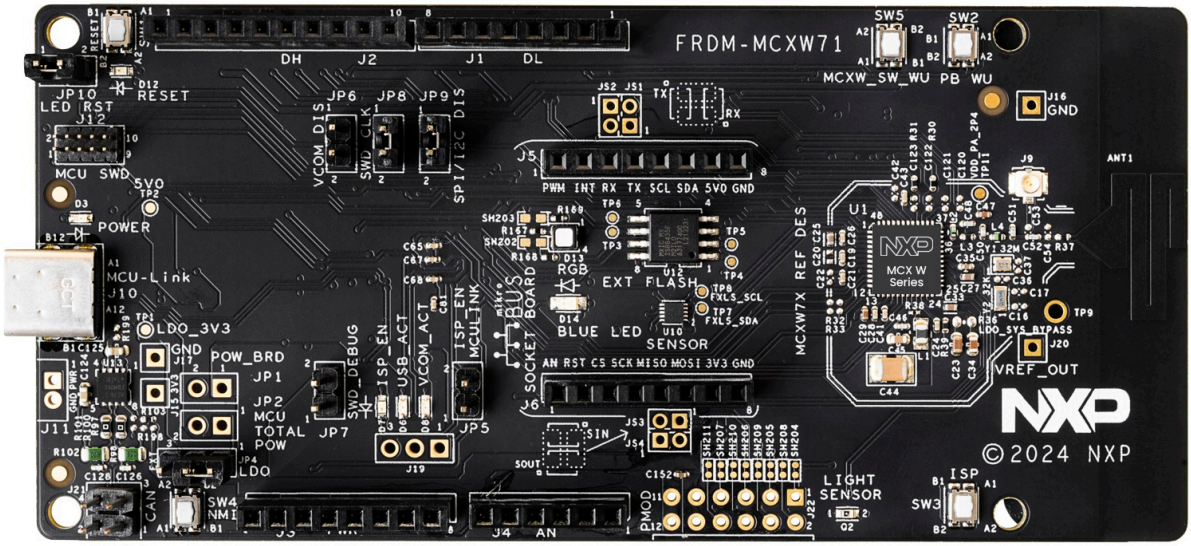
The figure below displays the top view of the KW45B41Z-LOC board.



Parent topic:*Hardware configurations*

FRDM-MCXW71 platform

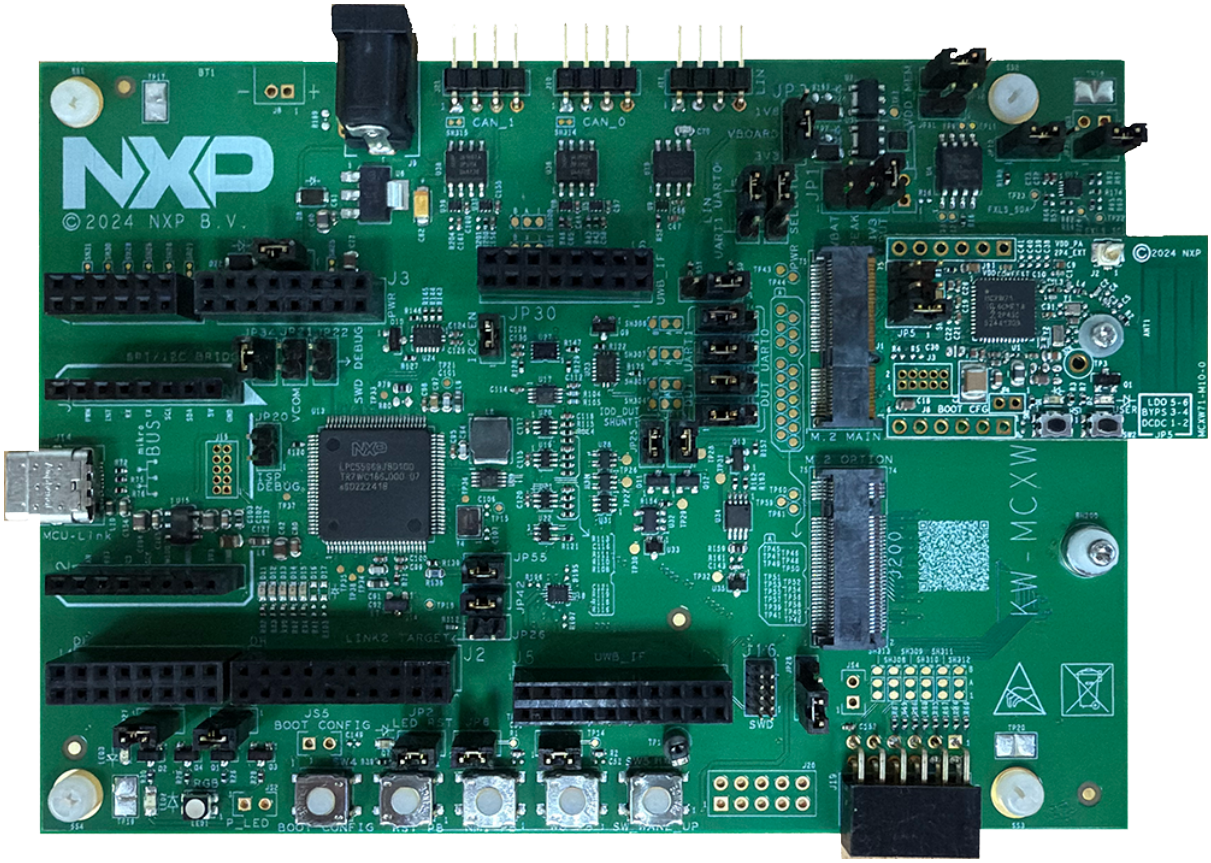
The figure below displays the top view of the FRDM-MCXW71 board.



Parent topic:*Hardware configurations*

MCX-W71-EVK platform

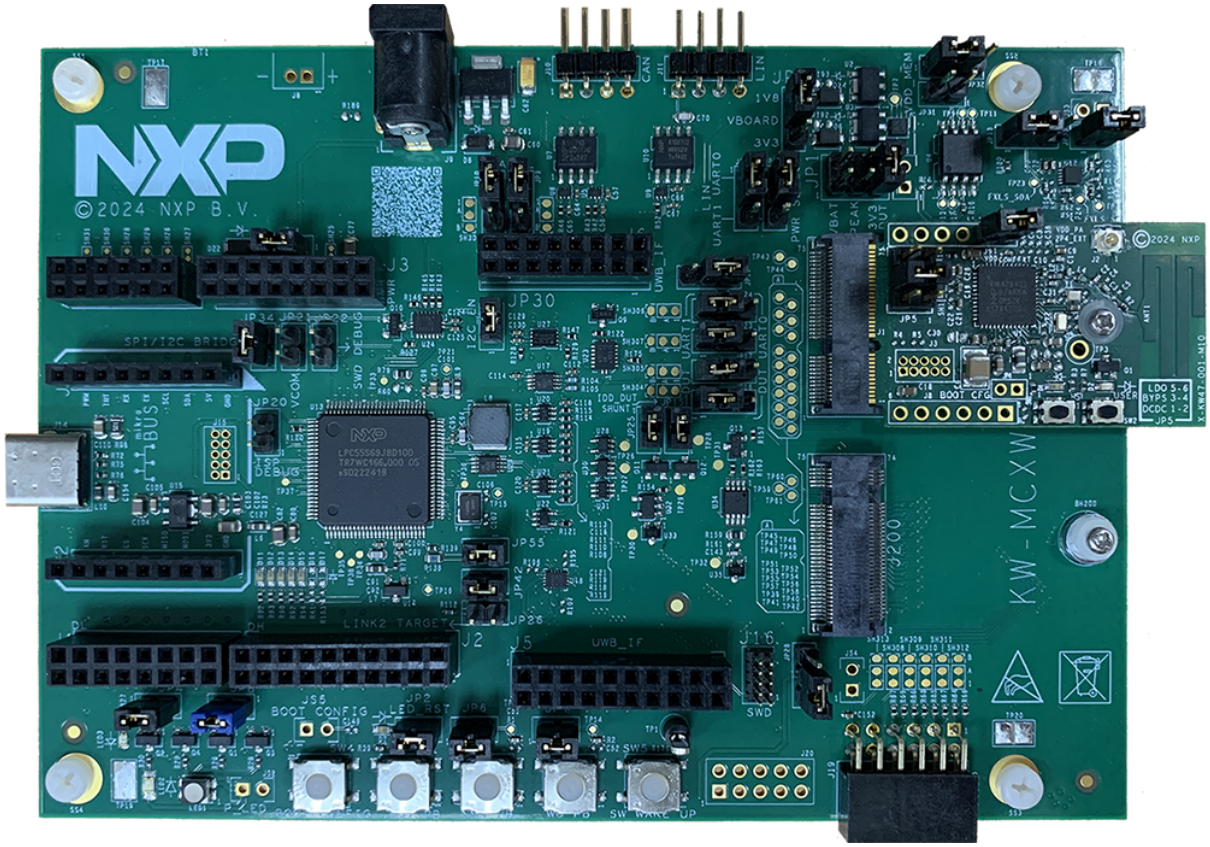
The figure below displays the top view of the MCX-W71-EVK board.



Parent topic:*Hardware configurations*

KW47-EVK platform

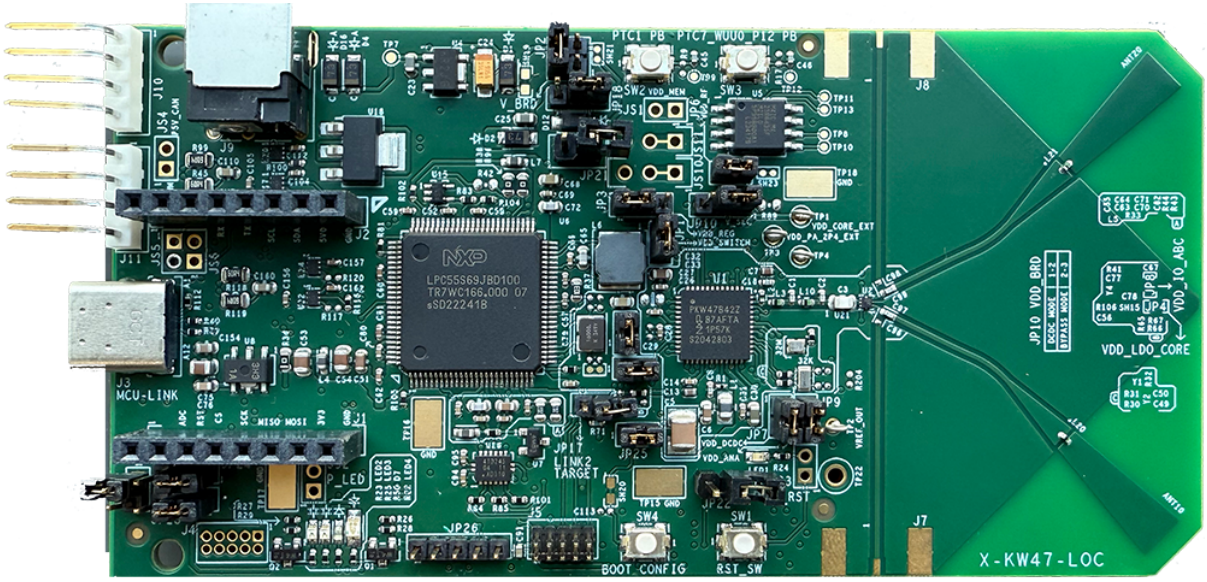
The figure below displays the top view of the KW47-EVK board.



Parent topic:*Hardware configurations*

KW47-LOC platform

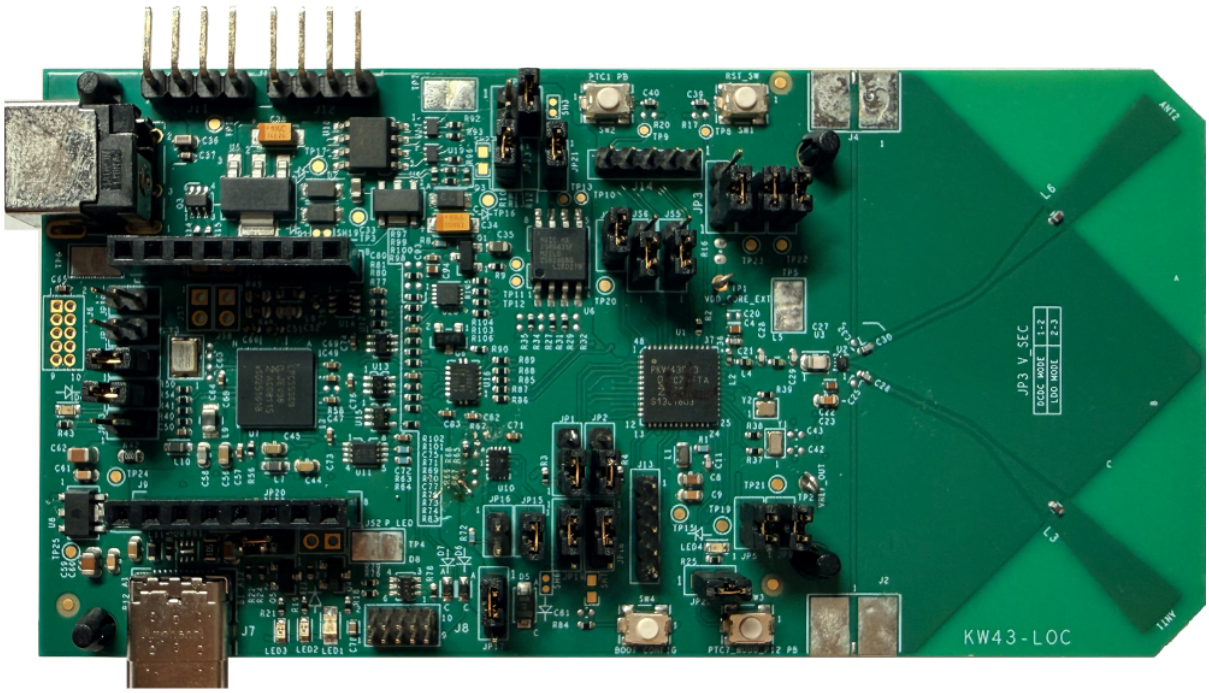
The figure below displays the top view of the KW47-LOC board.



Parent topic:*Hardware configurations*

KW43-LOC platform

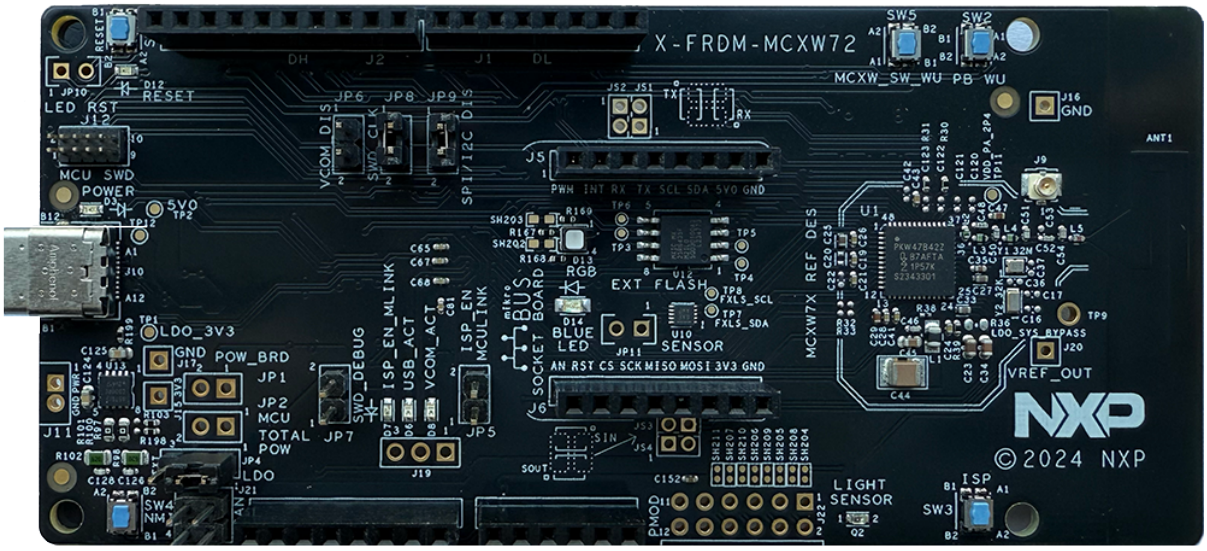
The figure below displays the top view of the KW43-LOC board.



Parent topic:*Hardware configurations*

FRDM-MCXW72 platform

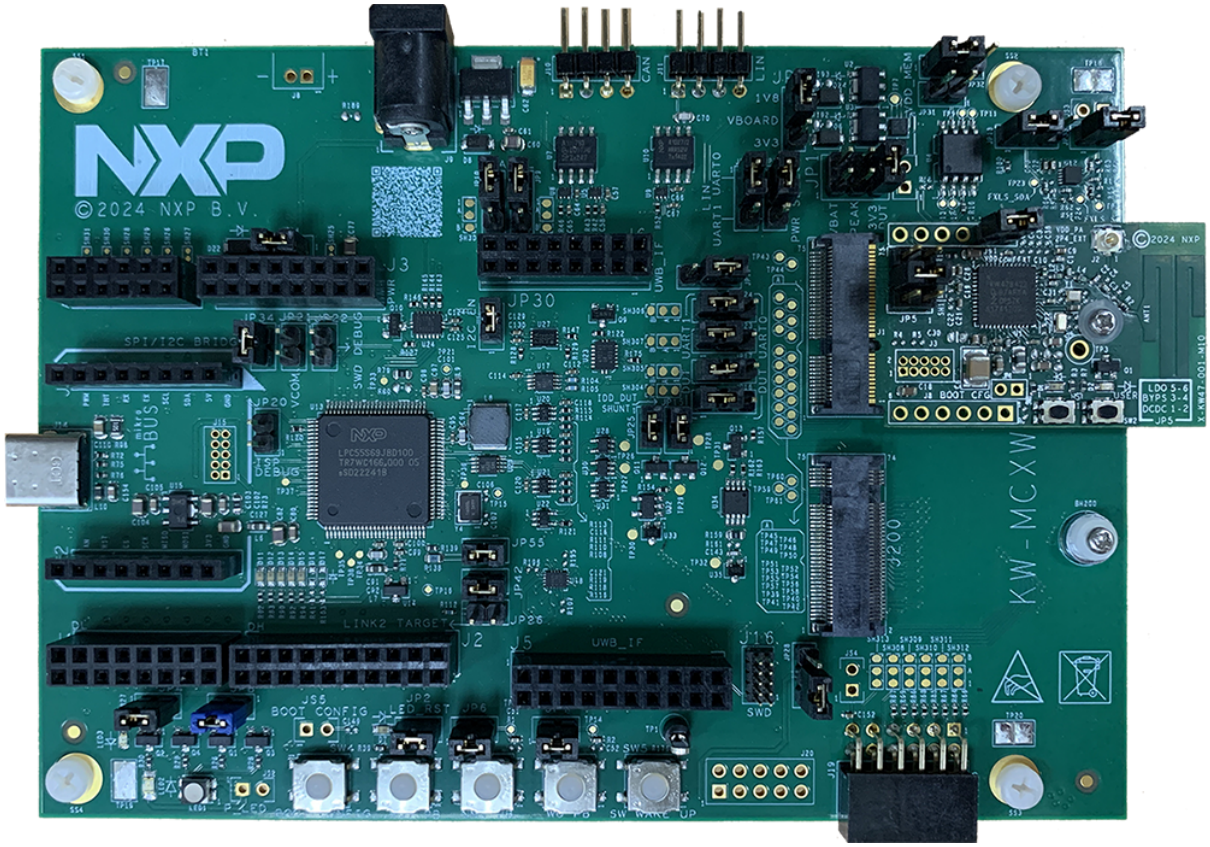
The figure below displays the top view of the FRDM-MCXW72 board.



Parent topic:*Hardware configurations*

MCX-W72-EVK platform

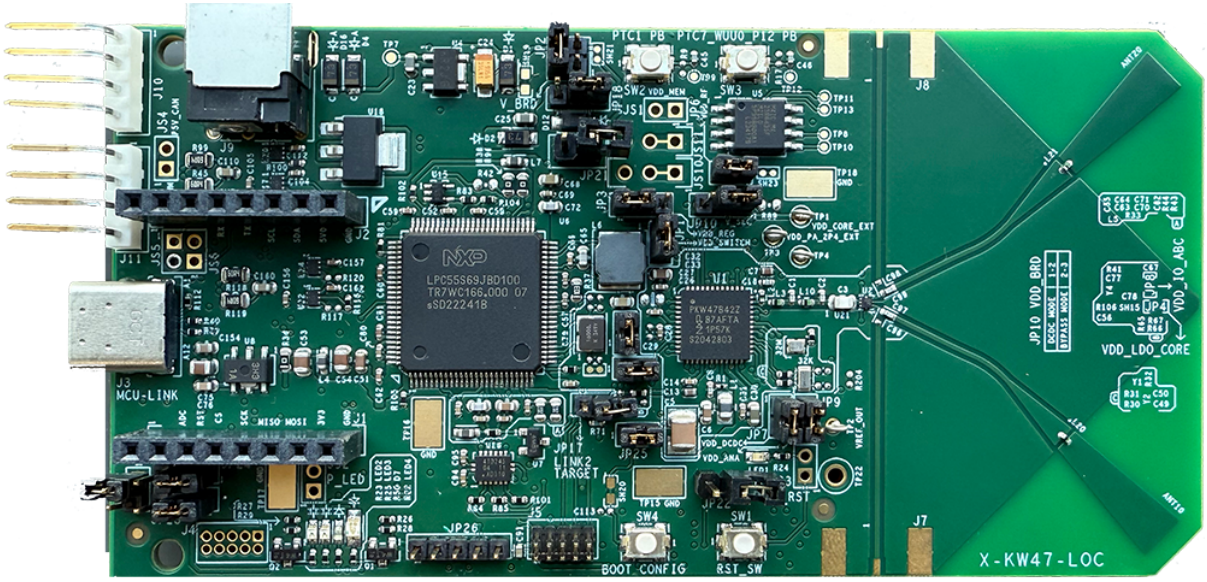
The figure below displays the top view of the MCX-W72-EVK board.



Parent topic:*Hardware configurations*

MCXW72-LOC platform

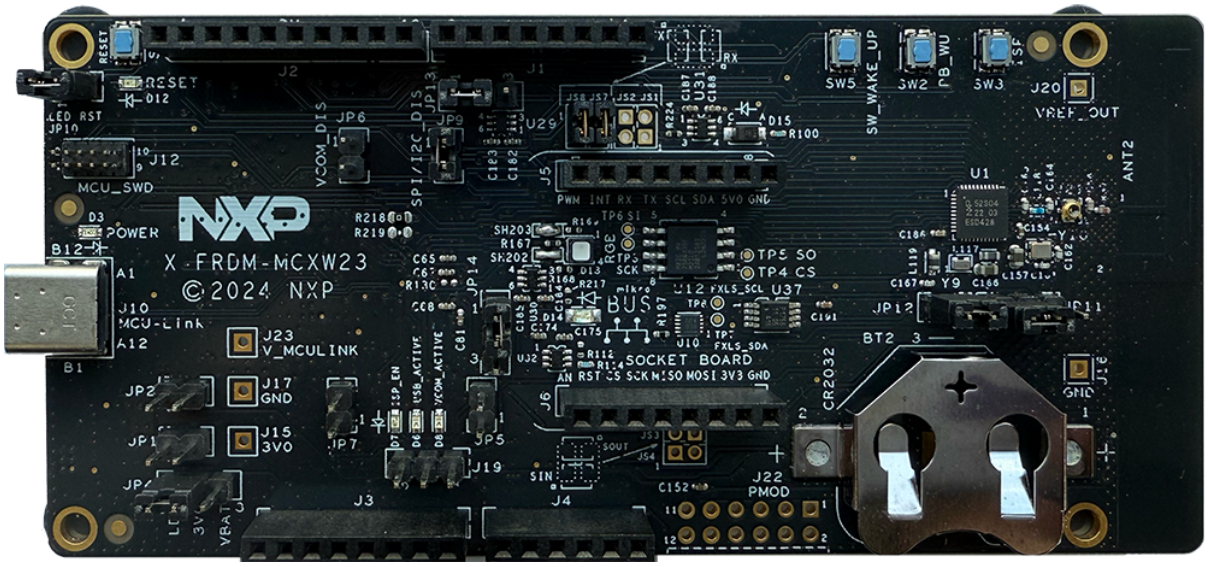
The figure below displays the top view of the MCXW72-LOC board.



Parent topic:*Hardware configurations*

FRDM-MCXW23 platform

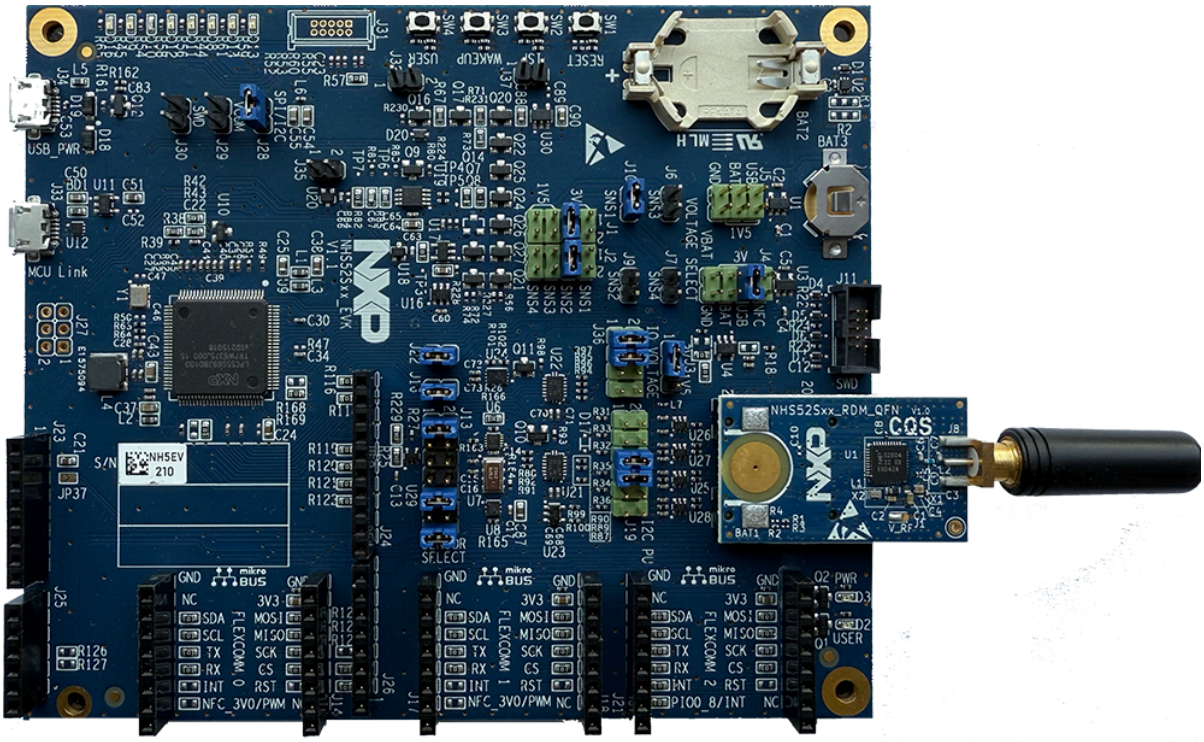
The figure below displays the top view of the FRDM-MCXW23 board.



Parent topic:*Hardware configurations*

MCXW23-EVK platform

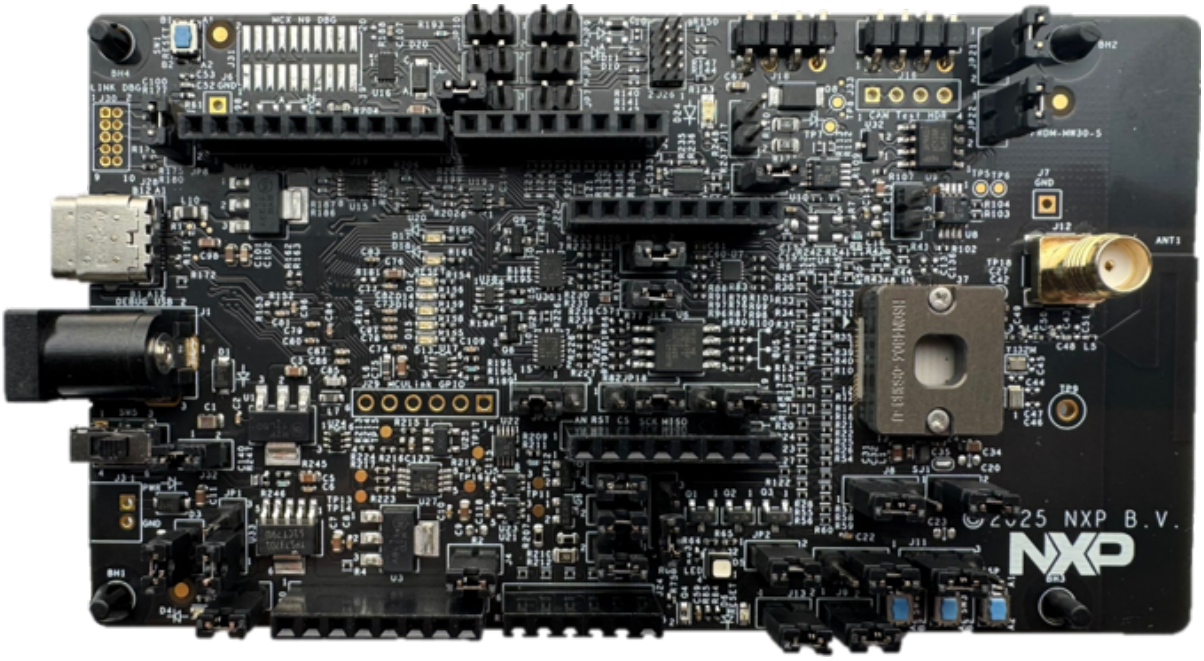
The figure below displays the top view of the MCXW23-EVK board.



Parent topic:*Hardware configurations*

FRDM-KW43 platform

The figure below displays the top view of the FRDM-KW43 board.



Parent topic:*Hardware configurations*

Building and running a Bluetooth LE example application

This section presents the general requirements for using and testing a demo application. To open, build, and run any example application on a specific board, refer to the Bluetooth Low Energy Quick Start Guide document for the corresponding board.

User interface

The demo applications that implement the Battery Service expose the current battery level, as measured on the board, through the Battery Level characteristic. The value represents a percentage between 0 and 100. The value can be read from the device from a connected GATT client.

The demo applications that implement the Device Information Service display various information regarding the current software, hardware, and firmware revisions. These values are used as an example and application developers can modify them when developing their product. The values can be read from the device through a connected GATT client.

Parent topic:*Building and running a Bluetooth LE example application*

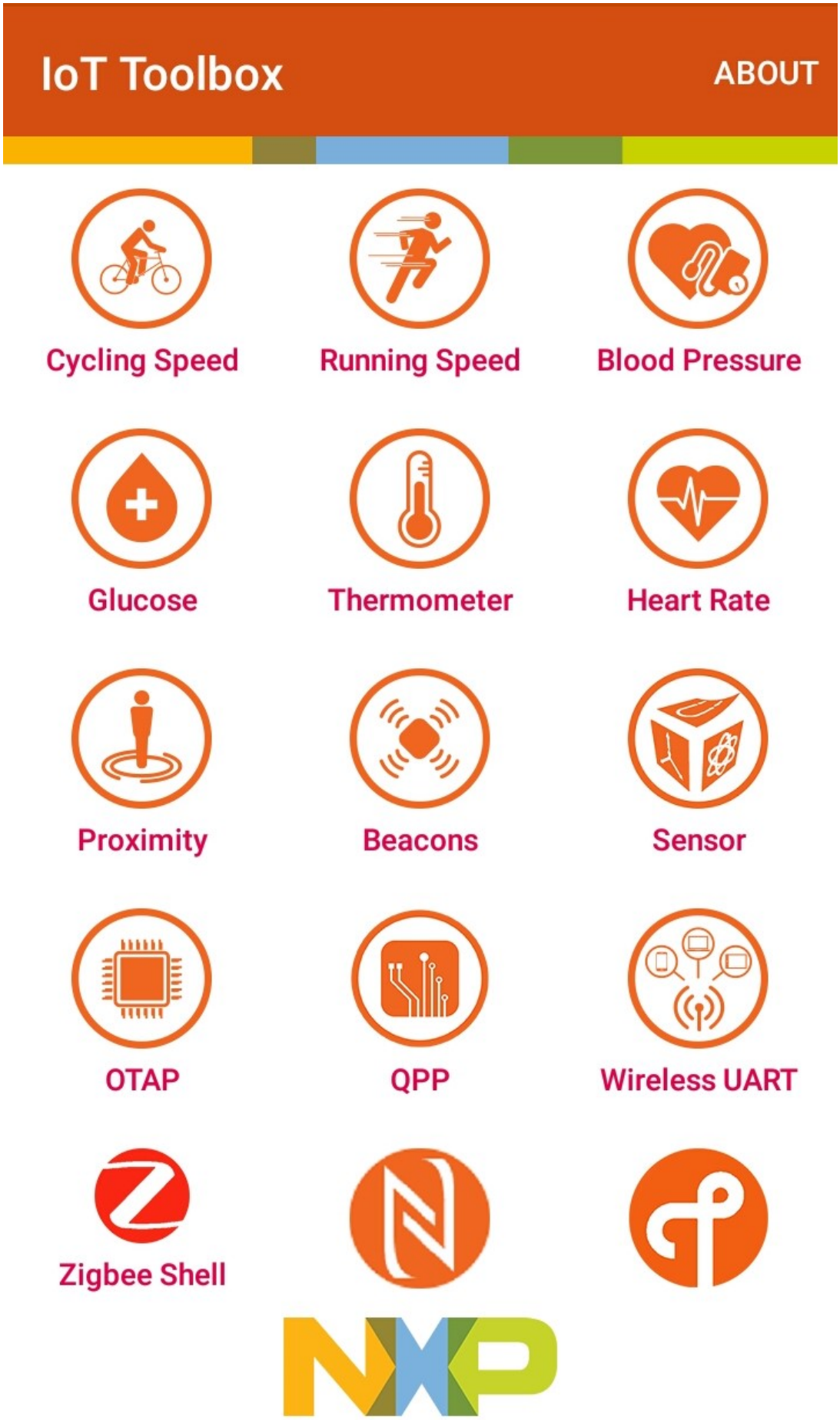
Security

The examples that enable pairing always generate a default passkey of 999999 that must be entered on the Central device, which is usually a smartphone or tablet.

Parent topic:*Building and running a Bluetooth LE example application*

Testing devices

To demonstrate the profile functionality, most of the scenarios require one of the supported platforms and a Bluetooth Low Energy capable central device. The device is usually a smartphone or a tablet that runs a compatible Bluetooth LE application. The figure below shows the **IoT Toolbox** UI.


IoT Toolbox
ABOUT
Cycling Speed
Running Speed
Blood Pressure
Glucose
Thermometer
Heart Rate
Proximity
Beacons
Sensor
OTAP
QPP
Wireless UART
Zigbee Shell

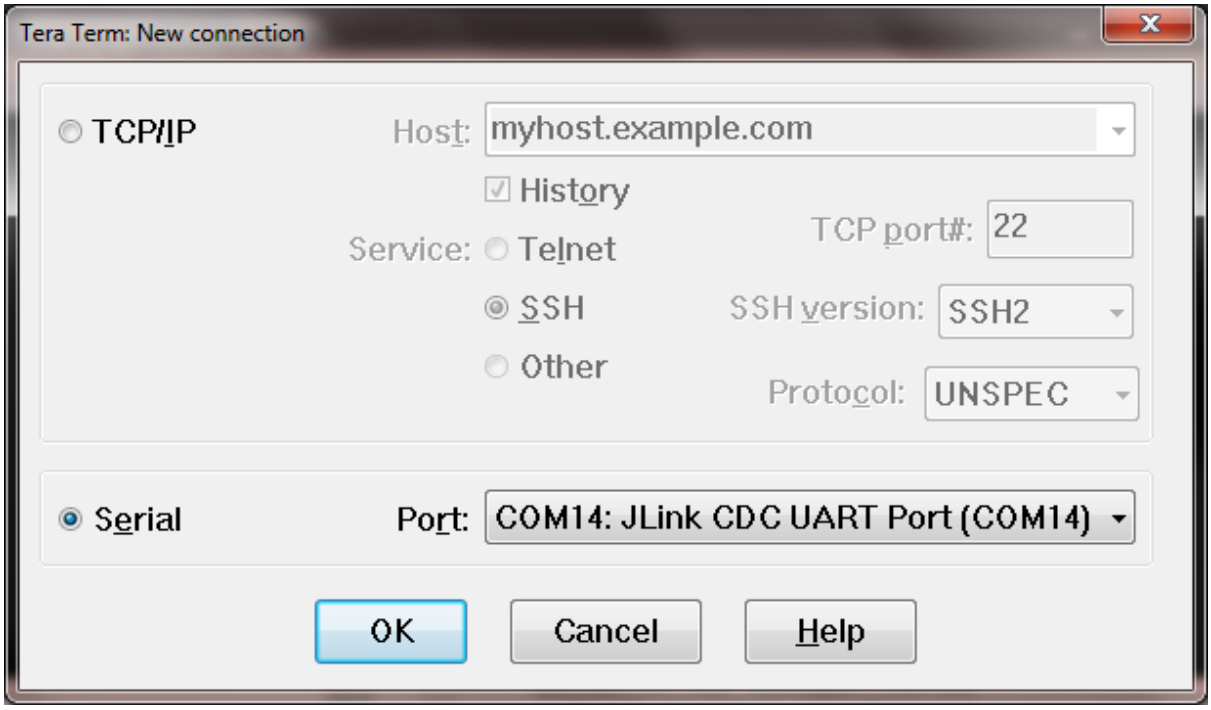
IOT toolbox

The recommended application is the IoT Toolbox, which can be installed on Apple iOS or Android OS handheld devices that support Bluetooth Low Energy. The application can be found on Apple Playstore or on Google Play.

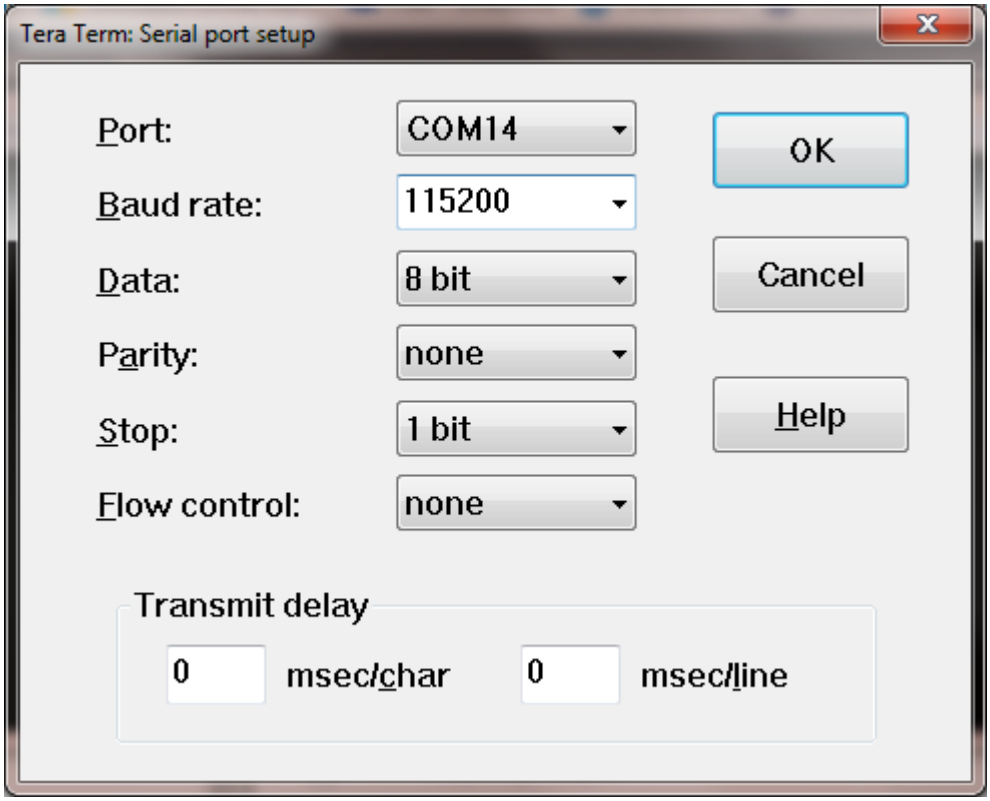
Other demos can be run by using two platforms, one for the peripheral and one for the central role. A few examples of such demos are listed below:

- Low-Power Temperature Sensor and Collector
- Wireless UART
- OTAP Client and Server
- HID Host and Device
- Extended Advertising Central and Peripheral
- EATT Central and Peripheral

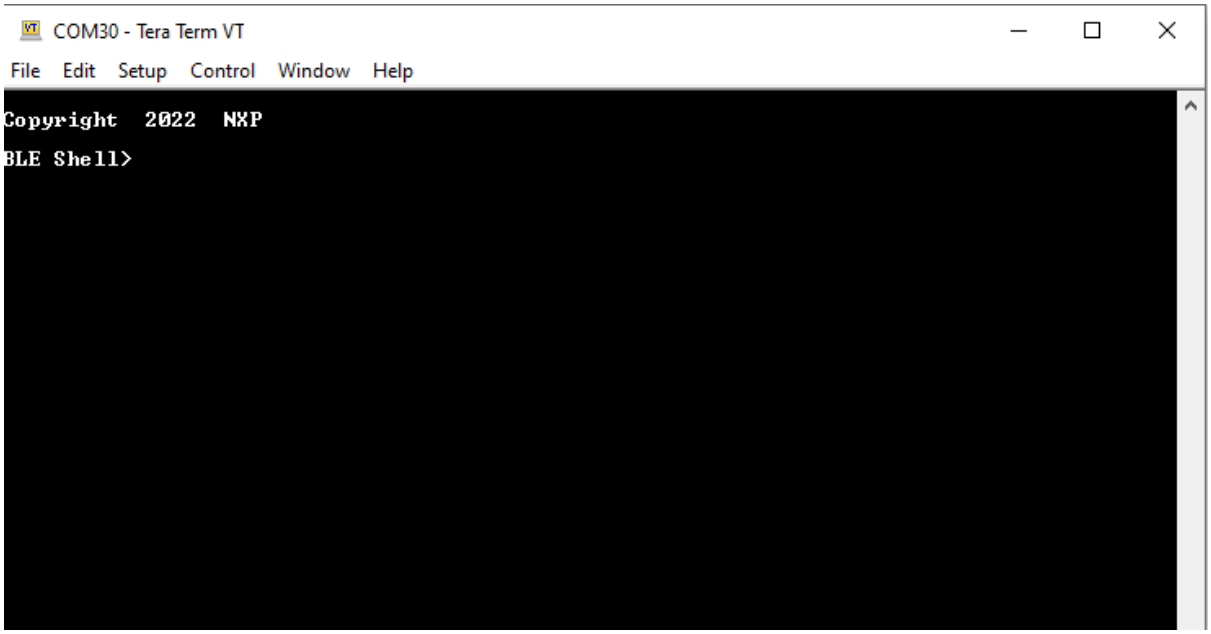
To provide feedback and more interaction, some examples use a shell console via the virtual COM port. To access the device, open a serial port terminal and as shown in the figure below. For this example, Tera Term VT and a KW45B41Z-EVK or a FRDM-MCXW71 board can be used. The communication parameters are 115200 and 8N1.



Connect it to the platform with parameters as shown in the figure below .



The start screen is displayed after the board is reset as shown in the figure below .



Parent topic:*Building and running a Bluetooth LE example application*

Time client devices

Some applications implement the Current Time Service. To enable this feature, define the gAppUseTimeService_d parameter in the app_preinclude.h file as 1. If the Time Client is enabled, the device must synchronize with a Time Server to update its internal date/time to the current date/time. If you connect the device to the phone, the Time Client synchronizes with the phone (pairing and bonding must be active).

Parent topic:*Building and running a Bluetooth LE example application*

Bluetooth LE stack and demo applications

ANCS/AMS client (ancs_c)

This section describes the implemented profiles, services, user interactions, and testing methods for the ANCS and AMS Client application.

Implemented profile and services

The ANCS/AMS Client application implements both an ANCS and an AMS Client for the custom ANCS Service and AMS Service available on iOS mobile devices.

Check the documentation available on the iOS website for details about the ANCS or AMS services, their characteristics, and supported features.

The demo application acts as a GAP Peripheral that advertises a service solicitation for the custom ANCS Service, followed by a solicitation to the AMS Service. It also acts as a GATT Client once connected to a device that offers the ANCS/AMS Service. The application offers some services such as the role of GATT Server.

Once connected to a mobile device offering the ANCS/AMS Service, the application displays information about ANCS Notifications received from that device. This information is followed by the AMS track information (Artist, Album, Title, Duration in seconds). The application also displays the possible remote commands that the device state allows (such as Play, Pause, VolumeUp, VolumeDown). The notifications are received via ATT Notifications, for which the ANCS Client must register on the peer ANCS Server. The same must be done for the AMS server. It initially configures the information that it wants to be notified about. The application also retrieves and displays additional information about the received ANCS notifications. For this purpose, it writes commands to specified characteristics on the ANCS/AMS Server and receives responses via ATT Notifications from other characteristics. All information is displayed to the user using a shell available over a serial communications interface.

Accessing the ANCS Service and AMS Service requires Bluetooth LE security to be enabled.

Parent topic:ANCS/AMS client (ancs_c)

Supported platforms

The ANCS/AMS Client application is supported on the following platform:

- KW45B41Z-EVK
- FRDM-MCXW71
- MCX-W71-EVK
- KW47-EVK
- FRDM-MCXW72
- MCX-W72-EVK
- FRDM-KW43

Parent topic:ANCS/AMS client (ancs_c)

User interface

After flashing the board, the device is in idle mode (all LEDs flashing). To start advertising, press the **ADVSW** button. When in GAP Discoverable Mode, **CONNLED** is flashing. When the ANCS/AMS Server (Gap Central) connects to the ANCS/AMS Client (GAP Peripheral), **CONNLED** turns solid. To disconnect, hold the **ADVSW** for 2-3 seconds. The ANCS/AMS Client then re-enters the advertising state.

For displaying operating information and ANCS Notifications (AMS information and commands), the demo application uses a shell exposed via a serial communication interface.

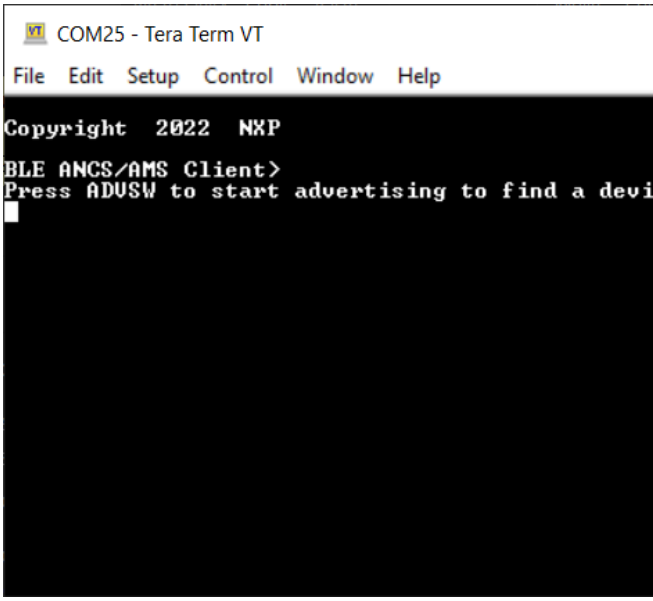
Parent topic:ANCS/AMS client (ancs_c)

Switches and pins

Usage

The ANCS/AMS Client demo application is designed to work with a peer mobile device that exposes the ANCS and AMS service. Also, a serial terminal application is required for displaying ANCS Notifications information and AMS commands and information.

1. Open a serial terminal application on the PC and connect it to the serial port corresponding to the board on which the ANCS/AMS Client runs. See the details in Testing devices, "User Interface". A start screen is displayed immediately after the board is reset. All LEDs must be flashing as shown in the figure below.



Start screen for ANCS/AMS Client demo application

2. Press the **ADVSW** button to start advertising. This instruction is also displayed in the serial terminal as shown in the figure above.
3. The peer device starts scanning for Bluetooth LE devices and connects to the ANCS/AMS Client device that is advertising.
4. Once connected to a peer, the application looks for the ANCS Service and AMS Service and their characteristics. If they are found, the ANCS Client tries to register for receiving notifications and AMS for the tracking data and commands. See Figure.
5. If any security-related ATT errors are encountered, then the application automatically performs Pairing and Bonding and retries the failed ATT operations. Depending on the negotiated pairing method, user interaction might be needed to complete the Pairing. Follow the onscreen instructions provided by both the ANCS/AMS Client and the mobile device. If the ANCS/AMS Client generates a passkey, then the default 999999 passkey is used. See the figure below that shows a 'Pairing is successful' message.

```
Pairing was successful!          Pairing completed successfully
Writing ANCS Notification Source CCCD...        Enabling ANCS Notifications
ANCS Notification Source CCCD written successfully.  Success!
Writing ANCS Data Source CCCD...        Configuring ANCS Notifications
ANCS Data Source CCCD written successfully.  Success!
Writing AMS Remote Command CCCD...        Enabling Remote Command
AMS Remote Command CCCD written successfully.  Success!
Writing AMS Entity Update CCCD...        Enabling Entity Update
AMS Entity Update CCCD written successfully.  Success!
Writing AMS Entity Update Track Subscription...  Configuring Entity Update for Track Information
AMS Entity Update Track Subscription written successfully.  Success!
```

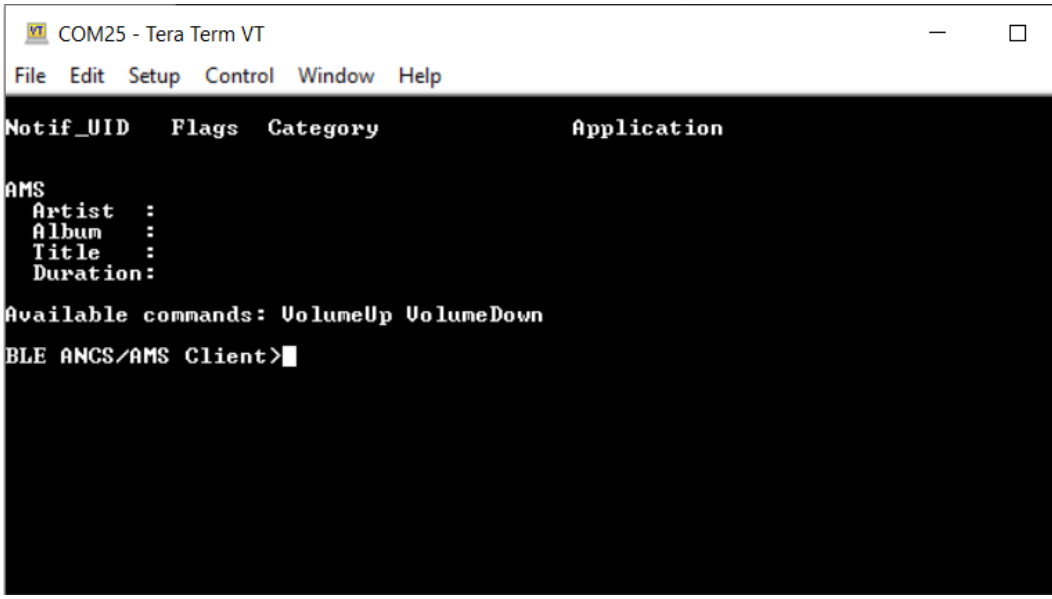
Pairing is successful message

6. After bidirectional communication is established via GATT, the ANCS/AMS Client starts displaying ANCS Notifications information as shown in the figure below.

```
BLE ANCS Client>
Notif_UID   Flags  Category        Application   3 existing notifications, 1 new
0x00000000  S_E_N  Social          Hangouts      notification for which the
0x00000001  S___N  Social          Hangouts      application name is not
0x00000002  S___N  Social          Hangouts      available yet
0x00000003  ____N  Email

BLE ANCS Client>
Notif_UID   Flags  Category        Application
0x00000000  S_E_N  Social          Hangouts      The application name for the
0x00000001  S___N  Social          Hangouts      new notification has been
0x00000002  S___N  Social          Hangouts      obtained from the ANCS Serv
0x00000003  ____N  Email           Mail

BLE ANCS Client>
Notif_UID   Flags  Category        Application   One notification was deleted
0x00000000  S_E_N  Social          Hangouts      by the ANCS Server and it has
0x00000001  S___N  Social          Hangouts      notified the ANCS Client to
0x00000002  S___N  Social          Hangouts      delete it too.

BLE ANCS Client>
Notif_UID   Flags  Category        Application   A second notification is delete
0x00000001  S___N  Social          Hangouts      by the ANCS Server.
0x00000002  S___N  Social          Hangouts

BLE ANCS Client>
Notif_UID   Flags  Category        Application   A third notification is deleted
0x00000002  S___N  Social          Hangouts      by the ANCS Server.

BLE ANCS Client>
Notif_UID   Flags  Category        Application   The last notification is deleted
                                                 by the ANCS Server.
BLE ANCS Client>
Notif_UID   Flags  Category        Application   1 new notification is received
0x00000004  ____N  Social                        from the ANCS Server for whic
                                                 the application name is not
BLE ANCS Client>                                 available yet.
Notif_UID   Flags  Category        Application   The application name for the
0x00000004  ____N  Social          Hangouts      last notification is retrieved fr
                                                 the ANCS Server.
BLE ANCS Client>
```

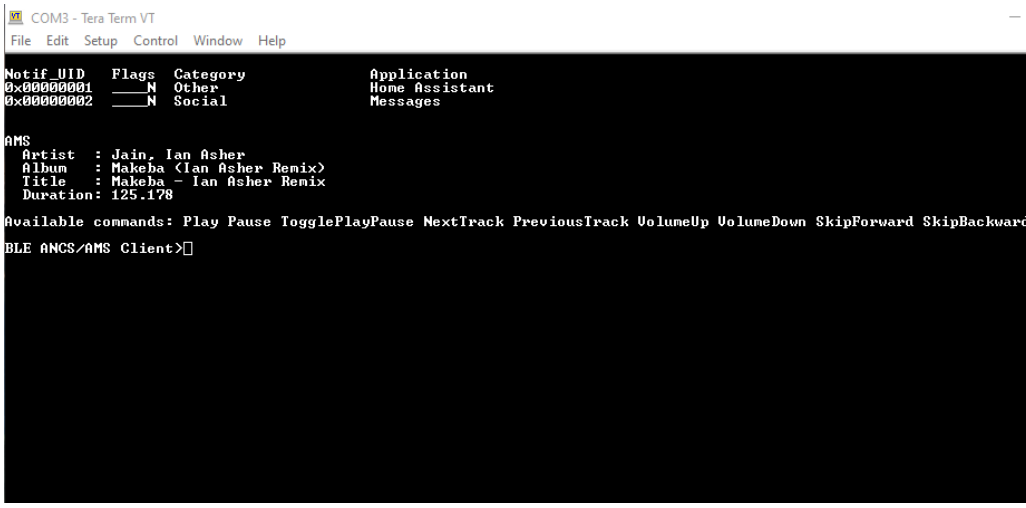
ANCS Notifications

- The combination of the two services (ANCS and AMS) is shown in the two figures below.
- If no media is playing and no player is active, the serial interface looks as shown in the figure below.



AMS no active player

- When media is playing, a player is active and the state must look similar to the figure below.



AMS with player active

Parent topic:ANCS/AMS client (ancs_c)

Parent topic:*Bluetooth LE stack and demo applications*

Beacon

This section presents the user interactions and testing methods for the Beacon application.

Advertising data

The beacons are non-connectable advertising packets that are sent on the three advertising channels. The latter contains the following fields.

- Company Identifier (2 bytes): 0x0025 (NXP ID as defined by the Bluetooth SIG).
- Beacon Identifier (1 byte): 0xBC (Allows identifying an NXP Beacon alongside with Company Identifier).
- UUID (16 bytes): Beacon sensor unique identifier.
- A (2 bytes): Beacon application data.
- B (2 bytes): Beacon application data.
- C (2 bytes): Beacon application data.

- RSSI at 1m (1 byte): Allows distance-based applications.

By default, the UUID value is a random value based on the unique identifier of the board.

Parent topic: Beacon

Supported platforms

The following platforms support the Beacon application:

- KW45B41Z-EVK
- FRDM-MCXW71
- MCX-W71-EVK
- KW47-EVK
- FRDM-MCXW72
- MCX-W72-EVK
- FRDM-KW43

Parent topic: Beacon

User interface

After flashing the beacon, the sensor is put in deep sleep (all LEDs are off). To flash the board in case the beacon is put in Deep-sleep mode, press the **ADVSW** or **RESET** button. After this step, any attached debugger loses its connection. The default configuration of the application enables low power, which disables LED support. The user can manually change the configuration and enable LED support, otherwise all subsequent LED behavior references are ignored.

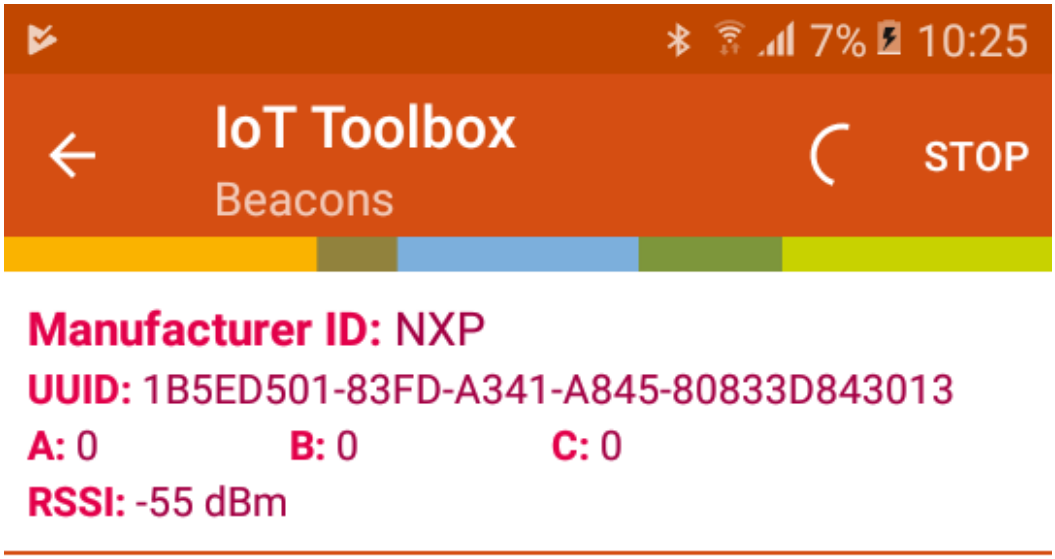
By default, the application uses extended advertising. However, it can be configured to use legacy advertising by setting the `gBeaconAE_c` define to `0`. In the legacy configuration, the first press of the advertising switch starts the legacy advertising and the second press stops it.

Parent topic: Beacon

Switches and pins

Usage

The beacon can be tested with any Bluetooth® Smart Ready products available on the market. The IoT Toolbox can also be used to showcase the profile functionality, as shown in the figure **IoT Toolbox Beacon Demo** below.


7% 10:25
IoT Toolbox
Beacons
STOP
Manufacturer ID: NXP
UUID: 1B5ED501-83FD-A341-A845-80833D843013
A: 0
B: 0
C: 0
RSSI: -55 dBm

Parent topic: Beacon

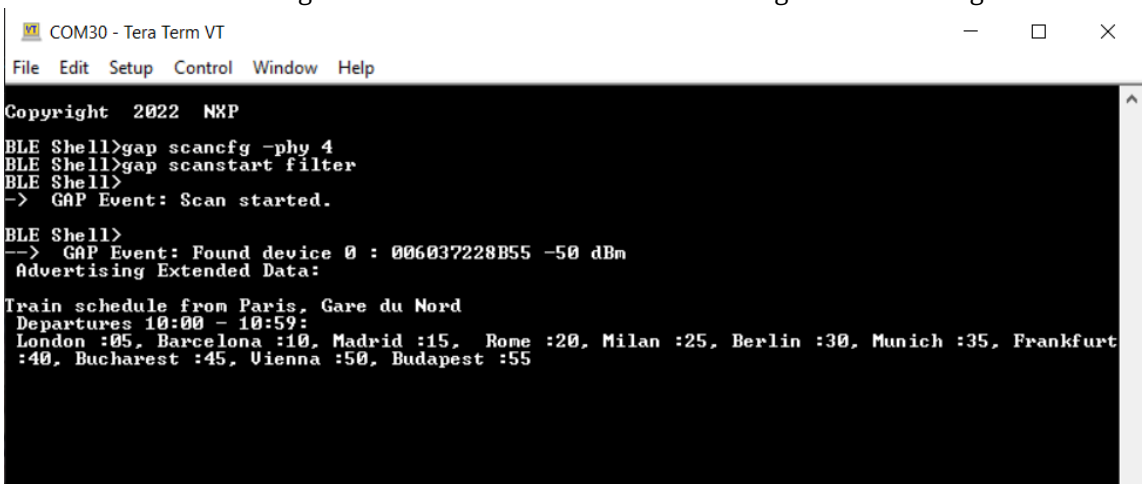
Beacon usage with extended advertising

To use the Beacon application with the advertising extensions capabilities, the gBeaconAE_c define option must be set to 1. Doing this enables the usage of extended advertising and periodic advertising. The application cycles between these modes are in the following manner:

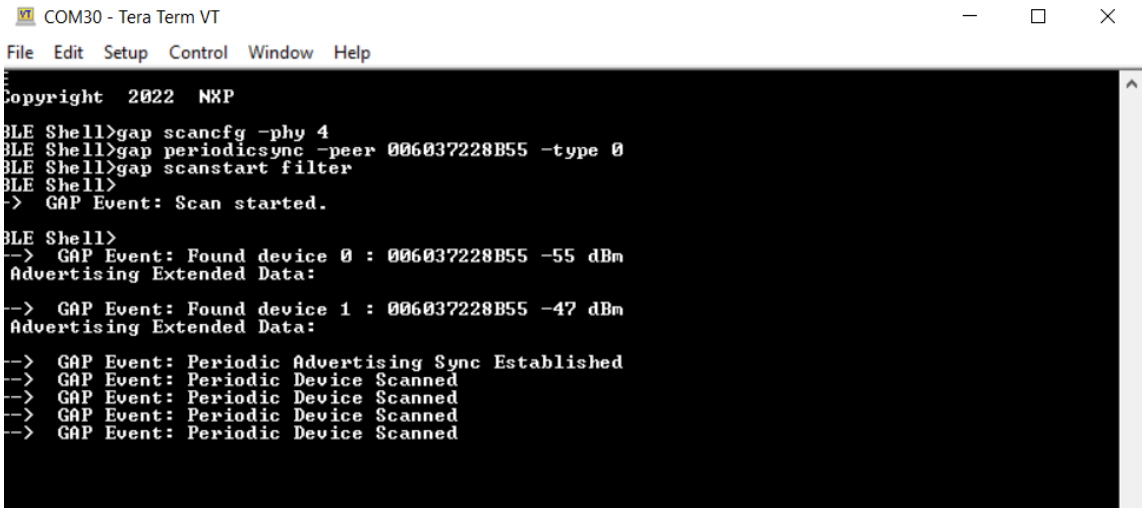
- The first **ADVSW** press starts legacy advertising, **CONNLED** turns solid.
- The second **ADVSW** press stops legacy advertising and starts extended advertising, **CONNLED** turns off, **EXTADVLED** turns solid.
- The third **ADVSW** press stops extended advertising carrying data and then starts extended advertising without data and periodic advertising, **EXTADVLED** starts flashing.
- The fourth **ADVSW** press stops periodic advertising and extended advertising without data and starts legacy advertising and extended advertising, both **CONNLED** and **EXTADVLED** turn solid.
- The fifth ADVSW press stops them all, both **CONNLED** and **EXTADVLED** turn off.

Not all smartphones support extended advertising, hence a different method to view the AE beacon is to use the ble_shell application. In order to do this, perform the following steps:

1. Flash a board with the beacon application, as described above.
2. Flash a board with the ble_shell application, as described in Bluetooth LE Shell and connect to it using a serial port.
3. Press the **ADVSW** button two times on the beacon to start extended advertising on the coded PHY.
4. To view the advertising data, enter the following commands in the shell terminal to set the scanning PHY to coded and start scanning. See the figure below.



5. To start the periodic advertising, press **ADVSW** button again on the beacon.
6. To sync with the beacon, issue the following commands on the shell terminal as shown in the figure below.

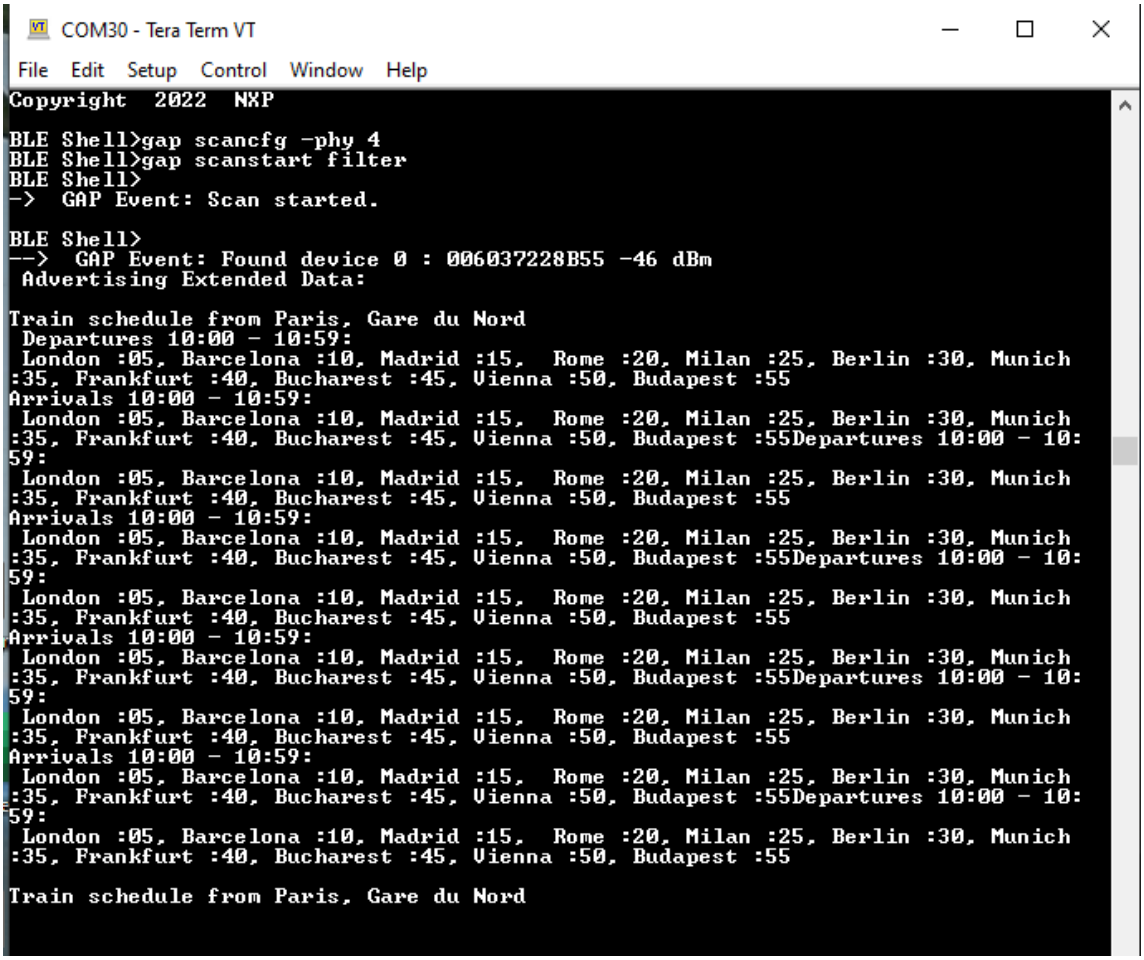


The peer parameter of the periodicsync command is the public address of the beacon.

Extended Advertising with very large data

To use very large advertising data for extended advertising, set the gBeaconLargeExtAdvData_c define to 1. The same steps are used to view the data using ble_shell :

1. Flash a board with the beacon application.
2. Flash a board with the ble_shell application, as described in Bluetooth LE Shell and connect to it using a serial port.
3. Press the **ADVSW** button two times on the beacon to start extended advertising on the coded PHY.
4. To view the advertising data, enter the following commands in the shell terminal to set the scanning PHY to coded and start scanning. See the figure below.



Parent topic: Beacon

Parent topic:*Bluetooth LE stack and demo applications*

Bluetooth LE FSCI Black Box

This section describes the functionality, user interactions, and testing methods for the Bluetooth LE FSCI Black Box demo application.

Description

The Bluetooth LE FSCI Black Box demo application gives access to the Bluetooth LE Host Stack via a serial interface using the FSCI protocol. See the *FSCI (Framework Serial Communication Interface) manual* for the format of the FSCI commands and a full list of supported commands.

The demo can be used with the Test Tool for Connectivity Products. Command Console application can be downloaded from the NXP website or using a custom application that supports the FSCI protocol and commands.

Parent topic: Bluetooth LE FSCI Black Box

Supported platforms

The following platforms support Bluetooth LE FSCI Black Box application:

- KW45B41Z-EVK
- FRDM-MCXW71
- MCX-W71-EVK

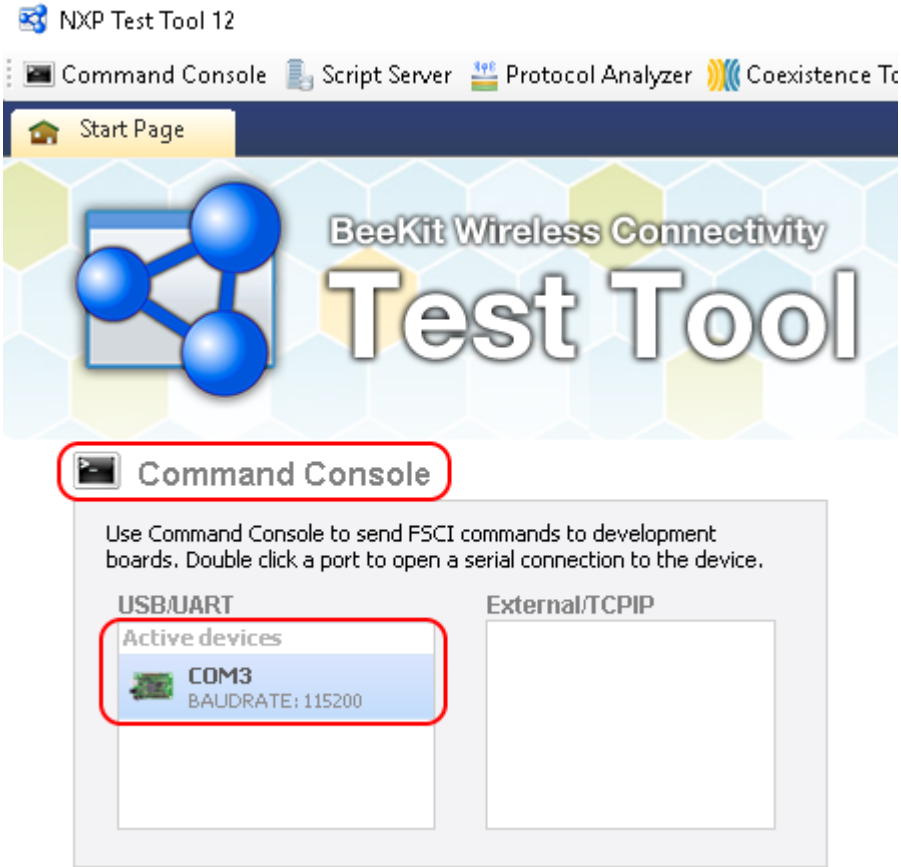
- KW47-EVK
- FRDM-MCXW72
- MCX-W72-EVK
- FRDM-KW43

Parent topic: Bluetooth LE FSCI Black Box

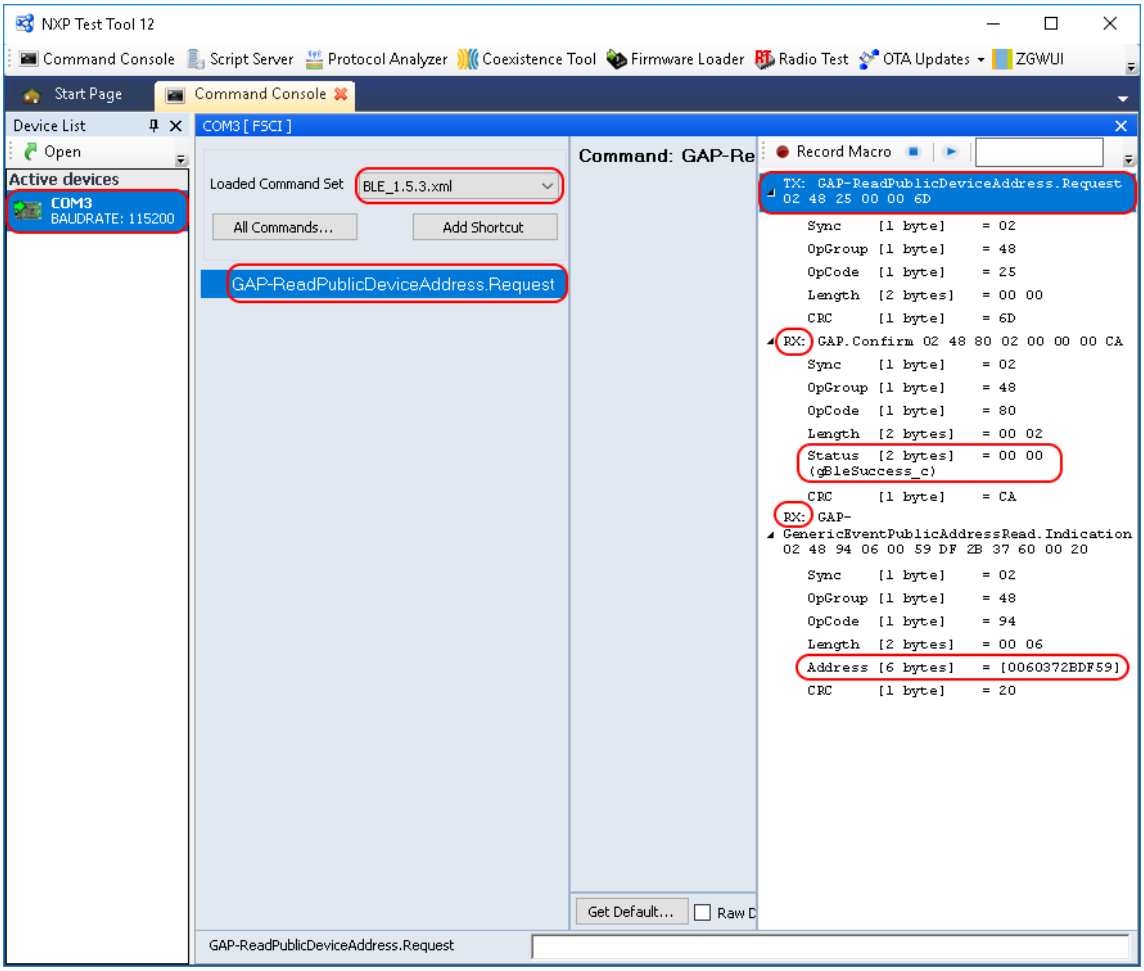
Usage with Test Tool for connectivity products

The Bluetooth LE FSCI Black Box demo application is designed to be used via serial interface. This can be done using the TEST Tool for Connectivity Products – Command Console application as described below.

1. Download the demo application onto a supported board.
2. Connect the board to a USB port of the PC. The UASB COM port drivers must be installed properly and a COM port corresponding to the board should be available.
3. Open the Test Tool application and connect to the serial port corresponding to the board on which the Bluetooth LE FSCI Black Box application runs. See Figure. The serial communication parameters are: baud rate 115200, 8N1, and no flow control.



4. Select the appropriate Test Tool XML file from the drop-down list for the release being used and send commands to the application. An example is shown in Figure.



Parent topic: Bluetooth LE FSCI Black Box

Parent topic:*Bluetooth LE stack and demo applications*

EATT Central

This section describes the implemented profiles and services, user interactions, and testing methods for the EATT Central application.

Implemented profiles and services

The EATT Central application implements a GATT server and the following profiles and services:

- Generic Attribute Profile

The application behaves as a GAP central node. It searches for an EATT peripheral to connect to. After connecting, it performs service discovery, initiates an EATT connection and configures indications on the peripheral for services A and B. The Central reports the received service data and the steps taken during the setup on a terminal connected to an UART port.

Parent topic:EATT Central

Supported platforms

The EATT Central application is supported on the following platforms:

- KW45B41Z-EVK
- FRDM-MCXW71
- MCX-W71-EVK

- KW47-EVK
- FRDM-MCXW72
- MCX-W72-EVK
- FRDM-KW43

Parent topic:EATT Central

User interface

After flashing the board, the central is in Idle mode (all LEDs flashing). To start scanning, press the **SCANSW** button. After connecting to the peripheral, the **CONNLED** turns solid. The data information together with the bearer it was received on is sent over UART. To disconnect the node, hold the **SCANSW** button pressed for 2-3 seconds. The node then restarts scanning.

Parent topic:EATT Central

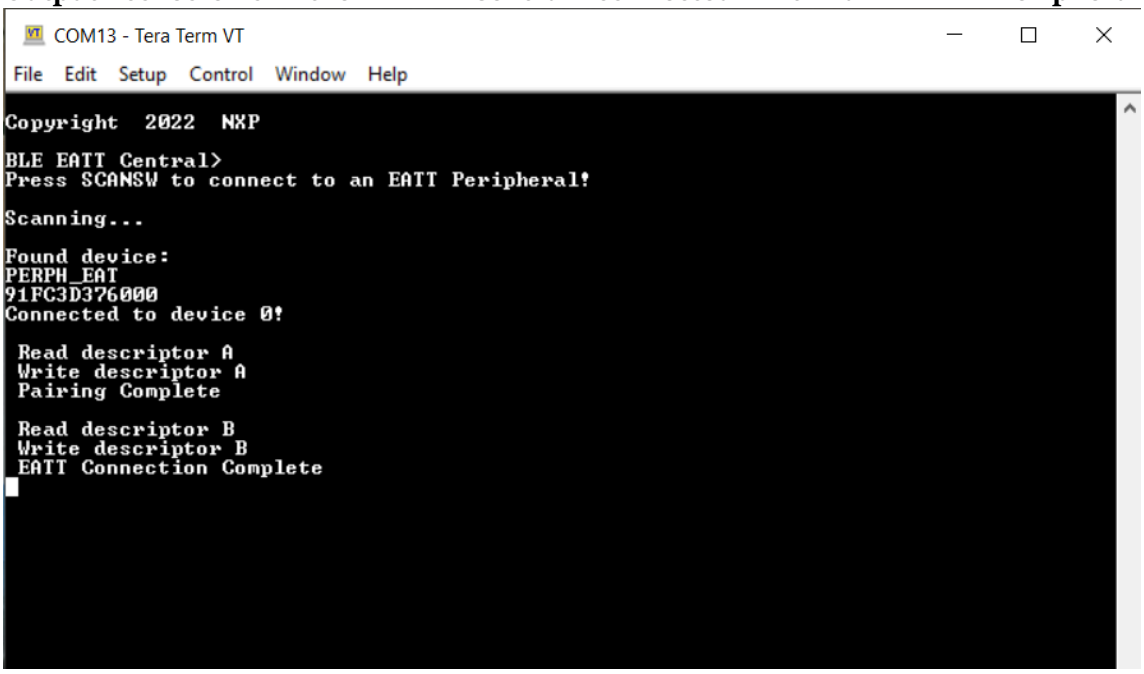
Switches and pins

Usage

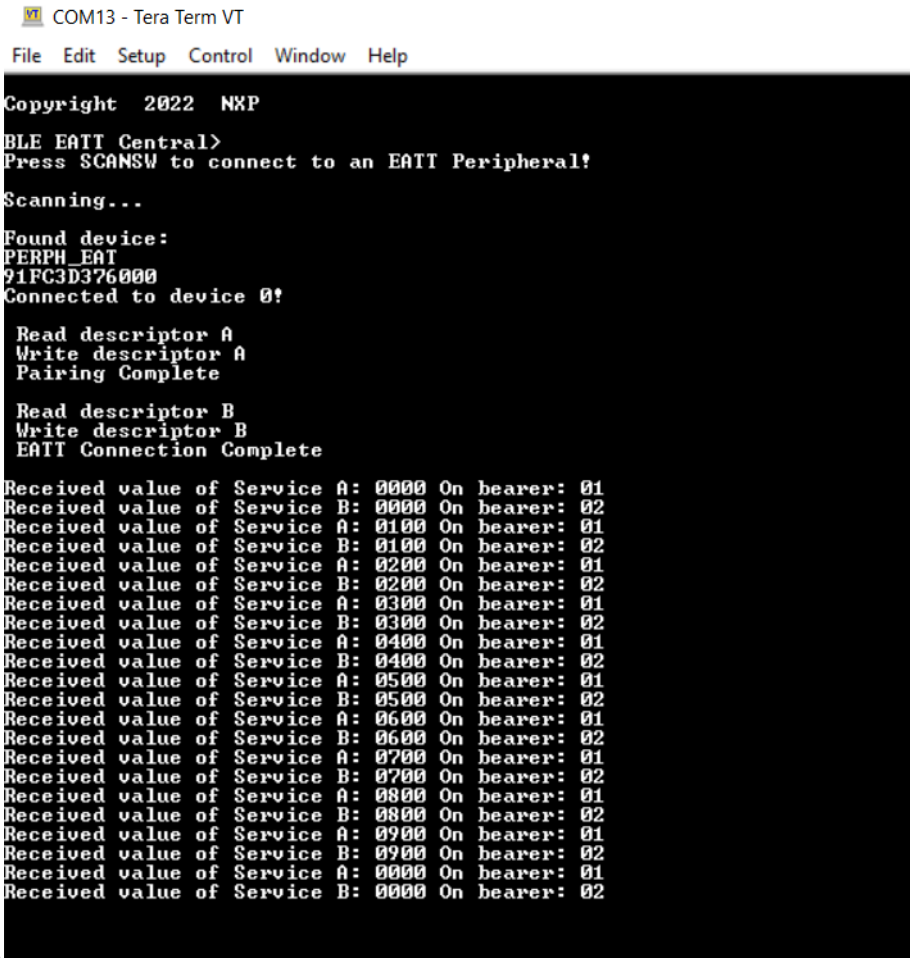
The application can be tested using another board flashed with the EATT Central application as described in the EATT Peripheral.

1. Open a serial port terminal and connect it to board, in the same manner described in Testing devices. The start screen is displayed after the board is reset.
2. To start scanning for devices, press the **SCANSW** button on the EATT Central board. To make it enter discoverable mode, perform the same step on the EATT Peripheral board. The host connects with the board after it sees it advertise the service A and service B UUIDs. After connecting, the central performs service discovery, indicates its EATT support to the server by writing the Client Supported Features characteristic, enables indications for services A and B, and then initiates an EATT connection with the server. See the figure below.

   Output console on the EATT Central connected with an EATT Peripheral



3. After the EATT connection is complete, the console displays the received data and the bearer on which it was sent, as shown in the figure below.



Output console on EATT Central

Parent topic:EATT Central

Parent topic:*Bluetooth LE stack and demo applications*

EATT Peripheral

This section describes the implemented profiles and services, user interactions, and testing methods for the EATT Peripheral application.

Implemented profiles and services

The EATT Peripheral application implements a GATT server and the following profiles and services:

- Battery Service v1.0
- Device Information Service v1.1
- Generic Attribute Profile

The Generic Attribute Profile includes the Server Supported Features characteristics, which is used to indicate EATT support to the peer, and the Client Supported Features characteristic, which is used by the peer to indicate its own EATT support.

The application behaves as a GAP peripheral node. It enters GAP General Discoverable Mode and waits for a GAP central node to connect. The application implements two custom services, Service A and Service B. After the EATT connection is completed, the peer must enable indications for the two services to periodically receive profile data over EATT.

Parent topic:EATT Peripheral

Supported platforms

The EATT Peripheral application is supported on the following platforms:

- KW45B41Z-EVK
- FRDM-MCXW71
- MCX-W71-EVK
- KW47-EVK
- FRDM-MCXW72
- MCX-W72-EVK
- FRDM-KW43

Parent topic:EATT Peripheral

User interface

After flashing the board, the peripheral is in idle mode (all LEDs flashing). To start advertising, press the ADVSW button. When in GAP Discoverable Mode, **CONNLED** is flashing. When a central node connects to the peripheral, **CONNLED** turns solid. The node then waits for an EATT connection request and for the client to configure indications for the two services. The service information is sent over enhanced bearers only. The values indicated are cycled between 0 and 10. To disconnect the node, hold the ADVSW button for 2-3 seconds. The node then re-enters GAP Discoverable Mode and starts advertising.

Parent topic:EATT Peripheral

Switches and pins

Usage

The application can be tested using another board flashed with the EATT Central application as described in the EATT Central Section.

Parent topic:EATT Peripheral

Parent topic:*Bluetooth LE stack and demo applications*

FSCI Bridge

This section describes the functionality, user interactions, and testing methods for the FSCI Bridge demo application.

Description

The FSCI Bridge demo application is a demonstration of the Extended NBU architectural concept, where the Bluetooth LE Host Stack and the Link Layer both run on the NBU core, while the user-facing application runs on the Application core. The communication between the application and the Host is done via the FSCI protocol running on the inter-core RPMSG transport

The FSCI Bridge runs on the Application core and it is a simple application whose purpose is to pass FSCI commands received over the serial interface to the NCP FSCI Black Box running on the NBU core. Effectively, the functionality is identical to the regular Bluetooth FSCI Black Box application.

The demo can be used with the Test Tool for Connectivity Products - Command Console application which can be downloaded from the NXP website or using a custom application that supports the FSCI protocol and commands.

Parent topic:FSCI Bridge

Supported platforms

The following platforms support the FSCI Bridge application:

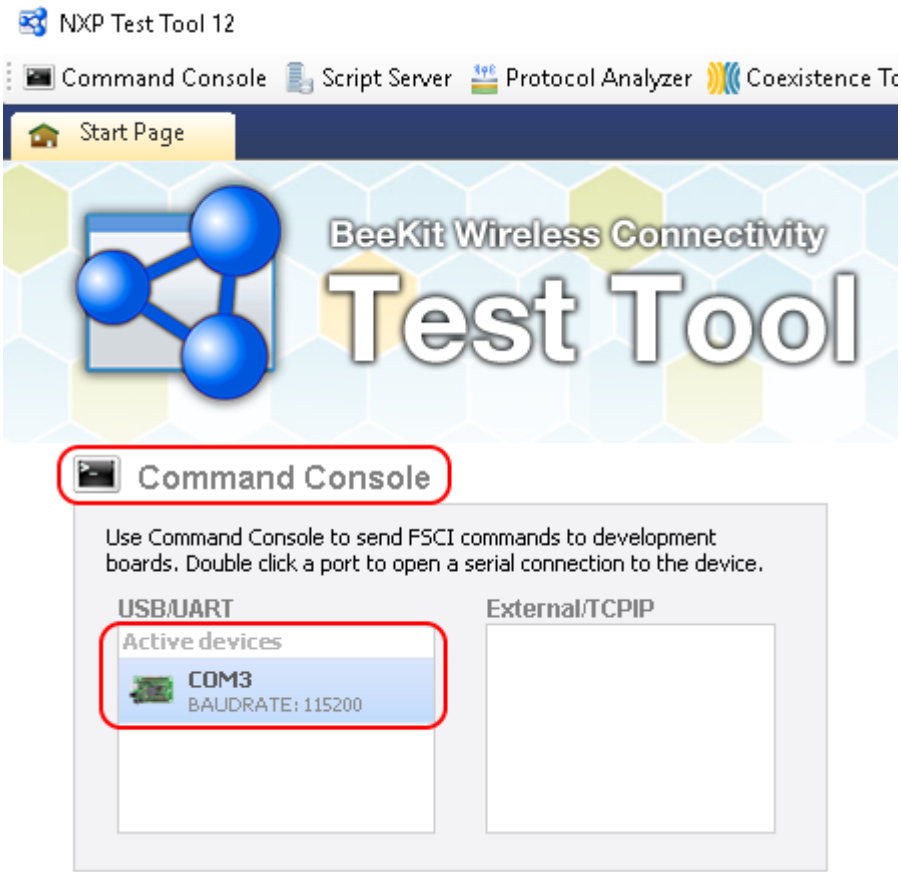
- MCX-W72-EVK
- FRDM-MCXW72

Parent topic:FSCI Bridge

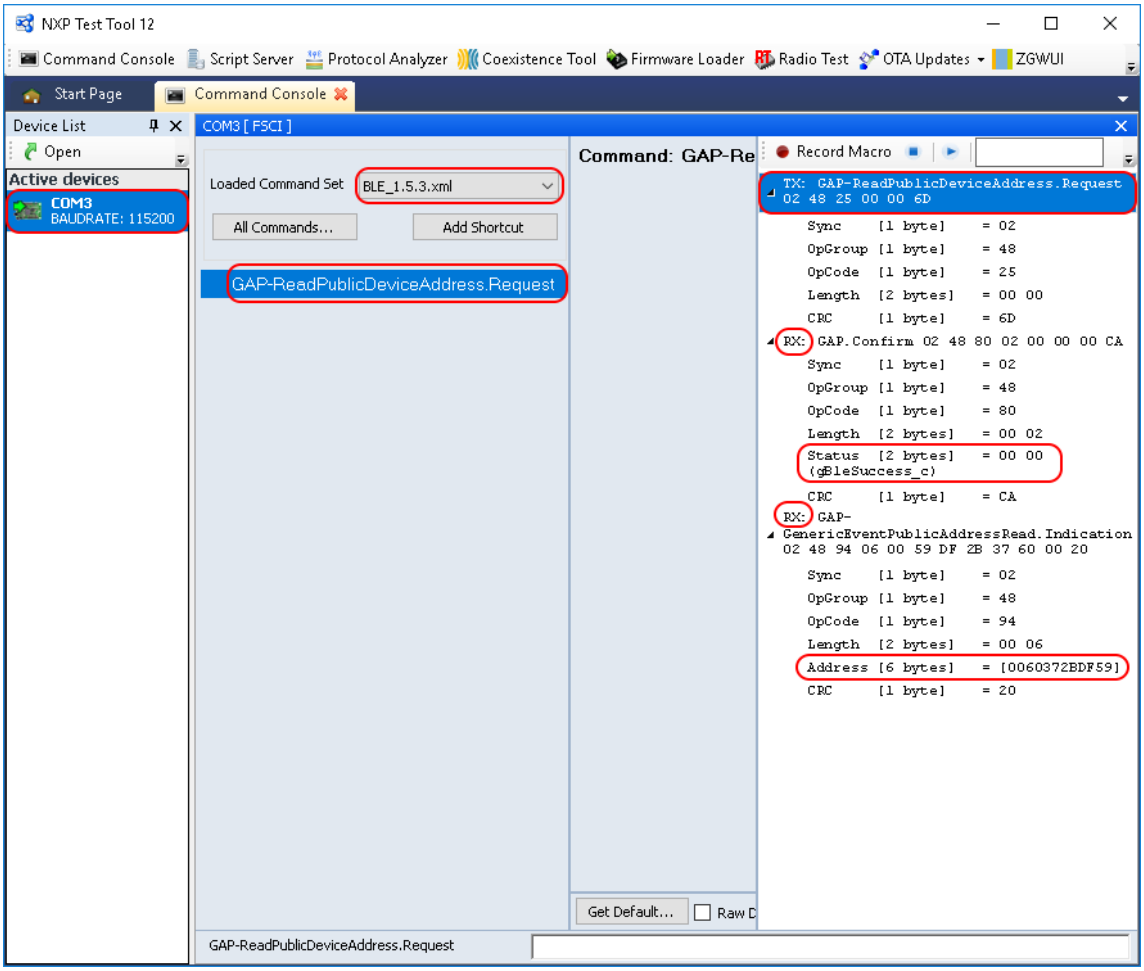
Usage with Test Tool for connectivity products

The FSCI Bridge demo application is designed to be used via serial interface. This can be done using the TEST Tool for Connectivity Products – Command Console application as described below.

1. Download the FSCI Bridge application onto a supported board's Application core, together with the NCP FSCI Black Box application on the NBU core.
2. Connect the board to a USB port of the PC. The USB COM port drivers must be installed properly and a COM port corresponding to the board should be available.
3. Open the Test Tool application and connect to the serial port corresponding to the board on which the FSCI Bridge application runs. See Figure. The serial communication parameters are: baud rate 115200, 8N1, and no flow control.



4. Select the appropriate Test Tool XML file from the drop-down list for the release being used and send commands to the application. An example is shown in Figure.



Parent topic:FSCI Bridge

Parent topic:*Bluetooth LE stack and demo applications*

HCI Black Box

This section describes the functionality, user interactions, and testing methods for the Bluetooth LE HCI Black Box demo application.

Description

The Bluetooth LE HCI Black Box demo application gives access to the Bluetooth LE Controller via a serial interface using the HCI protocol over serial interface. Refer to the Bluetooth Specification for the format of HCI commands and events over serial interface and the full list of supported commands and events.

The demo can be used with the Test Tool for Connectivity Products. The Command Console application can be downloaded from the NXP website. Alternatively you can also download it using a custom application that supports the HCI protocol and commands and events over serial interface.

Note: The HCI protocol encapsulation is dependent on the type of interface it is being used on. See the Bluetooth Specification for the HCI message format on each type of supported interface.

For instructions using the Bluetooth LE HCI Black Box with a serial terminal application or the hcitool in Linux, check the article "FRDM-KW40Z Bluetooth LE Controller Usage with the Linux hcitool".

Parent topic:HCI Black Box

Supported platforms

The following platforms support the HCI Black Box application:

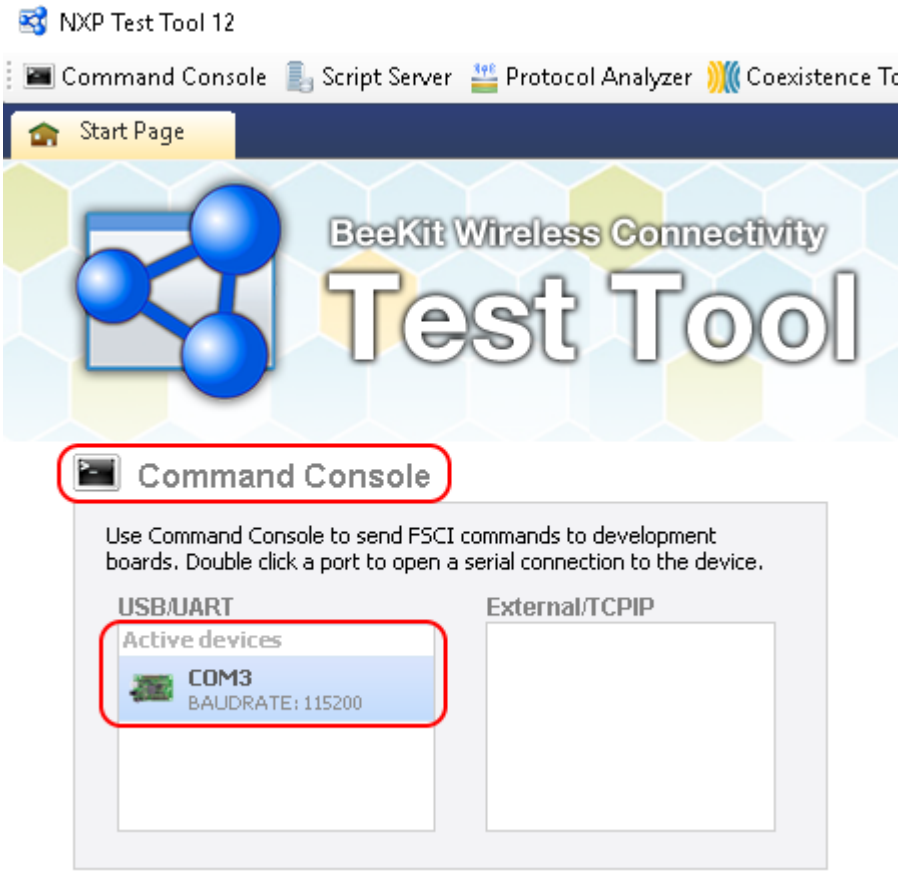
- KW45B41Z-EVK
- FRDM-MCXW71
- MCX-W71-EVK
- KW47-EVK
- FRDM-MCXW72
- MCX-W72-EVK
- FRDM-KW43

Parent topic:HCI Black Box

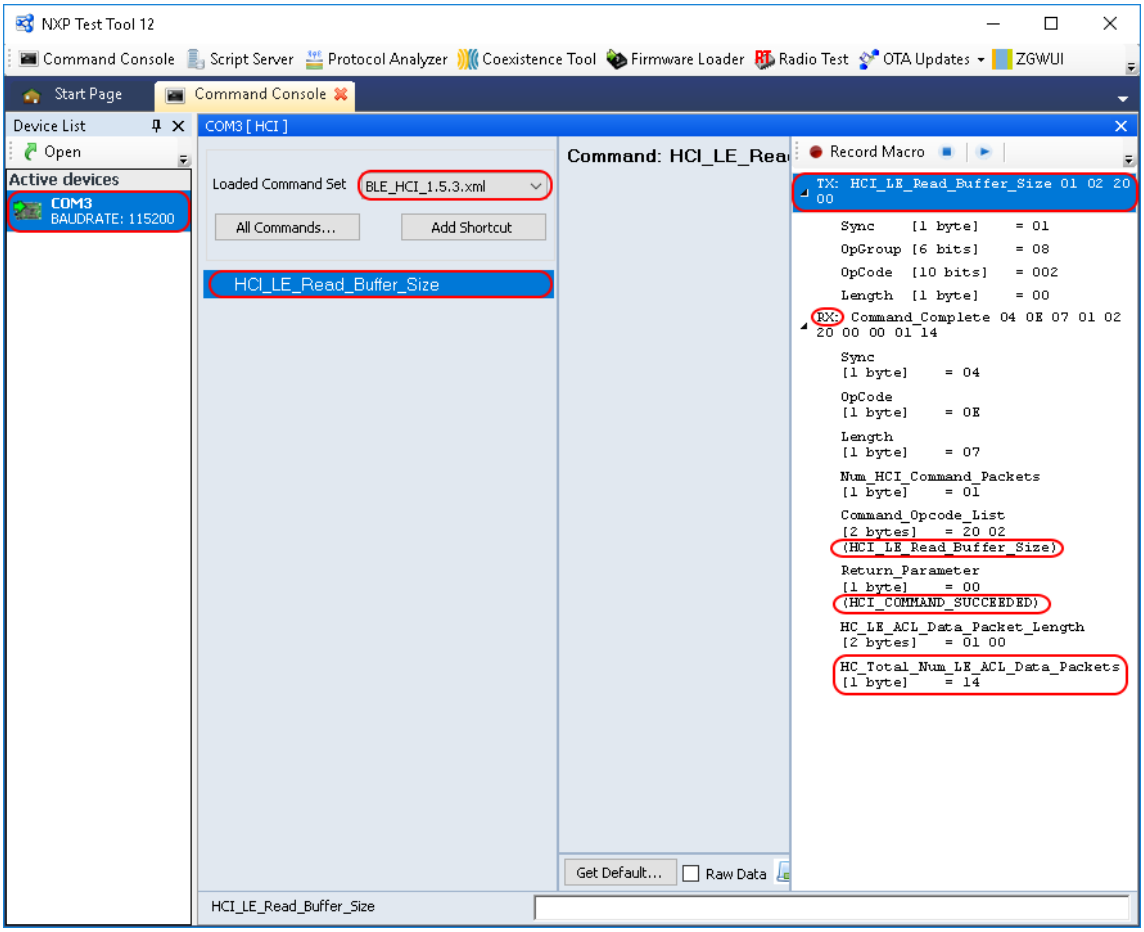
Usage with Test Tool for Connectivity products

The Bluetooth LE HCI Black Box demo application is designed to be used via serial interface. This can be achieved using the TEST Tool for Connectivity Products – Command Console application as described below.

1. Download the demo application to a supported board.
2. Connect the board to a USB port of the PC. The UASB COM port drivers must be installed properly and a COM port corresponding to the board should be available.
3. Open the Test Tool application and connect to the serial port corresponding to the board on which the Bluetooth LE HCI Black Box application runs. The serial communication parameters are: baud rate 115200, 8N1, and no flow control. See the figure **Test tool command console serial port selection** below.



4. Select the appropriate Test Tool HCI XML file from the drop-down list for the release you are using. Send a few commands to the application. An example is shown in the figure **HCI black box command example** below.



Parent topic:HCI Black Box

Parent topic:*Bluetooth LE stack and demo applications*

HID Device (Mouse)

This section describes implemented profiles and services, user interactions, and testing methods for the HID mouse application.

Implemented profiles and services

The HID Device application implements a GATT server and the following profile and services.

- HID over GATT Profile v1.0
- Human Interface Device Service v1.0
- Battery Service v1.0
- Device Information Service v1.1

The application behaves as a GAP peripheral node. It enters GAP General Discoverable Mode and waits for a GAP central node to connect. Security on the services and bonding is enabled on this device.

When the GATT client configures notification, the application starts sending HID reports every two seconds with the movement of the MOUSE_STEP. The demo moves the cursor in a square pattern between AXIS_MIN and AXIS_MAX. The report contains 3 bytes, one for button status, one for X axis, and one for Y axis. The report descriptor matches the example in chapter E.10 from

the USB Device Class Definition for Human Interface Devices (USB HID Specification), Version 1.11.

Parent topic:HID Device (Mouse)

Supported platforms

The following platforms support the HID Device application:

- KW45B41Z-EVK
- FRDM-MCXW71
- MCX-W71-EVK
- KW47-EVK
- FRDM-MCXW72
- MCX-W72-EVK
- FRDM-KW43

Parent topic:HID Device (Mouse)

User interface

After flashing the board, the device enters Idle mode with all LEDs flashing. To start advertising, press the **ADVSW** button. In GAP Discoverable mode, **CONNLED** is flashing. When the central node connects to the peripheral, **CONNLED** turns solid. To disconnect the node, hold the **ADVSW** pressed for 2-3 seconds. The node then re-enters GAP Discoverable Mode.

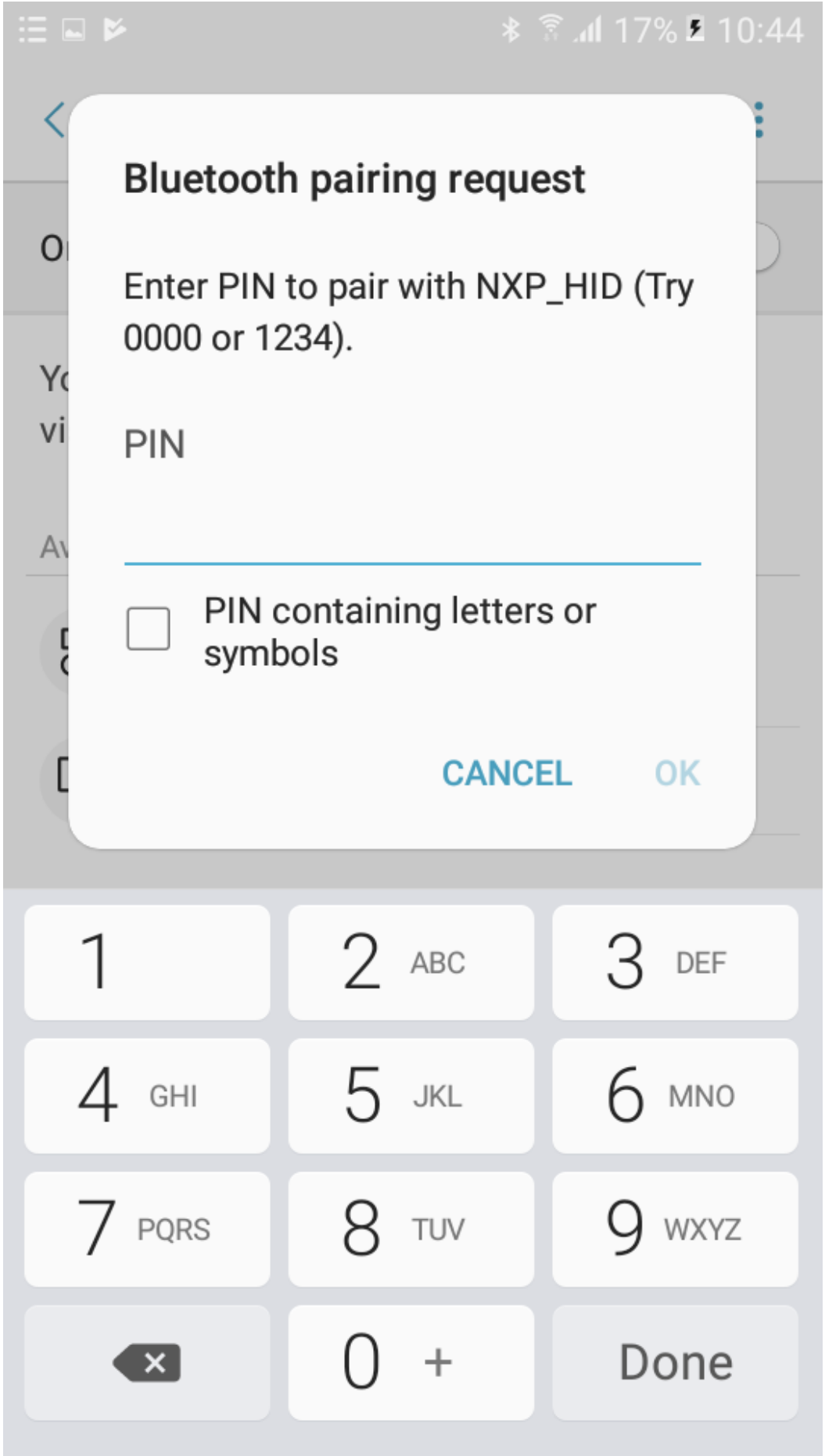
Parent topic:HID Device (Mouse)

Switches and pins

Usage

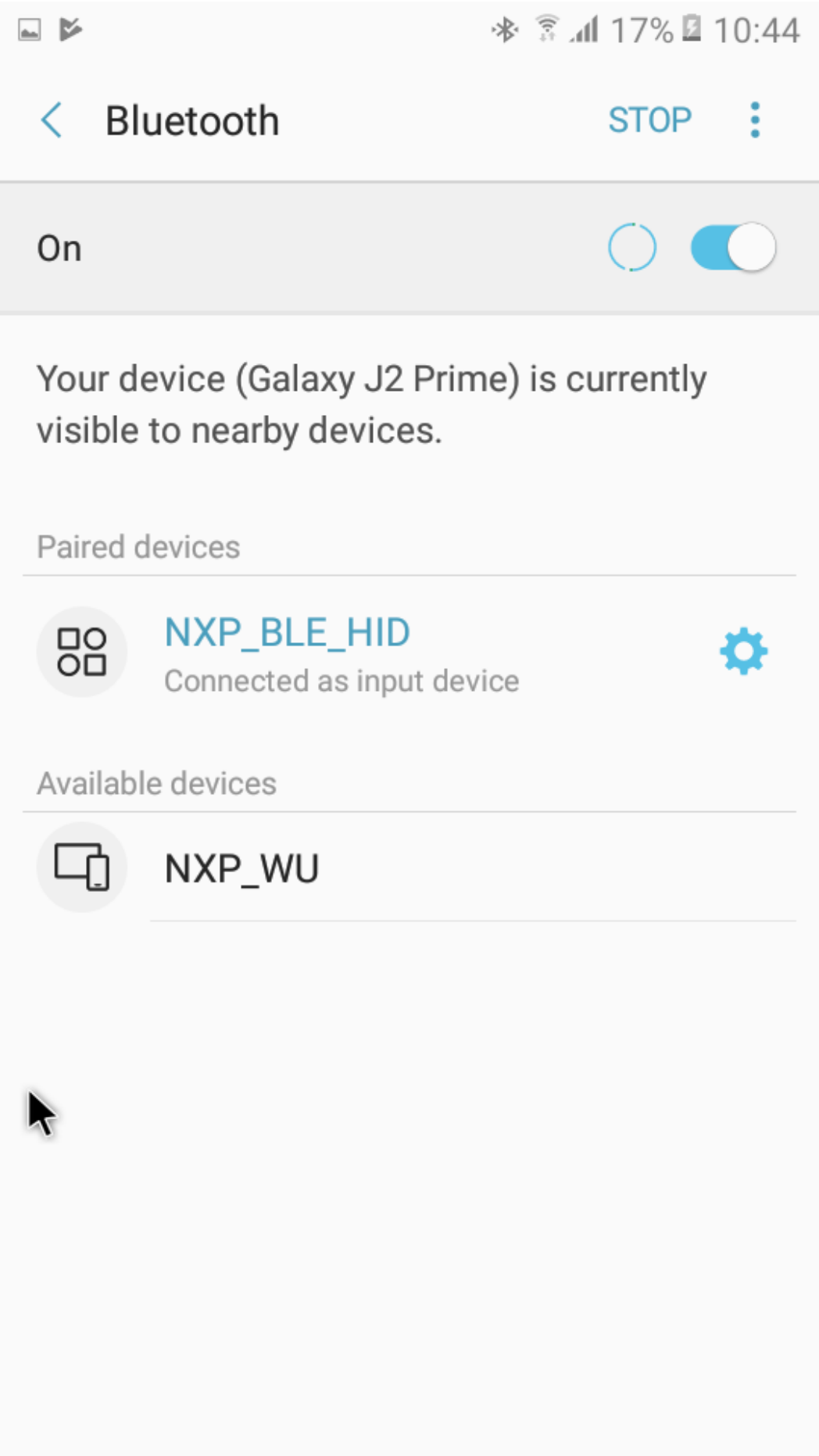
The HID mouse can be connected to any Bluetooth Smart Ready products available on the market that supports HID devices or to another supported platform running the HID Host example (setup steps detailed in the HID Host section).

To make the HID mouse visible, press the **ADVSW** button to start sending advertisements, which causes CONNLED to start flashing. See the figure **Enter PIN prompt on Android platform** below.


17% 10:44
Bluetooth pairing request
Enter PIN to pair with NXP_HID (Try 0000 or 1234).
PIN
PIN containing letters or symbols
CANCEL
OK
1
2 ABC
3 DEF
4 GHI
5 JKL
6 MNO
7 PQRS
8 TUV
9 WXYZ
0 +
Done

The sensor name “NXP_HID” shows on the device when its scanning is active. A solid CONNLED indicates a successful connection between the 2 devices. When prompted to enter the pin, type the 999999 passkey.

When configured, the HID mouse starts sending HID report, which is configured as explained above, with notifications every 100 milliseconds. The mouse cursor shows a square pattern movement on the screen as shown in the figure **HID Mouse detected by Android platform** below.


17% 10:44
Bluetooth
STOP
On
Your device (Galaxy J2 Prime) is currently visible to nearby devices.
Paired devices
NXP_BLE_HID
Connected as input device
Available devices
NXP_WU

Parent topic:HID Device (Mouse)

Parent topic:*Bluetooth LE stack and demo applications*

HID Host

This section presents the implemented profiles and services, user interactions, and testing methods for the HID Host application.

Implemented profiles and services

The HID Host application implements a GATT client or server for the following profile and service.

- HID over GATT Profile v1.0
- Battery Service v1.0
- Device Information Service v1.1

The application behaves as a GAP central node. It enters the GAP Limited Discovery Procedure and searches for HID devices to connect to. After connecting with the peripheral, it configures notifications and displays the received HID reports on a terminal connected to the UART port. The application uses pairing with bonding by default. When connected with the HID Device application, it sends the 999999 passcode to the host stack by default.

Parent topic:HID Host

Supported platforms

The following platforms support the HID Host application:

- KW45B41Z-EVK
- FRDM-MCXW71
- MCX-W71-EVK
- KW47-EVK
- FRDM-MCXW72
- MCX-W72-EVK
- FRDM-KW43

Parent topic:HID Host

User interface

After flashing the board, the device is in idle mode (all LEDs flashing). To start scanning, press the **SCANSW** button. When in GAP Limited Discovery Procedure, **CONNLED** is flashing. When the central node connects to the peripheral, **CONNLED** turns solid. To disconnect the node, hold the **SCANSW** button pressed for 2-3 seconds. The node then re-enters GAP Limited Discovery Procedure.

Parent topic:HID Host

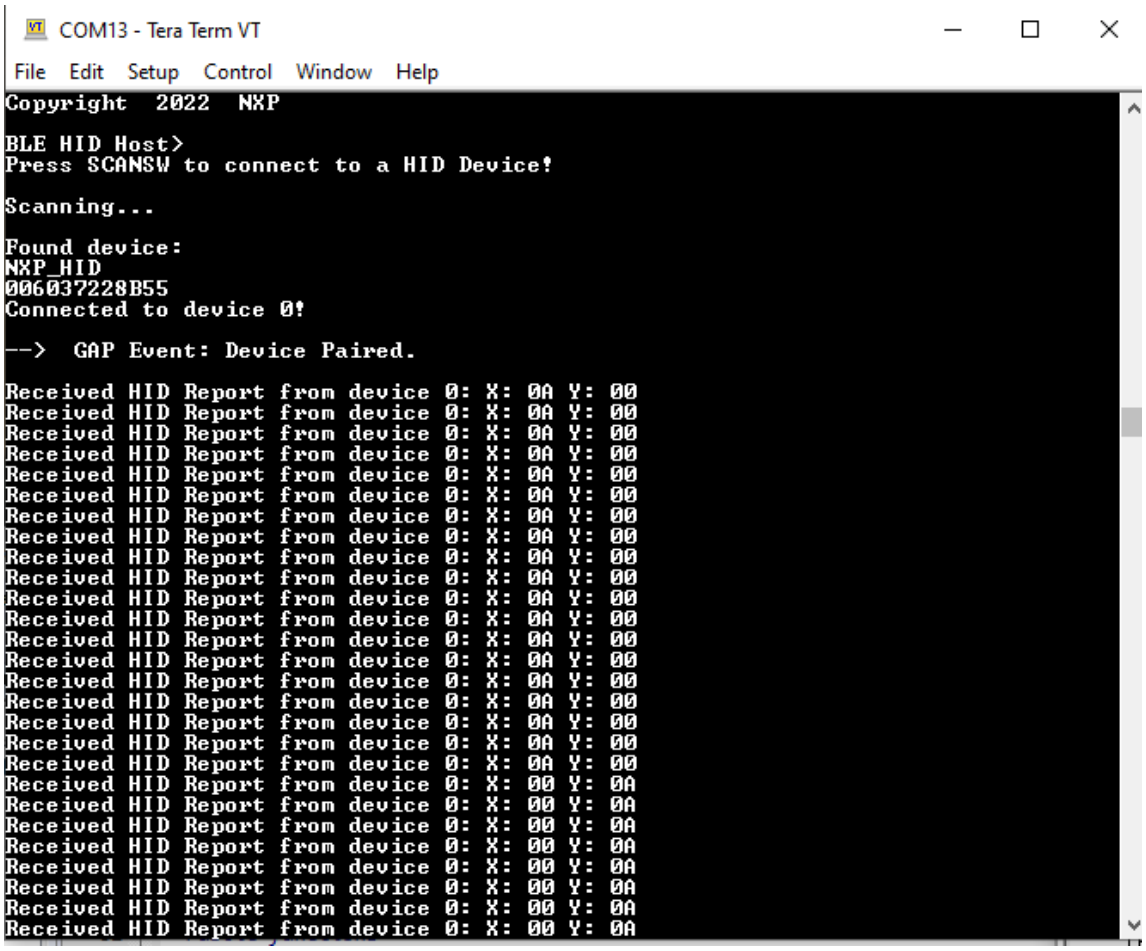
Switches and pins

Usage

The application is built to work only with the HID Device application presented in HID Device (Mouse). It supports up to 2 peripherals connected at the same time.

1. Open a serial port terminal and connect it to board, in the same manner described in Testing devices. The start screen is displayed after the board is reset.

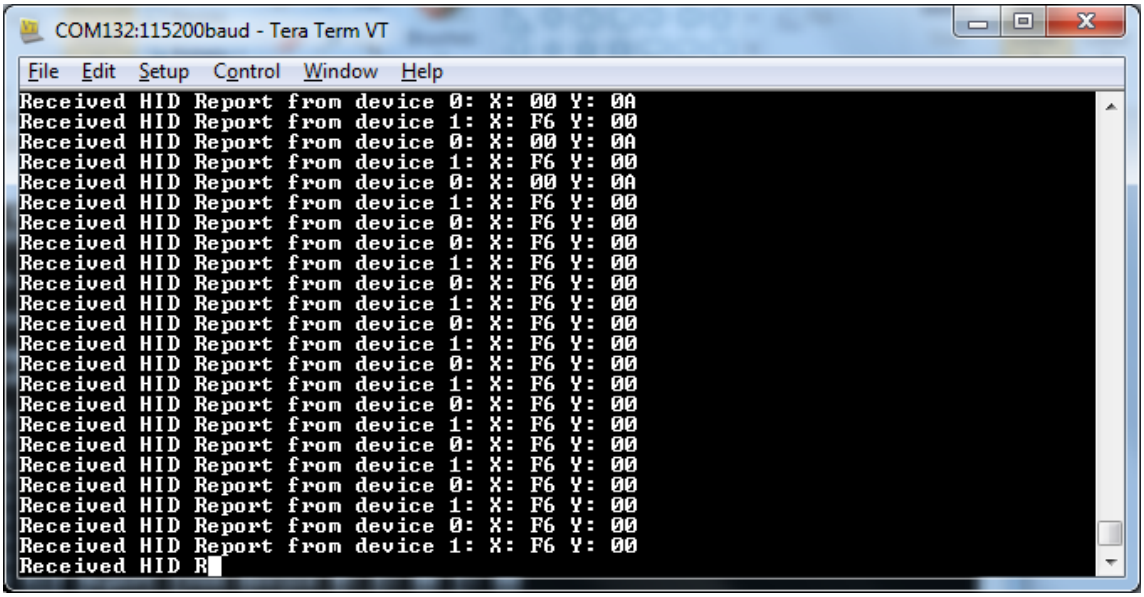
2. To start scanning for devices, press the **SCANSW** button on the HID Host board. To make it enter discoverable mode, perform the same step on the HID device board. The host connects with the board after it sees it advertise the HID service, connects to it, and configures report notifications. The device then starts sending HID reports, as shown in Figure.



HID host

3. To connect a second HID device, press again the **SCANSW** button on the HID Host board to start scanning for devices. Do the same on the second HID device board to make it enter discoverable mode. The host connects with the board after it sees it advertise the HID service, connects to it, and configures report notifications. The device then starts sending HID reports. The console displays reports from both devices, as shown in Figure.

Tera Term – Output Console on HID Host with 2 peripherals connected



Parent topic:HID Host

Parent topic:*Bluetooth LE stack and demo applications*

Localization Reader

This section describes the functionality of the Localization Reader demo application.

Please check out the *Localization Reader* section in the Bluetooth Low Energy Localization Application Note.

Parent topic:*Bluetooth LE stack and demo applications*

Localization Reader Host

This section describes the implemented profiles and services, user interactions, and testing methods for the Localization Reader Host application.

Description

The Loc Reader Host application is a Bluetooth Low Energy (BLE) localization application that runs on the application core of dual-core NXP platforms. It works in conjunction with the ncp_loc_reader application running on the radio core to perform distance measurement and localization using Channel Sounding (CS) technology.

Dual-Core Architecture

The application is part of a dual-core system where:

- **Core 0 (Application Core)**: Runs loc_reader_host
  - Receives uncompress IQ measurement data from Core 1 via FSCI messages
  - Executes localization algorithms (RADE, CDE)
  - Displays results via UART shell
- **Core 1 (Radio Core)**: Runs ncp_loc_reader
  - Handles BLE stack and Channel Sounding procedures
  - Collects and transmits uncompress local/remote IQ data to Core 0

Inter-Core Communication

Cores communicate using FSCI messages with event types:

- **gIQLocalTrStart_c/Cont_c/End_c**: Local uncompress IQ data fragments
- **gIQRemoteTrStart_c/Cont_c/End_c**: Remote uncompress IQ data fragments

Each message includes device ID, event type, and payload data (up to gFsciMaxPayloadLen_c).

Data Flow

1. Core 1 performs CS procedure and collects uncompress IQ samples
2. Core 1 fragments and transmits data to Core 0 via FSCI
3. Core 0 reassembles fragments into csAppData_t buffers
4. Core 0 runs localization algorithms and displays results

Parent topic: Localization Reader Host

Supported platforms

The following platforms support the Localization Reader Host application:

- MCX-W72-EVK
- FRDM-MCXW72

Parent topic: Localization Reader Host

User interface

Ensure that both NCP Localization Reader (for the NBU core) and Localization Reader Host (for the Application core) have been flashed to the board.

The application can be controlled using a shell interface via the UART connection to the host computer.

The behavior of the application and the user interraction is exactly the same as *Localization Reader* application.

Parent topic: Wireless UART Host

Parent topic:*Bluetooth LE stack and demo applications*

Localization User Device

This section describes the functionality of the Localization User Device demo application.

Please check out the *Localization User Device* section in the Bluetooth Low Energy Localization Application Note.

Parent topic:*Bluetooth LE stack and demo applications*

Low-power temperature sensor and collector

This section describes the implemented profiles and services, user interactions, and testing methods for the temperature sensor application.

Implemented profiles and services

The Temperature Sensor application implements a GATT server, a custom profile and the following services.

- Temperature Service (UUID: 01ff0200-ba5e-f4ee-5ca1-eb1e5e4b1ce0)

- Battery Service v1.0
- Device Information Service v1.1

The application behaves as a GAP peripheral node. It enters GAP General Discoverable Mode and waits for a GAP central node to connect and configure notifications for the temperature value.

The Temperature service is a custom service that implements the Temperature characteristic (UUID: 0x2A6E) with a Characteristic Presentation Format descriptor (UUID: 0x2904), both defined by the Bluetooth SIG.

The Temperature Collector application implements a GATT client or server for the following profile and services.

- Temperature Service (UUID: 01ff0200-ba5e-f4ee-5ca1-eb1e5e4b1ce0)
- Battery Service v1.0
- Device Information Service v1.1

The application behaves as a GAP central node. It enters GAP Limited Discovery Procedure and searches for sensor devices to pair with. After pairing with the peripheral, it configures notifications and displays temperature values on a terminal connected to the UART port.

Both application uses pairing with bonding by default. When connected with the Low-Power Temperature Sensor application, the collector sends the 999999 passcode to the host stack by default.

Parent topic: Low-power temperature sensor and collector

Supported platforms

The Temperature Sensor and Collector applications are supported by the following platforms:

- KW45B41Z-EVK
- FRDM-MCXW71
- MCX-W71-EVK
- KW47-EVK
- FRDM-MCXW72
- MCX-W72-EVK
- FRDM-KW43

Parent topic: Low-power temperature sensor and collector

User interface

After flashing the board, both nodes enter Low-power mode. In case the sensor is put in deep sleep, press **WAKESW** or **RESET**. To flash the board in case the sensor is put in deep sleep, press either **WAKESW** or **RESET** button. By default, the application is configured to be in low power mode, which disables LED support.

The user can manually change this configuration and enable LED support, else all subsequent LED behavior references are ignored and all LEDs are off. The devices disconnect and enter Deep-sleep only if low power is enabled. When the node is awake and communicating, **CONNLED** is on. To wake up the node, press the **WAKESW** button.

Parent topic: Low-power temperature sensor and collector

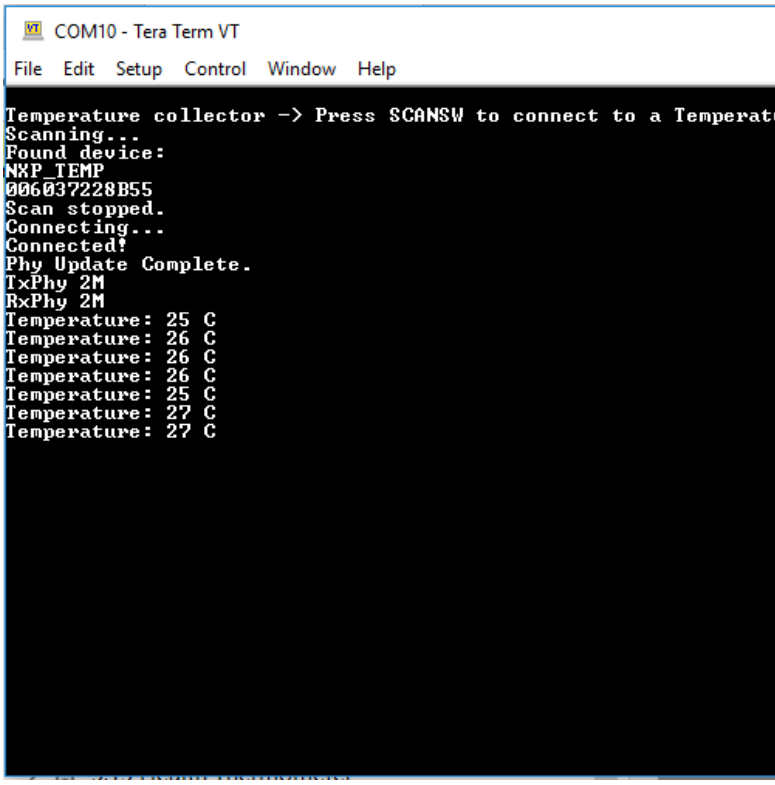
Switches and pins(Collector) Switches and pins(Sensor)

Usage

The setup requires two supported platforms, one for the temperature sensor and one for the temperature collector.

1. Open a serial port terminal and connect it to the temperature collector board, in the same manner as described in Testing devices. The start screen is displayed after the board is reset. At first the LEDs are off on both devices.
2. To start advertising on the sensor, press the WAKESW button and CONNLED lights up. The sensor enters into the Deep-sleep mode, which means that the MCU wakes up on any packet from the Link layer, in this case the connect request. If no connection is established in an interval of 30 seconds, the sensor stops advertising and enters into the Deep-sleep mode again. CONNLED turns off.
3. To start scanning on the collector, press the WAKESW button and CONNLED lights up. The device wakes up, enters into the Deep-sleep mode, scans, and connects to a compatible sensor device. If no connection is established within 30 seconds, the collector stops scanning and enters Deep-sleep mode again. CONNLED turns off.
4. If the collector connects to a sensor node, it bonds (if no bond was previously made), does service discovery (only the first time it connects with the sensor), and configures notification and waits for notifications from the sensor for 5 seconds. If no data is sent, the node disconnects and re-enters Deep-sleep mode. The sensor exits low power and sends a notification with the value of the temperature read through an ADC from the thermistor, if present, or random generated if not.

   Once the connection is established, the PHY is automatically updated to 2M, if both the sensor and the collector support this feature as shown in the figure below. The PHY update is configurable from the application.



Output Console on Temperature Collector

5. Subsequent key pressing triggers other notifications for the collector. If no key is pressed in an interval of 5 seconds, the sensor node disconnects and re-enters Deep-sleep mode.

Parent topic: Low-power temperature sensor and collector

Parent topic:*Bluetooth LE stack and demo applications*

Tire Pressure Monitoring System Sensor and Monitor

This section describes the implemented profiles and services, user interactions, and testing methods for the TPMS Sensor and TPMS Monitor applications.

Implemented profiles and services

The TPMS Sensor application implements a GATT server, the TPMS profile and the following services.

- Tire Pressure Monitoring System Service (TPMS)
- Battery Service v1.0
- Device Information Service v1.1

The application behaves as a GAP peripheral node. It enters GAP Limited Discoverable Mode to perform either connectable or non-connectable advertising. When performing connectable advertising, it waits for a Primary Monitor to connect and configure it, as defined by the TPMS Profile. When performing non-connectable advertising, it broadcasts packets containing the tire pressure and temperature values.

The TPMS Service is defined by the Bluetooth SIG.

The TPMS Monitor application implements a GATT client or server for the following profile and services.

- Tire Pressure Monitoring System Service (TPMS)
- Battery Service v1.0
- Device Information Service v1.1

The application behaves as a GAP central node. It enters GAP Limited Discovery Procedure and searches for connectable or non-connectable advertising packets from TPMS Sensors. When a TPMS Sensor is found, the Monitor can connect to it, pair, bond, and read/write TPMS characteristics as defined by the TPMS Profile. The Monitor can also receive non-connectable broadcast packets containing tire pressure and temperature values from sensors.

Parent topic:TPMS Sensor and Monitor

Supported platforms

The TPMS Sensor and Monitor applications are supported by the following platforms:

- KW45B41Z-EVK
- KW47-EVK
- FRDM-KW43

Parent topic:TPMS Sensor and Monitor

User interface

After flashing, both devices are in idle mode. To start scanning on the TPMS Monitor, press the **SCANSW** button. To start advertising on the TPMS Sensor, press either the **CONNADVSW** or **NCONNADVSW** buttons. When in GAP Limited Discovery Mode or scanning mode, **CONNLED** is flashing on either device. When the devices connect, **CONNLED** turns solid. To disconnect the TPMS Monitor, hold the **SCANSW** button pressed for 2-3 seconds.

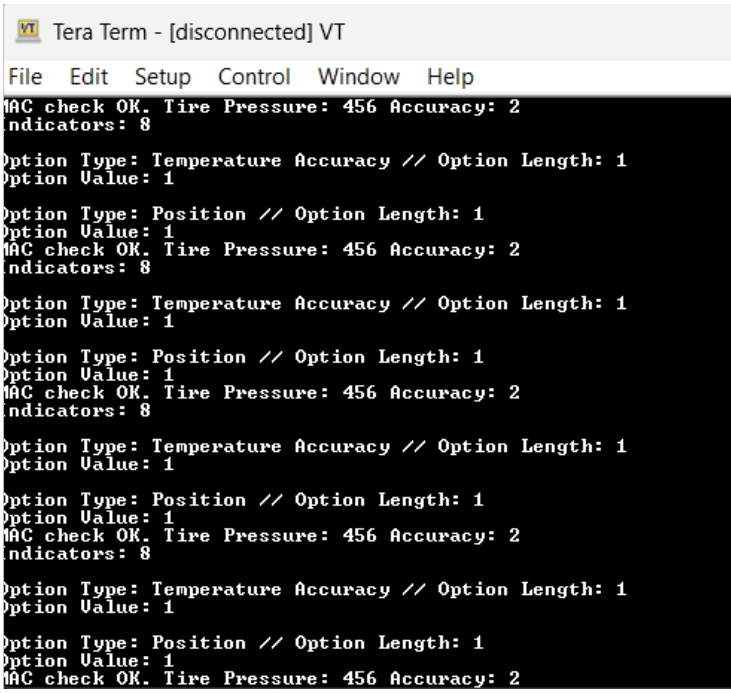
Parent topic:TPMS Sensor and Monitor

Switches and pins(Monitor) Switches and pins(Sensor)

Usage

The setup requires two supported platforms, one for the TPMS Sensor application and one for the TPMS Monitor application.

1. Open a serial port terminal and connect it to the TPMS Monitor board, in the same manner as described in Testing devices. The start screen is displayed after the board is reset.
2. To start connectable advertising on the TPMS Sensor, press the CONNADVSW button. The device will start advertising.
3. To start scanning on the TPMS Monitor, press the SCANSW button.
4. If the TPMS Monitor connects to a sensor node, it will:
    1. perform service discovery
    2. pair and bond to establish itself as a Primary Monitor
        1. if the Sensor already has a Primary Monitor, bonding will not be performed and the Monitor will have the role of Secondary Monitor
        2. The following steps assume the role is that of Primary Monitor
    3. read the TPMS Properties, TPMS Signing Key, TPMS Tire Pressure, TPMS Tire Temperature and TPMS Tire Acceleration characteristic values
    4. update the TPMS Duty Cycle by writing to the corresponding characteristic
    5. enable indications for the TPMS Signing Key characteristic
    6. enable notifications for the TPMS Tire Pressure and TPMS Tire Temperature characteristics
5. While in connection, pressing CONNADVSW on the TPMS Sensor will notify the Tire Pressure and Temperature to the Monitor, while pressing NCONNADVSW will update the Signing Key and indicate this to the Monitor.
6. At this point, the connection can be terminated by either disconnecting via long press of the SCANSW on the TPMS Monitor or by resetting the boards.
7. To start non-connectable advertising on the TPMS Sensor, press the NCONNADVSW button. The device will start advertising in non-connectable mode (if no Primary Monitor exists, the advertising will be connectable).
8. To start scanning on the TPMS Monitor, press the SCANSW button. The TPMS Monitor will display the TPMS data in the non-connectable advertising of the TPMS Sensor if it successfully authenticates the payload as described by the TPMS Profile specification.
9. To factory reset the Sensor, long press NCONNADVSW. For the Monitor, long press FACTORYRESETSW.



Output Console on TPMS Monitor

Parent topic:TPMS Sensor and Monitor

Parent topic:*Bluetooth LE stack and demo applications*

Low-power extended advertising Peripheral and Central

This section describes the implemented profiles and services, user interactions, and testing methods for the adv_ext_peripheral and adv_ext_central applications.

Implemented profile and services

The adv_ext_peripheral application implements a GATT server, a custom profile and the following services.

- Temperature Service (UUID: 01ff0200-ba5e-f4ee-5ca1-eb1e5e4b1ce0)
- Battery Service v1.0
- Device Information Service v1.1

The application behaves as a GAP peripheral node. It enters GAP General Discoverable Mode and waits for a GAP central node to connect and configure notifications for the temperature value.

The Temperature service is a custom service that implements the Temperature characteristic (UUID: 0x2A6E) with a Characteristic Presentation Format descriptor (UUID: 0x2904), both defined by the Bluetooth SIG.

The adv_ext_central application implements a GATT client or server for the following profile and services.

- Temperature Service (UUID: 01ff0200-ba5e-f4ee-5ca1-eb1e5e4b1ce0)
- Battery Service v1.0
- Device Information Service v1.1

The application behaves as a GAP central node. It enters GAP Limited Discovery Procedure and searches for peripherals devices to pair with. After pairing with the peripheral, it configures notifications and displays temperature values on a terminal connected to the UART port.

Both applications use pairing with bonding by default. When connected with the Low-Power Extended Advertising Peripheral application, the Extended Advertising Central application sends the 999999 passcode to the host stack by default.

To enable Periodic Advertising With Responses (PAWR), set the gAppPAWRSupport_d define 1 in app_preinclude.h.

In case PAWR is enabled and more than one ext_adv_centrals are to be synced with the PAWR train, a different response slot may be configured for each board using the gUserDefinedResponseSlot_c define in app_preinclude.h to avoid collisions. Otherwise, a random response slot will be chosen by the application after reset.

To enable Encrypted Advertising Data (EAD), set the gAppEADSupport_d define 1 in app_preinclude.h.

In case EAD is enabled, the advertising structures of gAdManufacturerSpecificData_c data type will be encrypted and transmitted over the air as gAdEncryptedAdvertisingData_c data type structures. On the receiver side, they will be decrypted and printed under the "Decrypted data :" text to mark them as EAD. This applies for extended advertising, periodic advertising, PAWR subevent data or responses.

Parent topic: Low-power extended advertising Peripheral and Central

Supported platforms

The following platforms support Extended Advertising Peripheral and Central applications:

- KW45B41Z-EVK
- FRDM-MCXW71
- MCX-W71-EVK
- KW47-EVK
- FRDM-MCXW72
- MCX-W72-EVK
- FRDM-KW43

Deep-sleep mode is used by default.

Parent topic: Low-power extended advertising Peripheral and Central

User interface

After flashing the board, both nodes enter Deep-sleep mode. To flash the board again, press **WAKESW**. The application default configuration enables low power that disables LED support. The user disables low power and enables LED support setting the gAppLowpowerEnabled_d define to 0. To wake up the node, press the **WAKESW** button. Both applications provide guidance over the UART.

Open a serial port terminal using the following settings:

baud rate 115200, data bits 8, parity none, stop bits 1.

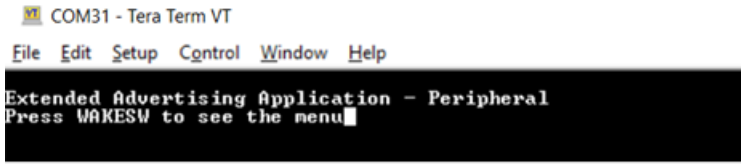
Parent topic: Low-power extended advertising Peripheral and Central

Switches and pins(Central) Switches and pins(Peripheral)

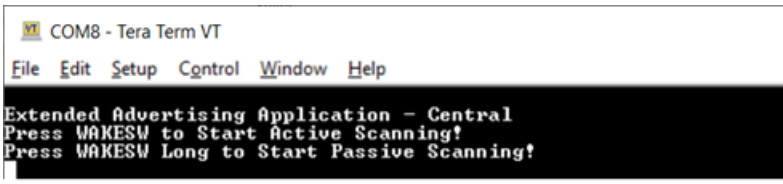
Usage

The setup requires two supported platforms, one for the adv_ext_peripheral, and one for the adv_ext_central.

1. Open a serial port terminal and connect it to each platform with the settings provided in the previous paragraph. The start screen is displayed after the boards are reset as shown in the figures below.

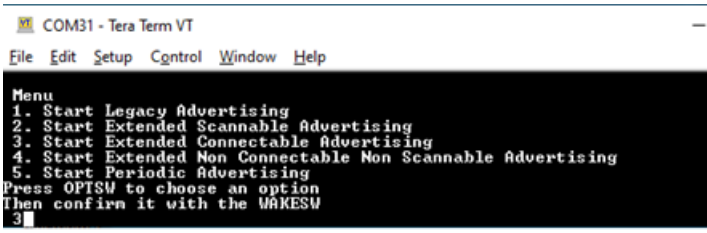


Advertizing external Peripheral start screen



Advertizing external Central start screen

2. On the board that implements the adv_ext_peripheral application, press the **WAKESW** button. The board exits Deep-sleep mode and displays the menu as shown in the figure below.



Choosing a menu option on adv_ext_peripheral

Use the **OPTSW** to choose an option. The option printed on the bottom changes every time the switch is pressed. When the option matches your intention (For example, 3 Starts Extended Connectable Advertising), press the **WAKESW** again to make a decision. The advertising type chosen is started and the board starts entering low-power between advertising events.

Next time, the **WAKESW** is pressed, an updated menu is printed (For example, at option 3 Stop Extended Connectable Advertising). There is no timeout for advertising. The board continues advertising until it is stopped, or a connection is established (for legacy and extended connectable advertising only) with an adv_ext_central device.

The connection is terminated five seconds after the central device configures notifications for the temperature value. When all advertising is off and all connections are terminated, the board enters low-power mode until the **WAKESW** button is pressed again.

When gAppLowpowerEnabled_d is set 0, LEDs are enabled. The ADVLED flashes whenever an advertising starts and is ON otherwise. The CONNLED flashes whenever there is a connection under way and is ON otherwise.

When PAWR is started , advertising data is transmitted on subevents zero and three and responses are expected on all 5 configured response slots of these subevents. The central application responds to the periodic data received with a six bytes data composed of a three bytes random number followed by a three bytes hash performed over the random number using the local IRK in the same fashion the RPA are generated. The advertiser prints the responses, uses the bonded devices IRKs to generate the hash over the random number and attempts to connect over PAWR with the responder in case the hashes match.

```
PAWR Started
PAWR Set Subevent Data Complete
Periodic advertising response received
Adv Handle: 0
Subevent: 0
Response slot: 2
Response data: EFBE6E573231
```

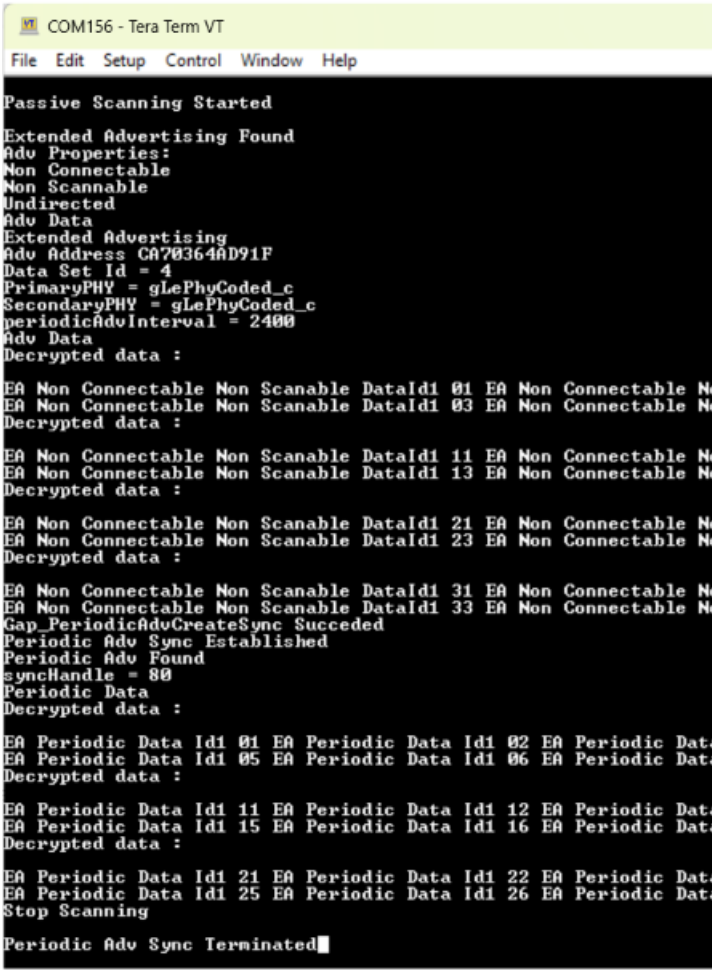
(continues on next page)

(continued from previous page)

```
Periodic advertising response received
Adv Handle: 0
Subevent: 3
Response slot: 2
Response data: 021CED6395B8
PAWR Set Subevent Data Complete
```

3. On the board that implements the **adv_ext_central** application, there are two options: Press **WAKESW** to start active scanning or long press **WAKESW** to start passive scanning. If catching extended scannable advertising is not an option, choose passive scanning. Otherwise, select active scanning. The device wakes up, starts scanning, and enters Deep-sleep mode. The scanning ends when the 60 seconds timeout is reached or when a connection with an adv_ext_peripheral device is established.

   During scanning, all advertisements caught from adv_ext_peripheral devices are displayed on the terminal window as shown in Figure. When an extended non-connectable, non-scannable advertising with a periodic advertising attached is detected, the *adv_ext_central*device attempts to sync with the periodic advertising train and prints the periodic advertising data on the terminal window.



Advertising caught on adv_ext_central console

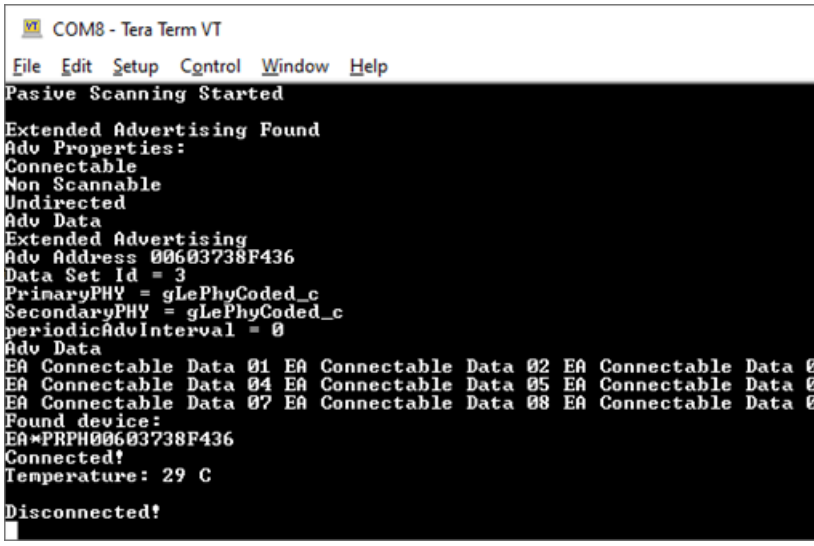
If the PAWR support is enabled and the periodic advertising train the adv_ext_central has synced to is with responses the application attempts to sync with the subevents zero and three, periodic advertising data printed being slightly different. Also, the application responds to the periodic data received with a six bytes data composed of a three bytes random number followed by a three bytes hash performed over the random number using the local IRK in the same fashion the RPA are generated. The advertiser uses the bonded devices IRKs to generate the hash over the random number and attempts to connect over

PAWR with the responder in case the hashes match.

```
Passive Scanning Started

Extended Advertising Found
Adv Properties:
Non Connectable
Non Scannable
Undirected
Adv Data
Extended Advertising
Adv Address C4603770BCC5
Data Set Id = 4
PrimaryPHY = gLePhyCoded_c
SecondaryPHY = gLePhyCoded_c
periodicAdvInterval = 2400
Adv Data
EA Non Connectable Non Scanable DataId1 01 EA Non Connectable Non Scanable DataId1 02
EA Non Connectable Non Scanable DataId1 03 EA Non Connectable Non Scanable DataId1 04
Gap_PeriodicAdvCreateSync Succeded
Periodic Adv Sync Established
Gap_SetPeriodicSyncSubevent Succeded
PAWR received
Sync Handle: 0050
Event Counter: 0007
Subevent: 0
Subevent data:
PAWR Data0 For Subevent 0
PAWR Data1 For Subevent 0
Gap_SetPeriodicAdvResponseData Succeded
Periodic Adv Set Response Data Complete
```

4. If the **adv_ext_central** connects to an **adv_ext_peripheral** device, it bonds (if no bond was previously made), does service discovery (only the first time it connects with the peripheral), configures notification and waits for notifications from the peripheral. If no data is sent within 5 seconds, the node disconnects and reenters Deep-sleep mode. The peripheral sends a notification with the value of the temperature read through an ADC from a thermistor, if present, or randomly generated, if not. When the central receives the notification, it displays it on the terminal window and disconnects in 5 seconds as shown in the figure below.

   When the 60 seconds timer expires or the connection ends, the device reenters Deep-sleep mode until the WAKESW is pressed again and all syncs with periodic advertising trains are terminated. If gAppLowpowerEnabled_d is set 0, LEDs are enabled. The SCANLED flashes, whenever the device is scanning and is ON otherwise. The CONNLED flashes, whenever there is a connection under way and is ON otherwise. See the figure below.



Connection on adv_ext_central console

When the PAWR support is enabled and a connection with a previously bonded device takes place over PAWR, the applications behavior is similar except in this case the adv_ext_central's role is peripheral and adv_ext_peripheral's role is central.

```
PAWR Started
PAWR Set Subevent Data Complete
Periodic advertising response received
Adv Handle: 0
Subevent: 3
Response slot: 2
Response data: 564DE1FC4957
 Gap_ConnectFromPawr Succeeded
Connected!

Disconnected!
```

```
PAWR received
Sync Handle: 0050
Event Counter: 0007
Subevent: 0
Subevent data:
PAWR Data0 For Subevent 0
PAWR Data1 For Subevent 0
Gap_SetPeriodicAdvResponseData Succeded
Periodic Adv Set Response Data Complete

Connected!
Temperature: 24 C

Disconnected!
```

Parent topic: Low-power extended advertising Peripheral and Central

Parent topic:*Bluetooth LE stack and demo applications*

NCP FSCI Black Box

This section describes the functionality of the NCP FSCI Black Box demo application.

Description

The NCP FSCI Black Box application is a demonstration of the Extended NBU architectural concept, where the Bluetooth LE Host Stack and the Link Layer both run on the NBU core, while the

user-facing application runs on the Application core. The communication between the application and the Host is done via the FSCI protocol running on the inter-core RPMSG transport.

The NCP FSCI Black Box application runs on the NBU core and provides access to the Bluetooth LE Host Stack through FSCI to the application running on the Application core. It can be paired with the FSCI Bridge application to obtain a functionality that is identical to the regular Bluetooth FSCI Black Box application, or with the Wireless UART Host application to obtain a functionality that is identical to the regular Wireless UART application.

Parent topic:NCP FSCI Black Box

Supported platforms

The following platforms support the NCP FSCI Black Box application:

- MCX-W72-EVK
- FRDM-MCXW72

Parent topic:NCP FSCI Black Box

Parent topic:*Bluetooth LE stack and demo applications*

NCP Localization Reader

This section describes the functionality of the NCP Localization Reader demo application.

Description

The BLE NCP Localization Reader application is a demonstration program for the radio core (Core 1) based on the MCUXpresso SDK. This application implements a BLE device that performs distance measurement using Channel Sounding (CS) technology.

The application is designed to work together with the corresponding loc_reader_host application running on the application core (Core 0) in a dual-core architecture.

Core 1 Responsibilities

The device handles all BLE-related operations:

- Connection establishment and management
- Channel Sounding procedure execution
- Service discovery (RAS/BTCS)
- Local and remote CS data collection
- Data fragmentation and transmission to Core 0 via FSCI

Data Transfer to Core 0

The application fragments Channel Sounding measurement data into FSCI-sized messages and transmits them to Core 0 using the following event sequence:

- **gIQLocalTrStart_c**: First fragment with measurement header
- **gIQLocalTrCont_c**: Continuation fragments
- **gIQLocalTrEnd_c**: Final fragment

The same sequence is repeated for remote data (gIQRemoteTrStart_c/Cont_c/End_c).

Each message includes device ID to support multi-peer scenarios.

Parent topic: NCP Localization Reader

Supported platforms

The following platforms support the NCP Localization Reader application:

- MCX-W72-EVK
- FRDM-MCXW72

Parent topic:NCP Localization Reader

Parent topic:*Bluetooth LE stack and demo applications*

Over the Air Programming (OTAP)

This section describes the implemented profiles and services, user interactions, and testing methods for the Bluetooth LE OTAP application.

Implemented profile and services

The Bluetooth LE OTAP applications implement the GATT client and server for the custom Bluetooth LE OTAP profile and service.

- **Bluetooth LE OTAP Service** (UUID: 01ff5550-ba5e-f4ee-5ca1-eb1e5e4b1ce0)

The Bluetooth LE OTAP Service is a custom service which has 2 characteristics.

- **OTAP Control Point Characteristic** (UUID: 01ff5551-ba5e-f4ee-5ca1-eb1e5e4b1ce0). This characteristic can be written and indicated to exchange OTAP Commands between the OTAP Server and the OTAP Client. Data chunks are not transferred using this characteristic.
- **OTAP Data Characteristic**(UUID: 01ff5552-ba5e-f4ee-5ca1-eb1e5e4b1ce0). This characteristic can be written without response by the OTAP Server to transfer image file data chunks to the OTAP Client only when an image block transfer is requested via the ATT transfer method. Data chunks can also be transferred via the L2CAP credit-based PSM channels method.

The demo runs using 3 applications: an OTAP Client embedded application, an OTAP Server embedded application, and an Over the Air Programming PC application. The OTAP Client embedded application has two versions, an ATT version and a L2CAP version each using a different transfer method.

The embedded OTAP Server application is a GAP Central application which scans for devices advertising the Bluetooth LE OTAP service. After it finds one, it connects to it and configures the OTAP Control Point CCC Descriptor to receive ATT Indications from the device then it waits for OTAP commands from this device.

Once commands start arriving from the OTAP Client via ATT Indications the OTAP Server relays them via serial interface to a PC application which responds. The responses are then sent back to the OTAP Client by writing the OTAP Control Point Characteristic. The embedded OTAP Server application effectively acts as a relay between the OTAP Client to which the image is sent over the air and the Over the Air Programming PC application which has an OTAP image file constructed using a binary '*.srec*' image or a '*.bin*' image.

The OTAP Client is a GAP Peripheral which advertises the Bluetooth LE OTAP Service and waits for a connection from an OTAP Server. After an OTAP Server connects, the OTAP Client waits for it to write the OTAP Control Point CCCD and then starts sending commands via ATT Indications. If the OTAP Client is configured to ask the data transfer via the L2CAP CoC PSM, it registers and tries to connect a predetermined L2CAP PSM before sending any commands to the OTAP Server.

Parent topic: Over the Air Programming (OTAP)

Supported platforms

The following platforms support the OTAP applications:

- KW45B41Z-EVK
- FRDM-MCXW71
- MCX-W71-EVK
- KW47-EVK
- FRDM-MCXW72
- MCX-W72-EVK
- FRDM-MCXW23
- MCXW23-EVK
- FRDM-KW43

Parent topic: Over the Air Programming (OTAP)

User interface

After flashing two boards with the OTAP Server and OTAP Client applications respectively, the devices are in Idle mode (all LEDs flashing). To start advertising, press the **ADVSW** button on the OTAP Client. To start scanning, press the **SCANSW** button on the OTAP Server. After the two devices connect and start exchanging commands. **CONNLED** becomes solid on the OTAP Server and on the OTAP Client.

Start the OTAP Server PC application after the embedded applications are flashed to the boards. The application creates an OTAP image file using the provided executable .srec or .bin file. It then connects to the embedded OTAP Server via the configured serial interface and waits for commands. The application shows details about the image file creation and allows the OTAP upgrade image file header to be configured. The log view of the application displays the interactions between the OTAP Client and the OTAP Server.

Parent topic: Over the Air Programming (OTAP)

Switches and pins(Client ATT) Switches and pins(Client L2CAP) Switches and pins(Server)

Usage with Over The Air Programming Tool

Below is a list of requirements for usage with Test Tool for Connectivity products:

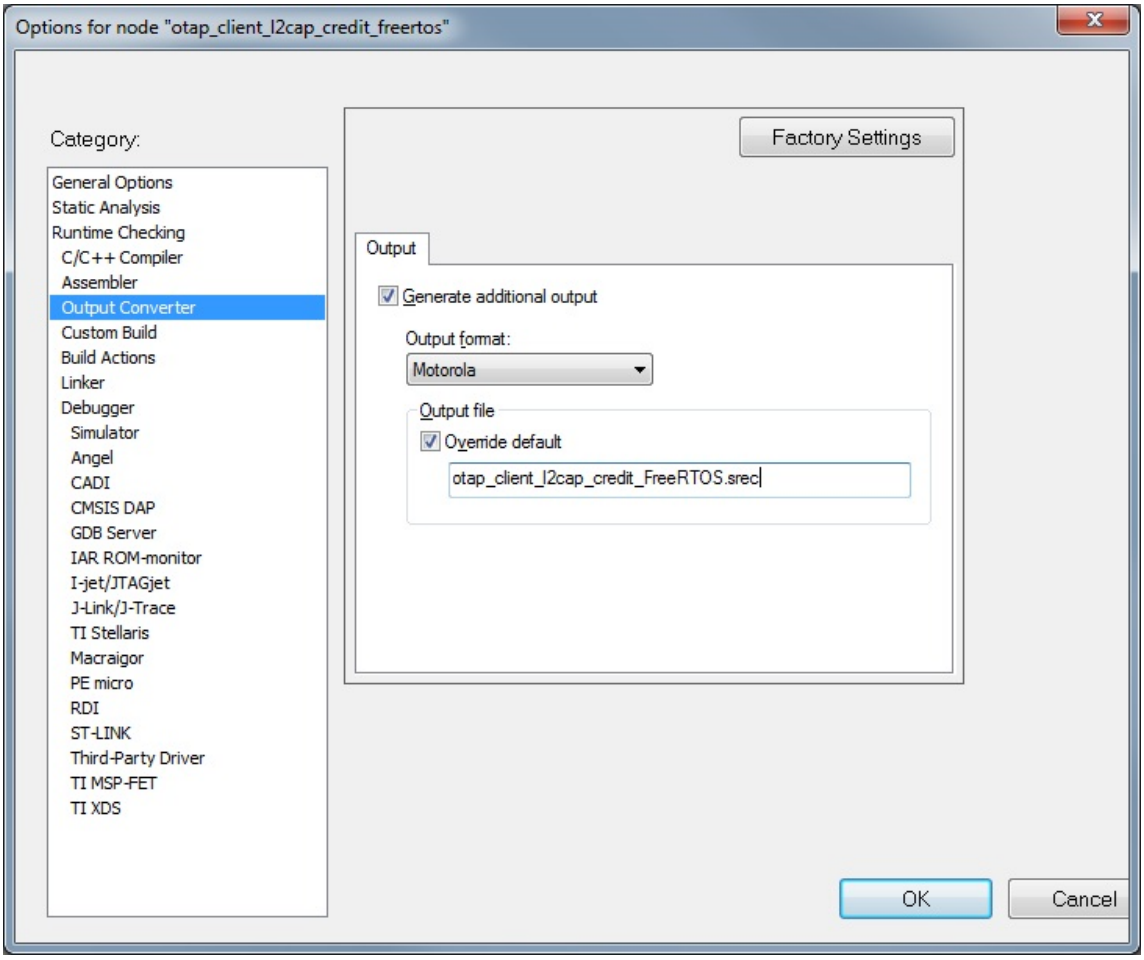
- Over The Air Programming Tool 1.4.0 or newer on CONNECTIVITY-TOOL-SUITE
- Serial COM port drivers – these are board-specific.

To run the application, follow the steps below:

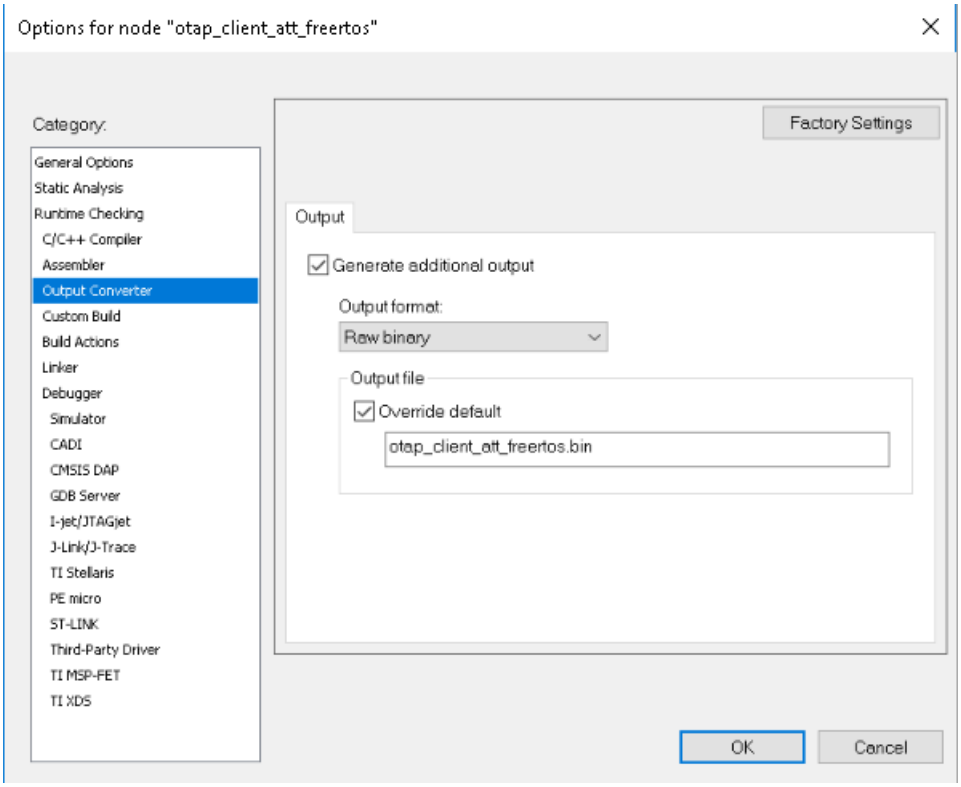
1. Flash the OTAP Server onto a supported platform and the OTAP Client to another supported platform. Make sure the board running the OTAP Server is connected to your PC and your PC has appropriate drivers for the USB to serial device on that board.
2. Create the application to send over the air. The executable must be provided in the .srec or .bin format. The .srec format executable can be obtained by using the IAR Output Converter and setting the output format to Motorola as shown in in the figure below.

   When compiling an image for the Over-the-Air update, the following changes are required:

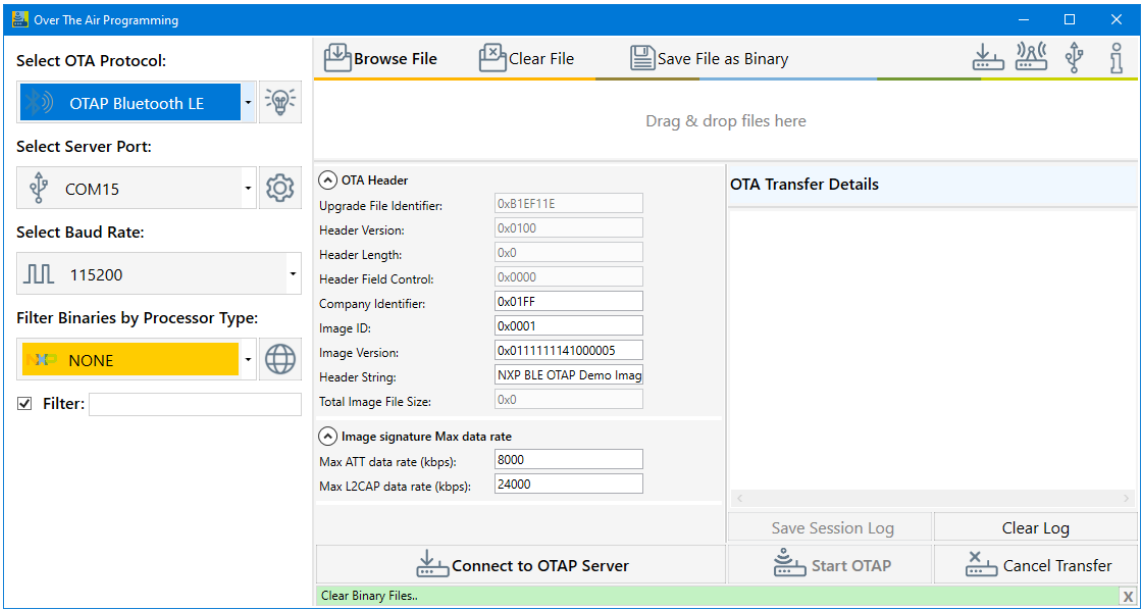
   - The gEraseNVMLink_d linker symbol must be set to 0. For GitHub ARMGCC this may be changed in the application reconfig.cmake file.
   - For ARMGCC, gNvmErasePartitionWhenFlashing_c must be set to 0 in app_preinclude.h



3. To obtain a .bin file from IAR, select the **Raw binary** option in the **IAR Output Converter** as seen in the in the figure below.

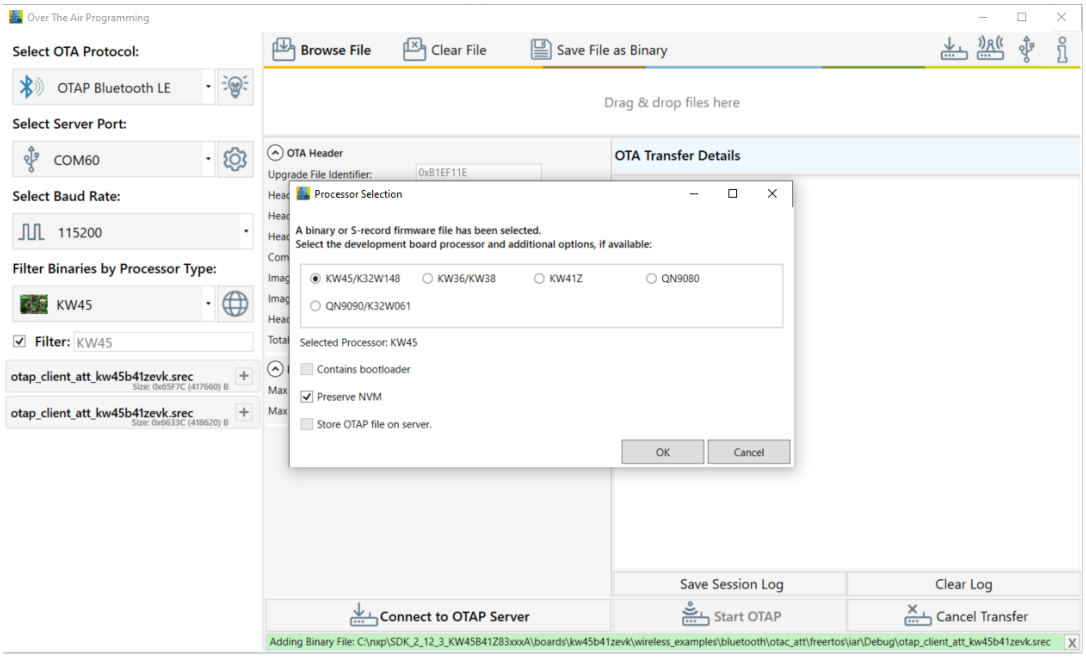


4. Start the **Over The Air Programming** application and select "**OTAP Bluetooth LE**" from the "**Select OTA Protocol**" combo box as shown in the figure below.

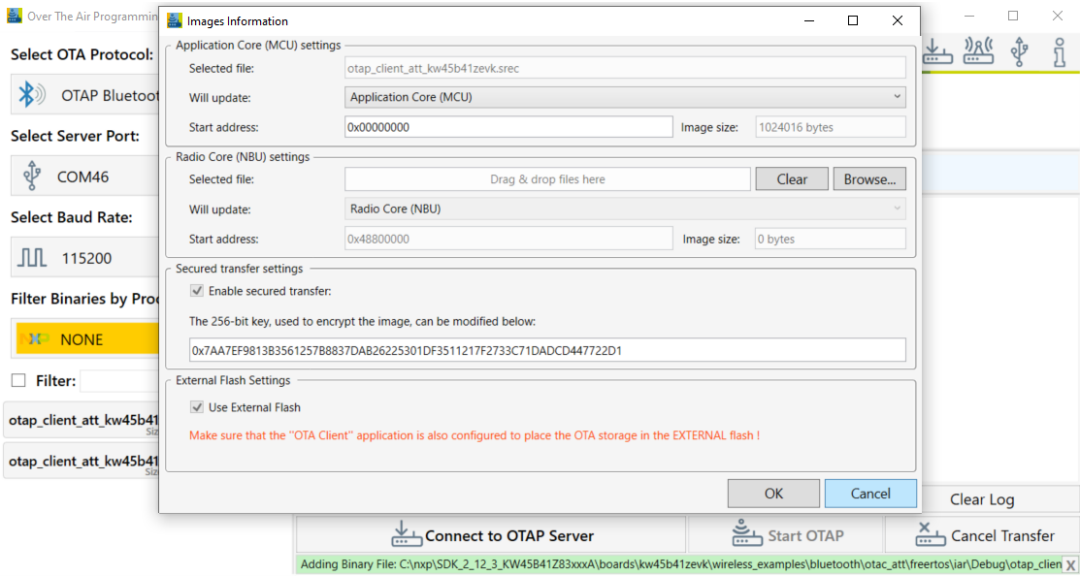


5. **Load the image file into the application, then configure the image file header and start the OTAP Server:**

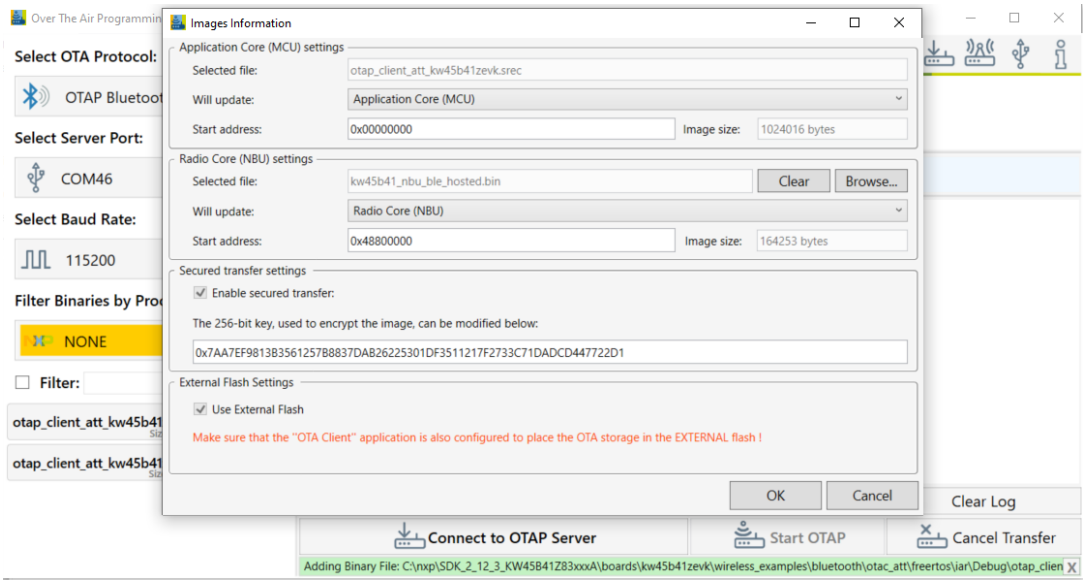
   - To select the updated image In the Over the Air Programming tool, select the "**Browse File**" button and then navigate to the .srec or .bin file containing the image to be sent to the OTAP Client. After the .srec or .bin file is chosen, a pop-up window asks to choose the target processor. In this example, the **KW45/K32W** processor will be used. Choose the **KW45/K32W** processor and press **OK**. See the figure below.



   - Once the processor is selected, a new pop-up window would appear that allows selecting the type of image as shown in the figure below. (In the specified case, we selected the KW45Z/KW45Z(MCU).

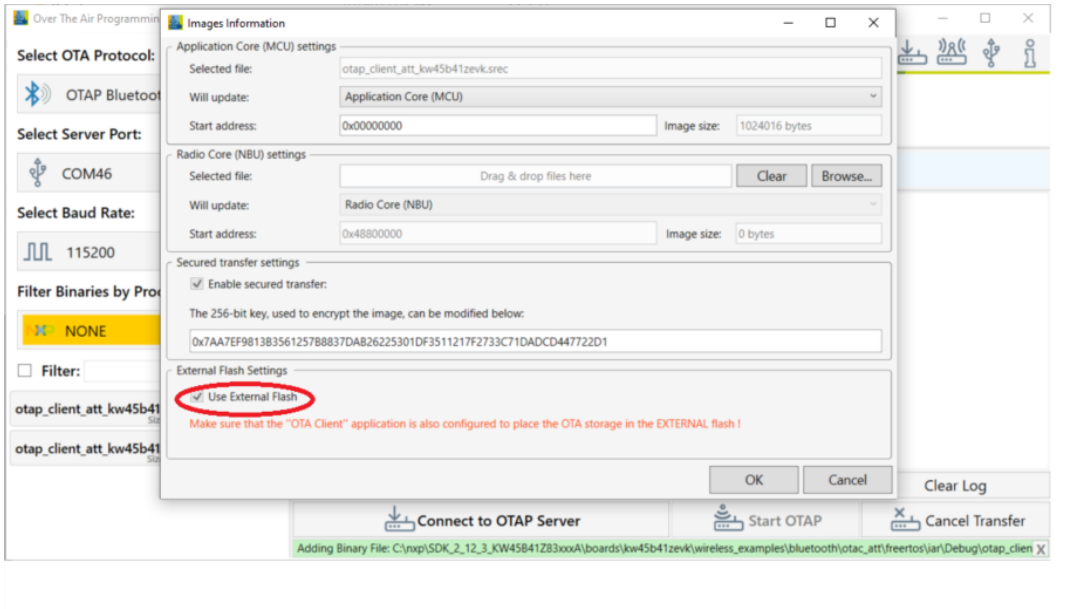


- To update the KW45B/KW32W1 (radio) image, select it by pressing the "**Browse**" button in the M3 group. Then navigate to the .bin file as shown in the figure below.



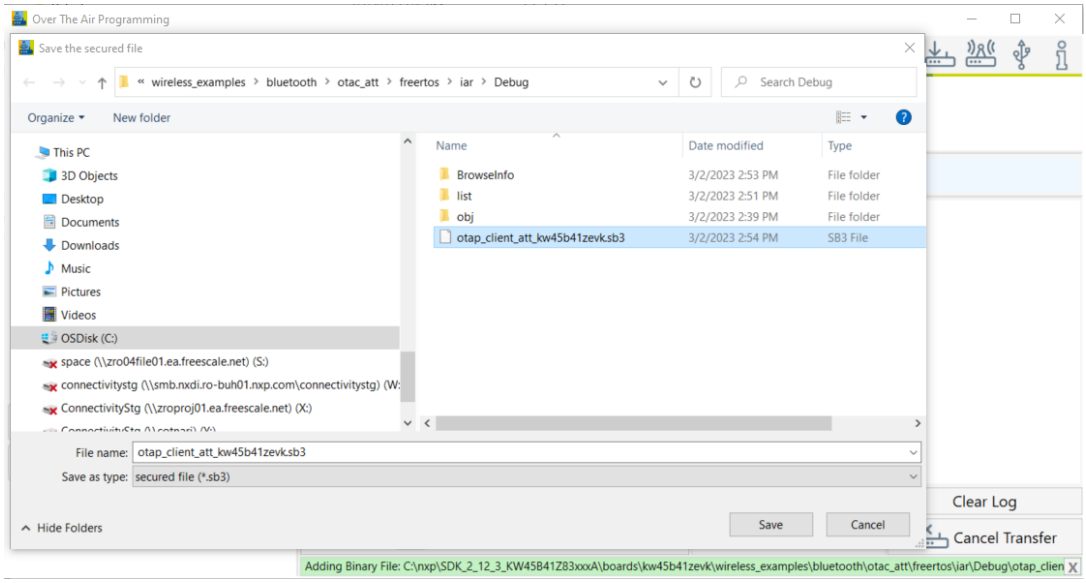
- If the OTA client has configured external memory support, then "**Use External Flash**" checkbox must be checked as in the figure below. If the OTA client has configured internal memory support, the checkbox must be left unchecked.

  This checkbox (if checked) instructs the OTA client bootloader to erase all the internal storage. This must be done only if external memory support is used as shown in the figure below. .



At this moment, click the **OK**button. A new window appears that prompts you to enter a location where the secured file should be stored. By default, the location of the original file is selected. See the figure below.

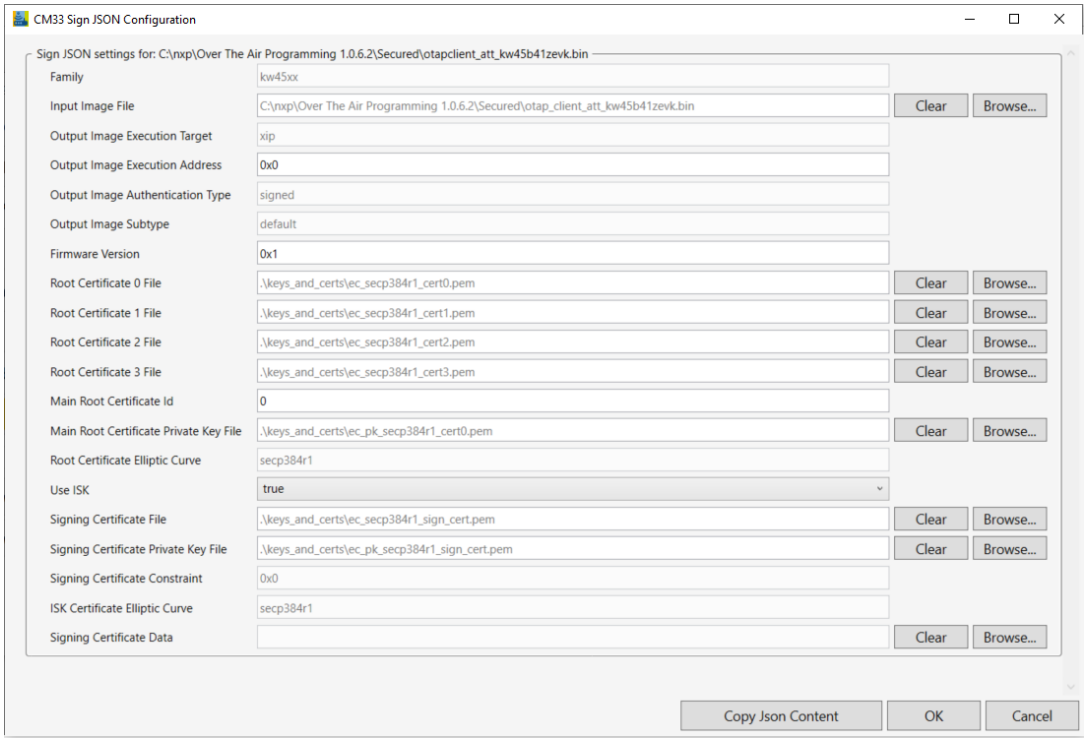
Note: The extension of the secured file is *.sb3. See the figure below.



- You can now configure two different JSON files, used to:
  - Sign the file that is uploaded to the MCU if an MCU file was selected.
  - Create the **.sb3*container that is sent OTA. The **.sb3* file can contain only the MCU file, only the radio file, or both.

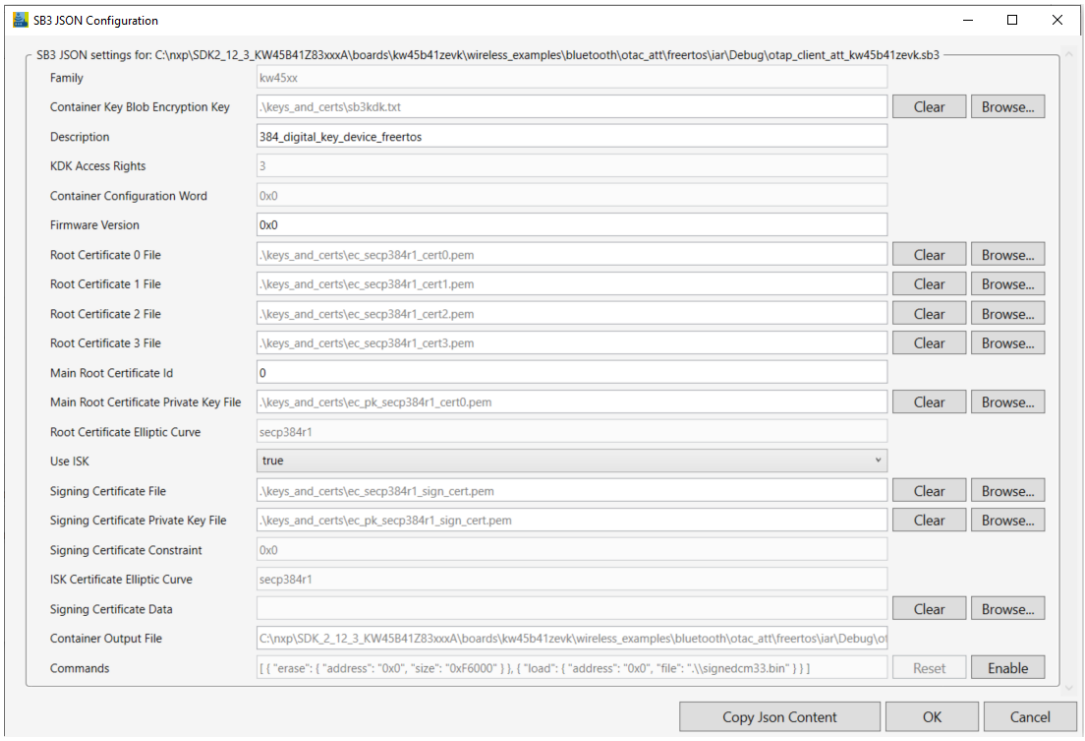
If you select a file that is written on the MCU, a new window appears as shown in the figure below. This window helps in configuring the root certificates and signing the certificates, by either dragging and dropping, or browsing for new files. For details on each field of the JSON, see */Documentation/KW45JsonDescription.pdf* provided with Over the Air Programming tool.

By default, the JSON is configured for the demo applications to run as shown in the figure below.



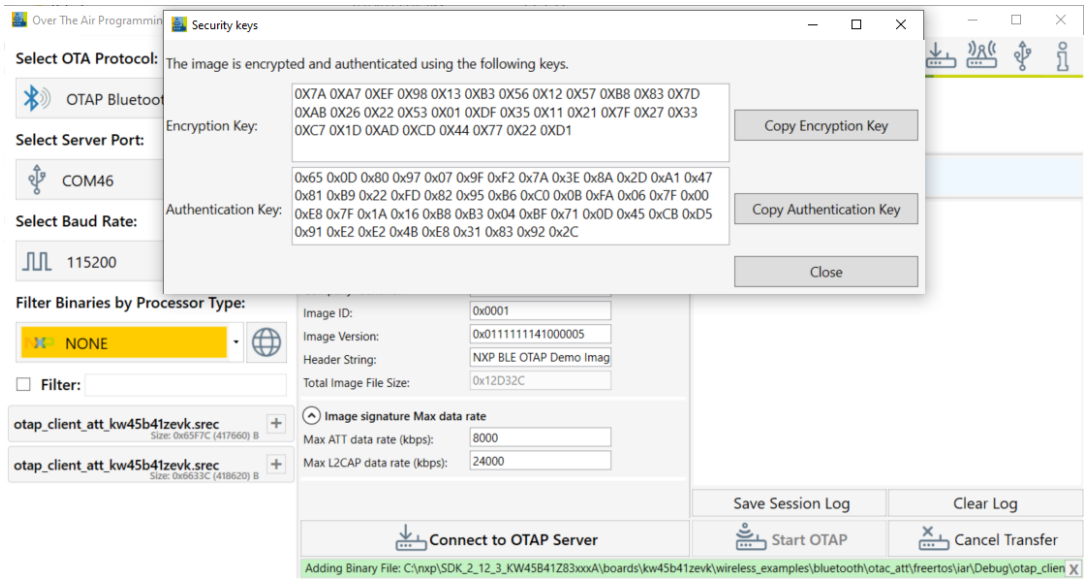
After configuring the JSON file used for signing the MCU file, a new similar window appears. As shown in the Figure, the window is designed for configuring the **.sb3* container. This window helps you to configure the encryption key file, the root certificates, and the signing certificates by either drag and dropping or browsing for new files. For details on each field of the JSON file, see */Documentation/KW45JsonDescription.pdf* provided with Over the Air Programming tool.

By default, the JSON is configured for the demo applications to run as shown in the figure below.

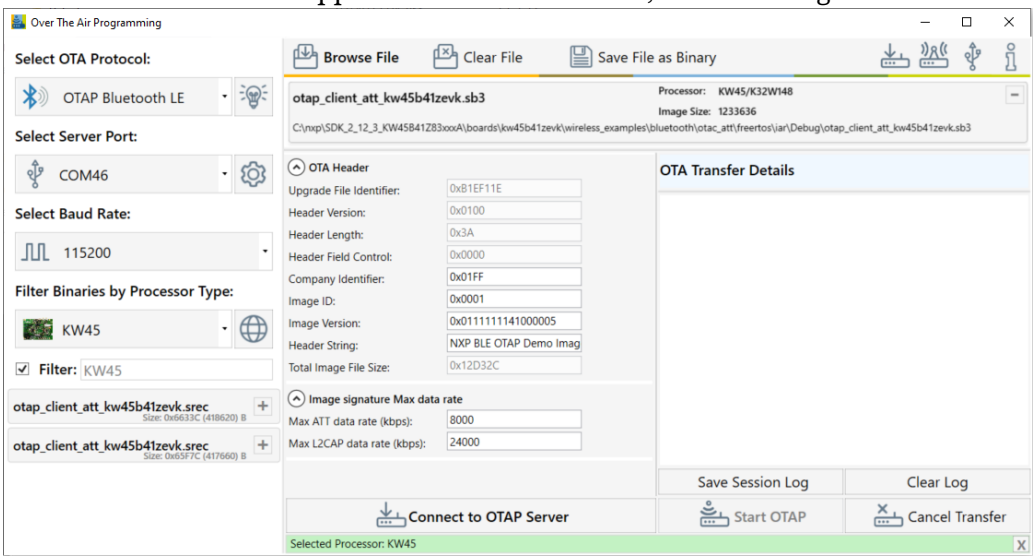


- After the .sb3 file is created, the "**Encryption Key**" and "**Authentication Key**" are presented. For the secured update to be successful, the destination board must have been

provisioned with these keys through fuse burning, as described in the accompanying document. Depending on the board type, it can either be already provisioned by NXP (KW45\KW47\MCX-W71\MCX-W72\KW43 samples) or not provisioned (loose samples). See the figure below.



- The OTA Header configuration options from the "**OTA Header**" box are used by the application to build the **OTAP Image File**, which is sent over the air. The default values of the OTA Header configuration work out of the box for the OTAP demo applications. For details about these configuration options, see the *Bluetooth LE Application Developer's Guide* document (*BLEADG*).
- After the image is loaded, go to the "**Select Server Port**" box, select the correct COM Port for the OTAP Server board. Also select the default baud rate of 115200 and press the "**Connect to OTAP Server**" button. A successful connection is displayed in the Message Log.
  - If the image is loaded before connecting to the OATP Server COM Port, then the OTAP Server of the application starts automatically.
  - If the connection to the COM Port is established before the image is loaded, then the "**Start OTAP**" button must be pressed to start the OTAP Server of the application. For details, see the figure below. .



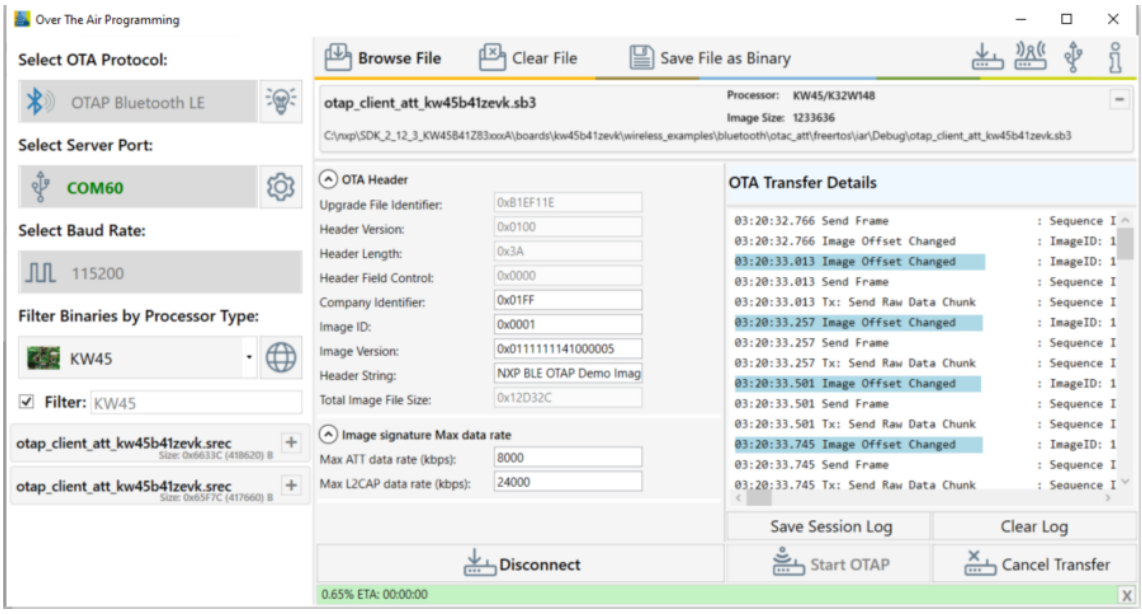
- Before starting the image transfer process, the data rate must be configured for each

transfer method (ATT or L2CAP CoC). The image chunks of a block are sent over the serial interface and over-the-air without waiting for confirmation. Data rate can significantly slow down if configuration is not done correctly and errors appear in the transfer process.

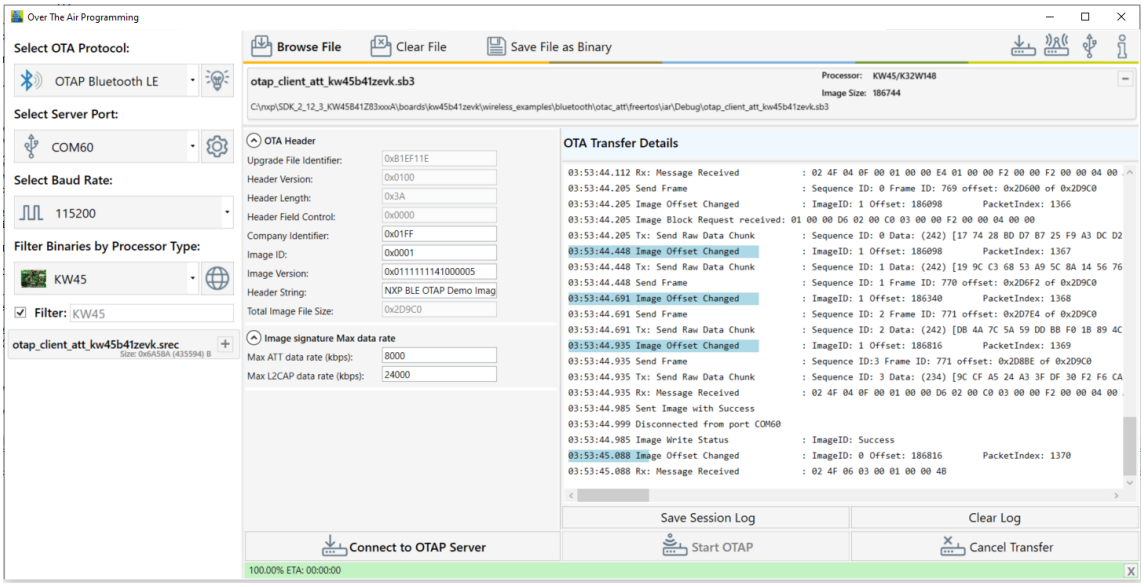
The optimal data rate depends on multiple factors. Some of these factors are listed below:

- Distance between boards
- Type of antenna
- Performance of the RF circuitry between the radio and antenna
- Type and level of noise in the environment
- Speed of the storage medium in which the image is saved on the OTAP Client
- Serial driver delay between PC and the OTAP Server board If the data rate is too high, then the OTAP Client receives a new chunk before it can process the previous one. In such a case, it sends an "**Unexpected Chunk Sequence Number**" error and restarts the transfer of the current block from where it left off. If the channel is too noisy, the transmitter can be flooded and some chunks might not reach the client triggering a similar type of error. The default data rate values should work for most configurations.

6. Start the embedded applications by pressing **ADVSW** first on the OTAP Client and then on the OTAP Server. The transfer progress and transfer-related messages and/or errors are shown in the application window. The duration of the transfer depends on the size of the image and the chosen data rate and transfer method. See the figure below.



7. After all the blocks are sent, the OTAP Client sends an Image Transfer Complete command to the OTAP Server. When the PC Application receives this command, it displays a Sent Image with Success message in the log window. See the figure below.



8. After the image transfer is complete, the OTAP Client triggers the bootloader and resets the MCU. The bootloader takes about 30 seconds to flash the image on the board. After this time frame, the MCU resets again and runs the new image.

Parent topic: Over the Air Programming (OTAP)

Storage type selection

OTAP Clients support both internal and external storage. By default, the applications come with internal storage enabled. To configure external storage, follow the steps bellow:

1. Set the gUseInternalStorageLink_d define to 0 in the project settings. For GitHub ARMGCC, replace CONFIG_MCUX_COMPONENT_middleware.wireless.framework.ota_support.no_flash_selected=y with CONFIG_MCUX_COMPONENT_middleware.wireless.framework.ota_support.external_flash=y in the application prj.conf file.
2. In app_preinclude.h, set the gAppOtaExternalStorage_c define to 1.

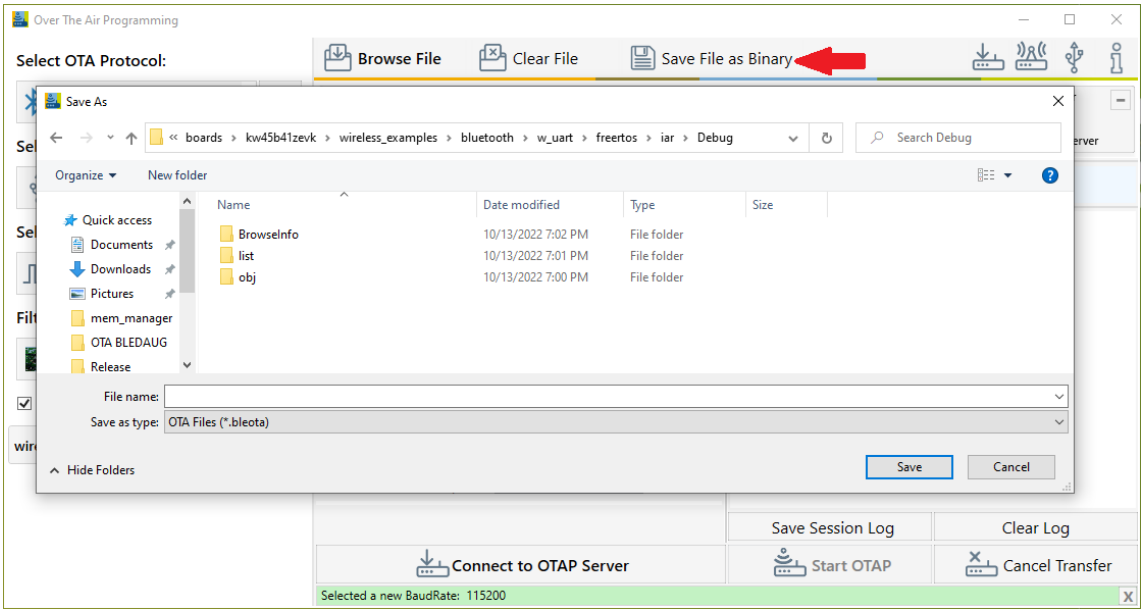
Usage with IoT Toolbox

This is the list of requirements.

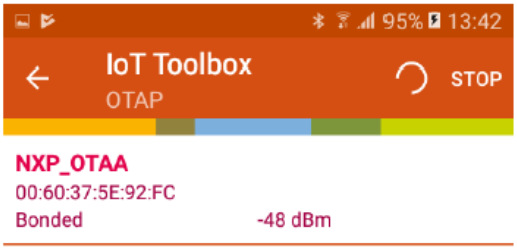
- **Mobile device** running **Android** platform or **iOS** with hardware and software supporting **Bluetooth 4.0 and later**.
- **Kinetis Bluetooth LE Toolbox** application – download from the specific application store for your device.

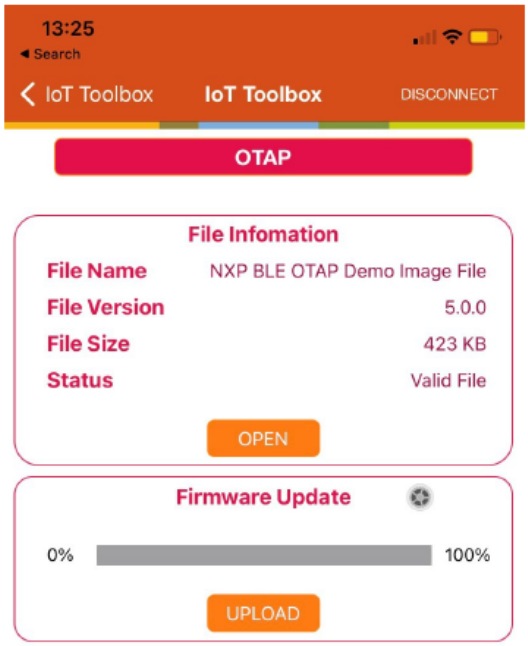
To run the application, perform the following steps:

1. Flash the OTAP Client ATT to either the KW45B41Z-EVK or the FRDM-MCXW71 platform. The Kinetis Bluetooth LE Toolbox only supports the ATT OTAP Client.
2. In order to send over the air in .bleota format, create the application. In order to load the image file into the Over the Air Programming application and create the .sb3 file, follow the instructions described in Usage with Over The Air Programming Tool. Once the .sb3 file is created, press the "**Save File as Binary**" button to create the .bleota file. See the figure below.



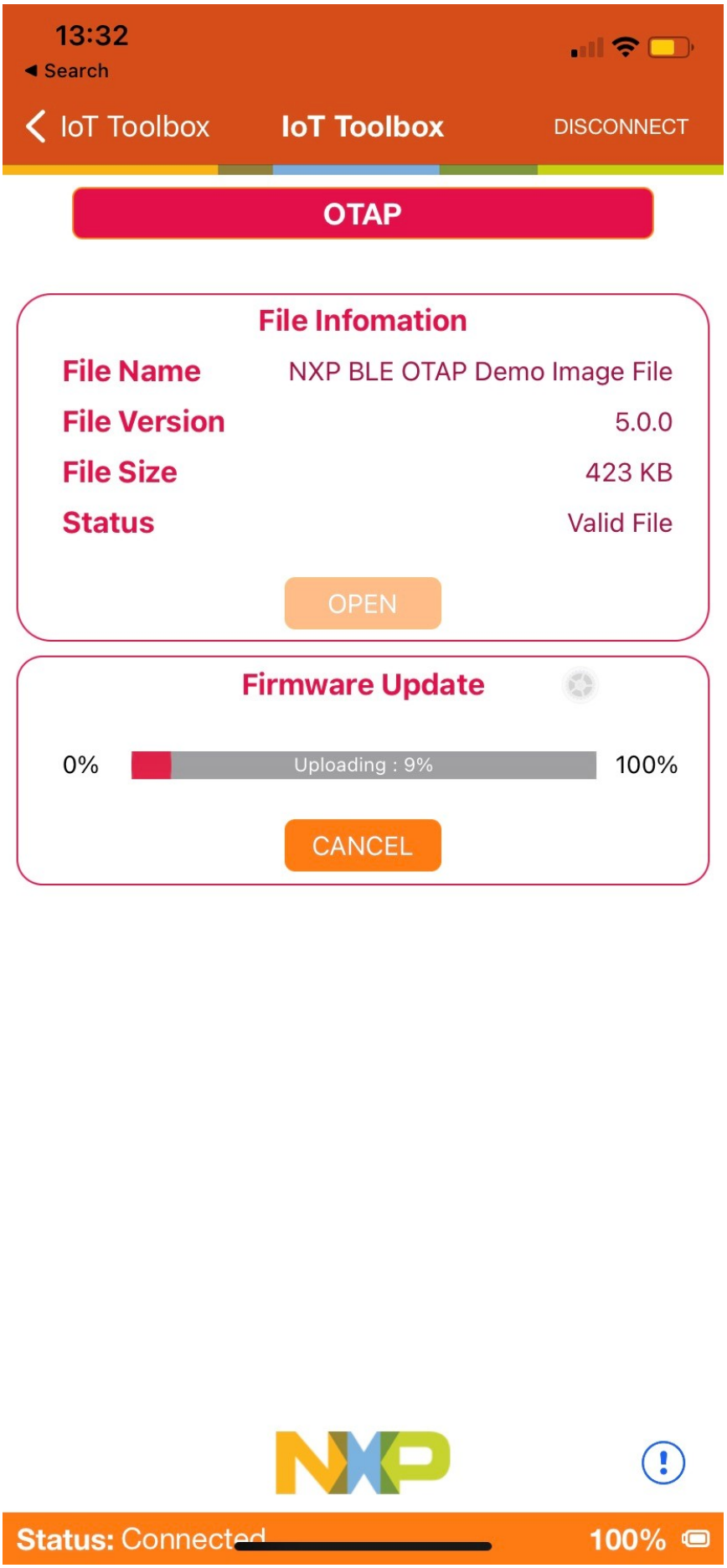
3. Start the **Kinetis Bluetooth LE Toolbox**application on your mobile device and start the **OTAP Tool**. The application starts scanning.
4. Press **ADVSW** on the board to start Advertising on the embedded OTAP Client application. The device should show up in the list of scanned devices. Touch the device in the scan list to connect to and the application performs service discovery and displays some information shown in the figure below.





5. Press the "**Open**" button and load the .bleota file to be sent over-the-air. Once the file is loaded, some information about it is displayed. Press the "**Upload**" button to start the image

transfer process. A progress bar is shown while the image transfer is ongoing. The progress bar displays 100% update after a successful transfer, as shown in the figure below.


13:32
Search
IoT Toolbox
IoT Toolbox
DISCONNECT
OTAP
File Infomation
File Name
NXP BLE OTAP Demo Image File
File Version
5.0.0
File Size
423 KB
Status
Valid File
OPEN
Firmware Update
0%
Uploading : 9%
100%
CANCEL
Status: Connected
100%

6. After the image transfer is complete, the OTAP Client triggers the bootloader and resets the MCU. The bootloader takes about 30 seconds to flash the image on the board. After this time passes, the MCU resets again and runs the new image.

Parent topic: Over the Air Programming (OTAP)

Parent topic:*Bluetooth LE stack and demo applications*

Wireless UART

This section describes the implemented profiles and services, user interactions, and testing methods for the Wireless UART application.

Implemented profile and services

The Wireless UART application implements both the GATT client and server for the custom Wireless UART profile and services.

- Wireless UART Service (UUID: 01ff0100-ba5e-f4ee-5ca1-eb1e5e4b1ce0)
- Battery Service v1.0
- Device Information Service v1.1

The Wireless UART service is a custom service that implements a custom writable ASCII Char characteristic (UUID: 01ff0101-ba5e-f4ee-5ca1-eb1e5e4b1ce0) that holds the character written by the peer device.

The application behaves at first as a GAP central node. It enters GAP Limited Discovery Procedure and searches for other Wireless UART devices to connect. To change the device role to GAP peripheral, use the ROLESW button. The device enters GAP General Discoverable Mode and waits for a GAP central node to connect.

Parent topic: Wireless UART

Supported platforms

The following platforms support the Wireless UART application:

- KW45B41Z-EVK
- KW45B41Z-LOC
- FRDM-MCXW71
- MCX-W71-EVK
- KW47-EVK
- KW47-LOC
- KW43-LOC
- FRDM-MCXW72
- MCX-W72-EVK
- MCXW72-LOC
- FRDM-MCXW23
- MCXW23-EVK
- FRDM-KW43

Parent topic: Wireless UART

User interface

After flashing the board, the device is in idle mode (all LEDs flashing). To start scanning, press the **SCANSW** button. When in GAP Limited Discovery Procedure of GAP General Discoverable Mode, **CONNLED** is flashing. When the node connects to a peer device, **CONNLED** turns solid. To disconnect the node, hold the **SCANSW** button pressed for 2-3 seconds. The node then re-enters GAP Limited Discovery Procedure.

Parent topic: Wireless UART

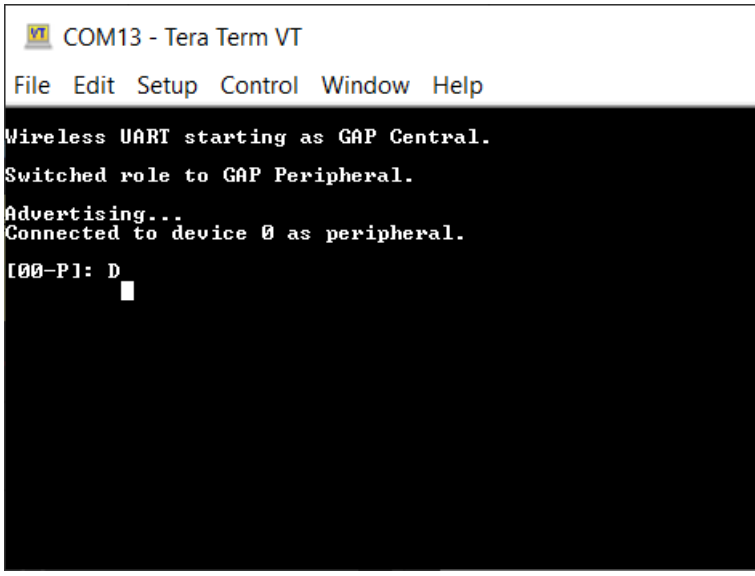
Switches and pins

Usage

The application is built to work with another supported platform running the same example or with the Wireless UART from the IoT Toolbox application.

When testing with two boards, perform the following steps:

1. Open a serial port terminal and connect them to the two boards, in the same manner described in Testing devices. The start screen is blank after the board is reset.
2. The application starts as a GAP central. To switch the role to a GAP peripheral, press the role switch. Depending on the role, when pressing the **SCANSW**, the application starts either scanning or advertising.
3. As soon as the **CONNLED** turns solid on both devices, the user can start writing in one of the consoles. The text appears on the other terminal.
4. After creating a connection, the role (central or peripheral) is displayed on the console. The role switch can be pressed again before creating a new connection. See the figure below.



Tera Term – received text on Wireless UART

When testing with a single board and the IoT Toolbox, perform the following steps:

1. Open a serial port terminal and connect the board in the same manner described in Testing devices. The start screen is blank after the board is reset.
2. Press the role switch button to behave as a GAP peripheral and then press the **SCANSW** button to start advertising. The IoT Toolbox app can then connect. Select UART instead of Console and start typing, as shown in the Figure.

Parent topic: Wireless UART

Out of band GATT Client

The application supports setting the required GATT client handles, thus avoiding service discovery after each connection. The method for obtaining the service handles is out of the scope of this example. To activate this feature, user must set gDbOobPopulated_c to 1. The required handles are the Wireless UART Service handle (expressed as gDbOobServiceHandle_c) and the UART stream service handle (expressed as gDbOobUartStreamHandle_c). When gDbOobPopulated_c is set, these two macros must be also set with corect values reflecting Server's handles.

Parent topic: Wireless UART

Parent topic:*Bluetooth LE stack and demo applications*

Bluetooth LE Shell

This section describes the functionality, user interactions, and testing methods for the Bluetooth LE Shell Application.

Implemented stack features

The Bluetooth LE Shell Application implements a console application that allows the user to interact with a full feature Bluetooth Low Energy stack library. It implements All GAP roles and both GATT client and server. Enabling these roles can be done using shell commands.

Parent topic: Bluetooth LE Shell

Implemented profile and services

The application implements a dynamic GATT database. The user can add services at runtime and also erase the database contents. The database is always populated with the GAP and GATT services. These services cannot be erased. The user can dynamically add the following services:

- Heart Rate Service (UUID: 0x180D)
- Battery Service (UUID: 0x180F)
- Device Information Service (UUID: 0x180A)
- Internet Support Profile Service (0x1820)

Parent topic: Bluetooth LE Shell

Supported platforms

The following platforms support the Bluetooth LE Shell application:

- KW45B41Z-EVK
- FRDM-MCXW71
- MCX-W71-EVK
- KW47-EVK
- FRDM-MCXW72
- MCX-W72-EVK
- FRDM-KW43

Parent topic: Bluetooth LE Shell

User interface

After flashing the board, the device is in idle mode. The interaction with the board is done entirely by using the shell commands via the serial communication terminal.

Parent topic: Bluetooth LE Shell

Usage

The application is built to work with any other Bluetooth LE device. To showcase the functionality, two platforms are used in the following setup.

1. Open a serial port terminal and connect them to the two boards, in the same manner described in Testing devices. The start screen is displayed after the board is reset. All LEDs are flashing on both devices.

2. Configure one of the devices as a GAP peripheral and a Heart Rate server. Change name to HRS. Start advertising on this device.

```
BLE Shell>gap devicename HRS
--> GATTDB Event: Attribute Written
HRS>gap advdata 1 6
HRS>gap advdata 8 HRS
HRS>gap advstart
HRS>
--> GAP Event: Advertising parameters successfully set.
HRS>
--> GAP Event: Advertising data successfully set.
HRS>
--> GAP Event: Advertising state changed successfully!
HRS>gattdb addservice 0x180D
--> Heart Rate
- Heart Rate Measurement Value Handle: 14
```

3. Configure the other device as a GAP central. Change its name to 'Collector'. Start scanning and connect to the HRS device by selecting the corresponding device index from the list of scanned devices. In the example below, the HRS device is device number 2. The number of listed scanned devices can be controller through the mShellGapMaxScannedDevicesCount_c define in shell_gap.c.

```
BLE Shell>gap devicename Collector
--> GATTDB Event: Attribute Written
Collector>gap scanstart filter
--> GAP Event: Scan started.
Collector>
--> GAP Event: Found device 0 : 880F102F500E 0 dBm
--> GAP Event: Found device 1 : NXP_CSCS 00049F000006 0 dBm
--> GAP Event: Found device 2 : HRS 00049F0000FF 0 dBm
Collector>gap connect 2
--> GAP Event: Scan stopped.
Collector>
--> GAP Event: Connected to peer 0
```

4. Optionally, the devices can be paired (gAppUsePairing_d and gAppUseBonding_d must be set in app_preinclude.h). On the collector initiate the pairing.

```
Collector>gap pair 0
--> Pairing...
--> GAP Event: Link Encrypted
--> GAP Event: Device Paired
```

5. On the Collector, start service discovery. The device discovers the GAP, GATT, and Heart Rate services.

```
Collector>gatt discover 0 -all
--> Discovered primary services: 3
--> Generic Attribute Start Handle: 1 End Handle: 4
- Service Changed Value Handle: 3
- Client Characteristic Configuration Descriptor Handle: 4
--> Generic Access Start Handle: 5 End Handle: 11
```

(continues on next page)

(continued from previous page)

```
- Device Name Value Handle: 7
- Appearance Value Handle: 9
- Peripheral Preferred Connection Parameters Value Handle: 11
--> Heart Rate Start Handle: 12 End Handle: 19
- Heart Rate Measurement Value Handle: 14
- Client Characteristic Configuration Descriptor Handle: 15
- Body Sensor Location Value Handle: 17
- Heart Rate Control Point Value Handle: 19
```

6. Configure the HRS to send notifications by writing the CCCD from the Collector. Send a GATT write command with value 1 to the CCCD handle discovered, 15.

```
Collector>gatt write 0 15 0x0001
--> GATT Event: Characteristic Value Written!
```

7. Send heart rate measurement notifications from the HRS device by using the value handle obtained after adding the service in the previous step.

```
HRS>gatt notify 0 14
```

8. A notification appears on the Collector console.

```
Collector>
--> GATT Event: Received Notification
Handle: 14
Value: B400
```

Extended advertising

Use the Bluetooth LE Shell application to exercise the advertising extension features:

On the GAP Peripheral device:

1. Configure the extended advertising parameters. In the below example, the advertising type is set to connectable and includes TX power and the primary PHY is set to Coded PHY.
2. Configure the extended advertising data. The Bluetooth LE Shell applications has the feature to send for test, a large data payload. Use the extended advertisement default configuration (not call "gap extadvcfg"), pass the command "gap extadvdata" with no parameters and the default data is added. The length is configurable at compile time through SHELL_EXT_ADV_DATA_SIZE and the data pattern is SHELL_EXT_ADV_DATA_PATTERN. Start the default test with call for "gap extadvstart".

   The advertising data type is set to shortened local name (8) and the advertising data content is set to test_ext_adv_data.

   Note: Users must note that extended connectable advertising does not allow for chained advertising data. The data length must be limited to what can fit in a single AUX_ADV_IND PDU (251 bytes at maximum). This means that passing the gap extadvdata with no parameters and the default value of SHELL_EXT_ADV_DATA_SIZE (500 bytes) after having set the advertising type to connectable will result in an error when trying to start advertising.
3. Start extended advertising.

```
BLE Shell>gap extadvcfg -type 65 -phy1 3
BLE Shell>gap extadvdata 8 test_ext_adv_data
BLE Shell>gap extadvstart
--> GAP Event: Extended
Advertising parameters successfully set.
--> GAP Event:
Extended Advertising data successfully set.
--> GAP Event: Advertising state changed successfully!
```

4. **On the GAP Central device**

   Set the scanning parameters. The scanning PHY is set to match the advertising PHY, in this case Coded PHY.

5. Start scanning and filter duplicates.

```
BLE Shell>gap scancfg -phy 4
BLE Shell>gap scanstart filter
BLE Shell>
-> GAP Event: Scan started.
BLE Shell>
--> GAP Event: Found device 0 : 0060375BCEC6 -23 dBm
Advertising Extended Data:
test_ext_adv_data
```

6. Set the connection initiating PHYs corresponding to the primary PHY on which the advertising is performed.

7. Connect to the desired device in the scanned devices list.

```
BLE Shell>gap connectcfg -phy 4
--> Connection Parameters:
--> Connection Interval: 200 ms
--> Connection Latency: 0
--> Supervision Timeout: 32000 ms
--> Connecting PHYs: Coded
BLE Shell>gap connect 0
BLE Shell>
-> GAP Event: Scan stopped.
BLE Shell>
--> GAP Event: Connected to peer 0
```

Parent topic: Usage

RSSI Monitor

RSSI Monitor is an application that allows monitoring the RSSI of a remote peer on advertising or connection channel. The GAP peripheral device can modify the output TX power on both advertising and connection channels.

1. **On GAP peripheral device**

   Set the primary advertising PHY to Coded PHY. Start advertising and read the address. Set the TX power in dBm to a value less than 20 dBm.

```
BLE Shell>gap address
BLE Shell>
--> GAP Event: Public Address:C4603770BCC5
BLE Shell>gap extadvcfg -phy1 3
BLE Shell>gap extadvstart
BLE Shell>
--> GAP Event: Extended Advertising parameters successfully set.
BLE Shell>
--> GAP Event: Extended Advertising data successfully set.
BLE Shell>
--> GAP Event: Advertising state changed successfully!
BLE Shell>gap txpower adv 0
BLE Shell>
--> GAP Event: Success!
```

2. **On GAP Central device**

   Set the scanning PHY to Coded PHY. Start monitoring the RSSI on advertising Channel using the address of the Peripheral device. Scanning starts automatically, if it is not previously

enabled.

```
BLE Shell>gap scancfg -phy 4
BLE Shell>gap rssimonitor C4603770BCC5--> Reading RSSI on advertising channel:
BLE Shell>
-> GAP Event: Scan started.
BLE Shell>
 RSSI: -27 dBm
 RSSI: -27 dBm
 RSSI: -27 dBm
 RSSI: -27 dBm
 RSSI: -27 dBm
 RSSI: -29 dBm
```

In the below example, the RSSI in monitored on a connection channel. On GAP Peripheral, start advertising in connectable mode on Coded PHY and adjust the TX power level.

```
BLE Shell>gap extadvcfg -type 65 -phy1 3
BLE Shell>gap extadvdata 8 rssimonitortest
BLE Shell>gap extadvstart
BLE Shell>
--> GAP Event: Extended Advertising parameters successfully set.
BLE Shell>
--> GAP Event: Extended Advertising data successfully set.
BLE Shell>
--> GAP Event: Advertising state changed successfully!
BLE Shell>
--> GAP Event: Connected to peer 0
BLE Shell>
--> GAP Event: Advertising stopped!
BLE Shell>gap txpower conn 10
BLE Shell>
--> GAP Event: Success!
```

On the GAP Central device, start scanning on the Coded PHY. Update the connection PHY also to Coded PHY, then connect to the remote device and monitor continuously the RSSI on the connection channel.

```
BLE Shell>gap scancfg -phy 4
BLE Shell>gap connectcfg -phy 4
-->  Connection Parameters:
    -->  Connection Interval: 200 ms
    -->  Connection Latency: 0
    -->  Supervision Timeout: 32000 ms
    -->  Connecting PHYs: Coded
BLE Shell>gap scanstart filter
BLE Shell>
->  GAP Event: Scan started.
BLE Shell>
-->  GAP Event: Found device 0 : C4603770BCC5 -21 dBm
 Advertising Extended Data:
rssimonitortest
gap connect 0
BLE Shell>
->  GAP Event: Scan stopped.
BLE Shell>
-->  GAP Event: Connected to peer 0
BLE Shell>gap rssimonitor 0 -c
-->  Reading RSSI from connected device:
BLE Shell>
 RSSI: -22 dBm
```

(continues on next page)

(continued from previous page)

```
 RSSI: -23 dBm
 RSSI: -21 dBm
 RSSI: -22 dBm
 RSSI: -22 dBm
BLE Shell>gap rssistop
```

3. Update the PHY preference and continue monitoring the RSSI. For coded PHY, the coding scheme can be configured between S2 and S8 (500 kbit/s and 125 kbit/s).

```
BLE Shell>gap phy 0 -tx 4 -rx 4 -o 1
BLE Shell>
--> GAP Event: Phy update complete with peer 0
--> TxPhy: Coded
--> RxPhy: Coded

BLE Shell>gap phy 0 -tx 2 -rx 2
BLE Shell>
--> GAP Event: Phy update complete with peer 0
--> TxPhy: 2M
--> RxPhy: 2M

BLE Shell>gap rssimonitor 0 -c
--> Reading RSSI from connected device:
BLE Shell>
RSSI: -23 dBm
RSSI: -23 dBm
RSSI: -21 dBm
RSSI: -21 dBm
--> GAP Event: Phy update complete with peer 0
--> TxPhy: Coded
--> RxPhy: Coded

BLE Shell>
RSSI: -22 dBm
RSSI: -21 dBm
RSSI: -22 dBm
RSSI: -23 dBm
RSSI: -23 dBm
--> GAP Event: Phy update complete with peer 0
--> TxPhy: 2M
--> RxPhy: 2M
BLE Shell>
RSSI: -22 dBm
RSSI: -21 dBm
RSSI: -21 dBm
RSSI: -20 dBm
```

Parent topic: Usage

Transient Application Key

Use the Bluetooth LE Shell application to exercise the TAK(Transient Application Key) feature. First, 'gAppUseTAK_d' must be enabled and 'gConnTakMaxEntries_c' must be configured to the maximum number of keys we want to support on a device. Also 'gBleHostAutoRejectLtkRequestForUnbondedDevices_c' must be set to FALSE in case the device is peripheral. The key is stored per device ID. The device ID is obtained after connection. Each key is erased after it is used, as per TAK specification requirements.

On the GAP Peripheral device:

1. Configure the extended advertising data to include TAK identifier and set the TAK key for the desired device.

The advertising data type is set to shortened local name (8) and the advertising data content is set to TAK_ID.

2. Start extended advertising.

```
BLE Shell>gap extadvdata 8 TAK_ID
BLE Shell>gap extadvstart
--> GAP Event: Extended
Advertising parameters successfully set.
--> GAP Event:
Extended Advertising data successfully set.
--> GAP Event: Advertising state changed successfully!
...
--> GAP Event: Connected to peer 0
BLE Shell>gap tak 0 00112233445566778899aabbccddeeff
```

3. **On the GAP Central device**

   Set the scanning parameters.

4. Start scanning.

```
BLE Shell>gap scanstart
BLE Shell>
-> GAP Event: Scan started.
BLE Shell>
--> GAP Event: Found device 0 : 0060375BCEC6 -23 dBm
Advertising Extended Data:
TAK_ID
...
BLE Shell>gap tak 0 00112233445566778899aabbccddeeff
```

5. Connect to the desired device in the scanned devices list.

```
BLE Shell>gap connect 0
BLE Shell>
-> GAP Event: Scan stopped.
BLE Shell>
--> GAP Event: Connected to peer 0
```

Parent topic: Usage

Parent topic: Bluetooth LE Shell

Throughput feature

The Bluetooth LE Shell application also has a throughput test feature that can be used to test different combinations of the parameters (connection interval, payload size, and packet count) to determine the best data-rate.

This feature requires two devices:

- GAP Peripheral: transmits the test packets
- GAP Central: receives the packets and displays a report

All throughput-related commands are grouped under the **thrput** keyword:

- thrput setparam: configures connection interval, packet count and payload size.
- thrput start tx: configures the device as a GAP Peripheral and starts advertising. Once the receiving device is connected, the packet transmission begins. The packet size and count can also be specified (-s<size_value>-c<count_value>).
- thrput start rx: configures the device as a GAP Central and starts scanning. Once a transmitter device is found, it connects to it and waits for the test to begin. The connection interval can also be configured (-ci<value>).

- thrput stop: stops the test and disconnects the devices.

Once a connection is established between the devices and initial throughput test is complete, one can start a new throughput transmission test with a new set of parameters (packet size / count).

The receiving device generates the report if no packets are received for more than three consecutive connection events.

The default configuration of the throughput test is the following:

- Packet count: 1000
- Payload size: 20 bytes

Connection interval (min, max): 160, 160 (200 ms)

The example of a test report is shown below:

```
BLE Shell>thrput start tx
BLE Shell>
-->  GAP Event: Advertising parameters successfully set.
BLE Shell>
-->  GAP Event: Advertising data successfully set.
BLE Shell>
-->  GAP Event: Advertising started.
-->  GAP Event: Connected to peer 0
BLE Shell>
-->  GAP Event: Advertising Throughput test started.
Sending packets...
-->  MTU Exchanged.
BLE Shell>
Throughput test with peer 0 has finished.

BLE Shell>thrput start rx
BLE Shell>
->  GAP Event: Scan started.
Found device:
THR_PER
0060375BCEC6
->  GAP Event: Scan stopped.
-->  GAP Event: Connected to peer 0
BLE Shell>Throughput test started.
Receiving packets...


************************************
***** TEST REPORT FOR PEER ID 0 ****
************************************
Packets received: 1000
Total bytes: 244000
Receive duration: 5017 ms
Average bitrate: 389 kbps
************************************
********** END OF REPORT ***********
```

Parent topic: Bluetooth LE Shell

Decision-Based Advertising Filtering (DBAF) feature

The Bluetooth LE Shell application also supports the Decision-Based Advertising Filtering (DBAF) feature, which can be enabled by performing the following steps:

- In app_preinclude.h file, set BLE_SHELL_DBAF_SUPPORT to 1.

This feature requires two devices:

- GAP Peripheral: transmits decision PDUs (ADV_DECISION_IND).
- GAP Central: scans for decision PDUs and handles filtering policies.

To showcase the functionality, two platforms are used in the following setup.

Steps to perform on the GAP Peripheral device:

1. Configure the extended advertising parameters to use decision PDUs. In the below example, the advertising type is set to connectable and includes TX power, uses decision PDUs and includes AdvA in the decision PDU. The primary PHY is set to Coded PHY.
2. Configure the extended advertising data using the gap extadvdecdata command. The resolvable tag and/or arbitrary data can be set using the parameters available.
3. Start extended advertising.

```
BLE Shell>gap extadvcfg -phy1 3 -type 449
BLE Shell>gap extadvdecdata -key 112233445566778899AABBCCDDEEFF00 -prand 5AC317 -decdata␣
↪6362 -datalen 2 -restag 0
BLE Shell>gap extadvstart
BLE Shell>
--> GAP Event: Extended Advertising parameters successfully set.
BLE Shell>
--> GAP Event: Extended Advertising data successfully set.
BLE Shell>
--> GAP Event: Extended Advertising Decision Data Setup Complete.
BLE Shell>
--> GAP Event: Advertising state changed successfully!
BLE Shell>
```

Steps to perform on the GAP Central device:

1. Set the scanning parameters to scan only decision PDUs. The scanning PHY is set to match the advertising PHY, in this case Coded PHY.
2. Set the connection parameters to use only decision PDUs and the connection initiating PHYs corresponding to the primary PHY on which the advertising is performed.

```
BLE Shell>gap scancfg -phy 4 -filter 12
BLE Shell>gap connectcfg -phy 4 -filter 2
--> Connection Parameters:
 --> Connection Interval: 200 ms
 --> Connection Latency: 0
 --> Supervision Timeout: 32000 ms
 --> Connecting PHYs: Coded
 --> Connection Filter Policy: 2
BLE Shell>
```

3. Add decision instructions using the gap adddecinstr command. A maximum of gMaxNumDecisionInstructions_c instructions can be added. If the set of instructions must be changed, the gap deldecinstr command deletes all current instructions.

```
BLE Shell>gap adddecinstr -group 1 -field 0 -criteria 1 -restagkey␣
↪112233445566778899AABBCCDDEEFF00
BLE Shell>gap adddecinstr -group 1 -field 6 -criteria 1 -advmode 6
BLE Shell>gap adddecinstr -group 0 -field 24 -criteria 1 -arbmask 00000000ffffff
BLE Shell>gap adddecinstr -group 0 -field 9 -criteria 1 -advacheck 2 -add1type 0 -add1 a6fb0d376000 -
↪add2type 0 -add2 a5fb0d376000
BLE Shell>gap adddecinstr -group 0 -field 7 -criteria 1 -rssimin -80 -rssimax 0
BLE Shell>gap adddecinstr -group 0 -field 8 -criteria 5 -lossmin 0 -lossmax 50
BLE Shell>
```

4. Set the decision instructions using the gap setdecinstr command. The instructions are used when listening for advertisements containing decision PDUs.
5. Start scanning and filter duplicates.
6. Connect to the desired device in the scanned devices list.

```
BLE Shell>gap setdecinstr
BLE Shell>
--> GAP Event: Decision Instructions Setup Complete.
BLE Shell>gap scanstart filter
BLE Shell>
-> GAP Event: Scan started.
BLE Shell>
--> GAP Event: Found device 0 : C4603770BCC5 -23 dBm
 Advertising Extended Data:
gap connect 0
BLE Shell>
-> GAP Event: Scan stopped.
BLE Shell>
--> GAP Event: Connected to peer 0
BLE Shell>
```

Parent topic: Bluetooth LE Shell

Periodic Advertising with Responses (PAwR) feature

The Bluetooth LE Shell application also supports the Periodic Advertising with Responses feature, which can be enabled by performing the following steps:

- In app_preinclude.h file, set BLE_SHELL_PAWR_SUPPORT to 1.

This feature requires two (or more) devices:

- GAP Broadcaster: transmits advertising PDUs at a fixed interval. (AUX_SYNC_SUBEVENT_IND)
- GAP Observer (one or more): scans for advertising PDUs and synchronizes with the advertising train. Sends responses PDUs in the synced subevents. (AUX_SYNC_SUBEVENT_RSP)

To showcase the functionality, two platforms are used in the following setup.

Steps to perform on the GAP Broadcaster device:

1. Configure the periodic advertising parameters. In the below example, a set of implicit values are used. There are 2 subevents per interval, each interval repeating at 1 second. There are 4 response slots configured per subevent, with the first response slot starting at 125ms from the advertising packet.

```
BLE Shell>gap periodiccfg

-->  Periodic Advertising Parameters:
   -->  Periodic Advertising Handle: 1
   -->  Periodic Advertising Interval: 2000 ms
   -->  Number of subevents: 2
   -->  Interval between subevents: 156.25 ms
   -->  Time between the advertising packet in a subevent and the first response slot: 125.0 ms
   -->  Time between response slots: 1.250 ms
   -->  Number of response slots: 4
BLE Shell>
-->  GAP Event: Periodic Advertising parameters successfully set.
```

   Alternatively, the command does accept all the mentioned parameters to be changed. The below command can be used to configure the exact same values.

```
BLE Shell>gap periodiccfg -numsubevents 2 -subint 125 -rspslotdelay 100 -rspslotspace 10 -numrspslot 4
BLE Shell>
-->  GAP Event: Periodic Advertising parameters successfully set.
```

2. Configure the subevent data on the advertiser for each subevent. Each instance of the gap periodicsubeventdata command sets each subevent data for the configured number of subevents. In this used example, the first command sets the data for the first subevent, the second command sets the data for the second subevent. The upper limit is the lowest of the following: SHELL_PER_ADV_MAX_NUM_SUBEVENTS from app_preinclude.h or the number of subevents set using gap periodiccfg.

```
BLE Shell>gap periodicsubeventdata 255 696E666F31
BLE Shell>gap periodicsubeventdata 255 696E666F32
```

When the last subevent data is set, any succeeding command will erase the previously set data for all subevents, and the data in the ongoing command will be set in the first subevent.

Alternatively, the command accepts more advertising data structures for one subevent, up to SHELL_EXT_ADV_DATA_MAX_AD_STRUCTURES.

```
BLE Shell>gap periodicsubeventdata 27 01CAFECAFECAFE 255 696E666F31
```

To manually clear the data set for all the currently set subevents, you may issue the following command:

```
BLE Shell>gap periodicsubeventdata -erase
```

3. Start extended advertising.

```
BLE Shell>gap extadvstart
BLE Shell>
-->  GAP Event: Extended Advertising parameters successfully set.

BLE Shell>
-->  GAP Event: Extended Advertising data successfully set.

BLE Shell>
-->  GAP Event: Advertising state changed successfully!
```

4. Start periodic advertising.

```
BLE Shell>gap periodicstart
BLE Shell>Periodic Advertising state changed successfully!
```

Steps to perform on the GAP Observer device:

1. Start scanning using the gap scanstart command. Optionally you can modify the scan parameters using the gap scancfg command.

```
BLE Shell>gap scanstart
BLE Shell>
->  GAP Event: Scan started.
```

2. Issue a periodic sync command with the target address of the Broadcaster device that is periodic advertising. You can use the gap address command on the Broadcaster to obtain the address.

```
BLE Shell>gap periodicsync -peer 006037A55E86
```

When the sync is established, events will be printed at the console.

```
-->  GAP Event: Periodic Advertising Sync Established
-->  GAP Event: Periodic Device Scanned
       Event Counter: 8
       Subevent Synced: 0
       RSSI: -38 dBm
       Advertising Data: info1
```

3. Synchronize the Observer device with the subevents needed. For this example, we will synchronize the Observer with both subevents using the following command. Implicitly, the Observer synchronizes to subevent 0.

```
BLE Shell>gap periodicsyncsubevent -peradvproperties 0 -numsubevents 2 -subevents 0x00,0x01
...
BLE Shell>Set Sync Subevent command successfully completed.
```

After this, GAP Events of type Periodic Device Scanned should be displayed in the console for each of the subevents synchronized.

```
-->  GAP Event: Periodic Device Scanned
       Event Counter: 169
       Subevent Synced: 0
       RSSI: -34 dBm
       Advertising Data: info1
-->  GAP Event: Periodic Device Scanned
       Event Counter: 169
       Subevent Synced: 1
       RSSI: -38 dBm
       Advertising Data: info2
```

4. The last step to have a complete exchange of data is to set the response(s) on the Observer device. The next 2 commands will set the data for the 2 subevents synced at the previous step.

```
BLE Shell>gap periodicresponsedata -responsesubevent 0 -responseslot 0 -data 255␣
↪64657669636531696E666F31
BLE Shell>gap periodicresponsedata -responsesubevent 1 -responseslot 0 -data 255␣
↪64657669636531696E666F32
```

The -responsesubevent parameter indicates the subevent the data is set for. The -responseslot parameter indicates the response slot the data is sent at this value should be within the range 0 to the number of response slots set at periodiccfg.

To scale the demo to use more Observer devices, repeat steps 1-3 for each of the added device. Also use example at step 4, with the adaptation of the -responseslot according to the device index. The demo is designed to have a new device for each of the response slots available. Also, change the data to reflect the response of the extra device.

Note: *The* -data *parameter must be the last of the* periodicresponsedata *command as it can contain a volatile number of advertising data structures.*

Alternatively, the command may contain additional advertising data structures.

```
BLE Shell>gap periodicresponsedata -responsesubevent 0 -responseslot 0 -data 27␣
↪01CAFECAFECAFE 255 64657669636531696E666F31
BLE Shell>gap periodicresponsedata -responsesubevent 1 -responseslot 0 -data 27␣
↪01CAFECAFECAFE 255 64657669636531696E666F32
```

To manually clear the data set for all the currently set subevents, you may issue the following command:

```
BLE Shell>gap periodicresponsedata -erase
```

Initiate a connection from PAwR

PAwR feature allows for connection initiation from the Broadcaster device. To do this, the devices have to be synchronized. The minimum steps required for this are steps 1, 3, 4 on the GAP Broadcaster device and steps 1, 2, GAP Observer device.

After performing the mentioned steps, the Broadcaster device must issue the following command:

```
BLE Shell>gap connectpawr -subevent 0 -peer 00603728063E -peeraddrtype 0
```

Upon successful connection the following message will be displayed.

```
-->  GAP Event: Connected to peer 0
```

Parent topic: Bluetooth LE Shell

Monitoring Advertisers (MonAdv) feature

The Bluetooth LE Shell application also supports the Monitoring Advertisers feature, which can be enabled by performing the following steps:

- In app_preinclude.h file, set BLE_SHELL_MONADV_SUPPORT to 1.

This feature requires two (or more) devices:

- GAP Broadcaster (one or more): transmits advertising PDUs.
- GAP Observer: scans for advertising PDUs and shall generate an event if, during the timeout period set by the Application, an advertisement has not been received from that peer with an RSSI value above the RSSI low threshold set by the Application.

To showcase the functionality, two platforms are used in the following setup.

Steps to perform on the GAP Broadcaster device:

1. Get the device address of the target device to be monitored:

```
BLE Shell>gap address

-->  GAP Event: Random Static Address:D7E448669EA4
```

Note the device address to be used later, at the scanner device.

2. Configure advertising parameters and start advertising.

For this example we will set advertising at 100 ms interval.

```
BLE Shell>gap advcfg -interval 100
BLE Shell>
-->  GAP Event: Advertising parameters successfully set.
```

```
BLE Shell>gap advstart
BLE Shell>
-->  GAP Event: Advertising parameters successfully set.

BLE Shell>
-->  GAP Event: Advertising data successfully set.

BLE Shell>
-->  GAP Event: Advertising state changed successfully!
```

Steps to perform on the GAP Observer device:

1. Using the device address previously obtained, add it to the Monitoring Advertisers list.

```
BLE Shell>gap monadvadd -addr D7E448669EA4 -type 1 -rssilow -65 -rssihigh -55 -timeout 1
BLE Shell>
-->  GAP Event: Device added to Monitored Advertisers List.
```

- The -type and -addr parameters are used to identify an advertising device.
- The -rssilow parameter is set to the RSSI value below which a Monitored Advertisers Report event shall be generated when the associated timer expires for that device.
- The -rssihigh parameter is set to the RSSI value equal to or above which a Monitored Advertisers Report event may be triggered for that device.
- The -timeout parameter is set to the timeout time in seconds for the device.

2. Start the scanning using the default, predefined parameters:

```
BLE Shell>gap scanstart
BLE Shell>
->  GAP Event: Scan started.

BLE Shell>
-->  GAP Event: Found device 0 : 18C23C3C23C1 -64 dBm
-->  GAP Event: Found device 1 : 7A7410BFD06C -55 dBm
-->  GAP Event: Found device 2 : 18C23C3C23CD -61 dBm
...
```

3. Enable monitoring advertisers.

```
BLE Shell>gap monadven -enable 1
BLE Shell>
-->  GAP Event: GAP Event: Monitoring Advertisers enabled.
```

If the devices are distant to each other, the event message should be printed at the console:

```
BLE Shell>
-->  GAP Event: Monitored Advertiser Report
    Address Type: 1
    Address: D7E448669EA4
    Condition: RSSI Low Threshold
```

If the devices are brought close together, the event message should be printed at the console:

```
-->  GAP Event: Monitored Advertiser Report
    Address Type: 1
    Address: D7E448669EA4
    Condition: RSSI High Threshold
```

To restart the Monitoring Advertisers reporting

Reuse the same command and same parameters as the enable command.

```
BLE Shell>gap monadven -enable 1
BLE Shell>
-->  GAP Event: Monitoring Advertisers restarted.
```

To stop the Monitoring Advertisers reporting

```
BLE Shell>gap monadven -enable 0
BLE Shell>
-->  GAP Event: Monitoring Advertisers disabled.
```

To remove a device from the Monitoring Advertisers list:

```
BLE Shell>gap monadvrem -addr D7E448669EA4 -type 1
BLE Shell>
-->  GAP Event: Device removed from Monitored Advertisers List.
```

To clear the Monitoring Advertisers list of the added devices:

Run the following command:

```
BLE Shell>gap monadvclear
BLE Shell>
-->  GAP Event: Monitored Advertisers List cleared.
```

To get the size of the Monitoring Advertisers:

Run the following command:

```
BLE Shell>gap monadvsize
```

The size should be printed as indicated in this output message:

```
BLE Shell>
-->  GAP Event: Monitored Advertisers List size: 8
```

Parent topic: Bluetooth LE Shell

Connection Subrating feature

The Bluetooth LE Shell application also supports the Connection Subrating feature, which can be enabled by performing the following steps:

- In app_preinclude.h file, set BLE_SHELL_CONN_SBR_SUPPORT to 1.

This feature requires two connected devices to be demonstrated. There are three ble_shell commands implemented to handle this feature.

Connection subrating configuration

- gap sbrcfg does not require a connection to be established.
- gap sbrcfg can be issued without any parameters to return the default configuration.

```
  BLE Shell>gap sbrcfg

-->  Connection Subrate Parameters:
   -->  subrateMin: 4
   -->  subrateMax: 4
   -->  Connection Latency: 0
   -->  continuationNumber: 2
   -->  Supervision Timeout: 32000 ms
BLE Shell>
```

- optional parameters can be used to configure connection subrating: gap sbrcfg[-sbrmin minSubrateFactor] [-sbrmax maxSubrateFactor] [-latency latency] [-contnum continuationNumber] [-timeout timeout in ms]

```
 BLE Shell>gap sbrcfg -sbrmin 4 -sbrmax 4 -latency 0 -contnum 2 -timeout 32000

-->  Connection Subrate Parameters:
   -->  subrateMin: 4
   -->  subrateMax: 4
   -->  Connection Latency: 0
   -->  continuationNumber: 2
   -->  Supervision Timeout: 32000 ms
BLE Shell>
```

Set default connection subrating parameters

- gap setdefsbrparam does not require a connection to be established.
- It can be issued only on the central side.
- It sets the parameters exposed/set by the gap sbrcfg command.

```
BLE Shell>gap setdefsbrparam
BLE Shell>
-->  GAP Event: Default Connection Subrate Parameters Setup Complete.

BLE Shell>
```

Connection subrate request

- gap connsbrreq peerID requires a connection to be established and the peerId as parameter.
- It can be issued on the central or on the peripheral side.
- When the connection subrating is applied the subrate change event is received.

```
LE Shell>gap connsbrreq 0
BLE Shell>
-->  GAP Event: Subrate Changed 0
   -->  subrateFactor: 4
   -->  peripheralLatency: 0
   -->  continuationNumber: 2
   -->  supervisionTimeout: 32000 ms
BLE Shell>
```

Usage

To showcase the functionality, the throughput feature can be used.

1. Configure the desired subrating configurations parameters on both central and peripheral.

```
BLE Shell>gap sbrcfg -sbrmin 4 -sbrmax 4 -latency 0 -contnum 2 -timeout 32000

-->  Connection Subrate Parameters:
   -->  subrateMin: 4
   -->  subrateMax: 4
   -->  Connection Latency: 0
   -->  continuationNumber: 2
   -->  Supervision Timeout: 32000 ms
BLE Shell>
```

2. Set the default connection subrating parameters on central.

```
BLE Shell>gap setdefsbrparam
BLE Shell>
-->  GAP Event: Default Connection Subrate Parameters Setup Complete.

BLE Shell>
```

3. Start the throughput tx on peripheral.

```
BLE Shell>thrput start tx
BLE Shell>
-->  GAP Event: Advertising parameters successfully set.

BLE Shell>
-->  GAP Event: Advertising data successfully set.

BLE Shell>
-->  GAP Event: Advertising started.>
```

4. Start the throughput rx on central.

```
BLE Shell>thrput start rx
BLE Shell>
-> GAP Event: Scan started.

Found device:
THR_PER
006037327E67
-> GAP Event: Scan stopped.

--> GAP Event: Connected to peer 0
BLE Shell>Throughput test started.
Receiving packets...

************************************
***** TEST REPORT FOR PEER ID 0 ****
************************************

Packets received: 1000
Total bytes: 244000
Receive duration: 2613 ms
Average bitrate: 747 kbps

************************************
********** END OF REPORT ***********
************************************
```

5. Request connection subrating from the peripheral. The subrate change event should be observed on both boards.

```
BLE Shell>gap connsbrreq 0
BLE Shell>
--> GAP Event: Subrate Changed 0
  --> subrateFactor: 4
  --> peripheralLatency: 0
  --> continuationNumber: 2
  --> supervisionTimeout: 32000 ms
BLE Shell>
```

6. Start the throughput rx on central again .

```
thrput start 0 rx
Throughput test started.
Receiving packets...
BLE Shell>
```

7. Start the throughput tx on peripheral again .

```
thrput start 0 tx
Throughput test started.
Sending packets...
```

8. When the test is finished the test report should be printed on the central side.

```
************************************
***** TEST REPORT FOR PEER ID 0 ****
************************************

Packets received: 1000
Total bytes: 244000
Receive duration: 2645 ms
Average bitrate: 737 kbps
```

(continues on next page)

(continued from previous page)

```
************************************
********** END OF REPORT ***********
************************************
```

As can be observed the throughput with the subrating applied is almost similar with the throughput without subrating while the energy efficiency when no data is being transmitted is improved given the effective connection interval is subrate factor bigger.

Parent topic: Bluetooth LE Shell

LE Power Control feature

The Bluetooth LE Shell application supports the LE Power Control feature, which allows monitoring and control of transmit power levels and path loss reporting on active connections. This feature is available on platforms supporting Bluetooth Core Specification 5.2 or later.

- In app_preinclude.h file, ensure the following defines are set:
  - gBLE52_d set to 1 (BLE 5.2 support)
  - gBLE52_LePowerControlSupport_d set to TRUE
  - gHostInitEnableExpmFeatures_c set to 1

Overview

LE Power Control provides the following capabilities:

- Read local and remote transmit power levels for different PHY types
- Monitor path loss between devices with configurable thresholds
- Receive notifications when power levels or path loss changes
- Configure path loss reporting parameters

This feature requires an active connection between two BLE devices.

Shell Commands

Enhanced Read Transmit Power Level

Reads the current and maximum transmit power levels of the local Controller on the ACL connection identified by the Connection_Handle parameter and the PHY indicated by the PHY parameter.

Command: gap pwrread <peerID> <phy>

Parameters:

- peerID: The peer device ID (0-based index)
- phy: PHY type:
  - 1 = LE 1M PHY (gPowerControlLePhy1M_c)
  - 2 = LE 2M PHY (gPowerControlLePhy2M_c)
  - 3 = LE Coded PHY with S=8 data coding (gPowerControlLePhyCodedS8_c)
  - 4 = LE Coded PHY with S=2 data coding (gPowerControlLePhyCodedS2_c)

Example:

```
BLE Shell>gap pwrread 0 1
BLE Shell>
-->  GAP Event: Enhanced Read Transmit Power Level
     PHY: LE 1M PHY
     Current TX Power: 0 dBm
     Max TX Power: 10 dBm
BLE Shell>
```

Read Remote Transmit Power Level

Reads the transmit power level used by the remote Controller on a specified PHY on an ACL connection.

Command: gap pwrremote <deviceId> <phy>

Parameters:

- deviceId: The peer device ID (0-based index)
- phy: PHY type (same values as above)

Example:

```
BLE Shell>gap pwrremote 0 1
BLE Shell>
-->  GAP Event: Transmit Power Reporting
     Reason: Read Remote TX Power completed
     PHY: LE 1M PHY
     TX Power Level: -11 dBm
     Delta: 0 dB
     Flags: [Normal Range]
BLE Shell>gap pwrremote 0 1
BLE Shell>
-->  GAP Event: Transmit Power Reporting
     Reason: Read Remote TX Power completed
     PHY: LE 1M PHY
     TX Power Level: 10 dBm
     Delta: 0 dB
     Flags: [At Maximum]
```

Set Path Loss Reporting Parameters

Configures the thresholds and parameters for path loss monitoring.

Command: gap pathlossparams <deviceId> <highThreshold> <highHysteresis> <lowThreshold> <lowHysteresis> <minTimeSpent>

Parameters:

- deviceId: The peer device ID (0-based index)
- highThreshold: High path loss threshold in dB
- highHysteresis: Hysteresis value for high threshold in dB
- lowThreshold: Low path loss threshold in dB
- lowHysteresis: Hysteresis value for low threshold in dB
- minTimeSpent: Minimum time to spend in a zone before reporting, in connection events

Example:

```
BLE Shell>gap pathlossparams 0 40 10 20 10 10
BLE Shell>
-->  GAP Event: Path Loss Reporting Parameters Setup Complete
```

(continues on next page)

(continued from previous page)

```
    Parameters configured successfully!
BLE Shell>
```

Enable Path Loss Reporting

Enables or disables path loss reporting on an active connection.

Command: gap pathlossenable <deviceId> <enable>

Parameters:

- deviceId: The peer device ID (0-based index)
- enable:
  - 0 = Disable path loss reporting
  - 1 = Enable path loss reporting

Example:

```
BLE Shell>gap pathlossenable 0 1
BLE Shell>
-->  GAP Event: Path Loss Reporting State Changed
    Path loss reporting is now enabled!
BLE Shell>
```

Once enabled, path loss threshold events will be reported:

```
BLE Shell>
-->  GAP Event: Path Loss Threshold HIGH zone entered
    Current Path Loss: 48 dB
BLE Shell>
```

Enable Transmit Power Reporting

Enables or disables transmit power change reporting.

Command: gap pwrenable <deviceId> <localEnable> <remoteEnable>

Parameters:

- deviceId: The peer device ID (0-based index)
- localEnable:
  - 0 = Disable local TX power reporting
  - 1 = Enable local TX power reporting
- remoteEnable:
  - 0 = Disable remote TX power reporting
  - 1 = Enable remote TX power reporting

Example:

```
BLE Shell>gap pwrenable 0 1 1
BLE Shell>
-->  GAP Event: Transmit Power Reporting State Changed
    Transmit power reporting enabled!
BLE Shell>
```

When enabled, power change events will be reported:

```
BLE Shell>
-->  GAP Event: Transmit Power Reporting
     Reason: Remote TX power changed
     PHY: LE 1M PHY
     TX Power Level: -5 dBm
     Delta: +3 dB
```

Events

Enhanced Read Transmit Power Level

Returned in response to the `gap pwrread` command.

Event fields:

- PHY: The PHY type (1M, 2M, Coded S=8, Coded S=2)
- Current TX Power: Current transmit power level in dBm (-127 to 20, or "Unavailable")
- Max TX Power: Maximum transmit power level in dBm (-127 to 20)

Path Loss Reporting Parameters Setup Complete

Confirms that path loss reporting parameters have been configured successfully.

Path Loss Reporting State Changed

Confirms that path loss reporting has been enabled or disabled.

Path Loss Threshold

Generated when path loss crosses a configured threshold and stays in the new zone for the minimum time.

Event fields:

- Current Path Loss: The measured path loss in dB
- Zone Entered:
  - `LOW`: Path loss is below low threshold (good signal)
  - `MID`: Path loss is between thresholds (moderate signal)
  - `HIGH`: Path loss is above high threshold (weak signal)

Transmit Power Reporting State Changed

Confirms that transmit power reporting has been enabled or disabled.

Transmit Power Reporting

Generated when local or remote transmit power changes, or in response to a remote power level read request.

Event fields:

- Reason:
  - `Local TX power changed`: Local controller changed TX power
  - `Remote TX power changed`: Remote device changed TX power
  - `Read Remote TX Power completed`: Response to `gap pwrremote` command
- PHY: The PHY type
- TX Power Level: Power level in dBm (-127 to 20)
- Delta: Change in power level in dB since last report (with +/- sign)
- Flags (for remote read only):

- [At Minimum]: Displayed when at minimum power level
- [At Maximum]: Displayed when at maximum power level
- [Normal Range]: Displayed when between min and max

Usage Example

The following example demonstrates path loss monitoring between two devices: First we create the connection between the devices. **On Device 1 (Peripheral):**

1. Start advertising:

```
BLE Shell>gap devicename HRS
-->  GATTDB Event: Attribute Written
HRS>gap advdata 1 6
HRS>gap advdata 8 HRS
HRS>gap advstart
HRS>
-->  GAP Event: Advertising parameters successfully set.
HRS>
-->  GAP Event: Advertising data successfully set.
HRS>
-->  GAP Event: Advertising state changed successfully!
HRS>gattdb addservice 0x180D
--> Heart Rate
    - Heart Rate Measurement  Value Handle: 14
      - CCCD Handle: 15
    - Body Sensor Location  Value Handle: 17
    - Heart Rate Control Point  Value Handle: 19
-->  GATTDB Event: Service Added in database.
```

On Device 2 (Central):

2. Scan and connect:

```
BLE Shell>gap devicename Collector
-->  GATTDB Event: Attribute Written
Collector>gap scanstart filter
Collector>
-->  GAP Event: Scan started.
Collector>
-->  GAP Event: Found device 0 : F697CFBF6A17 -93 dBm
-->  GAP Event: Found device 1 : HRS 006037CFEAE9 -29 dBm
Collector>gap connect 1
Collector>
-->  GAP Event: Scan stopped.
Collector>
-->  GAP Event: Connected to peer 0
```

3. Configure path loss parameters:

```
Collector>gap pathlossparams 0 40 10 20 10 10
Collector>
-->  GAP Event: Path Loss Reporting Parameters Setup Complete
    Parameters configured successfully!
```

4. Enable path loss reporting:

```
Collector>gap pathlossenable 0 1
Collector>
-->  GAP Event: Path Loss Reporting State Changed
    Path loss reporting is now enabled!
```

5. Monitor path loss (move devices to change signal strength, you will get a new notification when the path loss crosses a threshold and stays in the new zone for the minimum time):

```
Collector>
-->  GAP Event: Path Loss Threshold MID zone entered
     Current Path Loss: 30 dB
Collector>
-->  GAP Event: Path Loss Threshold LOW zone entered
     Current Path Loss: 18 dB
Collector>
-->  GAP Event: Path Loss Threshold HIGH zone entered
     Current Path Loss: 42 dB
```

6. Read current transmit power:

```
Collector>gap pwrread 0 1
Collector>
-->  GAP Event: Enhanced Read Transmit Power Level
     PHY: LE 1M PHY
     Current TX Power: -20 dBm
     Max TX Power: 10 dBm
```

7. Enable transmit power reporting:

```
Collector>gap pwrenable 0 1 1
Collector>
-->  GAP Event: Transmit Power Reporting State Changed
     Transmit power reporting enabled!
```

8. Monitor transmit power changes (move devices or wait for controller to adjust power):

```
Collector>
-->  GAP Event: Transmit Power Reporting
     Reason: Remote TX power changed
     PHY: LE 1M PHY
     TX Power Level: 1 dBm
     Delta: -3 dB
Collector>
-->  GAP Event: Transmit Power Reporting
     Reason: Remote TX power changed
     PHY: LE 1M PHY
     TX Power Level: -2 dBm
     Delta: -3 dB
Collector>
-->  GAP Event: Transmit Power Reporting
     Reason: Local TX power changed
     PHY: LE 1M PHY
     TX Power Level: -20 dBm
     Delta: -3 dB
```

Notes

- LE Power Control requires Bluetooth 5.2 or later support in both the host stack and the controller firmware
- The path loss zones (LOW/MIDDLE/HIGH) help applications make link adaptation decisions

Parent topic: Bluetooth LE Shell

Parent topic:*Bluetooth LE stack and demo applications*

Hybrid (Dual-mode) Bluetooth Low Energy and Generic FSK

The Hybrid (Dual-mode) Bluetooth Low Energy and Generic FSK application demonstrates Generic FSK transmission/reception and Bluetooth advertising/scanning/multiple connections coexistence.

The Bluetooth LE part of this demo implements a modified version of the Wireless UART demo application, capable of multiple Bluetooth LE connections.

Based on the Hardware link-layer implementation, the Bluetooth Low Energy has a higher priority than the Generic FSK protocol and as the effect, the Generic FSK communication is executed during the Idle states (inactive periods) of the Bluetooth LE. The coexistence of the two protocols is handled internally at the Controller level.

The Bluetooth LE part of the application behaves at first as a GAP central node. It enters GAP Limited Discovery Procedure and searches for other Wireless UART devices to connect. To change the device role to GAP peripheral, use the **ROLESW** button. The device enters GAP General Discoverable Mode and waits for a GAP central node to connect.

The Generic FSK part of the application can either enter in the receive state by double clicking the **SCANSW** button or it can start the periodic transmit by long pressing the **ROLESW** button. It cannot enter in both states at the same time.

The Generic FSK has lower priority than the Bluetooth LE. Therefore, any ongoing Generic FSK receive is paused by the Controller when Bluetooth LE activity is ongoing. The reception is automatically resumed by the Controller when there is no Bluetooth LE activity.

When using Generic FSK, packet loss should be expected. The Generic FSK communication does not offer an acknowledgement mechanism to guarantee success for packet transmission, therefore this communication method does not offer reliability. If the user wants to improve reliability, packet repetition may be used, or the user may implement a custom protocol at application level.

The first Generic FSK transmit command is buffered if there is continuous Bluetooth LE activity (for example, for continuous Bluetooth LE scanning). Any succeeding Generic FSK transmit command indicates failure in the command line interface, if the initial buffered transmit command was not sent yet.

This section describes the implemented profiles and services, user interactions, and testing methods for the Hybrid (Dual-Mode) Bluetooth Low Energy and Generic FSK application.

Implemented profile and services

The Hybrid (Dual-Mode) Bluetooth Low Energy and Generic FSK application implements the GATT client and server for the custom Wireless UART profile and services. It also acts as a Generic FSK transmitter/receiver, repeating a custom packet, at a fixed periodic interval. It uses a predefined identifier, isolated to the address used in the Bluetooth LE protocol of the demo.

- Wireless UART Service (UUID: 01ff0100-ba5e-f4ee-5ca1-eb1e5e4b1ce0)
- Battery Service v1.0
- Device Information Service v1.1

The Wireless UART service is a custom service that implements a custom writable ASCII Char characteristic (UUID: 01ff0101-ba5e-f4ee-5ca1-eb1e5e4b1ce0) that holds the character written by the peer device.

The application is ready to start either Bluetooth LE scanning for Wireless UART Service, Bluetooth LE advertising Wireless UART Service, Generic FSK periodic transmit of a custom packet or Generic FSK receive, in the available slots not used by the Bluetooth LE protocol.

Parent topic: Hybrid (Dual-mode) Bluetooth Low Energy and Generic FSK

Supported platforms

The following platforms support the Hybrid (dual-mode) Bluetooth Low Energy and Generic FSK application:

- KW45B41Z-EVK
- KW45B41Z-LOC
- KW47-EVK
- KW47-LOC
- KW43-LOC
- FRDM-KW43

Parent topic: Hybrid (Dual-mode) Bluetooth Low Energy and Generic FSK

User interface

After flashing the board, the device is in idle mode (all LEDs flashing). To start Bluetooth LE scanning, press the **SCANSW** button. When in GAP Limited Discovery Procedure of GAP General Discoverable Mode, **CONNLED** is flashing. When the node connects to a peer device, **CONNLED** turns solid. To disconnect the node, hold the **SCANSW** button pressed for 2-3 seconds. The node then re-enters GAP Limited Discovery Procedure.

To start Generic FSK receiving, double click the **SCANSW** button and the device starts receiving packets on the same channel as the Bluetooth LE channel 37, with the network address defined in `gGenFSK_NetworkAddress_c`. The receiver only prints the packets that have the predefined identifier at `gGenFSK_Identifier_c`. To stop Generic FSK reception, double click the **ROLESW** button.

On a different device, start Generic FSK transmit by long pressing the **ROLESW** button. To stop the periodic Generic FSK transmit, long press again the **ROLESW** button, if the transmit procedure is ongoing. The long press **ROLESW** acts as a toggle for the transmit.

Parent topic: Hybrid (Dual-mode) Bluetooth Low Energy and Generic FSK

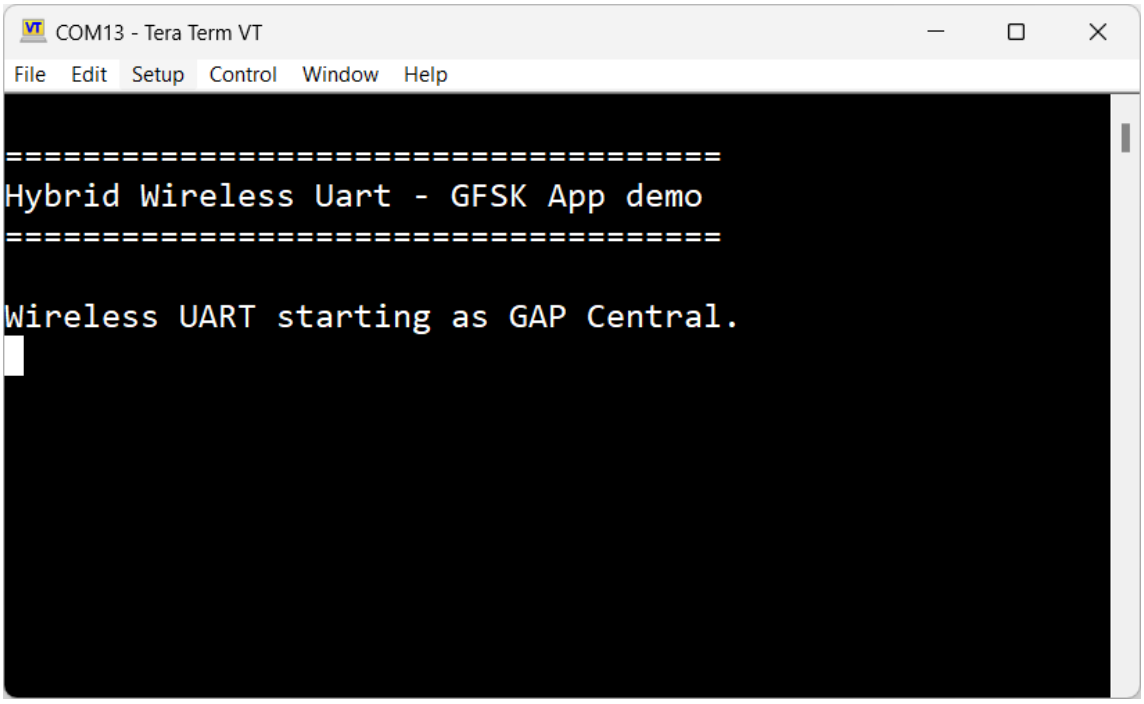
Usage

The application is built to work with another supported platform running the same example. When testing with two boards, perform the following steps:

Case 1 (Bluetooth LE)

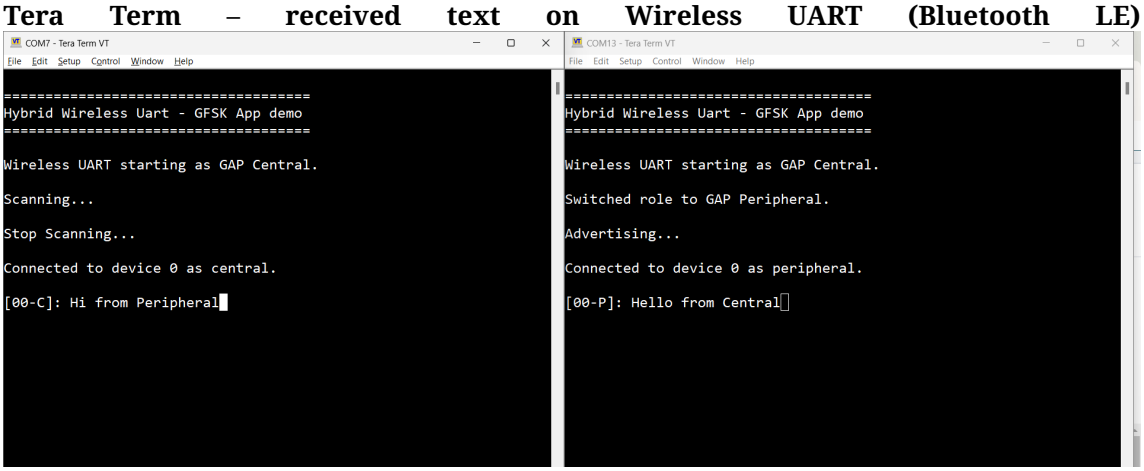
1. Open a serial port terminal and connect to the two boards, in the same manner described in Testing devices. The figure below displays the start screen after the board is reset.

   Tera Term – Hybrid Wireless UART (Bluetooth LE) - Generic FSK start screen



2. The application starts as a GAP central. To switch the role to a GAP peripheral, press the **ROLESW** button. Depending on the role, when pressing the **SCANSW** button, the application starts either scanning or advertising.
3. As soon as the **CONNLED** turns solid on both devices, the user can start writing in one of the consoles. The text appears on the other terminal.
4. After creating a connection, the role (central or peripheral) is displayed on the console. The role switch can be pressed again before creating a new connection. An output example can be observed in the figure below.

Tera Term – received text on Wireless UART (Bluetooth LE)



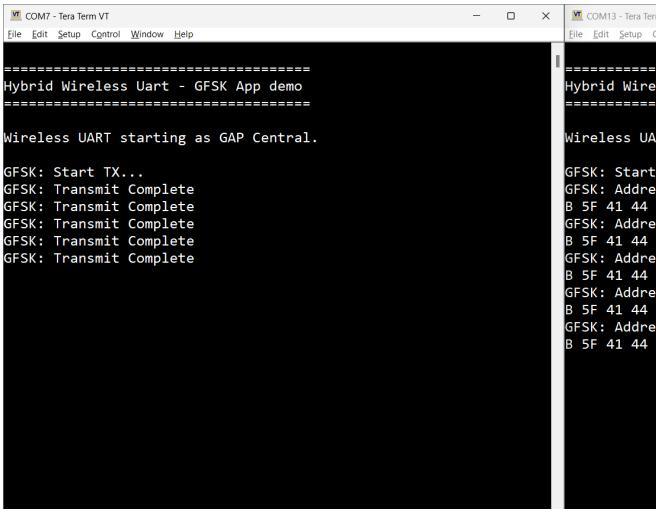
Parent topic: Usage

Case 2 (Generic FSK)

When operating the Generic FSK mode, the following steps should be followed.

1. Open a serial port terminal and connect to the two boards, in the same manner described in Testing devices. The start screen after the board is reset is the same as in Figure.
2. The Generic FSK communication direction is not preset. To start receiving, double click the **SCANSW** button on one board. Then, the device starts receiving packets on the same channel as the Bluetooth LE channel 37.

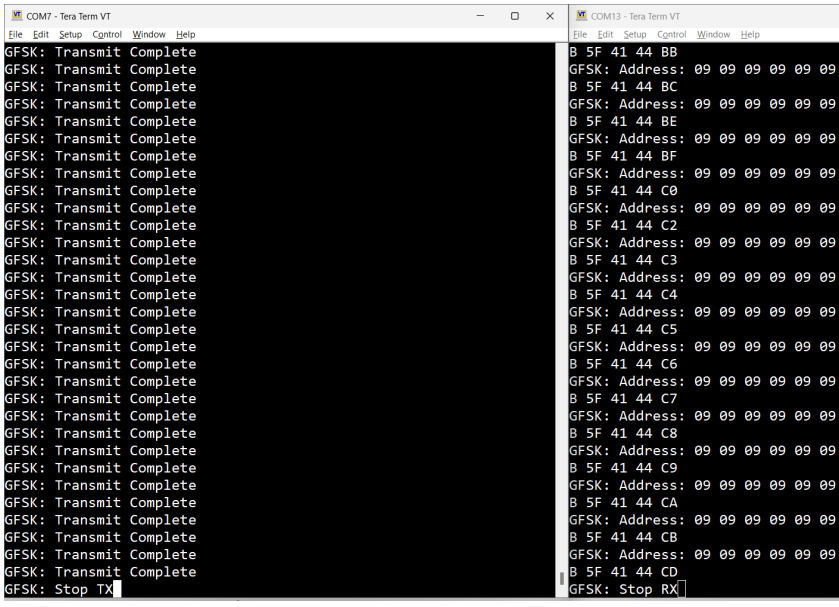
3. To start transmitting, long press the **ROLESW** button on the other board. The transmitting device uses an identifier known by the receiver and its packets are displayed in the CLI as shown in Figure:



Teraterm – packet transmit/receive on Generic FSK

4. To stop Generic FSK reception, double click the **ROLESW** button.
5. To stop the periodic Generic FSK transmit operation, long press the **ROLESW** button again, if the transmit procedure is ongoing. The long press of the **ROLESW** button acts as a toggle for the transmit. At this point, both devices can reverse the direction of communication by following the exact same steps.

   See Figure.



Tera Term – stop Generic FSK activity

Parent topic: Usage

Case 3 (Dual-mode)

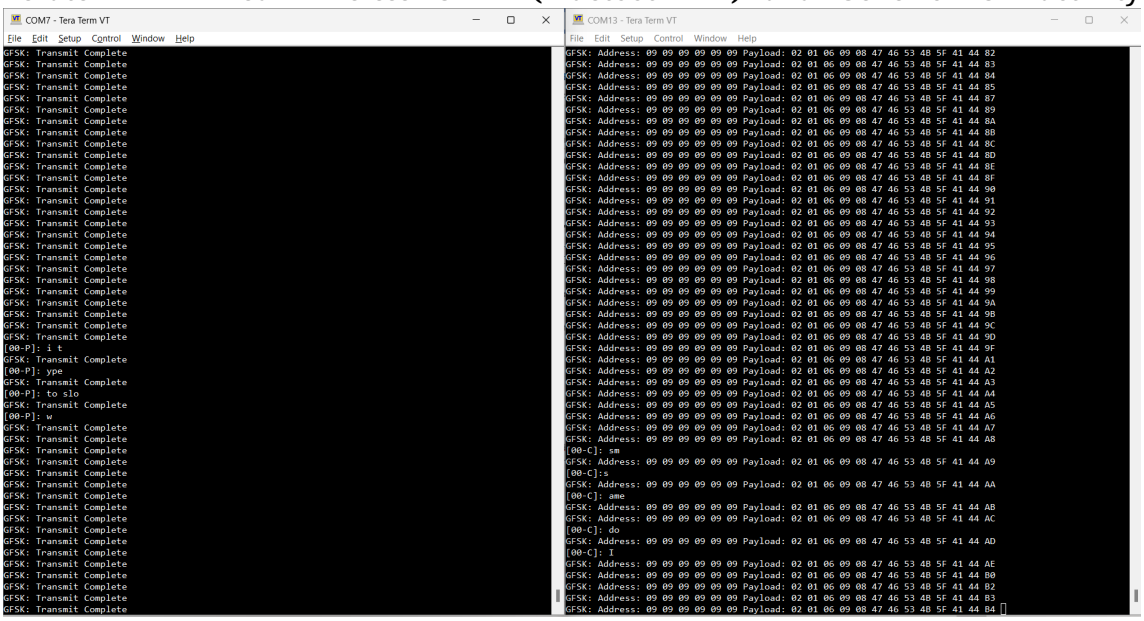
The application is ready to combine both scenarios. It can either:

- Establish a Bluetooth LE connection and perform Generic FSK activity.
- During Generic FSK activity, the Bluetooth LE connection can be established and the Controller pauses receiving or announcing the discarded Generic FSK transmissions. This activity is resumed after the Bluetooth LE activity is finished.

To run the Dual-mode scenario, follow the steps below:

1. Establish a Bluetooth LE connection as described in Case 1 (Bluetooth LE).
2. Start Generic FSK activity as described in Case 2 (Generic FSK), independent of the Bluetooth LE roles chosen in Case 1.
3. Start typing in either of the terminals. As seen in Figure, the Bluetooth LE activity is prioritized. In this step, characters are printed in the peer's terminal and Generic FSK continues in the available slots, not used by the Bluetooth LE link.

Teraterm – Mixed Wireless UART (Bluetooth LE) and Generic FSK activity



Parent topic: Usage

Parent topic: Hybrid (Dual-mode) Bluetooth Low Energy and Generic FSK

Customization

Use the steps below to change the default settings of this demo.

For Bluetooth LE, the default advertising config (gAdvParams) parameter is found in the app_config.c file. Also, the scanning parameters (gScanParams) can be found in this file.

The maximum number of connections supported by the Bluetooth LE, for this application, is defined in the app_preinclude.h file, with the macro name gAppMaxConnections_c.

Note: The Generic FSK protocol is active during the inactive periods of the Bluetooth LE protocol. The demo is currently configured to have the scan window equal to the scan interval to make the user aware of this, but this can be changed.

For Generic FSK, the following defines of interest can be found in genfsk_app.h, described below:

Name	Description
gGen-FSK_Netwc	This is the network address used for the Generic FSK, in the transmitter payload. It is implicitly set to the 0x8E89BED6, but this can be reconfigured. Ensure that it is also changed on the receiver in the hybrid_gfsk.c controller file.
gGen-FSK_H0Val	H0 value is used in the header.
gGen-FSK_Identi	This is the identifier used by the transmitter to be filtered at the receiver. The current implementation filters the Generic FSK packets received, based on this define.
gGenF-skApp_TxIn	This is the interval the transmitter will repeat the transmission of a packet. It is set in milliseconds.

Files of interest

The demo can be found in the w_uart_genfsk from the available examples.

The demo is based on the basic Wireless UART with the addition of some Generic FSK files required for working in dual-mode, described below.

File name	Description
genfsk_ap c	Application common module. Handles the HCI commands and events for the Generic FSK. Sends the events to the application.
genfsk_ap h	Application common module. Exposes public functions.
hybrid_gfs c	Controller common module. Handles initialization of Generic FSK.
hybrid_gfs h	Controller common module. Exposes public functions.

Parent topic: Hybrid (Dual-mode) Bluetooth Low Energy and Generic FSK

Parent topic:*Bluetooth LE stack and demo applications*

Wireless UART Host

This section describes the implemented profiles and services, user interactions, and testing methods for the Wireless UART Host application.

The Wireless UART Host application is a demonstration of the Extended NBU architectural concept, where the Bluetooth LE Host Stack and the Link Layer both run on the NBU core, while the user-facing application runs on the Application core. The communication between the application and the Host is done via the FSCI protocol running on the inter-core RPMSG transport.

The Wireless UART Host application must be flashed together with the NCP FSCI Blackbox application.

Implemented profile and services

The Wireless UART Host application implements both the GATT client and server for the custom Wireless UART profile and services.

- Wireless UART Service (UUID: 01ff0100-ba5e-f4ee-5ca1-eb1e5e4b1ce0)
- Battery Service v1.0
- Device Information Service v1.1

The Wireless UART service is a custom service that implements a custom writable ASCII Char characteristic (UUID: 01ff0101-ba5e-f4ee-5ca1-eb1e5e4b1ce0) that holds the character written by the peer device.

The application behaves at first as a GAP central node. It enters GAP Limited Discovery Procedure and searches for other Wireless UART devices to connect. To change the device role to GAP peripheral, use the **ROLESW** button. The device enters GAP General Discoverable Mode and waits for a GAP central node to connect.

Parent topic: Wireless UART Host

Supported platforms

The following platforms support the Wireless UART Host application:

- MCX-W72-EVK
- FRDM-MCXW72

Parent topic: Wireless UART Host

User interface

Ensure that both NCP FSCI Blackbox (for the NBU core) and Wireless UART Host (for the Application core) have been flashed to the board.

After flashing the board, the device is in idle mode (all LEDs flashing). To start scanning, press the **SCANSW** button. When in GAP Limited Discovery Procedure of GAP General Discoverable Mode, **CONNLED** is flashing. When the node connects to a peer device, **CONNLED** turns solid. To disconnect the node, hold the **SCANSW** button pressed for 2-3 seconds. The node then re-enters GAP Limited Discovery Procedure.

Parent topic: Wireless UART Host

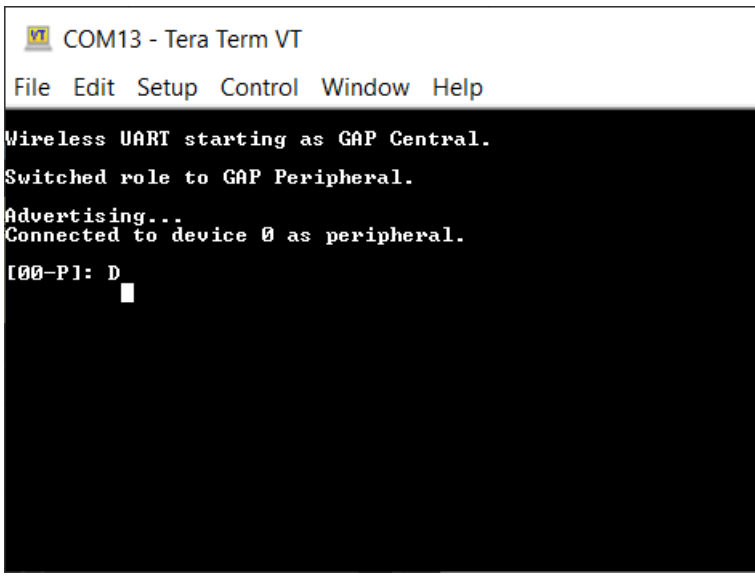
Switches and pins

Usage

The application is built to work with another supported platform running either Wireless UART or Wireless UART Host, or with the Wireless UART from the IoT Toolbox application.

When testing with two boards, perform the following steps:

1. Open a serial port terminal and connect to the two boards, in the same manner described in Testing devices. The start screen is blank after the board is reset.
2. The application starts as a GAP central. To switch the role to a GAP peripheral, press the role switch. Depending on the role, when pressing the **SCANSW**, the application starts either scanning or advertising.
3. As soon as the **CONNLED** turns solid on both devices, the user can start writing in one of the consoles. The text appears on the other terminal.
4. After creating a connection, the role (central or peripheral) is displayed on the console. The role switch can be pressed again before creating a new connection. See the figure below.



Tera Term – received text on Wireless UART

When testing with a single board and the IoT Toolbox, perform the following steps:

1. Open a serial port terminal and connect the board in the same manner described in Testing devices. The start screen is blank after the board is reset.

2. Press the role switch button to behave as a GAP peripheral and then press the **SCANSW** button to start advertising. The IoT Toolbox app can then connect. Select UART instead of Console and start typing, as shown in the Figure.

Parent topic: Wireless UART Host

Parent topic:*Bluetooth LE stack and demo applications*

References

For more information, refer to the following documents:

- *Bluetooth Low Energy Application Developer's Guide*(KW45_BLEADG)
- *Bluetooth Low Energy Host Stack API Reference Manual*(KW45_BLEHSAPIRM)
- *Bluetooth Low Energy Host Stack FSCI*(Framework Serial Connectivity Interface) *API Reference Manual* (KW45_BLEHSFSCIAPIRM)
- *Connectivity Framework Reference Manual* (KW45_CONNFWRM)
- *Bluetooth Low Energy CCC Digital Key R3 Application Note* (AN12791) See AN12791 here.

Acronyms

The following acronyms are used in this document.

Acronym	Description
ANCS	Apple Notification Center Service
ATT	Attribute Protocol
Bluetooth LE	Bluetooth Low Energy
CCC	Car Connectivity Consortium
CCCD	Client Characteristic Configuration Descriptor
DBAF	Decision-Based Advertising Filtering
EATT	Enhanced Attribute protocol
EVK	Evaluation Kit
FSCI	Framework Serial Connectivity Interface
GAP	Generic Access Profile
GATT	Generic Attribute Profile
HCI	Host Controller Interface
HID	Human Interface Device
HRS	Heart Rate Server
JSON	Javascript Object Notation
L2CAP	Logical Link Control and Adaptation Protocol
NVM	Non-volatile Memory
OTAP	Over the Air Programming
RF	Radio Frequency
RSSI	Received Signal Strength Indicator
RX	Receive
SDK	Software Development Kit
TX	Transmit
UART	Universal Asynchronous Receiver/Transmitter

Note about the source code in the document

Example code shown in this document has the following copyright and BSD-3-Clause license:

Copyright 2022-2025 NXP Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Bluetooth Low Energy CCC Digital Key R3 Application Note Introduction

NXP provides hardware platforms to implement Bluetooth Low Energy applications implementing the Car Connectivity Consortium (CCC) Digital Key R3 functionality. This document provides an overview of the deployment and operation of such applications. This document is also a user guide for the sample applications included in the SDK release package.

Audience

This document is for firmware and system developers who create CCC Digital Key-enabled products. The document also provides a high-level description of CCC Digital Key application scenarios that can be deployed on NXP development boards.

Parent topic:*Introduction*

CCC Digital Key technology overview

The Car Connectivity Consortium (CCC) is a cross-industry organization advancing global technologies for smartphone-to-car connectivity solutions. CCC has developed Digital Key, a new open standard to allow smart devices such as smartphones to act as a vehicle key. Digital Key lets drivers lock and unlock their cars and even allows them to start the engine and share access to friends or valets, using their phones.

The Digital Key Release 3.0 specification enhances Digital Key Release 2.0 by adding passive, location-aware keyless access. Rather than having to pull their mobile devices out to access a car, consumers are able to leave their mobile device in their bag or pocket when accessing and/or starting their vehicle. Passive access is not only vastly more convenient and a better overall user experience, it also allows vehicles to offer new location-aware features.

To support these new features, the CCC has developed a specification based on Bluetooth Low Energy in combination with Ultra-Wideband (UWB) to enable passive keyless access and to allow secure and accurate positioning.

CCC Digital Key applications overview

Note: The applications described below implement only the Bluetooth Low Energy functionality described by CCC Digital Key R3. They do not include any UWB or Secure Element functionality. The establishment of secure communications between the demos is simulated using dummy messages.

Digital Key Car Anchor

The Digital Key Car Anchor demo application implements one of the multiple Bluetooth Low Energy anchors that can reside inside the car. The anchor performs a dual role. Inside the CCC Digital Key scope, it is a Bluetooth Low Energy peripheral which, depending on the scenario, can perform both Legacy (1M PHY) and Extended Long Range (500 kbps Coded PHY) advertising, searching for a CCC Digital Key-enabled device such as a smartphone. Simultaneously it can also act as a Bluetooth Low Energy central, scanning for non-CCC key fobs. There can be multiple anchors inside a car. The anchors must act as a single device and share information between them (addresses, bonding data, application specific keys, and such data) such that a smartphone can connect to any one of them depending on positioning and not detect any difference.

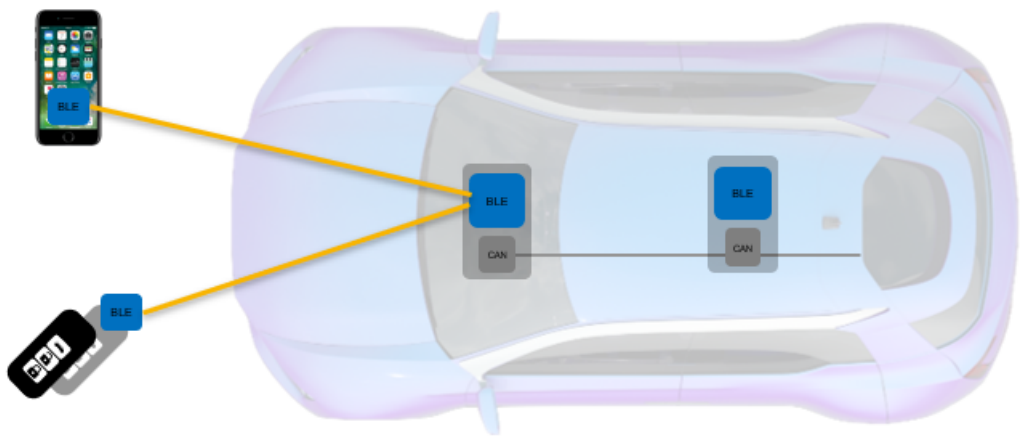
Parent topic:*CCC Digital Key applications overview*

Digital Key Device

The Digital Key Device demo application emulates a CCC Digital Key-enabled smartphone. It acts as a Bluetooth Low Energy central device, scanning for advertising coming from a Digital Key Car Anchor.

The Figure 1 shows an example deployment of CCC Digital Key R3. Inside the car, there are two Bluetooth Low Energy anchors connected to each other. The Controller Area Network (CAN) bus connects these anchors and is used to exchange security data and other information. Any anchor can perform both advertising and scanning and it is able to connect to a CCC Digital Key R3-enabled smartphone as well as to a non-CCC key fob.

Anchor communicating with a smartphone (CCC) and a key fob (non-CCC)



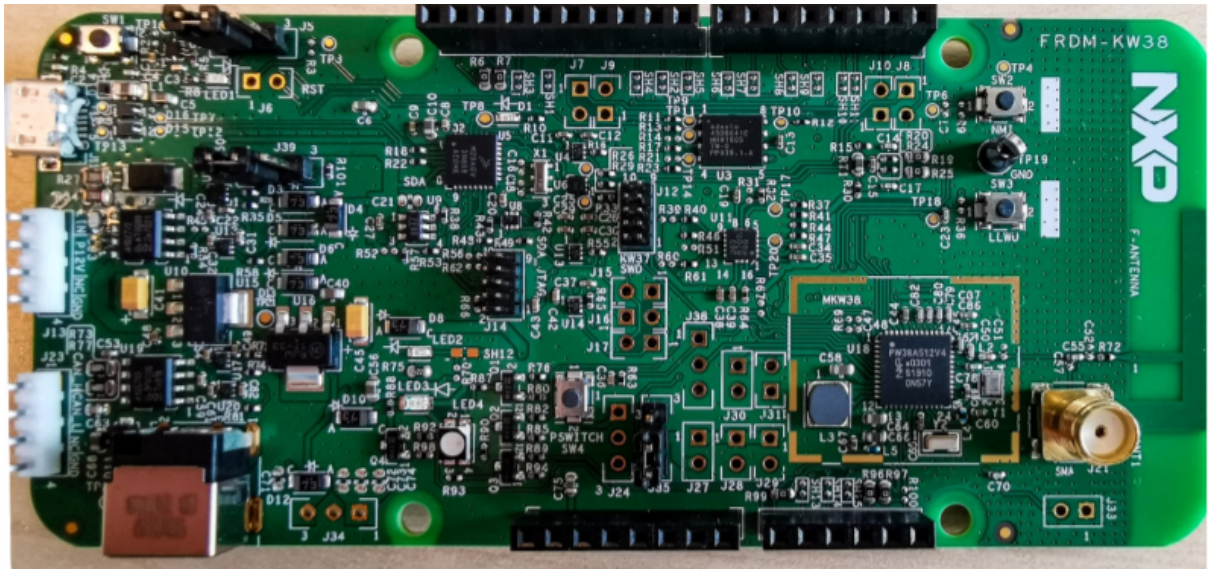
Parent topic:*CCC Digital Key applications overview*

CCC Digital Key hardware platforms

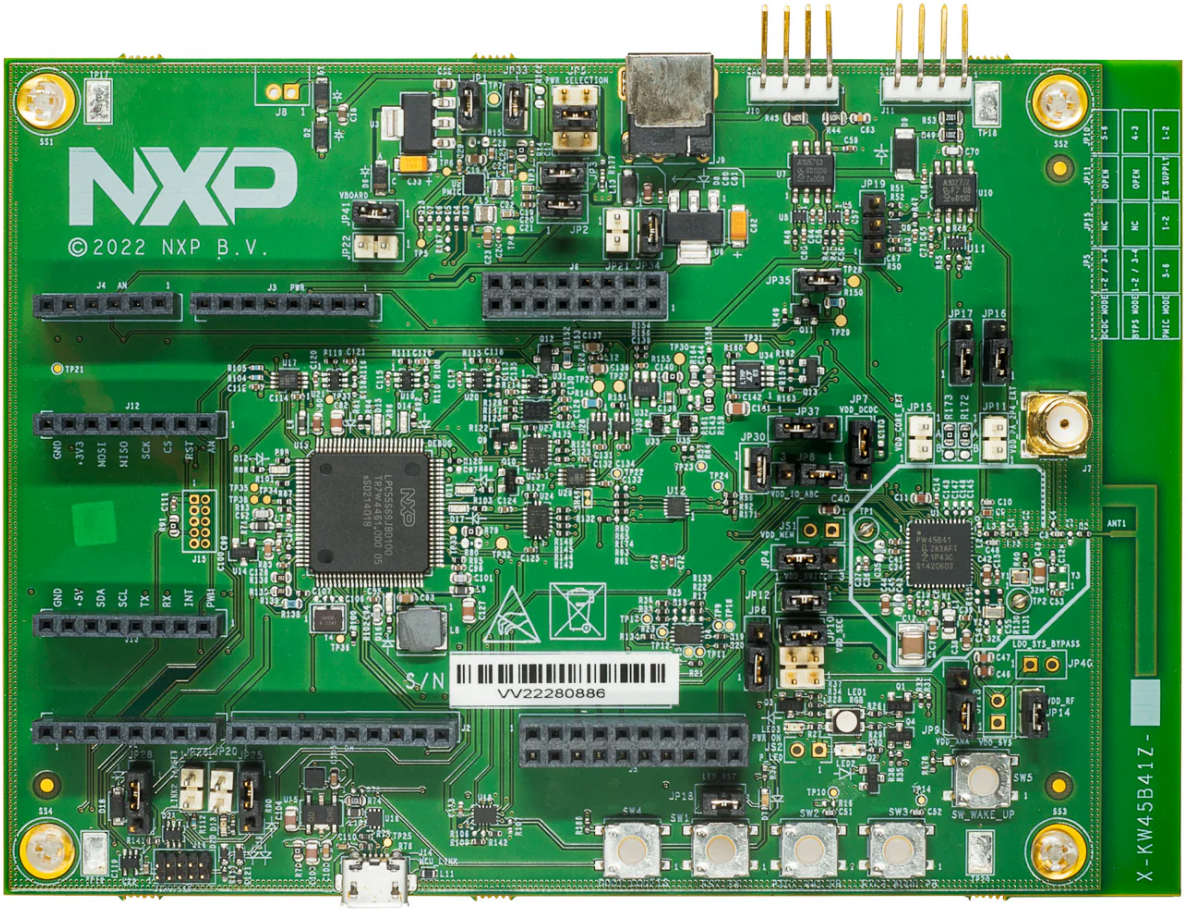
The CCC Digital Key R3 demo applications support the following platforms that have Bluetooth Low Energy transceiver capabilities:

- FRDM-KW38 (see Figure 1)
- KW45B41Z-EVK (see Figure 2)
- KW47-EVK (see Figure 3)
- FRDM-KW43 (see Figure 4)

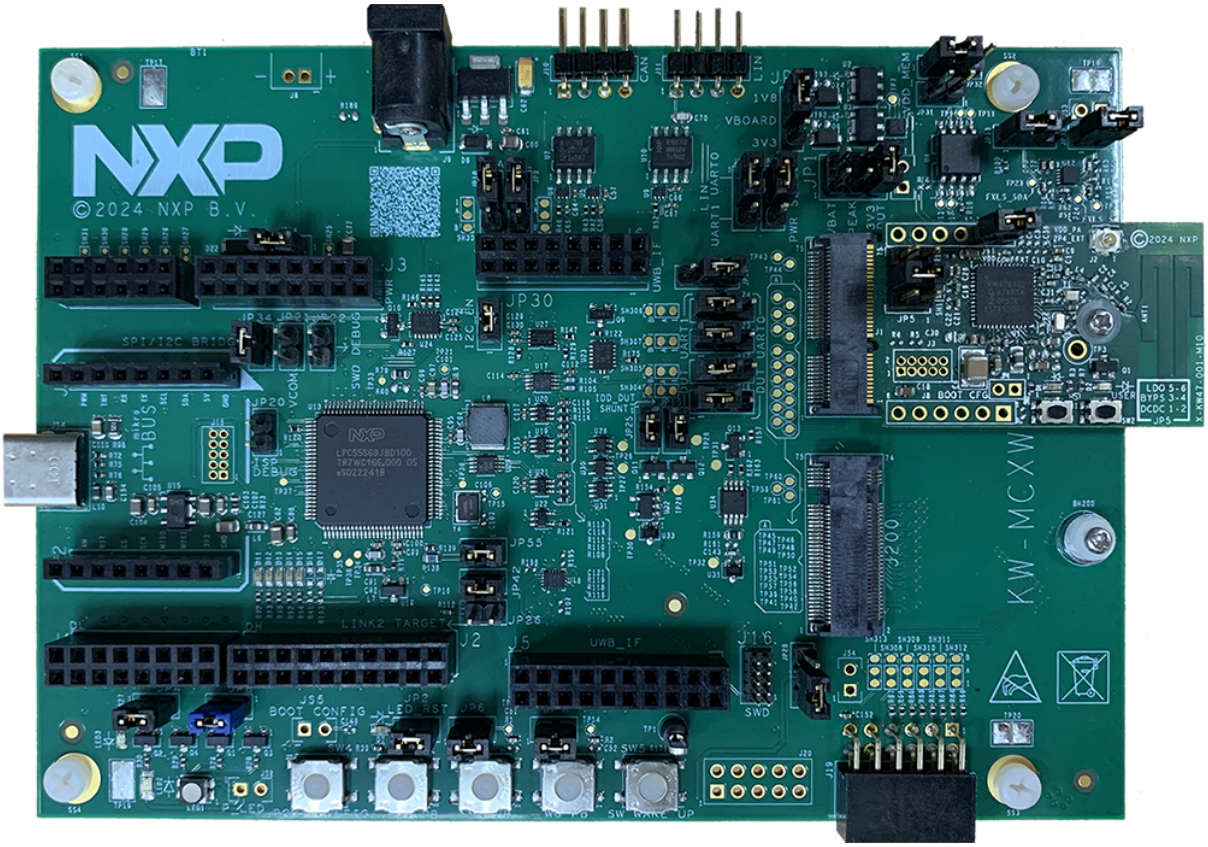
FRDM-KW38 board

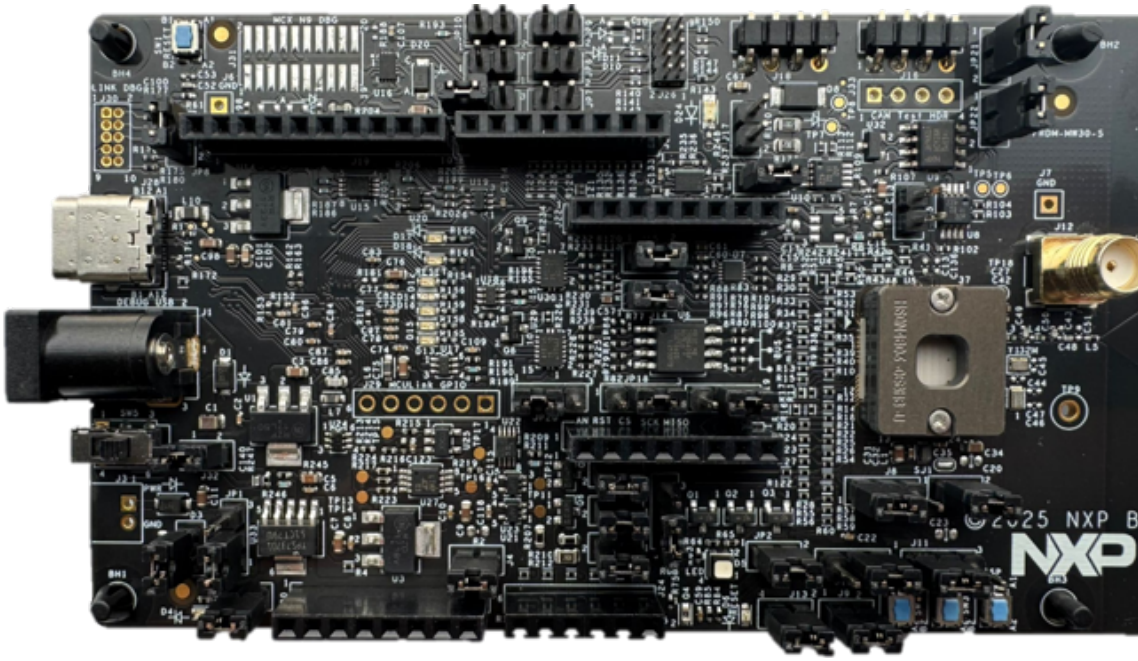


KW45B41Z-EVK board



KW47-EVK board





FRDM-KW43 board

Deploying CCC Digital Key applications software

The software package includes the components necessary for CCC Digital Key R3 application development on the selected NXP platform. These components include:

- Bluetooth Low Energy Host and Controller software libraries (KW38)
- NBU image binaries
- Examples of demo application projects corresponding to CCC Digital Key roles (Device and Car Anchor)

- Platform peripheral drivers, platform startup code, other generic platform software

Note:

Prior to loading any wireless SDK example, update your NBU image with the provided binaries in the following folder of the SDK:

../middleware/wireless/ble_controller/bin.

The demo applications were compiled and tested with **IAR Embedded Workbench** for Arm and **ARMGCC**. It is recommended to use one of these tools.

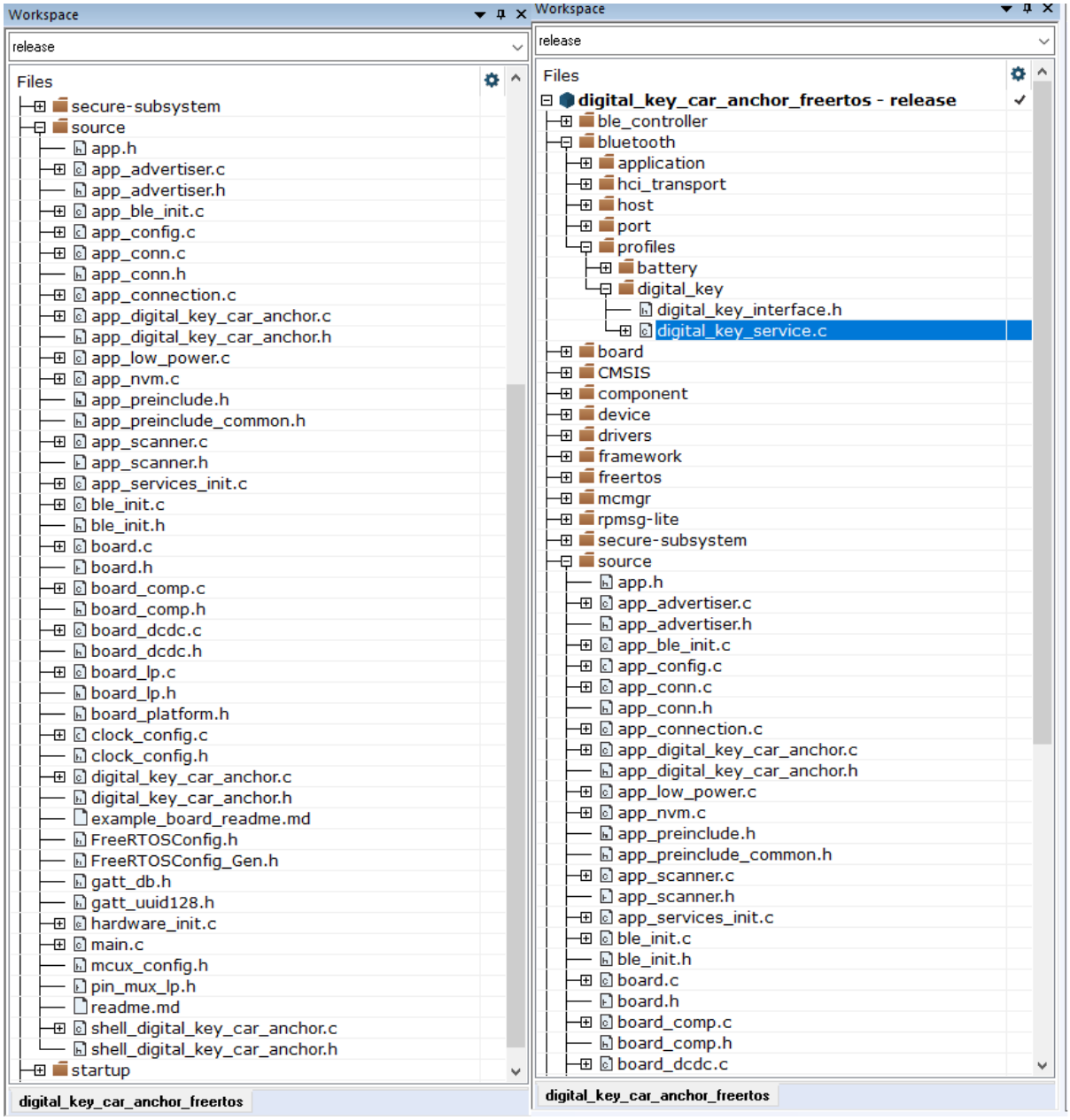
To open, build, and run any example application, see the *Bluetooth Low Energy Quick Start Guide* document of the corresponding board. The CCC Digital Key R3 examples are built and run in the same fashion as Bluetooth Low Energy examples.

The Figure 1 shows the application folder structure for the *digital_key_car_anchor* project.

The relevant files for CCC Digital Key R3 are the following:

- The main application source files source/digital_key_car_anchor.c and source/app_digital_key_car_anchor.c, and
- The files in bluetooth/profiles/digital_key, which implement the Digital Key Service defined by the CCC Digital Key R3 specification.

Application folder structure in workspace



Demo functionality overview

The two demo applications provided showcase the following Digital Key R3 features:

- Owner Pairing between a Device and a Car Anchor
- Passive Entry
- Synchronizing bonding data between Car Anchors
- Connection between a Car Anchor and a non-CCC key fob

The demo applications also showcase the NXP proprietary Connection Handover and Anchor Monitoring features, as well as the A2B feature (secured synchronization of bonding data).

Running CCC Digital Key scenarios using the Shell Interface

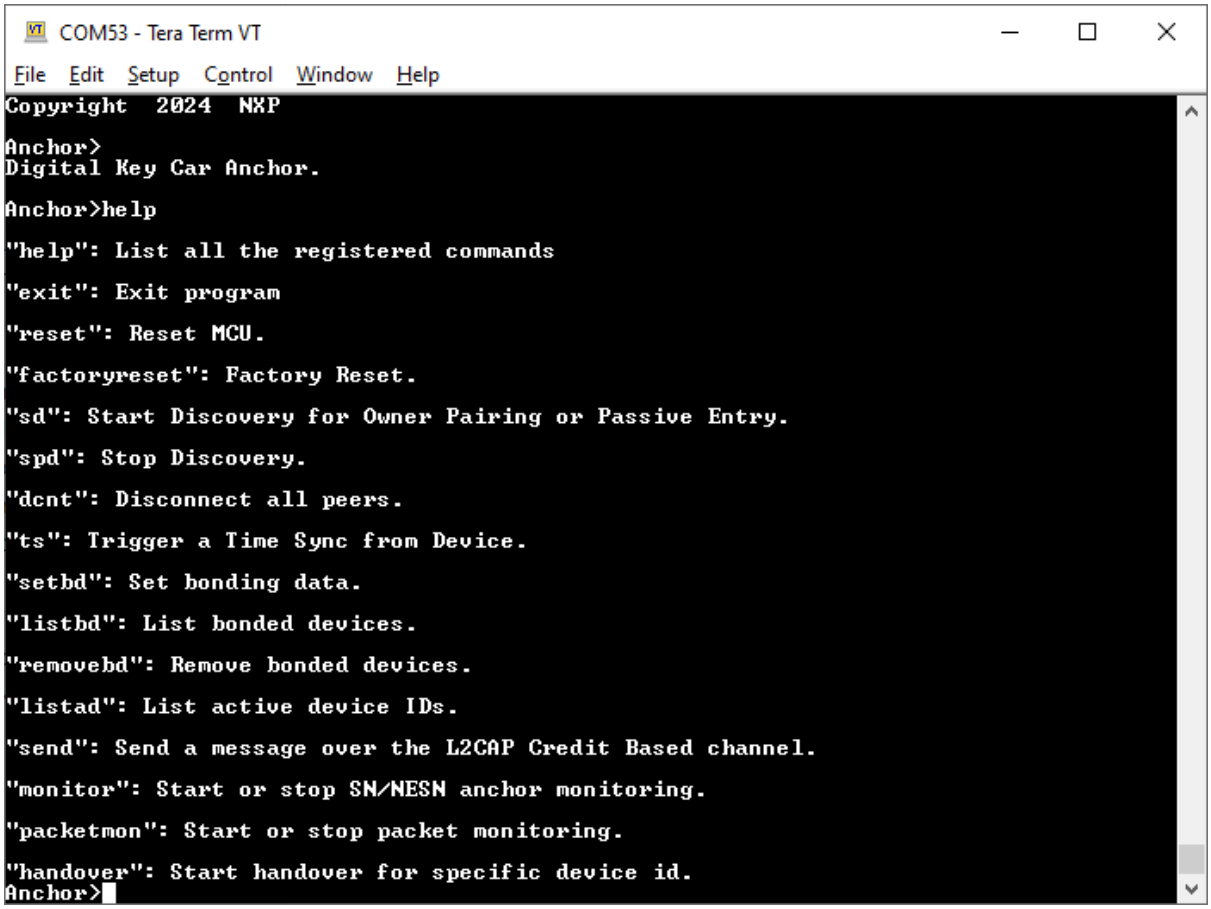
The following CCC Digital Key R3 examples are provisioned by default with a shell command line interface accessible via a Terminal application such as Tera Term:

- Digital Key Car Anchor

- Digital Key Device

After connecting through the Terminal application and pressing the RST switch on the board, a welcome message appears. Entering the command "help" displays the list of available commands and their descriptions. See Figure 1.

List of commands

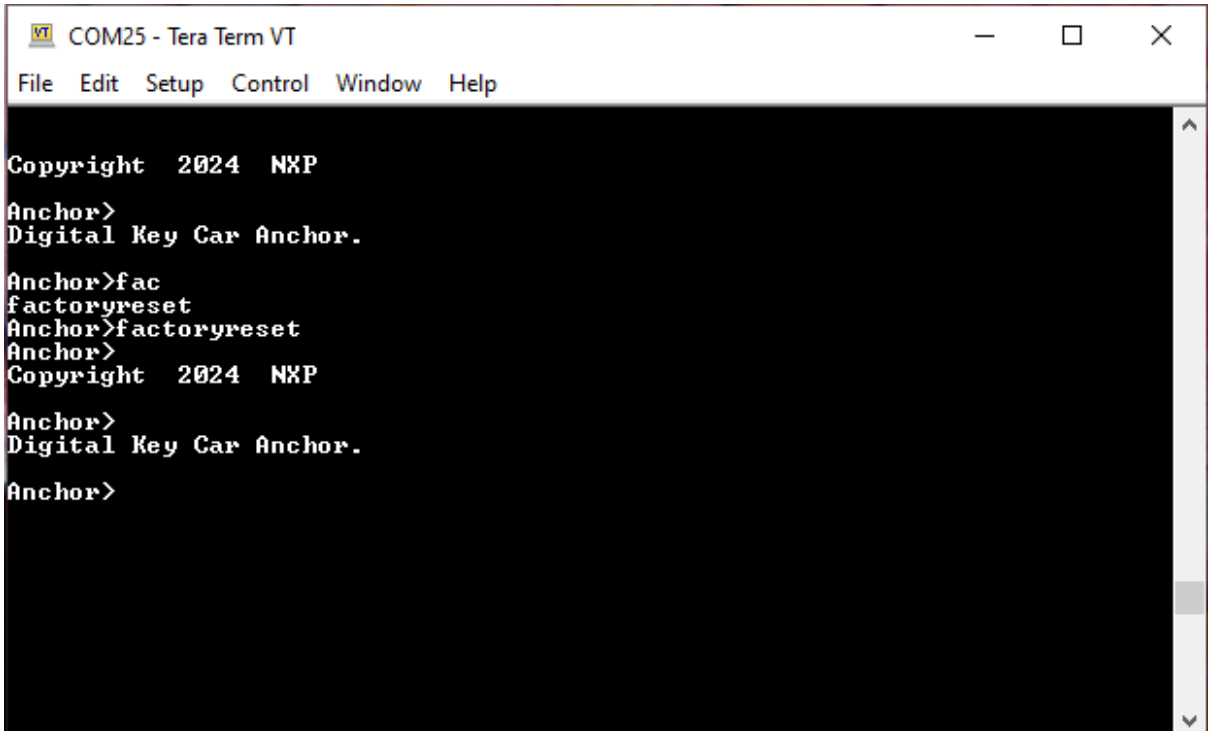


Owner Pairing Scenario

Owner Pairing establishes a bond between the Digital Key Car Anchor and the Digital Key Device as per the CCC Digital Key R3 specification. The Car Anchor starts advertising using Legacy mode on the 1M PHY. The Device scans and connects. After establishing the connection, the Device performs service discovery. This step enables it to discover the DK Service and learn the PSM value it would use in order to open an L2CAP channel to the Car Anchor. This channel is used to exchange data and security information as part of the Bluetooth Low Energy Out-of-Band (OOB) pairing process. All security information exchanged consists of dummy messages. The applications do not currently implement any of the UWB and Secure Element functionality described by CCC Digital Key R3.

The first step is to run the "factoryreset" command on both boards in order to ensure no previous bonding data is present in non-volatile memory. See Figure 1.

Factory reset on Car Anchor



The next step is to run the "sd" command (Start Discovery) on the Device and the "sd op" command on the Car Anchor (Owner Pairing advertising differs from Passive Entry, using a single Legacy advertising set on the 1M PHY). The Car Anchor starts advertising while the Device scans. Since both boards have been factory reset, they follow the Owner Pairing flow. The output of the commands is shown in Figure 2.

Owner Pairing on Car Anchor

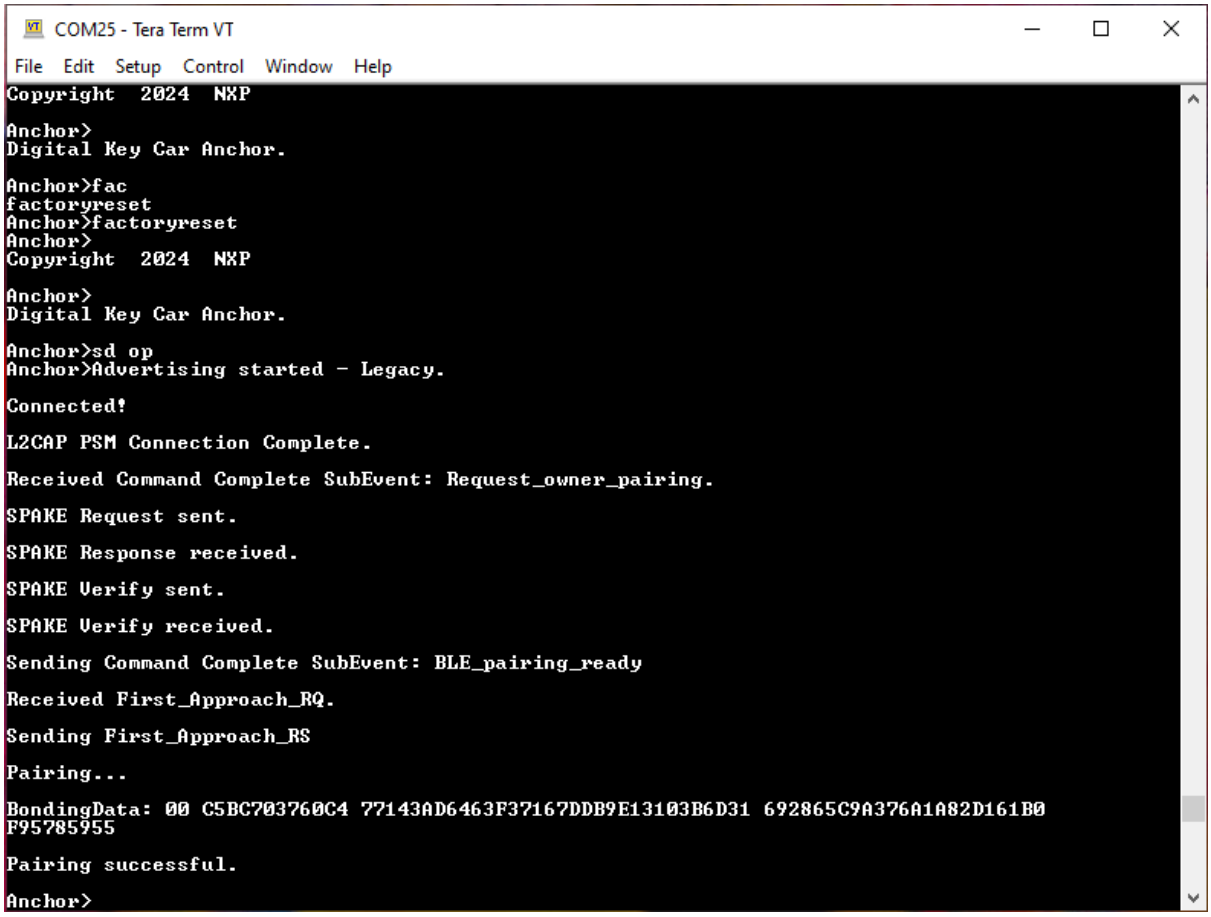
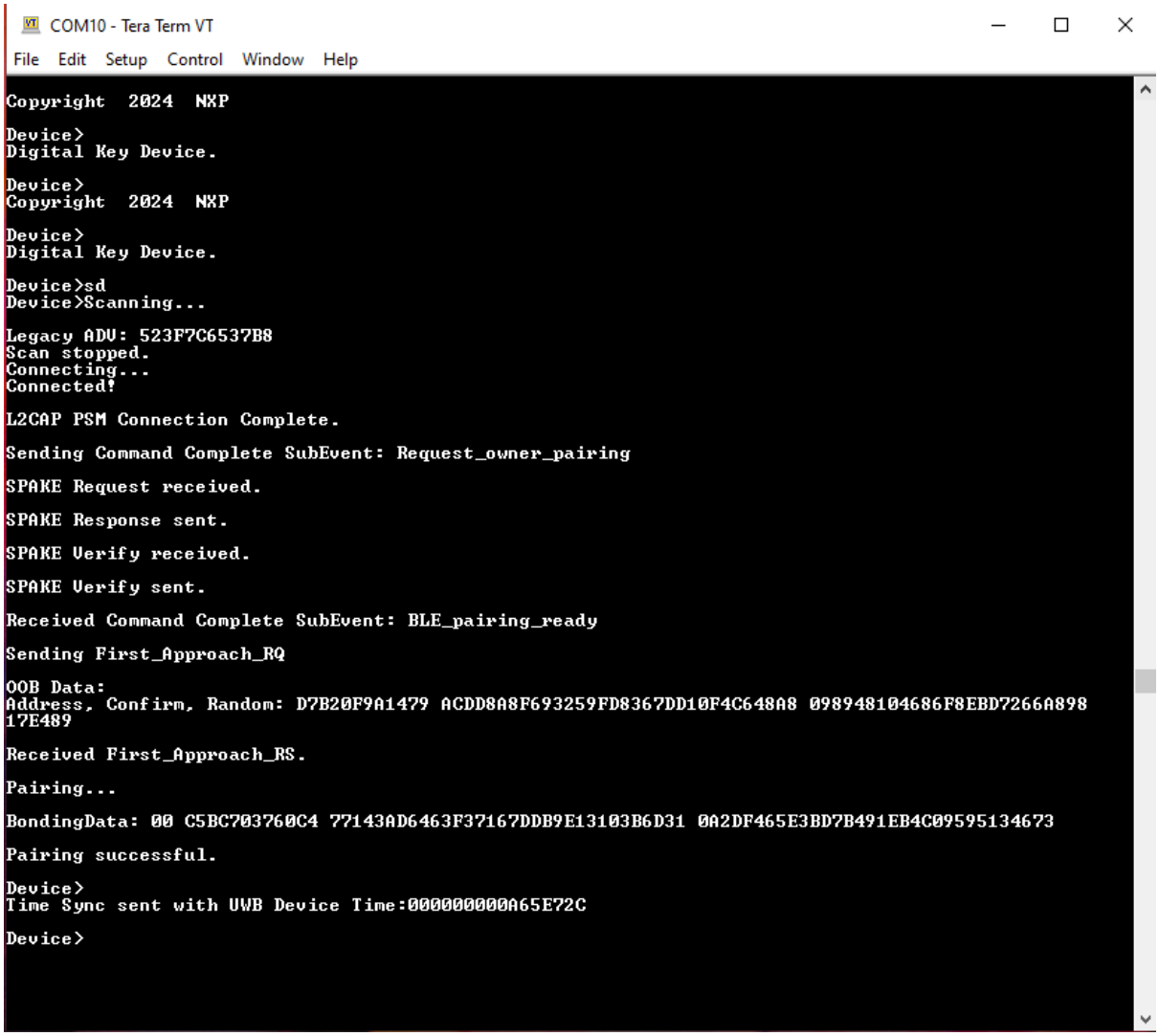


Figure 3 shows the owner pairing on the Device.

Owner Pairing on Device



Parent topic:*Running CCC Digital Key scenarios using the Shell Interface*

Passive Entry Scenario

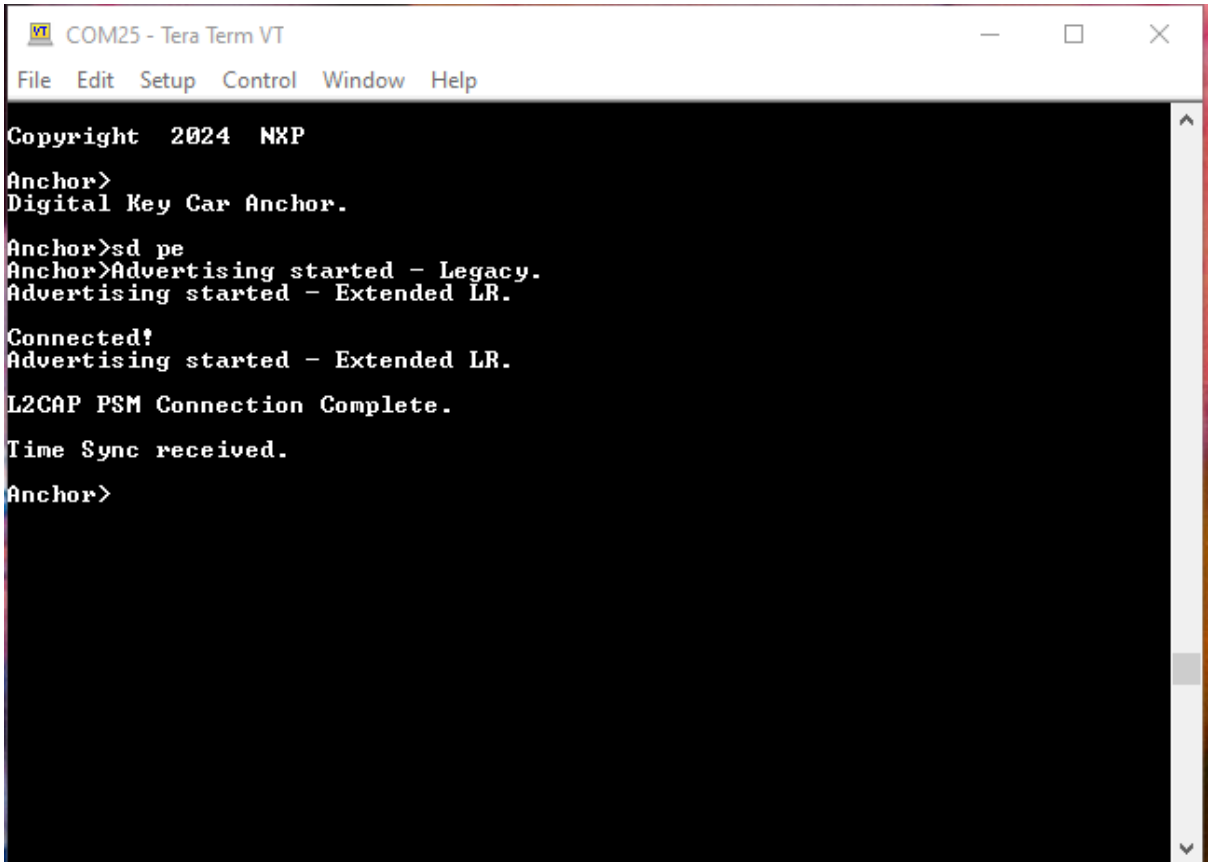
The Passive Entry scenario runs after Owner Pairing between a Car Anchor and a Device completes successfully.

To run the Passive Entry demo, first, reset the recently paired boards using either the **RST** switch or the "reset" command in the shell. Do NOT perform a factory reset, as this removes the bonding information from non-volatile memory.

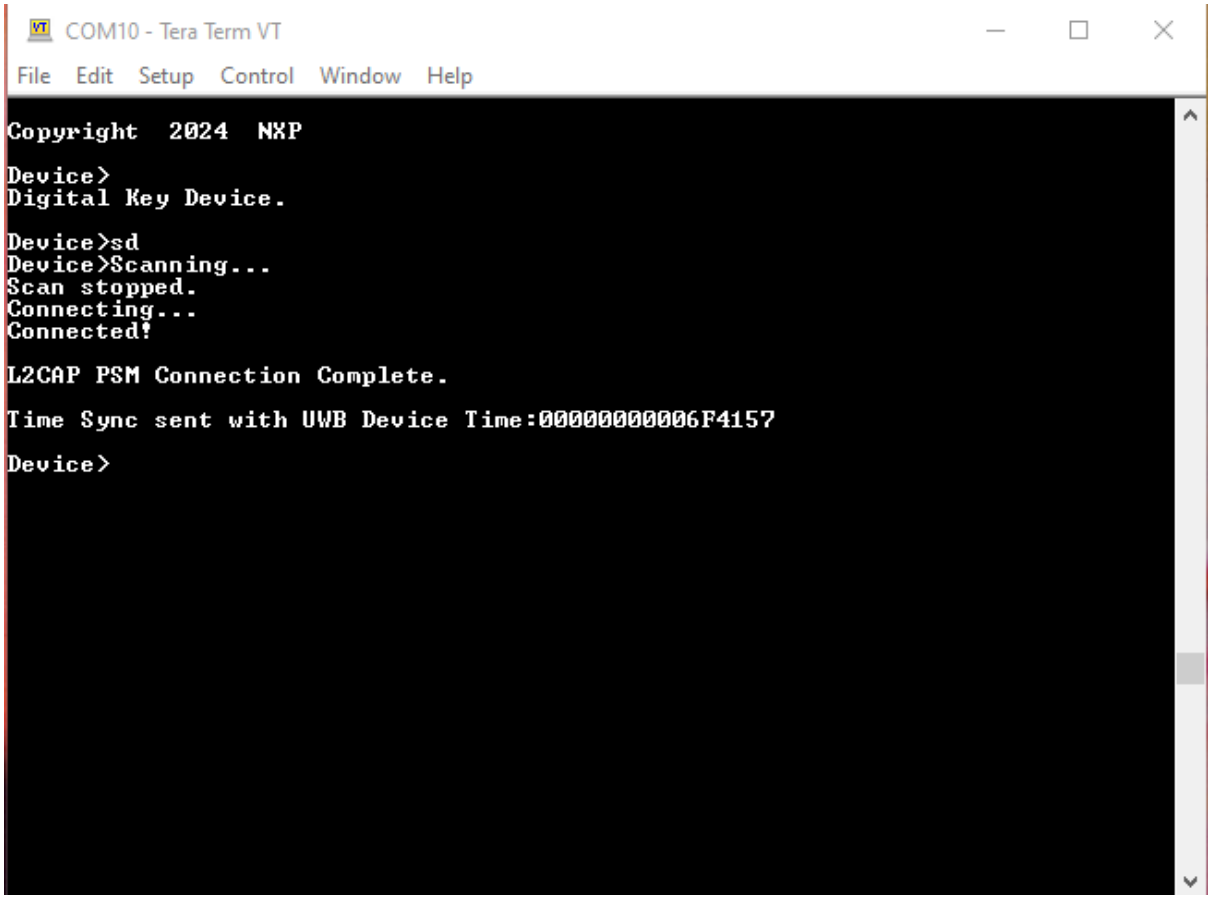
After reset, run the "sd pe" command on the Car Anchor and "sd" command on the Device as shown in Figure 1 and Figure 2. For the Passive Entry scenario, the Car Anchor advertises using two sets, one Legacy on the 1M PHY and one Extended on the Coded PHY. If bonding data is present, it follows the Passive Entry flow and stops after the successful creation of the L2CAP channel. At this point the application is functional. The output of the command is shown in Figure 1 and Figure 2. Similar to the Owner Pairing scenario, the applications implement only the Bluetooth Low Energy functionality.

Note: The CCC specification requires using coding scheme S=2 for Coded PHY advertising, equivalent to a data rate of 500 Kbits per sec. To experiment with other coding schemes, change the advertising parameters in the app_config.c file for the digital_key_car_anchor project.

Passive Entry on Car Anchor



Passive Entry on Device



Parent topic:*Running CCC Digital Key scenarios using the Shell Interface*

Synchronizing bonding data between Car Anchors

Note: On the KW45B41Z-EVK, KW47-EVK and FRDM-KW43 platforms, this feature is only valid when Advanced Secure Mode is disabled (gAppSecureMode_d is set to 0 in the app_preinclude.h file). To synchronize bonding data when Advanced Secure Mode is enabled, refer to *Running the A2B scenario*.

Multiple Car Anchors can reside on a car, acting as a single Bluetooth Low Energy device as far as the Device is concerned. When the Device pairs and bonds with a Car Anchor, that bonding data must be shared with all other anchors. To showcase this functionality, the Car Anchor shell demo offers the "setbd" command.

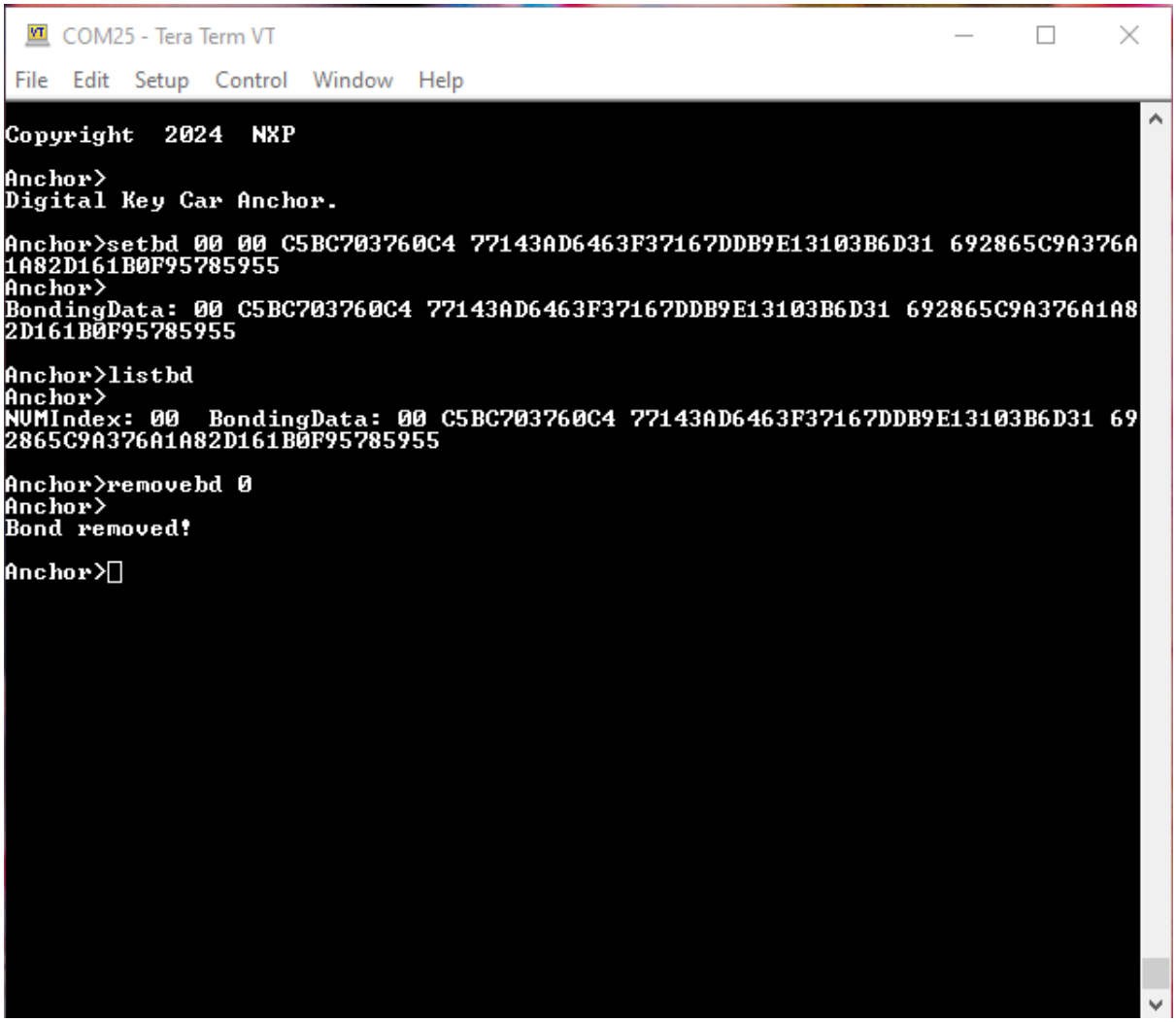
During owner pairing, the bonding data is displayed in the shell as seen in the Owner Pairing section. By passing this data to the "setbd" command on another Car Anchor, that anchor is able to connect with the original Device and perform the Passive Entry flow.

Note: Currently only the Bluetooth Low Energy bonding data is transferred between anchors. No CCC Digital Key R3-specific keys are exchanged.

Saved bonding data can also be viewed using the "listbd" command and a specific bond can be removed using the "removebd" command. See Figure 1 for details.

A bond can only be removed if a connection to that specific device is not currently active. The commands presented in this subsection are also supported on the Device for test purposes.

Adding bonding data to a Car Anchor, listing bonds and removing a bond



Note:

- To simulate anchors residing on the same car, Random Static Address required by the Digital Key protocol and the Identity Resolving Key are set at compile time for the digital_key_car_anchor project. To simulate anchors residing on different cars, change the values of the APP_BD_ADDR and APP_SMP_IRK macros in the project's app_preinclude.h.

Parent topic:*Running CCC Digital Key scenarios using the Shell Interface*

Connection with a non-CCC Key Fob Scenario

The Car Anchor also supports connecting as a Bluetooth Low Energy central device (scanner) to a non-CCC key fob (advertiser). To enable scanning on the digital_key_car_anchor project set the gAppScanNonCCC_d macro to 1 in the project's app_preinclude.h before building.

After flashing the board and running "factoryreset", run the "sd op" or "sd pe" command exactly as in the Owner Pairing or Passive Entry scenarios. The Car Anchor will simultaneously advertise for a CCC Device and scan for a non-CCC key fob.

Connection with the non-CCC key fob and the subsequent application functionality is currently out of this application note's scope.

Parent topic:*Running CCC Digital Key scenarios using the Shell Interface*

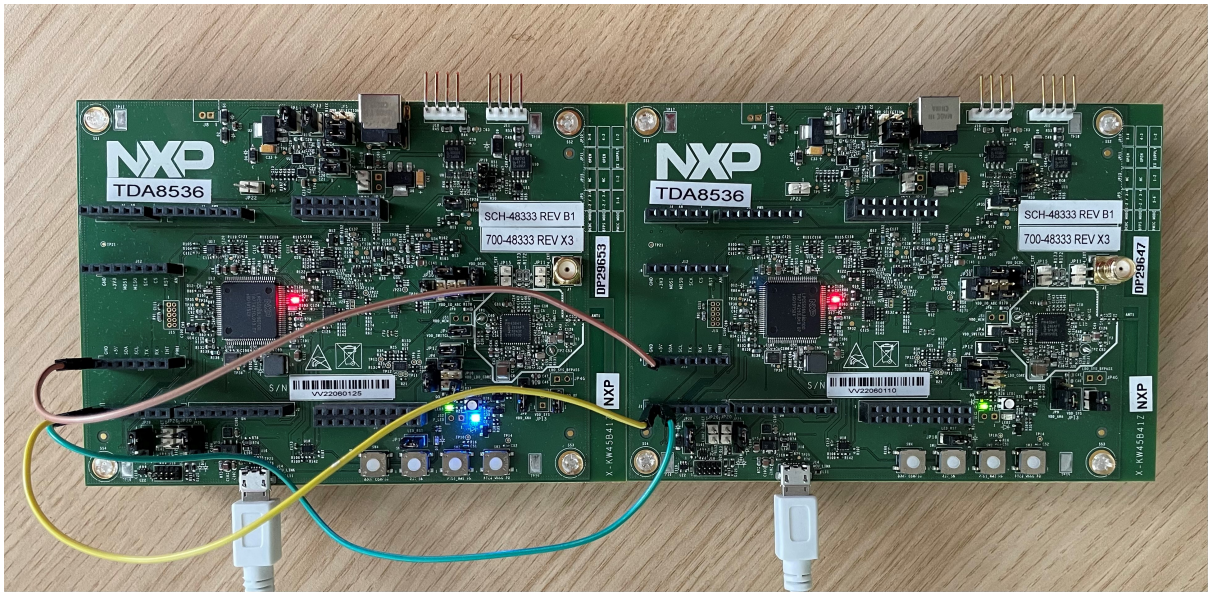
Running the Connection Handover scenario

Connection Handover is an NXP proprietary feature that enables a Bluetooth Low Energy connection to be seamlessly transferred from one peripheral to another while the central device remains unaware. It is best exemplified by (but not limited to) the CCC Digital Key use case. As a person carrying a phone moves around the vehicle, the Bluetooth Low Energy connection is transferred between Car Anchors in order to ensure the best user experience. From the Device's point of view, it stays in the same initial connection.

The CCC Digital Key demos showcase the Connection Handover feature. The handover is performed on demand via a button press. In a real scenario, the handover is performed based on criteria such as RSSI. The feature is currently supported on the KW45B41Z-EVK, KW47-EVK and FRDM-KW43 platforms.

Prerequisites:

- Three boards (two boards act as Car Anchors, one as a Device). The demo is currently limited to a maximum of two Car Anchors.
- The two Car Anchors must have a serial connection via the secondary UART as shown in the image below. The UART connection stands in for a real deployment solution such as a CAN bus.
- The *gHandoverDemo_d* macro must be set to 1 in app_preinclude.h for the ***digital_key_car_anchor*** project. Figure 1 shows KW45B41Z-EVK car anchors connected via secondary UART. *(J1-1 is connected to J1-2 and J1-2 is connected to J1-1. GND connection is J13-1 to J13-1.)*
- If KW47-EVK boards are used, make the UART connection between J1-1 to J1-3 and J1-3 to J1-1.
- If FRDM-KW43 boards are used, make the UART connection between JP19-1 (UART0_TX PTA17) to J22-1 (UART0_RX PTA18) and J22-1 to JP19-1. **Note:** The current firmware configuration uses PTA18 for UART0_RX instead of PTA16 as shown in the schematic. **Two Car Anchors connected via the secondary UART**



Demo steps:

- Connect a Car Anchor and a Device using either the Owner Pairing or Passive Entry scenario as detailed in the previous sections. If doing Owner Pairing, the bonding data synchronization between the two Car Anchors will happen automatically over the UART connection.
- Optional: Use the "send" shell command on the Car Anchor to send a test message over the L2CAP Credit-Based channel. The message is displayed in the console on the Device.
- Press the **SW3** switch or use the handover command with the peer device ID on the Car Anchor which is connected to the Device. To see a list of connected devices with their corresponding peer device IDs the listad command may be used.
- The connection is handed over to the second Car Anchor as seen in console messages. LED2 also turns solid blue on the Car Anchor, which has taken over the connection.
- **Optional**: Use the "send" shell command on the Car Anchor that has taken over the connection. The message is displayed in the console on the Device.
- The connection can continue to be handed over back and forth between the two Car Anchors by pressing the **SW3** switch on the Car Anchor which currently has the connection. Figure 2 shows first Car Anchor connects and pairs with the Device, sends the L2CAP test message and successfully performs handover when **SW3** is pressed.

Paring and connection of First Car Anchor with Device and handover

```
COM53 - Tera Term VT
File  Edit  Setup  Control  Window  Help

Copyright  2024  NXP

Anchor>
Digital Key Car Anchor.

Anchor>sd op
Anchor>Advertising started - Legacy.

Connected!

L2CAP PSM Connection Complete.

Received Command Complete SubEvent: Request_owner_pairing.

SPAKE Request sent.

SPAKE Response received.

SPAKE Verify sent.

SPAKE Verify received.

Sending Command Complete SubEvent: BLE_pairing_ready

Received First_Approach_RQ.

Sending First_Approach_RS

Pairing...

BondingData: 00 D92718376000 08ECC51B79A5849D3BEAB9B071977DE9 692865C9A376A1A82D161B0
F95785955

Pairing successful.

Anchor>
Time Sync received.

Anchor>send
Anchor>listad
Anchor>
DevId       AddrType    Address
00          00          484E0020907E

Anchor>handover 0
Anchor>
Handover started.

Handover complete, disconnected.

Anchor>
```

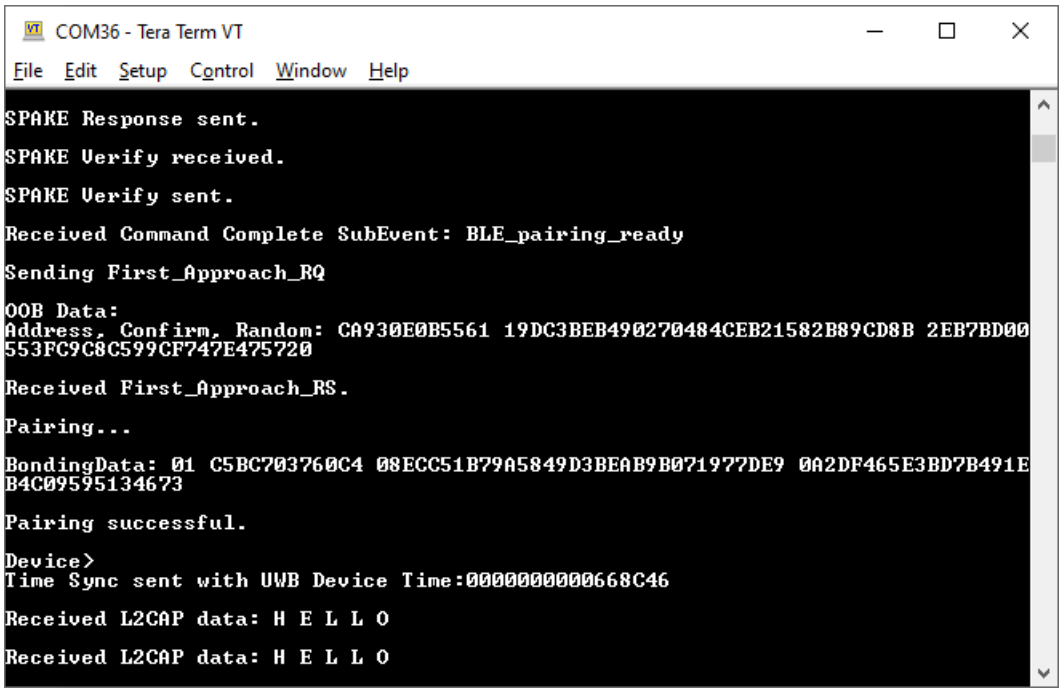
Figure 3 shows that the second Car Anchor receives the bonding data automatically via UART when the first Car Anchor pairs with the Device. After the handover is successfully completed, it sends the L2CAP test message to the Device.

Second Car Anchor receiving the bonding data automatically via UART when the first Car Anchor pairs with the Device

```
COM56 - Tera Term VT
File  Edit  Setup  Control  Window  Help

Copyright  2024  NXP

Anchor>
Digital Key Car Anchor.

Anchor>
BondingData: 00 D92718376000 08ECC51B79A5849D3BEAB9B071977DE9 692865C9A376A1A82D
161B0F95785955

Anchor>
Handover started.

Anchor>
Handover complete, connected.

Anchor>send
Anchor>
```

Figure 4 shows the Device receiving both L2CAP test messages. From the device's point of view, it is in a single uninterrupted connection.

The Device receives both L2CAP test messages



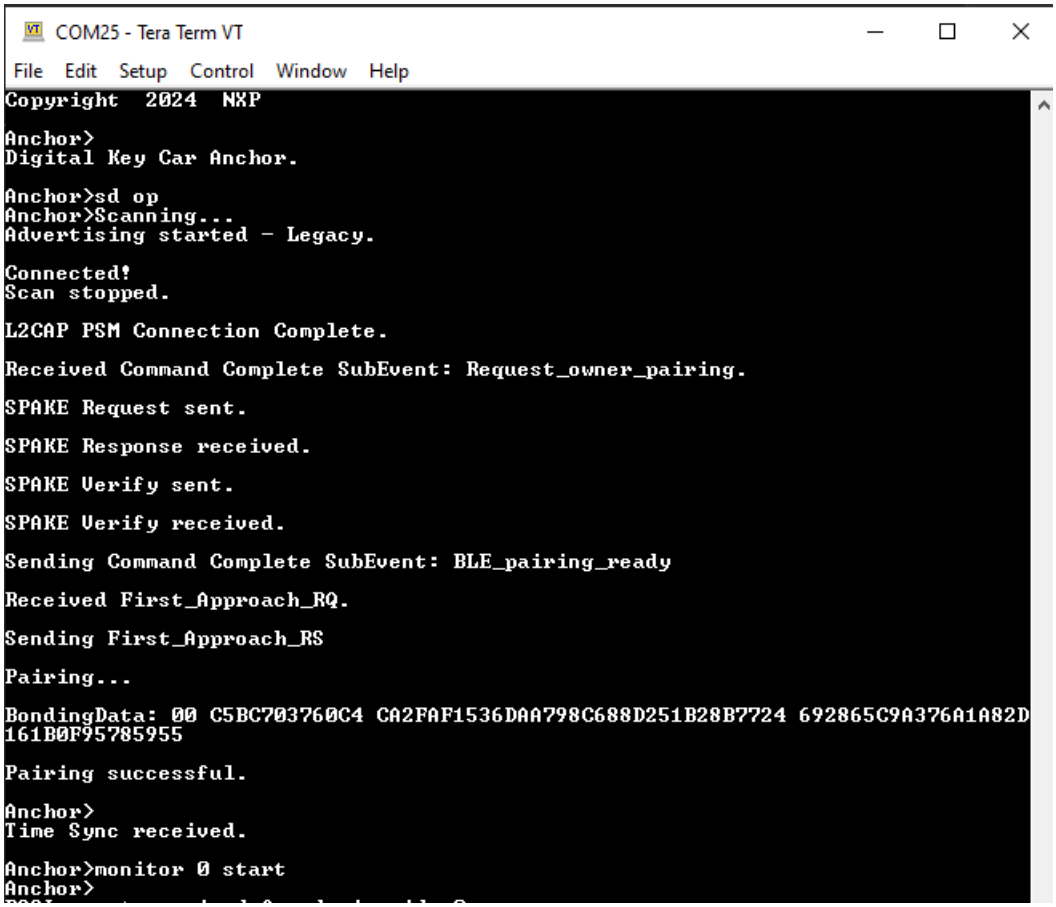
Running the Anchor Monitoring scenario

Anchor Monitoring is an NXP proprietary feature that allows a second Anchor to monitor the connection between the first Anchor and the Device. The prerequisites are the same hardware setup as used for the Connection Handover scenario. Note, however, that the Connection Handover scenario cannot be performed while Anchor Monitoring is in progress.

Demo steps:

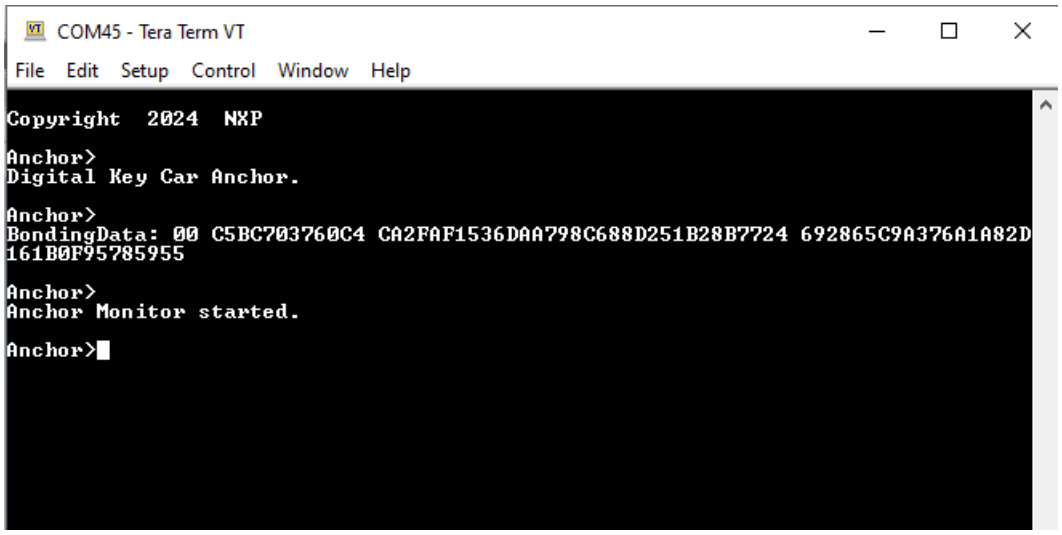
1. Connect a Car Anchor and a Device using either the Owner Pairing or Passive Entry scenario as described in Owner Pairing Scenario and Passive Entry Scenario.

Anchor has performed Owner Pairing and the "monitor start" command is run



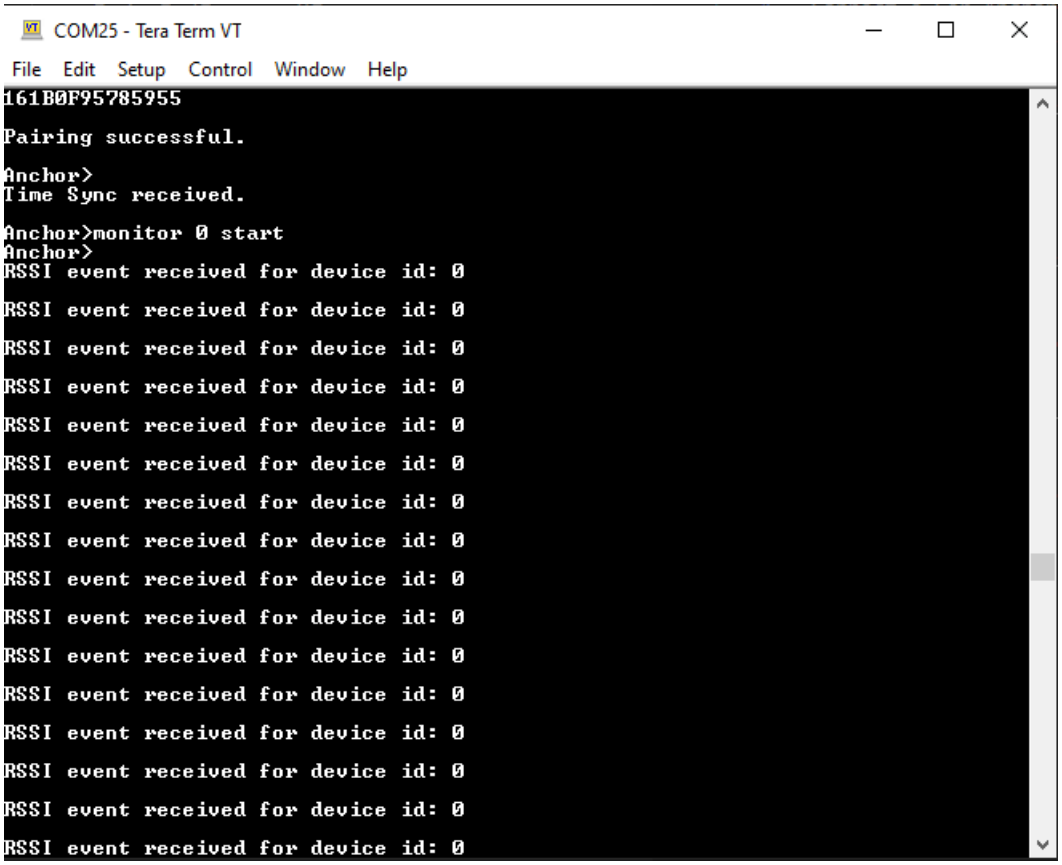
2. Run the "monitor 0 start" shell command on the Car Anchor, as seen in Figure 1.

The second Anchor has received the command over the serial interface and has started Anchor Monitoring



3. The second Car Anchor begins monitoring the connection and receives RSSI info events, which it forwards to the first Car Anchor (see Figure 2).

The first Anchor displays RSSI events received from the second Anchor.



4. The received RSSI messages are displayed in the console (see Figure 3).
5. To stop monitoring, run the "monitor 0 stop" shell command on either Car Anchor (the second one is recommended, as its console is not flooded by RSSI event messages).

Parent topic:*Running the Connection Handover scenario*

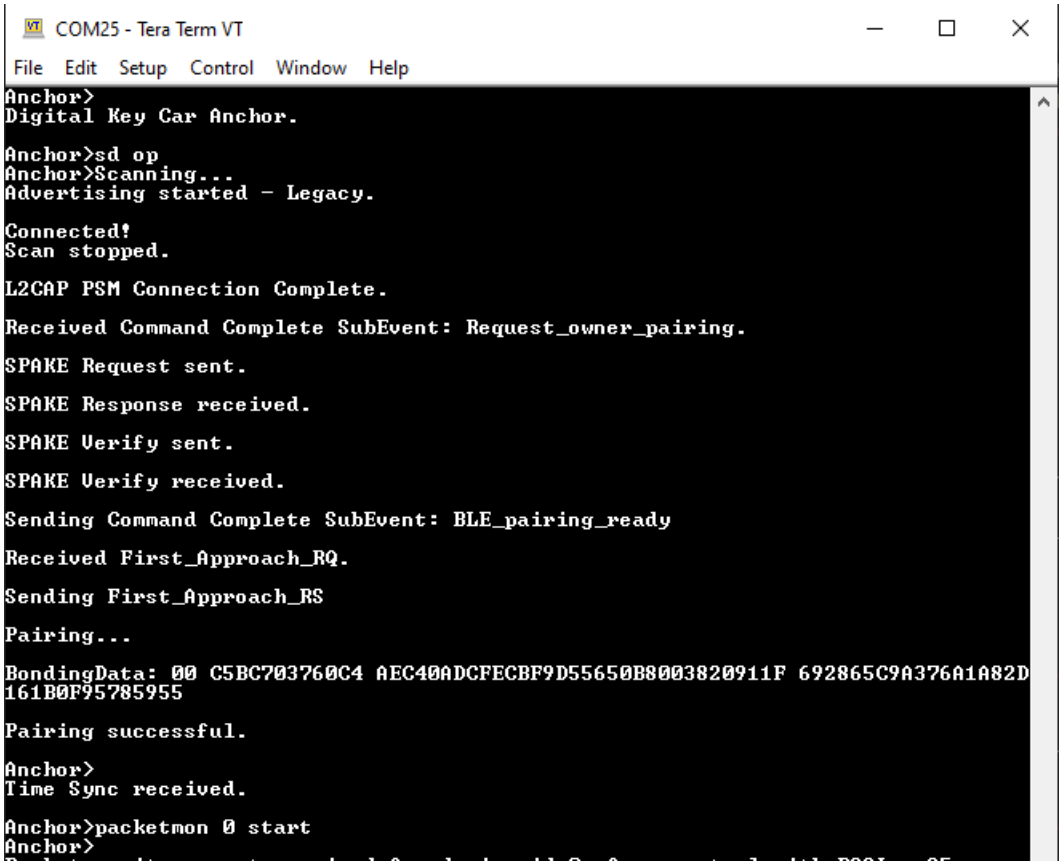
Running the Packet Monitoring scenario

Packet Monitoring is an NXP proprietary feature that allows a second Anchor to receive the packets from the connection between the first Anchor and the Device. The prerequisites are the same hardware setup as used for the *Running the Connection Handover scenario*. Note, however, that the Connection Handover scenario cannot be performed while Packet Monitoring is in progress. Packets received by the second Anchor are sent over the serial interface to the first Anchor for processing. The first Anchor determines the source of the packet based on bit 1 of the status parameter, which indicates whether the packet was sent by the central (1) or the peripheral (0), and displays the RSSI.

Demo steps:

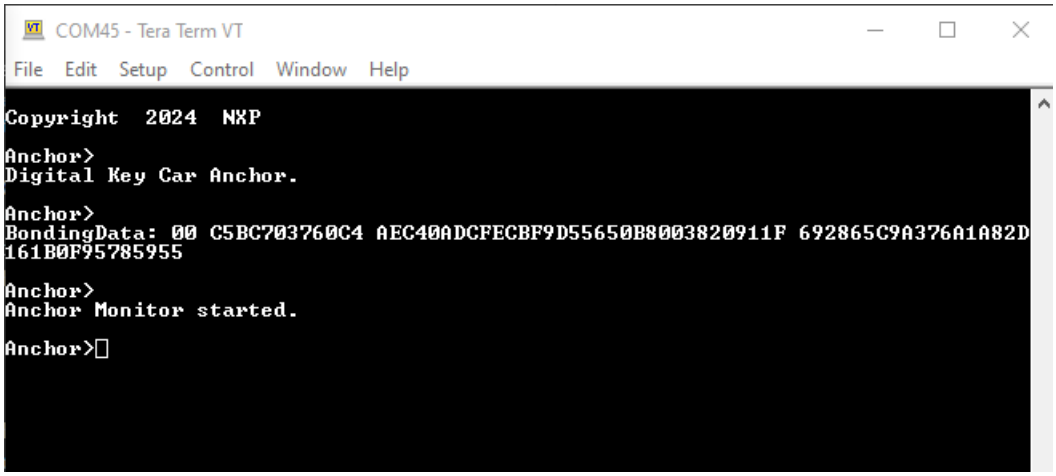
1. Connect the first Anchor and a Device using either the Owner Pairing or Passive Entry scenario as described in Owner Pairing Scenario and Passive Entry Scenario.

Anchor has performed Owner Pairing and the "packetmon start" command is run



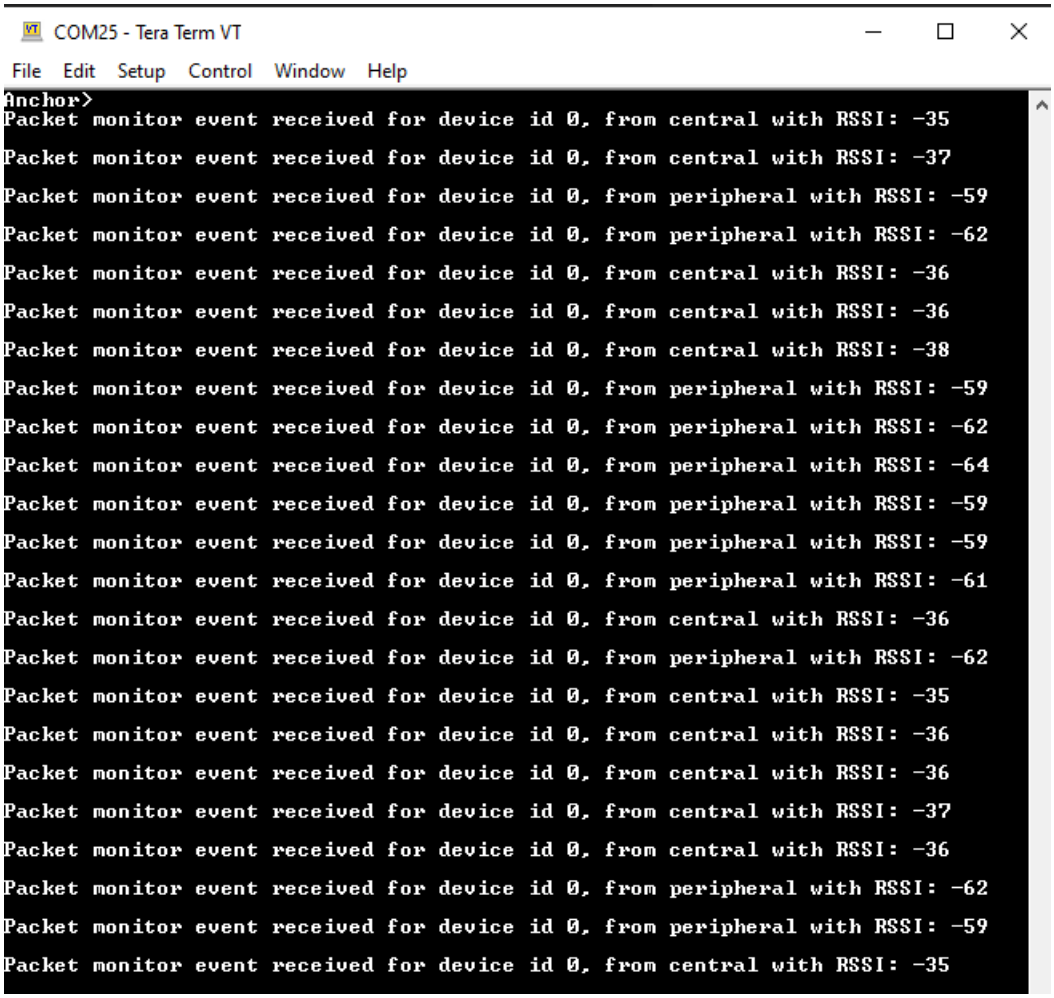
2. Run "packetmon 0 start" on the first Anchor, as shown in Figure 1.

The second Anchor has received the command over the serial interface and has started Packet Monitoring



3. The second Anchor begins monitoring the connection and receives packet information, which it forwards to the first Anchor. See Figure 2.

The first Anchor displays the RSSI and source device of the received packets from the second Anchor



4. The first Anchor receives the packet information from the second Anchor and determines the source of the packet based on bit 1 of the `status` parameter, which indicates whether the packet was sent by the central (1) or the peripheral (0). The source and the RSSI are displayed in the console. See Figure 3.
5. To stop monitoring, run "`packetmon 0 stop`" shell command on either Anchor (the second one is recommended, as its console is not flooded by packet information messages).

Parent topic:*Running the Connection Handover scenario*

Running the A2B scenario

The A2B feature allows for the secure transfer of Bluetooth Low Energy security keys information between the Car Anchor devices. The feature is currently supported on the following platforms:

- KW45B41Z-EVK with a lifecycle state of OEM-Open or higher, this feature is provided by the EdgeLock Secure Enclave.
- KW47-EVK with a lifecycle state of OEM-Open or higher, this feature is provided by th EdgeLock Secure Enclave.
- FRDM-KW43 with a lifecycle state of OEM-Open or higher, this feature is provided by the EdgeLock Secure Enclave.

The Car Anchor demo application makes use of the A2B feature to securely synchronize the local IRK and the Bonding Data (LTK and peer IRK) with another Car Anchor.

Prerequisites:

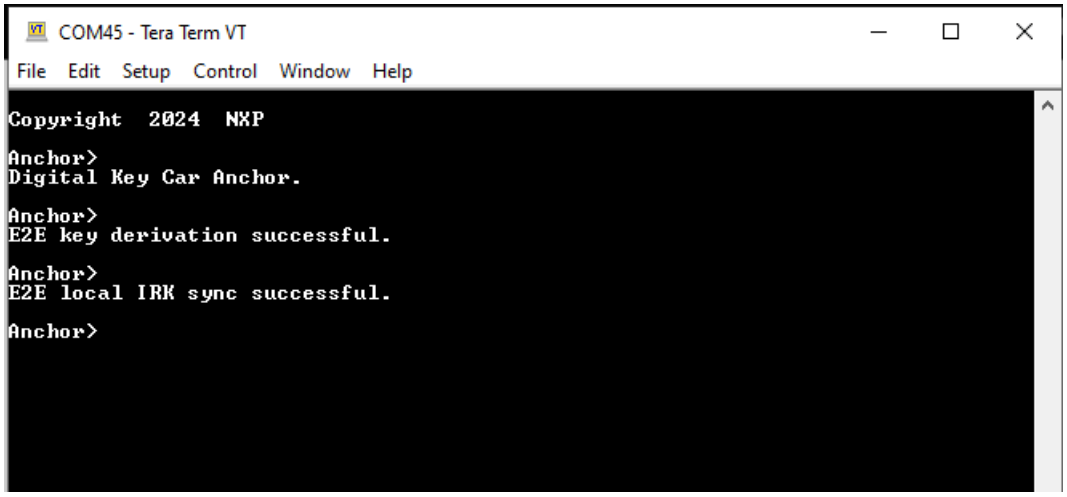
- Three boards (two act as Car Anchors, one as a Device). The demo is currently limited to a maximum of two Car Anchors.

- The two Car Anchors must be connected via a serial interface as described in the section above.
- Advanced Secure Mode must be enabled. The following macros must be defined and set to 1 in app_preinclude.h for the ***digital_key_car_anchor*** project:
  - gAppSecureMode_d
  - gA2BEnabled_d
- One of the Car Anchors must be configured with the gA2BInitiator_d macro set to 1, this is called Car Anchor A, and the other must be configured with the gA2BInitiator_d macro set to 0, called Car Anchor B. Car Anchor A triggers the EdgeLock-to-EdgeLock (E2E) key derivation and local IRK synchronization. Therefore, it should be started up after Car Anchor B.

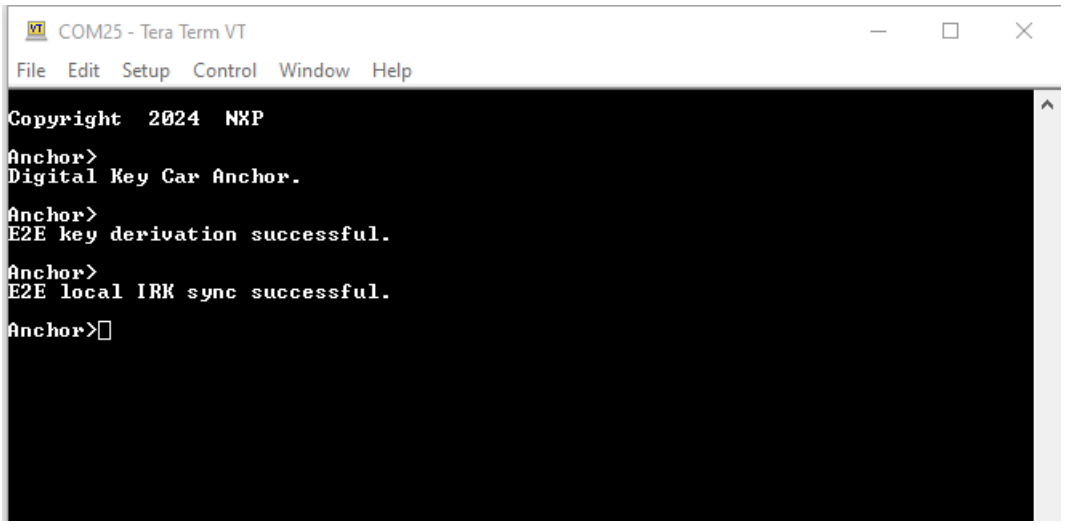
Demo steps:

- Start Car Anchor B, then Car Anchor A. At initialization the E2E key is derived and the local IRK of Car Anchor A is sent as a secure blob to Car Anchor B in order for both Car Anchors to have the same local IRK as shown in the Figure 1 and Figure 2.

Car Anchor A starts up, derives the E2E key and syncs with Car Anchor B



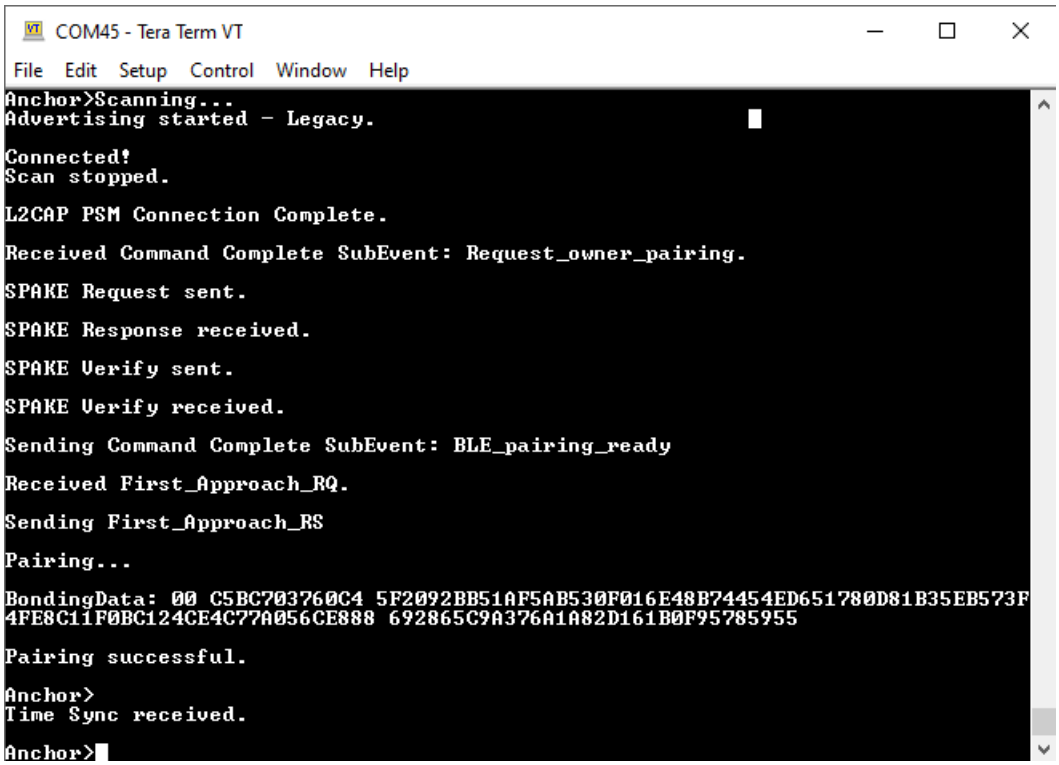
Car Anchor B starts up and syncs with Car Anchor B



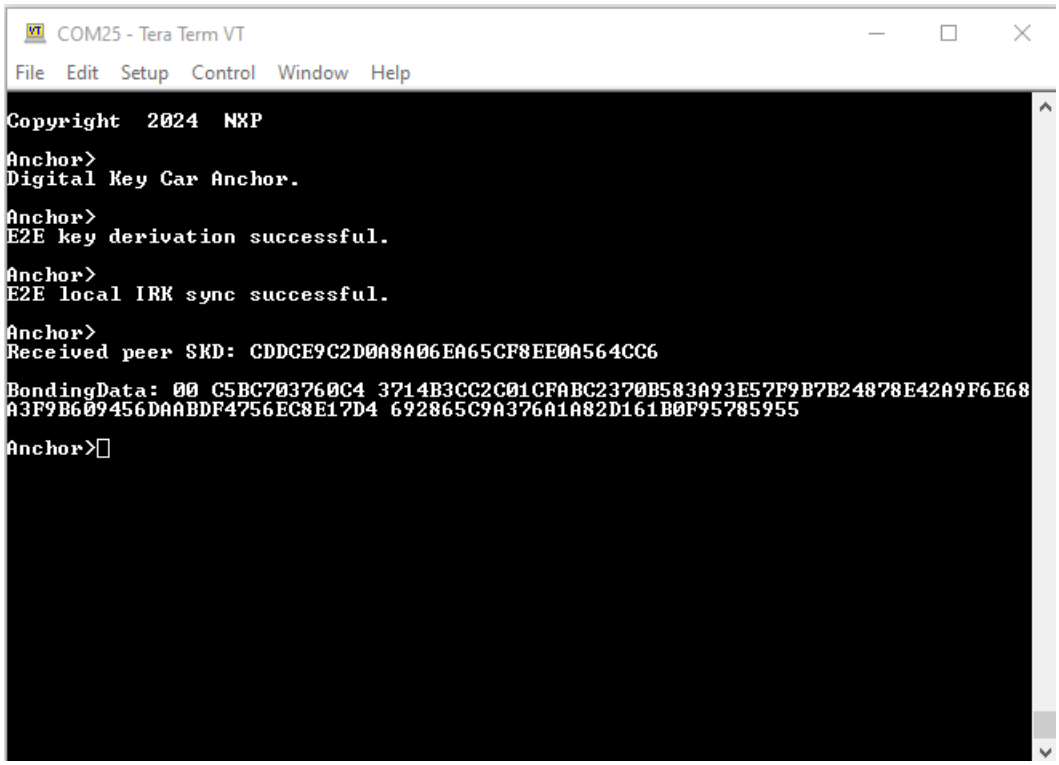
- Trigger the Owner Pairing scenario by sending the "sd" command on the Device and "sd op" on Car Anchor A. Once the bond is created, the Bonding Data containing secured blobs for the LTK and peer IRK is sent to Car Anchor B. The Car Anchors display the Bonding Data in shell with the LTK in blob form (the LTK is never available in plain text) and the peer IRK in

plain text. Now both Car Anchors have the same Bonding Data. Refer Figure 3 and Figure 4.

Car Anchor A pairs with Device



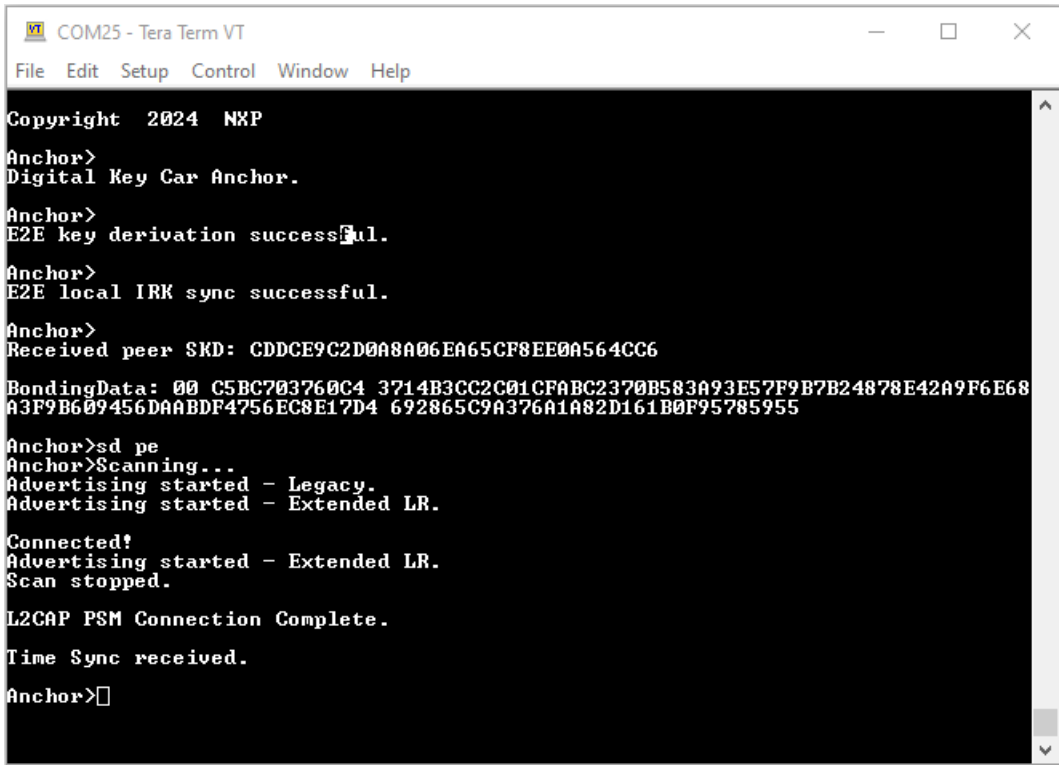
Car Anchor B has received the Bonding Data via the serial interface



- Disconnect Car Anchor A by sending the "dcnt" command.
- Once Car Anchor A is disconnected, send "sd" on the Device and "sd pe" on Car Anchor B. Car Anchor B is now able to perform a Passive Entry connection, without pairing as shown

in Figure 5.

Car Anchor B performs Passive Entry



Running Passive Entry Scenario with Decision Based Advertising Filtering (DBAF)

Decision Based Advertising Filtering can be used in the Passive Entry scenario. Using DBAF optimizes the scanning operation performed by the Digital Key Device of the advertisements sent by the Digital Key Car Anchor devices. This is accomplished by using 'ADV_DECISION_IND' PDUs, which contain the advertising address in the extended header. Using these PDUs allows the scanner to perform address resolution without the need to retrieve the AUX_ADV_IND PDU sent on the data channels.

The DBAF feature is supported on the following platforms:

- KW45B41Z-EVK
- KW47-EVK
- FRDM-KW43

Enabling DBAF

To enable DBAF, modify the Digital Key applications as per the steps listed below:

- In app_preinclude.h file, set gBLE60_DecisionBasedAdvertisingFilteringSupport_d to 1.

Parent topic:*Running Passive Entry Scenario with Decision Based Advertising Filtering (DBAF)*

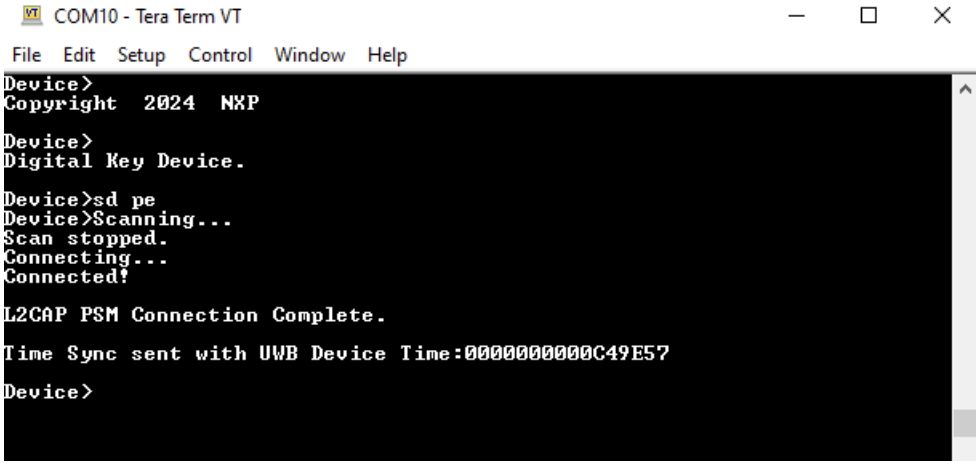
Running the Passive Entry with DBAF scenario

The Passive Entry with DBAF scenario runs after Owner Pairing between a Car Anchor and a Device successfully completes.

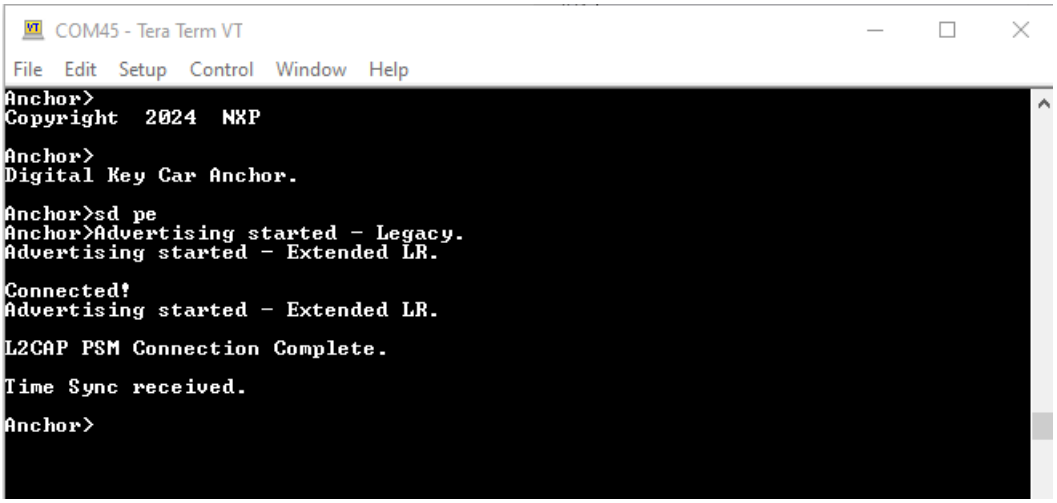
1. To run the Passive Entry with DBAF demo, reset the recently paired boards using either the **RST** switch or the "reset" command in the shell. Do NOT perform a factory reset, as this removes the bonding information from non-volatile memory.

2. After reset, run the "sd pe" command on the Car Anchor and "sd pe" command on the Device as shown in Figure 1 and Figure 2.

Passive Entry with DBAF on Device



Passive Entry with DBAF on Car Anchor



After following these steps, the Passive Entry scenario as described in Passive Entry Scenario runs.

Parent topic:*Running Passive Entry Scenario with Decision Based Advertising Filtering (DBAF)*

Intrusion Detection System

The Intrusion Detection System (IDS) is a proprietary feature which allows the Car Anchor to monitor anomalous events related to the traffic and conditions generated during the operation of the Bluetooth LE software stack. These events are detected at Host and Controller level. The application registers a callback where it receives and handles reports.

The IDS feature is supported on the following platforms:

- KW45B41Z-EVK
- KW47-EVK
- FRDM-KW43

To enable IDS, modify the Digital Key Car Anchor as per the steps listed below:

- In app_preinclude.h file, set gIntrusionDetectionSystem_d to 1.

Intrusion Detection System Test Mode

The IDS Test Mode is a development and testing feature that allows developers to simulate and validate IDS security event detection without requiring actual attack scenarios or malicious devices.

Enabling Test Mode

To enable IDS Test Mode, set gIntrusionDetectionSystemTestMode_d to 1 in app_preinclude.h.

This enables:

- Shell command idstest for triggering security events
- Pairing message interception hooks
- Test packet generation utilities

Supported Security Events

IDS Test Mode supports triggering the following security events:

Bit	Event Macro	Description	Category	Trigger Method
0	gSecEvt_Sweyntooth_MaxEncKeyS	Max Encryption Key Size (Sweyntooth)	Pairing	Hook
1	gSecEvt_DhKeyCheckZero_c	DH Key Check Zero	Pairing	Hook
2	gSecEvt_UnexpectedSmpMessage_	Unexpected SMP Message	Pairing	Hook
5	gSecEvt_MicFailureDisconnect_c	MIC Failure Disconnect	Connection	Packet
6	gSecEvt_EncReqNonBonded_c	Non-Bonded Device	Connection	Packet
7	gSecEvt_MalformedL2cap_c	Malformed L2CAP Packet	Connection	Packet
8	gSecEvt_MalformedAtt_c	Malformed ATT Packet	Connection	Packet
13	gSecEvt_ForcedRepairing_c	Forced Re-pairing	Pairing	Hook
14	gSecEvt_L2capFragmentsExceedMT	L2CAP Fragments Exceed MTU (Blue Frag)	Connection	Packet
16	gSecEvt_KNOB_c	Key Negotiation over Bluetooth (KNOB)	Pairing	Hook
18	gSecEvt_InvalidPublicKey_c	Invalid Public Key	Pairing	Hook
19	gSecEvt_WrongConfirmValue_c	Wrong Confirm Value	Pairing	Hook
20	gSecEvt_DHKeyCheckFailed_c	DH Key Check Failed	Pairing	Hook

Event Categories

Connection Events (Immediate Trigger)

These events are triggered immediately when the idstest command is executed:

- **BIT5**: MIC Failure Disconnect
- **BIT7**: Malformed L2CAP Packet
- **BIT8**: Malformed ATT Packet
- **BIT14**: L2CAP Fragments Exceed MTU

BIT6: Encryption Request from Non-Bonded Device

Natural Scenario:

1. Connect device to anchor with bonding enabled
2. Complete pairing and bonding successfully
3. Disconnect device
4. On anchor: Run factoryreset command
5. On device: Reconnect without clearing bonds
6. IDS event triggers when device sends encryption request

Pairing Events (Hook-based Trigger)

These events activate a message interception hook and trigger during the next pairing procedure:

- **BIT0**: Sweyntooth Max Encryption Key Size
- **BIT1**: DH Key Check Zero
- **BIT2**: Unexpected SMP Message
- **BIT13**: Forced Re-pairing
- **BIT16**: KNOB (Key Negotiation)
- **BIT18**: Invalid Public Key
- **BIT19**: Wrong Confirm Value
- **BIT20**: DH Key Check Failed

Using the idstest Command

idstest <event_bit> [device_id]

Examples

Trigger Connection Event (Immediate):

```
Anchor> idstest 5 0
[IDS TEST] Event BIT5 (mask 0x20000000) triggered for device 0.
Anchor> Received IDS event 32 triggered by peer 25008F195173
```

Trigger Pairing Event (During Next Pairing):

```
Anchor> idstest 0 0
[IDS TEST] Pairing event BIT0 (mask 0x00000001) will be triggered on next pairing.
[IDS TEST] Hook activated. Now run 'sd op' to start Owner Pairing.

Anchor> sd op
Advertising started - Legacy.
Connected!
Received IDS event 65536 triggered by peer 1BE0914D955A
```

References

For more information, refer to the following documents:

- Bluetooth Low Energy Application Developer's Guide (KW45_BLEADG)
- Bluetooth Low Energy Host Stack API Reference Manual (KW45_BLEHSAPIRM)

- Bluetooth Low Energy Host Stack FSCI (Framework Serial Connectivity Interface) API Reference Manual

  (KW45_BLEHSFSCIRM)
- Connectivity Framework Reference Manual (KW45_CONNFWRM)
- Bluetooth Low Energy CCC Digital Key with Channel Sounding Application Note (AN13979)

Acronyms and abbreviations

Table 1 lists the acronyms used in this document.

Acronym	Description
Bluetooth LE	Bluetooth Low Energy
CCC	Car Connectivity Consortium
DBAF	Decision Based Advertising Filtering
IDE	Integrated Design Environment
IDS	Intrusion Detection System
E2E	EdgeLock-to-EdgeLock
IRK	Identity Resolving Key
LTK	Long Term Key
OOB	Out-of-Band
RSSI	Received Signal Strength Indicator
SDK	Software Development Kit
UWB	Ultra-Wideband

Note about the source code in the document

Example code shown in this document has the following copyright and BSD-3-Clause license:

Copyright 2020-2025 NXP Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Bluetooth Low Energy Host Stack FSCI Application Programming Kinetis FSCI Host Application Programming Interface

About This Document

This document provides a detailed description for the Host Application Programming Interface (Host API) implementing the Framework Serial Connectivity Interface (FSCI) on a peripheral port such as UART, USB, and SPI. The Host API can be deployed from a PC tool or a host processor to perform control and monitoring of a wireless protocol stack running on the microcontroller. The software modules and libraries implementing the Host API is the Host Software Development Kit (SDK).

This version of the document describes the Bluetooth® Low Energy stack running on the Wireless Connectivity Microcontrollers (MCUs), which are interfaced from a high-level OS (Linux® OS, Windows® OS) by the Host API and the Host SDK.

Audience

This document is for software developers who create tools and multichip partitioned systems using a serial interface to a Bluetooth LE *black box* firmware running on a microcontroller.

Parent topic: *Bluetooth Low Energy Host Stack FSCI Application Programming*

Deploying Host Controlled Firmware

IAR Embedded Workbench for Arm® (EWARM) and ARMGCC are the development toolchains used to deploy the Bluetooth LE host stack software applications.

Detailed information about how to build, deploy and debug an IAR or ARMGCC-based project are presented in the *Bluetooth Low Energy Demo Applications User Guide*.

Bluetooth LE application configuration

To exercise the Host API, the Bluetooth LE *black box* firmware is required to be flashed on a compatible platform. The Bluetooth LE *black box* is represented by the ble_fscibb application firmware that can be interfaced and configured with FSCI commands over the serial interface.

The user can compile the *black box* image of the ble_fscibb software application using IAR Embedded Workbench for Arm (EWARM) or ARMGCC. For information on how to build the application see additional documentation provided in the package.

Parent topic: *Bluetooth Low Energy Host Stack FSCI Application Programming*

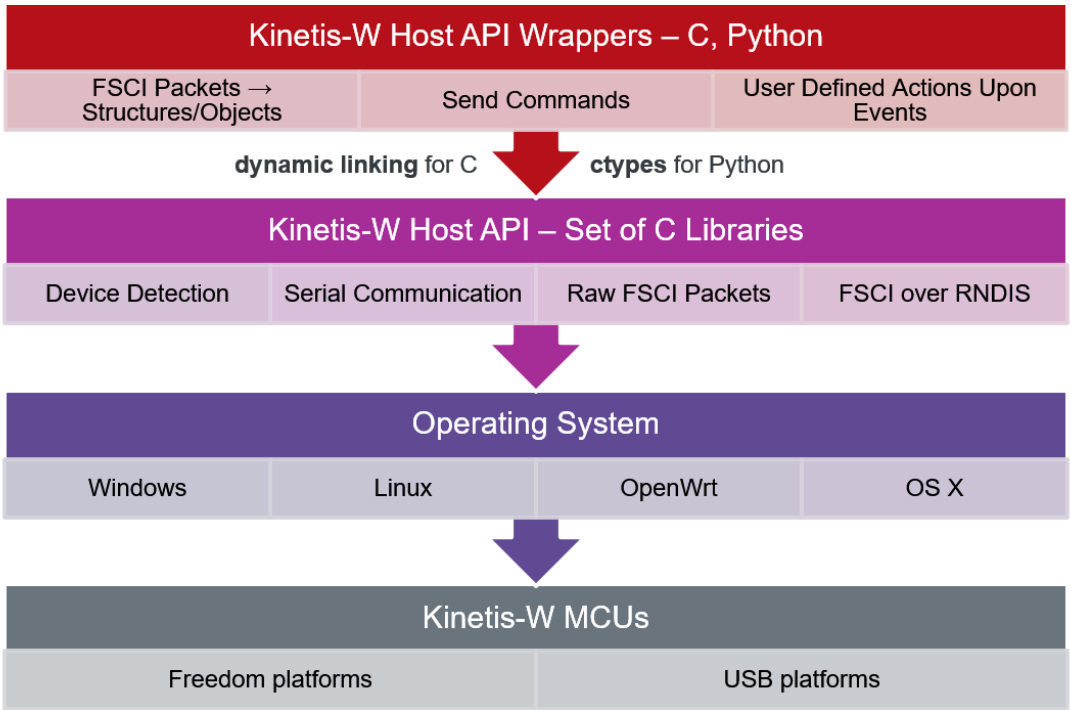
Host Software Overview

The FSCI (Framework Serial Communication Interface) module allows interfacing the protocol stack with a host system or PC tool using a serial communication interface.

FSCI can be demonstrated using various host software, one being the set of Linux OS libraries exposing the Host API described in this document. The NXP Test Tool for Connectivity Products PC application is another interfacing tool, running on the Windows OS. Bluetooth LE stack makes use of XML files which contain detailed meta-descriptors for commands and events transported over the FSCI.

The FSCI module is explained in the Wireless Framework - Services - FSCI documentation.

Wireless host software system block diagram



Directory tree

```
host_sdk
  hsdk
    demo                          A set of programs to demonstrate functionality.
      FSCIBootloader.c
      GetKinetisDevices.c             Outputs all Kinetis devices available on serial to the console.
      Makefile
      bin                         Folder containing libraries ".so".
    doc                           Folder containing documentation.
    include                        All the headers used are present in this folder.
      physical                     Headers specific to the physical serial bus or PCAP interface used by␣
↪the NXP device.
        PhysicalDevice.h
        PCAP
          PCAPDevice.h
        SPI
          SPIConfiguration.h        Handles SPI slave bus configuration(max speed Hz, bits per word).
          SPIDevice.h              Encapsulates an OS SPI device node into a well-defined structure.
        UART
          UARTConfiguration.h         Handles serial port configuration (baudrate, parity).
          UARTDevice.h             Encapsulates an OS UART device node into a well-defined structure.
          UARTDiscovery.h           Handles the discovery of UART connected devices.
      protocol                     Headers specific to the transmission of frames.
        Framer.h                    A state machine implementation for sending/ receiving frames.
        FSCI                       Headers specific to the FSCI protocol.
          FSCIFrame.h
          FSCIFramer.h
      sys                         General purpose headers for interaction with the OS, message queues and␣
↪more.
        EventManager.h                Handles event registering, notifying and callback submission.
        hsdkError.h              Macros for error reporting.
        hsdkLogger.h              Logger implementation for debugging.
        hsdkOSCommon.h                Interaction with OS specifics.
```

(continues on next page)

(continued from previous page)

```
            MessageQueue.h                A standard message queue implementation (linked list).
            RawFrame.h                    Describes the format of a frame, independent of the protocol.
            utils.h                       Various functions to manipulate structures and byte arrays.
        physical                          Implementation of the physical UART/SPI serial bus or PCAP interface
↪module.
          PCAP
            PCAPDevice.c
          PhysicalDevice.c
          SPI
            SPIConfiguration.c
            SPIDevice.c
          UART
            UARTConfiguration.c
            UARTDevice.c
            UARTDiscovery.c
        protocol                          Implementation of the protocol module relating to FSCI.
          Framer.c
          FSCI
            FSCIFrame.c
            FSCIFramer.c
        res
          77-mm-usb-device-blacklist.rules    Udev rules for disabling ModemManager.
          hsdk.conf                       Configuration file to control FSCI-ACKs.
        sys                               Implementation of the system/OS portable base module.
          EventManager.c
          hsdkEvent.c
          hsdkFile.c
          hsdkLock.c
          hsdkLogger.c
          hsdkSemaphore.c
          hsdkThread.c
          MessageQueue.c
          RawFrame.c
          utils.c
        ConnectivityLibrary.sln           Microsoft Visual Studio solution file.
        HSDK.rc
        Makefile
        README.md
        resource.h
    hsdk-c                                Described in Host API C Bindings
    hsdk-python                           Described in Host API Python Bindings
    README.txt
```

Device detection

The Wireless Host SDK can detect every USB attached peripheral device to a PC. On Linux OS, this is done via udev. Udev is the device manager for the Linux OS kernel and was introduced in Linux OS 2.5. Using the manager, the Wireless Host API can provide the Linux OS path for a device (for example, /dev/ttyACM0) and whether the device is a supported USB device, based on the vendor ID/product ID advertised. Upon device insertion, the USB cdc_acm kernel module is triggered by the kernel for interaction with TWR, USB and FRDM devices.

On Windows OS, attached peripherals are retrieved from the registry path HKEY_LOCAL_MACHINE\HARDWARE\DEVICEMAP\SERIALCOMM, resulting in names such as *COMxx* which must be used as input strings for the Python scripts which require a device name.

Serial port configuration

The Host SDK configures a serial UART port with the following default values:

Host SDK – UART default values

Configuration	Value
Baudrate	115200
ByteSize	EIGHTBITS
StopBits	ONE_STOPBIT
PARITY	NO_PARITY
HandleDSRControl	0
HandleDTRControl	ENABLEDTR
HandleRTSControl	ENABLERTS
InX	0
OutCtsFlow	1
OutDSRFlow	1
OutX	0

The library only allows the possibility to change the baudrate, as this is the most common scenario.

> **NOTE**: For devices using a USB connection interfaced directly (where the USB stack runs on the device and the system is NOT using an OpenSDA UART to USB converter), the baudrate is not necessary and setting it has no effect.

The Host SDK configures a serial SPI port with the following default values:

Host SDK – SPI default values

Configuration	Value
Transfer Mode	Mode SPI_MODE_0
Maximum SPI transfer speed (Hz)	1 MHz
Bits per word	8

The library only allows the possibility to change the maximum SPI transfer speed.

Logger

The Host SDK implements a logger functionality which is useful for debugging. Adding the compiler flag USE_LOGGER enables this functionality.

When running programs that make use of the Host SDK, a file named hsdk.log appears in the working directory. This is an excerpt from the log:

```
HSDK_INFO - [6684] [PhysicalDevice]InitPhysicalDevice:Allocated memory for PhysicalDevice
HSDK_INFO - [6684] [PhysicalDevice]InitPhysicalDevice:Initialized device's message queue
HSDK_INFO - [6684] [PhysicalDevice]InitPhysicalDevice:Created event manager for device
HSDK_INFO - [6684] [PhysicalDevice]InitPhysicalDevice:Created threadStart event
HSDK_INFO - [6684] [PhysicalDevice]InitPhysicalDevice:Created stopThread event
HSDK_INFO - [6684] [PhysicalDevice]AttachToConcreteImplementation:Attached to a concrete␣
↪implementation
HSDK_INFO - [6684] [PhysicalDevice]InitPhysicalDevice:Created and start device thread
HSDK_INFO - [6684] [Framer]InitializeFramer:Allocated memory for Framer
HSDK_INFO - [6684] [Framer]InitializeFramer:Created stopThread event
HSDK_INFO - [6684] [Framer]InitializeFramer:Initialized framer's message queue
HSDK_INFO - [6684] [Framer]InitializeFramer:Created event manager for framer
```

Parent topic: *Bluetooth Low Energy Host Stack FSCI Application Programming*

Linux OS Host Software Installation Guide

Libraries

Prerequisites

Packages: build-essential, libudev, libudev-dev, libpcap, libpcap-dev. Use apt-get install on Debian-based distributions. The Linux OS kernel version must be greater than 3.2.

Installation

```
$ pwd
/home/user/hsdk
$ make
$ find build/ -name "\*.so"
build/libframer.so
build/libsys.so
build/libphysical.so
build/libuart.so
build/libfsci.so
build/libspi.so
$ sudo make install
```

By default, make generates sharedlibraries (having .so extension). The step make install (superuser privileges required) copies these libraries to /usr/local/lib, which is part of the default Linux OS library path. The installation prefix may be changed by passing the variable PREFIX, e.g. make install PREFIX=/usr/lib. The user is responsible for making sure that PREFIX is part of the system's LD_LIBRARY_PATH.

Static libraries can be generated instead, by modifying the LIB_OPTION variable in the Makefile from dynamic to static (.a extension).

make install also disables the ModemManager control for the connected devices. Otherwise, the daemon starts sending AT commands that affect the responsiveness of the afore-mentioned devices in the first 20 seconds after plug in.

Demos

Installation

```
$ pwd
/home/user/hsdk/demo
$ make
$ ls bin/
GetKinetisDevices
```

These demos are provided in this package:

- GetKinetisDevices: this program detects the boards connected to the serial line and outputs the path to the console:

  ```
  $ ./GetKinetisDevices
  NXP Kinetis-W device on /dev/ttyACM0.
  NXP Kinetis-W device on /dev/ttyACM1.
  ```

Parent topic: *Bluetooth Low Energy Host Stack FSCI Application Programming*

Windows OS Host Software Installation Guide

Libraries

Prerequisites

Microsoft Visual Studio® is required to build the host software. Open the solution file ConnectivityLibrary.sln and build it for either Win32 or x64, depending on your setup requirements. The output directory contains a file named HSDK.dll, which can be thought of as a bundle of all the

shared libraries from Linux OS, except for SPI (libspi.so). Currently, SPI interface to the board is not supported by the Windows host software.

Prebuilt HSDK.dll files are available under directory hsdk-python\lib.

Installation

The host software for the Windows OS is designed to work in a Python environment by contrast to the Linux OS where standalone C demos also exist.

Download and install the Python 3.10.x/32b package from Python Downloads. When customizing the installation options, check Add python.exe to Path.

Using prebuilt library

1. Depending on your Python environment architure (not Windows architecture) copy the appropriate HSDK.dll from hsdk-python\lib\[x86|x64]to \<Python Directory\>\DLLs.
2. Download and install Visual C++ Redistributable Packages for Microsoft Visual Studio, depending on the Windows architecture of your system (vcredist_x86.exe or vcredist_x64.exe) from Microsoft Visual C++ Redistributable latest supported downloads.
3. Download and install the Microsoft Visual C++ Compiler for Python as described in the python Windows Complilers.

Using local built library

1. Depending on your Python environment architure (not Windows architecture), build the appropriate Microsoft Visual Studio solution configuration and then copy HSDK.dll to \<Python Directory\>\DLLs.
2. Download and install the Microsoft Visual C++ Compiler for Python as described in the python Windows Complilers.

Optionally, copy the hsdk\res\hsdk.conf to \<Python Directory\>\DLLs to control the behavior of the FSCI-ACK synchronization mechanism.

Demos

See *Host API Python Bindings*.

Parent topic: *Bluetooth Low Energy Host Stack FSCI Application Programming*

Host API C Bindings

The Host SDK includes a set of C bindings to interface with a *black-box*. Bindings are generated from the matching FSCI XML file that is available in the stack software package under tools\wireless\xml_fsci. The bindings are designed to be platform agnostic, with a minimal set of OS abstraction symbols required for building. Thus, the files can be easily integrated on a wide range of host platforms.

Directory Tree

```
hsdk-c/
  demo
    HeartRateSensor.c      Source file that implements the Bluetooth LE Heart Sensor Profile.
    Makefile
  inc
    ble_sig_defines.h      Standard Bluetooth SIG UUID values.
    cmd_<name>.h           Generated from the matching FSCI <name>.xml file.
    os_abstraction.h       Provides OS dependent symbols for building the interface.
  README.md
  src
    cmd_<name>.c           Generated from the matching FSCI <name>.xml file.
```

(continues on next page)

(continued from previous page)

```
    evt_<name>.c            Generated from the matching FSCI <name>.xml file.
    evt_printer_<name>.c    Generated from the matching FSCI <name>.xml file.
```

Tests and examples

Tests and examples that make use of the C bindings are placed in the hsdk-c/demo directory. Example of usage:

```
$ cd hsdk-c/demo/
$ make
$ ./HeartRateSensor /dev/ttyACM0
[...]
--> Setup finished, please open IoT Toolbox -> Heart Rate -> HSDK_HRS
```

Development

Header file cmd_<name>.h is generated from the corresponding <name>.xml FSCI XML file.

- Enumerations

```
/* Indicates whether the connection request is issued for a specific device or for all the devices in the␣
↪White List - default specific device */
typedef enum GAPConnectRequest_FilterPolicy_tag {
    GAPConnectRequest_FilterPolicy_gUseDeviceAddress_c = 0x00,
    GAPConnectRequest_FilterPolicy_gUseWhiteList_c     = 0x01
} GAPConnectRequest_FilterPolicy_t;
```

- Structures

```
typedef PACKED_STRUCT GAPConnectRequest_tag {
    uint16_t ScanInterval;          // Scanning interval - default 10ms
    uint16_t ScanWindow;            // Scanning window - default 10ms
    GAPConnectRequest_FilterPolicy_t FilterPolicy;          // Indicates whether the connection␣
↪request is issued for a specific device or for all the devices in the White List - default specific device
    GAPConnectRequest_OwnAddressType_t OwnAddressType;      // Indicates whether the address␣
↪used in connection requests will be the public address or the random address - default public address
    GAPConnectRequest_PeerAddressType_t PeerAddressType;    // When connecting to a specific␣
↪device, this indicates that device's address type - default public address
    uint8_t PeerAddress[6];         // When connecting to a specific device, this indicates that device's␣
↪address
    uint16_t ConnIntervalMin;       // The minimum desired connection interval - default 100ms
    uint16_t ConnIntervalMax;       // The maximum desired connection interval - default 200ms
    uint16_t ConnLatency;           // The desired connection latency (the maximum number of␣
↪consecutive connection events the Slave is allowed to ignore) - default 0
    uint16_t SupervisionTimeout;    // The maximum time interval between consecutive over-the-air␣
↪packets; if this timer expires, the connection is dropped - default 10s
    uint16_t ConnEventLengthMin;    // The minimum desired connection event length - default 0ms
    uint16_t ConnEventLengthMax;    // The maximum desired connection event length - default␣
↪maximum possible, ~41 s
    bool_t usePeerIdentityAddress;  // TRUE if the address defined in the peerAddressType and␣
↪peerAddress is an identity address
} GAPConnectRequest_t;
```

- Container for all possible event types

```
typedef struct bleEvtContainer_tag {
    uint16_t id;
    union {
        [...]
        GAPConnectionEventConnectedIndication_t GAPConnectionEventConnectedIndication;
```

(continues on next page)

(continued from previous page)

```
        [...]
    } Data;
} bleEvtContainer_t;
```

- Prototypes for sending commands

```
memStatus_t GAPConnectRequest(GAPConnectRequest_t *req, void *arg, uint8_t fsciInterface);
```

Header file os_abstraction.h provides the required symbols for building the generated interface. When integrating in a project different that Host SDK, the user needs to provide the implementation for

```
void FSCI_transmitPayload (
    void *arg,              /* Optional argument passed to thefunction */
    uint8_t og,              /* FSCI operation group */
    uint8_t oc,              /* FSCI operation code */
    void *msg,               /* Pointer to payload */
    uint16_t msgLen,           /* Payload length */
    uint8_t fsciInterface    /* FSCI interface ID */
);
```

that creates and sends a FSCI packet (0x02 | og | oc | msgLen | msg | crc +- fsciInterface) on the serial interface. Source files cmd_<name>.c, evt_<name>.c and evt_printer_<name>.c are generated from the correspondent <NAME>.xml FSCI XML file.

- Functions that handle command serialization in cmd_<name>.c

```
memStatus_t GAPConnectRequest(GAPConnectRequest_t *req, void *arg, uint8_t fsciInterface) {
    /* Sanity check */
    if (!req)
    {
        return MEM_UNKNOWN_ERROR_c;
    }
    FSCI_transmitPayload(arg, 0x48, 0x1C, req, sizeof(GAPConnectRequest_t), fsciInterface);
    return MEM_SUCCESS_c;
}
```

- Event dispatcher in evt_<name>.c

```
void KHC_BLE_RX_MsgHandler(void *pData, void *param, uint8_t fsciInterface) {
    if (!pData || !param)
    {
        return;
    }

    fsciPacket_t *frame = (fsciPacket_t *)pData;
    bleEvtContainer_t *container = (bleEvtContainer_t *)param;
    uint8_t og = frame->opGroup;
    uint8_t oc = frame->opCode;
    uint8_t *pPayload = frame->data;
    uint16_t id = (og << 8) + oc, i;

    for (i = 0; i < sizeof(evtHandlerTbl) / sizeof(evtHandlerTbl[0]); i++)
    {
        if (evtHandlerTbl[i].id == id)
        {
            evtHandlerTbl[i].handlerFunc(container, pPayload);
            break;
        }
    }
}
```

- Handler functions to perform event de-serialization in evt_<name>.c

```
static memStatus_t Load_GAPConnectionEventConnectedIndication(bleEvtContainer_t *container,␣
↪uint8_t *pPayload)
{
    GAPConnectionEventConnectedIndication_t *evt = &(container->Data.
↪GAPConnectionEventConnectedIndication);

    uint32_t idx = 0;

    /* Store (OG, OC) in ID */
    container->id = 0x489D;

    evt->DeviceId = pPayload[idx]; idx++;
    FLib_MemCpy(&(evt->ConnectionParameters.ConnInterval), pPayload + idx, sizeof(evt->
↪ConnectionParameters.ConnInterval)); idx += sizeof(evt->ConnectionParameters.ConnInterval);
    FLib_MemCpy(&(evt->ConnectionParameters.ConnLatency), pPayload + idx, sizeof(evt->
↪ConnectionParameters.ConnLatency)); idx += sizeof(evt->ConnectionParameters.ConnLatency);
    FLib_MemCpy(&(evt->ConnectionParameters.SupervisionTimeout), pPayload + idx, sizeof(evt->
↪ConnectionParameters.SupervisionTimeout)); idx += sizeof(evt->ConnectionParameters.
↪SupervisionTimeout);
    evt->ConnectionParameters.MasterClockAccuracy = (GAPConnectionEventConnectedIndication_
↪ConnectionParameters_MasterClockAccuracy_t)pPayload[idx]; idx++;
    evt->PeerAddressType = (GAPConnectionEventConnectedIndication_PeerAddressType_
↪t)pPayload[idx]; idx++;
    FLib_MemCpy(evt->PeerAddress, pPayload + idx, 6); idx += 6;
    evt->peerRpaResolved = (bool_t)pPayload[idx]; idx++;
    evt->localRpaUsed = (bool_t)pPayload[idx]; idx++;

    return MEM_SUCCESS_c;
}
```

- Event status console printer in evt_printer_<name>.c

```
void SHELL_BleEventNotify(void *param)
{
    bleEvtContainer_t *container = (bleEvtContainer_t *)param;

    switch (container->id) {
        [...]
        case 0x489D:
            shell_write("GAPConnectionEventConnectedIndication");
            shell_write(" -> ");
            switch (container->Data.GAPConnectionEventConnectedIndication.PeerAddressType)
            {
                case GAPConnectionEventConnectedIndication_PeerAddressType_gPublic_c:
                    shell_write(gPublic_c);
                    break;
                case GAPConnectionEventConnectedIndication_PeerAddressType_gRandom_c:
                    shell_write(gRandom_c);
                    break;
                default:
                    shell_printf("Unrecognized status 0x%02X", container->Data.
↪GAPConnectionEventConnectedIndication.PeerAddressType);
                    break;
            }
            break;
        [...]
    }
}
```

Parent topic: *Bluetooth Low Energy Host Stack FSCI Application Programming*

Host API Python Bindings

Prerequisites

Python 3.10.x/32b is necessary to run the Python bindings. The bindings use the Host API C libraries. On Linux and OS X operating systems, these are called from the installation location which is /usr/local/lib, while on Windows OS the library file is loaded in <Python Install Directory>\DLLs.

Platform setup

To run scripts from the command line, the PYTHONPATH must be set accordingly, so that the interpreter can find the imported modules.

Linux OS

Adding the source folder to the PYTHONPATH can be done by editing ~/.bashrc and adding the following line:

```
export PYTHONPATH=\$PYTHONPATH:/home/.../hsdk-python/src
```

Most of the Python scripts operate on boards connected on a serial bus and superuser privileges must be employed to access the ports. After running a command prefixed with sudo, the environment paths become those of root, so the locally set PYTHONPATH is not visible anymore. That is why /etc/sudoers is modified to keep the environment variable when changing user.

Edit /etc/sudoers with your favorite text editor. Modify:

```
Defaults env_reset -> Defaults env_keep="PYTHONPATH"
```

As an alternative to avoid modifying the :sudoers file, the PYTHONPATH can be adjusted programmatically, as in the example below:

```
import sys

if sys.platform.startswith('linux'):
    sys.path.append('/home/user/hsdk-python/src')
```

Windows OS

Add the source folder to the PYTHONPATH by following these steps:

1. Navigate to My Computer > Properties > Advanced System Settings > Environment Variables > System Variables.
2. Modify existing or create new variable named PYTHONPATH, with the absolute path of tools\wireless\host_sdk\hsdk-python\src.

Directory Tree

```
lib                                 Compiled host SDK libraries for Windows.
    README.md
    x64
        HSDK.dll
    x86
        HSDK.dll
src
    com
        nxp
            wireless_connectivity
                commands
                    ble             Generated files for Bluetooth LE support.
```

(continues on next page)

(continued from previous page)

```
                ble_sig_defines.py
                enums.py
                events.py
                frames.py
                gatt_database_dynamic.py
                heart_rate_interface.py
                __init__.py
                operations.py
                spec.py
                sync_requests.py
            comm.py
            firmware                Generated files for OTA/FSCI bootloader support.
                enums.py
                events.py
                frames.py
                __init__.py
                operations.py
                spec.py
                sync_requests.py
            fsci_data_packet.py
            fsci_frame_description.py
            fsci_operation.py
            fsci_parameter.py
            __init__.py
        hsdk
            CFsciLibrary.py
| |         config.py               Configuration file for the Python Host SDK subsystem.
            CUartLibrary.py
            device
                device_manager.py
                __init__.py
                physical_device.py
            framing
                fsci_command.py
                fsci_framer.py
                __init__.py
            __init__.py
            library_loader.py
            ota_server.py
            singleton.py
            sniffer.py
            utils.py
        __init__.py
        test                        Test and proof of concept scripts.
            hrs.py                  Script implementing a Bluetooth LE Heart Sensor profile .
            __init__.py
    __init__.py
__init__.py
__init__.py
```

Functional description

The interaction between Python and the C libraries is made by the ctypes module. Ctypes provides C compatible data types, and allows calling functions in DLLs or shared libraries. It can be used to wrap these libraries in pure Python. Because the use of shared libraries is a requirement, the LIB_OPTION variable must remain set on “dynamic” in hsdk/Makefile. Ctypes made into mainline Python starting with version 2.5.

The Python Bindings expose Bluetooth LE familiar API in the `com.nxp.wireless_connectivity.commands` package. Such a package contains the following modules:

```
bluetooth le
    enums.py            – Classes that resemble enums, constants are generated here.
    events.py           – Observer classes that can build an object from a byte array and deliver it to the user.
    frames.py           – Classes that map on Bluetooth LE/Firmware FSCI messages.
    operations.py       – Each Operation class encapsulates a request and one or multiple events that are to be␣
↪generated by the request.
    spec.py             – This file describes the name, size, order and relationship between the command␣
↪parameters.
    sync_requests.py    – Each Synchronous Request is a method which sends a request and returns the␣
↪triggered event.
```

Tests and examples are placed in the package com.nxp.wireless_connectivity.test.

Bluetooth LE Heart Rate Service use case

The Heart Rate Service is presented as use case for using the API of a Bluetooth LE *black box*, located in the example hsdk-python/src/com/nxp/wireless_connectivity/test/hrs.py.

The example populates the GATT Database dynamically with the GATT, GAP, heart rate, battery and device information services. It then configures the Bluetooth LE stack and starts advertising. There are also two connect and disconnect observers to handle specific events.

The user needs to connect to the Bluetooth LE or hybrid *black box* through a serial bus port that is passed as a command line argument, for example, /dev/ttyACM0.

```
# python hrs.py -h
usage: hrs.py [-h] [-p] serial_port

Bluetooth LE demo app which implements a ble_fsci_heart_rate_sensor.

positional arguments:
    serial_port Kinetis-W system device node.
optional arguments:
    -h, --help  show this help message and exit
    -p, --pair  Use pairing
```

It is important to first execute a CPU reset request to the Bluetooth LE *black box* before performing any other configuration to reset the Buetooth LE stack. This is done by the following command:

```
FSCICPUReset(serial_port, protocol=Protocol.BLE)
```

User sync request example

It is recommended for the user to access the Bluetooth LE API through sync requests. GATTDBDynamicAddCharacteristicDeclarationAndValue API is used as an example:

```
def gattdb_dynamic_add_cdv(self, char_uuid, char_prop, maxval_len, initval, val_perm):
    '''
    Declare a characteristic and assign it a value.

    @param char_uuid: UUID of the characteristic
    @param char_prop: properties of the characteristic
    @param maxval_len: maximum length of the value
    @param initval: initial value
    @param val_perm: access permissions on the value
    @return: handle of the characteristic
    '''
    ind = GATTDBDynamicAddCharacteristicDeclarationAndValue(
        self.serial_port,
        UuidType=UuidType.Uuid16Bits,
        Uuid=char_uuid,
```

(continues on next page)

(continued from previous page)

```
        CharacteristicProperties=char_prop,
        MaxValueLength=maxval_len,
        InitialValueLength=len(initval),
        InitialValue=initval,
        ValueAccessPermissions=val_perm,
        protocol=self.protocol
    )
    if ind is None:
        return self.gattdb_dynamic_add_cdv(char_uuid, char_prop, maxval_len, initval, val_perm)

    print '\tCharacteristic Handle for UUID 0x%04X ->' % char_uuid, ind.CharacteristicHandle
    self.handles[char_uuid] = ind.CharacteristicHandle
    return ind.CharacteristicHandle
```

Sync request internal implementation

As an example, for the GATTDBDynamicAddCharacteristicDeclarationAndValue API, the command is executed through a synchronous request. The sync request code creates an object of the following class:

```
class GATTDBDynamicAddCharacteristicDeclarationAndValueRequest(object):

    def __init__(self,
UuidType=GATTDBDynamicAddCharacteristicDeclarationAndValueRequestUuidType.Uuid16Bits, Uuid=[],
CharacteristicProperties=GATTDBDynamicAddCharacteristicDeclarationAndValueRequestCharacteristicProperties.
↪gNone_c,
MaxValueLength=bytearray(2), InitialValueLength=bytearray(2), InitialValue=[],
ValueAccessPermissions=GATTDBDynamicAddCharacteristicDeclarationAndValueRequestValueAccessPermissions.
↪gPermissionNone_c):
    '''
    @param UuidType: UUID type
    @param Uuid: UUID value
    @param CharacteristicProperties: Characteristic properties
    @param MaxValueLength: If the Characteristic Value length is variable, this is the maximum length; for␣
↪fixed lengths this must be set to 0
    @param InitialValueLength: Value length at initialization; remains fixed if maxValueLength is set to 0,␣
↪otherwise cannot be greater than maxValueLength
    @param InitialValue: Contains the initial value of the Characteristic
    @param ValueAccessPermissions: Access permissions for the value attribute
    '''
    self.UuidType = UuidType
    self.Uuid = Uuid
    self.CharacteristicProperties = CharacteristicProperties
    self.MaxValueLength = MaxValueLength
    self.InitialValueLength = InitialValueLength
    self.InitialValue = InitialValue
    self.ValueAccessPermissions = ValueAccessPermissions
```

An operation is represented by an object of the following class:

```
class GATTDBDynamicAddCharacteristicDescriptorOperation(FsciOperation):

    def subscribeToEvents(self):
        self.spec = Spec.GATTDBDynamicAddCharacteristicDescriptorRequestFrame
        self.observers =[GATTDBDynamicAddCharacteristicDescriptorIndicationObserver(
↪'GATTDBDynamicAddCharacteristicDescriptorIndication'), ]
        super(GATTDBDynamicAddCharacteristicDescriptorOperation, self).subscribeToEvents()
```

The Spec object is initialized and set to zero in the FSCI packet any parameter not passed through the object of a class, depending on its length. Also, when defining such an object, the parameters may take simple integer, boolean or even list values instead of byte arrays, the values are serialized as a byte stream.

The observer is an object of the following class:

```
class GATTDBDynamicAddCharacteristicDeclarationAndValueIndicationObserver(Observer):
    opGroup = Spec.GATTDBDynamicAddCharacteristicDeclarationAndValueIndicationFrame.opGroup
    opCode = Spec.GATTDBDynamicAddCharacteristicDeclarationAndValueIndicationFrame.opCode

    @overrides(Observer)
    def observeEvent(self, framer, event, callback, sync_request):
    # Call super, print common information
    Observer.observeEvent(self, framer, event, callback, sync_request)
    #Get payload
    fsciFrame = cast(event, POINTER(FsciFrame))
    data = cast(fsciFrame.contents.data, POINTER(fsciFrame.contents.length * c_uint8))
    packet = Spec.GATTDBDynamicAddCharacteristicDeclarationAndValueIndicationFrame.
↪getFsciPacketFromByteArray(data.contents, fsciFrame.contents.length)
    # Create frame object
    frame = GATTDBDynamicAddCharacteristicDeclarationAndValueIndication()
    frame.CharacteristicHandle = packet.getParamValueAsNumber("CharacteristicHandle")
    framer.event_queue.put(frame) if sync_request else None

    if callback is not None:
        callback(frame)
    else:
        print_event(self.deviceName, frame)
    fsciLibrary.DestroyFSCIFrame(event)
```

The status of the request is printed at the console by the following general status handler:

```
def subscribe_to_async_ble_events_from(device, ack_policy=FsciAckPolicy.GLOBAL):
    ble_events = [
        L2CAPConfirmObserver('L2CAPConfirm'),
        GAPConfirmObserver('GAPConfirm'),
        GATTConfirmObserver('GATTConfirm'),
        GATTDBConfirmObserver('GATTDBConfirm'),
        GAPGenericEventInitializationCompleteIndicationObserver(
↪'GAPGenericEventInitializationCompleteIndication'),
        GAPAdvertisingEventCommandFailedIndicationObserver(
↪'GAPAdvertisingEventCommandFailedIndication'),
        GATTServerErrorIndicationObserver('GATTServerErrorIndication'),
        GATTServerCharacteristicCccdWrittenIndicationObserver(
↪'GATTServerCharacteristicCccdWrittenIndication')
    ]

    for ble_event in ble_events:
        FsciFramer(device, FsciAckPolicy.GLOBAL, Protocol.BLE, Baudrate.BR115200).addObserver(ble_
↪event)
```

Connect and disconnect observers

The following code adds observers for the connect and disconnect events in the user class:

```
class BLEDevice(object):
    '''
    Class which defines the actions performed on a generic Bluetooth LE device.
    Services implemented: GATT, GAP, Device Info.
    '''
    self.framer.addObserver(
        GAPConnectionEventConnectedIndicationObserver('GAPConnectionEventConnectedIndication'),
        self.cb_gap_conn_event_connected_cb)
    self.framer.addObserver(GAPConnectionEventDisconnectedIndicationObserver(
↪'GAPConnectionEventDisconnectedIndication'),
    self.cb_gap_conn_event_disconnected_cb)
```

where the callbacks are:

```
def cb_gap_conn_event_connected_cb(self, event):
    '''
    Callback executed when a smartphone connects to this device.
    @param event: GAPConnectionEventConnectedIndication
    '''
    print_event(self.serial_port, event)
    self.client_device_id = event.DeviceId
    self.gap_event_connected.set()

def cb_gap_conn_event_disconnected_cb(self, event):
    '''
    Callback executed when a smartphone disconnects from this device.
    @param event: GAPConnectionEventdisConnectedIndication
    '''
    print_event(self.serial_port, event)
    self.gap_event_connected.clear()
```

From an Android™ or iOS-based smartphone, the user can use the Bluetooth LE Toolbox application in the Heart Rate profile. Random heart rate measurements in the range 60-100 are displayed every second, while battery values change every 10 seconds.

Parent topic: *Bluetooth Low Energy Host Stack FSCI Application Programming*

FSCI Messages XML

Overview

The Framework Serial Communication Interface (FSCI) module supports interfacing the BLE Host Stack with a host or a PC tool (Test Tool for Connectivity Products) using a serial communication line. If the FSCI is used with the NXP Test Tool for Connectivity products, the tool can use an XML file containing a detailed meta-description for serial commands and events. This XML file is provided in the software packages that include this document.

Bluetooth LE Host Stack interface set

The Bluetooth LE Host Stack layers that offer access using FSCI are L2CAP, L2CAP CB, GATT, GATTDB, GAP, FSCI, and NVM. Each layer provides primitives that an upper layer (profile/application) uses to access services of that layer. This document includes a detailed list of the provided primitives.

Basic requirements

Following are the hardware requirements for using FSCI API RM:

- IAR Embedded Workbench for Arm architecture
- NXP development boards (to be loaded with the **ble_fscibb** application)
- Optional Bluetooth LE sniffer for testing
- PC tool (Test Tool for Connectivity Products) or another host processor capable of communicating through UART, USB, SPI, or I2C

FSCI message description

This chapter describes the L2CAP, L2CAP CB, GATT, GATTDB, GAP, FSCI, and NVM commands (requests) and events (confirms and indications) in detail.

The BLE.xml document is found in the ble_fsci_app subfolder, and provides an XML representation of all these commands and events. The BLE_x.x.x.xml document is found in the ..\tools\wireless\xml_fsci\ subfolder, and provides an XML representation of all these commands and events.

How to interpret the XML

The XML is structured in <Commands> and <Events> tags, the contents grouped by layer. Each group contains the description <GroupDesc> and multiple <Cmd> tags (as shown below).

```
<BLE>
 <Commands>
  <L2CAP>
   <GroupDesc>L2CAP Commands</GroupDesc>
   <Cmd>
    <CmdName>L2CAP-Init.Request</CmdName>
    <CmdDesc>L2CAP initialization function</CmdDesc>
    <CmdHeader>41 01</CmdHeader>
    <CmdParms>
    </CmdParms>
   </Cmd>
  [...]
  </L2CAP>
  <L2CAPCB>
  [...]
  </L2CAPCB>
 [...]
 </Commands>
 <Events>
  [...]
 </Events>
 [...]
</BLE>
```

Every command or event is described in a <Cmd> that contains:

- the name - <CmdName>
- the description - <CmdDesc>
- the opGroup and opCode - <CmdHeader>
- the parameters - <CmdParms>

The parameters of a command/event are specified using the <Parm> tag and contain:

- the name - <ParmName>
- the description - <ParmDesc>
- the size - <ParmSize>
- the type - <ParmType>
- the last value - <ParmLastValue>
- the default value - <ParmDefaultValue>
- the type specifics - <ParmTypeSpecifics>

The type specifics tag is used where needed to specify certain behaviours:

- <ParmItem> for specifying *enum* items
- <ParmState> for specifying the state of a variable (e.g. *ReadOnly*)
- <ParmRange> for specifying the range of the values taken by a variable
- <Depend> for specifying a dependence on another variable (e.g.an *array*'s dependence on the length variable)
- <Parm> for specifying the parameters of a *struct*

FSCI message example

For example, the L2CAP-SendAttData.Request command has the following XML structure:

```xml
<Cmd>
 <CmdName>L2CAP-SendAttData.Request</CmdName>
 <CmdDesc>Sends a data packet through ATT Channel</CmdDesc>
 <CmdHeader>41 03</CmdHeader>
 <CmdParms>
  <Parm>
   <ParmName>DeviceId</ParmName>
   <ParmDesc>The DeviceId for which the command is intended</ParmDesc>
   <ParmSize>1</ParmSize>
   <ParmType>tInt</ParmType>
   <ParmLastValue>00</ParmLastValue>
   <ParmDefaultValue>00</ParmDefaultValue>
  </Parm>
  <Parm>
   <ParmName>PacketLength</ParmName>
   <ParmDesc>Length of the ATT data packet to be sent</ParmDesc>
   <ParmSize>2</ParmSize>
   <ParmType>tInt</ParmType>
   <ParmLastValue>0001</ParmLastValue>
   <ParmDefaultValue>0001</ParmDefaultValue>
   <ParmTypeSpecifics>
    <ParmRange>0x0001-0xFFFF</ParmRange>
   </ParmTypeSpecifics>
  </Parm>
  <Parm>
   <ParmName>Packet</ParmName>
   <ParmDesc>The ATT data packet to be transmitted</ParmDesc>
   <ParmSize>1</ParmSize>
   <ParmType>tArray</ParmType>
   <ParmLastValue>00</ParmLastValue>
   <ParmDefaultValue>00</ParmDefaultValue>
   <ParmTypeSpecifics>
    <Depend>
     <DependName>PacketLength</DependName>
     <DependMask>0xFFFF</DependMask>
     <DependShift>0x0000</DependShift>
    </Depend>
   </ParmTypeSpecifics>
  </Parm>
 </CmdParms>
</Cmd>
```

This translates to:

Tag	Value	Description
<CmdName	L2CAP-SendAttData.Request	Name of the command/event
<CmdDesc	Sends a data packet through ATT Channel	Description of the command/event
<CmdHead	41 03	opGroup and opCode of the command/event in hexadecimal format
<CmdParm	<Parm>	List of the parameters used in the command/event
<ParmNam	DeviceId	Name of the parameter
<ParmDesc	The DeviceId for which the command is intended	Description of the parameter
<ParmSize	1	Parameter size (in bytes)
<ParmType	tInt	Type of the parameter
<ParmLast	00	Last value of the parameter in hexadecimal format
<ParmDefa	00	Default value of the parameter in hexadecimal format
<ParmType	<ParmRange>	Specify range of the parameter
	<Depend>	Specify dependency of the parameter "Packet" of type "tArray" of the variable <DependName> that is the PacketLength

Parent topic: *Bluetooth Low Energy Host Stack FSCI Application Programming*

References

For more information, refer to the NXP website or contact your local Field Application Engineer (FAE).

Parent topic: *Bluetooth Low Energy Host Stack FSCI Application Programming*

Acronyms and Abbreviations

Acronym	Description
ATT	Attribute Protocol
BLE	Bluetooth Low Energy
FSCI	Framework Serial Communication Interface
GAP	Generic Access Profile
GATT	Generic Attribute Profile
GATTDB	Generic Attribute Profile Database
HCI	Host Controller Interface
L2CAP	Logical Link Control and Adaptation
L2CAP CB	Logical Link Control and Adaptation Credit Based
SM	Security Manager

Parent topic: *Bluetooth Low Energy Host Stack FSCI Application Programming*

Bluetooth Low Energy Connection Handover Introduction

Connection Handover is an NXP proprietary feature that enables a Bluetooth Low Energy connection to be seamlessly transferred from one device to another while the connected peer remains unaware.

This document explains how to integrate the NXP Bluetooth Low Energy Connection Handover feature in an application and provides detailed explanation of the most commonly used APIs and code examples.

Prerequisites

The Bluetooth Low Energy Connection Handover feature can be integrated in an application that fulfills the following requirements:

- Supported platforms: KW45, KW47 and KW43.
- Bluetooth Low Energy application that is using the lib_ble_OPT_host_cm33_iar.a host library.
- Communication channel between the Source and Target devices involved in the Connection Handover procedure.

Integration and APIs

Integration

All the APIs referenced in this section are available in the the full-featured libraries.

- *lib_ble_OPT_host_cm33_iar.a* (for IAR projects)
- *lib_ble_OPT_host_cm33_gcc.a* (for MCUX projects).

Parent topic:*Bluetooth LE Connection Handover Integration and APIs*

Connection Handover API availability

Gap_HandoverGetDataSize

```
bleResult_t Gap_HandoverGetDataSize(deviceId_t deviceId, uint32_t *pDataSize);
```

Description: Returns the size in octets of the Handover Data for the given peer device id.

Parameters:

- deviceId: Peer device identifier.
- pDataSize: Pointer to the size in octets of the Handover Data.

Returns: gBleSuccess_c or error.

Gap_HandoverGetData

```
bleResult_t Gap_HandoverGetData(deviceId_t deviceId, uint8_t *pData);
```

Description: Copies the data used by the Handover feature for the given peer device id.

Parameters:

- deviceId: Peer device identifier.
- pData: Pointer to memory location where the Handover data is to be copied.

Returns: gBleSuccess_c or error.

Gap_HandoverSetData

```
bleResult_t Gap_HandoverSetData(uint8_t *pData);
```

Description: Sets the data used by the Handover feature.

Parameters:

- pData: Pointer to memory location where the Handover data is found.

Returns: gBleSuccess_c or error.

Gap_HandoverSetLlPendingData

```
bleResult_t Gap_HandoverSetLlPendingData(uint16_t connectionHandle, uint8_t *pTxData);
```

Description: Sets the LL data pending on the target device to be transmitted after handover connect.

Parameters:

- connectionHandle: Handover connection identifier.
- pTxData: Pointer to memory location where the pending data is found.

Returns: gBleSuccess_c or error.

Gap_HandoverFreeData

```
bleResult_t Gap_HandoverFreeData(void);
```

Description: Frees reference to handover data provided through Gap_HandoverSetData().

Returns: gBleSuccess_c or gBleInvalidState_c if handover data reference is not set.

Gap_HandoverGetTime

```
bleResult_t Gap_HandoverGetTime(void);
```

Description: Gets timing information from Link Layer to be used in Handover process.

Returns: gBleSuccess_c or error.

Gap_HandoverSuspendTransmit

```
bleResult_t Gap_HandoverSuspendTransmit(deviceId_t deviceId, bleHandoverSuspendTransmitMode_t␣
↪mode, uint16_t eventCounter, uint8_t noOfConnIntervals);
```

Description: Suspends TX for the given connection.

Parameters:

- deviceId: Peer device identifier.
- mode: Mode.
- eventCounter: Event counter.
- noOfConnIntervals: Number of connection intervals during which TX is suspended. Use 0 for manual resume.

Returns: gBleSuccess_c or error.

Gap_HandoverResumeTransmit

```
bleResult_t Gap_HandoverResumeTransmit(deviceId_t deviceId);
```

Description: Resumes TX for the given connection.

Parameters:

- deviceId: Peer device identifier.

Returns: gBleSuccess_c or error.

Gap_HandoverAnchorNotification

```
bleResult_t Gap_HandoverAnchorNotification(deviceId_t deviceId, bleHandoverAnchorNotificationEnable_t␣
↪enable, uint8_t noOfReports);
```

Description: Enables or disables anchor notifications for the given peer.

Parameters:

- deviceId: Peer device identifier.
- enable: Enables or disables notifications.
- noOfReports: Number of reports to receive (0 for manual stop).

Returns: gBleSuccess_c or error.

Gap_HandoverAnchorSearchStart

```
bleResult_t Gap_HandoverAnchorSearchStart(gapHandoverAnchorSearchStartParams_t *pSearchParams);
```

Description: Starts anchor search with the given parameters.

Parameters:

- pSearchParams: Pointer to structure containing anchor search parameters. The structure contains:
  - startTime625: Slot of the anchor point timing of the connection event.
  - startTimeOffset: Slot offset of the anchor point timing of the connection event.
  - lastRxInstant: Last successful access address reception instant (unit 625 us).
  - accessAddress: Access address (4 bytes).
  - crcInit: CRC initialization value (3 bytes).
  - channelMap: Channel map (5 bytes, FF FF FF FF 1F if all channels are used).
  - connInterval: Connection interval (unit 1.25ms).
  - latency: Latency (unit connection interval).
  - supervisionTimeout: Supervision timeout (unit 10ms).
  - eventCounter: Current connection event counter.
  - centralSca: Sleep clock accuracy (0 to 7).
  - role: Role (0 for central and 1 for peripheral).
  - centralPhy: TX/RX PHY.
  - seqNum: Sequence number; bits 1-0: latest received SN and NESN; bits 5-4: latest transmitted SN and NESN.
  - hopAlgo2: Hop algorithm (0/1 for hop algorithm #1 or #2).
  - unmappedChannelIndex: Unmapped channel index (used only for hop algorithm #1).
  - hopIncrement: Hop increment.
  - ucNbReports: Number of connection intervals to monitor (0: continuous, 1-255: automatic stop).
  - uiEventCounterAdvance: Event counter delay to perform the search (0 to start ASAP).
  - timeout: Timeout in connection intervals (0 if no timeout).
  - timingDiffSlot: Slot difference of the LL timing.
  - timingDiffOffset: Slot offset difference of the LL timing.
  - mode: Search mode setting.

- reportLatency: gHandoverAnchorMonitorEvent_c report latency. Applies only after a valid RSSI detection (0 = send all HCI events, use gHandoverAnchorMonitorReportLatencySendAll_c; 1-254 = skip that many events with valid RSSI; 255 = drop all HCI events with valid RSSI, use gHandoverAnchorMonitorReportLatencyDropAll_c).
- connEvtRxLatency: Connection events RX operation latency. Applies only after a valid RSSI detection (0 = monitor all connection events for the radio RX; 1-255 = skip that many connection events during the radio RX).

Returns: gBleSuccess_c or error.

Gap_HandoverAnchorSearchStop

```
bleResult_t Gap_HandoverAnchorSearchStop(uint16_t connHandle);
```

Description: Stops anchor search for the given connection handle.

Parameters:

- connHandle: Connection handle for which to stop the anchor search.

Returns: gBleSuccess_c or error.

Gap_HandoverTimeSyncTransmit

```
bleResult_t Gap_HandoverTimeSyncTransmit(gapHandoverTimeSyncTransmitParams_t *pTransmitParams);
```

Description: Starts time sync advertising with given parameters.

Parameters:

- pTransmitParams: Pointer to structure containing time sync transmit parameters. The structure contains:
  - enable: Enables or disables the transmit operation (0 = disable, 1 = enable).
  - advChannel: BLE channel to use for advertising (0 to 39).
  - deviceAddress: Packet identifier, can be the BD address or any value (6 bytes).
  - phys: PHY setting (0 = 1M, 1 = 2M, 2 = LR S8, 3 = LR S2).
  - txPowerLevel: TX Power level (0 to 31).
  - txIntervalSlots625: Packet interval in slots (4 to 255).

Returns: gBleSuccess_c or error.

Gap_HandoverTimeSyncReceive

```
bleResult_t Gap_HandoverTimeSyncReceive(gapHandoverTimeSyncReceiveParams_t *pReceiveParams);
```

Description: Starts time sync scanning with the given parameters.

Parameters:

- pReceiveParams: Pointer to structure containing time sync receive parameters. The structure contains:
  - enable: Enables or disables the receive operation (0 = disable, 1 = enable).
  - scanChannel: BLE channel to use for scanning (0 to 39).
  - deviceAddress: Packet identifier, can be the BD address or any value (6 bytes).
  - phys: PHY setting (0 = 1M, 1 = 2M, 2 = LR S8, 3 = LR S2).
  - stopWhenFound: One-shot setting (0 = continue scanning, 1 = stop when packet is found).

Returns: gBleSuccess_c or error.

Gap_HandoverGetConnParams

```
bleResult_t Gap_HandoverGetConnParams(uint16_t connHandle, gapConnectionCallback_t␣
↪connectionCallback, uint8_t anchorNotification);
```

Description: Connects to a peer device initialized through the Connection Handover procedure.

Parameters:

- connHandle: Handover Connection handle.
- connectionCallback: Pointer to the connection event handler function.
- anchorNotification: Anchor search setting after handover, 0 - stop anchor notification, 1 - keep anchor notification.

Returns: gBleSuccess_c or error.

Gap_HandoverDisconnect

```
bleResult_t Gap_HandoverDisconnect(deviceId_t deviceId);
```

Description: Disconnects a peer device for which the connection was handed over to another device.

Parameters:

- deviceId: Peer device identifier.

Returns: gBleSuccess_c or error.

Gap_HandoverInit

```
bleResult_t Gap_HandoverInit(void);
```

Description: Initializes the connection handover feature. Must be called before Ble_Initialize().

Returns: gBleInvalidState_c if Ble_Initialize() was already called, gBleSuccess_c otherwise.

Gap_HandoverSetSkd

```
bleResult_t Gap_HandoverSetSkd(uint8_t nvmIndex, uint8_t *pSkd);
```

Description: Sets the SKD of the connection that is to be handed over. Used only if gAppSecureMode_d is enabled in the application. Requires Handover Data to be set.

Parameters:

- nvmIndex: Bonding Data NVM index.
- pSkd: Pointer to 16 octets SKD.

Returns: gBleInvalidState_c if Ble_Initialize() was already called, gBleSuccess_c otherwise.

Gap_HandoverGetCsLlContext

```
bleResult_t Gap_HandoverGetCsLlContext(deviceId_t deviceId);
```

Description: Copies the LL CS-related context data for the given peer device id.

Parameters:

- deviceId: Peer device identifier.

Returns: gBleSuccess_c or error.

Gap_HandoverSetCsLlContext

```
bleResult_t Gap_HandoverSetCsLlContext(deviceId_t deviceId, uint16_t mask, uint8_t contextLength,
↪uint8_t *pContextData);
```

Description: Sets the CS-related context data into the LL, for the given peer device id.

Parameters:

- deviceId: Peer device identifier.
- mask: Context data bitmask (used by the LL).
- contextLength: Length of the context data.
- pContextData: Pointer to context data.

Returns: gBleSuccess_c or error.

Gap_HandoverUpdateConnParams

```
bleResult_t Gap_HandoverUpdateConnParams(gapHandoverUpdateConnParams_t *pConnParams);
```

Description: Updates the channel map and/or PHY used in the current anchor monitoring process.

Parameters:

- pConnParams: Pointer to connection parameters to be updated.

Returns: gBleSuccess_c or error.

Gap_HandoverApplyConnectionUpdateProcedure

```
bleResult_t Gap_
↪HandoverApplyConnectionUpdateProcedure(gapHandoverApplyConnectionUpdateProcedure_t
↪*pConnParams);
```

Description: Applies the new connection parameters in case of Connection Update Procedure on the active controller.

Parameters:

- pConnParams: Pointer to connection parameters to be applied.

Returns: gBleSuccess_c or error.

Connection Handover Events

The following events can be received through the GAP generic callback during the handover operations:

gHandoverGetComplete_c (0x31)

Description: Event generated when handover data get operation is complete.

Event Data: handoverGetData_t containing:

- status: Command status.
- pData: Pointer to connection handover data.

gHandoverSetComplete_c (0x32)

Description: Event generated when handover data set operation is complete.

Event Data: handoverSetData_t containing:

- status: Command status.
- pData: Pointer to connection handover data.

gHandoverGetCsLlContextComplete_c (0x33)

Description: Event generated when handover CS LL context data get operation is complete.

Event Data: handoverGetCsLlContext_t containing:

- status: Command status.
- responseMask: LL context bitmap indicating the Channel Sounding context.
- llContextLength: Context data length.
- llContext: LL Context data.

gHandoverSetCsLlContextComplete_c (0x34)

Description: Event generated when handover CS LL context data set operation is complete.

Event Data: Status of the operation.

gHandoverGetTime_c (0x35)

Description: Event generated when Handover Get Time command is complete.

Event Data: handoverGetTime_t containing:

- status: Command status.
- slot: LL timing slot counter (unit 625us).
- us_offset: LL timing slot offset (0 to 624, unit us).

gHandoverSuspendTransmitComplete_c (0x36)

Description: Event generated when Handover Suspend Transmit command is complete.

Event Data: handoverSuspendTransmitCompleteEvent_t containing:

- connectionHandle: Connection identifier.
- noOfPendingAclPackets: Number of pending ACL packets.
- sizeOfPendingAclPackets: Pending ACL packet data size.
- sizeOfDataTxInOldestPacket: Size of data transmitted and acked in the oldest ACL packet.
- sizeOfDataNAckInOldestPacket: Size of data transmitted but not acked in the oldest ACL packet.

gHandoverResumeTransmitComplete_c (0x37)

Description: Event generated when Handover Resume Transmit command is complete.

Event Data: handoverResumeTransmitCompleteEvent_t containing:

- connectionHandle: Connection identifier.

gHandoverAnchorNotificationStateChanged_c (0x38)

Description: Event generated when Handover Anchor Notification command is complete.

Event Data: handoverAnchorNotificationStateChanged_t containing:

- connectionHandle: Connection identifier.

gHandoverAnchorSearchStarted_c (0x39)

Description: Event generated when Handover Anchor Search Start command is complete.

Event Data: handoverAnchorSearchStart_t containing:

- status: Command status.
- connectionHandle: Connection identifier.

gHandoverAnchorSearchStopped_c (0x3A)

Description: Event generated when Handover Anchor Search Stop command is complete.

Event Data: handoverAnchorSearchStop_t containing:

- status: Command status.
- connectionHandle: Connection identifier.

gHandoverTimeSyncTransmitStateChanged_c (0x3B)

Description: Event generated when Handover Time Sync Transmit command is complete.

Event Data: Status of the operation.

gHandoverTimeSyncReceiveComplete_c (0x3C)

Description: Event generated when Handover Time Sync Receive command is complete.

Event Data: Status of the operation.

gHandoverAnchorMonitorEvent_c (0x3D)

Description: Event received from Controller - Handover Anchor Monitor.

Event Data: handoverAnchorMonitorEvent_t containing:

- connectionHandle: Connection identifier.
- connEvent: Current connection event counter.
- rssiRemote: RSSI of the packet from the remote device (+127 if not available).
- lqiRemote: LQI (Link Quality Indicator) of the packet from the remote device.
- statusRemote: Status of the packet from the remote device.
- rssiActive: RSSI of the packet from the active device (+127 if not available).
- lqiActive: LQI (Link Quality Indicator) of the packet from the active device.
- statusActive: Status of the packet from the active device.
- anchorClock625Us: Slot of the anchor point timing of the connection event.
- anchorDelay: Slot offset of the anchor point timing of the connection event.
- chIdx: BLE channel index.
- ucNbReports: Number of remaining reports to receive.

gHandoverTimeSyncEvent_c (0x3E)

Description: Event received from Controller - Handover Time Sync.

Event Data: handoverTimeSyncEvent_t containing:

- txClkSlot: Transmitter packet start time in slot (625 us).
- txUs: Transmitter packet start time offset inside the slot (0 to 624 us).
- rxClkSlot: Receiver packet start time in slot (625 us).
- rxUs: Receiver packet start time offset inside the slot (0 to 624 us).
- rssi: RSSI value.

gHandoverConnParamUpdateEvent_c (0x3F)

Description: Event received from Controller - Handover Connection Parameters Update.

Event Data: handoverConnParamUpdateEvent_t containing:

- status: Status indicating the event trigger source.
- connectionHandle: Connection identifier.
- ulTxAccCode: Access address.
- aCrcInitVal: CRC initialization value (3 bytes).
- uiConnInterval: Connection interval (unit 1.25ms).
- uiSuperTO: Supervision timeout (unit 10ms).
- uiConnLatency: Latency (unit connection interval).
- aChMapBm: Channel map (5 bytes).
- ucChannelSelection: Hop algorithm (0/1 for algorithm #1 or #2).
- ucHop: Hop increment.
- ucUnMapChIdx: Unmapped channel index.
- ucCentralSca: Sleep clock accuracy.
- ucRole: Role (0 for central and 1 for peripheral).
- aucRemoteMasRxPHY: TX/RX PHY.
- seqNum: Sequence number.
- uiConnEvent: Current connection event counter.
- ulAnchorClk: Slot of the anchor point timing.
- uiAnchorDelay: Slot offset of the anchor point timing.
- ulRxInstant: Last successful access address reception instant.

gHandoverFreeComplete_c (0x46)

Description: Event generated when handover data free operation is complete.

Event Data: Status of the operation.

gHandoverUpdateConnParamsComplete_c (0x47)

Description: Event generated when Handover Update Connection Parameters command is complete.

Event Data: handoverUpdateConnParams_t containing:

- status: Command status.
- connectionHandle: Connection identifier.

gHandoverLlPendingData_c (0x4A)

Description: Event indicating ACL Data pending to be transmitted by the LL.

Event Data: handoverLlPendingDataIndication_t containing:

- dataSize: Pending HCI ACL data packet size.
- pData: Pointer to message containing the HCI ACL data packet (should be freed by the application).

gHandoverAnchorMonitorPacketEvent_c (0x44)

Description: Event received from Controller - Handover Anchor Monitor Packet.

Event Data: handoverAnchorMonitorPacketEvent_t containing:

- packetCounter: Packet counter, incremented for each event.
- connectionHandle: Connection identifier.
- statusPacket: Status of the packet.
  - b0: CRC status - valid (1) or invalid (0).
  - b1: packet transmitter - central (1) or peripheral (0).
- phy: PHY (0/1/2/3 for 1M/2M/LR S8/LR S2).
- chIdx: BLE channel index.
- rssiPacket: RSSI of the packet (+127 if not available).
- lqiPacket: LQI (Link Quality Indicator) of the packet.
- connEvent: Current connection event counter.
- anchorClock625Us: Slot value of packet start time (in 625us unit).
- anchorDelay: Slot offset value of packet start time (in 1us unit).
- ucNbConnIntervals: Number of remaining connection intervals to monitor.
- pduSize: PDU length.
- pPdu: Pointer to PDU data (must be freed by the application).

gHandoverAnchorMonitorPacketContinueEvent_c (0x45)

Description: Event received from Controller - Handover Anchor Monitor Packet Continue.

Event Data: handoverAnchorMonitorPacketContinueEvent_t containing:

- packetCounter: Packet counter, incremented for each event.
- connectionHandle: Connection identifier.
- pduSize: PDU length.
- pPdu: Pointer to PDU data (must be freed by the application).

gHandoverConnectionUpdateProcedureEvent_c (0x4F)

Description: Event reporting new connection parameters indicated during the Connection Update procedure.

Event Data: handoverConnectionUpdateProcedureEvent_t containing:

- connectionHandle: Connection identifier.
- winSize: Transmit window size value (in 1.25 ms units).
- winOffset: Transmit window offset value (in 1.25 ms units).
- interval: Connection interval value (in 1.25 ms units).
- latency: Connection peripheral latency value.
- timeout: Connection supervision timeout value (in 10 ms units).
- instant: Connection event counter value when new parameters will be applied.
- currentEventCounter: Current connection event counter.

gHandoverApplyConnectionUpdateProcedureComplete_c (0x50)

Description: Event reporting completion of the Connection Update procedure application.

Event Data: handoverApplyConnectionUpdateProcedure_t containing:

- connectionHandle: Connection identifier.

Application common modules

The following application common modules are meant to offer a simplified API for generic Connection Handover procedures and allow for quick integration into existing applications.

Handover application common module

Introduction

The Connection Handover application common module allows for quick integration of the Connection Handover feature in a Bluetooth LE application.

Integration

All the APIs referenced in this section are available in the Connection Handover application common module.

Connection handover requires a communication interface with the other devices involved in the connection handover process.

Perform the following steps to integrate Connection Handover in an application:

1. Add app_handover.c and app_handover.h from middleware\wireless\bluetooth\application\common\auto\ in the application project.
2. Initialize the Connection Handover application common module by calling AppHandover_Init().
3. Call AppHandover_ProcessA2ACommand() to handle connection handover commands received.
4. Call AppHandover_GenericCallback() in the application generic events callback to handle the following events:
   - gHandoverAnchorSearchStarted_c
   - gHandoverAnchorSearchStopped_c
   - gHandoverAnchorMonitorEvent_c
   - gHandoverSuspendTransmitComplete_c
   - gHandoverGetComplete_c
   - gGetConnParamsComplete_c

- gHandoverTimeSyncEvent_c
- gHandoverTimeSyncTransmitStateChanged_c
- gHandoverTimeSyncReceiveComplete_c
- gLlSkdReportEvent_c
- gInternalError_c
- gHandoverAnchorMonitorPacketEvent_c
- gHandoverAnchorMonitorPacketContinueEvent_c
- gHandoverFreeComplete_c
- gHandoverLlPendingData_c
- gHandoverConnectionUpdateProcedureEvent_c
- gHandoverApplyConnectionUpdateProcedureComplete_c

5. Call AppHandover_ConnectionCallback in the application connection events callback to handle the following events:
   - gConnEvtHandoverConnected_c
   - gHandoverDisconnected_c
6. To trigger connection handover:
   - **Standard flow** (Time Synchronization required): call AppHandover_SetPeerDevice() followed by AppHandover_StartTimeSync(TRUE).
   - **Fast flow** (Anchor/Packet Monitoring already active on Target): call AppHandover_StartConnectionHandover(deviceId). This skips Time Synchronization and Anchor Search entirely.

Connection Handover application common module APIs

AppHandover_Init

```
bleResult_t AppHandover_Init(appHandoverEventCb_t pfAppEventCb, gapConnectionCallback_t␣
↪pfConnectionCallback, appHandoverA2AInterfaceCb_t pfAppA2AInterfaceCb);
```

Description: Initializes the handover application common module.

Parameters:

- pfAppEventCb: Application callback for handover events.
- pfConnectionCallback: Connection callback for connection events.
- pfAppA2AInterfaceCb: Communication callback for handover data.

Returns: bleResult_t

AppHandover_TimeSyncTransmit

```
bleResult_t AppHandover_TimeSyncTransmit(bleHandoverTimeSyncEnable_t enable);
```

Description: Enables/disables transmission of time synchronization information.

Parameters:

- enable: Enables or disables the transmission of time synchronization parameters.

Returns: bleResult_t

AppHandover_TimeSyncReceive

```
bleResult_t AppHandover_TimeSyncReceive(bleHandoverTimeSyncEnable_t enable);
```

Description: Enables/disables reception of time synchronization information.

Parameters:

- enable: Enables or disables the reception of time synchronization parameters.

Returns: bleResult_t

AppHandover_StartTimeSync

```
void AppHandover_StartTimeSync(bool_t bTimeSyncForHandover);
```

Description: Triggers handover time synchronization.

Parameters:

- bTimeSyncForHandover: TRUE if handover is following, FALSE if RSSI sniffing is following.

Returns: None

AppHandover_TimeSyncTransmitSetParams

```
void AppHandover_TimeSyncTransmitSetParams(gapHandoverTimeSyncTransmitParams_t *pParams);
```

Description: Sets parameters for the transmission of time synchronization information.

Parameters:

- pParams: Tx params.

Returns: None

AppHandover_TimeSyncReceiveSetParams

```
void AppHandover_TimeSyncReceiveSetParams(gapHandoverTimeSyncReceiveParams_t *pParams);
```

Description: Sets parameters for the reception of time synchronization information.

Parameters:

- pParams: Rx params.

Returns: None

AppHandover_StartConnectionHandover

```
bleResult_t AppHandover_StartConnectionHandover(deviceId_t deviceId);
```

Description: Triggers connection handover without requiring time synchronization or anchor search. Anchor monitoring must already be in progress for the specified connection on the Target device.

Parameters:

- deviceId: Peer device identifier of the connection to be handed over.

Returns: bleResult_t

- gBleSuccess_c: Handover initiated successfully.
- gBleInvalidParameter_c: Invalid device identifier.
- gBleInvalidState_c: Handover already in progress.
- gBleUnavailable_c: Anchor monitoring not active for the specified connection.

AppHandover_SetPeerDevice

```
void AppHandover_SetPeerDevice(deviceId_t deviceId);
```

Description: Sets the peer device for connection handover.

Parameters:

- deviceId: Peer device identifier.

Returns: None

AppHandover_ProcessA2ACommand

```
void AppHandover_ProcessA2ACommand(uint8_t cmdId, uint32_t cmdLen, uint8_t *pCmdData);
```

Description: Processes handover commands.

Parameters:

- cmdId: Command identifier.
- cmdLen: Command length.
- pCmdData: Command data.

Returns: None

AppHandover_SetPeerSkd

```
bleResult_t AppHandover_SetPeerSkd(uint8_t *pSkd, uint8_t nvmIndex);
```

Description: Sets the peer LL SKD for connection handover with advanced secure mode.

Parameters:

- nvmIndex: Peer device NVM index.
- pSkd:

A2A application common module

Introduction

The A2A (Anchor-to-Anchor) application common module implements a sample communication interface between devices involved in the connection handover process.

Integration

Perform the following steps to integrate A2A application common module in an application:

1. Add app_a2a_interface.c and app_a2a_interface.h from middleware\wireless\bluetooth\application\common\auto\ in the application project.
2. Initialize the Connection Handover application common module by calling A2A_Init().
3. Use A2A_SendCommand() as the communication callback for the Connection Handover application common module.
4. Use A2A_SendSetBondingDataCommand() in the application to send the bonding data from the connected anchor to the target anchor after pairing a new device is successful.

A2A application common module APIs

A2A_Init

```
bleResult_t A2A_Init(serial_handle_t pSerialHandle, appA2ADataIndicationCb_t pfDataIndCb);
```

Description: Performs initialization of the Anchor to Anchor serial communication.

Parameters:

- pSerialHandle: Serial interface handle to be used for communication.
- pfConnectionCallback: Application callback for received packets.

Returns: bleResult_t

A2A_SendCommand

```
void A2A_SendCommand(uint8_t opGroup, uint8_t opCode, uint8_t *pPayload, uint16_t len);
```

Description: Send packet over the Anchor to Anchor interface.

Parameters:

- opGroup: Group identifier.
- opCode: Command identifier.
- pPayload: Pointer to command data.
- len: Command length.

Returns: None

A2B application common module

Introduction

The A2B (Alice to Bob) application common module takes advantage of the EdgeLock to EdgeLock Key Exchange feature of the EdgeLock Secure Enclave integrated in the KW45, KW47 and KW43 to derive a special internal key (E2E key) that may be used to import/export encrypted key material between two devices. This may be used to allow for secure transfer of the key material between the anchors involved in the Connection Handover process. The LTK and IRK included in the bonding data are transferred as encrypted blobs that may only be decrypted by the EdgeLock Secure Enclave of the destination device.

Integration

Perform the following steps to integrate A2B application common module in an application:

1. Add app_a2b.c and app_a2b.h from middleware\wireless\bluetooth\application\common\auto\ in the application project.
2. Initialize the A2B application common module by calling A2B_Init().
3. One of the device involved in the Connection Handover process must be configured with the gA2BInitiator_d macro set to 1, the other must be configured with the gA2BInitiator_d macro set to 0. The initiator device triggers the EdgeLock-to-EdgeLock (E2E) key derivation and local IRK synchronization. Therefore, it should be started last.
4. After initialization the E2E key will be derived and the gSecLibFunctions.pfSecLib_ExportA2BBlob() and gSecLibFunctions.pfSecLib_ImportA2BBlob() functions can be used to export and import encrypted key material between the two devices.

A2B application common module APIs

A2B_Init

```
bleResult_t A2B_Init(appA2BEventCb_t pfAppEventCb, appA2BA2ACommInterfaceCb_t␣
↪pfAppA2AInterfaceCb);
```

Description: Trigger the A2B feature initialization by generating the public key used for the Edgelock to Edgelock key derivation.

Parameters:

- pfAppEventCb: Application callback functions for A2B events.
- pfAppA2AInterfaceCb: Communication callback for A2B data.

Returns: bleResult_t - Result of the operation.

A2B_ProcessA2ACommand

```
void A2B_ProcessA2ACommand(uint8_t cmdId, uint32_t cmdLen, uint8_t *pCmdData);
```

Description: Process A2B commands received through the A2A interface.

Parameters:

- cmdId: Command identifier.
- cmdLen: Command length.
- pCmdData: Command data.

Returns: None

A2B_FreeE2EKey

```
bleResult_t A2B_FreeE2EKey(void);
```

Description: Free the EdgeLock to EdgeLock key.

Parameters: None

Returns: bleResult_t - Result of the operation.

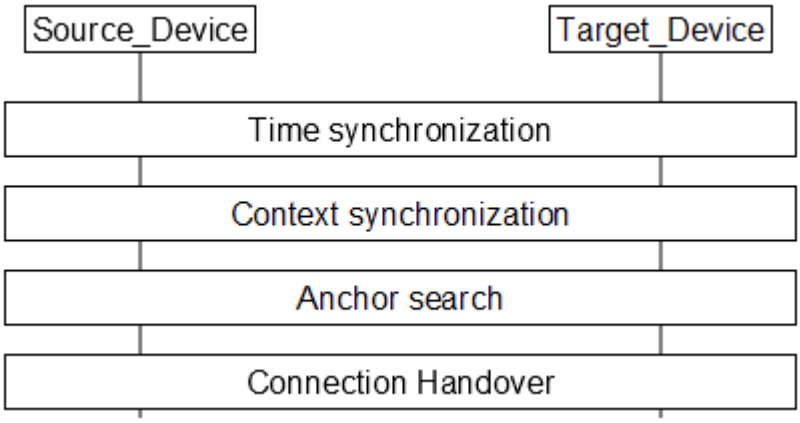
Procedures

This section describes the available Connection Handover procedures. A procedure is executed between a device in a connection, referred as the Source device, and another device involved in the Connection Handover or Anchor/Packet Monitoring process, referred to as the Target device.

Procedures - Top level view

The Connection Handover and Anchor/Packet Monitoring process comprises several procedures as illustrated below.

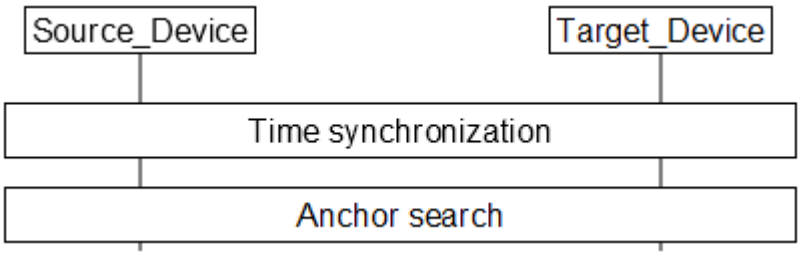
Connection Handover



Connection Handover (Monitoring Active)

When Anchor or Packet Monitoring is already active on the Target device, the Connection Handover can be performed directly without Time Synchronization or Anchor Search. The application triggers this flow by calling AppHandover_StartConnectionHandover(). The process skips directly to Context Synchronization and uses the existing monitoring state for the handover connect.

Anchor/Packet monitoring



Individual procedures

This section describes each individual procedure in detail.

Time synchronization

The Time Synchronization procedure enables precise timing alignment between the Source and Target devices for the Connection Handover and Anchor/Packet monitoring processes. The timing information obtained is further required for the Anchor Search procedure.

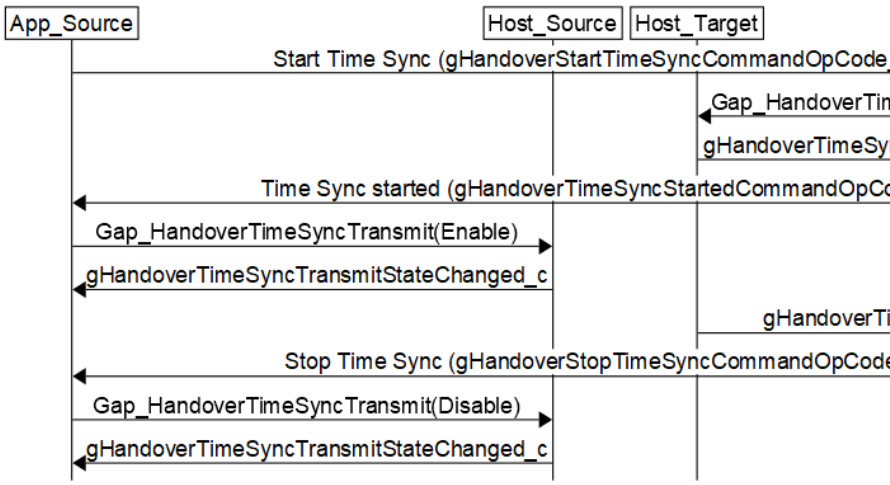
To start the time synchronization procedure, one of the devices must call GAP_HandoverTimeSyncReceive() and the other device must call GAP_HandoverTimeSyncTransmit(). Either device, Source or Target, can initiate the procedure. The Connection Handover application common module initiates the procedure on the Source device by notifying the Target device to call the Gap_HandoverTimeSyncReceive() function.

Time Synchronization steps

Time synchronization steps for two devices where the Target device starts the procedure:

1. Target device application calls GAP_HandoverTimeSyncReceive() with the stopWhenFound parameter set to 'gTimeSyncStopWhenFound_c'.
2. Target device receives the 'gHandoverTimeSyncReceiveComplete_c' event indicating that Source Link Layer is waiting for the time synchronization event.
3. Source device application calls GAP_HandoverTimeSyncTransmit()
4. Source device receives the 'gHandoverTimeSyncTransmitStateChanged_c' event indication that time synchronization transmission is enabled.

5. The Target device receives the 'gHandoverTimeSyncEvent_c' event containing the timing information.
6. The Target device computes the 'Gap_HandoverAnchorSearchStart' 'timingDiffSlot' and 'timingDiffOffset' parameters using the timing information received in the gHandoverTimeSyncEvent_c event as shown below: 'searchParams.timingDiffSlot = mSlotLocal - mSlotRemote' 'searchParams.timingDiffOffset = mOffsetLocal - mOffsetRemote;'



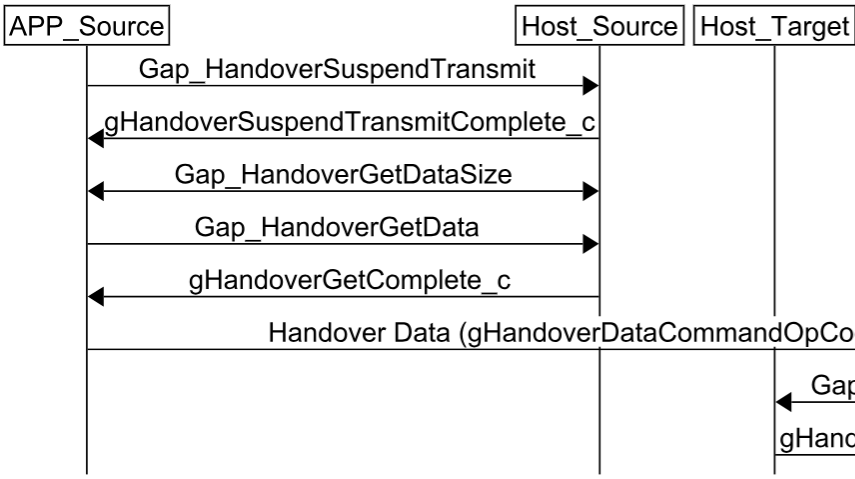
Time Synchronization signaling chart

Context Synchronization

In order to transfer an existing connection from a Source device to a Target device, the Host and Link Layer connection context must be synchronized. The Handover Data includes all the required connection information, which must be retrieved from the Source device and transferred to the Target device. Potential issues include instances where the Host or Link Layer context is updated during the Connection Handover process. To avoid such issues, the Bluetooth LE communication between the Source device and the connected device must be suspended until the Connection Handover is complete or aborted. In case the Connection Handover process is aborted, the original connection can be resumed. This procedure is required only for the Connection Handover process and not for the Anchor/Packet Monitoring process.

Context Synchronization steps

1. Source device calls Gap_HandoverSuspendTransmit() to suspend the Bluetooth LE communication with the connected device.
2. After the gHandoverSuspendTransmitComplete_c event is received, the Handover Data may be retrieved using Gap_HandoverGetData(). The application must provided memory for the Handover Data. To obtain the required data size, use the function Gap_HandoverGetDataSize().
3. Wait for the gHandoverGetComplete_c event to confirm the Handover Data retrieval.
4. Transfer the Handover Data from the Source device to the Target device.
5. On the Target device, call Gap_HandoverSetData() to set the received Handover Data.
6. Wait for the gHandoverSetComplete_c event to confirm the Handover Data has been successfully set.



Context Synchronization signaling chart

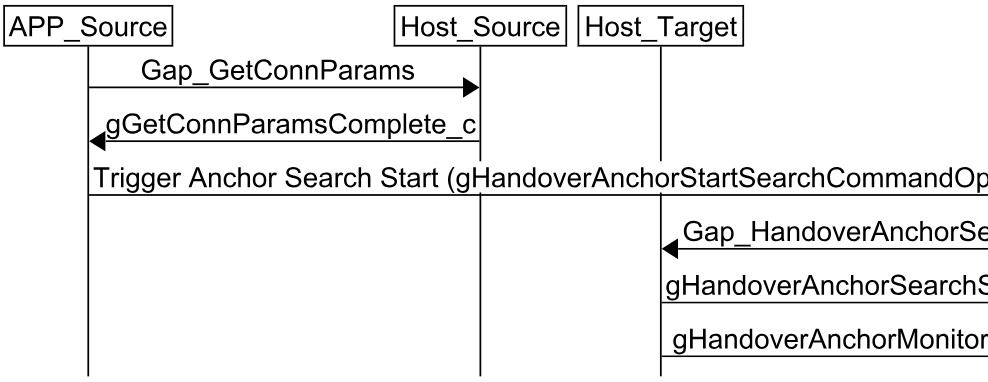
Anchor Search

The Anchor Search procedure takes the connection parameters and timing information obtained in the Time Synchronization procedure to locate the anchor point of the connection between the source device and the connected device.

Anchor Search steps

1. Source device calls Gap_GetConnParams() to retrieve current connection parameters. The connection parameters will be reported through the gGetConnParamsComplete_c event.
2. Send the connection parameters and the timing information obtained from Time Synchronization to the Target device.
3. On the Target device, call Gap_HandoverAnchorSearch() with the received connection parameters and timing information. Set the 'mode' parameter according to the current process:
   - 'gSuspendTxMode_c' for Connection Handover.
   - 'gRssiSniffingMode_c' for Anchor Monitoring.
   - 'gPacketMode_c' for Packet Monitoring.

   The Anchor Search parameters also include two settings that control the rate of the reported events. Both apply only after a valid RSSI detection:
   - 'reportLatency': controls the report rate of the gHandoverAnchorMonitorEvent_c events with valid RSSI. Set to 0 (gHandoverAnchorMonitorReportLatencySendAll_c) to send all HCI events; set to a value between 1 and 254 to skip that many events with valid RSSI; set to 255 (gHandoverAnchorMonitorReportLatencyDropAll_c) to drop all HCI events with valid RSSI.
   - 'connEvtRxLatency': controls the connection events RX operation latency. Set to 0 to monitor all the connection events for the radio RX; set to a value between 1 and 255 to skip that many connection events during the radio RX.
4. Wait for the gHandoverAnchorSearchStarted_c event to confirm the Anchor Search procedure has started.
5. If Anchor Search is successful the requested events will be reported to the application through the gHandoverAnchorMonitorEvent_c or gHandoverAnchorMonitorPacketEvent_c event.



Anchor Search signaling chart

Connection Handover

The Connection Handover procedure consists in the Target device taking over the connection from the Source device. It may be performed in two cases:

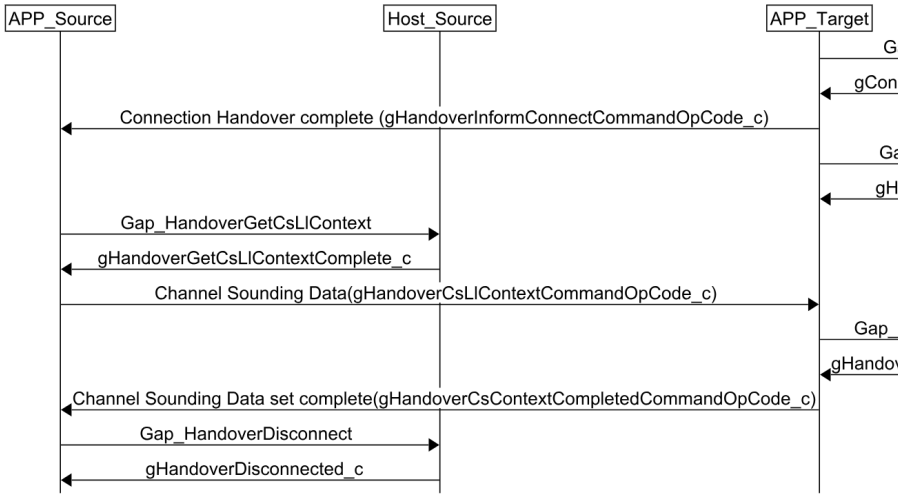
- After a successful Anchor Search procedure, as part of the standard handover flow.
- Directly without Time Synchronization or Anchor Search, when Anchor or Packet Monitoring is already active on the Target device.

Connection Handover steps

1. Target device calls Gap_HandoverConnect() to initiate the connection handover process.
2. Wait for the gConnEvtHandoverConnected_c event on the Target device. Once received, the connection has been successfully transferred.
3. Notify the Source device that the Target device has successfully connected to the Device.
4. If the application does not include Channel Sounding, call Gap_HandoverDisconnect on the Source device to terminate its connection.
5. The Handover Data is no longer required and may be freed by calling Gap_HandoverFreeData() on the Target device.
6. Application context may be transferred at this point through application specific means. If Channel Sounding is enabled in the application, the Channel Sounding Context must be transferred.

Additional steps for Channel Sounding applications:

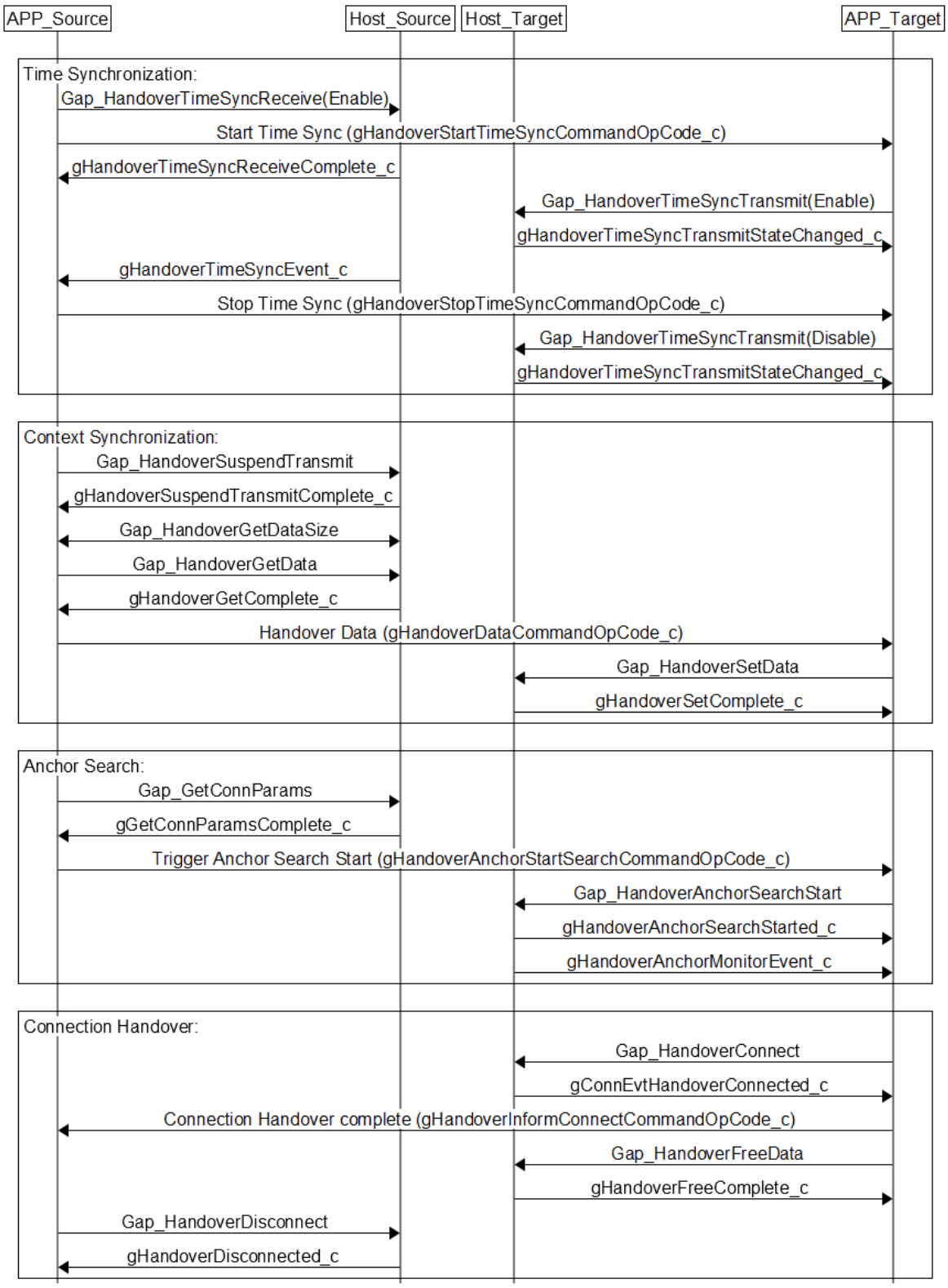
1. After step 3 above, retrieve the Channel Sounding context on the Source device by calling Gap_HandoverGetCsLlContext() and wait for the gHandoverGetCsLlContextComplete event.
2. Send the Channel Sounding context to the Target device.
3. Set the context in the Target device by calling Gap_HandoverSetCsLlContext() and wait for the gHandoverSetCsLlContextComplete_c event.
4. Notify the Source device that the Channel Sounding context has been successfully set.
5. Call Gap_HandoverDisconnect() on the Source device to terminate its connection. When complete, the gHandoverDisconnected_c event will be sent.



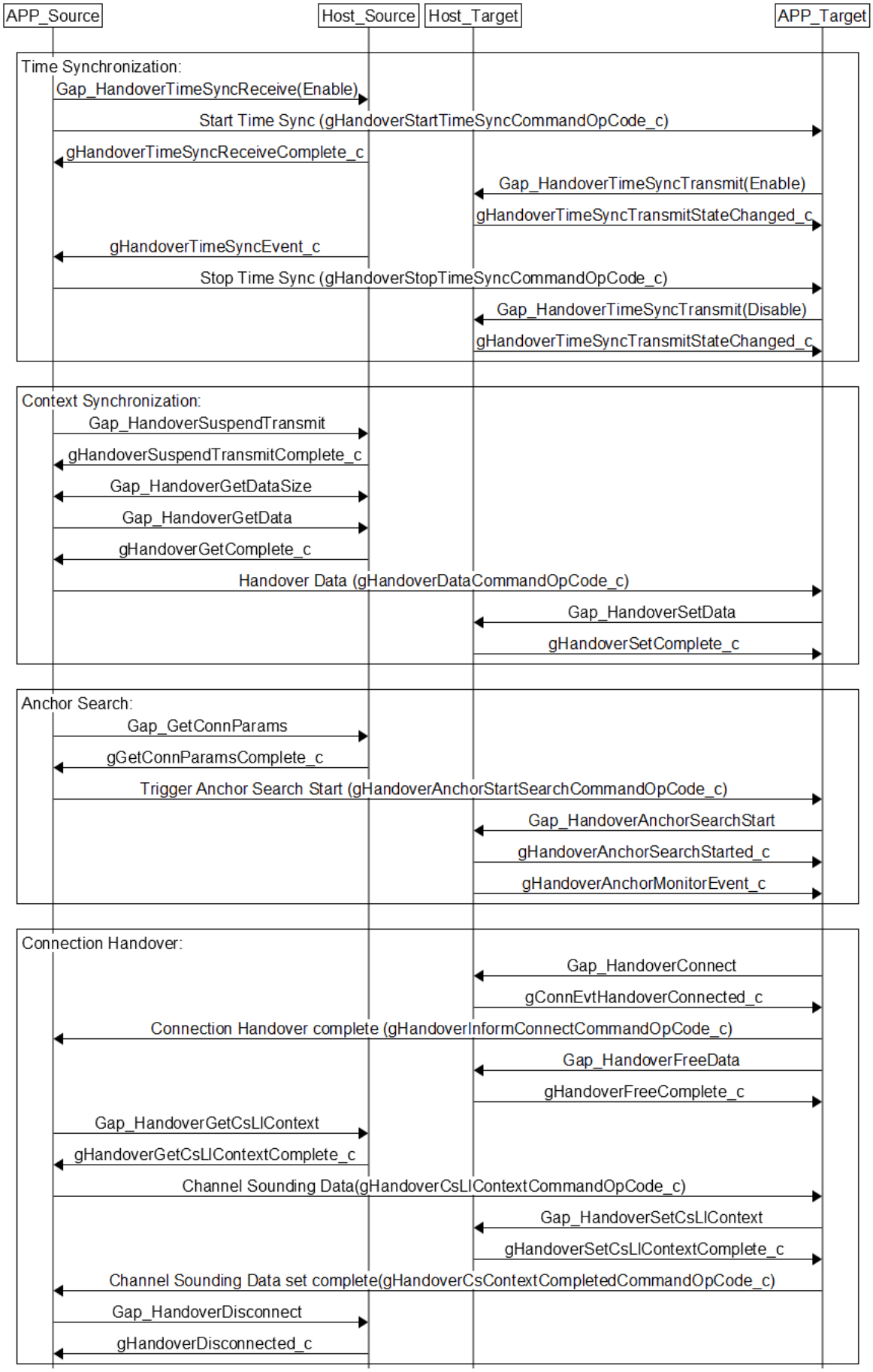
Connection Handover signaling chart

Procedures - Detail view

Connection Handover



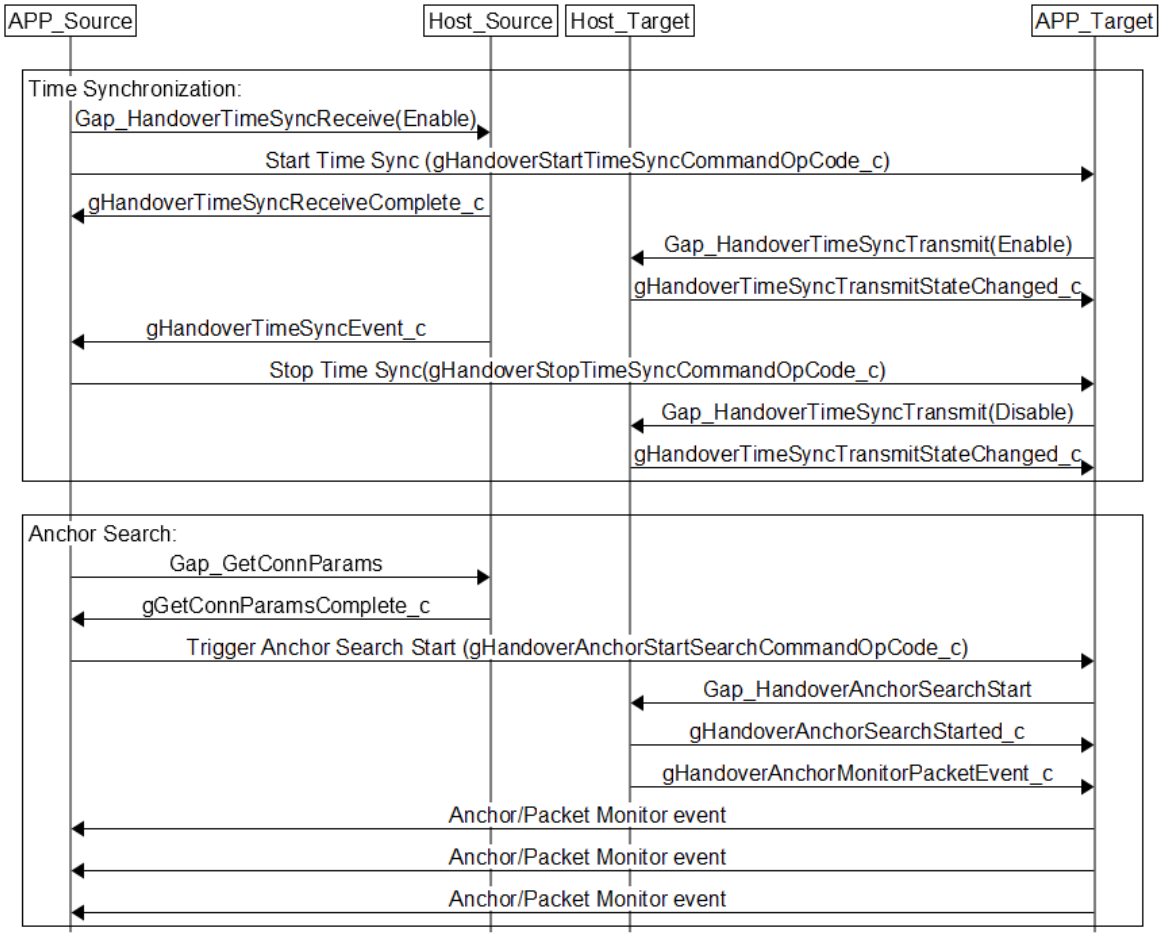
Connection Handover with Channel Sounding


APP_Source
Host_Source
Host_Target
APP_Target
Time Synchronization:
Gap_HandoverTimeSyncReceive(Enable)
Start Time Sync (gHandoverStartTimeSyncCommandOpCode_c)
gHandoverTimeSyncReceiveComplete_c
Gap_HandoverTimeSyncTransmit(Enable)
gHandoverTimeSyncTransmitStateChanged_c
gHandoverTimeSyncEvent_c
Stop Time Sync (gHandoverStopTimeSyncCommandOpCode_c)
Gap_HandoverTimeSyncTransmit(Disable)
gHandoverTimeSyncTransmitStateChanged_c
Context Synchronization:
Gap_HandoverSuspendTransmit
gHandoverSuspendTransmitComplete_c
Gap_HandoverGetDataSize
Gap_HandoverGetData
gHandoverGetComplete_c
Handover Data (gHandoverDataCommandOpCode_c)
Gap_HandoverSetData
gHandoverSetComplete_c
Anchor Search:
Gap_GetConnParams
gGetConnParamsComplete_c
Trigger Anchor Search Start (gHandoverAnchorStartSearchCommandOpCode_c)
Gap_HandoverAnchorSearchStart
gHandoverAnchorSearchStarted_c
gHandoverAnchorMonitorEvent_c
Connection Handover:
Gap_HandoverConnect
gConnEvtHandoverConnected_c
Connection Handover complete (gHandoverInformConnectCommandOpCode_c)
Gap_HandoverFreeData
gHandoverFreeComplete_c
Gap_HandoverGetCsLlContext
gHandoverGetCsLlContextComplete_c
Channel Sounding Data(gHandoverCsLlContextCommandOpCode_c)
Gap_HandoverSetCsLlContext
gHandoverSetCsLlContextComplete_c
Channel Sounding Data set complete(gHandoverCsContextCompletedCommandOpCode_c)
Gap_HandoverDisconnect
gHandoverDisconnected_c

Connection Handover (Monitoring Active)

When Anchor or Packet Monitoring is already active on the Target device, the Connection Handover bypasses Time Synchronization and Anchor Search. The Source device suspends TX, retrieves and sends the Handover Data, and the Target device performs the handover connect on the next monitor event. See the Connection Handover section for the detailed step-by-step procedure and signaling chart.

Anchor/Packet Monitoring



Bluetooth Low Energy Host Stack API Reference Manual Groups

GAP

enum gapRole_t

GAP Role of a BLE device

Values:

enumerator gGapCentral_c

Central scans and connects to Peripherals.

enumerator gGapPeripheral_c

Peripheral advertises and connects to Centrals.

enumerator gGapObserver_c
Observer only scans and makes no connections.

enumerator gGapBroadcaster_c
Broadcaster only advertises and makes no connections.

enum gapKeypressNotification_tag
Values:

enumerator gKnPasskeyEntryStarted_c
Start of the Passkey Entry.

enumerator gKnPasskeyDigitStarted_c
Digit entered.

enumerator gKnPasskeyDigitErased_c
Digit erased.

enumerator gKnPasskeyCleared_c
Passkey cleared.

enumerator gKnPasskeyEntryCompleted_c
Passkey Entry completed.

enum gapScanMode_t
Scan Mode options; used as parameter for Gap_SetScanMode.

Values:

enumerator gDefaultScan_c
Reports all scanned devices to the application.

enumerator gLimitedDiscovery_c
Reports only devices in Limited Discoverable Mode, i.e., containing the Flags AD with the LE Limited Discoverable Flag set.

enumerator gGeneralDiscovery_c
Reports only devices in General Discoverable Mode, i.e., containing the Flags AD with the LE General Discoverable Flag set.

enumerator gAutoConnect_c
Automatically connects with devices with known addresses and does not report any scanned device to the application.

enum gapAdvertisingChannelMapFlags_t
Advertising Channel Map flags - setting a bit activates advertising on the respective channel.

Values:

enumerator gAdvChanMapFlag37_c
Bit for channel 37.

enumerator gAdvChanMapFlag38_c

Bit for channel 38.

enumerator gAdvChanMapFlag39_c

Bit for channel 39.

enum gapAdvertisingFilterPolicy_t

Advertising Filter Policy values

Values:

enumerator gProcessAll_c

Default value: accept all connect and scan requests.

enumerator gProcessConnAllScanWL_c

Accept all connect requests, but scan requests only from devices in Filter Accept List.

enumerator gProcessScanAllConnWL_c

Accept all scan requests, but connect requests only from devices in Filter Accept List.

enumerator gProcessFilterAcceptListOnly_c

Accept connect and scan requests only from devices in Filter Accept List.

enum gapPeriodicAdvSyncMode_tag

Periodic Advertisement Sync Transfer parameters

Values:

enumerator gapPeriodicSyncNoSyncMode_c

No attempt is made to synchronize to the periodic advertising and no gHciLePeriodicAdvSyncTransferReceived_c event is sent to the Host

enumerator gapPeriodicSyncNoReports_c

A gHciLePeriodicAdvSyncTransferReceived_c event is sent to the Host. gHciLePeriodicAdvReportEvent_c events will be disabled.

enumerator gapPeriodicSyncReportsEnabled_c

A gHciLePeriodicAdvSyncTransferReceived_c event is sent to the Host. gHciLePeriodicAdvReportEvent_c events will be enabled with duplicate filtering disabled

enumerator gapPeriodicSyncReportsEnabledWithDF_c

A gHciLePeriodicAdvSyncTransferReceived_c event is sent to the Host. gHciLePeriodicAdvReportEvent_c events will be enabled with duplicate filtering enabled

enum gapFilterDuplicates_t

Values:

enumerator gGapDuplicateFilteringDisable_c

Duplicate filtering disabled

enumerator gGapDuplicateFilteringEnable_c

Duplicate filtering enabled

enumerator gGapDuplicateFilteringPeriodicEnable_c

Duplicate filtering enabled, reset for each scan period

enum gapCreateSyncReqFilterPolicy_tag

Values:

enumerator gUseCommandParameters_c

Use the SID, peerAddressType, and peerAddress parameters to determine which advertiser to listen to.

enumerator gUsePeriodicAdvList_c

Use the Periodic Advertiser List to determine which advertiser to listen to

enum gapPrivateKeyType_t

Values:

enumerator gUseGeneratedKey_c

The private key generated by LE_Read_Local_P-256_Public_Key command, used for DH key generation

enumerator gUseDebugKey_c

The private debug key, used for DH key generation

enum gapAdType_t

AD Type values as defined by Bluetooth SIG used when defining *gapAdStructure_t* structures for advertising or scan response data.

Values:

enumerator gAdFlags_c

Defined by the Bluetooth SIG.

enumerator gAdIncomplete16bitServiceList_c

Defined by the Bluetooth SIG.

enumerator gAdComplete16bitServiceList_c

Defined by the Bluetooth SIG.

enumerator gAdIncomplete32bitServiceList_c

Defined by the Bluetooth SIG.

enumerator gAdComplete32bitServiceList_c

Defined by the Bluetooth SIG.

enumerator gAdIncomplete128bitServiceList_c

Defined by the Bluetooth SIG.

enumerator gAdComplete128bitServiceList_c
Defined by the Bluetooth SIG.

enumerator gAdShortenedLocalName_c
Defined by the Bluetooth SIG.

enumerator gAdCompleteLocalName_c
Defined by the Bluetooth SIG.

enumerator gAdTxPowerLevel_c
Defined by the Bluetooth SIG.

enumerator gAdClassOfDevice_c
Defined by the Bluetooth SIG.

enumerator gAdSimplePairingHashC192_c
Defined by the Bluetooth SIG.

enumerator gAdSimplePairingRandomizerR192_c
Defined by the Bluetooth SIG.

enumerator gAdSecurityManagerTkValue_c
Defined by the Bluetooth SIG.

enumerator gAdSecurityManagerOobFlags_c
Defined by the Bluetooth SIG.

enumerator gAdPeripheralConnectionIntervalRange_c
Defined by the Bluetooth SIG.

enumerator gAdServiceSolicitationList16bit_c
Defined by the Bluetooth SIG.

enumerator gAdServiceSolicitationList32bit_c
Defined by the Bluetooth SIG.

enumerator gAdServiceSolicitationList128bit_c
Defined by the Bluetooth SIG.

enumerator gAdServiceData16bit_c
Defined by the Bluetooth SIG.

enumerator gAdServiceData32bit_c
Defined by the Bluetooth SIG.

enumerator gAdServiceData128bit_c
Defined by the Bluetooth SIG.

enumerator gAdPublicTargetAddress_c
Defined by the Bluetooth SIG.

enumerator gAdRandomTargetAddress_c
Defined by the Bluetooth SIG.

enumerator gAdAppearance_c
Defined by the Bluetooth SIG.

enumerator gAdAdvertisingInterval_c
Defined by the Bluetooth SIG.

enumerator gAdLeDeviceAddress_c
Defined by the Bluetooth SIG.

enumerator gAdLeRole_c
Defined by the Bluetooth SIG.

enumerator gAdSimplePairingHashC256_c
Defined by the Bluetooth SIG.

enumerator gAdSimplePairingRandomizerR256_c
Defined by the Bluetooth SIG.

enumerator gAd3dInformationData_c
Defined by the Bluetooth SIG.

enumerator gAdUniformResourceIdentifier_c
Defined by the Bluetooth SIG.

enumerator gAdLeSupportedFeatures_c
Defined by the Bluetooth SIG.

enumerator gAdChannelMapUpdateIndication_c
Defined by the Bluetooth SIG.

enumerator gAdAdvertisingIntervalLong_c
Defined by the Bluetooth SIG.

enumerator gAdEncryptedAdvertisingData_c
Defined by the Bluetooth SIG.

enumerator gAdManufacturerSpecificData_c
Defined by the Bluetooth SIG.

enum gapDecisionInstructionsRelevantField_t
Decision Instruction Relevant Field Type.

Values:

enumerator gDIRF_ResolvableTag_c

enumerator gDIRF_AdvMode_c

enumerator gDIRF_RSSI_c

enumerator gDIRF_PathLoss_c

enumerator gDIRF_AdvAddress_c

enumerator gDIRF_ArbitraryDataOfExactly_1Byte_c

enumerator gDIRF_ArbitraryDataOfExactly_2Bytes_c

enumerator gDIRF_ArbitraryDataOfExactly_3Bytes_c

enumerator gDIRF_ArbitraryDataOfExactly_4Bytes_c

enumerator gDIRF_ArbitraryDataOfExactly_5Bytes_c

enumerator gDIRF_ArbitraryDataOfExactly_6Bytes_c

enumerator gDIRF_ArbitraryDataOfExactly_7Bytes_c

enumerator gDIRF_ArbitraryDataOfExactly_8Bytes_c

enumerator gDIRF_ArbitraryDataOfAtLeast_1Byte_c

enumerator gDIRF_ArbitraryDataOfAtLeast_2Bytes_c

enumerator gDIRF_ArbitraryDataOfAtLeast_3Bytes_c

enumerator gDIRF_ArbitraryDataOfAtLeast_4Bytes_c

enumerator gDIRF_ArbitraryDataOfAtLeast_5Bytes_c

enumerator gDIRF_ArbitraryDataOfAtLeast_6Bytes_c

enumerator gDIRF_ArbitraryDataOfAtLeast_7Bytes_c

enumerator gDIRF_ArbitraryDataOfAtLeast_8Bytes_c

enumerator gDIRF_ArbitraryDataOfAtMost_1Byte_c

enumerator gDIRF_ArbitraryDataOfAtMost_2Bytes_c

enumerator gDIRF_ArbitraryDataOfAtMost_3Bytes_c

enumerator gDIRF_ArbitraryDataOfAtMost_4Bytes_c

enumerator gDIRF_ArbitraryDataOfAtMost_5Bytes_c

enumerator gDIRF_ArbitraryDataOfAtMost_6Bytes_c

enumerator gDIRF_ArbitraryDataOfAtMost_7Bytes_c

enumerator gDIRF_ArbitraryDataOfAtMost_8Bytes_c

enum gapDecisionInstructionsTestPassCriteria_t

Values:

enumerator gDITPC_Never_c

enumerator gDITPC_CheckPasses_c

enumerator gDITPC_CheckFails_c

enumerator gDITPC_RelevantFieldPresent_c

enumerator gDITPC_RelevantFieldNotPresent_c

enumerator gDITPC_CheckPassesOrRelevantFieldNotPresent_c

enumerator gDITPC_CheckFailsOrRelevantFieldNotPresent_c

enumerator gDITPC_Always_c

enum gapDecisionInstructionsTestGroup_t

Values:

enumerator gDITG_SameTestGroup_c

enumerator gDITG_NewTestGroup_c

enum gapDecisionInstructionsAdvAChecks_t

Values:

enumerator gDIAAC_AdvAinFilterAcceptList_c

enumerator gDIAAC_AdvAmatchAddress1_c

enumerator gDIAAC_AdvAmatchAddress1orAddress2_c

enum gapRadioPowerLevelReadType_t

Enumeration used by the Gap_ReadRadioPowerLevel function.

Values:

enumerator gTxPowerCurrentLevelInConnection_c

Reading the instantaneous TX power level in a connection.

enumerator gTxPowerMaximumLevelInConnection_c

Reading the maximum TX power level achieved during a connection.

enumerator gTxPowerLevelForAdvertising_c

Reading the TX power on the advertising channels.

enumerator gRssi_c

Reading the Received Signal Strength Indication in a connection.

enum gapControllerTestCmd_t

Enumeration for Controller Test commands.

Values:

enumerator gControllerTestCmdStartRx_c

Start Receiver Test.

enumerator gControllerTestCmdStartTx_c

Start Transmitter Test.

enumerator gControllerTestCmdEnd_c

End Test.

enum gapControllerTestTxType_tag

Values:

enumerator gControllerTestTxPrbs9_c

PRBS9 sequence '11111111100000111101'...

enumerator gControllerTestTxF0_c

Repeated '11110000'

enumerator gControllerTestTxAA_c

Repeated '10101010'

enumerator gControllerTestTxPrbs15_c

PRBS15 sequence

enumerator gControllerTestTxFF_c

Repeated '11111111'

enumerator gControllerTestTx00_c
Repeated '00000000'

enumerator gControllerTestTx0F_c
Repeated '00001111'

enumerator gControllerTestTx55_c
Repeated '01010101'

enum gapSleepClockAccuracy_t
Enumeration for Sleep Clock Accuracy command.

Values:

enumerator gSwitchToMoreAccurateClock_c

enumerator gSwitchToLessAccurateClock_c

enum gapAdvertisingEventType_t
Advertising event type enumeration, as contained in the *gapAdvertisingEvent_t*.

Values:

enumerator gAdvertisingStateChanged_c
Event received when advertising has been successfully enabled or disabled.

enumerator gAdvertisingCommandFailed_c
Event received when advertising could not be enabled or disabled. Reason contained in gapAdvertisingEvent_t.eventData.failReason.

enumerator gExtAdvertisingStateChanged_c
Event received when extended advertising has been successfully enabled or disabled.

enumerator gAdvertisingSetTerminated_c
Event received when advertising in a given advertising set has stopped.

enumerator gExtScanNotification_c
Event indicates that a SCAN_REQ PDU or an AUX_SCAN_REQ PDU has been received by the extended advertiser.

enumerator gPerAdvSubeventDataRequest_c
Event received when doing PAWR.

enumerator gPerAdvResponse_c
Periodic Advertising Response.

enum gapScanningEventType_t
Scanning event type enumeration, as contained in the *gapScanningEvent_t*.

Values:

enumerator gScanStateChanged_c

Event received when scanning had been successfully enabled or disabled, or a Scan duration time-out has occurred.

enumerator gScanCommandFailed_c

Event received when scanning could not be enabled or disabled. Reason contained in gapScanningEvent_t.eventData.failReason.

enumerator gDeviceScanned_c

Event received when an advertising device has been scanned. Device data contained in gapScanningEvent_t.eventData.scannedDevice.

enumerator gExtDeviceScanned_c

Event received when an advertising device has been scanned. Device data contained in gapScanningEvent_t.eventData.extScannedDevice.

enumerator gPeriodicDeviceScanned_c

Event received when an Periodic advertising device has been scanned. Device data contained in gapScanningEvent_t.eventData.periodicScannedDevice.

enumerator gPeriodicDeviceScannedV2_c

Event received when an Periodic advertising device has been scanned. Device data contained in gapScanningEvent_t.eventData.periodicScannedDeviceV2.

enumerator gPeriodicAdvSyncEstablished_c

Event received when a sync with a periodic advertiser was established.

enumerator gPeriodicAdvSyncLost_c

Event received when a sync with a periodic advertiser have been lost.

enumerator gPeriodicAdvSyncTerminated_c

Event received when a sync with a periodic advertiser have been terminated.

enumerator gConnectionlessIqReportReceived_c

Event received when the Controller has reported IQ information from the CTE of a received advertising packet

enumerator gMonAdvReportEventReceived_c

Event received when Monitored Advertisers is enabled

enum bleMonAdvCondition_t

Values:

enumerator gBleMonAdvConditionRssiLowThreshold_c

Monitored Advertisers RSSI value below the RSSI low threshold.

enumerator gBleMonAdvConditionRssiHighThreshold_c

Monitored Advertisers RSSI value greater than or equal to the RSSI high threshold.

enum gapConnectionEventType_t

Connection event type enumeration, as contained in the *gapConnectionEvent_t*.

Values:

enumerator gConnEvtConnected_c

A connection has been established. Data in gapConnectionEvent_t.eventData.connectedEvent.

enumerator gConnEvtPairingRequest_c

A pairing request has been received from the peer Central. Data in gapConnectionEvent_t.eventData.pairingEvent.

enumerator gConnEvtPeripheralSecurityRequest_c

A Peripheral Security Request has been received from the peer Peripheral. Data in gapConnectionEvent_t.eventData.peripheralSecurityRequestEvent.

enumerator gConnEvtPairingResponse_c

A pairing response has been received from the peer Peripheral. Data in gapConnectionEvent_t.eventData.pairingEvent.

enumerator gConnEvtAuthenticationRejected_c

A link encryption or pairing request has been rejected by the peer device. Data in gapConnectionEvent_t.eventData.authenticationRejectedEvent.

enumerator gConnEvtPasskeyRequest_c

Peer has requested a passkey (maximum 6 digit PIN) for the pairing procedure. Device should respond with Gap_EnterPasskey.

enumerator gConnEvtOobRequest_c

Out-of-Band data must be provided for the pairing procedure. Central or Peripheral should respond with Gap_ProvideOob.

enumerator gConnEvtPasskeyDisplay_c

The pairing procedure requires this Peripheral to display the passkey for the Central's user.

enumerator gConnEvtKeyExchangeRequest_c

The pairing procedure requires the SMP keys to be distributed to the peer. Data in gapConnectionEvent_t.eventData.keyExchangeRequestEvent.

enumerator gConnEvtKeysReceived_c

SMP keys distributed by the peer during pairing have been received. Data in gapConnectionEvent_t.eventData.keysReceivedEvent.

enumerator gConnEvtLongTermKeyRequest_c

The bonded peer Central has requested link encryption and the LTK must be provided. Peripheral should respond with Gap_ProvideLongTermKey. Data in gapConnectionEvent_t.eventData.longTermKeyRequestEvent.

enumerator gConnEvtEncryptionChanged_c

Link's encryption state has changed, e.g., during pairing or after a reconnection with a bonded peer. Data in gapConnectionEvent_t.eventData.encryptionChangedEvent.

enumerator gConnEvtPairingComplete_c

Pairing procedure is complete, either successfully or with failure. Data in gapConnectionEvent_t.eventData.pairingCompleteEvent.

enumerator gConnEvtDisconnected_c

A connection has been terminated. Data in gapConnectionEvent_t.eventData.disconnectedEvent.

enumerator gConnEvtRssiRead_c

RSSI for an active connection has been read. Data in gapConnectionEvent_t.eventData.rssi_dBm.

enumerator gConnEvtTxPowerLevelRead_c

TX power level for an active connection has been read. Data in gapConnectionEvent_t.eventData.txPowerLevel_dBm.

enumerator gConnEvtPowerReadFailure_c

Power reading could not be performed. Data in gapConnectionEvent_t.eventData.failReason.

enumerator gConnEvtParameterUpdateRequest_c

A connection parameter update request has been received. Data in gapConnectionEvent_t.eventData.connectionUpdateRequest.

enumerator gConnEvtParameterUpdateComplete_c

The connection has new parameters. Data in gapConnectionEvent_t.eventData.connectionUpdateComplete.

enumerator gConnEvtLeDataLengthChanged_c

The new TX/RX Data Length parameters. Data in gapConnectionEvent_t.eventData.rssi_dBm.leDataLengthChanged.

enumerator gConnEvtLeScOobDataRequest_c

Event sent to request LE SC OOB Data (r, Cr and Addr) received from a peer.

enumerator gConnEvtLeScDisplayNumericValue_c

Event sent to display and confirm a Numeric Comparison Value when using the LE SC Numeric Comparison pairing method.

enumerator gConnEvtLeScKeypressNotification_c

Remote Keypress Notification received during Passkey Entry Pairing Method.

enumerator gConnEvtChannelMapRead_c

Channel Map was read for a connection. Data is contained in gapConnectionEvent_t.eventData.channelMap

enumerator gConnEvtChannelMapReadFailure_c

Channel Map reading could not be performed. Data in gapConnectionEvent_t.eventData.failReason.

enumerator gConnEvtChanSelectionAlgorithm2_c

LE Channel Selection Algorithm #2 is used on the data channel connection.

enumerator gConnEvtPairingNoLtk_c

No LTK was found for the Central peer. Pairing shall be performed again.

enumerator gConnEvtPairingAlreadyStarted_c

Pairing process was already started

enumerator gConnEvtIqReportReceived_c

Controller has reported IQ information received from a connected peer. Data in gapConnectionEvent_t.eventData.connIqReport.

enumerator gConnEvtCteRequestFailed_c

CTE Request to a connected peer has failed. Data in gapConnectionEvent_t.eventData.cteRequestFailedEvent.

enumerator gConnEvtCteReceiveParamsSetupComplete_c

Connection CTE receive parameters have been successfully set.

enumerator gConnEvtCteTransmitParamsSetupComplete_c

Connection CTE transmit parameters have been successfully set.

enumerator gConnEvtCteReqStateChanged_c

Controller started or stopped initiating the CTE Request procedure.

enumerator gConnEvtCteRspStateChanged_c

Controller enabled or disabled sending CTE Responses for a connection.

enumerator gConnEvtPathLossThreshold_c

Received a Path Loss Threshold event. Data in gapConnectionEvent_t.eventData.pathLossThreshold.

enumerator gConnEvtTransmitPowerReporting_c

Received a Transmit Power report. Data in gapConnectionEvent_t.eventData.transmitPowerReporting.

enumerator gConnEvtEnhancedReadTransmitPowerLevel_c

Local information has been read from Controller. Data in gapConnectionEvent_t.eventData.transmitPowerInfo.

enumerator gConnEvtPathLossReportingParamsSetupComplete_c

Path Loss Reporting parameters have been successfully set.

enumerator gConnEvtPathLossReportingStateChanged_c

Path Loss Reporting has been enabled or disabled.

enumerator gConnEvtTransmitPowerReportingStateChanged_c

Transmit Power Reporting has been enabled or disabled for local and/or remote Controllers.

enumerator gConnEvtEattConnectionRequest_c

Received Enhanced Connection Request for EATT channels.

enumerator gConnEvtEattConnectionComplete_c

Received Enhanced Connection Response for EATT channels.

enumerator gConnEvtEattChannelReconfigureResponse_c

Received Enhanced Channel Reconfigure Response for EATT channels.

enumerator gConnEvtEattBearerStatusNotification_c

Enhanced Bearer status updated

enumerator gConnEvtHandoverConnected_c

A connection has been established through the Handover feature. Data in gapConnectionEvent_t.eventData.handoverConnectedEvent.

enumerator gHandoverDisconnected_c

A connection has been terminated as a result of connection handover.

enumerator gConnEvtLeSetDataLengthFailure_c

The Set Data Length command has failed.

enumerator gConnEvtSmError_c

Security Manager error occured.

enumerator gConnEvtLeSubrateChange_c

The connection has new parameters. Data in gapConnectionEvent_t.eventData.gapSubrateChangeEvent.

enumerator gConnEvtLeSetLocalPeripheralLatencyEnableComplete_c

Le Set Local Peripheral Latency Enable complete. Data in gapConnectionEvent_t.eventData.leSetLocalPeripheralLatencyEnableComplete.

enumerator gConnEvtRemoteFeaturesRead_c

enum gapEattBearerStatus_t

Enumeration for Bearer Status Notification command status.

Values:

enumerator gEnhancedBearerActive_c

enumerator gEnhancedBearerSuspendedNoLocalCredits_c

enumerator gEnhancedBearerNoPeerCredits_c

enumerator gEnhancedBearerDisconnected_c

enumerator gEnhancedBearerStatusEnd_c

enum gapCarSupport_t

Central Address Resolution characteristic state

Values:

enumerator CAR_Unknown

The Central Address Resolution characteristic was not read

enumerator CAR_Unavailable

The device tried to read the Central Address Resolution characteristic, but it's unavailable

enumerator CAR_Unsupported

The device has read the Central Address Resolution characteristic, and the it's value is FALSE

enumerator CAR_Supported

The device has read the Central Address Resolution characteristic, and the it's value is TRUE

enum gapPeriodicAdvListOperation_t

Values:

enumerator gAddDevice_c

enumerator gRemoveDevice_c

enumerator gRemoveAllDevices_c

enum gapAppearance_tag

Appearance characteristic enumeration, also used in advertising.

Values:

enumerator gUnknown_c

enumerator gGenericPhone_c

enumerator gGenericComputer_c

enumerator gGenericWatch_c

enumerator gSportsWatch_c

enumerator gGenericClock_c

enumerator gGenericDisplay_c

enumerator gGenericRemoteControl_c

enumerator gGenericEyeglasses_c

enumerator gGenericTag_c

enumerator gGenericKeyring_c

enumerator gGenericMediaPlayer_c

enumerator gGenericBarcodeScanner_c

enumerator gGenericThermometer_c

enumerator gThermometerEar_c

enumerator gGenericHeartrateSensor_c

enumerator gHeartRateSensorHeartRateBelt_c

enumerator gGenericBloodPressure_c

enumerator gBloodPressureArm_c

enumerator gBloodPressureWrist_c

enumerator gHumanInterfaceDevice_c

enumerator gKeyboard_c

enumerator gMouse_c

enumerator gJoystick_c

enumerator gGamepad_c

enumerator gDigitizerTablet_c

enumerator gCardReader_c

enumerator gDigitalPen_c

enumerator gBarcodeScanner_c

enumerator gGenericGlucoseMeter_c

enumerator gGenericRunningWalkingSensor_c

enumerator gRunningWalkingSensorInShoe_c

enumerator gRunningWalkingSensorOnShoe_c

enumerator gRunningWalkingSensorOnHip_c

enumerator gGenericCycling_c

enumerator gCyclingComputer_c

enumerator gCyclingSpeedSensor_c

enumerator gCyclingCadenceSensor_c

enumerator gCyclingPowerSensor_c

enumerator gCyclingSpeedandCadenceSensor_c

enumerator gGenericPulseOximeter_c

enumerator gFingertip_c

enumerator gWristWorn_c

enumerator gGenericWeightScale_c

enumerator gGenericOutdoorSportsActivity_c

enumerator gLocationDisplayDevice_c

enumerator gLocationandNavigationDisplayDevice_c

enumerator gLocationPod_c

enumerator gLocationAndNavigationPod_c

typedef uint8_t gapIoCapabilities_t

I/O Capabilities as defined by the SMP

typedef uint8_t gapSmpKeyFlags_t

Flags indicating the Keys to be exchanged by the SMP during the key exchange phase of pairing.

typedef uint8_t gapSecurityMode_t

LE Security Mode

typedef uint8_t gapSecurityLevel_t

LE Security Level

typedef uint8_t gapSecurityModeAndLevel_t

Security Mode-and-Level definitions

typedef uint8_t gapKeypressNotification_t

Keypress Notification Types

typedef enum *gapKeypressNotification_tag* gapKeypressNotification_tag

typedef uint8_t gapAuthenticationRejectReason_t

Reason for rejecting the pairing request.

typedef struct *gapExtAdvertisingParameters_tag* gapExtAdvertisingParameters_t

Extended Advertising Parameters; for defaults see gGapDefaultExtAdvertisingParameters_d.

typedef struct *gapPeriodicAdvParameters_tag* gapPeriodicAdvParameters_t

Periodic Advertising Parameters [v2]; for defaults see gGapDefaultPeriodicAdvParameters_d.

typedef enum *gapPeriodicAdvSyncMode_tag* gapPeriodicAdvSyncMode_t

Periodic Advertisement Sync Transfer parameters

typedef struct *gapPeriodicAdvSyncTransfer_tag* gapPeriodicAdvSyncTransfer_t

typedef struct *gapPeriodicAdvSetInfoTransfer_tag* gapPeriodicAdvSetInfoTransfer_t

typedef struct *gapSetPeriodicAdvSyncTransferParams_tag* gapSetPeriodicAdvSyncTransferParams_t

typedef uint8_t gapCreateSyncReqFilterPolicy_t

Create Sync Request Filter Policy values

typedef struct *gapCreateSyncReqOptions_tag* gapCreateSyncReqOptions_t

Create Sync Request Options

typedef struct *gapPeriodicAdvSyncReq_tag* gapPeriodicAdvSyncReq_t

Periodic Advertising Sync Request parameters

typedef uint8_t gapAdTypeFlags_t

Values of the AD Flags advertising data structure.

typedef uint8_t gapDecisionInstructionsAdvMode_t

typedef struct *gapDecisionInstructionsData_tag* gapDecisionInstructionsData_t

typedef *gapAdvertisingData_t* gapScanResponseData_t

Scan Response Data structure : a list of several *gapAdStructure_t* structures.

typedef uint8_t gapControllerTestTxType_t

Enumeration for Controller Transmitter Test payload types.

typedef *bleResult_t* gapDisconnectionReason_t

Disconnection reason alias - reasons are contained in HCI error codes.

typedef void (*gapAdvertisingCallback_t)(*gapAdvertisingEvent_t* *pAdvertisingEvent)

Advertising Callback prototype.

typedef void (*gapScanningCallback_t)(*gapScanningEvent_t* *pScanningEvent)

Scanning Callback prototype.

typedef void (*gapConnectionCallback_t)(*deviceId_t* deviceId, *gapConnectionEvent_t* *pConnectionEvent)

Connection Callback prototype.

typedef enum *gapAppearance_tag* gapAppearance_t

Appearance characteristic enumeration, also used in advertising.

typedef struct *gapHostVersion_tag* gapHostVersion_t

bleResult_t Gap_SetDataRelatedAddressChanges(uint8_t advertisingHandle, uint8_t changeReasons)

Command specifies circumstances when the Controller shall refresh any Resolvable Private Address used by the advertising set identified by the Advertising_Handle parameter.

Parameters

- advertisingHandle – **[in]** Identify an advertising set.
- changeReasons – **[in]** Change reasons. Any combination of gAdvDataChange_c and gScanRspDataChange_c. 0U to disable the feature.

Return values

- gBleSuccess_c –
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range

bleResult_t Gap_LeChasConfig(int8_t initialFilterSeed, uint8_t filterThreshold, uint8_t filterWeight, uint16_t measurementInterval)

Command to configure Channel Assessment settings.

Parameters

- initialFilterSeed – **[in]** Initial filter seed (-108 to -80 dBm).
- filterThreshold – **[in]** Filter threshold (not used).
- filterWeight – **[in]** Filter weight (1 to 255).

- measurementInterval – **[in]** Measurement interval (0 to 0x7FFF).

Return values

bleResult_t – gBleSuccess_c or error.

bleResult_t Gap_LeModulationTest(uint8_t testMode, uint8_t channelNumber, uint8_t options, int8_t txPower)

Command to transmit continuously unmodulated or modulated signal or measure the RSSI of a channel.

Parameters

- testMode – **[in]** Test mode (0=disable, 1=unmodulated, 2=mod 0's, 3=mod 1's, 4=mod PRBS, 255=RSSI measurement).
- channelNumber – **[in]** Channel number (0-39 for BLE RF, 0-127 for generic).
- options – **[in]** Options bitmap (bit0=PHY, bit1=channel type, bit2=power type).
- txPower – **[in]** TX power in dBm or power index (0-62) based on options.

Return values

bleResult_t – gBleSuccess_c or error.

bleResult_t Gap_RegisterDeviceSecurityRequirements(const *gapDeviceSecurityRequirements_t* *pSecurity)

Registers the device security requirements. This function includes a central security for all services and, optionally, additional stronger security settings for services as required by the profile and/or application.

Remark

pSecurity or any other contained security structure pointers that are NULL are ignored, i.e., defaulted to No Security (Security Mode 1 Level 1, No Authorization, Minimum encryption key size). This function executes synchronously.

Parameters

- pSecurity – **[in]** A pointer to the application-allocated *gapDeviceSecurityRequirements_t* structure.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range

bleResult_t Gap_SetAdvertisingParameters(const *gapAdvertisingParameters_t* *pAdvertisingParameters)

Sets up the Advertising Parameters.

Remark

GAP Peripheral-only API function.

Parameters

- pAdvertisingParameters – **[in]** Pointer to *gapAdvertisingParameters_t* structure.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range
- gBleOutOfMemory_c – Cannot allocate memory for the Host task
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version

bleResult_t Gap_SetAdvertisingData(const *gapAdvertisingData_t* *pAdvertisingData, const *gapScanResponseData_t* *pScanResponseData)

Sets up the Advertising and Scan Response Data.

Remark

Any of the parameters may be NULL, in which case they are ignored. Therefore, this function can be used to set any of the parameters individually or both at once. The standard advertising packet payload is 37 bytes. Some of the payload may be occupied by the Advertiser Address which takes up 6 bytes and for some advertising PDU types also by the Initiator Address which takes another 6 bytes. This leaves 25-31 bytes to the application to include advertising structures (Length [1Byte], AD Type [1 Byte], AD Data[Length-1 Bytes])

Remark

GAP Peripheral-only API function.

Parameters

- pAdvertisingData – **[in]** Pointer to *gapAdvertisingData_t* structure or NULL.
- pScanResponseData – **[in]** Pointer to gapScanResponseData_t structure or NULL.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range
- gGapAdvDataTooLong_c – The advertising data length is too big
- gBleOutOfMemory_c – Cannot allocate memory for the Host task
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version

bleResult_t Gap_StartAdvertising(*gapAdvertisingCallback_t* advertisingCallback, *gapConnectionCallback_t* connectionCallback)

Commands the controller to start advertising.

Remark

The advertisingCallback confirms or denies whether the advertising has started. The connectionCallback is only used if a connection gets established during advertising.

Remark

GAP Peripheral-only API function.

Parameters

- advertisingCallback – **[in]** Callback used by the application to receive advertising events. Can be NULL.
- connectionCallback – **[in]** Callback used by the application to receive connection events. Can be NULL.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version
- gBleInvalidState_c – Advertising is already started or pending to be started
- gBleOutOfMemory_c – Cannot allocate memory for the Host task
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version

bleResult_t Gap_StopAdvertising(void)

Commands the controller to stop advertising.

Remark

GAP Peripheral-only API function.

Return values

- gBleSuccess_c –
- gBleInvalidState_c – Advertising is already stopped or pending to be stopped
- gBleOutOfMemory_c – Cannot allocate memory for the Host task
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version

bleResult_t Gap_Authorize(*deviceId_t* deviceId, uint16_t handle, *gattDbAccessType_t* access)

Authorizes a peer for an attribute in the database.

Remark

This function executes synchronously.

Remark

GATT Server-only API function.

Parameters

- deviceId – **[in]** The peer being authorized.
- handle – **[in]** The attribute handle.
- access – **[in]** The type of access granted (gAccessRead_c or gAccessWrite_c).

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range
- gBleOutOfMemory_c – The nvm index of the deviceId is bigger than the number of allowed bonded devices, or the nvm saving operation did not have enough memory
- gBleInvalidParameter_c – The deviceId parameter is not valid or the access parameter is not gAccessRead_c or gAccessWrite_c
- gBleUnexpectedError_c – Saving to NVM failed (module not initialized or corrupted NVM)
- gBleOverflow_c – the number of handles to be read or written from the NVM is bigger than gcGapMaxAuthorizationHandles_c
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version

bleResult_t Gap_SaveCccd(*deviceId_t* deviceId, uint16_t handle, *gattCccdFlags_t* cccd)

Save the CCCD value for a specific Client and CCCD handle.

Remark

The GATT Server layer saves the CCCD value automatically when it is written by the Client. This API should only be used to save the CCCD in other situations, e.g., when for some reason the application decides to disable notifications/indications for a specific Client.

Remark

This function executes synchronously.

Remark

GATT Server-only API function.

Parameters

- deviceId – **[in]** The peer GATT Client.
- handle – **[in]** The handle of the CCCD as defined in the GATT Database.
- cccd – **[in]** The bit mask representing the CCCD value to be saved.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range
- gBleUnexpectedError_c – Saving to NVM failed (module not initialized or corrupted NVM)
- gBleOutOfMemory_c – The nvm index of the deviceId is bigger than the number of allowed bonded devices, or the nvm saving operation did not have enough memory
- gDevDbCccdLimitReached_c – No more room in NVM

bleResult_t Gap_CheckNotificationStatus(*deviceId_t* deviceId, uint16_t handle, bool_t *pOutIsActive)

Retrieves the notification status for a given Client and a given CCCD handle.

Remark

This function executes synchronously.

Remark

GATT Server-only API function.

Parameters

- deviceId – **[in]** The peer GATT Client.
- handle – **[in]** The handle of the CCCD.
- pOutIsActive – **[out]** The address to store the status into.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range
- gDevDbCccdNotFound_c – The handle is not CCCD
- gBleUnavailable_c – The bond data entry for this device is corrupted

bleResult_t Gap_CheckIndicationStatus(*deviceId_t* deviceId, uint16_t handle, bool_t *pOutIsActive)

Retrieves the indication status for a given Client and a given CCCD handle.

Remark

This function executes synchronously.

Remark

GATT Server-only API function.

Parameters

- deviceId – **[in]** The peer GATT Client.
- handle – **[in]** The handle of the CCCD.
- pOutIsActive – **[out]** The address to store the status into.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gDevDbCccdNotFound_c – The handle is not CCCD.
- gBleUnavailable_c – The bond data entry for this device is corrupted.

bleResult_t Gap_GetBondedDevicesIdentityInformation(*gapIdentityInformation_t* *aOutIdentityAddresses, uint8_t maxDevices, uint8_t *pOutActualCount)

Retrieves a list of the identity information of bonded devices, if any.

Remark

This API may be useful when creating a filter accept list or a resolving list.

Remark

This function executes synchronously.

Parameters

- aOutIdentityAddresses – **[out]** Array of identities to be filled.
- maxDevices – **[in]** Maximum number of identities to be obtained.
- pOutActualCount – **[out]** The actual number of identities written.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version
- gBleInvalidParameter_c – A parameter has an invalid value, is outside the accepted range, or the NVM entry is invalid

bleResult_t Gap_Pair(*deviceId_t* deviceId, const *gapPairingParameters_t* *pPairingParameters)

Initiates pairing with a peer device.

Remark

GAP Central-only API function.

Parameters

- deviceId – **[in]** The peer to pair with.
- pPairingParameters – **[in]** Pairing parameters as required by the SMP.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range
- gBleOutOfMemory_c – Cannot allocate memory for the Host task

bleResult_t Gap_SendPeripheralSecurityRequest(*deviceId_t* deviceId, const *gapPairingParameters_t* *pPairingParameters)

Informs the peer Central about the local security requirements.

Remark

The procedure has the same parameters as the pairing request, but, because it is initiated by the Peripheral, it has no pairing effect. It only informs the Central about the requirements.

Remark

GAP Peripheral-only API function.

Parameters

- deviceId – **[in]** The GAP peer to pair with.
- pPairingParameters – **[in]** Pairing parameters as required by the SMP.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version
- gBleOutOfMemory_c – Cannot allocate memory for the Host task

bleResult_t Gap_EncryptLink(*deviceId_t* deviceId)

Encrypts the link with a bonded peer.

Remark

GAP Central-only API function.

Parameters

- deviceId – **[in]** Device ID of the peer.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.
- gGapDeviceNotBonded_c – Trying to execute an API that is only available for bonded devices.

bleResult_t Gap_EncryptLinkTak(*deviceId_t* deviceId, uint8_t *pTak)

Encrypts the link using a Transient Application Key.

Parameters

- deviceId – **[in]** Device ID of the peer.
- ptak – **[in]** Pointer to the Transient Key.

Return values

bleResult_t – gBleSuccess_c or error.

bleResult_t Gap_AcceptPairingRequest(*deviceId_t* deviceId, const *gapPairingParameters_t* *pPairingParameters)

Accepts the pairing request from a peer.

Remark

This should be called in response to a gPairingRequest_c event.

Remark

GAP Peripheral-only API function.

Parameters

- deviceId – **[in]** The peer requesting authentication.
- pPairingParameters – **[in]** Pairing parameters as required by the SMP.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.

bleResult_t Gap_RejectPairing(*deviceId_t* deviceId, *gapAuthenticationRejectReason_t* reason)

Rejects the peer's authentication request.

Parameters

- deviceId – **[in]** The GAP peer who requested authentication.
- reason – **[in]** Reason why the current device rejects the authentication.

Return values

- gBleSuccess_c –
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.

bleResult_t Gap_EnterPasskey(*deviceId_t* deviceId, uint32_t passkey)

Enters the passkey requested by the peer during the pairing process.

Parameters

- deviceId – **[in]** The GAP peer that requested a passkey entry.
- passkey – **[in]** The peer's secret passkey.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.

bleResult_t Gap_ProvideOob(*deviceId_t* deviceId, const uint8_t *aOob)

Provides the Out-Of-Band data for the SMP Pairing process.

Parameters

- deviceId – **[in]** The pairing device.
- aOob – **[in]** Pointer to OOB data (array of gcSmpOobSize_c size).

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.

bleResult_t Gap_RejectPasskeyRequest(*deviceId_t* deviceId)

Rejects the passkey request from a peer.

Remark

GAP Central-only API function.

Parameters

- deviceId – **[in]** The GAP peer that requested a passkey entry.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_SendSmpKeys(*deviceId_t* deviceId, const *gapSmpKeys_t* *pKeys)

Sends the SMP keys during the SMP Key Exchange procedure.

Parameters

- deviceId – **[in]** The GAP peer who initiated the procedure.
- pKeys – **[in]** The SMP keys of the local device.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.

bleResult_t Gap_RejectKeyExchangeRequest(*deviceId_t* deviceId)

Rejects the Key Exchange procedure with a paired peer.

Parameters

- deviceId – **[in]** The GAP peer who requested the Key Exchange procedure.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_LeScRegeneratePublicKey(void)

Regenerates the private/public key pair used for LE Secure Connections pairing.

Remark

The application should listen for the gLeScPublicKeyRegenerated_c generic event.

Return values

- gBleSuccess_c –
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_LeScValidateNumericValue(*deviceId_t* deviceId, bool_t valid)

Validates the numeric value during the Numeric Comparison LE Secure Connections pairing.

Parameters

- deviceId – **[in]** Device ID of the peer.
- valid – **[in]** TRUE if user has indicated that numeric values are matched, FALSE otherwise.

Return values

- gBleSuccess_c –
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_LeScGetLocalOobData(void)

Retrieves local OOB data used for LE Secure Connections pairing.

Remark

The application should listen for the gLeScLocalOobData_c generic event.

Return values

- gBleSuccess_c –
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_LeScSetPeerOobData(*deviceId_t* deviceId, const *gapLeScOobData_t* *pPeerOobData)

Sets peer OOB data used for LE Secure Connections pairing.

Remark

This function should be called in response to the gConnEvtLeScOobDataRequest_c event.

Parameters

- deviceId – **[in]** Device ID of the peer.
- pPeerOobData – **[in]** OOB data received from the peer.

Return values

- gBleSuccess_c –
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.

bleResult_t Gap_LeScSendKeypressNotification(*deviceId_t* deviceId, *gapKeypressNotification_t* keypressNotification)

Sends a Keypress Notification to the peer.

Remark

This function shall only be called during the passkey entry process and only if both peers support Keypress Notifications.

Parameters

- deviceId – **[in]** Device ID of the peer.
- keypressNotification – **[in]** Value of the Keypress Notification.

Return values

- gBleSuccess_c –
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_ProvideLongTermKey(*deviceId_t* deviceId, const uint8_t *aLtk, uint8_t ltkSize)

Provides the Long Term Key (LTK) to the controller for encryption setup.

Remark

The application should provide the same LTK used during bonding with the respective peer.

Remark

GAP Peripheral-only API function.

Parameters

- deviceId – **[in]** The GAP peer who requested encryption.
- aLtk – **[in]** The Long Term Key.
- ltkSize – **[in]** The Long Term Key size.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.

bleResult_t Gap_ProvideLongTermKeyTak(*deviceId_t* deviceId, const uint8_t *aLtk, uint8_t ltkSize)

Provides the Long Term Key (TAK) to the controller for encryption setup. GAP Peripheral-only API function.

Parameters

- deviceId – **[in]** The GAP peer who requested encryption.
- aLtk – **[in]** The Transient Application Key.
- ltkSize – **[in]** The Transient Application Key size.

Return values

bleResult_t – gBleSuccess_c or error.

bleResult_t Gap_DenyLongTermKey(*deviceId_t* deviceId)

Rejects an LTK request originating from the controller.

Remark

GAP Peripheral-only API function.

Parameters

- deviceId – **[in]** The GAP peer who requested encryption.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_LoadEncryptionInformation(*deviceId_t* deviceId, uint8_t *aOutLtk, uint8_t *pOutLtkSize)

Loads the encryption key for a bonded device.

Remark

This function executes synchronously.

Parameters

- deviceId – **[in]** Device ID of the peer.
- aOutLtk – **[out]** Array of size gcMaxLtkSize_d to be filled with the LTK.
- pOutLtkSize – **[out]** The LTK size.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleUnavailable_c – The bond data entry for this device is corrupted.

bleResult_t Gap_SetLocalPasskey(uint32_t passkey)

Sets the SMP passkey for this device.

Remark

This is the PIN that the peer's user must enter during pairing.

Remark

This function executes synchronously.

Remark

GAP Peripheral-only API function.

Parameters

- passkey – **[in]** The SMP passkey.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gSmInvalidDeviceId_c – The device ID is not valid or it has been disconnected in the meantime.
- gSmInvalidCommandCode_c – An invalid internal state machine code was requested.
- gSmPairingAlreadyStarted_c – Pairing process was already started.
- gSmInvalidInternalOperation_c – A memory corruption or invalid operation may have occurred.
- gSmSmpTimeoutOccurred_c – An SMP timeout has occurred for the peer device.
- gSmInvalidCommandParameter_c – One of the parameters of the SM command is not valid.
- gSmCommandNotSupported_c – The Security Manager does not have the required features or version to support this command
- gSmUnexpectedCommand_c – This command is not or cannot be handled in the current context of the SM.
- gSmSmpPacketReceivedAfterTimeoutOccurred_c – A SMP packet has been received from a peer device for which a pairing procedure has timed out.
- gSmUnexpectedPairingTerminationReason_c – The upper layer tried to cancel the pairing procedure with an unexpected pairing failure reason for the current phase of the pairing procedure.

bleResult_t Gap_SetScanMode(*gapScanMode_t* scanMode, *gapAutoConnectParams_t* *pAutoConnectParams, *gapConnectionCallback_t* connCallback)

Sets internal scan filters and actions.

Remark

This function can be called before Gap_StartScanning. If this function is never called, then the default value of gDefaultScan_c is considered and all scanned devices are reported to the application without any additional filtering or action.

Remark

This function executes synchronously.

Remark

GAP Central-only API function.

Parameters

- scanMode – **[in]** The scan mode to be activated. Default is gDefaultScan_c.
- pAutoConnectParams – **[in]** Pointer to the *gapAutoConnectParams_t* structures if scanMode is set to gAutoConnect_c. The memory used must be persistent and should not change during the next scan periods or until another Gap_SetScanMode is called.
- connCallback – **[in]** Auto-Connect callback. Must be set if scanMode is set to gAutoConnect_c.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.

bleResult_t Gap_SetDecisionInstructions(uint8_t numTest, const *gapDecisionInstructionsData_t* *pDecisionInstructions)

Sets up the Decision Instructions.

Remark

GAP Central-only API function.

Remark

Application should wait for gDecisionInstructionsSetupComplete_c generic event or for gInternalError_c generic event with errorSource = gSetDecisionInstructions_c.

Parameters

- numTest – **[in]** The number of tests in the decision instructions.
- pDecisionInstructions – **[in]** Pointer to an array of gapDecisionInstructionsData_t structures. containing decision instruction for each test.

Return values

- gBleFeatureNotSupported_c – in case gLeExtendedAdv_c or gLeDecisionBasedAdvertisingFiltering_c are not supported.
- gBleInvalidParameter_c – in case at least 1 parameter is incorrect.
- gBleOutOfMemory_c – in case the memory allocation for the app to host message fails.
- gBleSuccess_c – otherwise.

bleResult_t Gap_InitMonitoringAdvertisers(void)

Initialize function table for Monitored Advertisers functionality.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – Feature is not supported.

bleResult_t Gap_AddDeviceToMonAdvList(*bleAddressType_t* addressType, *bleDeviceAddress_t* address, int8_t rssiLowThreshold, int8_t rssiHighThreshold, uint8_t timeout)

This function adds a device to the Monitored Advertisers List.

Parameters

- addressType – **[in]** Address type of the device.
- address – **[in]** Address of the device.
- rssiLowThreshold – **[in]** RSSI low threshold value.
- rssiHighThreshold – **[in]** RSSI high threshold value.
- timeout – **[in]** Timeout value.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – One or more parameters are invalid.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_RemoveDeviceFromMonAdvList(*bleAddressType_t* addressType, *bleDeviceAddress_t* address)

This function removes a device from the Monitored Advertisers List.

Parameters

- addressType – **[in]** Address type of the device.
- address – **[in]** Address of the device.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – One or more parameters are invalid.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_ClearMonAdvList(void)

This function clears the Monitored Advertisers List.

Return values

- gBleSuccess_c –
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_EnableMonAdv(bool_t enable)

This function enables or disables monitoring of advertisers.

Parameters

- enable – **[in]** Enable or disable monitoring.

Return values

- gBleSuccess_c –
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_ReadMonAdvListSize(void)

This function reads the size of the Monitored Advertisers List.

Return values

- gBleSuccess_c –
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_StartScanning(const *gapScanningParameters_t* *pScanningParameters, *gapScanningCallback_t* scanningCallback, *gapFilterDuplicates_t* enableFilterDuplicates, uint16_t duration, uint16_t period)

Optionally sets the scanning parameters and begins scanning.

Remark

Use this API to both set the scanning parameters and start scanning. If pScanningParameters is NULL, scanning is started with the existing settings.

Remark

GAP Central-only API function.

Parameters

- pScanningParameters – **[in]** The scanning parameters; may be NULL.
- scanningCallback – **[in]** The scanning callback.
- enableFilterDuplicates – **[in]** Enable or disable duplicate advertising report filtering
- duration – **[in]** Scan duration expressed in units of 10 ms. Set 0 for continuous scan until explicitly disabled. Used only for BLE5.0, otherwise ignored.
- period – **[in]** Time interval expressed in units of 1.28 seconds from when the Controller started its last Scan_Duration until it begins the subsequent Scan_Duration. Set 0 to disable periodic scanning. Used only for BLE5.0, otherwise ignored.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.
- gBleInvalidState_c – The requested API cannot be called in the current state, scanning was already started.

bleResult_t Gap_StopScanning(void)

Commands the controller to stop scanning.

Remark

GAP Central-only API function.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.
- gBleInvalidState_c – The requested API cannot be called in the current state, scanning is in the process of stopping.

bleResult_t Gap_Connect(const *gapConnectionRequestParameters_t* *pParameters, *gapConnectionCallback_t* connCallback)

Connects to a scanned device.

Remark

GAP Central-only API function.

Parameters

- pParameters – **[in]** Create Connection command parameters.
- connCallback – **[in]** Callback used to receive connection events.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.

bleResult_t Gap_ConnectFromPawr(const *gapConnectionFromPawrParameters_t* *pParameters, *gapConnectionCallback_t* connCallback)

Connects to a device which is doing periodic advertising with responses.

Remark

GAP Central-only API function.

Parameters

- pParameters – **[in]** Create Connection command parameters.
- connCallback – **[in]** Callback used to receive connection events.

Return values

- gBleSuccess_c –

- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.

bleResult_t Gap_Disconnect(*deviceId_t* deviceId)

Initiates disconnection from a connected peer device.

Parameters

- deviceId – **[in]** The connected peer to disconnect from.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_SaveCustomPeerInformation(*deviceId_t* deviceId, const uint8_t *pInfo, uint16_t offset, uint16_t infoSize)

Saves custom peer information in raw data format.

Remark

This function can be called by the application to save custom information about the peer device, e.g., Service Discovery data (to avoid doing it again on reconnection).

Remark

This function executes synchronously.

Parameters

- deviceId – **[in]** Device ID of the GAP peer.
- pInfo – **[in]** Pointer to the beginning of the data.
- offset – **[in]** Offset from the beginning of the reserved memory area.
- infoSize – **[in]** Data size (maximum equal to gcReservedFlashSizeForCustomInformation_d).

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleOutOfMemory_c – The nvm index of the deviceId is bigger than the number of allowed bonded devices, or the nvm saving operation did not have enough memory.

bleResult_t Gap_LoadCustomPeerInformation(*deviceId_t* deviceId, uint8_t *pOutInfo, uint16_t offset, uint16_t infoSize)

Loads the custom peer information in raw data format.

Remark

This function can be called by the application to load custom information about the peer device, e.g., Service Discovery data (to avoid doing it again on reconnection).

Remark

This function executes synchronously.

Parameters

- deviceId – **[in]** Device ID of the GAP peer.
- pOutInfo – **[out]** Pointer to the beginning of the allocated memory.
- offset – **[in]** Offset from the beginning of the reserved memory area.
- infoSize – **[in]** Data size (maximum equal to gcReservedFlashSizeForCustomInformation_d).

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleUnavailable_c – The bond data entry for this device is corrupted.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleOutOfMemory_c – The nvm index of the deviceId is bigger than the number of allowed bonded devices, or the nvm saving operation did not have enough memory.

bleResult_t Gap_LoadCustomBondedDeviceInformation(uint8_t nvmIndex, uint8_t *pOutInfo, uint16_t offset, uint16_t infoSize)

Loads the custom bonded device information in raw data format. Unlike Gap_LoadCustomPeerInformation, it does not require an active connection.

Remark

This function can be called by the application to load custom information about the bonded device.

Remark

This function executes synchronously.

Parameters

- nvmIndex – **[in]** Index of the device in NVM bonding area.
- pOutInfo – **[out]** Pointer to the beginning of the allocated memory.
- offset – **[in]** Offset from the beginning of the reserved memory area.
- infoSize – **[in]** Data size (maximum equal to gcReservedFlashSizeForCustomInformation_d).

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleUnavailable_c – The bond data entry for this device is corrupted.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleOutOfMemory_c – The nvm index of the deviceId is bigger than the number of allowed bonded devices, or the nvm saving operation did not have enough memory.

bleResult_t Gap_CheckIfBonded(*deviceId_t* deviceId, bool_t *pOutIsBonded, uint8_t *pOutNvmIndex)

Returns whether or not a connected peer device is bonded and the NVM index.

Remark

This function executes synchronously.

Parameters

- deviceId – **[in]** Device ID of the GAP peer.
- pOutIsBonded – **[out]** Boolean to be filled with the bonded flag.
- pOutNvmIndex – **[out]** If bonded, to be filled optionally with the NVM index.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.

bleResult_t Gap_CheckIfConnected(*bleAddressType_t* addressType, const *bleDeviceAddress_t* aAddress, bool_t addrResolved, bool_t *pOutIsConnected)

Checks if a connection exists between the local device and another device that has the provided address information. This function executes synchronously.

Parameters

- addressType – **[in]** Peer address type
- aAddress – **[in]** Peer address
- addrResolved – **[in]** Set to TRUE if the address contained in the addressType and aAddress fields is the identity address of a resolved RPA.
- pOutIsConnected – **[out]** TRUE if the device is found

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.

bleResult_t Gap_CheckNvmIndex(uint8_t nvmIndex, bool_t *pOutIsFree)

Returns whether or not the given NVM index is free.

Remark

This function executes synchronously.

Parameters

- nvmIndex – **[in]** NVM index.
- pOutIsFree – **[out]** TRUE if free, FALSE if occupied.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.

bleResult_t Gap_ReadFilterAcceptListSize(void)

Retrieves the size of the Filter Accept List.

Remark

Response is received in the gFilterAcceptListSizeRead_c generic event.

Return values

- gBleSuccess_c –
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_ClearFilterAcceptList(void)

Removes all addresses from the Filter Accept List, if any.

Remark

Confirmation is received in the gFilterAcceptListCleared_c generic event.

Return values

gBleSuccess_c –

bleResult_t Gap_AddDeviceToFilterAcceptList(*bleAddressType_t* addressType, const *bleDeviceAddress_t* address)

Adds a device address to the Filter Accept List.

Parameters

- address – **[in]** The address of the filter accept listed device.
- addressType – **[in]** The device address type (public or random).

Return values

- gBleSuccess_c –
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_RemoveDeviceFromFilterAcceptList(*bleAddressType_t* addressType, const *bleDeviceAddress_t* address)

Removes a device address from the Filter Accept List.

Parameters

- address – **[in]** The address of the filter accept listed device.
- addressType – **[in]** The device address type (public or random).

Return values

- gBleSuccess_c –
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_ReadPublicDeviceAddress(void)

Reads the device's public address from the controller.

Remark

The application should listen for the gPublicAddressRead_c generic event.

Return values

- gBleSuccess_c –
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_CreateRandomDeviceAddress(const uint8_t *aIrk, const uint8_t *aRandomPart)

Requests the controller to create a random address.

Remark

The application should listen for the gRandomAddressReady_c generic event. Note that this does not set the random address in the Controller. To set the random address, call *Gap_SetRandomAddress()* with the generated address contained in the event data.

Parameters

- aIrk – **[in]** The Identity Resolving Key to be used for a private resolvable address or NULL for a private non-resolvable address (fully random).
- aRandomPart – **[in]** If aIrk is not NULL, this is a 3-byte array containing the Random Part of a Private Resolvable Address, in LSB to MSB order; the most significant two bits of the most significant byte (aRandomPart[3] & 0xC0) are ignored. This may be NULL, in which case the Random Part is randomly generated internally.

Return values

- gBleSuccess_c –
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_SaveDeviceName(*deviceId_t* deviceId, const uchar_t *pName, uint8_t cNameSize)

Store the name of a bonded device.

Remark

This function copies cNameSize characters from the pName array and adds the NULL character to terminate the string.

Remark

This function executes synchronously.

Parameters

- deviceId – **[in]** Device ID for the active peer which name is saved.
- pName – **[in]** Array of characters holding the name.
- cNameSize – **[in]** Number of characters to be saved.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleOutOfMemory_c – The nvm index of the deviceId is bigger than the number of allowed bonded devices, or the nvm saving operation did not have enough memory.

bleResult_t Gap_GetBondedDevicesCount(uint8_t *pOutBondedDevicesCount)

Retrieves the number of bonded devices.

Remark

This function executes synchronously.

Parameters

- pOutBondedDevicesCount – **[out]** Pointer to integer to be written.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.

bleResult_t Gap_GetBondedDeviceName(uint8_t nvmIndex, uchar_t *pOutName, uint8_t maxNameSize)

Retrieves the name of a bonded device.

Remark

nvmIndex is an integer ranging from 0 to N-1, where N is the number of bonded devices and can be obtained by calling Gap_GetBondedDevicesCount(&N).

Remark

This function executes synchronously.

Parameters

- nvmIndex – **[in]** Index of the device in NVM bonding area.
- pOutName – **[out]** Destination array to copy the name into.
- maxNameSize – **[in]** Maximum number of characters to be copied, including the terminating NULL character.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleUnavailable_c – The bond data entry for this device is corrupted.

bleResult_t Gap_SetBondedDeviceName(uint8_t nvmIndex, const uchar_t *pName, uint8_t cNameSize)

Store the name of a bonded device.

Remark

This function copies cNameSize characters from the pName array and adds the NULL character to terminate the string.

Remark

This function executes synchronously.

Parameters

- nvmIndex – **[in]** Index of the device in NVM bonding area.
- pName – **[in]** Array of characters holding the name.
- cNameSize – **[in]** Number of characters to be saved.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleOutOfMemory_c – The nvm index of the deviceId is bigger than the number of allowed bonded devices, or the nvm saving operation did not have enough memory.

bleResult_t Gap_RemoveBond(uint8_t nvmIndex)

Removes the bond with a device.

Remark

This API requires that there are no active connections at call time. The *Gap_CheckIfBonded()* API can be called to obtain the nvmIndex of a bonded peer.

Remark

This function executes synchronously.

Parameters

- nvmIndex – **[in]** Index of the device in the NVM bonding area.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleUnavailable_c – The bond data entry for this device is corrupted.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleInvalidState_c – The bond is not active.

bleResult_t Gap_RemoveAllBonds(void)

Removes all bonds with other devices.

Remark

This API requires that there are no active connections at call time.

Remark

This function executes synchronously.

Return values

- gBleSuccess_c –

- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version
- gBleInvalidState_c – At least one bond is still active.

bleResult_t Gap_ReadRadioPowerLevel(*gapRadioPowerLevelReadType_t* txReadType, *deviceId_t* deviceId)

Reads the power level of the controller's radio. The response is contained in the gConnEvtTxPowerLevelRead_c connection event when reading connection TX power level, the gAdvTxPowerLevelRead_c generic event when reading the advertising TX power level, or the gConnEvtRssiRead_c connection event when reading the RSSI.

Parameters

- txReadType – **[in]** Advertising or connection Tx power
- deviceId – **[in]** Peer identifier (for connections only, otherwise ignored)

Return values

- gBleSuccess_c –
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_SetTxPowerLevel(uint8_t powerLevel, *bleTransmitPowerChannelType_t* channelType)

Sets the Tx power level on the controller's radio.

Remark

The application should listen for the gTxPowerLevelSetComplete_c generic event.

Remark

This function executes synchronously.

Remark

For QN908x platform this command is not supported by the controller.

Remark

Instead, use RF_SetTxPowerLevel API to set the desired TX power level.

Parameters

- powerLevel – **[in]** Power level as specified in the controller interface.
- channelType – **[in]** The advertising or connection channel type.

Return values

gBleSuccess_c –

bleResult_t Gap_VerifyPrivateResolvableAddress(uint8_t nvmIndex, const *bleDeviceAddress_t* aAddress)

Verifies a Private Resolvable Address with a bonded device's IRK.

Remark

nvmIndex is an integer ranging from 0 to N-1, where N is the number of bonded devices and can be obtained by calling Gap_GetBondedDevicesCount(&N); the application should listen to the gPrivateResolvableAddressVerified_c event.

Parameters

- nvmIndex – **[in]** Index of the device in NVM bonding area whose IRK must be checked.
- aAddress – **[in]** The Private Resolvable Address to be verified.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_SetRandomAddress(const *bleDeviceAddress_t* aAddress)

Sets a random address into the Controller.

Remark

The application should listen for the gRandomAddressSet_c generic event.

Parameters

- aAddress – **[in]** The Private Resolvable, Private Non-Resolvable, or Static Random Address.

Return values

- gBleSuccess_c –
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_ReadControllerLocalRPA(const *bleIdentityAddress_t* *pIdAddress)

Reads the device's Local Private Address for a specific peer device from the controller.

Remark

The application should listen for the gControllerLocalRPARead_c generic event.

Parameters

- pIdAddress – **[in]** pointer to the peer identity address the local private address should be retrieved for.

Return values

- gBleSuccess_c –
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.
- gBleInvalidState_c – The Controller privacy is not enabled.

bleResult_t Gap_SetDefaultPairingParameters(const *gapPairingParameters_t* *pPairingParameters)

Sets the default pairing parameters to be used by automatic pairing procedures.

Remark

When these parameters are set, the Security Manager automatically responds to a Pairing Request or a Peripheral Security Request using these parameters. If NULL is provided, it returns to the default state where all security requests are sent to the application.

Remark

This function executes synchronously.

Parameters

- pPairingParameters – **[in]** Pairing parameters as required by the SMP or NULL.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.

bleResult_t Gap_UpdateConnectionParameters(*deviceId_t* deviceId, uint16_t intervalMin, uint16_t intervalMax, uint16_t peripheralLatency, uint16_t timeoutMultiplier, uint16_t minCeLength, uint16_t maxCeLength)

Request a set of new connection parameters.

Parameters

- deviceId – **[in]** The DeviceID for which the command is intended
- intervalMin – **[in]** The minimum value for the connection event interval
- intervalMax – **[in]** The maximum value for the connection event interval
- peripheralLatency – **[in]** The peripheral latency parameter
- timeoutMultiplier – **[in]** The connection timeout parameter
- minCeLength – **[in]** The minimum value for the connection event length
- maxCeLength – **[in]** The maximum value for the connection event length

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.

- gBleOutOfMemory_c – Cannot allocate memory for the Host task.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.

Pre
A connection must be in place

bleResult_t Gap_EnableUpdateConnectionParameters(*deviceId_t* deviceId, bool_t enable)
Update the connection parameters.

Remark

The LE central Host may accept the requested parameters or reject the request

---

Parameters

- deviceId – **[in]** The DeviceID for which the command is intended
- enable – **[in]** Allow/disallow the parameters update

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.
- gBleInvalidParameter_c – The device id is not valid.

Pre
A connection must be in place

bleResult_t Gap_UpdateLeDataLength(*deviceId_t* deviceId, uint16_t txOctets, uint16_t txTime)
Update the Tx Data Length.

Remark

The response is contained in the gConnEvtLeDataLengthChanged_c connection event.

Parameters

- deviceId – **[in]** The DeviceID for which the command is intended
- txOctets – **[in]** Maximum transmission number of payload octets
- txTime – **[in]** Maximum transmission time

Return values

- gBleSuccess_c –
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.
- gBleInvalidParameter_c – The device id is not valid.

Pre
A connection must be in place

bleResult_t Gap_SetDefaultConnectionSubrateParameters(const *gapConnectionSubrateParameters_t* *pConnSubrateParam)

The function sets the initial values of the acceptable parameters for subrating requests, for all future connections where the Controller is Central. User should wait for a gLeSetDefaultConnectionSubrateParametersSetupComplete_c generic event or for a gInternalError_c generic event with source gSetDefaultConnectionSubrateParameters_c .

Parameters

- pConnSubrateParam – **[in]** Pointer to the new set of connection subrate parameters

Return values

bleResult_t – gBleSuccess_c or error.

bleResult_t Gap_ConnectionSubrateRequest(*deviceId_t* deviceId, const *gapConnectionSubrateParameters_t* *pConnSubrateParam)

Requests a set of new connection subrate parameters. A connection must be in place. When initiated on a Central this procedure triggers the Link Layer Connection Subrate Update procedure. When initiated on a Peripheral this procedure triggers the Link Layer Connection Subrate Request procedure. User should wait for a gConnEvtLeSubrateChange_c connection event or for a gInternalError_c generic event with source gConnectionSubrateRequest_c.

Parameters

- deviceId – **[in]** The DeviceID for which the command is intended
- pConnSubrateParam – **[in]** Pointer to the new set of connection subrate parameters

Return values

bleResult_t – gBleSuccess_c or error.

bleResult_t Gap_EnableHostPrivacy(bool_t enable, const uint8_t *aIrk)

Enables or disables Host Privacy (automatic regeneration of a Private Address).

Remark

The application should listen for the gHostPrivacyStateChanged_c generic event.

Parameters

- enable – **[in]** TRUE to enable, FALSE to disable.
- aIrk – **[in]** Local IRK to be used for Resolvable Private Address generation or NULL for Non-Resolvable Private Address generation. Ignored if enable is FALSE.

Return values

- gBleSuccess_c –
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_EnableControllerPrivacy(bool_t enable, const uint8_t *aOwnIrk, uint8_t peerIdCount, const *gapIdentityInformation_t* *aPeerIdentities)

Enables or disables Controller Privacy (Enhanced Privacy feature).

Remark

The application should listen for the gControllerPrivacyStateChanged_c generic event.

Parameters

- enable – **[in]** TRUE to enable, FALSE to disable.
- aOwnIrk – **[in]** Local IRK. Ignored if enable is FALSE, otherwise shall not be NULL.
- peerIdCount – **[in]** Size of aPeerIdentities array. Shall not be zero or greater than gcGapControllerResolvingListSize_c. Ignored if enable is FALSE.
- aPeerIdentities – **[in]** Array of peer identity addresses and IRKs. Ignored if enable is FALSE, otherwise shall not be NULL.

Return values

- gBleSuccess_c –
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleFeatureNotSupported_c – Cannot use Device Privacy mode if controller does not support LE Set Privacy Mode Command.

bleResult_t Gap_SetPrivacyMode(uint8_t nvmIndex, *blePrivacyMode_t* privacyMode)

Sets the privacy mode to an existing bond.

Remark

The change has no effect (other than the change in NVM) unless controller privacy is enabled for the bonded identities.

Parameters

- nvmIndex – **[in]** Index of the device in the NVM bonding area.
- privacyMode – **[in]** Controller privacy mode: Network or Device

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – Cannot use Device Privacy mode if controller does not support LE Set Privacy Mode Command.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleUnavailable_c – The bond data entry for this device is corrupted.
- gBleOutOfMemory_c – There is no more room to allocate memory for a bonding entry.

bleResult_t Gap_ControllerTest(*gapControllerTestCmd_t* testCmd, uint8_t radioChannel, uint8_t txDataLength, *gapControllerTestTxType_t* txPayloadType)

Commands a Controller Test procedure.

Remark

The application should listen for the gControllerTestEvent_c generic event.

Remark

This API function is available only in the full-featured host library.

Parameters

- testCmd – Command type - "start TX test", "start RX test" or "end test".
- radioChannel – Radio channel index. Valid range: 0-39. Frequency will be F[MHz] = 2402 + 2 * index. Effective range: 2402-2480 MHz. Ignored if command is "end test".
- txDataLength – Size of packet payload for TX tests. Ignored if command is "start RX test" or "end test".
- txPayloadType – Type of packet payload for TX tests. Ignored if command is "start RX test" or "end test".

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.

bleResult_t Gap_ModifySleepClockAccuracy(*gapSleepClockAccuracy_t* action)

Requests the Controller to change its sleep clock accuracy for testing purposes.

Remark

The application should listen for the generic event.

Parameters

- action – Specifies whether the sleep clock should be changed to one that is more accurate or one that is less accurate.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.

bleResult_t Gap_LeReadPhy(*deviceId_t* deviceId)

Read the Tx and Rx Phy on the connection with a device.

Remark

The application should listen for the gLePhyEvent_c generic event. This API is available only in the Bluetooth 5.0 Host Stack.

Parameters

- deviceId – **[in]** Device ID of the peer.

Return values

- gBleSuccess_c –
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_LeSetPhy(bool_t defaultMode, *deviceId_t* deviceId, uint8_t allPhys, uint8_t txPhys, uint8_t rxPhys, uint16_t phyOptions)

Set the Tx and Rx Phy preferences on the connection with a device or all subsequent connections.

Remark

The application should listen for the gLePhyEvent_c generic event. This API is available only in the Bluetooth 5.0 Host Stack.

Parameters

- defaultMode – **[in]** Use the provided values for all subsequent connections
- deviceId – **[in]** Device ID of the peer Ignored if defaultMode is TRUE.
- allPhys – **[in]** Host preferences on Tx and Rx Phy, as defined by gapLeAllPhyFlags_t
- txPhys – **[in]** Host preferences on Tx Phy, as defined by gapLePhyFlags_t, ignored for gLeTxPhyNoPreference_c
- rxPhys – **[in]** Host preferences on Rx Phy, as defined by gapLePhyFlags_t, ignored for gLeRxPhyNoPreference_c
- phyOptions – **[in]** Host preferences on Coded Phy, as defined by gapLePhyOptionsFlags_t Ignored if defaultMode is TRUE.

Return values

- gBleSuccess_c –
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version

bleResult_t Gap_ControllerEnhancedNotification(uint16_t eventType, *deviceId_t* deviceId)

Configures enhanced notifications on advertising, scanning and connection events on the controller.

Remark

The application should listen for the gControllerNotificationEvent_c generic event.

Remark

This function executes synchronously.

Parameters

- eventType – **[in]** Event type selection as specified by bleNotificationEventType_t.
- deviceId – **[in]** Device ID of the peer, used only for connection events.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleOutOfMemory_c – The nvm index of the deviceId is bigger than the number of allowed bonded devices, or the nvm saving operation did not have enough memory.

bleResult_t Gap_BleAdvIndexChange(*bleAdvIndexType_t* advIndexType, uint8_t aUserDefinedChannels[3])

This function configures the advertising index type.

Parameters

- advIndexType – **[in]** Advertising index type
- aUserDefinedChannels – **[in]** User defined channels array

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.

bleResult_t Gap_LoadKeys(uint8_t nvmIndex, *gapSmpKeys_t* *pOutKeys, *gapSmpKeyFlags_t* *pOutKeyFlags, bool_t *pOutLeSc, bool_t *pOutAuth)

Retrieves the keys from an existing bond with a device.

Remark

This API requires that the aAddress in the pOutKeys shall not be NULL.

Remark

The application will check pOutKeyFlags to see which information is valid in pOutKeys.

Remark

This function executes synchronously.

Parameters

- nvmIndex – **[in]** Index of the device in the NVM bonding area.
- pOutKeys – **[out]** Pointer to fill the keys distributed during pairing.
- pOutKeyFlags – **[out]** Pointer to indicate which keys were distributed during pairing.
- pOutLeSc – **[out]** Pointer to mark if LE Secure Connections was used during pairing.
- pOutAuth – **[out]** Pointer to mark if the device was authenticated for MITM during pairing.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.

bleResult_t Gap_SaveKeys(uint8_t nvmIndex, const *gapSmpKeys_t* *pKeys, bool_t leSc, bool_t auth)

Saves the keys to a new or existing bond based on OOB information.

Remark

This API requires that the aAddress in the pKeys shall not be NULL.

Remark

If any of the keys are passed as NULL, they will not be saved.

Remark

The application listen for gBondCreatedEvent_c to confirm the bond was created.

Parameters

- nvmIndex – **[in]** Index of the device in the NVM bonding area.
- pKeys – **[in]** Pointer to the keys distributed during pairing.
- leSc – **[in]** Indicates if LE Secure Connections was used during pairing.
- auth – **[in]** Indicates if the device was authenticated for MITM during pairing.

Return values

- gBleSuccess_c –

- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_SetChannelMap(const *bleChannelMap_t* channelMap)

Set the channel map in the Controller and trigger an LL channel map update. Specifies the channel map for data, secondary advertising and periodic physical channels.

Remark

The application should listen for the gChannelMapSet_c generic event.

Remark

This function executes synchronously.

Remark

API supported on GAP Central devices or Broadcaster with Extended Advertising support.

Parameters

- channelMap – **[in]** Array with the channels using 0 for bad channels and 1 for unknown.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.

bleResult_t Gap_ReadChannelMap(*deviceId_t* deviceId)

Reads the channel map of a connection.

Remark

The application should listen for the gConnEvtChannelMapRead_c connection event.

Remark

This function executes synchronously.

Parameters

- deviceId – **[in]** Device ID of the peer.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.

- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.

bleResult_t Gap_SetExtAdvertisingParameters(*gapExtAdvertisingParameters_t* *pAdvertisingParameters)

Sets up the Extended Advertising Parameters.

Remark

GAP Peripheral-only API function.

Parameters

- pAdvertisingParameters – **[in]** Pointer to gapExtAdvertisingParameters_t structure.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.
- gGapAnotherProcedureInProgress_c – Another set extended advertisment parameters command is in progress.

bleResult_t Gap_SetExtAdvertisingData(uint8_t handle, *gapAdvertisingData_t* *pAdvertisingData, *gapScanResponseData_t* *pScanResponseData)

Sets up the Extended Advertising and Extended Scan Response Data.

Remark

Any of the parameters may be NULL, in which case they are ignored. Therefore, this function can be used to set any of the parameters individually or both at once.

Remark

GAP Peripheral-only API function.

Parameters

- handle – **[in]** The ID of the advertising set
- pAdvertisingData – **[in]** Pointer to *gapAdvertisingData_t* structure or NULL.
- pScanResponseData – **[in]** Pointer to gapScanResponseData_t structure or NULL.

Return values

- gBleSuccess_c –

- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleOutOfMemory_c – The nvm index of the deviceId is bigger than the number of allowed bonded devices, or the nvm saving operation did not have enough memory.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleInvalidState_c – The advertising set was not configured previously using Gap_SetExtAdvertisingParameters.
- gGapAdvDataTooLong_c – The advertising data is too long.

bleResult_t Gap_SetExtAdvertisingDecisionData(uint8_t handle, const *gapAdvertisingDecisionData_t* *pAdvertisingDecisionData)

Sets up the Extended Advertising Decision Data.

Remark

pDecisionData or pKey in the *gapAdvertisingDecisionData_t* structure may be NULL but not both.

Remark

GAP Peripheral-only API function.

Remark

Application should wait for gExtAdvertisingDecisionDataSetupComplete_c or generic event or for gInternalError_c generic event with errorSource = gSetExtAdvDecisionData_c.

Parameters

- handle – **[in]** The ID of the advertising set
- pAdvertisingDecisionData – **[in]** Pointer to *gapAdvertisingDecisionData_t* structure.

Return values

- gBleFeatureNotSupported_c – in case gLeExtendedAdv_c or gLeDecisionBasedAdvertisingFiltering_c are not supported.
- gBleInvalidParameter_c – in case at least 1 parameter is incorrect.
- gBleOutOfMemory_c – in case the memory allocation for the app to host message fails.
- gBleSuccess_c – otherwise.

bleResult_t Gap_StartExtAdvertising(*gapAdvertisingCallback_t* advertisingCallback, *gapConnectionCallback_t* connectionCallback, uint8_t handle, uint16_t duration, uint8_t maxExtAdvEvents)

Commands the controller to start the extended advertising.

Remark

The advertisingCallback confirms or denies whether the advertising has started. The connectionCallback is only used if a connection gets established during advertising.

Remark

GAP Peripheral-only API function.

Parameters

- advertisingCallback – **[in]** Callback used by the application to receive advertising events. Can be NULL.
- connectionCallback – **[in]** Callback used by the application to receive connection events. Can be NULL.
- handle – **[in]** The ID of the advertising set
- duration – **[in]** The duration of the advertising
- maxExtAdvEvents – **[in]** The maximum number of advertising events

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.
- gBleInvalidState_c – The advertising is already started or pending to be started.

bleResult_t Gap_StopExtAdvertising(uint8_t handle)

Commands the controller to stop extended advertising for set ID.

Remark

GAP Peripheral-only API function.

Parameters

- handle – **[in]** The ID of the advertising set

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.

- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleInvalidState_c – The advertising set is not started.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_RemoveAdvSet(uint8_t handle)

Commands the controller to remove the specified advertising set and all its data.

Remark

GAP Peripheral-only API function.

Parameters

- handle – **[in]** The ID of the advertising set

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_SetPeriodicAdvParameters(*gapPeriodicAdvParameters_t* *pAdvertisingParameters)

Sets up the Periodic Advertising Parameters.

Remark

GAP Peripheral-only API function.

Parameters

- pAdvertisingParameters – **[in]** Pointer to gapPeriodicAdvParameters_t structure.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_SetPeriodicAdvSubeventData(uint8_t advHandle, const *gapPeriodicAdvertisingSubeventData_t* *pData)

Sets up the Periodic Advertising Subevent Data.

Remark

GAP Peripheral-only API function.

Parameters

- advHandle – **[in]** Advertising handle.
- gapPeriodicAdvertisingSubeventData_t – Pointer to *gapPeriodicAdvertising-SubeventData_t* structure.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_SetPeriodicAdvResponseData(uint16_t syncHandle, const *gapPeriodicAdvertisingResponseData_t* *pData)

Sets up the Periodic Advertising Response Data.

Parameters

- syncHandle – **[in]** Sync handle.
- gapPeriodicAdvertisingResponseData_t – Pointer to *gapPeriodicAdvertisin-gResponseData_t* structure.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_SetPeriodicSyncSubevent(uint16_t syncHandle, const *gapPeriodicSyncSubeventParameters_t* *pParams)

Instructs the controller to sync with a subset of the subevents within a PAWR train identified by syncHandle.

Parameters

- syncHandle – **[in]** Sync handle.
- gapPeriodicSyncSubeventParameters_t – Pointer to *gapPeriodicSync-SubeventParameters_t* structure.

Return values

- gBleSuccess_c –

- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_SetPeriodicAdvertisingData(uint8_t handle, *gapAdvertisingData_t* *pAdvertisingData, bool_t bUpdateDID)

Sets up the Periodic Advertising Data.

Remark

GAP Peripheral-only API function.

Parameters

- handle – **[in]** The ID of the periodic advertising set
- pAdvertisingData – **[in]** Pointer to *gapAdvertisingData_t* structure.
- bUpdateDid – **[in]** If TRUE, use operation mode gHciExtDataUnchanged_c.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleInvalidState_c – The advertising set was not previously configured using Gap_SetPeriodicAdvParameters.
- gGapAdvDataTooLong_c – The advertising data is too long.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_StartPeriodicAdvertising(uint8_t handle, bool_t bIncludeADI)

Commands the controller to start periodic advertising for set ID.

Remark

GAP Peripheral-only API function.

Parameters

- handle – **[in]** The ID of the periodic advertising set
- bIncludeADI – **[in]** If TRUE, include ADI field in AUX_SYNC_IND PDUs

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.

- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleInvalidState_c – The advertising set was not previously configured using Gap_SetPeriodicAdvParameters.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_StopPeriodicAdvertising(uint8_t handle)

Commands the controller to stop periodic advertising for set ID.

Remark

GAP Peripheral-only API function.

Parameters

- handle – **[in]** The ID of the periodic advertising set

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleInvalidState_c – The advertising set was not previously configured using Gap_SetPeriodicAdvParameters.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_UpdatePeriodicAdvList(*gapPeriodicAdvListOperation_t* operation, *bleAddressType_t* addrType, uint8_t *pAddr, uint8_t SID)

Manage the periodic advertising list.

Remark

GAP Central-only API function.

Parameters

- operation – **[in]** The list operation: add/remove a device, or clear all.
- addrType – **[in]** The address type of the periodic advertiser.
- pAddr – **[in]** Pointer to the advertiser's address.
- SID – **[in]** The ID of the advertising set.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.

- gGapAnotherProcedureInProgress_c – Periodic advertising create sync is in progress.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_PeriodicAdvCreateSync(*gapPeriodicAdvSyncReq_t* *pReq)

Start tracking periodic advertisements. Scanning is required to be ON for this request to be processed, so the scanning callback will receive the periodic advertising events.

Remark

GAP Central-only API function.

Parameters

- pReq – **[in]** Pointer to the Sync Request parameters.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gGapAnotherProcedureInProgress_c – Periodic advertising create sync is in progress.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_PeriodicAdvTerminateSync(uint16_t syncHandle)

Stop tracking periodic advertisements.

Remark

GAP Central-only API function.

Parameters

- syncHandle – **[in]** Used to identify the periodic advertiser

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gGapAnotherProcedureInProgress_c – Periodic advertising create sync is in progress.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_PeriodicAdvReceiveEnable(uint16_t syncHandle, bool_t enableDuplicateFiltering)

Enable Periodic Advertisement Sync Transfer Controller feature.

Parameters

- syncHandle – **[in]** Used to identify the periodic advertiser
- enableDuplicateFiltering – **[in]** Used to enable or disable duplicate filtering

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_PeriodicAdvReceiveDisable(uint16_t syncHandle)

Disable Periodic Advertisement Sync Transfer Controller feature.

Parameters

- syncHandle – **[in]** Used to identify the periodic advertiser

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_PeriodicAdvSyncTransfer(*gapPeriodicAdvSyncTransfer_t* *pParam)

Instruct the Controller to send synchronization information about the periodic advertising train identified by the sync handle to a connected device.

Parameters

- pParam – **[in]** Pointer to the command arguments

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_PeriodicAdvSetInfoTransfer(*gapPeriodicAdvSetInfoTransfer_t* *pParam)

Instruct the Controller to send synchronization information about the periodic advertising to a connected device.

Parameters

- pParam – **[in]** Pointer to the command arguments

Return values

- gBleSuccess_c –

- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_SetPeriodicAdvSyncTransferParams(*gapSetPeriodicAdvSyncTransferParams_t* *pParam)

Specify how the Controller will process periodic advertising synchronization information received from the device identified by the device id.

Parameters

- pParam – **[in]** Pointer to the command arguments

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_SetDefaultPeriodicAdvSyncTransferParams(*gapSetPeriodicAdvSyncTransferParams_t* *pParam)

Specify the initial value for the mode, skip, timeout, and Constant Tone Extension type (set by the Gap_SetPeriodicAdvSyncTransferParams command) to be used for all subsequent connections over the LE transport.

Parameters

- pParam – **[in]** Pointer to the command arguments

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_ResumeLeScStateMachine(computeDhKeyParam_t *pData)

Resume the pairing process. At this point the ECDH key must be computed. This function should be called only for secured LE connections. In any other cases the user should make his own code for handling the case when the ECDH computation is completed.

Parameters

- pData – **[in]** Pointer to the data used to resume the host state machine. The data is allocated by the stack when it requests an ECDH multiplication. It is also freed by the stack at the end of the multiplication.

Return values

- gBleSuccess_c –
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.

bleResult_t Gap_SetConnectionlessCteTransmitParameters(*gapConnectionlessCteTransmitParams_t* *pTransmitParams)

Set Connectionless CTE Transmit Parameters for an advertising set.

Parameters

- pTransmitParams – **[in]** Pointer to struct containing parameters.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_EnableConnectionlessCteTransmit(uint8_t handle, *bleCteTransmitEnable_t* enable)

Enable or disable Connectionless CTE Transmit for an advertising set.

Parameters

- handle – **[in]** Advertising set handle.
- enable – **[in]** Enable or disable CTE Transmit.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_EnableConnectionlessIqSampling(uint16_t syncHandle, *gapConnectionlessIqSamplingParams_t* *pIqSamplingParams)

Enable or disable Connectionless IQ sampling for an advertising train.

Parameters

- syncHandle – **[in]** Used to identify advertising train.
- pIqSamplingParams – **[in]** Pointer to struct containing parameters.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_SetConnectionCteReceiveParameters(*deviceId_t* deviceId, *gapConnectionCteReceiveParams_t* *pReceiveParams)

Set CTE Receive Parameters for a certain connection.

Parameters

- deviceId – **[in]** Peer device ID.
- pReceiveParams – **[in]** Pointer to struct containing parameters.

Return values

- gBleSuccess_c –

- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_SetConnectionCteTransmitParameters(*deviceId_t* deviceId, *gapConnectionCteTransmitParams_t* *pTransmitParams)

Set CTE Transmit Parameters for a certain connection.

Parameters

- deviceId – **[in]** Peer device ID.
- pTransmitParams – **[in]** Pointer to struct containing parameters.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_EnableConnectionCteRequest(*deviceId_t* deviceId, *gapConnectionCteReqEnableParams_t* *pCteReqEnableParams)

Enable or disable CTE Request procedure for a certain connection.

Parameters

- deviceId – **[in]** Peer device ID.
- pCteReqEnableParams – **[in]** Pointer to struct containing parameters.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_EnableConnectionCteResponse(*deviceId_t* deviceId, *bleCteRspEnable_t* enable)

Enable or disable sending CTE Responses for a certain connection.

Parameters

- deviceId – **[in]** Peer device ID.
- enable – **[in]** Enable or disable sending CTE Responses.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_ReadAntennaInformation(void)

Read Antenna Information.

Remark

Antenna information contained in gAntennaInformationRead_c generic event.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_EnhancedReadTransmitPowerLevel(*deviceId_t* deviceId, *blePowerControlPhyType_t* phy)

Read local current and maximum tx power levels for a certain connection and PHY.

Parameters

- deviceId – **[in]** Peer device ID.
- phy – **[in]** PHY.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_ReadRemoteTransmitPowerLevel(*deviceId_t* deviceId, *blePowerControlPhyType_t* phy)

Read remote tx power for a certain connection and PHY.

Parameters

- deviceId – **[in]** Peer device ID.
- phy – **[in]** PHY.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_SetPathLossReportingParameters(*deviceId_t* deviceId, *gapPathLossReportingParams_t* *pPathLossReportingParams)

Set path loss threshold reporting parameters for a certain connection.

Parameters

- deviceId – **[in]** Peer device ID.
- pPathLossReportingParams – **[in]** Pointer to struct containing parameters.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_EnablePathLossReporting(*deviceId_t* deviceId, *blePathLossReportingEnable_t* enable)

Enable or disable path loss threshold reporting for a certain connection.

Parameters

- deviceId – **[in]** Peer device ID.
- enable – **[in]** Enable or disable path loss threshold reporting.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_EnableTransmitPowerReporting(*deviceId_t* deviceId, *bleTxPowerReportingEnable_t* localEnable, *bleTxPowerReportingEnable_t* remoteEnable)

Enable or disable tx power reporting for a certain connection.

Parameters

- deviceId – **[in]** Peer device ID.
- localEnable – **[in]** Enable or disable local tx power reports.
- removeEnable – **[in]** Enable or disable remote tx power reports.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_GenerateDhKeyV2(ecdhPublicKey_t *pRemoteP256PublicKey, *gapPrivateKeyType_t* keyType)

Initiates generation of a Diffie-Hellman key in the Controller for use over the LE transport. Version 2 of LE Generate DHKey command.

Remark

Generated key is contained in gHciLeGenerateDhKeyCompleteEvent_c event.

Parameters

- pRemoteP256PublicKey – **[in]** Pointer to the remote P-256 public key used as input for DH key generation.
- keyType – **[in]** The private key used of DH key generation.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_EattConnectionRequest(*deviceId_t* deviceId, uint16_t mtu, uint8_t cBearers, uint16_t initialCredits, bool_t autoCreditsMgmt)

Open up to 5 Enhanced ATT bearers.

Parameters

- deviceId – **[in]** Peer device id
- mtu – **[in]** MTU
- cBearers – **[in]** Number of bearers
- initialCredits – **[in]** Initial credits
- autoCreditsMgmt – **[in]** EATT will automatically send credits in chunks of initialCredits if TRUE.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_EattConnectionAccept(*deviceId_t* deviceId, bool_t accept, uint16_t localMtu, uint16_t initialCredits, bool_t autoCreditsMgmt)

Accept or reject a received EATT Connection Request.

Parameters

- deviceId – **[in]** Peer device id
- accept – **[in]** TRUE - accept received EattConnectionRequest FALSE - reject incoming EattConnectionRequest
- localMtu – **[in]** Local MTU
- Initial (initialCredits,) – **[in]** credits
- autoCreditsMgmt – **[in]** EATT will automatically send credits in chunks of initialCredits if TRUE.

Return values

- gBleSuccess_c –

- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_EattReconfigureRequest(*deviceId_t* deviceId, uint16_t mtu, uint16_t mps, uint8_t cBearers, *bearerId_t* *pBearers)

Reconfigure the MTU of up to 5 Enhanced ATT bearers.

Parameters

- deviceId – **[in]** Peer device id
- mtu – **[in]** New MTU value to be configured.
- mps – **[in]** New MPS value to be configured. Set to 0 to use current maximum mps value of the channels being reconfigured
- cBearers – **[in]** Number of bearers
- *pBearers – **[in]** Initial credits

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_EattSendCredits(*deviceId_t* deviceId, *bearerId_t* bearerId, uint16_t credits)

Send L2cap credits for Enhanced ATT bearers.

Parameters

- deviceId – **[in]** Peer device id
- bearerId – **[in]** Enhanced ATT bearer id.
- credits – **[in]** Number of credits.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_EattDisconnect(*deviceId_t* deviceId, *bearerId_t* bearerId)

Disconnect the specified bearer.

Parameters

- deviceId – **[in]** Peer device id
- bearerId – **[in]** Enhanced ATT bearer id to be disconnected.

Return values

- gBleSuccess_c –

- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_GetDeviceIdFromConnHandle(uint16_t connHandle, *deviceId_t* *pDeviceId)

Returns the deviceId associated with the received connection handle.

Parameters

- connHandle – **[in]** Connection identifier
- pDeviceId – **[out]** Corresponding device id

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.

bleResult_t Gap_GetConnectionHandleFromDeviceId(*deviceId_t* deviceId, uint16_t *pConnHandle)

Returns the deviceId associated with the received connection handle.

Parameters

- deviceId – **[in]** Connection identifier
- pConnHandle – **[out]** Corresponding connection handle

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.

bleResult_t Gap_GetHostVersion(*gapHostVersion_t* *pOutHostVersion)

Retrieves Host Version information.

Parameters

- pOutHostVersion – **[out]** Pointer to the memory location where the Host Version information should be stored.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.

bleResult_t Gap_ReadRemoteVersionInformation(*deviceId_t* deviceId)

Reads the version information of a peer device.

Parameters

- deviceId – **[in]** Peer device identifier.

Return values

- gBleSuccess_c –
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_GetConnParamsMonitoring(*deviceId_t* deviceId, uint8_t mode)

Get the Connection parameters for the given deviceId in the given mode.

Parameters

- deviceId – **[in]** Peer device Id.
- mode – **[in]** Connection parameters event report mode: bit 0: update the local sequence number for central / peripheral differentiation and send the LE_Handover_Connection_Parameters_Update_Event after remote sequence number change. bit 1: Send LE_Handover_Connection_Parameters_Update_Event when the connection parameters are updated following an LL procedure

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_LeChannelOverride(*bleChannelOverrideMode_t* mode, uint8_t channelListLength, uint8_t *pChannelList)

This function sets the channels to be used for advertising, scanning or initiation.

If the mode is 0x00 (advertising), the specified channels are used by the following advertising commands. The channels used by the ongoing advertising will not change until the advertising is disabled, even if a new channel override command with mode 0x00 is received. This allows different advertising channels for different advertising sets. The mode 0x00 (advertising) can be issued when an advertising is ongoing. The setting will be used by the next advertising enable command.

Remark

In the channel list, the MSB of each byte indicates if the channel is BLE channel index or generic channel index:

- BLE channel index: 0 to 39
- Generic channel index: 0x80+0 to 0x80+127

If the mode is 0x01 or 0x02 (scan or initiation), the specified channels are used by the following scan and initiation. The command with mode 0x01 or 0x02 should be issued when the scan or initiation is not ongoing.

When a generic channel is used, the whitening is initialized by the lower 6 bits of the generic channel index. When a BLE channel is used, the whitening is initialized by the BLE channel index. So the BLE channel 37/38/39 are not strictly the same as the generic channels 42/66/120: they operate on the same frequency but their channel whitening is different.

Parameters

- mode – **[in]** Override the advertising, scanning or initiation channels.
- channelListLength – **[in]** Length of channel list, must be between 1 and 6.
- pChannelList – **[in]** List of channels (see remarks for more details).

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – One or more parameters are invalid.

- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_LeSetSchedulerPriority(uint16_t priorityHandle)

This function sets the priority for one connection in case of several connections by calling the corresponding HCI command.

Parameters

- priorityHandle – **[in]** Parameter that sets the priority for connections

Return values

- gBleSuccess_c –
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_LePeriodicAdvUpdateSync(uint16_t syncHandle, uint16_t skip, uint16_t syncTimeout)

This function updates the synchronization parameters for periodic advertising.

Parameters

- syncHandle – **[in]** Handle identifying the periodic advertising sync.
- skip – **[in]** Number of periodic advertising packets to skip.
- syncTimeout – **[in]** Synchronization timeout value.

Return values

- gBleSuccess_c –
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_LeSetLocalPeripheralLatencyEnable(*deviceId_t* deviceId, bool_t enable)

This function enables or disables local Peripheral latency for a connection.

Parameters

- deviceId – **[in]** Handle identifying the connection.
- enable – **[in]** FALSE to disable, TRUE to enable local Peripheral latency.

Return values

- gBleSuccess_c –
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_LeSetHostFeature(uint8_t bitNumber, bool_t enable)

This function is used by the Host to set or clear a bit controlled by the Host in the Link Layer FeatureSet.

Parameters

- bitNumber – **[in]** Bit position in the FeatureSet
- enable – **[in]** Host feature is enabled/disabled

Return values

- gBleSuccess_c –
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

bleResult_t Gap_EncryptAdvertisingData(const *gapAdvertisingData_t* *pAdvertisingData, const uint8_t *pKey, const uint8_t *pIV, uint8_t *pOutput)

This function is used to encrypt a series of AD structures into an encrypted AD structure.

Parameters

- pAdvertisingData – **[in]** Pointer to AD(s) to be encrypted.

- pKey – **[in]** Pointer to 16-byte key to be used for encryption (must be big endian).
- pIV – **[in]** Pointer to 8-byte initialization vector (must be little endian).
- pOutput – **[out]** Pointer for location of output encrypted data. Must be large enough to accommodate the size of the total data to be encrypted plus the size of the Randomizer and MIC fields.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – At least one of the provided parameters is invalid.
- gBleOutOfMemory_c – Could not allocate memory for the buffers used in CCM.
- gBleRngError_c – Could not generate the randomizer.
- gBleSecLibError_c – CCM call failed.

bleResult_t Gap_DecryptAdvertisingData(uint8_t *pData, uint16_t dataLength, const uint8_t *pKey, const uint8_t *pIV, uint8_t *pOutput)

This function is used to decrypt advertising data.

Parameters

- pData – **[in]** Pointer to encrypted data (AD payload).
- dataLength – **[in]** Length of the encrypted data (includes randomizer and MIC fields).
- pKey – **[in]** Pointer to 16-byte key to be used for decryption (must be big endian).
- pIV – **[in]** Pointer to 8-byte initialization vector (must be little endian).
- pOutput – **[out]** Pointer to the location of decrypted data.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – At least one of the provided parameters is invalid.
- gBleOutOfMemory_c – Could not allocate memory for the buffers used in CCM.
- gBleSecLibError_c – CCM call failed.

void Gap_SetConnectionCallback(*gapConnectionCallback_t* pfConnectionCallback)

Set the GAP connection callback.

Parameters

- pfConnectionCallback – **[in]** The function called when one of the events defined in gapConnectionEventType_t occurs.

Returns

none

void Gap_SetScanningCallback(*gapScanningCallback_t* pfScanningCallback)

Set the GAP scanning callback.

Parameters

- pfScanningCallback – **[in]** The function called when one of the events defined in gapScanningEventType_t occurs.

Returns
none

Gap_AddSecurityModesAndLevels(modeLevelA, modeLevelB)

Macro used to combine two security mode-levels.

Remark

This macro is useful when two different security requirements must be satisfied at the same time, such as a device central security requirement and a service-specific security requirement.

Parameters

- modeLevelB (modeLevelA,) – **[in]** The two security mode-levels.

Returns
The resulting security mode-level.

Gap_CancelInitiatingConnection()

Macro used to cancel a connection initiated by Gap_Connect(...).

Remark

This macro can only be used for a connection that has not yet been established, such as the "gConnEvtConnected_c" has not been received. For example, call this when a connection request has timed out. Application should listen for gCreateConnectionCanceled_c generic event.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – The requested feature is not supported by this stack version.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

Gap_ReadAdvertisingTxPowerLevel()

Macro used to read the radio transmitter power when advertising.

Remark

The result is contained in the gAdvTxPowerLevelRead_c generic event.

Return values

- gBleSuccess_c –
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

Gap_ReadRssi(deviceId)

Macro used to read the RSSI of a radio connection.

Remark

The result is contained in the gConnEvtRssiRead_c connection event. The RSSI value is a signed byte, and the unit is dBm. If the RSSI cannot be read, the gConnEvtPowerReadFailure_c connection event is generated.

Parameters

- deviceId – **[in]** Device ID identifying the radio connection.

Return values

- gBleSuccess_c –
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

Gap_ReadTxPowerLevelInConnection(deviceId)

Macro used to read the radio transmitting power level of a radio connection.

Remark

The result is contained in the gConnEvtTxPowerLevelRead_c connection event. If the TX Power cannot be read, the gConnEvtPowerReadFailure_c connection event is generated.

Parameters

- deviceId – **[in]** Device ID identifying the radio connection.

Return values

- gBleSuccess_c –
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

Gap_GetConnParams(deviceId)

Get the Connection parameters for the given deviceId.

Parameters

- deviceId – **[in]** Peer device Id.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – A parameter has an invalid value or is outside the accepted range.
- gBleOutOfMemory_c – Cannot allocate memory for the Host task.

gCancelOngoingInitiatingConnection_d

Use this value as a parameter to the Gap_Disconnect(deviceId) function to cancel any ongoing connection initiation, for example if the connection has timed out.

gMode_2_Mask_d

Mask to check if a Security Mode-and-Level is Mode 2

getSecurityLevel(modeLevel)

Extracts the security level (see gapSecurityLevel_t) from the combined security mode-level (gapSecurityModeAndLevel_t).

getSecurityMode(modeLevel)

Extracts the security mode (see gapSecurityMode_t) from the combined security mode-level (gapSecurityModeAndLevel_t).

isMode_2(modeLevel)

isMode_1(modeLevel)

isSameMode(modeLevelA, modeLevelB)

addSameSecurityModes(modeLevelA, modeLevelB)

addMode1AndMode2(mode1, mode2)

addDifferentSecurityModes(modeLevelA, modeLevelB)

gDefaultEncryptionKeySize_d

The default (minimum) value for the LTK size

gMaxEncryptionKeySize_d

The maximum value for the LTK size

gGapDefaultDeviceSecurity_d

The default value for the Device Security (no requirements)

gGapDefaultSecurityRequirements_d

The default value for a Security Requirement

gGapAdvertisingIntervalRangeMinimum_c

Minimum advertising interval (20 ms)

gGapAdvertisingIntervalDefault_c

Default advertising interval (1.28 s)

gGapAdvertisingIntervalRangeMaximum_c

Maximum advertising interval (10.24 s)

gGapExtAdvertisingIntervalRangeMinimum_c

Minimum extended advertising interval (20 ms)

gGapExtAdvertisingIntervalDefault_c

Default extended advertising interval (1.28 s)

gGapExtAdvertisingIntervalRangeMaximum_c

Maximum extended advertising interval (10485.76 s)

gGapPeriodicAdvIntervalRangeMinimum_c
: Minimum periodic advertising interval (7.5 ms)

gGapPeriodicAdvIntervalDefault_c
: Default periodic advertising interval (2.56 s)

gGapPeriodicAdvIntervalRangeMaximum_c
: Maximum periodic advertising interval (81.91875 s)

gGapAdvertisingChannelMapDefault_c
: Default Advertising Channel Map - all 3 channels are enabled

gGapDefaultAdvertisingParameters_d
: Default value for Advertising Parameters struct

gGapDefaultExtAdvertisingParameters_d
: Default value for Extended Advertising Parameters struct

gGapDefaultPeriodicAdvParameters_d
: Default value for Periodic Advertising Parameters struct

gGapScanIntervalMin_d
: Minimum scan interval (2.5 ms)

gGapScanIntervalDefault_d
: Default scan interval (10 ms)

gGapScanIntervalMax_d
: Maximum scan interval (10.24 s)

gGapScanWindowMin_d
: Minimum scan window (2.5 ms)

gGapScanWindowDefault_d
: Default scan window (10 ms)

gGapScanWindowMax_d
: Maximum scan window (10.24 s)

gGapRssiMin_d
: Minimum valid value for RSSI (dB)

gGapRssiMax_d
: Maximum valid value for RSSI (dB)

gGapRssiNotAvailable_d
: A special invalid value for the RSSI indicating that the measurement is not available.

gGapScanContinuously_d

Default value for Scanning duration - Scan continuously until explicitly disable

gGapScanPeriodicDisabled_d

Default value for Scanning period - Periodic scanning disabled

gGapDefaultScanningParameters_d

Default value for Scanning Parameters struct

gGapConnIntervalMin_d

Minimum connection interval (7.5 ms)

gGapConnIntervalMax_d

Maximum connection interval (4 s)

gGapConnLatencyMin_d

Minimum connection latency value (0 - no connection event may be ignored)

gGapConnLatencyMax_d

Maximum connection latency value (499 connection events may be ignored)

gGapConnSuperTimeoutMin_d

Minimum supervision timeout (100 ms)

gGapConnSuperTimeoutMax_d

Maximum supervision timeout (32 s)

gGapConnEventLengthMin_d

Minimum value of the connection event length (0 ms)

gGapConnEventLengthMax_d

Maximum value of the connection event length (~41 s)

gGapDefaultConnectionLatency_d

Default connection latency: 0

gGapDefaultSupervisionTimeout_d

Default supervision timeout: 10s

gGapDefaultMinConnectionInterval_d

Default minimum connection interval: 100ms

gGapDefaultMaxConnectionInterval_d

Default maximum connection interval: 200ms

gGapDefaultConnectionRequestParameters_d

The default value for the Connection Request Parameters structure

gGapChSelAlgorithmNo2

"Channel Selection Algorithm #2 is used" value in LE Channel Selection Algorithm Event

gBlePeriodicAdvOngoingSyncCancelHandle

Sync handle value for which to call the create sync cancel command instead of terminate sync

gGapInvalidSyncHandle

Sync handle used to differentiate extended advertising reports from periodic advertising reports

gNone_c

Values of the AD Flags advertising data structure. No information.

gLeLimitedDiscoverableMode_c

This device is in Limited Discoverable mode.

gLeGeneralDiscoverableMode_c

This device is in General Discoverable mode.

gBrEdrNotSupported_c

This device supports only Bluetooth Low Energy; no support for Classic Bluetooth.

gSimultaneousLeBrEdrCapableController_c

This device's Controller also supports Classic Bluetooth.

gSimultaneousLeBrEdrCapableHost_c

This device's Host also supports Classic Bluetooth.

gNoKeys_c

Flags indicating the Keys to be exchanged by the SMP during the key exchange phase of pairing. No key can be distributed.

gLtk_c

Long Term Key.

gIrk_c

Identity Resolving Key.

gCsrk_c

Connection Signature Resolving Key. Deprecated. Not used.

gSecurityMode_1_c

LE Security Mode values for gapSecurityMode_t Mode 1 - Encryption required (except for Level 1).

gSecurityMode_2_c

Mode 2 - Deprecated. Not used.

gSecurityLevel_NoSecurity_c

LE Security Level values for gapSecurityLevel_t No security (combined only with Mode 1).

gSecurityLevel_NoMitmProtection_c

Unauthenticated (no MITM protection).

gSecurityLevel_WithMitmProtection_c

Authenticated (MITM protection by PIN or OOB).

gSecurityLevel_LeSecureConnections_c

Authenticated with LE Secure Connections.

gSecurityMode_1_Level_1_c

Security Mode-and-Level definitions values for gapSecurityModeAndLevel_t Mode 1 Level 1 - No Security.

gSecurityMode_1_Level_2_c

Mode 1 Level 2 - Encryption without authentication.

gSecurityMode_1_Level_3_c

Mode 1 Level 3 - Encryption with authentication.

gSecurityMode_1_Level_4_c

Mode 1 Level 4 - Encryption with LE Secure Connections pairing.

gSecurityMode_2_Level_1_c

Mode 2 Level 1 - Deprecated. Not used.

gSecurityMode_2_Level_2_c

Mode 2 Level 2 - Deprecated. Not used.

gOobNotAvailable_c

Reason for rejecting the pairing request used by gapAuthenticationRejectReason_t. These values are equal to the corresponding reasons from SMP. This device does not have the required OOB for authenticated pairing.

gIncompatibleIoCapabilities_c

The combination of I/O capabilities does not allow pairing with the desired level of security.

gPairingNotSupported_c

This device does not support pairing.

gLowEncryptionKeySize_c

The peer's encryption key size is too low for this device's required security level.

gUnspecifiedReason_c

The host has rejected the pairing for an unknown reason.

gRepeatedAttempts_c

This device is the target of repeated unsuccessful pairing attempts and does not allow further pairing attempts at the moment.

gLinkEncryptionFailed_c

Link could not be encrypted. This reason may not be used by Gap_RejectPairing!

gIoDisplayOnly_c

I/O Capabilities as defined by the SMP used by gapIoCapabilities_t May display a PIN, no input.

gIoDisplayYesNo_c

May display a PIN and has a binary input (e.g., YES and NO buttons).

gIoKeyboardOnly_c

Has keyboard input, no display.

gIoNone_c

No input and no display.

gIoKeyboardDisplay_c

Has keyboard input and display.

gGapCteMinLength_c

AoA/AoD Min CTE length in 8 us units.

gGapCteMaxLength_c

Max CTE length in 8 us units.

gGapCteMinCount_c

Min number of CTEs to transmit in each periodic advertising interval.

gGapCteMaxCount_c

Max number of CTEs to transmit in each periodic advertising interval.

gGapMinSwitchingPatternLength_c

Min number of antennae in the pattern.

gGapMaxSwitchingPatternLength_c

Max number of antennae in the pattern.

gGapEattMaxBearers

Maximum number of bearer Ids in one GAP request/response.

gbmpDIAM_NonConnectableNonScannableUndirected_c

gbmpDIAM_ConnectableUndirected_c

gbmpDIAM_ScannableUndirected_c

struct gapSmpKeys_t

#include <gap_types.h> Structure containing the SMP information exchanged during pairing.

Public Members

uint8_t cLtkSize

Encryption Key Size. If aLtk is NULL, this is ignored.

uint8_t *aLtk

Long Term (Encryption) Key. NULL if LTK is not distributed, else size is given by cLtkSize.

uint8_t *aIrk

Identity Resolving Key. NULL if aIrk is not distributed.

uint8_t *aCsrk

Deprecated - not used.

uint8_t cRandSize

Size of RAND; usually equal to gcMaxRandSize_d. If aLtk is NULL, this is ignored.

uint8_t *aRand

RAND value used to identify the LTK. If aLtk is NULL, this is ignored.

uint16_t ediv

EDIV value used to identify the LTK. If aLtk is NULL, this is ignored.

bleAddressType_t addressType

Public or Random address. If aAddress is NULL, this is ignored.

uint8_t *aAddress

Device Address. NULL if address is not distributed. If aIrk is NULL, this is ignored.

struct gapSecurityRequirements_t

#include <gap_types.h> Security Requirements structure for a Device, a Service or a Characteristic

Public Members

gapSecurityModeAndLevel_t securityModeLevel

Security mode and level.

bool_t authorization

Authorization required.

uint16_t minimumEncryptionKeySize

Minimum encryption key (LTK) size. Ignored if gSecurityMode_1_Level_4_c is required (set to gMaxEncryptionKeySize_d automatically)

struct gapServiceSecurityRequirements_t

#include <gap_types.h> Service Security Requirements

Public Members

uint16_t serviceHandle

Handle of the Service declaration in the GATT Database.

gapSecurityRequirements_t requirements

Requirements for all attributes in this service.

struct gapDeviceSecurityRequirements_t

#include <gap_types.h> Device Security - Security Requirements + Service Security Requirements

Public Members

gapSecurityRequirements_t *pSecurityRequirements

Security requirements added to all services.

uint8_t cNumServices

Number of service-specific requirements; must be less than or equal to gcGapMaxServiceSpecificSecurityRequirements_c.

gapServiceSecurityRequirements_t *aServiceSecurityRequirements

Array of service-specific requirements.

struct gapHandleList_t

#include <gap_types.h> List of Attribute Handles for authorization lists.

Public Members

uint8_t cNumHandles

Number of handles in this list.

uint16_t aHandles[(1U)]

List of handles.

struct gapConnectionSecurityInformation_t

#include <gap_types.h> Connection Security Information structure

Public Members

bool_t authenticated

TRUE if pairing was performed with MITM protection.

gapHandleList_t authorizedToRead

List of handles the peer has been authorized to read.

gapHandleList_t authorizedToWrite

List of handles the peer has been authorized to write.

struct gapPairingParameters_t

#include <gap_types.h> Pairing parameters structure for the Gap_Pair and Gap_AcceptPairingRequest APIs

Public Members

bool_t withBonding

TRUE if this device is able to and wants to bond after pairing, FALSE otherwise.

gapSecurityModeAndLevel_t securityModeAndLevel

The desired security mode-level.

uint8_t maxEncryptionKeySize

Maximum LTK size supported by the device.

gapIoCapabilities_t localIoCapabilities

I/O capabilities used to determine the pairing method.

bool_t oobAvailable

TRUE if this device has Out-of-Band data that can be used for authenticated pairing. FALSE otherwise.

gapSmpKeyFlags_t centralKeys

Indicates the SMP keys to be distributed by the Central.

gapSmpKeyFlags_t peripheralKeys

Indicates the SMP keys to be distributed by the Peripheral.

bool_t leSecureConnectionSupported

Indicates if device supports LE Secure Connections pairing. Conflict if this is FALSE and securityModeAndLevel is gSecurityMode_1_Level_4_c.

bool_t useKeypressNotifications

Indicates if device supports Keypress Notification PDUs during Passkey Entry pairing. Conflict if this is TRUE and localIoCapabilities is set to gIoNone_c.

struct gapPeripheralSecurityRequestParameters_t

#include <gap_types.h> Parameters of a Peripheral Security Request.

Public Members

bool_t bondAfterPairing

TRUE if the Peripheral supports bonding.

bool_t authenticationRequired

TRUE if the Peripheral requires authentication for MITM protection.

struct gapAdvertisingParameters_t

#include <gap_types.h> Advertising Parameters; for defaults see gGapDefaultAdvertising-Parameters_d.

Public Members

uint16_t minInterval

Minimum desired advertising interval. Default: 1.28 s.

uint16_t maxInterval

Maximum desired advertising interval. Default: 1.28 s.

bleAdvertisingType_t advertisingType

Advertising type. Default: connectable undirected.

bleAddressType_t ownAddressType

Indicates whether the advertising address is the public address (BD_ADDR) or the random address (set by Gap_SetRandomAddress). Default: public address. If Controller Privacy is enabled, this parameter is irrelevant as Private Resolvable Addresses are always used.

bleAddressType_t peerAddressType

Address type of the peer; only used in directed advertising and Enhanced Privacy.

bleDeviceAddress_t peerAddress

Address of the peer; same as above.

gapAdvertisingChannelMapFlags_t channelMap

Bit mask indicating which of the three advertising channels are used. Default: all three.

gapAdvertisingFilterPolicy_t filterPolicy

Indicates whether the connect and scan requests are filtered using the Filter Accept List. Default: does not use Filter Accept List (process all).

struct gapExtAdvertisingParameters_tag

#include <gap_types.h> Extended Advertising Parameters; for defaults see gGapDefaultExtAdvertisingParameters_d.

Public Members

uint8_t SID

ID of the advertising set chosen by application. Shall be lower than gBleExtAdvMaxSetId_c

uint8_t handle

ID of the advertising set handled by controller. Shall be lower than gNumOfSupportedAdvSets

uint32_t minInterval

Minimum desired advertising interval. Shall be at least equal or higher than gGapExtAdvertisingIntervalRangeMinimum_c

uint32_t maxInterval

Maximum desired advertising interval. Shall be higher than gGapExtAdvertisingIntervalRangeMinimum_c and higher than minInterval

bleAddressType_t ownAddressType

Indicates whether the advertising address is the public address (BD_ADDR) or the random address (set by Gap_SetRandomAddress). Default: public address. If Controller Privacy is enabled, this parameter is irrelevant as Private Resolvable Addresses are always used.

bleDeviceAddress_t ownRandomAddr

The random address used for advertising on the current handle

bleAddressType_t peerAddressType

Address type of the peer; only used in directed advertising and Enhanced Privacy.

bleDeviceAddress_t peerAddress

Address of the peer; same as above.

gapAdvertisingChannelMapFlags_t channelMap

Bit mask indicating which of the three advertising channels are used for primary advertising

gapAdvertisingFilterPolicy_t filterPolicy

Indicates whether the connect and scan requests are filtered using the Filter Accept List

bleAdvRequestProperties_t extAdvProperties

Type of advertising event

int8_t txPower

The maximum power level at which the adv packets are to be transmitted. The Controller shall choose a power level lower than or equal to the one specified by the Host. Valid range: -127 to 20

gapLePhyMode_t primaryPHY

The PHY on which the advertising packets are transmitted (1M or Coded PHY). Used for sending ADV_EXT_IND

gapLePhyMode_t secondaryPHY

The PHY used for sending AUX_ADV_IND PDU. Used only for Extended Advertising Events

uint8_t secondaryAdvMaxSkip

Maximum number of advertising events that can be skipped before the AUX_ADV_IND can be sent

bool_t enableScanReqNotification

Indicates whether the Controller shall send notifications upon the receipt of a scan request PDU

gapLePhyOptionsFlags_t primaryAdvPhyOptions

Preferred or required coding when transmitting on the primary LE Coded PHY

gapLePhyOptionsFlags_t secondaryAdvPhyOptions

Preferred or required coding when transmitting on the secondary LE Coded PHY

struct gapPeriodicAdvParameters_tag

#include <gap_types.h> Periodic Advertising Parameters [v2]; for defaults see gGapDefaultPeriodicAdvParameters_d.

Public Members

uint8_t handle

ID of the advertising set handled by controller. Shall be lower than gNumOfSupportedAdvSets

bool_t addTxPowerInAdv

Set this option to include the Tx power in advertising packet.

uint16_t minInterval

Minimum advertising interval for periodic advertising.

uint16_t maxInterval

Maximum advertising interval for periodic advertising. Should be different and higher than minInterval.

uint8_t numSubevents

Number of subevents.

uint8_t subeventInterval

Interval between subevents.

uint8_t responseSlotDelay
Time between the advertising packet in a subevent and the first response slot.

uint8_t responseSlotSpacing
Time between response slots.

uint8_t numResponseSlots
Number of response slots.

struct gapPeriodicAdvSyncTransfer_tag
#include <gap_types.h>

Public Members

deviceId_t deviceId
Connection identifier

uint16_t serviceData
Value provided by the host

uint16_t syncHandle
Identifier for the periodic advertising train

struct gapPeriodicAdvSetInfoTransfer_tag
#include <gap_types.h>

Public Members

deviceId_t deviceId
Connection identifier

uint16_t serviceData
Value provided by the host

uint16_t advHandle
Identifier for the advertising set

struct gapSetPeriodicAdvSyncTransferParams_tag
#include <gap_types.h>

Public Members

deviceId_t deviceId
Connection identifier

gapPeriodicAdvSyncMode_t mode

The action to be taken when periodic advertising synchronization information is received

uint16_t skip

The number of periodic advertising packets that can be skipped after a successful receive

uint16_t syncTimeout

Synchronization timeout for the periodic advertising train

bleSyncCteType_t CTEType

0: Do not sync to packets with an AoA Constant Tone Extension 1: Do not sync to packets with an AoD Constant Tone Extension with 1 us slots 2: Do not sync to packets with an AoD Constant Tone Extension with 2 us slots 4: Do not sync to packets without a Constant Tone Extension

struct gapScanningParameters_t

#include <gap_types.h> Scanning parameters; for defaults see gGapDefaultScanningParameters_d.

Public Members

bleScanType_t type

Scanning type. Default: passive.

uint16_t interval

Scanning interval. Default: 10 ms.

uint16_t window

Scanning window. Default: 10 ms.

bleAddressType_t ownAddressType

Indicates whether the address used in scan requests is the public address (BD_ADDR) or the random address (set by Gap_SetRandomAddress). Default: public address. If Controller Privacy is enabled, this parameter is irrelevant as Private Resolvable Addresses are always used.

bleScanningFilterPolicy_t filterPolicy

Indicates whether the advertising packets are filtered using the Filter Accept List. Default: does not use Filter Accept List (scan all).

gapLePhyFlags_t scanningPHYs

Indicates the PHYs on which the advertising packets should be received on the primary advertising channel.

struct gapCreateSyncReqOptions_tag

#include <gap_types.h> Create Sync Request Options

struct gapPeriodicAdvSyncReq_tag

#include <gap_types.h> Periodic Advertising Sync Request parameters

Public Members

gapCreateSyncReqOptions_t options

Bit 0: Filter Policy. Bit 1: Reporting enabled/disabled. Bit 2: Duplicate filtering disabled/enabled.

uint8_t SID

The SID advertised by the periodic advertiser in the ADI field.

bleAddressType_t peerAddressType

Periodic advertiser's address type (Public or Random)

bleDeviceAddress_t peerAddress

Periodic advertiser's address

uint16_t skipCount

The number of consecutive periodic advertising packets that the receiver may skip after successfully receiving a periodic advertising packet.

uint16_t timeout

The maximum permitted time between successful receives. If this time is exceeded, synchronization is lost.

bleSyncCteType_t cteType

Specifies whether to only sync to certain types of CTE

struct gapConnectionRequestParameters_t

#include <gap_types.h> Connection request parameter structure to be used in the Gap_Connect function; for API-defined defaults, use gGapDefaultConnectionRequestParameters_d.

Public Members

uint16_t scanInterval

Scanning interval. Default: 10 ms.

uint16_t scanWindow

Scanning window. Default: 10 ms.

bleInitiatorFilterPolicy_t filterPolicy

Indicates whether the connection request is issued for a specific device or for all the devices in the Filter Accept List. Default: specific device.

bleAddressType_t ownAddressType

Indicates whether the address used in connection requests is the public address (BD_ADDR) or the random address (set by Gap_SetRandomAddress). Default: public address.

bleAddressType_t peerAddressType

When connecting to a specific device (see filterPolicy), this indicates that device's address type. Default: public address.

bleDeviceAddress_t peerAddress

When connecting to a specific device (see filterPolicy), this indicates that device's address.

uint16_t connIntervalMin

The minimum desired connection interval. Default: 100 ms.

uint16_t connIntervalMax

The maximum desired connection interval. Default: 200 ms.

uint16_t connLatency

The desired connection latency (the maximum number of consecutive connection events the Peripheral is allowed to ignore). Default: 0.

uint16_t supervisionTimeout

The maximum time interval between consecutive over-the-air packets; if this timer expires, the connection is dropped. Default: 10 s.

uint16_t connEventLengthMin

The minimum desired connection event length. Default: 0 ms.

uint16_t connEventLengthMax

The maximum desired connection event length. Default: maximum possible, ~41 s. (lets the Controller decide).

bool_t usePeerIdentityAddress

If Controller Privacy is enabled and this parameter is TRUE, the address defined in the peerAddressType and peerAddress is an identity address. Otherwise, it is a device address.

gapLePhyFlags_t initiatingPHYs

Indicates the PHY on which the advertising packets should be received on the primary advertising channel and the PHY for which connection parameters have been specified.

struct gapConnectionFromPawrParameters_t

#include <gap_types.h> Connection request parameter structure to be used in the Gap_ConnectFromPawr function

Public Members

uint16_t scanInterval

Scanning interval. Default: 10 ms.

uint16_t scanWindow

Scanning window. Default: 10 ms.

bleInitiatorFilterPolicy_t filterPolicy

Indicates whether the connection request is issued for a specific device or for all the devices in the Filter Accept List. Default: specific device.

bleAddressType_t ownAddressType

Indicates whether the address used in connection requests is the public address (BD_ADDR) or the random address (set by Gap_SetRandomAddress). Default: public address.

bleAddressType_t peerAddressType

When connecting to a specific device (see filterPolicy), this indicates that device's address type. Default: public address.

bleDeviceAddress_t peerAddress

When connecting to a specific device (see filterPolicy), this indicates that device's address.

uint16_t connIntervalMin

The minimum desired connection interval. Default: 100 ms.

uint16_t connIntervalMax

The maximum desired connection interval. Default: 200 ms.

uint16_t connLatency

The desired connection latency (the maximum number of consecutive connection events the Peripheral is allowed to ignore). Default: 0.

uint16_t supervisionTimeout

The maximum time interval between consecutive over-the-air packets; if this timer expires, the connection is dropped. Default: 10 s.

uint16_t connEventLengthMin

The minimum desired connection event length. Default: 0 ms.

uint16_t connEventLengthMax

The maximum desired connection event length. Default: maximum possible, ~41 s. (lets the Controller decide).

bool_t usePeerIdentityAddress

If Controller Privacy is enabled and this parameter is TRUE, the address defined in the peerAddressType and peerAddress is an identity address. Otherwise, it is a device address.

gapLePhyFlags_t initiatingPHYs

Indicates the PHY on which the advertising packets should be received on the primary advertising channel and the PHY for which connection parameters have been specified.

uint8_t advHandle

Advertising handle identifying the periodic advertising train.

uint8_t subevent

Subevent where the connection request is to be sent.

struct gapConnectionParameters_t

#include <gap_types.h> Connection parameters as received in the gConnEvtConnected_c connection event.

Public Members

uint16_t connInterval

Interval between connection events.

uint16_t connLatency

Number of consecutive connection events the Peripheral may ignore.

uint16_t supervisionTimeout

The maximum time interval between consecutive over-the-air packets; if this timer expires, the connection is dropped.

bleCentralClockAccuracy_t centralClockAccuracy

Accuracy of central's clock, allowing for frame detection optimizations.

struct gapConnectionSubrateParameters_t

#include <gap_types.h> Connection subrate parameters

Public Members

uint16_t subrateMin

Minimum subrate factor to be applied to the underlying connection interval.

uint16_t subrateMax

Maximum subrate factor to be applied to the underlying connection interval.

uint16_t latencyMax

Maximum Peripheral latency for the connection in units of subrated connection intervals.

uint16_t continuationNumber

Minimum number of underlying connection events to remain active after a packet containing a Link Layer PDU with a non-zero Length field is sent or received.

uint16_t supervisionTimeout

The maximum time interval between consecutive over-the-air packets in 10ms units; if this timer expires, the connection is dropped.

struct gapGenerateDHKeyV2Params_t

#include <gap_types.h> DH Key Generate V2

Public Members

ecdhPublicKey_t remoteP256PublicKey

The remote P-256 public key

gapPrivateKeyType_t keyType

The private key type used for generating the DH key

struct gapConnectionlessCteTransmitParams_t

#include <gap_types.h> Parameter structure to be used in the Gap_SetConnectionlessCteTransmitParameters function.

Public Members

uint8_t handle

Advertising set handle.

uint8_t cteLength

Constant Tone Extension length in 8 us.

bleCteType_t cteType

Type of CTE.

uint8_t cteCount

Number of CTEs to transmit in each periodic advertising interval.

uint8_t switchingPatternLength

Number of Antenna IDs in pattern.

uint8_t aAntennaIds[1]

List of Antenna IDs in pattern.

struct gapConnectionlessIqSamplingParams_t

#include <gap_types.h> Parameter structure to be used in the Gap_EnableConnectionlessIqSampling function.

Public Members

bleIqSamplingEnable_t iqSamplingEnable
Enable or disable IQ sampling.

bleSlotDurations_t slotDurations
Switching and sampling slot durations.

uint8_t maxSampledCtes
Maximum number of CTEs to sample and report in each periodic adv interval.

uint8_t switchingPatternLength
Number of Antenna IDs in pattern.

uint8_t aAntennaIds[1]
List of Antenna IDs in pattern.

struct gapConnectionCteTransmitParams_t
#include <gap_types.h> Parameter structure to be used in the Gap_SetConnectionCteTransmitParameters function.

Public Members

bleCteAllowedTypesMap_t cteTypes
Allowed CTE types.

uint8_t switchingPatternLength
Number of Antenna IDs in pattern.

uint8_t aAntennaIds[1]
List of Antenna IDs in pattern.

struct gapConnectionCteReceiveParams_t
#include <gap_types.h> Parameter structure to be used in the Gap_SetConnectionCteReceiveParameters function.

Public Members

bleIqSamplingEnable_t iqSamplingEnable
Enable or disable IQ sampling.

bleSlotDurations_t slotDurations
Switching and sampling slot durations.

uint8_t switchingPatternLength
Number of Antenna IDs in pattern.

uint8_t aAntennaIds[1]
List of Antenna IDs in pattern.

struct gapConnectionCteReqEnableParams_t

#include <gap_types.h> Parameter structure to be used in the Gap_EnableConnectionCteRequest function.

Public Members

bleCteReqEnable_t cteReqEnable

Enable or disable CTE Req procedure.

uint16_t cteReqInterval

Requested interval for initiating CTE Req procedure.

uint8_t requestedCteLength

Minimum length of CTEs requested in 8 us units.

bleCteType_t requestedCteType

Requested types of CTE.

struct gapPathLossReportingParams_t

#include <gap_types.h> Parameter structure to be used in the Gap_SetPathLossReportingParameters function.

Public Members

uint8_t highThreshold

High threshold for the path loss. Units: dB.

uint8_t highHysteresis

Hysteresis value for the high threshold. Units: dB.

uint8_t lowThreshold

Low threshold for the path loss. Units: dB.

uint8_t lowHysteresis

Hysteresis value for the low threshold. Units: dB.

uint16_t minTimeSpent

Minimum time in number of connection events to be observed once the path crosses the threshold before an event is generated.

struct gapAdStructure_t

#include <gap_types.h> Definition of an AD Structure as contained in Advertising and Scan Response packets. An Advertising or Scan Response packet contains several AD Structures.

Public Members

uint8_t length

Total length of the [adType + aData] fields. Equal to 1 + lengthOf(aData).

gapAdType_t adType

AD Type of this AD Structure.

uint8_t *aData

Data contained in this AD Structure; length of this array is equal to (*gapAdStructure_t.length* - 1).

struct gapAdvertisingData_t

#include <gap_types.h> Advertising Data structure : a list of several *gapAdStructure_t* structures.

Public Members

uint8_t cNumAdStructures

Number of AD Structures.

gapAdStructure_t *aAdStructures

Array of AD Structures.

struct gapAdvertisingDecisionData_t

#include <gap_types.h> Extended Advertising Decision Data structure.

Public Members

uint8_t *pKey

Pointer to the key for the resolvable tag. When pKey != NULL Decision data should contain only arbitrary data or pDecisionData should be NULL.

uint8_t *pPrand

Used only when pKey != NULL. When pPrand != NULL it should point the 3 bytes value of the prand. Otherwise it will be generated by the host.

uint8_t *pDecisionData

Pointer to the decision data. When pDecisionData != NULL decision data may contain the resolvable tag and/or arbitrary data.

uint8_t dataLength

The length of the decision data pointed by pDecisionData. When pKey is NULL it should not exceed 8. Otherwise it should not exceed 2

bool_t resolvableTagPresent

When pKey is NULL it indicates whether decision data contains the resolvable tag. In this case the 6 <= dataLength <= 8

struct gapDecisionInstructionsData_tag

#include <gap_types.h>

struct gapSubeventDataStructure_t

#include <gap_types.h> Definition of a Subevent Data structure as contained in Periodic Advertising subevents.

Public Members

uint8_t subevent

Subevent index for this data.

uint8_t responseSlotStart

The first response slots to be used in this subevent.

uint8_t responseSlotCount

The number of response slots to be used.

gapAdvertisingData_t *pAdvertisingData

Advertising data.

struct gapPeriodicAdvertisingSubeventData_t

#include <gap_types.h> Advertising Data structure : a list of several *gapAdStructure_t* structures.

Public Members

uint8_t cNumSubevents

Number of subevents.

gapSubeventDataStructure_t *aSubeventDataStructures

Data for each subevent.

struct gapPeriodicAdvertisingResponseData_t

#include <gap_types.h> Periodic Advertising Response data

Public Members

uint16_t requestEvent

The value of paEventCounter (see [Vol 6] Part B, Section 4.4.2.1) for the periodic advertising packet that the Host is responding to

uint8_t requestSubevent

The subevent for the periodic advertising packet that the Host is responding to.

uint8_t responseSubevent
Used to identify the subevent of the PAwR train.

uint8_t responseSlot
Used to identify the response slot of the PAwR train.

gapAdvertisingData_t *pResponseData
Response data.

struct gapPeriodicSyncSubeventParameters_t
#include <gap_types.h> Periodic Sync Subevent Parameters

Public Members

uint16_t perAdvProperties
Bit 6: Include TxPower in the advertising PDU

uint8_t numSubevents
Number of subevents.

uint8_t aSubevents[1]
List of subevents

struct gapExtScanNotification_t
#include <gap_types.h>

Public Members

uint8_t handle
Advertising Handle

bleAddressType_t scannerAddrType
Scanner device's address type.

bleDeviceAddress_t aScannerAddr
Scanner device's address.

bool_t scannerAddrResolved
Whether the address corresponds to Resolved Private Address.

struct gapAdvertisingSetTerminated_t
#include <gap_types.h>

Public Members

bleResult_t status
Status of advertising set termination

uint8_t handle
Advertising Handle

deviceId_t deviceId
Valid only if the advertising ended with a connection

uint8_t numCompletedExtAdvEvents
Number of advertising events sent by Controller

struct gapPerAdvSubeventDataRequest_t
#include <gap_types.h>

Public Members

uint8_t handle
Used to identify a periodic advertising train.

uint8_t subeventStart
The first subevent that data is requested for.

uint8_t subeventDataCount
The number of subevents that data is requested for.

struct gapPerAdvResponse_t
#include <gap_types.h>

struct gapAdvertisingEvent_t
#include <gap_types.h> Advertising event structure: type + data.

Public Members

gapAdvertisingEventType_t eventType
Event type.

union *gapAdvertisingEvent_t* eventData
Event data, to be interpreted according to *gapAdvertisingEvent_t.eventType*.

struct gapScannedDevice_t
#include <gap_types.h> Scanned device information structure, obtained from LE Advertising Reports.

Public Members

bleAddressType_t addressType

Device's advertising address type.

bleDeviceAddress_t aAddress

Device's advertising address.

int8_t rssi

RSSI on the advertising channel; may be compared to the TX power contained in the AD Structure of type gAdTxPowerLevel_c to estimate distance from the advertiser.

uint8_t dataLength

Length of the advertising or scan response data.

uint8_t *data

Advertising or scan response data.

bleAdvertisingReportEventType_t advEventType

Advertising report type, indicating what type of event generated this data (advertising, scan response).

bool_t directRpaUsed

TRUE if directed advertising with Resolvable Private Address as Direct Address was detected while Enhanced Privacy is enabled.

bleDeviceAddress_t directRpa

Resolvable Private Address set as Direct Address for directed advertising. Valid only when directRpaUsed is TRUE.

bool_t advertisingAddressResolved

If this is TRUE, the address contained in the addressType and aAddress fields is the identity address of a resolved RPA from the Advertising Address field. Otherwise, the address from the respective fields is the public or random device address contained in the Advertising Address field.

struct gapExtScannedDevice_t

#include <gap_types.h>

Public Members

bleAddressType_t addressType

Device's advertising address type.

bleDeviceAddress_t aAddress

Device's advertising address.

uint8_t SID

Advertising set id

bool_t advertisingAddressResolved

If this is TRUE, the address contained in the addressType and aAddress fields is the identity address of a resolved RPA from the Advertising Address field. Otherwise, the address from the respective fields is the public or random device address contained in the Advertising Address field.

bleAdvReportEventProperties_t advEventProperties

Advertising report properties, indicating what type of event generated this data (advertising, scan response).

int8_t rssi

RSSI on the advertising channel; may be compared to the TX power contained in the AD Structure of type gAdTxPowerLevel_c to estimate distance from the advertiser.

int8_t txPower

The Tx power level of the advertiser

uint8_t primaryPHY

Advertiser PHY for primary channel

uint8_t secondaryPHY

Advertiser PHY for secondary channel

uint16_t periodicAdvInterval

Interval of the periodic advertising. Zero if not periodic advertising.

bool_t directRpaUsed

TRUE if directed advertising with Resolvable Private Address as Direct Address was detected while Enhanced Privacy is enabled.

bleAddressType_t directRpaType

Address type for directed advertising. Valid only when directRpaUsed is TRUE.

bleDeviceAddress_t directRpa

Resolvable Private Address set as Direct Address for directed advertising. Valid only when directRpaUsed is TRUE.

uint16_t dataLength

Length of the advertising or scan response data.

uint8_t *pData

Advertising or scan response data.

struct gapPeriodicScannedDevice_t

#include <gap_types.h>

Public Members

uint16_t syncHandle

Sync Handle

int8_t txPower

The Tx power level of the advertiser

int8_t rssi

RSSI on the advertising channel; may be compared to the TX power contained in the AD Structure of type gAdTxPowerLevel_c to estimate distance from the advertiser.

bleCteType_t cteType

Type of Constant Tone Extension in the periodic advertising packets.

uint16_t dataLength

Length of the advertising or scan response data.

uint8_t *pData

Advertising or scan response data.

struct gapPeriodicScannedDeviceV2_t

#include <gap_types.h>

Public Members

uint16_t syncHandle

Sync Handle

int8_t txPower

The Tx power level of the advertiser

int8_t rssi

RSSI on the advertising channel; may be compared to the TX power contained in the AD Structure of type gAdTxPowerLevel_c to estimate distance from the advertiser.

bleCteType_t cteType

Type of Constant Tone Extension in the periodic advertising packets.

uint16_t periodicEventCounter

The value of paEventCounter (see [Vol 6] Part B, Section 4.4.2.1) for the reported periodic advertising packet.

uint8_t subevent

Subevent number.

uint16_t dataLength

Length of the advertising or scan response data.

uint8_t *pData
Advertising or scan response data.

struct gapSyncEstbEventData_t
#include <gap_types.h>

Public Members

bleResult_t status
Status of the Sync Established Event

uint16_t syncHandle
Sync Handle

uint8_t SID
Value of the Advertising SID subfield in the ADI field of the PDU

bleAddressType_t peerAddressType
Advertiser Address Type

bleDeviceAddress_t peerAddress
Advertiser Address

gapLePhyMode_t PHY
Advertiser PHY

uint16_t periodicAdvInterval
Periodic advertising interval

bleAdvertiserClockAccuracy_t advertiserClockAccuracy
Advertiser Clock Accuracy

uint8_t numSubevents
Only relevant for v2 of the event, otherwise value is 0

uint8_t subeventInterval
Only relevant for v2 of the event, otherwise value is 0

uint8_t responseSlotDelay
Only relevant for v2 of the event, otherwise value is 0

uint8_t responseSlotSpacing
Only relevant for v2 of the event, otherwise value is 0

struct gapSyncLostEventData_t
#include <gap_types.h>

Public Members

uint16_t syncHandle
Sync Handle

struct gapConnectionlessIqReport_t
#include <gap_types.h>

Public Members

uint16_t syncHandle
Sync Handle.

uint8_t channelIndex
Index of the channel on which the packet was received.

int16_t rssi
RSSI of the packet.

uint8_t rssiAntennaId
Antenna ID.

bleCteType_t cteType
Type of CTE on the received packet (AoA, AoD 1 us, AoD 2 us).

bleSlotDurations_t slotDurations
Durations of switching/sampling slots.

bleIqReportPacketStatus_t packetStatus
Packet status, information about CRC.

uint16_t periodicEventCounter
The value of paEventCounter for the reported AUX_SYNC_IND PDU.

uint8_t sampleCount
Total number of sample pairs.

int8_t *aI_samples
List of sampleCount I_samples.

int8_t *aQ_samples
List of sampleCount Q_samples.

struct gapMonAdvReportReport_t
#include <gap_types.h>

Public Members

bleAddressType_t peerAddressType
Advertiser Address Type

bleDeviceAddress_t peerAddress
Advertiser Address

bleMonAdvCondition_t condition
Condition parameter

struct gapScanningEvent_t
#include <gap_types.h> Scanning event structure: type + data.

Public Members

gapScanningEventType_t eventType
Event type.

union *gapScanningEvent_t* eventData
Event data, to be interpreted according to *gapScanningEvent_t.eventType*.

struct gapConnectedEvent_t
#include <gap_types.h> Event data structure for the gConnEvtConnected_c event.

Public Members

gapConnectionParameters_t connParameters
Connection parameters established by the Controller.

bleAddressType_t peerAddressType
Connected device's address type.

bleDeviceAddress_t peerAddress
Connected device's address.

bool_t peerRpaResolved
If this is TRUE, the address defined by peerAddressType and peerAddress is the identity address of the peer, and the peer used an RPA that was resolved by the Controller and is contained in the peerRpa field. Otherwise, it is a device address. This parameter is irrelevant if Controller Privacy is not enabled.

bleDeviceAddress_t peerRpa
Peer Resolvable Private Address if Controller Privacy is active and peerRpaResolved is TRUE.

bool_t localRpaUsed

If this is TRUE, the Controller has used an RPA contained in the localRpa field. This parameter is irrelevant if Controller Privacy is not enabled.

bleDeviceAddress_t localRpa

Local Resolvable Private Address if Controller Privacy is active and localRpaUsed is TRUE.

bleLlConnectionRole_t connectionRole

Connection Role - central or peripheral.

uint8_t advHandle

Only relevant if the connection was from PAWR.

uint16_t syncHandle

Only relevant if the connection was from PAWR.

struct gapKeyExchangeRequestEvent_t

#include <gap_types.h> Event data structure for the gConnEvtKeyExchangeRequest_c event.

Public Members

gapSmpKeyFlags_t requestedKeys

Mask identifying the keys being requested.

uint8_t requestedLtkSize

Requested size of the encryption key.

struct gapKeysReceivedEvent_t

#include <gap_types.h> Event data structure for the gConnEvtKeysReceived_c event.

Public Members

gapSmpKeys_t *pKeys

The SMP keys distributed by the peer.

struct gapAuthenticationRejectedEvent_t

#include <gap_types.h> Event data structure for the gConnEvtAuthenticationRejected_c event.

Public Members

gapAuthenticationRejectReason_t rejectReason

Peripheral's reason for rejecting the authentication.

struct gapPairingCompleteEvent_t

#include <gap_types.h> Event data structure for the gConnEvtPairingComplete_c event.

Public Members

bool_t pairingSuccessful

TRUE if pairing succeeded, FALSE otherwise.

union *gapPairingCompleteEvent_t* pairingCompleteData

Information of completion, selected upon the value of *gapPairingCompleteEvent_t.pairingSuccessful*.

struct gapLongTermKeyRequestEvent_t

#include <gap_types.h> Event data structure for the gConnEvtLongTermKeyRequest_c event.

Public Members

uint16_t ediv

The Encryption Diversifier, as defined by the SMP.

uint8_t aRand[(8U)]

The Random number, as defined by the SMP.

uint8_t randSize

Usually equal to gcMaxRandSize_d.

struct gapEncryptionChangedEvent_t

#include <gap_types.h> Event data structure for the gConnEvtEncryptionChanged_c event.

Public Members

bool_t newEncryptionState

TRUE if link has been encrypted, FALSE if encryption was paused or removed.

struct gapDisconnectedEvent_t

#include <gap_types.h> Event data structure for the gConnEvtDisconnected_c event.

Public Members

gapDisconnectionReason_t reason

Reason for disconnection.

struct gapConnParamsUpdateReq_t

#include <gap_types.h> Event data structure for the gConnEvtParameterUpdateRequest_c event.

Public Members

uint16_t intervalMin

Minimum interval between connection events.

uint16_t intervalMax

Maximum interval between connection events.

uint16_t peripheralLatency

Number of consecutive connection events the Peripheral may ignore.

uint16_t timeoutMultiplier

The maximum time interval between consecutive over-the-air packets; if this timer expires, the connection is dropped.

struct gapConnParamsUpdateComplete_t

#include <gap_types.h> Event data structure for the gConnEvtParameterUpdateComplete_c event.

Public Members

uint16_t connInterval

Interval between connection events.

uint16_t connLatency

Number of consecutive connection events the Peripheral may ignore.

uint16_t supervisionTimeout

The maximum time interval between consecutive over-the-air packets; if this timer expires, the connection is dropped.

struct gapConnLeDataLengthChanged_t

#include <gap_types.h> Event data structure for the gConnEvtLeDataLengthChanged_c event.

Public Members

uint16_t maxTxOctets

The maximum number of payload octets in a Link Layer Data Channel PDU to transmit on this connection.

uint16_t maxTxTime

The maximum time that the local Controller will take to send a Link Layer Data Channel PDU on this connection.

uint16_t maxRxOctets

The maximum number of payload octets in a Link Layer Data Channel PDU to receive on this connection.

uint16_t maxRxTime

The maximum time that the local Controller will take to receive a Link Layer Data Channel PDU on this connection.

struct gapConnIqReport_t

#include <gap_types.h> Event data structure for the gConnEvtIqReportReceived_c event.

Public Members

gapLePhyMode_t rxPhy

Identifies receiver PHY for this connection.

uint8_t dataChannelIndex

Index of the data channel on which the Data Physical Channel PDU was received.

int16_t rssi

RSSI of the packet.

uint8_t rssiAntennaId

Antenna ID.

bleCteType_t cteType

Type of CTE on the received packet (AoA, AoD 1 us, AoD 2 us).

bleSlotDurations_t slotDurations

Durations of switching/sampling slots.

bleIqReportPacketStatus_t packetStatus

Packet status, information about CRC.

uint16_t connEventCounter

The value of the connEventCounter for the reported PDU.

uint8_t sampleCount

Total number of sample pairs.

int8_t *aI_samples

List of sampleCount I_samples

int8_t *aQ_samples

List of sampleCount Q_samples

struct gapConnCteRequestFailed_t

#include <gap_types.h> Event data structure for the gConnEvtCteRequestFailed_c event.

Public Members

bleResult_t status

Status (0x00 - received response but without CTE, 0x01-0xFF - peer rejected request

struct gapPathLossThresholdEvent_t

#include <gap_types.h> Event data structure for the gConnEvtPathLossThreshold_c event.

Public Members

uint8_t currentPathLoss

Current path loss. Units: dB.

blePathLossThresholdZoneEntered_t zoneEntered

Low, middle or high.

struct gapTransmitPowerReporting_t

#include <gap_types.h> Event data structure for the gConnEvtTransmitPowerReporting_c event.

Public Members

bleTxPowerReportingReason_t reason

Reason for generating this event

- local tx power changed
- remote tx power changed
- a HCI_LE_Read_Remote_Transmit_Power_Level command completed

blePowerControlPhyType_t phy

The PHY for which the event is generated.

int8_t txPowerLevel

New transmit power level on this connection and PHY.

bleTxPowerLevelFlags_t flags

Tx power level flags.

int8_t delta

Change in tx power level. Units: dB.

struct gapSubrateChangeEvent_t

#include <gap_types.h> Event data structure for the gConnEvtLeSubrateChange_c event.

Public Members

uint16_t subrateFactor

subrate factor applied to the underlying connection interval.

uint16_t peripheralLatency

Peripheral latency for the connection in units of subrated connection intervals.

uint16_t continuationNumber

Minimum number of underlying connection events to remain active after a packet containing a Link Layer PDU with a non-zero Length field is sent or received.

uint16_t supervisionTimeout

The maximum time interval between consecutive over-the-air packets in 10ms units; if this timer expires, the connection is dropped.

struct gapTransmitPowerInfo_t

#include <gap_types.h> Event data structure for the gConnEvtEnhancedReadTransmitPowerLevel_c event.

Public Members

blePowerControlPhyType_t phy

The PHY for which the event is generated.

int8_t currTxPowerLevel

Current tx power level on this connection and PHY.

int8_t maxTxPowerLevel

Maximum tx power level on this connection and PHY.

struct gapEattConnectionRequest_t

#include <gap_types.h> Event data structure for the gConnEvtEattConnectionRequest_c event.

struct gapEattConnectionComplete_t

#include <gap_types.h> Event data structure for the gConnEvtEattConnectionComplete_c event.

struct gapEattReconfigureResponse_t

#include <gap_types.h> Event data structure for the gConnEvtEattChannelReconfigureResponse_c event.

struct gapEattBearerStatusNotification_t

#include <gap_types.h> Event data structure for the bearer status update event.

struct gapHandoverConnectedEvent_t

#include <gap_types.h> Event data structure for the gConnEvtHandoverConnected_c event.

Public Members

bleAddressType_t peerAddressType
Connected device's address type.

bleDeviceAddress_t peerAddress
Connected device's address.

bleLlConnectionRole_t connectionRole
Connection Role - master or slave.

struct gapHandoverDisconnectedEvent_t
#include <gap_types.h> Event data structure for the gHandoverDisconnected_c event.

Public Members

bleResult_t status
Status for the handover disconnect command

struct gapConnectionEvent_t
#include <gap_types.h> Connection event structure: type + data.

Public Members

gapConnectionEventType_t eventType
Event type

union *gapConnectionEvent_t* eventData
Event data, to be interpreted according to *gapConnectionEvent_t.eventType*.

struct gapIdentityInformation_t
#include <gap_types.h> Identity Information structure definition

Public Members

bleIdentityAddress_t identityAddress
Identity Address - Public or Random Static

uint8_t irk[(16U)]
Identity Resolving Key - must not be all-zero if Network Privacy is used

blePrivacyMode_t privacyMode
Privacy mode to be used for the entry on the resolving list

struct gapAutoConnectParams_t
#include <gap_types.h> Parameters for the Auto Connect Scan Mode.

Public Members

uint8_t cNumAddresses

Number of device addresses to automatically connect to.

bool_t writeInFilterAcceptList

If set to TRUE, the device addresses are written in the Filter Accept List before scanning is enabled.

gapConnectionRequestParameters_t *aAutoConnectData

The array of connection request parameters, of size equal to cNumAddresses.

struct gapHostVersion_tag

#include <gap_types.h>

union testParameters

Public Members

uint8_t resolvableTagKey[(16U)]

struct *gapDecisionInstructionsData_tag* arbitraryData

struct *gapDecisionInstructionsData_tag* rssi

struct *gapDecisionInstructionsData_tag* pathLoss

struct *gapDecisionInstructionsData_tag* advA

gapDecisionInstructionsAdvMode_t advMode

struct arbitraryData

struct rssi

struct pathLoss

struct advA

union eventData

Public Members

bleResult_t failReason

Event data for gAdvertisingCommandFailed_c event type: reason of failure to enable or disable advertising.

gapExtScanNotification_t scanNotification

Event data for gExtScanNotification_c event type: Scan Request Received Event.

gapAdvertisingSetTerminated_t advSetTerminated

Event received when advertising in a given advertising set has stopped.

gapPerAdvSubeventDataRequest_t subeventDataRequest

Event received when doing PAWR.

gapPerAdvResponse_t perAdvResponse

Periodic Advertising Response.

uint8_t advHandle

Event data for gExtAdvertisingStateChanged_c event type.

union eventData

Public Members

bleResult_t failReason

Event data for gScanCommandFailed_c or gPeriodicAdvSyncEstablished_c event type: reason of failure to enable/disable scanning or to establish sync.

gapScannedDevice_t scannedDevice

Event data for gDeviceScanned_c event type: scanned device information.

gapExtScannedDevice_t extScannedDevice

Event data for gExtDeviceScanned_c event type: extended scanned device information.

gapPeriodicScannedDevice_t periodicScannedDevice

Event data for gPeriodicDeviceScanned_c event type.

gapPeriodicScannedDeviceV2_t periodicScannedDeviceV2

Event data for gPeriodicDeviceScannedV2_c event type.

gapSyncEstbEventData_t syncEstb

Event data for gPeriodicAdvSyncEstablished_c event type: Sync handle information for the application.

gapSyncLostEventData_t syncLost

Event data for gPeriodicAdvSyncLost_c event type: Sync handle information for the application.

gapConnectionlessIqReport_t iqReport

Event data for gConnectionlessIqReportReceived_c event type: IQ information for the application.

gapMonAdvReportReport_t monAdvReport

Event data for gMonAdvReportEventReceived_c event type.

union pairingCompleteData

Public Members

bool_t withBonding

If pairingSuccessful is TRUE, this indicates whether the devices bonded.

bleResult_t failReason

If pairingSuccessful is FALSE, this contains the reason of failure.

union eventData

Public Members

gapConnectedEvent_t connectedEvent

Data for gConnEvtConnected_c: information about the connection parameters.

gapPairingParameters_t pairingEvent

Data for gConnEvtPairingRequest_c, gConnEvtPairingResponse_c: pairing parameters.

gapAuthenticationRejectedEvent_t authenticationRejectedEvent

Data for gConnEvtAuthenticationRejected_c: reason for rejection.

gapPeripheralSecurityRequestParameters_t peripheralSecurityRequestEvent

Data for gConnEvtPeripheralSecurityRequest_c: Peripheral's security requirements.

gapKeyExchangeRequestEvent_t keyExchangeRequestEvent

Data for gConnEvtKeyExchangeRequest_c: mask indicating the keys that were requested by the peer.

gapKeysReceivedEvent_t keysReceivedEvent

Data for gConnEvtKeysReceived_c: the keys received from the peer.

gapPairingCompleteEvent_t pairingCompleteEvent

Data for gConnEvtPairingComplete_c: fail reason or (if successful) bonding state.

gapLongTermKeyRequestEvent_t longTermKeyRequestEvent

Data for gConnEvtLongTermKeyRequest_c: encryption diversifier and random number.

gapEncryptionChangedEvent_t encryptionChangedEvent

Data for gConnEvtEncryptionChanged_c: new encryption state.

gapDisconnectedEvent_t disconnectedEvent

Data for gConnEvtDisconnected_c: reason for disconnection.

int8_t rssi_dBm

Data for gConnEvtRssiRead_c: value of the RSSI in dBm.

int8_t txPowerLevel_dBm

Data for gConnEvtTxPowerLevelRead_c: value of the TX power.

bleResult_t failReason

Data for gConnEvtPowerReadFailure_c: reason for power reading failure.

uint32_t passkeyForDisplay

gapConnParamsUpdateReq_t connectionUpdateRequest

Data for gConnEvtParameterUpdateRequest_c: connection parameters update.

gapConnParamsUpdateComplete_t connectionUpdateComplete

Data for gConnEvtParameterUpdateComplete_c: connection parameters update.

gapConnLeDataLengthChanged_t leDataLengthChanged

Data for gConnEvtLeDataLengthChanged_c: new data length parameters

gapKeypressNotification_t incomingKeypressNotification

uint32_t numericValueForDisplay

bleChannelMap_t channelMap

Data for gConnEvtChannelMapRead_c: channel map read from the Controller

gapConnIqReport_t connIqReport

Data for gConnEvtIqReportReceived_c: IQ information received from a peer

gapConnCteRequestFailed_t cteRequestFailedEvent

Data for gConnEvtCteRequestFailed_c: CTE Request to peer has failed

bleResult_t perAdvSyncTransferStatus

gapPathLossThresholdEvent_t pathLossThreshold

Data for gConnEvtPathLossThreshold_c: current path loss and zone entered.

gapTransmitPowerReporting_t transmitPowerReporting

Data for gConnEvtTransmitPowerReporting_c: changes in power levels.

gapTransmitPowerInfo_t transmitPowerInfo

Data for gConnEvtEnhancedReadTransmitPowerLevel_c: local power level information.

gapEattConnectionRequest_t eattConnectionRequest

Data for gConnEvtEattConnectionRequest_c: notification of L2Cap Enhanced Connect request for EATT.

gapEattConnectionComplete_t eattConnectionComplete

Data for gConnEvtEattConnectionComplete_c: result of *Gap_EattConnectionRequest()*.

gapEattReconfigureResponse_t eattReconfigureResponse

Data for gConnEvtEattChannelReconfigureResponse_c: result of *Gap_EattReconfigureRequest()*.

gapEattBearerStatusNotification_t eattBearerStatusNotification

Data for gConnEvtEattBearerStatusNotification_c: bearer status update.

gapHandoverConnectedEvent_t handoverConnectedEvent

Data for gConnEvtHandoverConnected_c: information about the connection parameters.

gapHandoverDisconnectedEvent_t handoverDisconnectedEvent

Data for gHandoverDisconnected_c: status of the Gap_HandoverDisconnect.

bleResult_t smError

Data for gConnEvtSmError_c: SM error status.

gapSubrateChangeEvent_t gapSubrateChangeEvent

Data for gConnEvtLeSubrateChange_c: connection parameters update.

uint8_t remoteFeatures[8U]

Data for gConnEvtRemoteFeaturesRead_c: 8-byte mask of features supported by the peer.

GATT

enum attErrorCode_t

ATT error codes

Values:

enumerator gAttErrCodeNoError_c

No error has occured.

enumerator gAttErrCodeInvalidHandle_c

The attribute handle given was not valid on this server.

enumerator gAttErrCodeReadNotPermitted_c

The attribute cannot be read.

enumerator gAttErrCodeWriteNotPermitted_c
The attribute cannot be written.

enumerator gAttErrCodeInvalidPdu_c
The attribute PDU was invalid.

enumerator gAttErrCodeInsufficientAuthentication_c
The attribute requires authentication before it can be read or written.

enumerator gAttErrCodeRequestNotSupported_c
ATT Server does not support the request received from the client.

enumerator gAttErrCodeInvalidOffset_c
Offset specified was past the end of the attribute.

enumerator gAttErrCodeInsufficientAuthorization_c
The attribute requires authorization before it can be read or written.

enumerator gAttErrCodePrepareQueueFull_c
Too many prepare writes have been queued.

enumerator gAttErrCodeAttributeNotFound_c
No attribute found within the given attribute handle range.

enumerator gAttErrCodeAttributeNotLong_c
The attribute cannot be read using the ATT_READ_BLOB_REQ PDU.

enumerator gAttErrCodeInsufficientEncryptionKeySize_c
The Encryption Key Size used for encrypting this link is too short.

enumerator gAttErrCodeInvalidAttributeValueLength_c
The attribute value length is invalid for the operation.

enumerator gAttErrCodeUnlikelyError_c
The attribute request could not be completed due to an unlikely error.

enumerator gAttErrCodeInsufficientEncryption_c
The attribute requires encryption before it can be read or written.

enumerator gAttErrCodeUnsupportedGroupType_c
The attribute type is not a supported grouping attribute.

enumerator gAttErrCodeInsufficientResources_c
Insufficient Resources to complete the request.

enumerator gAttErrCodeDatabaseOutOfSync_c
The server requests the client to rediscover the database.

enumerator gAttErrCodeValueNotAllowed_c
The attribute parameter value was not allowed.

enumerator gAttErrCodeWriteRequestRejected_c
Write request cannot be fulfilled for reasons other than permissions.

enumerator gAttErrCodeCccdImproperlyConfigured_c
CCCD is not configured according to the requirements of the profile or service.

enumerator gAttErrCodeProcedureAlreadyInProgress_c
Profile or service request cannot be serviced because another operation is still in progress.

enumerator gAttErrCodeOutOfRange_c
Attribute value is out of range as defined by a profile or service specification.

enum gattCachingClientState_c
GATT Caching Client states

Values:

enumerator gGattClientChangeUnaware_c
Gatt client is "change-unaware" regarding changes in the database definitions on the Server.

enumerator gGattClientStateChangePending_c
Gatt client is in the process of going from "change-unaware" to "change-aware".

enumerator gGattClientChangeAware_c
Gatt client is "change-aware" regarding the database definitions on the Server.

enum gattClientHashUpdateType_t
Hash Update type

Values:

enumerator gattClientFirstConnection_c
Hash update for the first connection with a peer

enumerator gattClientReconnectBondedPeer_c
Hash update after reconnecting with a bonded peer

enumerator gattClientActiveConnectionUpdate
Hash update when a change occurs during an active connection

enumerator gattClientNoChange
Idle state - no change

enum gattProcedureType_t
GATT Client Procedure type

Values:

enumerator gGattProcExchangeMtu_c
MTU Exchange

enumerator gGattProcDiscoverAllPrimaryServices_c
Primary Service Discovery

enumerator gGattProcDiscoverPrimaryServicesByUuid_c
Discovery of Services by UUID

enumerator gGattProcFindIncludedServices_c
Discovery of Included Services within a Service range

enumerator gGattProcDiscoverAllCharacteristics_c
Characteristic Discovery within Service range

enumerator gGattProcDiscoverCharacteristicByUuid_c
Characteristic Discovery by UUID

enumerator gGattProcDiscoverAllCharacteristicDescriptors_c
Characteristic Descriptor Discovery

enumerator gGattProcReadCharacteristicValue_c
Characteristic Reading using Value handle

enumerator gGattProcReadUsingCharacteristicUuid_c
Characteristic Reading by UUID

enumerator gGattProcReadMultipleCharacteristicValues_c
Reading multiple Characteristics at once

enumerator gGattProcWriteCharacteristicValue_c
Characteristic Writing

enumerator gGattProcReadCharacteristicDescriptor_c
Reading Characteristic Descriptors

enumerator gGattProcWriteCharacteristicDescriptor_c
Writing Characteristic Descriptors

enumerator gGattProcUpdateDatabaseCopy_c
Inform the application to update its database copy

enumerator gGattProcSignalServiceDiscoveryComplete_c
Inform the application that service discovery has finished

enumerator gGattProcReadMultipleVarLengthCharValues_c
Read Multiple Variable Length Characteristic Values

enum gattProcedurePhase_t

GATT Procedure phase

Values:

enumerator gattProcPhaseInitiated

GATT procedure has been initiated.

enumerator gattProcPhaseRunning

GATT procedure is running.

typedef struct *gattService_tag* gattService_t

GATT Service structure definition

typedef uint8_t gattCccdFlags_t

Flags for the value of the Client Characteristic Configuration Descriptor.

bleResult_t Gatt_Init(void)

Initializes the GATT module.

Remark

If the GAP module is present, this function is called internally by *Ble_HostInitialize()*. Otherwise, the application must call this function once at device start-up.

Remark

This function executes synchronously.

Return values

gBleSuccess_c –

bleResult_t Gatt_GetMtu(*deviceId_t* deviceId, uint16_t *pOutMtu)

Retrieves the MTU used with a given connected device.

Remark

This function executes synchronously.

Parameters

- deviceId – **[in]** The device ID of the connected peer.
- pOutMtu – **[out]** Pointer to integer to be written.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.

gCccdEmpty_c

Nothing is enabled.

gCccdNotification_c

Enables notifications.

gCccdIndication_c

Enabled indications.

struct gattAttribute_t

#include <gatt_types.h> GATT Attribute structure definition

Public Members

uint16_t handle

Attribute handle.

bleUuidType_t uuidType

Type of the UUID.

bleUuid_t uuid

The attribute's UUID.

uint16_t valueLength

Length of the attribute value array.

uint16_t maxValueLength

Maximum length of the attribute value array; if this is set to 0, then the attribute's length is fixed and cannot be changed.

uint8_t *paValue

Attribute value array.

struct gattCharacteristic_t

#include <gatt_types.h> GATT Characteristic structure definition

Public Members

gattCharacteristicPropertiesBitFields_t properties

Characteristic Properties as defined by GATT.

gattAttribute_t value

Characteristic Value attribute.

uint8_t cNumDescriptors

Size of the Characteristic Descriptors array.

gattAttribute_t *aDescriptors

Characteristic Descriptors array.

struct gattService_tag

#include <gatt_types.h> GATT Service structure definition

Public Members

uint16_t startHandle

The handle of the Service Declaration attribute.

uint16_t endHandle

The last handle belonging to this Service (followed by another Service declaration of the end of the database).

bleUuidType_t uuidType

Service UUID type.

bleUuid_t uuid

Service UUID.

uint8_t cNumCharacteristics

Size of the Characteristic array.

gattCharacteristic_t *aCharacteristics

Characteristic array.

uint8_t cNumIncludedServices

Size of the Included Services array.

struct *gattService_tag* *aIncludedServices

Included Services array.

struct gattDbCharPresFormat_t

#include <gatt_types.h> Characteristic Presentation Format Descriptor structure

Public Members

uint8_t format

Format of the value of this characteristic.

int8_t exponent

Exponent field to determine how the value of this characteristic is further formatted.

uint16_t unitUuid16

The unit of this characteristic.

uint8_t ns
The name space of the description.

uint16_t description
The description of this characteristic.

struct gattHandleRange_t
#include <gatt_types.h> GATT Handle Range structure definition

Public Members

uint16_t startHandle
Start Handle.

uint16_t endHandle
End Handle - shall be greater than or equal to Start Handle.

struct procStatus_t
#include <gatt_types.h> GATT Procedure Status structure definition

Public Members

bool_t isOngoing
Indicates whether the GATT procedure type is ongoing.

gattProcedureType_t ongoingProcedureType
Procedure type.

gattProcedurePhase_t ongoingProcedurePhase
Procedure phase.

struct procDataStruct_t
#include <gatt_types.h> GATT Procedure Data structure definition. Structure members are ordered to minimize the padding size

Public Members

uint16_t index
Number of entries currently stored.

uint16_t max
Maximum number of entries to store.

bleUuid_t charUuid
UUID of the characteristic to be discovered.

bleUuidType_t charUuidType
UUID type of the characteristic to be discovered.

bool_t reliableLongWrite
Indicates whether to do reliable long writes.

bool_t bAllocatedArray
Indicates whether the array is allocated.

union *procDataStruct_t* pOutActualCount
Pointer to the memory locatio where to store the final number of added entries.

union *procDataStruct_t* array
Array where data is stored.

union *procDataStruct_t* reqParams
Parameters of the GATT procedure

union pOutActualCount

Public Members

uint8_t *pCount8b
Pointer to a uint8_t memory location where to store the final number of added entries.

uint16_t *pCount16b
Pointer to a uint16_t memory location where to store the final number of added entries.

union array

Public Members

gattService_t *aServices
Array of services.

gattCharacteristic_t *aChars
Array of characteristics.

gattAttribute_t *aDescriptors
Array of descriptors.

uint8_t *aBytes
Array of bytes.

uint16_t *aHandles
Array of handles.

union reqParams

Public Members

attReadByGroupTypeRequestParams_t rbgtParams
Read By Group Type Request Parameters.

attFindByTypeValueRequestParams_t fbtvParams
Find By Type Value Request Parameters.

attReadByTypeRequestParams_t rbtParams
Read By Type Request Parameters.

attFindInformationRequestParams_t fiParams
Find Information Request Parameters.

attReadRequestParams_t rParams
Read Request Parameters.

attReadBlobRequestParams_t rbParams
Read Blob Request Parameters.

attReadMultipleRequestParams_t rmParams
Read Multiple Request Parameters.

attVarWriteRequestAndCommandParams_t wParams
Write Request and Write Command Parameters for variable value length.

attPrepareWriteRequestResponseParams_t pwParams
Prepare Write Request and Prepare Write Response Parameters.

attExecuteWriteRequestParams_t ewParams
Execute Write Request Parameters

GATT Client

enum gattProcedureResult_t
GATT Client Procedure Result type

Values:

enumerator gGattProcSuccess_c
The procedure was completed successfully.

enumerator gGattProcError_c
The procedure was terminated due to an error.

typedef void (*gattClientProcedureCallback_t)(*deviceId_t* deviceId, *gattProcedureType_t* procedureType, *gattProcedureResult_t* procedureResult, *bleResult_t* error)

GATT Client Procedure Callback type

typedef void (*gattClientNotificationCallback_t)(*deviceId_t* deviceId, uint16_t characteristicValueHandle, uint8_t *aValue, uint16_t valueLength)

GATT Client Notification Callback prototype

typedef *gattClientNotificationCallback_t* gattClientIndicationCallback_t

GATT Client Indication Callback prototype

typedef void (*gattClientMultipleValueNotificationCallback_t)(*deviceId_t* deviceId, uint8_t *aHandleLenValue, uint32_t totalLength)

GATT Client Multiple Value Notification Callback prototype

typedef void (*gattClientEnhancedProcedureCallback_t)(*deviceId_t* deviceId, *bearerId_t* bearerId, *gattProcedureType_t* procedureType, *gattProcedureResult_t* procedureResult, *bleResult_t* error)

GATT Client Enhanced Procedure Callback type

typedef void (*gattClientEnhancedNotificationCallback_t)(*deviceId_t* deviceId, *bearerId_t* bearerId, uint16_t characteristicValueHandle, uint8_t *aValue, uint16_t valueLength)

GATT Client Enhanced Notification Callback prototype

typedef *gattClientEnhancedNotificationCallback_t* gattClientEnhancedIndicationCallback_t

GATT Client Enhanced Indication Callback prototype

typedef void (*gattClientEnhancedMultipleValueNotificationCallback_t)(*deviceId_t* deviceId, *bearerId_t* bearerId, uint8_t *aHandleLenValue, uint32_t totalLength)

GATT Client Enhanced Multiple Value Notification Callback prototype

procDataStruct_t *pProcedureData[]

bleResult_t GattClient_Init(void)

Initializes the GATT Client functionality.

Remark

This should be called once at device startup, if necessary.

Remark

This function executes synchronously.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – Host library was compiled without GATT client support.

bleResult_t GattClient_ResetProcedure(void)

Resets any ongoing GATT Client Procedure.

Remark

This function should be called if an ongoing Client procedure needs to be stopped.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – Host library was compiled without GATT client support.

bleResult_t GattClient_RegisterProcedureCallback(*gattClientProcedureCallback_t* callback)

Installs the application callback for the GATT Client module Procedures.

Remark

This function executes synchronously.

Parameters

- callback – **[in]** Application defined callback to be triggered by this module.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – Host library was compiled without GATT client support.

bleResult_t GattClient_RegisterNotificationCallback(*gattClientNotificationCallback_t* callback)

Installs the application callback for Server Notifications.

Remark

This function executes synchronously.

Parameters

- callback – **[in]** Application defined callback to be triggered by this module.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – Host library was compiled without GATT client support.

bleResult_t GattClient_RegisterIndicationCallback(*gattClientIndicationCallback_t* callback)

Installs the application callback for Server Indications.

Remark

This function executes synchronously.

Parameters

- callback – **[in]** Application defined callback to be triggered by this module.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – Host library was compiled without GATT client support.

bleResult_t GattClient_RegisterMultipleValueNotificationCallback(*gattClientMultipleValueNotificationCallback_t* callback)

Installs the application callback for Server Multiple Value Notification.

Remark

This function executes synchronously.

Parameters

- callback – **[in]** Application defined callback to be triggered by this module.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – Host library was compiled without GATT client support.

bleResult_t GattClient_ExchangeMtu(*deviceId_t* deviceId, uint16_t mtu)

Initializes the MTU Exchange procedure.

Remark

If gBleSuccess_c is returned, the completion of this procedure is signaled by the Client Procedure callback.

Parameters

- deviceId – **[in]** Device ID of the connected peer.
- mtu – **[in]** Desired MTU size.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.

- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleFeatureNotSupported_c – Host library was compiled without GATT client support.

bleResult_t GattClient_DiscoverAllPrimaryServices(*deviceId_t* deviceId, *gattService_t* *aOutPrimaryServices, uint8_t maxServiceCount, uint8_t *pOutDiscoveredCount)

Initializes the Primary Service Discovery procedure.

Remark

If gBleSuccess_c is returned, the completion of this procedure is signaled by the Client Procedure callback.

Parameters

- deviceId – **[in]** Device ID of the connected peer.
- aOutPrimaryServices – **[out]** Statically allocated array of gattService_t. The GATT module fills each Service's handle range and UUID.
- maxServiceCount – **[in]** Maximum number of services to be filled.
- pOutDiscoveredCount – **[out]** The actual number of services discovered.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleFeatureNotSupported_c – Host library was compiled without GATT client support.

bleResult_t GattClient_DiscoverPrimaryServicesByUuid(*deviceId_t* deviceId, *bleUuidType_t* uuidType, const *bleUuid_t* *pUuid, *gattService_t* *aOutPrimaryServices, uint8_t maxServiceCount, uint8_t *pOutDiscoveredCount)

Initializes the Primary Service Discovery By UUID procedure.

Remark

If gBleSuccess_c is returned, the completion of this procedure is signaled by the Client Procedure callback.

Parameters

- deviceId – **[in]** Device ID of the connected peer.
- uuidType – **[in]** Service UUID type.
- pUuid – **[in]** Service UUID.

- aOutPrimaryServices – **[out]** Statically allocated array of gattService_t. The GATT module fills each Service's handle range.
- maxServiceCount – **[in]** Maximum number of services to be filled.
- pOutDiscoveredCount – **[out]** The actual number of services discovered.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleFeatureNotSupported_c – Host library was compiled without GATT client support.

bleResult_t GattClient_FindIncludedServices(*deviceId_t* deviceId, *gattService_t* *pIoService, uint8_t maxServiceCount)

Initializes the Find Included Services procedure.

Remark

If gBleSuccess_c is returned, the completion of this procedure is signaled by the Client Procedure callback.

Parameters

- deviceId – **[in]** Device ID of the connected peer.
- pIoService – **[inout]** The service within which inclusions should be searched. The GATT module uses the Service's handle range and fills the included Services' handle ranges, UUID types and the UUIDs if they are 16-bit UUIDs.
- maxServiceCount – **[in]** Maximum number of included services to be filled.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleFeatureNotSupported_c – Host library was compiled without GATT client support.

bleResult_t GattClient_DiscoverAllCharacteristicsOfService(*deviceId_t* deviceId, *gattService_t* *pIoService, uint8_t maxCharacteristicCount)

Initializes the Characteristic Discovery procedure for a given Service.

Remark

If gBleSuccess_c is returned, the completion of this procedure is signaled by the Client Procedure callback.

Parameters

- deviceId – **[in]** Device ID of the connected peer.
- pIoService – **[inout]** The service within which characteristics should be searched. The GATT module uses the Characteristic's range.
- maxCharacteristicCount – **[in]** Maximum number of characteristics to be filled.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleFeatureNotSupported_c – Host library was compiled without GATT client support.

bleResult_t GattClient_DiscoverCharacteristicOfServiceByUuid(*deviceId_t* deviceId, *bleUuidType_t* uuidType, const *bleUuid_t* *pUuid, const *gattService_t* *pService, *gattCharacteristic_t* *aOutCharacteristics, uint8_t maxCharacteristicCount, uint8_t *pOutDiscoveredCount)

Initializes the Characteristic Discovery procedure for a given Service, with a given UUID.

Remark

If gBleSuccess_c is returned, the completion of this procedure is signaled by the Client Procedure callback.

Parameters

- deviceId – **[in]** Device ID of the connected peer.
- uuidType – **[in]** Characteristic UUID type.
- pUuid – **[in]** Characteristic UUID.
- pService – **[in]** The service within which characteristics should be searched.
- aOutCharacteristics – **[out]** The allocated array of Characteristics to be filled.
- maxCharacteristicCount – **[in]** Maximum number of characteristics to be filled.
- pOutDiscoveredCount – **[out]** The actual number of characteristics discovered.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleFeatureNotSupported_c – Host library was compiled without GATT client support.

bleResult_t GattClient_DiscoverAllCharacteristicDescriptors(*deviceId_t* deviceId, *gattCharacteristic_t* *pIoCharacteristic, uint16_t endingHandle, uint8_t maxDescriptorCount)

Initializes the Characteristic Descriptor Discovery procedure for a given Characteristic.

Remark

If gBleSuccess_c is returned, the completion of this procedure is signaled by the Client Procedure callback. The endingHandle parameter should be known by the application if Characteristic Discovery was performed, i.e., if the next Characteristic declaration handle is known, then subtract 1 to obtain the endingHandle for the current Characteristic. If the last handle of the Characteristic is still unknown, set the endingHandle parameter to 0xFFFF.

Parameters

- deviceId – **[in]** Device ID of the connected peer.
- pIoCharacteristic – **[inout]** The characteristic within which descriptors should be searched. The GATT module uses the Characteristic's handle and fills each descriptor's handle and UUID.
- endingHandle – **[in]** The last handle of the Characteristic.
- maxDescriptorCount – **[in]** Maximum number of descriptors to be filled.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleFeatureNotSupported_c – Host library was compiled without GATT client support.

bleResult_t GattClient_ReadCharacteristicValue(*deviceId_t* deviceId, *gattCharacteristic_t* *pIoCharacteristic, uint16_t maxReadBytes)

Initializes the Characteristic Read procedure for a given Characteristic.

Remark

If gBleSuccess_c is returned, the completion of this procedure is signaled by the Client Procedure callback.

Parameters

- deviceId – **[in]** Device ID of the connected peer.
- pIoCharacteristic – **[inout]** The characteristic whose value must be read. The GATT module uses the value handle and fills the value and length.
- maxReadBytes – **[in]** Maximum number of bytes to be read.

Return values

- gBleSuccess_c –

- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleFeatureNotSupported_c – Host library was compiled without GATT client support.

bleResult_t GattClient_ReadUsingCharacteristicUuid(*deviceId_t* deviceId, *bleUuidType_t* uuidType, const *bleUuid_t* *pUuid, const *gattHandleRange_t* *pHandleRange, uint8_t *aOutBuffer, uint16_t maxReadBytes, uint16_t *pOutActualReadBytes)

Initializes the Characteristic Read By UUID procedure.

Remark

This procedure returns the Characteristics found within the specified range with the specified UUID. aOutBuffer will contain the Handle-Value pair length (1 byte), then Handle-Value pairs for all Characteristic Values found with the specified UUID.

Remark

If gBleSuccess_c is returned, the completion of this procedure is signaled by the Client Procedure callback.

Parameters

- deviceId – **[in]** Device ID of the connected peer.
- uuidType – **[in]** Characteristic UUID type.
- pUuid – **[in]** Characteristic UUID.
- pHandleRange – **[in]** Handle range for the search or NULL. If this is NULL, the search range is 0x0001-0xffff.
- aOutBuffer – **[out]** The allocated buffer to read into.
- maxReadBytes – **[in]** Maximum number of bytes to be read.
- pOutActualReadBytes – **[out]** The actual number of bytes read.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleFeatureNotSupported_c – Host library was compiled without GATT client support.

bleResult_t GattClient_ReadMultipleCharacteristicValues(*deviceId_t* deviceId, uint8_t cNumCharacteristics, *gattCharacteristic_t* *aIoCharacteristics)

Initializes the Characteristic Read Multiple procedure.

Remark

If gBleSuccess_c is returned, the completion of this procedure is signaled by the Client Procedure callback.

Parameters

- deviceId – **[in]** Device ID of the connected peer.
- aIoCharacteristics – **[inout]** Array of the characteristics whose values are to be read. The GATT module uses each Characteristic's value handle and maxValueLength fills each value and length.
- cNumCharacteristics – **[in]** Number of characteristics in the array.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleFeatureNotSupported_c – Host library was compiled without GATT client support.

bleResult_t GattClient_WriteCharacteristicValue(*deviceId_t* deviceId, const *gattCharacteristic_t* *pCharacteristic, uint16_t valueLength, const uint8_t *aValue, bool_t withoutResponse, bool_t signedWrite, bool_t doReliableLongCharWrites, const uint8_t *aCsrk)

Initializes the Characteristic Write procedure for a given Characteristic.

Remark

If gBleSuccess_c is returned, the completion of this procedure is signaled by the Client Procedure callback.

Parameters

- deviceId – **[in]** Device ID of the connected peer.
- pCharacteristic – **[in]** The characteristic whose value must be written. The GATT module uses the value handle.
- valueLength – **[in]** Number of bytes to be written.
- aValue – **[in]** Array of bytes to be written.
- withoutResponse – **[in]** Indicates if a Write Command is used.
- signedWrite – **[in]** Deprecated parameter.
- doReliableLongCharWrites – **[in]** Indicates Reliable Long Writes.
- aCsrk – **[in]** Deprecated parameter.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleOverflow_c – TX queue for device is full.

- gBleFeatureNotSupported_c – Host library was compiled without GATT client support.

bleResult_t GattClient_ReadCharacteristicDescriptor(*deviceId_t* deviceId, *gattAttribute_t* *pIoDescriptor, uint16_t maxReadBytes)

Initializes the Characteristic Descriptor Read procedure for a given Characteristic Descriptor.

Remark

If gBleSuccess_c is returned, the completion of this procedure is signaled by the Client Procedure callback.

Parameters

- deviceId – **[in]** Device ID of the connected peer.
- pIoDescriptor – **[inout]** The characteristic descriptor whose value must be read. The GATT module uses the attribute's handle and fills the attribute's value and length.
- maxReadBytes – **[in]** Maximum number of bytes to be read.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleFeatureNotSupported_c – Host library was compiled without GATT client support.

bleResult_t GattClient_WriteCharacteristicDescriptor(*deviceId_t* deviceId, const *gattAttribute_t* *pDescriptor, uint16_t valueLength, const uint8_t *aValue)

Initializes the Characteristic Descriptor Write procedure for a given Characteristic Descriptor.

Remark

If gBleSuccess_c is returned, the completion of this procedure is signaled by the Client Procedure callback.

Parameters

- deviceId – **[in]** Device ID of the connected peer.
- pDescriptor – **[in]** The characteristic descriptor whose value must be written. The GATT module uses the attribute's handle.
- valueLength – **[in]** Number of bytes to be written.
- aValue – **[in]** Array of bytes to be written.

Return values

- gBleSuccess_c –

- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleFeatureNotSupported_c – Host library was compiled without GATT client support.

bleResult_t GattClient_ReadMultipleVariableCharacteristicValues(*deviceId_t* deviceId, uint8_t cNumCharacteristics, *gattCharacteristic_t* *pIoCharacteristics)

Initializes the Read Multiple Variable Length Characteristic Values procedure.

Remark

If gBleSuccess_c is returned, the completion of this procedure is signaled by the Client Procedure callback.

Parameters

- deviceId – **[in]** Device ID of the connected peer.
- cNumCharacteristics – **[in]** Number of characteristics in the array.
- pIoCharacteristics – **[in]** Pointer to the array of the characteristics whose values are to be read.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleFeatureNotSupported_c – Host library was compiled without GATT client support.

bleResult_t GattClient_RegisterEnhancedProcedureCallback(*gattClientEnhancedProcedureCallback_t* callback)

Installs the application callback for the GATT Client module Procedures on Enhanced ATT bearers.

Remark

This function executes synchronously.

Parameters

- callback – **[in]** Application defined callback to be triggered by this module.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – Host library was compiled without GATT client support.

bleResult_t GattClient_RegisterEnhancedNotificationCallback(*gattClientEnhancedNotificationCallback_t* callback)

Installs the application callback for Server Notifications.

Remark

This function executes synchronously.

Parameters

- callback – **[in]** Application defined callback to be triggered by this module.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – Host library was compiled without GATT client support.

bleResult_t GattClient_RegisterEnhancedIndicationCallback(*gattClientEnhancedIndicationCallback_t* callback)

Installs the application callback for Server Indications.

Remark

This function executes synchronously.

Parameters

- callback – **[in]** Application defined callback to be triggered by this module.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – Host library was compiled without GATT client support.

bleResult_t GattClient_RegisterEnhancedMultipleValueNotificationCallback(*gattClientEnhancedMultipleValueNotific* callback)

Installs the application callback for Server Multiple Value Notification.

Remark

This function executes synchronously.

Parameters

- callback – **[in]** Application defined callback to be triggered by this module.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – Host library was compiled without GATT client support.

bleResult_t GattClient_EnhancedDiscoverAllPrimaryServices(*deviceId_t* deviceId, *bearerId_t* bearerId, *gattService_t* *aOutPrimaryServices, uint8_t maxServiceCount, uint8_t *pOutDiscoveredCount)

Initializes the Primary Service Discovery procedure.

Remark

If gBleSuccess_c is returned, the completion of this procedure is signaled by the Client Enhanced Procedure callback.

Parameters

- deviceId – **[in]** Device ID of the connected peer.
- bearerId – **[in]** Enhanced ATT bearer ID of the connected peer.
- aOutPrimaryServices – **[out]** Statically allocated array of gattService_t. The GATT module fills each Service's handle range and UUID.
- maxServiceCount – **[in]** Maximum number of services to be filled.
- pOutDiscoveredCount – **[out]** The actual number of services discovered.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleFeatureNotSupported_c – Host library was compiled without GATT client support.

bleResult_t GattClient_EnhancedDiscoverPrimaryServicesByUuid(*deviceId_t* deviceId, *bearerId_t* bearerId, *bleUuidType_t* uuidType, const *bleUuid_t* *pUuid, *gattService_t* *aOutPrimaryServices, uint8_t maxServiceCount, uint8_t *pOutDiscoveredCount)

Initializes the Primary Service Discovery By UUID procedure.

Remark

If gBleSuccess_c is returned, the completion of this procedure is signaled by the Client Enhanced Procedure callback.

Parameters

- deviceId – **[in]** Device ID of the connected peer.
- bearerId – **[in]** Enhanced ATT bearer ID of the connected peer.
- uuidType – **[in]** Service UUID type.
- pUuid – **[in]** Service UUID.

- aOutPrimaryServices – **[out]** Statically allocated array of gattService_t. The GATT module fills each Service's handle range.
- maxServiceCount – **[in]** Maximum number of services to be filled.
- pOutDiscoveredCount – **[out]** The actual number of services discovered.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleFeatureNotSupported_c – Host library was compiled without GATT client support.

bleResult_t GattClient_EnhancedFindIncludedServices(*deviceId_t* deviceId, *bearerId_t* bearerId, *gattService_t* *pIoService, uint8_t maxServiceCount)

Initializes the Find Included Services procedure.

Remark

If gBleSuccess_c is returned, the completion of this procedure is signaled by the Client Enhanced Procedure callback.

Parameters

- deviceId – **[in]** Device ID of the connected peer.
- bearerId – **[in]** Enhanced ATT bearer ID of the connected peer.
- pIoService – **[inout]** The service within which inclusions should be searched. The GATT module uses the Service's handle range and fills the included Services' handle ranges, UUID types and the UUIDs if they are 16-bit UUIDs.
- maxServiceCount – **[in]** Maximum number of included services to be filled.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleFeatureNotSupported_c – Host library was compiled without GATT client support.

bleResult_t GattClient_EnhancedDiscoverAllCharacteristicsOfService(*deviceId_t* deviceId, *bearerId_t* bearerId, *gattService_t* *pIoService, uint8_t maxCharacteristicCount)

Initializes the Characteristic Discovery procedure for a given Service.

Remark

If gBleSuccess_c is returned, the completion of this procedure is signaled by the Client Enhanced Procedure callback.

Parameters

- deviceId – **[in]** Device ID of the connected peer.
- bearerId – **[in]** Enhanced ATT bearer ID of the connected peer.
- pIoService – **[inout]** The service within which characteristics should be searched. The GATT module uses the Characteristic's range.
- maxCharacteristicCount – **[in]** Maximum number of characteristics to be filled.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleFeatureNotSupported_c – Host library was compiled without GATT client support.

bleResult_t GattClient_EnhancedDiscoverCharacteristicOfServiceByUuid(*deviceId_t* deviceId, *bearerId_t* bearerId, *bleUuidType_t* uuidType, const *bleUuid_t* *pUuid, const *gattService_t* *pService, *gattCharacteristic_t* *aOutCharacteristics, uint8_t maxCharacteristicCount, uint8_t *pOutDiscoveredCount)

Initializes the Characteristic Discovery procedure for a given Service, with a given UUID.

Remark

If gBleSuccess_c is returned, the completion of this procedure is signaled by the Client Enhanced Procedure callback.

Parameters

- deviceId – **[in]** Device ID of the connected peer.
- bearerId – **[in]** Enhanced ATT bearer ID of the connected peer.
- uuidType – **[in]** Characteristic UUID type.
- pUuid – **[in]** Characteristic UUID.
- pService – **[in]** The service within which characteristics should be searched.
- aOutCharacteristics – **[out]** The allocated array of Characteristics to be filled.
- maxCharacteristicCount – **[in]** Maximum number of characteristics to be filled.

- pOutDiscoveredCount – **[out]** The actual number of characteristics discovered.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleFeatureNotSupported_c – Host library was compiled without GATT client support.

bleResult_t GattClient_EnhancedDiscoverAllCharacteristicDescriptors(*deviceId_t* deviceId, *bearerId_t* bearerId, *gattCharacteristic_t* *pIoCharacteristic, uint16_t endingHandle, uint8_t maxDescriptorCount)

Initializes the Characteristic Descriptor Discovery procedure for a given Characteristic.

Remark

If gBleSuccess_c is returned, the completion of this procedure is signaled by the Client Enhanced Procedure callback. The endingHandle parameter should be known by the application if Characteristic Discovery was performed, i.e., if the next Characteristic declaration handle is known, then subtract 1 to obtain the endingHandle for the current Characteristic. If the last handle of the Characteristic is still unknown, set the endingHandle parameter to 0xFFFF.

Parameters

- deviceId – **[in]** Device ID of the connected peer.
- bearerId – **[in]** Enhanced ATT bearer ID of the connected peer.
- pIoCharacteristic – **[inout]** The characteristic within which descriptors should be searched. The GATT module uses the Characteristic's handle and fills each descriptor's handle and UUID.
- endingHandle – **[in]** The last handle of the Characteristic.
- maxDescriptorCount – **[in]** Maximum number of descriptors to be filled.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleFeatureNotSupported_c – Host library was compiled without GATT client support.

bleResult_t GattClient_EnhancedReadCharacteristicValue(*deviceId_t* deviceId, *bearerId_t* bearerId, *gattCharacteristic_t* *pIoCharacteristic, uint16_t maxReadBytes)

Initializes the Characteristic Read procedure for a given Characteristic.

Remark

If gBleSuccess_c is returned, the completion of this procedure is signaled by the Client Enhanced Procedure callback.

Parameters

- deviceId – **[in]** Device ID of the connected peer.
- bearerId – **[in]** Enhanced ATT bearer ID of the connected peer.
- pIoCharacteristic – **[inout]** The characteristic whose value must be read. The GATT module uses the value handle and fills the value and length.
- maxReadBytes – **[in]** Maximum number of bytes to be read.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleFeatureNotSupported_c – Host library was compiled without GATT client support.

bleResult_t GattClient_EnhancedReadUsingCharacteristicUuid(*deviceId_t* deviceId, *bearerId_t* bearerId, *bleUuidType_t* uuidType, const *bleUuid_t* *pUuid, const *gattHandleRange_t* *pHandleRange, uint8_t *aOutBuffer, uint16_t maxReadBytes, uint16_t *pOutActualReadBytes)

Initializes the Characteristic Read By UUID procedure.

Remark

This procedure returns the Characteristics found within the specified range with the specified UUID. aOutBuffer will contain the Handle-Value pair length (1 byte), then Handle-Value pairs for all Characteristic Values found with the specified UUID.

Remark

If gBleSuccess_c is returned, the completion of this procedure is signaled by the Client Enhanced Procedure callback.

Parameters

- deviceId – **[in]** Device ID of the connected peer.
- bearerId – **[in]** Enhanced ATT bearer ID of the connected peer.
- uuidType – **[in]** Characteristic UUID type.
- pUuid – **[in]** Characteristic UUID.
- pHandleRange – **[in]** Handle range for the search or NULL. If this is NULL, the search range is 0x0001-0xffff.

- aOutBuffer – **[out]** The allocated buffer to read into.
- maxReadBytes – **[in]** Maximum number of bytes to be read.
- pOutActualReadBytes – **[out]** The actual number of bytes read.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleFeatureNotSupported_c – Host library was compiled without GATT client support.

bleResult_t GattClient_EnhancedReadMultipleCharacteristicValues(*deviceId_t* deviceId, *bearerId_t* bearerId, uint8_t cNumCharacteristics, *gattCharacteristic_t* *aIoCharacteristics)

Initializes the Characteristic Read Multiple procedure.

Remark

If gBleSuccess_c is returned, the completion of this procedure is signaled by the Client Enhanced Procedure callback.

Parameters

- deviceId – **[in]** Device ID of the connected peer.
- bearerId – **[in]** Enhanced ATT bearer ID of the connected peer.
- aIoCharacteristics – **[inout]** Array of the characteristics whose values are to be read. The GATT module uses each Characteristic's value handle and maxValueLength fills each value and length.
- cNumCharacteristics – **[in]** Number of characteristics in the array.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleFeatureNotSupported_c – Host library was compiled without GATT client support.

bleResult_t GattClient_EnhancedWriteCharacteristicValue(*deviceId_t* deviceId, *bearerId_t* bearerId, const *gattCharacteristic_t* *pCharacteristic, uint16_t valueLength, const uint8_t *aValue, bool_t withoutResponse, bool_t doReliableLongCharWrites, const uint8_t *aCsrk)

Initializes the Characteristic Write procedure for a given Characteristic.

Remark

If gBleSuccess_c is returned, the completion of this procedure is signaled by the Client Enhanced Procedure callback.

Parameters

- deviceId – **[in]** Device ID of the connected peer.
- bearerId – **[in]** Enhanced ATT bearer ID of the connected peer.
- pCharacteristic – **[in]** The characteristic whose value must be written. The GATT module uses the value handle.
- valueLength – **[in]** Number of bytes to be written.
- aValue – **[in]** Array of bytes to be written.
- withoutResponse – **[in]** Indicates if a Write Command is used.
- signedWrite – **[in]** Deprecated parameter.
- doReliableLongCharWrites – **[in]** Indicates Reliable Long Writes.
- aCsrk – **[in]** Deprecated parameter.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOverflow_c – TX queue for device is full.
- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleFeatureNotSupported_c – Host library was compiled without GATT client support.

bleResult_t GattClient_EnhancedReadCharacteristicDescriptor(*deviceId_t* deviceId, *bearerId_t* bearerId, *gattAttribute_t* *pIoDescriptor, uint16_t maxReadBytes)

Initializes the Characteristic Descriptor Read procedure for a given Characteristic Descriptor.

Remark

If gBleSuccess_c is returned, the completion of this procedure is signaled by the Client Enhanced Procedure callback.

Parameters

- deviceId – **[in]** Device ID of the connected peer.
- bearerId – **[in]** Enhanced ATT bearer ID of the connected peer.
- pIoDescriptor – **[inout]** The characteristic descriptor whose value must be read. The GATT module uses the attribute's handle and fills the attribute's value and length.
- maxReadBytes – **[in]** Maximum number of bytes to be read.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleFeatureNotSupported_c – Host library was compiled without GATT client support.

bleResult_t GattClient_EnhancedWriteCharacteristicDescriptor(*deviceId_t* deviceId, *bearerId_t* bearerId, const *gattAttribute_t* *pDescriptor, uint16_t valueLength, const uint8_t *aValue)

Initializes the Characteristic Descriptor Write procedure for a given Characteristic Descriptor.

Remark

If gBleSuccess_c is returned, the completion of this procedure is signaled by the Client Enhanced Procedure callback.

Parameters

- deviceId – **[in]** Device ID of the connected peer.
- bearerId – **[in]** Enhanced ATT bearer ID of the connected peer.
- pDescriptor – **[in]** The characteristic descriptor whose value must be written. The GATT module uses the attribute's handle.
- valueLength – **[in]** Number of bytes to be written.
- aValue – **[in]** Array of bytes to be written.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleFeatureNotSupported_c – Host library was compiled without GATT client support.

bleResult_t GattClient_EnhancedReadMultipleVariableCharacteristicValues(*deviceId_t* deviceId, *bearerId_t* bearerId, uint8_t cNumCharacteristics, *gattCharacteristic_t* *pIoCharacteristics)

Initializes the Read Multiple Variable Length Characteristic Values procedure.

Remark

If gBleSuccess_c is returned, the completion of this procedure is signaled by the Client Enhanced Procedure callback.

Parameters

- deviceId – **[in]** Device ID of the connected peer.
- bearerId – **[in]** Enhanced ATT bearer ID of the connected peer.
- cNumCharacteristics – **[in]** Number of characteristics in the array.
- pIoCharacteristics – **[in]** Pointer to the array of the characteristics whose values are to be read.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleFeatureNotSupported_c – Host library was compiled without GATT client support.

GattClient_SimpleCharacteristicWrite(deviceId, pChar, valueLength, aValue)

Executes the basic Characteristic Write operation (with server confirmation).

Parameters

- deviceId – **[in]** Device ID of the connected GATT Server.
- pChar – **[in]** Pointer to the Characteristic being written.
- valueLength – **[in]** Size in bytes of the value to be written.
- aValue – **[in]** Array of bytes to be written.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOverflow_c – TX queue for device is full.
- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleFeatureNotSupported_c – Host library was compiled without GATT client support.

GattClient_CharacteristicWriteWithoutResponse(deviceId, pChar, valueLength, aValue)

Executes the Characteristic Write Without Response operation.

Parameters

- deviceId – **[in]** Device ID of the connected GATT Server.
- pChar – **[in]** Pointer to the Characteristic being written.
- valueLength – **[in]** Size in bytes of the value to be written.
- aValue – **[in]** Array of bytes to be written.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOverflow_c – TX queue for device is full.
- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleFeatureNotSupported_c – Host library was compiled without GATT client support.

GATT Server

enum gattServerEventType_t

GATT Server Event type enumeration

Values:

enumerator gEvtMtuChanged_c

ATT_MTU was changed after the MTU exchange.

enumerator gEvtHandleValueConfirmation_c

Received a Handle Value Confirmation from the Client.

enumerator gEvtAttributeWritten_c

An attribute registered with GattServer_RegisterHandlesForWriteNotifications was written. After receiving this event, application must call GattServer_SendAttributeWrittenStatus. Application must write the Attribute in the Database if it considers necessary.

enumerator gEvtCharacteristicCccdWritten_c

A CCCD was written. Application should save the CCCD value with Gap_SaveCccd.

enumerator gEvtAttributeWrittenWithoutResponse_c

An attribute registered with GattServer_RegisterHandlesForWriteNotifications was written without response (with ATT Write Command). Application must write the Attribute Value in the Database if it considers necessary.

enumerator gEvtError_c

An error appeared during a Server-initiated procedure.

enumerator gEvtLongCharacteristicWritten_c

A long characteristic was written.

enumerator gEvtAttributeRead_c

An attribute registered with GattServer_RegisterHandlesForReadNotifications is being read. After receiving this event, application must call GattServer_SendAttributeReadStatus.

enumerator gEvtInvalidPduReceived_c

An invalid PDU was received from Client. Application decides if disconnection is required

enum gattServerProcedureType_t

Server-initiated procedure type enumeration

Values:

enumerator gSendAttributeWrittenStatus_c

Procedure initiated by GattServer_SendAttributeWrittenStatus.

enumerator gSendAttributeReadStatus_c

Procedure initiated by GattServer_SendAttributeReadStatus.

enumerator gSendNotification_c

Procedure initiated by GattServer_SendNotification.

enumerator gSendIndication_c

Procedure initiated by GattServer_SendIndication.

enumerator gSendMultipleValNotification_c

Procedure initiated by GattServer_SendMultipleHandleValueNotification.

typedef void (*gattServerCallback_t)(*deviceId_t* deviceId, *gattServerEvent_t* *pServerEvent)

GATT Server Callback prototype

typedef void (*gattServerEnhancedCallback_t)(*deviceId_t* deviceId, *bearerId_t* bearerId, *gattServerEvent_t* *pServerEvent)

GATT Server Enhanced Callback prototype

bleResult_t GattServer_Init(void)

Initializes the GATT Server module.

Remark

Application does not need to call this function if *Gatt_Init()* is called.

Remark

This function executes synchronously.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – Host library was compiled without GATT server support.

bleResult_t GattServer_RegisterCallback(*gattServerCallback_t* callback)

Installs an application callback for the GATT Server module.

Remark

This function executes synchronously.

Parameters

- callback – **[in]** Application-defined callback to be triggered by this module.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – Host library was compiled without GATT server support.

bleResult_t GattServer_RegisterHandlesForWriteNotifications(uint8_t handleCount, const uint16_t *aAttributeHandles)

Registers the attribute handles that will be notified through the GATT Server callback when a GATT Client attempts to modify the attributes' values.

Remark

The application is responsible for actually writing the new requested values in the GATT database. Service and profile-specific control-point characteristics should have their value handles in this list so that the application may get notified when a GATT Client writes it.

Remark

This function executes synchronously.

Parameters

- handleCount – **[in]** Number of handles in array.
- aAttributeHandles – **[in]** Array of handles.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOverflow_c – The maximum number of supported handles for write notifications has been reached.
- gBleFeatureNotSupported_c – Host library was compiled without GATT server support.

bleResult_t GattServer_UnregisterHandlesForWriteNotifications(uint8_t handleCount, const uint16_t *aAttributeHandles)

Unregisters the attribute handles that will be notified through the GATT Server callback when a GATT Client attempts to modify the attributes' values.

Remark

To unregister all the list, pass 0 count and NULL.

Remark

This function executes synchronously.

Parameters

- handleCount – **[in]** Number of handles in array.
- aAttributeHandles – **[in]** Array of handles.

Return values

- gBleSuccess_c –

- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleFeatureNotSupported_c – Host library was compiled without GATT server support.

bleResult_t GattServer_SendAttributeWrittenStatus(*deviceId_t* deviceId, uint16_t attributeHandle, uint8_t status)

Responds to an intercepted attribute write operation.

Remark

This function must be called by the application when receiving the gEvtAttributeWritten_c Server event. The status value may contain application- or profile-defined error codes.

Parameters

- deviceId – **[in]** The device ID of the connected peer.
- attributeHandle – **[in]** The attribute handle that was written.
- status – **[in]** The status of the write operation. If this parameter is equal to gAttErrCodeNoError_c then an ATT Write Response will be sent to the peer. Otherwise, an ATT Error Response with the provided status will be sent to the peer.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleFeatureNotSupported_c – Host library was compiled without GATT server support.

bleResult_t GattServer_RegisterHandlesForReadNotifications(uint8_t handleCount, const uint16_t *aAttributeHandles)

Registers the attribute handles that will be notified through the GATT Server callback when a GATT Client attempts to read the attributes' values.

Remark

The application may modify the attribute's value in the GATT Database before sending the response with GattServer_SendAttributeReadStatus.

Remark

This function executes synchronously.

Parameters

- handleCount – **[in]** Number of handles in array.
- aAttributeHandles – **[in]** Array of handles.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleOverflow_c – The maximum number of supported handles for read notifications has been reached.
- gBleFeatureNotSupported_c – Host library was compiled without GATT server support.

bleResult_t GattServer_UnregisterHandlesForReadNotifications(uint8_t handleCount, const uint16_t *aAttributeHandles)

Unregisters the attribute handles that will be notified through the GATT Server callback when a GATT Client attempts to read the attributes' values.

Remark

To unregister all the list, pass 0 count and NULL.

Remark

This function executes synchronously.

Parameters

- handleCount – **[in]** Number of handles in array.
- aAttributeHandles – **[in]** Array of handles.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleFeatureNotSupported_c – Host library was compiled without GATT server support.

bleResult_t GattServer_SendAttributeReadStatus(*deviceId_t* deviceId, uint16_t attributeHandle, uint8_t status)

Responds to an intercepted attribute read operation.

Remark

This function must be called by the application when receiving the gEvtAttributeRead_c Server event. The status value may contain application- or profile-defined error codes.

Parameters

- deviceId – **[in]** The device ID of the connected peer.
- attributeHandle – **[in]** The attribute handle that was being read.

- status – **[in]** The status of the read operation. If this parameter is equal to gAttErrCodeNoError_c then an ATT Read Response will be sent to the peer containing the attribute value from the GATT Database. Otherwise, an ATT Error Response with the provided status will be sent to the peer.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleFeatureNotSupported_c – Host library was compiled without GATT server support.

bleResult_t GattServer_SendNotification(*deviceId_t* deviceId, uint16_t handle)

Sends a notification to a peer GATT Client using the Characteristic Value from the GATT Database.

Parameters

- deviceId – **[in]** The device ID of the connected peer.
- handle – **[in]** Handle of the Value of the Characteristic to be notified.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleFeatureNotSupported_c – Host library was compiled without GATT server support.

bleResult_t GattServer_SendIndication(*deviceId_t* deviceId, uint16_t handle)

Sends an indication to a peer GATT Client using the Characteristic Value from the GATT Database.

Parameters

- deviceId – **[in]** The device ID of the connected peer.
- handle – **[in]** Handle of the Value of the Characteristic to be indicated.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleFeatureNotSupported_c – Host library was compiled without GATT server support.

bleResult_t GattServer_SendInstantValueNotification(*deviceId_t* deviceId, uint16_t handle, uint16_t valueLength, const uint8_t *aValue)

Sends a notification to a peer GATT Client with data given as parameter, ignoring the GATT Database.

Parameters

- deviceId – **[in]** The device ID of the connected peer.
- handle – **[in]** Handle of the Value of the Characteristic to be notified.
- valueLength – **[in]** Length of data to be notified.

- aValue – **[in]** Data to be notified.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOverflow_c – TX queue for device is full.
- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleFeatureNotSupported_c – Host library was compiled without GATT server support.

bleResult_t GattServer_SendMultipleHandleValueNotification(*deviceId_t* deviceId, uint32_t totalLength, const uint8_t *pHandleLengthValueList)

Sends a notification to a peer GATT Client with data given as parameter, ignoring the GATT Database.

Parameters

- deviceId – **[in]** The device ID of the connected peer.
- totalLength – **[in]** Length of the handle, value, value length tuples.
- pHandleLengthValueList – **[in]** Pointer to data to be notified.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleFeatureNotSupported_c – Host library was compiled without GATT server support.

bleResult_t GattServer_SendInstantValueIndication(*deviceId_t* deviceId, uint16_t handle, uint16_t valueLength, const uint8_t *aValue)

Sends an indication to a peer GATT Client with data given as parameter, ignoring the GATT Database.

Parameters

- deviceId – **[in]** The device ID of the connected peer.
- handle – **[in]** Handle of the Value of the Characteristic to be indicated.
- valueLength – **[in]** Length of data to be indicated.
- aValue – **[in]** Data to be indicated.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleOverflow_c – TX queue for device is full.
- gBleFeatureNotSupported_c – Host library was compiled without GATT server support.

bleResult_t GattServer_RegisterUniqueHandlesForNotifications(bool_t bWrite, bool_t bRead)

Registers all GATT DB dynamic attribute handles with unique value buffers to be notified through the GATT Server callback when a GATT Client attempts to read/write the attributes' values.

Remark

This function executes synchronously.

Remark

This function should be called when adding GATT DB unique value buffer characteristics or descriptors.

Parameters

- bWrite – **[in]** Enables/Disables write notifications.
- bRead – **[in]** Enables/Disables read notifications.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – Host library was compiled without GATT server support.

bleResult_t GattServer_RegisterEnhancedCallback(*gattServerEnhancedCallback_t* callback)

Installs an application callback for the Enhanced ATT GATT Server module.

Remark

This function executes synchronously.

Parameters

- callback – **[in]** Application-defined callback to be triggered by this module.

Return values

- gBleSuccess_c –
- gBleFeatureNotSupported_c – Host library was compiled without GATT server support.

bleResult_t GattServer_EnhancedSendAttributeWrittenStatus(*deviceId_t* deviceId, *bearerId_t* bearerId, uint16_t attributeHandle, uint8_t status)

Responds to an intercepted attribute write operation.

Remark

This function must be called by the application when receiving the gEvtAttributeWritten_c Server event. The status value may contain application- or profile-defined error codes.

Parameters

- deviceId – **[in]** The device ID of the connected peer.
- bearerId – **[in]** The Enhanced ATT bearer ID of the connected peer.
- attributeHandle – **[in]** The attribute handle that was written.
- status – **[in]** The status of the write operation. If this parameter is equal to gAttErrCodeNoError_c then an ATT Write Response will be sent to the peer. Else an ATT Error Response with the provided status will be sent to the peer.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleFeatureNotSupported_c – Host library was compiled without GATT server support.

bleResult_t GattServer_EnhancedSendAttributeReadStatus(*deviceId_t* deviceId, *bearerId_t* bearerId, uint16_t attributeHandle, uint8_t status)

Responds to an intercepted attribute read operation.

Remark

This function must be called by the application when receiving the gEvtAttributeRead_c Server event. The status value may contain application- or profile-defined error codes.

Parameters

- deviceId – **[in]** The device ID of the connected peer.
- bearerId – **[in]** The Enhanced ATT bearer ID of the connected peer.
- attributeHandle – **[in]** The attribute handle that was being read.
- status – **[in]** The status of the read operation. If this parameter is equal to gAttErrCodeNoError_c then an ATT Read Response will be sent to the peer containing the attribute value from the GATT Database. Else an ATT Error Response with the provided status will be sent to the peer.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleFeatureNotSupported_c – Host library was compiled without GATT server support.

bleResult_t GattServer_EnhancedSendNotification(*deviceId_t* deviceId, *bearerId_t* bearerId, uint16_t handle)

Sends a notification to a peer GATT Client using the Characteristic Value from the GATT Database.

Parameters

- deviceId – **[in]** The device ID of the connected peer.
- bearerId – **[in]** The Enhanced ATT bearer ID of the connected peer.
- handle – **[in]** Handle of the Value of the Characteristic to be notified.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleFeatureNotSupported_c – Host library was compiled without GATT server support.

bleResult_t GattServer_EnhancedSendIndication(*deviceId_t* deviceId, *bearerId_t* bearerId, uint16_t handle)

Sends an indication to a peer GATT Client using the Characteristic Value from the GATT Database.

Parameters

- deviceId – **[in]** The device ID of the connected peer.
- bearerId – **[in]** The Enhanced ATT bearer ID of the connected peer.
- handle – **[in]** Handle of the Value of the Characteristic to be indicated.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleFeatureNotSupported_c – Host library was compiled without GATT server support.

bleResult_t GattServer_EnhancedSendInstantValueNotification(*deviceId_t* deviceId, *bearerId_t* bearerId, uint16_t handle, uint16_t valueLength, const uint8_t *aValue)

Sends a notification to a peer GATT Client with data given as parameter, ignoring the GATT Database.

Parameters

- deviceId – **[in]** The device ID of the connected peer.
- bearerId – **[in]** The Enhanced ATT bearer ID of the connected peer.
- handle – **[in]** Handle of the Value of the Characteristic to be notified.
- valueLength – **[in]** Length of data to be notified.
- aValue – **[in]** Data to be notified.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.

- gBleOverflow_c – TX queue for device is full.
- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleFeatureNotSupported_c – Host library was compiled without GATT server support.

bleResult_t GattServer_EnhancedSendMultipleHandleValueNotification(*deviceId_t* deviceId, *bearerId_t* bearerId, uint32_t totalLength, const uint8_t *pHandleLengthValueList)

Sends a notification to a peer GATT Client with data given as parameter, ignoring the GATT Database.

Parameters

- deviceId – **[in]** The device ID of the connected peer.
- bearerId – **[in]** The Enhanced ATT bearer ID of the connected peer.
- totalLength – **[in]** Length of the handle, value, value length tuples.
- pHandleLengthValueList – **[in]** Pointer to data to be notified.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleFeatureNotSupported_c – Host library was compiled without GATT server support.

bleResult_t GattServer_EnhancedSendInstantValueIndication(*deviceId_t* deviceId, *bearerId_t* bearerId, uint16_t handle, uint16_t valueLength, const uint8_t *aValue)

Sends an indication to a peer GATT Client with data given as parameter, ignoring the GATT Database.

Parameters

- deviceId – **[in]** The device ID of the connected peer.
- bearerId – **[in]** The Enhanced ATT bearer ID of the connected peer.
- handle – **[in]** Handle of the Value of the Characteristic to be indicated.
- valueLength – **[in]** Length of data to be indicated.
- aValue – **[in]** Data to be indicated.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – An invalid parameter was provided.
- gBleOverflow_c – TX queue for device is full.
- gBleOutOfMemory_c – Could not allocate message for Host task.
- gBleFeatureNotSupported_c – Host library was compiled without GATT server support.

struct gattServerMtuChangedEvent_t

#include <gatt_server_interface.h> GATT Server MTU Changed Event structure

Public Members

uint16_t newMtu
Value of the agreed ATT_MTU for this connection.

struct gattServerAttributeWrittenEvent_t
#include <gatt_server_interface.h> GATT Server Attribute Written Event structure

Public Members

uint16_t handle
Handle of the attribute.

uint16_t cValueLength
Length of the attribute value array.

uint8_t *aValue
Attribute value array attempted to be written.

bearerId_t bearerId
Used by EATT. Send response on the same bearer. For ATT value is 0.

struct gattServerLongCharacteristicWrittenEvent_t
#include <gatt_server_interface.h> GATT Server Long Characteristic Written Event structure

Public Members

uint16_t handle
Handle of the Characteristic Value.

uint16_t cValueLength
Length of the value written.

uint8_t *aValue
Pointer to the attribute value in the database.

struct gattServerCccdWrittenEvent_t
#include <gatt_server_interface.h> GATT Server CCCD Written Event structure

Public Members

uint16_t handle
Handle of the CCCD attribute.

gattCccdFlags_t newCccd
New value of the CCCD.

struct gattServerAttributeReadEvent_t

#include <gatt_server_interface.h> GATT Server Attribute Read Event structure

Public Members

uint16_t handle

Handle of the attribute.

struct gattServerProcedureError_t

#include <gatt_server_interface.h> Server-initiated procedure error structure

Public Members

gattServerProcedureType_t procedureType

Procedure that generated error.

bleResult_t error

Error generated.

struct gattServerInvalidPdu_t

#include <gatt_server_interface.h> ATT PDU that generated the error

Public Members

attOpcode_t attOpCode

The invalid ATT op code that generated the error

struct gattServerEvent_t

#include <gatt_server_interface.h> GATT Server Event structure: type + data.

Public Members

gattServerEventType_t eventType

Event type.

union *gattServerEvent_t* eventData

Event data : selected according to event type.

union eventData

Public Members

gattServerMtuChangedEvent_t mtuChangedEvent

For event type gEvtMtuChanged_c: the new value of the ATT_MTU.

gattServerAttributeWrittenEvent_t attributeWrittenEvent

For event types gEvtAttributeWritten_c, gEvtAttributeWrittenWithoutResponse_c: handle and value of the attempted write.

gattServerCccdWrittenEvent_t charCccdWrittenEvent

For event type gEvtCharacteristicCccdWritten_c: handle and value of the CCCD.

gattServerProcedureError_t procedureError

For event type gEvtError_c: error that terminated a Server-initiated procedure.

gattServerLongCharacteristicWrittenEvent_t longCharWrittenEvent

For event type gEvtLongCharacteristicWritten_c: handle and value.

gattServerAttributeReadEvent_t attributeReadEvent

For event types gEvtAttributeRead_c: handle of the attempted read.

gattServerInvalidPdu_t attributeOpCode

For event type gEvtInvalidPduReceived_c: the ATT PDU that generated the error

GATT Database

enum gattCharacteristicPropertiesBitFields_tag

Values:

enumerator gGattCharPropNone_c

No Properties selected.

enumerator gGattCharPropBroadcast_c

Characteristic can be broadcast.

enumerator gGattCharPropRead_c

Characteristic can be read.

enumerator gGattCharPropWriteWithoutRsp_c

Characteristic can be written without response.

enumerator gGattCharPropWrite_c

Characteristic can be written with response.

enumerator gGattCharPropNotify_c

Characteristic can be notified.

enumerator gGattCharPropIndicate_c

Characteristic can be indicated.

enumerator gGattCharPropAuthSignedWrites_c
Deprecated. Not used.

enumerator gGattCharPropExtendedProperties_c
Extended Characteristic properties.

enum gattDbAccessType_t
Attribute access type

Values:

enumerator gAccessRead_c

enumerator gAccessWrite_c

enumerator gAccessNotify_c

typedef uint8_t gattCharacteristicPropertiesBitFields_t
Bit fields for Characteristic properties

typedef uint8_t gattAttributePermissionsBitFields_t
Bit fields for attribute permissions

uint16_t gGattDbAttributeCount_c
The number of attributes in the GATT Database.

gattDbAttribute_t *gattDatabase
Reference to the GATT database

uint16_t mServerServiceChangedCharHandle

uint16_t mServerServiceChangedCCCDHandle

uint32_t gGattDynamicAttrSize

uint8_t gaGattDynamicAttrBlob[]

uint32_t gGattDynamicValSize

uint8_t gaGattDynamicValBlob[]

uint16_t GattDb_GetIndexOfHandle(uint16_t handle)
Returns the database index for a given attribute handle.

Parameters

- handle – **[in]** The attribute handle.

Returns
The index of the given attribute in the database or gGattDbInvalidHandleIndex_d.

uint16_t GattDb_ServiceStartHandle(uint16_t handle)

Returns the handle of the service to which the given attribute belongs.

Parameters

- handle – **[in]** The attribute handle.

Returns

The handle of the service or gGattDbInvalidHandleIndex_d.

bleResult_t GattDb_FindServiceRange(uint16_t serviceHandle, uint16_t *pOutStartIndex, uint16_t *pOutAttributeCount)

Finds the start index and attribute count for a given service.

Parameters

- serviceHandle – **[in]** The service handle.
- pOutStartIndex – **[in]** The index in the database where the service declaration begins.
- pOutAttributeCount – **[in]** The number of attributes contained by the service.

Return values

- gBleSuccess_c –
- gGattDbInvalidHandleIndex_d – Service does not exist in gatt database.

uint16_t GattDb_GetAttributeValueSize(uint16_t handle)

Returns the value length for a given attribute handle.

Parameters

- handle – **[in]** The attribute handle.

Returns

The length of the value of the attribute at the given handle or maxVariableValueLength in case of variable length attributes.

bleResult_t GattDb_ComputeDatabaseHash(void)

Computes the database hash for a static or a dynamic database.

Parameters

- void. –

Return values

- gBleSuccess_c –
- gBleInvalidState_c – Database not initialized or empty.
- gBleOutOfMemory_c – Could not allocate memory for the database content used to compute the database hash.
- gGattDbInvalidHandleIndex_d – Service does not exist in gatt database.

bleResult_t GattDb_Init(void)

Initializes the GATT database at runtime.

Remark

This function should be called only once at device start-up. In the current stack implementation, it is called internally by Ble_HostInitialize.

Remark

This function executes synchronously.

Return values

- gBleSuccess_c –
- gBleOutOfMemory_c – Could not allocate memory for the GATT dynamic database.
- gBleAlreadyInitialized_c – GATT database already initialized.

bleResult_t GattDb_Deinit(void)

Function performing runtime deinitialization of the GATT database.

Remark

This function has effect for dynamic data base only .

Parameters

- none. –

Return values

- gBleSuccess_c –
- gBleInvalidState_c – GATT dynamic database not initialized.

bleResult_t GattDb_WriteAttribute(uint16_t handle, uint16_t valueLength, const uint8_t *aValue)

Writes an attribute from the application level.

This function can be called by the application code to modify an attribute in the database. It should only be used by the application to modify a Characteristic's value based on the application logic (e.g., external sensor readings).

Remark

This function executes synchronously.

Parameters

- handle – **[in]** The handle of the attribute to be written.
- valueLength – **[in]** The number of bytes to be written.
- aValue – **[in]** The source buffer containing the value to be written.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – aValue NULL and length higher than 0.
- gAttStatusBase_c – + gAttErrCodeInvalidHandle_c handle not found in GATT database.

- gGattInvalidValueLength_c – valueLength not equal to attribute value length, or higher than the maximum length for variable size attributes.
- gBleFeatureNotSupported_c – GATT Server not enabled.

bleResult_t GattDb_ReadAttribute(uint16_t handle, uint16_t maxBytes, uint8_t *aOutValue, uint16_t *pOutValueLength)

Reads an attribute from the application level.

This function can be called by the application code to read an attribute in the database.

Remark

This function executes synchronously.

Parameters

- handle – **[in]** The handle of the attribute to be read.
- maxBytes – **[in]** The maximum number of bytes to be received.
- aOutValue – **[out]** The pre-allocated buffer ready to receive the bytes.
- pOutValueLength – **[out]** The actual number of bytes received.

Return values

- gBleSuccess_c –
- gBleInvalidParameter_c – aOutValue NULL, pOutValueLength NULL or maxBytes 0.
- gAttStatusBase_c – + gAttErrCodeInvalidHandle_c handle not found in GATT database.
- gBleFeatureNotSupported_c – GATT Server not enabled.

bleResult_t GattDb_FindServiceHandle(uint16_t startHandle, *bleUuidType_t* serviceUuidType, const *bleUuid_t* *pServiceUuid, uint16_t *pOutServiceHandle)

Finds the handle of a Service Declaration with a given UUID inside the database.

Remark

This function executes synchronously.

Remark

The startHandle should be set to 0x0001 when this function is called for the first time. If multiple Services with the same UUID are expected, then after the first successful call the function may be called again with the startHandle equal to the found service handle plus one.

Parameters

- startHandle – **[in]** The handle to start the search. Should be 0x0001 on the first call.

- serviceUuidType – **[in]** Service UUID type.
- pServiceUuid – **[in]** Service UUID.
- pOutServiceHandle – **[out]** Pointer to the service declaration handle to be written.

Return values

- gBleSuccess_c – Service Declaration found, handle written in pOutCharValueHandle.
- gGattDbInvalidHandle_c – Invalid Start Handle.
- gBleInvalidParameter_c – pServiceUuid or pOutServiceHandle NULL.
- gGattDbServiceNotFound_c – Service with given UUID not found.
- gBleFeatureNotSupported_c – GATT Server not enabled.

bleResult_t GattDb_FindCharValueHandleInService(uint16_t serviceHandle, *bleUuidType_t* characteristicUuidType, const *bleUuid_t* *pCharacteristicUuid, uint16_t *pOutCharValueHandle)

Finds the handle of a Characteristic Value with a given UUID inside a Service.

The Service is input by its declaration handle.

Remark

This function executes synchronously.

Parameters

- serviceHandle – **[in]** The handle of the Service declaration.
- characteristicUuidType – **[in]** Characteristic UUID type.
- pCharacteristicUuid – **[in]** Characteristic UUID.
- pOutCharValueHandle – **[out]** Pointer to the characteristic value handle to be written.

Return values

- gBleSuccess_c – Characteristic Value found, handle written in pOutCharValueHandle.
- gGattDbInvalidHandle_c – Handle not found or not a Service declaration.
- gBleInvalidParameter_c – pCharacteristicUuid or pOutCharValueHandle NULL.
- gGattDbCharacteristicNotFound_c – Characteristic Value with given UUID not found.
- gBleFeatureNotSupported_c – GATT Server not enabled.

bleResult_t GattDb_FindCccdHandleForCharValueHandle(uint16_t charValueHandle, uint16_t *pOutCccdHandle)

Finds the handle of a Characteristic's CCCD given the Characteristic's Value handle.

Remark

This function executes synchronously.

Parameters

- charValueHandle – **[in]** The handle of the Service declaration.
- pOutCccdHandle – **[out]** Pointer to the CCCD handle to be written.

Return values

- gBleSuccess_c – CCCD found, handle written in pOutCccdHandle.
- gBleInvalidParameter_c – pOutCccdHandle is NULL.
- gGattDbInvalidHandle_c – Invalid Characteristic Value handle.
- gGattDbCccdNotFound_c – CCCD not found for this Characteristic.
- gBleFeatureNotSupported_c – GATT Server not enabled.

bleResult_t GattDb_FindDescriptorHandleForCharValueHandle(uint16_t charValueHandle, *bleUuidType_t* descriptorUuidType, const *bleUuid_t* *pDescriptorUuid, uint16_t *pOutDescriptorHandle)

Finds the handle of a Characteristic Descriptor given the Characteristic's Value handle and Descriptor's UUID.

Remark

This function executes synchronously.

Parameters

- charValueHandle – **[in]** The handle of the Service declaration.
- descriptorUuidType – **[in]** Descriptor's UUID type.
- pDescriptorUuid – **[in]** Descriptor's UUID.
- pOutDescriptorHandle – **[out]** Pointer to the Descriptor handle to be written.

Return values

- gBleSuccess_c – Descriptor found.
- gBleInvalidParameter_c – pDescriptorUuid or pOutDescriptorHandle are NULL.
- gGattDbInvalidHandle_c – Invalid Characteristic Value handle.
- gGattDbDescriptorNotFound_c – Descriptor not found for this Characteristic.
- gBleFeatureNotSupported_c – GATT Server not enabled.

gGattDbInvalidHandleIndex_d

Special value returned by GattDb_GetIndexOfHandle to signal that an invalid attribute handle was given as parameter.

gGattDbInvalidHandle_d

Special value used to mark an invalid attribute handle. Attribute handles are strictly positive.

gPermissionNone_c

No permissions selected.

gPermissionFlagReadable_c

Attribute can be read.

gPermissionFlagReadWithEncryption_c

Attribute may be read only if link is encrypted.

gPermissionFlagReadWithAuthentication_c

Attribute may be read only by authenticated peers.

gPermissionFlagReadWithAuthorization_c

Attribute may be read only by authorized peers.

gPermissionFlagWritable_c

Attribute can be written.

gPermissionFlagWriteWithEncryption_c

Attribute may be written only if link is encrypted.

gPermissionFlagWriteWithAuthentication_c

Attribute may be written only by authenticated peers.

gPermissionFlagWriteWithAuthorization_c

Attribute may be written only by authorized peers.

gGattDatabaseHashSize_c

struct gattDbAttribute_t

#include <gatt_database.h> Attribute structure

Public Members

uint16_t handle

The attribute handle - cannot be 0x0000. The attribute handles need not be consecutive, but must be strictly increasing.

uint16_t permissions

Attribute permissions as defined by the ATT.

uint32_t uuid

The UUID should be read according to the *gattDbAttribute_t.uuidType* member: for 2-byte and 4-byte UUIDs, this contains the value of the UUID; for 16-byte UUIDs, this is a pointer to the allocated 16-byte array containing the UUID.

uint8_t *pValue
A pointer to allocated value array.

uint16_t valueLength
The size of the value array.

uint16_t uuidType
Identifies the length of the UUID; values interpreted according to the *bleUuidType_t* enumeration.

uint16_t maxVariableValueLength
The maximum length of the attribute value array; if this is set to 0, then the attribute's length is fixed and cannot be changed.

Bluetooth Low Energy

enum bleResult_tag
BLE result type - the return value of BLE API functions

Values:

enumerator gBleStatusBase_c
General status base.

enumerator gBleSuccess_c
Function executed successfully.

enumerator gBleInvalidParameter_c
Parameter has an invalid value or is outside the accepted range.

enumerator gBleOverflow_c
An internal limit is reached.

enumerator gBleUnavailable_c
A requested parameter is not available.

enumerator gBleFeatureNotSupported_c
The requested feature is not supported by this stack version.

enumerator gBleOutOfMemory_c
An internal memory allocation failed.

enumerator gBleAlreadyInitialized_c
Ble_HostInitialize function is incorrectly called a second time.

enumerator gBleOsError_c
An error occurred at the OS level.

enumerator gBleUnexpectedError_c
A "should never get here"-type error occurred.

enumerator gBleInvalidState_c
The requested API cannot be called in the current state.

enumerator gBleTimerError_c
Timer allocation failed.

enumerator gBleReassemblyInProgress_c
HCI Packet reassembly was in progress. The old packet was discarded

enumerator gBleNVMError_c
An error occurred when performing an NVM operation

enumerator gBleRngError_c
error generated by a RNG function call e.g. RNG_GetPseudoRandomNo

enumerator gBleSecLibError_c
error generated by a SecLib function call e.g. SecLib_VerifyBluetoothAh

enumerator gHciStatusBase_c
HCI status base.

enumerator gHciSuccess_c
Alias.

enumerator gHciUnknownHciCommand_c
The HCI Command packet opcode is unknown.

enumerator gHciUnknownConnectionIdentifier_c
The connection does not exist or it is of a wrong type.

enumerator gHciHardwareFailure_c
A failure occurred in the Controller.

enumerator gHciPageTimeout_c
A page timed out based on the Page Timeout parameter.

enumerator gHciAuthenticationFailure_c
Pairing or authentication failed. Possible causes: incorrect PIN or Link Key.

enumerator gHciPinOrKeyMissing_c
Pairing failed because of missing PIN or authentication failed because of missing Key.

enumerator gHciMemoryCapacityExceeded_c
The controller doesn't have enough memory to store new parameters.

enumerator gHciConnectionTimeout_c
The link supervision timeout has expired for a connection or the synchronization timeout has expired for a broadcast.

enumerator gHciConnectionLimitExceeded_c

The maximum number of connections has already been reached.

enumerator gHciSynchronousConnectionLimitToADeviceExceeded_c

The maximum number of synchronous connections has already been reached.

enumerator gHciAclConnectionAlreadyExists_c

A connection to this device already exists.

enumerator gHciCommandDisallowed_c

The controller can't process the command at this time.

enumerator gHciConnectionRejectedDueToLimitedResources_c

The connection was rejected due to limited resources.

enumerator gHciConnectionRejectedDueToSecurityReasons_c

The connection was rejected because the security requirements were not fulfilled.

enumerator gHciConnectionRejectedDueToUnacceptableBdAddr_c

The connection was rejected because the device does not accept the given BD_ADDR. A possible cause is that the device only accepts connections from specific BD_ADDRs.

enumerator gHciConnectionAcceptTimeoutExceeded_c

The connection accept timeout for this connection has been exceeded.

enumerator gHciUnsupportedFeatureOrParameterValue_c

The feature or parameter in the HCI command is not supported.

enumerator gHciInvalidHciCommandParameters_c

At least one of the HCI command parameters is invalid.

enumerator gHciRemoteUserTerminatedConnection_c

The remote device's user terminated the connection or stopped broadcasting.

enumerator gHciRemoteDeviceTerminatedConnectionLowResources_c

The remote device terminated the connection due to low resources.

enumerator gHciRemoteDeviceTerminatedConnectionPowerOff_c

The remote device terminated the connection because it will power off.

enumerator gHciConnectionTerminatedByLocalHost_c

The local device terminated the connection, synchronization with a broadcaster or stopped broadcasting.

enumerator gHciRepeatedAttempts_c

The Controller disallows the authentication or pairing because not enough time has passed since the failed attempt.

enumerator gHciPairingNotAllowed_c

The device does not allow pairing.

enumerator gHciUnknownLpmPdu_c

The Controller has received an unknown LMP code.

enumerator gHciUnsupportedRemoteFeature_c

The remote device doesn't support the feature associated with the issued command, LMP PDU or LL Control PDU.

enumerator gHciScoOffsetRejected_c

The offset requested in the LMP_SCO_LINK_REQ PDU has been rejected.

enumerator gHciScoIntervalRejected_c

The interval requested in the LMP_SCO_LINK_REQ PDU has been rejected.

enumerator gHciScoAirModeRejected_c

The air mode requested in the LMP_SCO_LINK_REQ PDU has been rejected.

enumerator gHciInvalidLpmParameters_c

Some LMP PDU or LL Control PDU parameters were invalid.

enumerator gHciUnspecifiedError_c

No other error code was appropriate.

enumerator gHciUnsupportedLpmParameterValue_c

An LMP PDU or LL Control PDU contains at least one parameter value not supported by the Controller at this time.

enumerator gHciRoleChangeNotAllowed_c

The Controller doesn't allow a role change at this time.

enumerator gHciLLResponseTimeout_c

An LMP transaction failed to respond within the LMP response timeout or an LL transaction failed to respond within the LL response timeout.

enumerator gHciLmpErrorTransactionCollision_c

An LMP transaction or LL procedure has collided with the same transaction or procedure which is already in progress.

enumerator gHciLmpPduNotAllowed_c

A controller sent an LMP PDU with an opcode that isn't allowed.

enumerator gHciEncryptionModeNotAcceptable_c

The requested encryption mode is not allowed at this time.

enumerator gHciLinkKeyCannotBeChanged_c

The link key can't be changed because a fixed unit key is being used.

enumerator gHciRequestedQosNotSupported_c

The requested QoS is not supported.

enumerator gHciInstantPassed_c

An LMP PDU or LL PDU that includes an instant cannot be performed because the instant when this would have occurred has passed.

enumerator gHciPairingWithUnitKeyNotSupported_c

It wasn't possible to pair because a unit key was requested but it is not supported.

enumerator gHciDifferentTransactionCollision_c

An LMP transaction or LL Procedure was started that collides with another one.

enumerator gHciReserved_0x2B_c

enumerator gHciQosNotAcceptableParameter_c

The specified QoS parameters couldn't be accepted at this time.

enumerator gHciQosRejected_c

The specified QoS parameters couldn't be accepted and QoS negotiation should be terminated.

enumerator gHciChannelClassificationNotSupported_c

The Controller can't perform a channel assessment because it isn't supported.

enumerator gHciInsufficientSecurity_c

The HCI command or LMP PDU sent is only possible on an encrypted link.

enumerator gHciParameterOutOfMandatoryRange_c

A parameter value requested is outside the mandatory range of parameters for the given HCI command or LMP PDU. The recipient does not accept that value.

enumerator gHciReserved_0x31_c

enumerator gHciRoleSwitchPending_c

A role switch is pending.

enumerator gHciReserved_0x33_c

enumerator gHciReservedSlotViolation_c

The Synchronous negotiation was terminated with its state set to Reserved Slot Violation.

enumerator gHciRoleSwitchFailed_c

A role switch was attempted but it failed. The original piconet structure is restored.

enumerator gHciExtendedInquiryResponseTooLarge_c

The extended inquiry response, with the requested requirements for FEC, is too large to fit in any of the packet types supported by the Controller.

enumerator gHciSecureSimplePairingNotSupportedByHost_c

The IO capabilities request / response was rejected because the sending Host does not support Secure Simple Pairing even though the receiving Link Manager does.

enumerator gHciHostBusyPairing_c

The Host is busy with another pairing request. Pairing should be retried again later.

enumerator gHciConnectionRejectedDueToNoSuitableChannelFound_c

An appropriate value for the Channel selection operation couldn't be calculated by the Controller.

enumerator gHciControllerBusy_c

The operation was rejected because the Controller was busy.

enumerator gHciUnacceptableConnectionParameters_c

The remote device terminated the connection or rejected a request because of at least one unacceptable connection parameter.

enumerator gHciDirectedAdvertisingTimeout_c

Advertising for a fixed duration completed. For directed advertising, advertising completed without a connection being created.

enumerator gHciConnectionTerminatedDueToMicFailure_c

The connection or the synchronization was terminated because the Message Integrity Check failed.

enumerator gHciConnectionFailedToBeEstablishedOrSyncTimeout_c

The LL initiated a connection / synchronization to periodic advertising but the connection has failed to be established or the LL failed to synchronize with the periodic advertising within 6 periodic advertising events of the first attempt.

enumerator gHciMacConnectionFailed_c

enumerator gHciCoarseClockAdjustmentRejected_c

The Central is unable to make a coarse adjustment to the piconet clock, using the given parameters. It will attempt to move the clock using clock dragging.

enumerator gHciType0SubmapNotDefined_c

The LMP PDU is rejected because the Type 0 submap is not currently defined.

enumerator gHciUnknownAdvertisingIdentifier_c

A command was sent from the Host that should identify an Advertising or Sync handle, but that handle doesn't exist.

enumerator gHciLimitReached_c

The number of operations requested has been reached.

enumerator gHciOperationCancelledByHost_c

A request sent from the Host to the Controller which was still pending has been successfully canceled.

enumerator gHciPacketTooLong_c

An attempt was made to send or receive a packet that exceeds the maximum allowed packet length.

enumerator gHciPacketTooLate_c

Information was provided too late to the Controller.

enumerator gHciPacketTooEarly_c

Information was provided too early to the Controller.

enumerator gHciAlreadyInit_c

HCI has already been initialized.

enumerator gHciInvalidParameter_c

At least one of the HCI command parameters is invalid.

enumerator gHciCallbackNotInstalled_c

Callback was not installed.

enumerator gHciCallbackAlreadyInstalled_c

Callback has already been installed.

enumerator gHciCommandNotSupported_c

The HCI command is not supported.

enumerator gHciEventNotSupported_c

The HCI event is not supported.

enumerator gHciTransportError_c

HCI layer initialization failure. HCI data buffering semaphore allocation error. HCI layer received an unknown packet type. Serial initialization or write failure.

enumerator gCtrlStatusBase_c

Controller status base.

enumerator gCtrlSuccess_c

Alias.

enumerator gL2caStatusBase_c

L2CAP status base.

enumerator gL2caSuccess_c

Alias.

enumerator gL2caAlreadyInit_c

L2CAP has already been initialized.

enumerator gL2caInsufficientResources_c

L2CAP could not allocate resources to perform operations (memory or timers).

enumerator gL2caCallbackNotInstalled_c
Callback was not installed.

enumerator gL2caCallbackAlreadyInstalled_c
Callback has already been installed.

enumerator gL2caLePsmInvalid_c
Invalid LE_PSM value.

enumerator gL2caLePsmAlreadyRegistered_c
LE_PSM has already been registered.

enumerator gL2caLePsmNotRegistered_c
LE_PSM value was not registered.

enumerator gL2caLePsmInsufficientResources_c
No free LE_PSM registration slot was found.

enumerator gL2caChannelInvalid_c
L2CA channel is invalid.

enumerator gL2caChannelClosed_c
L2CA channel is closed.

enumerator gL2caChannelAlreadyConnected_c
L2CA channel is already connected.

enumerator gL2caConnectionParametersRejected_c
Connection parameters were rejected.

enumerator gL2caChannelBusy_c
L2CA channel is busy.

enumerator gL2caInvalidParameter_c
The command contains at least one invalid parameter.

enumerator gL2caInternalError_c
There are connected L2CAP channels for the LE_PSM.

enumerator gSmStatusBase_c
Security Manager status base.

enumerator gSmSuccess_c
Alias.

enumerator gSmNullCBFunction_c

enumerator gSmCommandNotSupported_c

The Security Manager (SM) does not have the required features or version to support this command

enumerator gSmUnexpectedCommand_c

This command is not or cannot be handled in the current context of the SM.

enumerator gSmInvalidCommandCode_c

The provided SM command code is invalid.

enumerator gSmInvalidCommandLength_c

The provided command length is not valid for the SM command code.

enumerator gSmInvalidCommandParameter_c

One of the parameters of the SM command is not valid.

enumerator gSmInvalidDeviceId_c

The provided Device ID is invalid.

enumerator gSmInvalidInternalOperation_c

There is a problem with the internal state of the SM. This should not happen during normal operation. A memory corruption or invalid operation may have occurred.

enumerator gSmInvalidConnectionHandle_c

The target device does not have a valid connection handle. It might be disconnected.

enumerator gSmInproperKeyDistributionField_c

The Responder upper layer has set to "1" one or more flags in the Initiator or Responder Key Distribution Fields from the Pairing Request which were set to "0" by the peer device.

enumerator gSmUnexpectedKeyType_c

The Responder upper layer has set a key type field in the Passkey Request Reply command, which is different than the field negotiated with the peer device.

enumerator gSmUnexpectedPairingTerminationReason_c

The upper layer tried to cancel the pairing procedure with an unexpected pairing failure reason for the current phase of the pairing procedure.

enumerator gSmUnexpectedKeyset_c

The Responder upper layer is trying to distribute keys which were not requested during the pairing procedure or the peer device has sent a Key Distribution packet which was not expected.

enumerator gSmSmpTimeoutOccurred_c

An SMP timeout has occurred for the peer device. No more operations are accepted until a new physical link is established.

enumerator gSmUnknownSmpPacketType_c

An SMP packet with an unknown (or invalid) type has been received.

enumerator gSmInvalidSmpPacketLength_c

An SMP packet with an invalid length for the SMP packet type has been received.

enumerator gSmInvalidSmpPacketParameter_c

An SMP packet with an invalid parameter has been received.

enumerator gSmReceivedUnexpectedSmpPacket_c

An unexpected SMP packet was received.

enumerator gSmReceivedSmpPacketFromUnknownDevice_c

An SMP packet is received but the source Device ID cannot be identified.

enumerator gSmReceivedUnexpectedHciEvent_c

An HCI event has been received which cannot be handled by the SM or cannot be handled in the current context.

enumerator gSmReceivedHciEventFromUnknownDevice_c

An HCI event is received but the source Device ID cannot be identified.

enumerator gSmInvalidHciEventParameter_c

An HCI Event is received with an invalid parameter.

enumerator gSmLlConnectionEncryptionInProgress_c

A Link Layer Connection encryption was requested by the upper layer or attempted internally by the SM, but it could no be completed because an encryption was already in progress. This situation could lead to an SMP Pairing Failure when the SM cannot encrypt the link with the STK. An unspecified pairing failure reason is used in this instance.

enumerator gSmLlConnectionEncryptionFailure_c

The Link Layer connection encryption procedure has failed.

enumerator gSmInsufficientResources_c

The SM could not allocate resources to perform operations (memory or timers).

enumerator gSmOobDataAddressMismatch_c

The address of the peer contained in the remote OOB data sent to the stack does not match the address used by the remote device for the connection/pairing procedure.

enumerator gSmSmpPacketReceivedAfterTimeoutOccurred_c

A SMP packet has been received from a peer device for which a pairing procedure has timed out. No further operations are permitted until a new connection is established.

enumerator gSmReceivedTimerEventForUnknownDevice_c

An Timer event is received but the source Device ID cannot be identified.

enumerator gSmUnattainableLocalDeviceSecRequirements_c

The provided pairing parameters cannot lead to a Pairing Procedure which satisfies the minimum security properties for the local device.

enumerator gSmUnattainableLocalDeviceMinKeySize_c

The provided pairing parameters cannot lead to a Pairing Procedure which satisfies the minimum encryption key size for the local device.

enumerator gSmUnattainablePeripheralSecReqRequirements_c

The provided pairing parameters cannot lead to a Pairing Procedure which satisfies the minimum security properties requested by the local device via an SMP Peripheral Security Request.

enumerator gSmInvalidPeerPublicKey_c

This status covers the case where the peer provides a public key with an identical X coordinate to our own.

enumerator gSmPairingErrorPasskeyEntryFailed_c

The passkey entry failed.

enumerator gSmPairingErrorConfirmValueFailed_c

The received random confirm value does not match the computed random confirm value.

enumerator gSmPairingErrorCommandNotSupported_c

The command is not supported.

enumerator gSmPairingErrorInvalidParameters_c

The command contains at least one invalid parameter.

enumerator gSmPairingErrorUnknownReason_c

Unknown pairing failure reason.

enumerator gSmPairingErrorTimeout_c

The pairing procedure timed out.

enumerator gSmPairingErrorAuthenticationRequirements_c

Authentication requirements were not met.

enumerator gSmPairingAlreadyStarted_c

The pairing process has already started.

enumerator gSmPairingErrorKeyRejected_c

The key was rejected.

enumerator gSmPairingErrorPairingNotSupported_c

Pairing is not supported by the device.

enumerator gSmPairingErrorEncryptionKeySize_c

The resultant encryption key size is not long enough for the security requirements of this device.

enumerator gSmPairingErrorDhKeyCheckFailed_c

DHKey Check value received doesn't match the one calculated by the local device.

enumerator gSmPairingErrorNumericComparisonFailed_c

Indicates that the confirm values in the numeric comparison protocol do not match.

enumerator gSmPairingErrorOobNotAvailable_c

Indicates that the OOB data is not available.

enumerator gSmPairingErrorRepeatedAttempts_c

Indicates that the pairing or authentication procedure is disallowed because too little time has elapsed since last pairing request or security request.

enumerator gSmPairingErrorBusy_c

Indicates that the device is not ready to perform a pairing procedure

enumerator gSmTbResolvableAddressDoesNotMatchIrk_c

The provided Resolvable Private Address and IRK do not match.

enumerator gSmKeySessionKeyDerivationFailed_c

The session key derivation failed.

enumerator gAttStatusBase_c

ATT status base.

enumerator gAttSuccess_c

Alias.

enumerator gGattStatusBase_c

GATT status base.

enumerator gGattSuccess_c

Alias.

enumerator gGattAnotherProcedureInProgress_c

Trying to start a GATT procedure while one is already in progress.

enumerator gGattLongAttributePacketsCorrupted_c

Writing a Long Characteristic failed because Prepare Write Request packets were corrupted.

enumerator gGattMultipleAttributesOverflow_c

Too many Characteristics are given for a Read Multiple Characteristic procedure.

enumerator gGattUnexpectedReadMultipleResponseLength_c

Read Multiple Characteristic procedure failed because unexpectedly long data was read.

enumerator gGattInvalidValueLength_c

An invalid value length was supplied to a Characteristic Read/Write operation.

enumerator gGattServerTimeout_c

No response was received from the Server.

enumerator gGattIndicationAlreadyInProgress_c

A Server Indication is already waiting for Client Confirmation.

enumerator gGattClientConfirmationTimeout_c

No Confirmation was received from the Client after a Server Indication.

enumerator gGattInvalidPduReceived_c

An invalid PDU length was received.

enumerator gGattPeerDisconnected_c

An ongoing GATT procedure could not be finished due to peer's disconnection.

enumerator gGattMtuExchangeInProgress_c

A Server Indication is already waiting for Client Confirmation.

enumerator gGattOutOfSyncProceduresOngoing_c

Client can't read the Database Hash until all pending procedures are done.

enumerator gGattConnectionSecurityRequirementsNotMet_c

Client cannot initiate communication if the device security requirements are not met, e.g., link is not yet encrypted/authenticated.

enumerator gGapStatusBase_c

GAP status base.

enumerator gGapSuccess_c

Alias.

enumerator gGapAdvDataTooLong_c

Trying to set too many bytes in the advertising payload.

enumerator gGapScanRspDataTooLong_c

Trying to set too many bytes in the scan response payload.

enumerator gGapDeviceNotBonded_c

Trying to execute an API that is only available for bonded devices.

enumerator gGapAnotherProcedureInProgress_c

Trying to start a GAP procedure while one is already in progress.

enumerator gDevDbStatusBase_c

DeviceDatabase status base.

enumerator gDevDbSuccess_c

Alias.

enumerator gDevDbCccdLimitReached_c
CCCD value cannot be saved because Server's CCCD list is full for the current client.

enumerator gDevDbCccdNotFound_c
CCCD with the given handle is not found in the Server's list for the current client.

enumerator gGattDbStatusBase_c
GATT Database status base.

enumerator gGattDbSuccess_c
Alias.

enumerator gGattDbInvalidHandle_c
An invalid handle was passed as parameter.

enumerator gGattDbCharacteristicNotFound_c
Characteristic was not found.

enumerator gGattDbCccdNotFound_c
CCCD was not found.

enumerator gGattDbServiceNotFound_c
Service Declaration was not found.

enumerator gGattDbDescriptorNotFound_c
Characteristic Descriptor was not found.

enumerator gGattDbServiceOrCharAlreadyDeclared_c
Service or characteristic already declared

enumerator gCsStatusBase_c
Channel Sounding status base.

enumerator gCsSuccess_c
Alias.

enumerator gCsCallbackAlreadyInstalled_c
The selected callback was already installed.

enumerator gCsSecurityCheckFail_c
GAP Mode 4 security check fail.

enum bearerStatus_tag
Enhanced ATT bearer status values

Values:

enumerator gEattBearerActive_c

enumerator gEattBearerReconfInProgress_c

enumerator gEattBearerSuspended_c

enumerator gEattBearerAllocated_c

enumerator gEattBearerFree_c

enumerator gEattBearerAlreadyAllocated_c

enum bleAdvertisingType_t

Advertising Type

Values:

enumerator gAdvConnectableUndirected_c

Answers to both connect and scan requests.

enumerator gAdvDirectedHighDutyCycle_c

Answers only to connect requests; smaller advertising interval for quicker connection.

enumerator gAdvScannable_c

Answers only to scan requests.

enumerator gAdvNonConnectable_c

Does not answer to connect nor scan requests.

enumerator gAdvDirectedLowDutyCycle_c

Answers only to connect requests; larger advertising interval.

enum bleAdvReportEventProperties_tag

Values:

enumerator gAdvEventConnectable_c

Connectable Advertisement

enumerator gAdvEventScannable_c

Scannable Advertisement

enumerator gAdvEventDirected_c

Directed Advertisement

enumerator gAdvEventScanResponse_c

Scan Response

enumerator gAdvEventLegacy_c

Legacy Advertisement PDU

enumerator gAdvEventAnonymous_c
Anonymous Advertisement

enum bleAdvRequestProperties_tag
Values:

enumerator gAdvReqConnectable_c
Connectable Advertising

enumerator gAdvReqScannable_c
Scannable Advertising

enumerator gAdvReqDirected_c
Directed Advertising

enumerator gAdvReqHighDutyCycle_c
High Duty Cycle

enumerator gAdvReqLegacy_c
Legacy Advertising PDU

enumerator gAdvReqAnonymous_c
Anonymous Advertising

enumerator gAdvIncludeTxPower_c
Set this option to include the Tx power in advertising packet.

enumerator gAdvUseDecisionPDU_c
Use ADV_DECISION_IND PDU(0b1001).

enumerator gAdvIncludeAdvAinDecisionPDU_c
Include AdvA in the extended header of all decision PDUs.

enumerator gAdvIncludeADIinDecisionPDU_c
Include ADI in the extended header of all decision PDUs.

enum bleAdvertisingFilterPolicy_t
Values:

enumerator gBleAdvFilterAllowScanFromAnyAllowConnFromAny_c
Filter Accept List is ignored.

enumerator gBleAdvFilterAllowScanFromWLAllowConnFromAny_c
Filter Accept List is used only for Scan Requests.

enumerator gBleAdvFilterAllowScanFromAnyAllowConnFromWL_c
Filter Accept List is used only for Connection Requests.

enumerator gBleAdvFilterAllowScanFromWLAllowConnFromWL_c
Filter Accept List is used for both Scan and Connection Requests.

enum bleLlConnectionRole_t
Values:

enumerator gBleLlConnectionCentral_c
Link Layer Central Role

enumerator gBleLlConnectionPeripheral_c
Link Layer Peripheral Role

enum gapLeModulationTestMode_t
LE Modulation Test Mode values
Values:

enumerator gLeModulationTestDisable_c
Disable modulation test

enumerator gLeModulationTestUnmodulated_c
Unmodulated signal

enumerator gLeModulationTestModulatedZeros_c
Modulated with 0's

enumerator gLeModulationTestModulatedOnes_c
Modulated with 1's

enumerator gLeModulationTestModulatedPRBS_c
Modulated with pseudo-random

enumerator gLeModulationTestRssiMeasurement_c
RSSI measurement

enum bleCentralClockAccuracy_tag
Values:

enumerator gBleCentralClkAcc500ppm_c

enumerator gBleCentralClkAcc250ppm_c

enumerator gBleCentralClkAcc150ppm_c

enumerator gBleCentralClkAcc100ppm_c

enumerator gBleCentralClkAcc75ppm_c

enumerator gBleCentralClkAcc50ppm_c

enumerator gBleCentralClkAcc30ppm_c

enumerator gBleCentralClkAcc20ppm_c

enum bleAdvertiserClockAccuracy_tag

Values:

enumerator gBleAdvertiserClkAcc500ppm_c

enumerator gBleAdvertiserClkAcc250ppm_c

enumerator gBleAdvertiserClkAcc150ppm_c

enumerator gBleAdvertiserClkAcc100ppm_c

enumerator gBleAdvertiserClkAcc75ppm_c

enumerator gBleAdvertiserClkAcc50ppm_c

enumerator gBleAdvertiserClkAcc30ppm_c

enumerator gBleAdvertiserClkAcc20ppm_c

enum bleAdvertisingReportEventType_t

Values:

enumerator gBleAdvRepAdvInd_c

enumerator gBleAdvRepAdvDirectInd_c

enumerator gBleAdvRepAdvScanInd_c

enumerator gBleAdvRepAdvNonconnInd_c

enumerator gBleAdvRepScanRsp_c

enum hciPacketType_t

Values:

enumerator gHciCommandPacket_c

HCI Command

enumerator gHciDataPacket_c

L2CAP Data Packet

enumerator gHciSynchronousDataPacket_c
Not used in BLE

enumerator gHciEventPacket_c
HCI Event

enumerator gHciIsoDataPacket_c
HCI ISO data packet

enum bleScanType_t
Scanning type enumeration.

Values:

enumerator gScanTypePassive_c
Passive Scanning - advertising packets are immediately reported to the Host.

enumerator gScanTypeActive_c
Active Scanning - the scanner sends scan requests to the advertiser and reports to the Host after the scan response is received.

enum bleTransmitPowerLevelType_t
Values:

enumerator gReadCurrentTxPowerLevel_c
Current TX Power level.

enumerator gReadMaximumTxPowerLevel_c
Maximum recorded TX Power level.

enum bleTransmitPowerChannelType_t
Values:

enumerator gTxPowerAdvChannel_c
Advertising channel type when setting Tx Power.

enumerator gTxPowerConnChannel_c
Connection channel type when setting Tx Power.

enum bleChannelOverrideMode_t
Values:

enumerator gOverrideAdvChannels_c
Set the advertising channels

enumerator gOverrideScanChannels_c
Set the scanning channels

enumerator gOverrideInitChannels_c
Set the connection initiation channels

enum bleChannelFrequency_t

Values:

enumerator gBleFreq2402MHz_c

enumerator gBleFreq2404MHz_c

enumerator gBleFreq2406MHz_c

enumerator gBleFreq2408MHz_c

enumerator gBleFreq2410MHz_c

enumerator gBleFreq2412MHz_c

enumerator gBleFreq2414MHz_c

enumerator gBleFreq2416MHz_c

enumerator gBleFreq2418MHz_c

enumerator gBleFreq2420MHz_c

enumerator gBleFreq2422MHz_c

enumerator gBleFreq2424MHz_c

enumerator gBleFreq2426MHz_c

enumerator gBleFreq2428MHz_c

enumerator gBleFreq2430MHz_c

enumerator gBleFreq2432MHz_c

enumerator gBleFreq2434MHz_c

enumerator gBleFreq2436MHz_c

enumerator gBleFreq2438MHz_c

enumerator gBleFreq2440MHz_c

enumerator gBleFreq2442MHz_c

enumerator gBleFreq2444MHz_c

enumerator gBleFreq2446MHz_c

enumerator gBleFreq2448MHz_c

enumerator gBleFreq2450MHz_c

enumerator gBleFreq2452MHz_c

enumerator gBleFreq2454MHz_c

enumerator gBleFreq2456MHz_c

enumerator gBleFreq2458MHz_c

enumerator gBleFreq2460MHz_c

enumerator gBleFreq2462MHz_c

enumerator gBleFreq2464MHz_c

enumerator gBleFreq2466MHz_c

enumerator gBleFreq2468MHz_c

enumerator gBleFreq2470MHz_c

enumerator gBleFreq2472MHz_c

enumerator gBleFreq2474MHz_c

enumerator gBleFreq2476MHz_c

enumerator gBleFreq2478MHz_c

enumerator gBleFreq2480MHz_c

enum bleTxTestPacketPayload_t

Values:

enumerator gBleTestPacketPayloadPrbs9_c

enumerator gBleTestPacketPayloadPattern11110000_c

enumerator gBleTestPacketPayloadPattern10101010_c

enumerator gBleTestPacketPayloadPrbs15_c

enumerator gBleTestPacketPayloadPatternAllBits1_c

enumerator gBleTestPacketPayloadPatternAllBits0_c

enumerator gBleTestPacketPayloadPattern00001111_c

enumerator gBleTestPacketPayloadPattern01010101_c

enum bleHardwareErrorCode_t

Values:

enumerator bleHwErrCodeNoError_c

enum leSupportedFeatures_tag

Values:

enumerator gLeEncryption_c

enumerator gLeConnectionParametersRequestProcedure_c

enumerator gLeExtendedRejectIndication_c

enumerator gLePeripheralInitiatedFeaturesExchange_c

enumerator gLePing_c

enumerator gLeDataPacketLengthExtension_c

enumerator gLeLlPrivacy_c

enumerator gLeExtendedScannerFilterPolicies_c

enumerator gLe2MbPhy_c

enumerator gLeStableModulationIdxTx_c

enumerator gLeStableModulationIdxRx_c

enumerator gLeCodedPhy_c

enumerator gLeExtendedAdv_c

enumerator gLePeriodicAdv_c

enumerator gLeChannelSelAlg2_c

enumerator gLePowerClass1_c

enumerator gLeMinNumOfUsedChanProcedure_c

enumerator gLePeriodicAdvSyncTransferSender_c

enumerator gLePeriodicAdvSyncTransferReceiver_c

enumerator gLePowerControlRequest1_c

enumerator gLePowerControlRequest2_c

enumerator gLePathLossMonitoring_c

enumerator gLeConnectionSubrating_c

enumerator gLeConnectionSubratingHostSupport_c

enumerator gAdvertisingCodingSelection_c

enumerator gAdvertisingCodingSelectionHostSupport_c

enumerator gLeDecisionBasedAdvertisingFiltering_c

enumerator gLePawrAdvertiser_c

enumerator gLePawrScanner_c

enumerator gLeChannelSounding_c

enumerator gLeChannelSoundingHostSupport_c

enumerator gLeExtendedFeatures_c

enum leExtendedSupportedFeatures_tag

Values:

enumerator gLeMonitoringAdvertisers_c

enumerator gCsEnhancements1_c

enum gapGenericEventType_t

Generic Event Type

Values:

enumerator gInitializationComplete_c

Initial setup started by Ble_HostInitialize is complete.

enumerator gInternalError_c

An internal error occurred.

enumerator gAdvertisingSetupFailed_c

Error during advertising setup.

enumerator gAdvertisingParametersSetupComplete_c

Advertising parameters have been successfully set. Response to Gap_SetAdvertisingParameters.

enumerator gAdvertisingDataSetupComplete_c

Advertising and/or scan response data has been successfully set. Response to Gap_SetAdvertisingData.

enumerator gFilterAcceptListSizeRead_c

Contains the Filter Accept List size. Response to Gap_ReadFilterAcceptListSize.

enumerator gDeviceAddedToFilterAcceptList_c

Device has been added to Filter Accept List. Response to Gap_AddDeviceToFilterAcceptList.

enumerator gDeviceRemovedFromFilterAcceptList_c

Device has been removed from the Filter Accept List. Response to Gap_RemoveDeviceFromFilterAcceptList.

enumerator gFilterAcceptListCleared_c

Filter Accept List has been cleared. Response to Gap_ClearFilterAcceptList.

enumerator gRandomAddressReady_c

A random device address has been created. Response to Gap_CreateRandomDeviceAddress.

enumerator gCreateConnectionCanceled_c

Connection initiation was successfully cancelled. Response to Gap_CancelInitiatingConnection.

enumerator gPublicAddressRead_c

Contains the public device address. Response to Gap_ReadPublicDeviceAddress.

enumerator gAdvTxPowerLevelRead_c

Contains the TX power on the advertising channel. Response to Gap_ReadAdvertisingTxPowerLevel.

enumerator gPrivateResolvableAddressVerified_c

Contains the result of PRA verification. Response to Gap_VerifyPrivateResolvableAddress.

enumerator gRandomAddressSet_c

Random address has been set into the Controller. Response to Gap_SetRandomAddress.

enumerator gLeScPublicKeyRegenerated_c

The private/public key pair used for LE Secure Connections pairing has been regenerated.

enumerator gLeScLocalOobData_c

Local OOB data used for LE Secure Connections pairing.

enumerator gHostPrivacyStateChanged_c

Host Privacy was enabled or disabled.

enumerator gControllerPrivacyStateChanged_c

Controller Privacy was enabled or disabled.

enumerator gControllerTestEvent_c

Controller Test was started or stopped.

enumerator gTxPowerLevelSetComplete_c

Controller Tx Power Level set complete or invalid.

enumerator gLePhyEvent_c

Phy Mode of a connection has been updated by the Controller.

enumerator gControllerNotificationEvent_c

Controller Enhanced Notification received.

enumerator gGetConnParamsComplete_c

Get Connection Parameters command complete

enumerator gBondCreatedEvent_c

Bond Created Event signalling the stack created a bond after pairing or at app request.

enumerator gChannelMapSet_c

Channel map set complete in the Controller.

enumerator gExtAdvertisingParametersSetupComplete_c

Extended advertising parameters have been successfully set.

enumerator gExtAdvertisingDataSetupComplete_c

Extended advertising data has been successfully set.

enumerator gExtAdvertisingSetRemoveComplete_c

An advertising set has been removed from the Controller.

enumerator gPeriodicAdvParamSetupComplete_c

Periodic advertising parameters have been successfully set.

enumerator gPeriodicAdvDataSetupComplete_c

Periodic advertising data have been successfully set.

enumerator gPeriodicAdvertisingStateChanged_c

Event received when periodic advertising has been successfully enabled or disabled.

enumerator gPeriodicAdvListUpdateComplete_c

Periodic advertiser list has been successfully updated.

enumerator gPeriodicAdvCreateSyncCancelled_c

Periodic advertising create sync command was successfully cancelled

enumerator gTxEntryAvailable_c

This event is generated when a TX entry becomes available after they were all in use.

enumerator gControllerLocalRPARead_c

Contains the resolvable private device address. Response to Gap_ReadControllerLocalRPA.

enumerator gConnectionlessCteTransmitParamsSetupComplete_c

Connectionless CTE transmit parameters have been successfully set.

enumerator gConnectionlessCteTransmitStateChanged_c

Connectionless CTE for an advertising set was enabled or disabled.

enumerator gConnectionlessIqSamplingStateChanged_c

Connectionless CTE IQ sampling for an advertising train was enabled or disabled.

enumerator gAntennaInformationRead_c

Antenna information was read from the controller.

enumerator gModifiedSleepClockAccuracy_c

The Sleep Clock accuracy was changed

enumerator gPeriodicAdvRecvEnableComplete_c

Enable or disable reports for the periodic advertising train command is complete

enumerator gPeriodicAdvSyncTransferComplete_c

The command used to instruct the Controller to send synchronization information about the periodic advertising train identified by the Sync_Handle parameter to a connected device is complete

enumerator gPeriodicAdvSetInfoTransferComplete_c

The command used to instruct the Controller to send synchronization information about the periodic advertising in an advertising set to a connected device is complete

enumerator gSetPeriodicAdvSyncTransferParamsComplete_c

The command specifying how the Controller will process periodic advertising synchronization information is complete

enumerator gSetDefaultPeriodicAdvSyncTransferParamsComplete_c

The command which set the default parameters for periodic advertising synchronization information is complete

enumerator gPeriodicAdvSyncTransferSucceeded_c

Event received when Controller has succeeded the Synchronization to the periodic advertising train

enumerator gPeriodicAdvSyncTransferFailed_c

Event received when Controller has failed the Synchronization to the periodic advertising train

enumerator gConnEvtLeGenerateDhKeyComplete_c

DHKey generation is complete. Key can be found in gapConnectionEvent_t.eventData.leGenerateDhKeyCompleteEvent

enumerator gHandoverGetComplete_c

Handover data get complete.

enumerator gHandoverSetComplete_c

Handover data set complete.

enumerator gHandoverGetCsLlContextComplete_c

Handover CS LL context data get complete.

enumerator gHandoverSetCsLlContextComplete_c

Handover CS LL context data set complete.

enumerator gHandoverGetTime_c

Handover Get Time command complete

enumerator gHandoverSuspendTransmitComplete_c

Handover Suspend Transmit command complete

enumerator gHandoverResumeTransmitComplete_c

Handover Resume Transmit command complete

enumerator gHandoverAnchorNotificationStateChanged_c

Handover Anchor Notification command complete

enumerator gHandoverAnchorSearchStarted_c

Handover Anchor Search Start command complete

enumerator gHandoverAnchorSearchStopped_c

Handover Anchor Search Stop command complete

enumerator gHandoverTimeSyncTransmitStateChanged_c

Handover Time Sync Transmit command complete

enumerator gHandoverTimeSyncReceiveComplete_c

Handover Time Sync Receive command complete

enumerator gHandoverAnchorMonitorEvent_c

Event received from Controller - Handover Anchor Monitor

enumerator gHandoverTimeSyncEvent_c

Event received from Controller - Handover Time Sync

enumerator gHandoverConnParamUpdateEvent_c

Event received from Controller - Handover Conn Params Update

enumerator gRemoteVersionInformationRead_c

Version Information of a peer was read

enumerator gLlSkdReportEvent_c

Session Key Diversifier report

enumerator gLeSetSchedulerPriorityComplete_c

LE Set Scheduler Priority command complete

enumerator gDeInitializationComplete_c

Event received when Ble_HostDeInitialize is complete.

enumerator gHandoverAnchorMonitorPacketEvent_c

Event received from Controller - Handover Anchor Monitor Packet. pPdu must be freed by the application

enumerator gHandoverAnchorMonitorPacketContinueEvent_c

Event received from Controller - Handover Anchor Monitor Packet Continue. pPdu must be freed by the application

enumerator gHandoverFreeComplete_c

Handover data free complete.

enumerator gHandoverUpdateConnParamsComplete_c

Handover Update Connection Parameters command complete

enumerator gExtAdvertisingDecisionDataSetupComplete_c

Extended advertising decision data has been successfully set.

enumerator gDecisionInstructionsSetupComplete_c

decision instructions used when listening for decision advertisements PDUs has been successfully set.

enumerator gHandoverLlPendingData_c
ACL Data pending to be transmited by the LL

enumerator gLeChannelOverrideComplete_c
LE Channel Override command complete

enumerator gPeriodicAdvSetSubeventDataComplete_c
Periodic advertising subevent data has been successfully set.

enumerator gPeriodicAdvSetResponseDataComplete_c
Periodic advertising response data has been successfully set.

enumerator gPeriodicSyncSubeventComplete_c
Set Sync Subevent command successfully completed.

enumerator gHandoverConnectionUpdateProcedureEvent_c
This event is used to report the new connection parameters indicated during the Connection Update procedure

enumerator gHandoverApplyConnectionUpdateProcedureComplete_c
This event is used to report the new connection parameters indicated during the Connection Update procedure

enumerator gVendorUnitaryTestComplete_c
Vendor Unitary Test command complete

enumerator gLeSetDataRelatedAddressChangesComplete_c
Set Data Related Address Changes command complete

enumerator gLePeriodicAdvUpdateSyncComplete_c
Le Periodic Adv Update Sync command complete

enumerator gDeviceAddedToMonAdvList_c
LE Add Device To Monitored Advertisers List command complete

enumerator gDeviceRemovedFromMonAdvList_c
Remove Device From Monitored Advertisers List command complete

enumerator gMonAdvListCleared_c
Clear Monitored Advertisers List command complete

enumerator gMonAdvEnabled_c
Enable Monitoring Advertisers command complete

enumerator gMonAdvListSizeRead_c
Read Monitored Advertisers List Size command complete

enumerator gVendorDebugEvent_c
Vendor specific debugging event signaling NBU fatal error

enumerator gLeSetDefaultConnectionSubrateParametersSetupComplete_c
Default Connection Subrate Parameters.have been successfully set

enumerator gLeChasConfigComplete_c
LE CHAS Config command completed successfully

enumerator gLeModulationTestComplete_c
LE Modulation Test command completed successfully

enum gapInternalErrorSource_t
Internal Error Source - the command that triggered the error
Values:

enumerator gHciCommandStatus_c

enumerator gCheckPrivateResolvableAddress_c

enumerator gAddNewConnection_c

enumerator gResetController_c

enumerator gSetEventMask_c

enumerator gReadLeBufferSize_c

enumerator gSetLeEventMask_c

enumerator gReadDeviceAddress_c

enumerator gReadLocalSupportedFeatures_c

enumerator gReadFilterAcceptListSize_c

enumerator gClearFilterAcceptList_c

enumerator gAddDeviceToFilterAcceptList_c

enumerator gRemoveDeviceFromFilterAcceptList_c

enumerator gCancelCreateConnection_c

enumerator gReadRadioPower_c

enumerator gSetRandomAddress_c

enumerator gCreateRandomAddress_c

enumerator gEncryptLink_c

enumerator gProvideLongTermKey_c

enumerator gDenyLongTermKey_c

enumerator gConnect_c

enumerator gDisconnect_c

enumerator gTerminatePairing_c

enumerator gSendPeripheralSecurityRequest_c

enumerator gEnterPasskey_c

enumerator gProvideOob_c

enumerator gSendSmpKeys_c

enumerator gWriteSuggestedDefaultDataLength_c

enumerator gReadSuggestedDefaultDataLength_c

enumerator gUpdateLeDataLength_c

enumerator gEnableHostPrivacy_c

enumerator gEnableControllerPrivacy_c

enumerator gLeScSendKeypressNotification_c

enumerator gLeScSetPeerOobData_c

enumerator gLeScGetLocalOobData_c

enumerator gLeScValidateNumericValue_c

enumerator gLeScRegeneratePublicKey_c

enumerator gLeSetResolvablePrivateAddressTimeout_c

enumerator gDefaultPairingProcedure_c

enumerator gLeControllerTest_c

enumerator gLeReadPhy_c

enumerator gLeSetPhy_c

enumerator gSaveKeys_c

enumerator gSetChannelMap_c

enumerator gReadLocalSupportedCommands_c

enumerator gEnableLdmTimer_c

enumerator gRemoveAdvertisingSet_c

enumerator gLePeriodicAdvSyncEstb_c

enumerator gLePeriodicAdvSyncLost_c

enumerator gLeRemoveDeviceFromPeriodicAdvList_c

enumerator gLeClearPeriodicAdvList_c

enumerator gLeAddDeviceToPeriodicAdvList_c

enumerator gLeReadNumOfSupportedAdvSets_c

enumerator gLeReadPeriodicAdvListSize_c

enumerator gLeReadMaxAdvDataLen_c

enumerator gPeriodicAdvCreateSync

enumerator gPeriodicAdvCancelSync

enumerator gPeriodicAdvTerminateSync

enumerator gL2capRxPacket_c

enumerator gExtAdvReportProcess_c

enumerator gReadControllerLocalRPA_c

enumerator gHciEventReceiveHandler_c

enumerator gSetConnectionlessCteTransmitParams_c

enumerator gSetConnectionlessCteTransmitEnable_c

enumerator gSetConnectionlessIqSamplingEnable_c

enumerator gReadAntennaInformation_c

enumerator gSetConnectionCteReceiveParams_c

enumerator gSetConnectionCteTransmitParams_c

enumerator gConnectionCteReqEnable_c

enumerator gConnectionCteRspEnable_c

enumerator gGenerateDHKeyV2_c

enumerator gModifySleepClockAccuracy_c

enumerator gPeriodicAdvRcvEnable_c

enumerator gPeriodicAdvSyncTransfer_c

enumerator gPeriodicAdvSetInfoTransfer_c

enumerator gSetPeriodicAdvSyncTransferParams_c

enumerator gSetDefaultPeriodicAdvSyncTransferParams_c

enumerator gEnhancedReadTransmitPowerLevel_c

enumerator gReadRemoteTransmitPowerLevel_c

enumerator gSetPathLossReportingParams_c

enumerator gSetPathLossReportingEnable_c

enumerator gSetTransmitPowerReportingEnable_c

enumerator gEattConnectionRequest_c

enumerator gEattConnectionAccept_c

enumerator gEattReconfigureRequest_c

enumerator gEattSendCreditsRequest_c

enumerator gEattDisconnectRequest_c

enumerator gEattL2caCancelConnection_c

enumerator gEattL2caSendLeFlowControlCredit_c

enumerator gEattL2caDisconnectLePsm_c

enumerator gEattL2caHandleSendLeCbData_c

enumerator gEattL2caHandleRecvLeCbData_c

enumerator gEattL2caEnhancedReconfigureReq_c

enumerator gEattL2caEnhancedCancelConnection_c

enumerator gHciRecvFragmentOfPacket_c

enumerator gHciDataDiscardedAlloc_c
A memory allocation failure occurred in the HCI layer. Data was discarded.

enumerator gHciDataDiscardedInvalidStateParam_c
A packet with an invalid parameter or length was received by the HCI layer. Data was discarded.

enumerator gGetConnParams_c
An error occurred during the Get Connection Params procedure

enumerator gHandover_c
An error occurred during the connection handover process

enumerator gHandoverGetLlContext_c
An error occurred during the Handover Get LL Context procedure

enumerator gHandoverConnect_c
An error occurred during the Handover Connect procedure

enumerator gHandoverDisconnect_c
An error occurred during the Handover Disconnect procedure

enumerator gHandoverSetSkd_c
An error occurred during the Handover Set SKD procedure

enumerator gHandoverSuspendTransmitLlProcInProgress_c
Could not suspend transmit at LL level due to an LL procedure being in progress

enumerator gHandoverSuspendTransmitHostTxInProgress_c
Could not suspend transmit at Host level due to ongoing data transfers

enumerator gHandoverGetCsLlContext_c
An error occurred during the Handover Get CS LL Context procedure

enumerator gHandoverSetCsLlContext_c
An error occurred during the Handover Set CS LL Context procedure

enumerator gHandoverUpdateConnParams_c
An error occurred during the Handover Update Connection Parameters procedure

enumerator gReadRemoteVersionInfo_c

enumerator gLeSetSchedulerPriority_c

enumerator gLeSetHostFeature_c

enumerator gSetExtAdvDecisionData_c

enumerator gSetDecisionInstructions_c

enumerator gSetExpmSupportedFeatures_c

enumerator gHandoverSuspendTransmitHciTx_c
Could not suspend transmit due to HCI tx error

enumerator gHandoverResumeTransmitHciTx_c
Could not resume transmit due to HCI tx error

enumerator gHandoverAnchorNotifHciTx_c
Could not enable/disable Anchor Notification due to HCI tx error

enumerator gHandoverAnchorSearchStartHciTx_c
Could not start Anchor Search due to HCI tx error

enumerator gHandoverAnchorSearchStopHciTx_c
Could not stop Anchor Search due to HCI tx error

enumerator gHandoverTimeSyncTxHciTx_c
Could not start/stop Anchor Time Sync transmit due to HCI tx error

enumerator gHandoverTimeSyncRxHciTx_c
Could not start/stop Anchor Time Receive transmit due to HCI tx error

enumerator gHandoverSuspendTransmitLl_c
Handover Suspend Transmit Link Layer error

enumerator gHandoverGetTimeLl_c
Handover Get Time Link Layer error

enumerator gHandoverResumeTransmitLl_c
Handover Resume Transmit Link Layer error

enumerator gHandoverAnchorNotifLl_c
Handover Anchor Notification Link Layer error

enumerator gHandoverAnchorSearchStartLl_c
Handover Anchor Search Start Link Layer error

enumerator gHandoverAnchorSearchStopLl_c
Handover Anchor Search Stop Link Layer error

enumerator gHandoverTimeSyncTxLl_c
Handover Time Sync Transmit Link Layer error

enumerator gHandoverTimeSyncRxLl_c
Handover Time Sync Receive Link Layer error

enumerator gHandoverSetLlPendingData_c
Handover Set LL Pending Data error

enumerator gLeChannelOverride_c
LE Channel Override command unsuccessful

enumerator gLeSetPeriodicAdvParamsV2_c

enumerator gLeSetPeriodicAdvSubeventData_c

enumerator gLeSetPeriodicAdvResponseData_c

enumerator gLeSetPeriodicSyncSubevent_c

enumerator gLePeriodicAdvResponseReport_c

enumerator gHandoverApplyConnectionUpdateProcedure_c
An error occurred during the Handover Apply Connection Update procedure

enumerator gVendorUnitaryTest_c
An error occurred during the Vendor Unitary Test procedure

enumerator gSetDataRelatedAddressChanges_c
An error occurred during the Set Data Related Address procedure

enumerator gLePeriodicAdvUpdateSync_c
An error occurred during the Le Periodic Adv Update Sync procedure

enumerator gReadAllLocalSupportedFeatures_c
An error occurred during the LE Read All Local Supported Features command

enumerator gAddDeviceToMonAdvList_c
An error occurred during the Add Device To Monitored Advertisers List

enumerator gRemoveDeviceFromMonAdvList_c
An error occurred during the Remove Device From Monitored Advertisers List

enumerator gClearMonAdvList_c
An error occurred during the Clear Monitored Advertisers List

enumerator gEnableMonAdv_c
An error occurred during the Enable Monitoring Advertisers

enumerator gReadMonAdvListSize_c
An error occurred during the Read Monitored Advertisers List Size

enumerator gSetDefaultConnectionSubrateParameters_c
An error occurred while setting the Default Connection Subrate Parameters

enumerator gConnectionSubrateRequest_c
An error occurred during the Connection Subrate procedure

enumerator gEncryptLinkTak_c
An error occurred during Encrypt link with TAK procedure

enumerator gVendorEnableExpmFeatures_c
An error occurred during gHciVendorEnableExpmFeaturesCommand_c execution

enumerator gLeSetLocalPeripheralLatencyEnable_c
An error occurred during the Le Set Local Peripheral Latency Enable procedure

enumerator gLeChasConfig_c
An error occurred during LE CHAS Config command execution

enumerator gLeModulationTest_c
An error occurred during LE Modulation Test command execution

enumerator gVendorEnableLePowerControl_c
An error occurred during gHciVendorEnableLePowerControlCommand_c execution

enum gapControllerTestEventType_t
Controller Test Event Type

Values:

enumerator gControllerReceiverTestStarted_c

enumerator gControllerTransmitterTestStarted_c

enumerator gControllerTestEnded_c

enum gapLeAllPhyFlags_t

Le All Phys Preferences flags.

Values:

enumerator gLeTxPhyNoPreference_c

Host has no preference for Tx Phy

enumerator gLeRxPhyNoPreference_c

Host has no preference for Rx Phy

enum gapLePhyOptionsFlags_t

Le Phys Options Preferences flags.

Values:

enumerator gLeCodingNoPreference_c

Host has no preference on the LE Coded Phy

enumerator gLeCodingS2_c

Host prefers to use S=2 on the LE Coded Phy

enumerator gLeCodingS8_c

Host prefers to use S=8 on the LE Coded Phy

enumerator gLeCodingS2Req_c

Host requires to use S=2 on the LE Coded Phy

enumerator gLeCodingS8Req_c

Host requires to use S=8 on the LE Coded Phy

enum gapLePhyMode_tag

Values:

enumerator gLePhy1M_c

Tx/Rx Phy on the connection is LE 1M

enumerator gLePhy2M_c

Tx/Rx Phy on the connection is LE 2M

enumerator gLePhyCoded_c

Tx/Rx Phy on the connection is LE Coded

enum gapPhyEventType_t

Phy Event Type

Values:

enumerator gPhySetDefaultComplete_c

Gap_LeSetPhy default mode was successful

enumerator gPhyRead_c
Gap_LeReadPhy return values

enumerator gPhyUpdateComplete_c
Gap_LeSetPhy return values for a connection or an update occurred

enum bleNotificationEventType_tag
Values:

enumerator gNotifEventNone_c
No enhanced notification event enabled

enumerator gNotifConnEventOver_c
Connection event over

enumerator gNotifConnRxPdu_c
Connection Rx PDU

enumerator gNotifAdvEventOver_c
Advertising event over

enumerator gNotifAdvTx_c
Advertising ADV transmitted

enumerator gNotifAdvScanReqRx_c
Advertising SCAN REQ Rx

enumerator gNotifAdvConnReqRx_c
Advertising CONN REQ Rx

enumerator gNotifScanEventOver_c
Scanning event over

enumerator gNotifScanAdvPktRx_c
Scanning ADV PKT Rx

enumerator gNotifScanRspRx_c
Scanning SCAN RSP Rx

enumerator gNotifScanReqTx_c
Scanning SCAN REQ Tx

enumerator gNotifConnCreated_c
Connection created

enumerator gNotifChannelMatrix_c
Enable channel status monitoring (KW37 only)

enumerator gNotifPhyReq_c
Phy Req Pdu ack received (KW37 only)

enumerator gNotifConnChannelMapUpdate_c
Channel map update

enumerator gNotifConnInd_c
Connect indication

enumerator gNotifPhyUpdateInd_c
Phy update indication

enum bleCteTransmitEnable_t
Values:

enumerator gCteTransmitDisable_c

enumerator gCteTransmitEnable_c

enum bleIqSamplingEnable_t
Values:

enumerator gIqSamplingDisable_c

enumerator gIqSamplingEnable_c

enum bleCteReqEnable_t
Values:

enumerator gCteReqDisable_c

enumerator gCteReqEnable_c

enum bleCteRspEnable_t
Values:

enumerator gCteRspDisable_c

enumerator gCteRspEnable_c

enum bleCteType_tag
Values:

enumerator gCteTypeAoA_c
AoA Constant Tone Extension

enumerator gCteTypeAoD1us_c
AoD Constant Tone Extension with 1 microsecond slots

enumerator gCteTypeAoD2us_c

AoD Constant Tone Extension with 2 microsecond slots

enumerator gCteTypeNoCte_c

No Constant Tone Extension

enum bleSlotDurations_tag

Values:

enumerator gSlotDurations1us_c

Switching and sampling slot durations are 1 microsecond each

enumerator gSlotDurations2us_c

Switching and sampling slot durations are 2 microseconds each

enum bleIqReportPacketStatus_tag

Values:

enumerator gIqReportPacketStatusCorrectCrc_c

CRC was correct

enumerator gIqReportPacketStatusCrcIncorrectUsedLength_c

CRC was incorrect and the Length and CTETime fields of the packet were used to determine sampling points

enumerator gIqReportPacketStatusCrcIncorrectUsedOther_c

CRC was incorrect but the Controller has determined the position and length of the Constant Tone Extension in some other way

enumerator gIqReportPacketStatusInsufficientResources_c

Insufficient resources to sample (Channel_Index, CTE_Type, and Slot_Durations invalid).

enum blePathLossThresholdZoneEntered_tag

Values:

enumerator gPathLossThresholdLowZone_c

Entered low zone.

enumerator gPathLossThresholdMiddleZone_c

Entered middle zone.

enumerator gPathLossThresholdHighZone_c

Entered high zone.

enum bleTxPowerReportingReason_tag

Values:

enumerator gLocalTxPowerChanged_c
Local transmit power changed.

enumerator gRemoteTxPowerChanged_c
Remote transmit power changed.

enumerator gReadRemoteTxPowerLevelCommandCompleted_c
HCI_LE_Read_Remote_Transmit_Power_Level command completed.

enum blePowerControlPhyType_tag
Values:

enumerator gPowerControlLePhy1M_c
LE 1M Phy.

enumerator gPowerControlLePhy2M_c
LE 2M Phy.

enumerator gPowerControlLePhyCodedS8_c
LE Coded Phy with S=8 data coding.

enumerator gPowerControlLePhyCodedS2_c
LE Coded Phy with S=2 data coding.

enum blePathLossReportingEnable_t
Values:

enumerator gPathLossReportingDisable_c

enumerator gPathLossReportingEnable_c

enum bleTxPowerReportingEnable_t
Values:

enumerator gTxPowerReportingDisable_c

enumerator gTxPowerReportingEnable_c

enum bleHandoverSuspendTransmitMode_t
Values:

enumerator gDoNotUseEventCounter_c

enumerator gUseEventCounter_c

enumerator gSafeStopLlTx_c

enumerator gSafeStopHostLlTx_c

enum bleHandoverAnchorSearchMode_t

Values:

enumerator gSuspendTxMode_c

enumerator gRssiSniffingMode_c

enumerator gPacketMode_c

enum bleGetConnParamsMode_t

Values:

enumerator gSendEventAfterRemoteSNChange_c

enumerator gSendEventAfterLLProcUpdate_c

enum bleHandoverAnchorNotificationEnable_t

Values:

enumerator gAnchorNotificationDisable_c

enumerator gAnchorNotificationEnable_c

enum bleHandoverTimeSyncEnable_t

Values:

enumerator gTimeSyncDisable_c

enumerator gTimeSyncEnable_c

enum bleHandoverTimeSyncStopWhenFound_t

Values:

enumerator gTimeSyncDoNotStopWhenFound_c

enumerator gTimeSyncStopWhenFound_c

enum bleAdvIndexType_t

Values:

enumerator gAdvIndexAscend_c

enumerator gAdvIndexDescend_c

enumerator gAdvIndexUser_c

enumerator gAdvIndexRandom_c

typedef enum *bleResult_tag* bleResult_t

BLE result type - the return value of BLE API functions

typedef uint8_t deviceId_t

Unique identifier type for a connected device.

typedef uint8_t bearerId_t

ATT bearer identifier.

typedef enum *bearerStatus_tag* bearerStatus_t

Enhanced ATT bearer status values

typedef uint8_t bleAddressType_t

Bluetooth Device Address Type - Size: 1 Octet, Range: [gBleAddrTypePublic_c:gBleAddrTypeRandom_c]

typedef uint8_t bleDeviceAddress_t[(6U)]

Bluetooth Device Address - array of 6 bytes.

typedef uint8_t bleUuidType_t

Bluetooth UUID type - values chosen to correspond with the ATT UUID format

typedef uint16_t bleAdvReportEventProperties_t

Advertising Event properties

typedef uint16_t bleAdvRequestProperties_t

Advertising Request properties

typedef uint8_t bleCentralClockAccuracy_t

typedef uint8_t bleAdvertiserClockAccuracy_t

typedef uint8_t bleScanningFilterPolicy_t

Scanning filter policy enumeration - Size: 1 Octet, Range: [gScanAll_c:gScanWithFilterAcceptList_c]

typedef uint8_t bleInitiatorFilterPolicy_t

Initiator filter policy enumeration - Size: 1 Octet, Range: [gUseDeviceAddress_c:gUseFilterAcceptList_c]

typedef uint8_t blePrivacyMode_t

Privacy Mode enumeration - Size: 1 Octet, Range: [gNetworkPrivacy_c:gDevicePrivacy_c]

typedef uint8_t bleChannelMap_t[(5U)]

Bluetooth Channel map - array of 5 bytes.

typedef uint64_t leSupportedFeatures_t

typedef uint8_t gapLePhyFlags_t
Le Tx/Rx Phys Preferences flags.

typedef uint8_t gapLePhyMode_t
Le Tx/Rx Phys.

typedef uint16_t bleNotificationEventType_t
Controller Enhanced Notification Event Type

typedef struct *bleBondCreatedEvent_tag* bleBondCreatedEvent_t
Bond Created Event

typedef struct *gapAddrReadyEvent_t_tag* gapAddrReadyEvent_t
Address Ready Event

typedef struct *bleAntennaInformation_tag* bleAntennaInformation_t
Antenna Information Read Event

typedef struct *periodicAdvSyncTransferEvent_tag* periodicAdvSyncTransferEvent_t

typedef struct *periodicAdvSetInfoTransferEvent_tag* periodicAdvSetInfoTransferEvent_t

typedef struct *periodicAdvSetSyncTransferParamsEvent_tag*
periodicAdvSetSyncTransferParamsEvent_t

typedef struct *gapSyncTransferReceivedEventData_tag* gapSyncTransferReceivedEventData_t

typedef struct *getConnParams_tag* getConnParams_t

typedef struct *handoverGetTime_tag* handoverGetTime_t

typedef struct *handoverAnchorSearchStart_tag* handoverAnchorSearchStart_t

typedef struct *handoverAnchorSearchStop_tag* handoverAnchorSearchStop_t

typedef struct *handoverConnect_tag* handoverConnect_t

typedef struct *handoverGetData_tag* handoverGetData_t

typedef struct *handoverSetData_tag* handoverSetData_t

typedef struct *handoverGetCsLlContext_tag* handoverGetCsLlContext_t

typedef struct *handoverAnchorMonitorEvent_tag* handoverAnchorMonitorEvent_t

typedef struct *handoverAnchorMonitorPacketEvent_tag* handoverAnchorMonitorPacketEvent_t

typedef struct *handoverAnchorMonitorPacketContinueEvent_tag* handoverAnchorMonitorPacketContinueEvent_t

typedef struct *handoverConnectionUpdateProcedureEvent_tag* handoverConnectionUpdateProcedureEvent_t

typedef struct *handoverTimeSyncEvent_tag* handoverTimeSyncEvent_t

typedef struct *handoverConnParamUpdateEvent_tag* handoverConnParamUpdateEvent_t

typedef struct *handoverSuspendTransmitCompleteEvent_tag* handoverSuspendTransmitCompleteEvent_t

typedef struct *handoverResumeTransmitCompleteEvent_tag* handoverResumeTransmitCompleteEvent_t

typedef struct *handoverUpdateConnParams_tag* handoverUpdateConnParams_t

typedef struct *handoverApplyConnectionUpdateProcedure_tag* handoverApplyConnectionUpdateProcedure_t

typedef struct *handoverAnchorNotificationStateChanged_tag* handoverAnchorNotificationStateChanged_t

typedef struct *handoverLlPendingDataIndication_tag* handoverLlPendingDataIndication_t

typedef struct *gapRemoteVersionInfoRead_tag* gapRemoteVersionInfoRead_t

typedef struct *gapLlSkdReport_tag* gapLlSkdReport_t

typedef struct *vendorUnitaryTestEvent_tag* vendorUnitaryTestEvent_t

typedef struct *vendorSpecificDbgEvent_tag* vendorSpecificDbgEvent_t

typedef void (*gapGenericCallback_t)(*gapGenericEvent_t* *pGenericEvent)

Generic Callback prototype.

typedef *bleResult_t* (*hciHostToControllerInterface_t)(*hciPacketType_t* packetType, void *pPacket, uint16_t packetSize)

Host-to-Controller API prototype.

typedef uint8_t bleCteType_t

typedef struct *bleSyncCteType_tag* bleSyncCteType_t

typedef uint8_t bleSlotDurations_t

typedef uint8_t bleIqReportPacketStatus_t

typedef uint8_t blePathLossThresholdZoneEntered_t

typedef uint8_t bleTxPowerReportingReason_t

typedef uint8_t blePowerControlPhyType_t

const uint8_t gBleMaxActiveConnections

const uint8_t gBleMaxAdvSets

const uint16_t gcConnectionEventMinDefault_c

const uint16_t gcConnectionEventMaxDefault_c

const uint8_t gBleEattMaxConnectionChannels

const uint16_t gBleEattPsmMtu

const uint8_t gMaxAdvReportQueueSize

const bool_t gUseHciCommandFlowControl

Enables/Disables Hci Command Flow Control if the host lib supports this feature

bool_t gHostInitEnableExpmFeatures

bool_t gHostInitEnableLePowerControl

bool_t gEnableCoredumpPackets

messaging_t gApp2Host_TaskQueue

App to Host message queue for the Host Task

messaging_t gHci2Host_TaskQueue

HCI to Host message queue for the Host Task

bleResult_t Ble_HostInitialize(*gapGenericCallback_t* genericCallback, *hciHostToControllerInterface_t* hostToControllerInterface)

Performs initialization of the BLE Host stack.

Remark

Application must wait for the gInitializationComplete_c generic event.

Parameters

- genericCallback – **[in]** Callback used to propagate GAP generic events to the application.
- hostToControllerInterface – **[in]** LE Controller uplink interface function pointer

Returns
gBleSuccess_c or error.

bleResult_t Ble_HostDeInitialize(void)

performs De-Initialize the Bluetooth LE Host

Remark

Application must wait for the gDeInitializationComplete_c generic event.

Parameters

- none –

Return values
gBleSuccess_c – or error.

bleResult_t Ble_HciRecv(*hciPacketType_t* packetType, void *pHciPacket, uint16_t packetSize)

This is the BLE Host downlink interface function.

Remark

This function must be registered as a callback by the LE Controller and called to send HCI packets (events and LE-U data) to the BLE Host.

Parameters

- packetType – **[in]** The type of the packet sent by the LE Controller
- pHciPacket – **[in]** Pointer to the packet sent by the LE Controller
- packetSize – **[in]** Number of bytes sent by the LE Controller

Returns
gBleSuccess_c or gBleOutOfMemory_c

bleResult_t Ble_HciRecvFromIsr(*hciPacketType_t* packetType, void *pHciPacket, uint16_t packetSize)

This is the BLE Host downlink interface function to be used in ISR context.

Remark

This function must be registered as a callback by the HCI Transport layer and called to send HCI packets from the controller (events and LE-U data) to the BLE Host.

Parameters

- packetType – **[in]** The type of the packet sent by the LE Controller
- pHciPacket – **[in]** Pointer to the packet sent by the LE Controller
- packetSize – **[in]** Number of bytes sent by the LE Controller

Returns
gBleSuccess_c or gBleOutOfMemory_c

bool_t Ble_CheckMemoryStorage(void)

This function performs runtime checks to determine whether memory storage is properly configured for the application settings.

Returns
True if memory is configured correctly False otherwise.

OSA_EVENT_HANDLE_DEFINE (gHost_TaskEvent)

Event for the Host Task Queue

void Host_TaskHandler(void *args)

Contains the Host Task logic.

Remark

This function must be called exclusively by the Host Task code from the application.

---

static inline uint16_t IncrementU16Saturating(uint16_t val, uint16_t increment)

Adds an increment to a 16-bit unsigned value, saturating at UINT16_MAX to avoid unsigned integer wraparound.

Parameters

- val – **[in]** The base value to increment.
- increment – **[in]** The value to add.

Returns
uint16_t The saturated result of val + increment.

static inline uint8_t IncrementU8Saturating(uint8_t val, uint8_t increment)

Adds an increment to an 8-bit unsigned value, saturating at UINT8_MAX to avoid unsigned integer wraparound.

Parameters

- val – **[in]** The base value to increment.
- increment – **[in]** The value to add.

Returns
uint8_t The saturated result of val + increment.

gInvalidDeviceId_c

gInvalidNvmIndex_c

gcConnectionIntervalMin_c

Boundary values for the Connection Parameters (Standard GAP).

gcConnectionIntervalMax_c

gcConnectionPeripheralLatencyMax_c

gcConnectionSupervisionTimeoutMin_c

gcConnectionSupervisionTimeoutMax_c

gcConnectionSubrateFactorMin_c

gcConnectionSubrateFactorMax_c

gcConnectionSubrateContinuationNumberMax_c

gcConnectionIntervalMinDefault_c

Default values for the Connection Parameters (Preferred).

connIntervalmin = Conn_Interval_Min * 1.25 ms

Value of 0xFFFF indicates no specific minimum.

gcConnectionIntervalMaxDefault_c

connIntervalmax = Conn_Interval_Max * 1.25 ms

Value of 0xFFFF indicates no specific maximum.

gcConnectionPeripheralLatencyDefault_c

gcConnectionSupervisionTimeoutDefault_c

Time = N * 10 ms

STATIC

When unit testing is performed, access from unit test module to static functions/variables within the tested module is not possible and therefore the static storage class identifier shall be removed

gBleAddrTypePublic_c

Bluetooth Device Address Types Public Device Address - fixed into the Controller by the manufacturer.

gBleAddrTypeRandom_c

Random Device Address - set by the Host into the Controller for privacy reasons.

Ble_IsPrivateResolvableDeviceAddress(bleAddress)

PRA condition: check the 6th byte - MSB should be 0; 2nd MSB should be 1.

Ble_IsPrivateNonresolvableDeviceAddress(bleAddress)

PNRA condition: check the 6th byte - MSB should be 0; 2nd MSB should be 0.

Ble_IsRandomStaticDeviceAddress(bleAddress)

RSA condition: check the 6th byte - MSB should be 1; 2nd MSB should be 1.

Ble_DeviceAddressesMatch(bleAddress1, bleAddress2)

A macro used to compare two device addresses

Ble_CopyDeviceAddress(destinationAddress, sourceAddress)

A macro used to copy device addresses

gBleUuidType16_c

16-bit standard UUID

gBleUuidType128_c

128-bit long/custom UUID

gBleUuidType32_c

32-bit UUID - not available as ATT UUID format

gLePhy1MFlag_c

Host prefers to use LE 1M Tx/Rx Phy, possibly among others

gLePhy2MFlag_c

Host prefers to use LE 2M Tx/Rx Phy, possibly among others

gLePhyCodedFlag_c

Host prefers to use LE Coded Tx/Rx Phy, possibly among others

gAdvDataChange_c

Advertising Data Change reason

gScanRspDataChange_c

Scan Response Data Change reason

gUseDeviceAddress_c

Initiator filter policy values Initiates a connection with a specific device identified by its address.

gUseFilterAcceptList_c

Initiates connections with all the devices in the Filter Accept List at the same time.

gUseDeviceAddressNoDecisionPDUs_c

Initiator filter policy values extended for gBLE60_DecisionBasedAdvertisingFilteringSupport_d == TRUE Filter Accept List is not used to determine which advertiser to connect to. Decision PDUs shall be ignored. Peer_Address_Type and Peer_Address shall be used.

gUseFilterAcceptListNoDecisionPDUs_c

Filter Accept List is used to determine which advertiser to connect to. Decision PDUs shall be ignored. Peer_Address_Type and Peer_Address shall be ignored.

gUseOnlyDecisionPDUs_c

Filter Accept List is not used to determine which advertiser to connect to. Only Decision PDUs shall be processed. Peer_Address_Type and Peer_Address shall be ignored.

gUseFilterAcceptListAllPDUs_c

Filter Accept List is used to determine which advertiser to connect to. All PDUs shall be processed. Peer_Address_Type and Peer_Address shall be ignored.

gUseDecisionPDU_UseFilterAcceptListForOtherPDUs_c

All decision PDUs shall be processed. Filter Accept List is used to determine which other PDUs to process. Peer_Address_Type and Peer_Address shall be ignored.

gScanAll_c

Basic Scanning filter policy values. Kept for apps backward compatibility Basic unfiltered scanning filter policy.

gScanWithFilterAcceptList_c

Basic filtered scanning filter policy (using the Filter Accept List).

gBasicUnfilteredScan_c

Scanning filter policy values. Basic unfiltered scanning filter policy.

gBasicFilteredScan_c

Basic filtered scanning filter policy (using the Filter Accept List).

gExtendedUnfilteredScan_c

Extended unfiltered scanning filter policy.

gExtendedFilteredScan_c

Extended filtered scanning filter policy(using the Filter Accept List).

gScanOnlyNonDecisionPDUs_c

PDU Scanning filter policy values. To be used in bitwise OR operations with scanning filter policy above only with gBLE60_DecisionBasedAdvertisingFilteringSupport_d == TRUE e.g. ((uint8_t)(gExtendedUnfilteredScan_c | gScanAllPDUs_c)) Scan Only Non Decision Advertising PDU.

gScanAllPDUs_c

Scan Decision and Non Decision Advertising PDU.

gScanOnlyDecisionPDUs_c

Scan Only Decision Advertising PDU.

gNetworkPrivacy_c

Privacy mode values Use Network Privacy Mode for the peer device (default)

gDevicePrivacy_c

Use Device Privacy Mode for the peer device

gUnenhancedBearerId_c

bearer id 0 is used for the Unenhanced ATT bearer

gDHKeySize_c

LE Secure Connections 256 bit DHKey

gSkdSize_c
: LL Session Key Diversfier size

gHCICSChannelMapSize
: CS Channel Map size

gCSReflectorTableSize_c
: CS reflector table size size

gCSNumPhysMax_c
: CS reflector table size size

gCSNumToneAntennaIds_c
: CS number of antenna IDs used in the pattern

gCSTestMaxChannelLength_c
: CS number of channels used in the the pattern

gDRBGKeySize_c
: CS debugger key size

gCSSyncRandomSize_c
: CS Sync Random size

gCSMaxSubeventLen_c
: CS maximum duration for each CS sub-event in microseconds

gCsAccessAdressSize_c
: CS Access Address used in SYNC packets

LL_PHY_1M

LL_PHY_2M

LL_PHY_S8

LL_PHY_S2

gVendorHandoverMaxCsLlContextSize_c
: Maximum size of the LL Context for Handover (CS context is largest)

gVendorUnitaryTestSize_c
: Maximum size of the VENDOR_UNITARY_TEST response

gVendorDebugEventSize_c
: Maximum size of the debug payload

gLeExtendedFeaturesSize_c

Current size of the leExtendedFeatures, can be raised up to 240

getLeExtendedFeatureByte(bitNumber)

getLeExtendedFeatureBit(bitNumber)

isSupportedLeExtendedFeature(features, bitNumber)

gLeModulationTestOptionPhy2M_c

LE Modulation Test Options bit definitions bit 0: 0=1M PHY, 1=2M PHY

gLeModulationTestOptionGenericChannel_c

bit 1: 0=BLE RF index, 1=generic channel

gLeModulationTestOptionPowerIndex_c

bit 2: 0=tx power in dBm, 1=power index

gBleSig_PrimaryService_d

Bluetooth SIG UUID constants for GATT declarations

Primary Service declaration UUID

gBleSig_SecondaryService_d

Secondary Service declaration UUID

gBleSig_Include_d

Include declaration UUID

gBleSig_Characteristic_d

Characteristic declaration UUID

gBleSig_CharExtendedProperties_d

Characteristic Extended Properties UUID

gBleSig_CharUserDescription_d

Client Characteristic User Description UUID

gBleSig_CCCD_d

Client Characteristic Configuration Descriptor declaration UUID

gBleSig_SCCD_d

Server Characteristic Configuration Descriptor declaration UUID

gBleSig_CharPresFormatDescriptor_d

Characteristic Presentation Format declaration UUID

gBleSig_CharAggregateFormat_d

Characteristic Aggregate Format UUID

gBleSig_ValidRangeDescriptor_d
: Valid Range Descriptor declaration UUID

gBleSig_GenericAccessProfile_d
: GAP Service UUID

gBleSig_GenericAttributeProfile_d
: GATT Service UUID

gBleSig_ImmediateAlertService_d
: Immediate Alert Service UUID

gBleSig_LinkLossService_d
: Link Loss Service UUID

gBleSig_TxPowerService_d
: Tx Power Service UUID

gBleSig_CurrentTimeService_d
: Current Time Service UUID

gBleSig_ReferenceTimeUpdateService_d
: Reference Time Update Service UUID

gBleSig_NextDSTChangeService_d
: Next DST Change Service UUID

gBleSig_GlucoseService_d
: Glucose Service UUID

gBleSig_HealthThermometerService_d
: Health Thermometer Service UUID

gBleSig_DeviceInformationService_d
: Device Information Service UUID

gBleSig_HeartRateService_d
: Heart Rate Service UUID

gBleSig_PhoneAlertStatusService_d
: Phone Alert Status Service UUID

gBleSig_BatteryService_d
: Battery Service UUID

gBleSig_BloodPressureService_d
: Blood Pressure Service UUID

gBleSig_AlertNotificationService_d
: Alert Notification Service UUID

gBleSig_HidService_d
: HID Service UUID

gBleSig_RunningSpeedAndCadenceService_d
: Running Speed And Cadence Service UUID

gBleSig_CyclingSpeedAndCadenceService_d
: Cycling Speed And Cadence Service UUID

gBleSig_CyclingPowerService_d
: Cycling Power Service UUID

gBleSig_LocationAndNavigationService_d
: Location And Navigation Service UUID

gBleSig_IpsService_d
: Internet Protocol Support Service UUID

gBleSig_PulseOximeterService_d
: Pulse Oximeter Service UUID

gBleSig_HTTPProxyService_d
: HTTP Proxy Service UUID

gBleSig_WPTService_d
: Wireless Power Transfer Service UUID

gBleSig_BtpService_d
: BTP Service UUID

gBleSig_GapDeviceName_d
: GAP Device Name Characteristic UUID

gBleSig_GapAppearance_d
: GAP Appearance Characteristic UUID

gBleSig_GapPpcp_d
: GAP Peripheral Preferred Connection Parameters Characteristic UUID

gBleSig_GattServiceChanged_d
: GATT Service Changed Characteristic UUID

gBleSig_GattClientSupportedFeatures_d
: GATT Client Supported Features Characteristic UUID

gBleSig_GattServerSupportedFeatures_d
: GATT Server Supported Features Characteristic UUID

gBleSig_GattDatabaseHash_d
: GATT Database Hash Characteristic UUID

gBleSig_AlertLevel_d
: Alert Level Characteristic UUID

gBleSig_TxPower_d
: TX Power Characteristic UUID

gBleSig_LocalTimeInformation_d
: Local Time Information Characteristic UUID

gBleSig_TimeWithDST_d
: Time With DST Characteristic UUID

gBleSig_ReferenceTimeInformation_d
: Reference Time Information Characteristic UUID

gBleSig_TimeUpdateControlPoint_d
: Time Update Control Point Characteristic UUID

gBleSig_TimeUpdateState_d
: Time Update State Characteristic UUID

gBleSig_GlucoseMeasurement_d
: Glucose Measurement Characteristic UUID

gBleSig_BatteryLevel_d
: Battery Level Characteristic UUID

gBleSig_TemperatureMeasurement_d
: Temperature Measurement Characteristic UUID

gBleSig_TemperatureType_d
: Temperature Type Characteristic UUID

gBleSig_IntermediateTemperature_d
: Intermediate Temperature Characteristic UUID

gBleSig_MeasurementInterval_d
: Measurement Interval Characteristic UUID

gBleSig_SystemId_d
: System ID Characteristic UUID

gBleSig_ModelNumberString_d
: Model Number String Characteristic UUID

gBleSig_SerialNumberString_d
: Serial Number String Characteristic UUID

gBleSig_FirmwareRevisionString_d
: Firmware Revision String Characteristic UUID

gBleSig_HardwareRevisionString_d
: Hardware Revision String Characteristic UUID

gBleSig_SoftwareRevisionString_d
: Software Revision String Characteristic UUID

gBleSig_ManufacturerNameString_d
: Manufacturer Name String Characteristic UUID

gBleSig_IeeeRcdl_d
: IEEE 11073-20601 Regulatory Certification Data List Characteristic UUID

gBleSig_CurrentTime_d
: Current Time Characteristic UUID

gBleSig_BootKeyboardInputReport_d
: Boot Keyboard Input Report UUID

gBleSig_BootKeyboardOutputReport_d
: Boot Keyboard output Report UUID

gBleSig_BootMouseInputReport_d
: Boot Mouse Input Report UUID

gBleSig_GlucoseMeasurementContext_d
: Glucose Measurement Context Characteristic UUID

gBleSig_BpMeasurement_d
: Blood Pressure Measurement UUID

gBleSig_BpEnhancedMeasurement_d
: Blood Pressure Enhanced Measurement UUID

gBleSig_BpRecord_d
: Blood Pressure Record UUID

gBleSig_IntermediateCuffPressure_d
: Intermediate Cuff Pressure UUID

gBleSig_HrMeasurement_d
Heart Rate Measurement UUID

gBleSig_BodySensorLocation_d
Body Sensor Location UUID

gBleSig_HrControlPoint_d
Heart Rate Control Point UUID

gBleSig_AlertStatus_d
Alert Status UUID

gBleSig_RingerControlPoint_d
Ringer Control Point UUID

gBleSig_RingerSetting_d
Ringer Setting UUID

gBleSig_AlertNotifControlPoint_d
Alert Notif Control Point UUID

gBleSig_UnreadAlertStatus_d
Unread Alert Status UUID

gBleSig_NewAlert_d
New Alert UUID

gBleSig_SupportedNewAlertCategory_d
Supported New Alert Category UUID

gBleSig_SupportedUnreadAlertCategory_d
Supported Unread Alert Category UUID

gBleSig_BloodPressureFeature_d
Blood Pressure Feature UUID

gBleSig_HidInformation_d
HID Information UUID

gBleSig_HidBootMouseInputReport_d
HID Boot Mouse Input Report UUID

gBleSig_HidCtrlPoint_d
HID Control Point UUID

gBleSig_Report_d
Report UUID

gBleSig_ProtocolMode_d
: Protocol Mode UUID

gBleSig_ScanIntervalWindow_d
: Scan Interval Window UUID

gBleSig_PnpId_d
: PnP Id UUID

gBleSig_GlucoseFeature_d
: Glucose Feature Characteristic UUID

gBleSig_RaCtrlPoint_d
: Record Access Ctrl Point Characteristic UUID

gBleSig_RscMeasurement_d
: RSC Measurement UUID

gBleSig_RscFeature_d
: RSC Feature UUID

gBleSig_ScControlPoint_d
: SC Control Point UUID

gBleSig_CscMeasurement_d
: CSC Measurement Characteristic UUID

gBleSig_CscFeature_d
: CSC Feature Characteristic UUID

gBleSig_SensorLocation_d
: Sensor Location Characteristic UUID

gBleSig_PlxSCMeasurement_d
: PLX Spot-Check Measurement Characteristic UUID

gBleSig_PlxContMeasurement_d
: PLX Continuous Measurement Characteristic UUID

gBleSig_PulseOximeterFeature_d
: PLX Feature Characteristic UUID

gBleSig_CpMeasurement_d
: CP Measurement Characteristic UUID

gBleSig_CpVector_d
: CP Measurement Vector UUID

gBleSig_CpFeature_d
: CP Feature CharacteristicUUID

gBleSig_CpControlPoint_d
: CP Control Point UUID

gBleSig_LocationAndSpeed_d
: Location and Speed Characteristic UUID

gBleSig_Navigation_d
: Navigation Characteristic UUID

gBleSig_PositionQuality_d
: Position Quality Characteristic UUID

gBleSig_LnFeature_d
: LN Feature Characteristic UUID

gBleSig_LnControlPoint_d
: LN Control Point Characteristic UUID

gBleSig_Temperature_d
: Temperature Characteristic UUID

gBleSig_CentralAddressResolution_d
: Central Address Resolution Characteristic UUID

gBleSig_URI_d
: URI Characteristic UUID

gBleSig_HTTP_Headers_d
: HTTP Headers Characteristic UUID

gBleSig_HTTP_StatusCode_d
: HTTP Status Code Characteristic UUID

gBleSig_HTTP_EntityBody_d
: HTTP Entity Body Characteristic UUID

gBleSig_HTTP_ControlPoint_d
: HTTP Control Point Characteristic UUID

gBleSig_HTTPS_Security_d
: HTTPS Security Characteristic UUID

gBleSig_ResolvablePrivateAddressOnly_d
: Resolvable Private Address Only Characteristic UUID

gBleSig_MeshProvisioningService_d
BLE Mesh Provisioning Service UUID

gBleSig_MeshProxyService_d
BLE Mesh Proxy Service UUID

gBleSig_MeshProvDataIn_d
BLE Mesh Prov Data In Char UUID

gBleSig_MeshProvDataOut_d
BLE Mesh Prov Data Out Char UUID

gBleSig_MeshProxyDataIn_d
BLE Mesh Proxy Data In Char UUID

gBleSig_MeshProxyDataOut_d
BLE Mesh Proxy Data Out Char UUID

gBleSig_GattSecurityLevels_d
LE GATT Security Levels characteristic UUID

gBleSig_EncryptedDataKeyMaterial_d
Encrypted Data Key Material characteristic UUID

gBleSig_CAR_NotSupported_d
Central Address Resolution Characteristic Values

gBleSig_CAR_Supported_d

gBleSig_RPAO_Used_d
Resolvable Private Address Only Characteristic Values

gBleSig_RangingService_d
Ranging Service UUID

gBleSig_RasFeature_d
RAS Feature Characteristic UUID

gBleSig_RasRealTimeProcData_d
RAS Real-time Procedure Data Characteristic UUID

gBleSig_RasOnDemandProcData_d
RAS On-demand Procedure Data Characteristic UUID

gBleSig_RasControlPoint_d
RAS Control Point Characteristic UUID

gBleSig_RasProcDataReady_d
RAS Procedure Data Ready Characteristic UUID

gBleSig_RasprocDataOverwritten_d
RAS Procedure Data Overwritten Characteristic UUID

gBleSig_TirePressureMonitoringService_d
Tire Pressure Monitoring System Service UUID

gBleSig_TirePressure_d
Tire Pressure Characteristic UUID

gBleSig_TireTemperature_d
Tire Temperature Characteristic UUID

gBleSig_TireAcceleration_d
Tire Acceleration Characteristic UUID

gBleSig_TpmsProperties_d
TPMS Properties Characteristic UUID

gBleSig_TpmsMonitoringModeDutyCycle_d
TPMS Monitoring Mode Duty Cycle Characteristic UUID

gBleSig_TpmsPosition_d
TPMS Position Characteristic UUID

gBleSig_TpmsSigningKey_d
TPMS Signing Key UUID

BleSig_IsGroupingAttributeUuid16(uuid16)
Macro that returns whether or not an input 16-bit UUID is a grouping type.

BleSig_IsServiceDeclarationUuid16(uuid16)
Macro that returns whether or not an input 16-bit UUID is a Service declaration.

Uuid16(uuid)
Macro that declares a 16 bit UUID in a *bleUuid_t* union.

Uuid32(uuid)
Macro that declares a 32 bit UUID in a *bleUuid_t* union.

UuidArray(value)

PACKED_STRUCT
Type qualifier - does not affect local variables of integral type

__noreturn
Type qualifier - does not affect local variables of integral type

Type qualifier - does not affect local variables of integral type

Storage class modifier - alignment of a variable. It does not affect the type of the function

Marks a function that never returns.

Utils_ExtractTwoByteValue(buf)
Returns a uint16_t from a buffer, little-endian

Utils_ExtractThreeByteValue(buf)
Returns a 3-byte value from a buffer, little-endian

Utils_ExtractFourByteValue(buf)
Returns a uint32_t from a buffer, little-endian

Utils_ExtractEightByteValue(buf)
Returns a uint64_t from a buffer, little-endian

Utils_BeExtractTwoByteValue(buf)
Returns a uint16_t from a buffer, big-endian

Utils_BeExtractThreeByteValue(buf)
Returns a 3-byte value from a buffer, big-endian

Utils_BeExtractFourByteValue(buf)
Returns a uint32_t from a buffer, big-endian

Utils_PackTwoByteValue(value, buf)
Writes a uint16_t into a buffer, little-endian

Utils_PackThreeByteValue(value, buf)
Writes a 3-byte value into a buffer, little-endian

Utils_PackFourByteValue(value, buf)
Writes a uint32_t into a buffer, little-endian

Utils_PackEightByteValue(value, buf)
Writes a uint64_t into a buffer, little-endian

Utils_BePackTwoByteValue(value, buf)
Writes a uint16_t into a buffer, big-endian

Utils_BePackThreeByteValue(value, buf)
Writes a 3-byte value into a buffer, big-endian

Utils_BePackFourByteValue(value, buf)
Writes a uint32_t into a buffer, big-endian

Utils_Copy8(ptr, val8)
Writes a uint8_t into a buffer, little-endian, and increments the pointer.

Utils_Copy16(ptr, val16)
Writes a uint16_t into a buffer, little-endian, and increments the pointer.

Utils_Copy32(ptr, val32)
Writes a uint32_t into a buffer, little-endian, and increments the pointer.

Utils_Copy64(ptr, val64)
Writes a uint64_t into a buffer, little-endian, and increments the pointer.

Utils_RevertByteArray(array, size)
Reverts the order of bytes in an array - useful for changing the endianness

Uint32toUint8(x)

Uint32toUint16(x)

Uint16toUint8(x)

struct bleIdentityAddress_t
#include <ble_general.h> Bluetooth Identity Address - array of 6 bytes.

Public Members

bleAddressType_t idAddressType

Public or Random (static).

bleDeviceAddress_t idAddress

6-byte address.

union bleUuid_t

#include <ble_general.h> Union for a Bluetooth UUID; selected according to an accompanying bleUuidType_t

Public Members

uint16_t uuid16

For gBleUuidType16_c.

uint32_t uuid32

For gBleUuidType32_c.

uint8_t uuid128[16]

For gBleUuidType128_c.

struct bleAdvertisingChannelMap_t

#include <ble_general.h>

Public Members

uint8_t enableChannel37

Bit for channel 37.

uint8_t enableChannel38

Bit for channel 38.

uint8_t enableChannel39

Bit for channel 39.

uint8_t reserved

Reserved for future use.

struct gapLeScOobData_t

#include <ble_general.h>

Public Members

uint8_t randomValue[(16U)]

LE SC OOB r (Random value)

uint8_t confirmValue[(16U)]

LE SC OOB Cr (Random Confirm value)

struct gapInternalError_t

#include <ble_general.h> Internal Error Event Data

Public Members

bleResult_t errorCode

Host Stack error code.

gapInternalErrorSource_t errorSource

The command that generated the error; useful when it is not obvious from the error code.

uint16_t hciCommandOpcode

Only for errorSource = gHciCommandStatus_c; the HCI Command that received an error status.

struct gapControllerTestEvent_t

#include <ble_general.h> Controller Test Event

struct gapPhyEvent_t

#include <ble_general.h> Phy Event

struct bleNotificationEvent_t

#include <ble_general.h> Controller Enhanced Notification Event

Public Members

bleNotificationEventType_t eventType

Enhanced notification event type

deviceId_t deviceId

Device id of the peer, valid for connection events

int8_t rssi

RSSI, valid for Rx event types

uint8_t channel

Channel, valid for conn event over or Rx/Tx events

uint16_t ce_counter

Connection event counter, valid for conn event over or Conn Rx event

bleResult_t status

Status of the request to select which events to be enabled/disabled

uint32_t timestamp

Timestamp in microseconds, valid for Conn Rx event and Conn Created event

uint8_t adv_handle

Advertising Handle, valid for advertising events, if multiple ADV sets supported

bleDeviceAddress_t scanned_addr

Scanned address, valid for gNotifScanAdvPktRx_c event, all-zeroes otherwise

struct gapInitComplete_t

#include <ble_general.h> gInitializationComplete_c event data

struct bleBondCreatedEvent_tag

#include <ble_general.h> Bond Created Event

Public Members

uint8_t nvmIndex

NVM index for the new created bond

bleAddressType_t addressType

Public or Random (static) address of the bond

bleDeviceAddress_t address

Address of the bond

struct gapAddrReadyEvent_t_tag

#include <ble_general.h> Address Ready Event

Public Members

bleDeviceAddress_t aAddress

Generated device address

uint8_t advHandle

Advertising set handle if the generated device address will be used on an extended set. Reserved value 0xFF for other purposes: legacy advertising or scanning and initiating address.

struct bleSupportedSwitchingSamplingRates_t

#include <ble_general.h>

Public Members

uint8_t switchingSupportedAodTransmission
Bit to mark that 1 us switching is supported for AoD transmission.

uint8_t samplingSupportedAodReception
Bit to mark that 1 us sampling is supported for AoD reception.

uint8_t switchingSamplingSupportedAoaReception
Bit to mark that 1 us switching and sampling is supported for AoA reception.

uint8_t reserved
Reserved for future use.

struct bleAntennaInformation_tag
#include <ble_general.h> Antenna Information Read Event

struct periodicAdvSyncTransferEvent_tag
#include <ble_general.h>

struct periodicAdvSetInfoTransferEvent_tag
#include <ble_general.h>

struct periodicAdvSetSyncTransferParamsEvent_tag
#include <ble_general.h>

struct gapSyncTransferReceivedEventData_tag
#include <ble_general.h>

Public Members

deviceId_t deviceId
Id of the connected device from which we received periodic advertising sync transfer

uint16_t serviceData
A value provided by the peer device

uint16_t syncHandle
Sync Handle identifying the periodic advertising train

uint8_t advSID
Value of the Advertising SID used to advertise the periodic advertising

bleAddressType_t advAddressType
Device's advertising address type.

bleDeviceAddress_t advAddress
Device's advertising address.

gapLePhyMode_t advPhy
PHY used for advertising

uint16_t periodicAdvInt
Periodic advertising interval

bleCentralClockAccuracy_t advClockAccuracy
Advertiser clock accuracy

uint8_t numSubevents
Only relevant for v2 of the event, otherwise value is 0

uint8_t subeventInterval
Only relevant for v2 of the event, otherwise value is 0

uint8_t responseSlotDelay
Only relevant for v2 of the event, otherwise value is 0

uint8_t responseSlotSpacing
Only relevant for v2 of the event, otherwise value is 0

struct getConnParams_tag
#include <ble_general.h>

Public Members

uint16_t connectionHandle
Connection identifier

uint32_t ulTxAccCode
Access address

uint8_t aCrcInitVal[3U]
CRC init

uint16_t uiConnInterval
Connection interval (unit 1.25ms)

uint16_t uiSuperTO
Supervision timeout (unit 10ms)

uint16_t uiConnLatency
Latency (unit connection interval)

uint8_t aChMapBm[5U]
Channel map (FF FF FF FF 1F if all channels are used)

uint8_t ucChannelSelection
Hop algorithm (0/1 for hop algorithm #1 or #2)

uint8_t ucHop
Hop increment

uint8_t ucUnMapChIdx
Unmapped channel index (used only for hop algorithm #1)

uint8_t ucCentralSCA
Sleep clock accuracy (0 to 7)

uint8_t ucRole
Role (0 for central and 1 for peripheral)

uint8_t aucRemoteMasRxPHY
TX/RX PHY

uint8_t seqNum
Sequence number; b1 and b0: the latest received SN and NESN; b5 and b4: the latest transmitted SN and NESN; other fields are reserved.

uint16_t uiConnEvent
Current connection event counter

uint32_t ulAnchorClk
Slot of the anchor point timing of the connection event

uint16_t uiAnchorDelay
Slot offset of the anchor point timing of the connection event

uint32_t ulRxInstant
Last successful access address reception instant (unit 625us)

struct handoverGetTime_tag
#include <ble_general.h>

Public Members

bleResult_t status
Command status

uint32_t slot
LL timing slot counter (unit 625us)

uint16_t us_offset
LL timing slot offset (0 to 624, unit us)

struct handoverAnchorSearchStart_tag

#include <ble_general.h>

Public Members

bleResult_t status

Command status

uint16_t connectionHandle

Connection identifier

struct handoverAnchorSearchStop_tag

#include <ble_general.h>

Public Members

bleResult_t status

Command status

uint16_t connectionHandle

Connection identifier

struct handoverConnect_tag

#include <ble_general.h>

Public Members

bleResult_t status

Command status

uint16_t connectionHandle

Connection identifier

struct handoverGetData_tag

#include <ble_general.h>

Public Members

bleResult_t status

Command status

uint32_t *pData

Connection handover data

struct handoverSetData_tag

#include <ble_general.h>

Public Members

bleResult_t status
Command status

uint32_t *pData
Connection handover data

struct handoverGetCsLlContext_tag
#include <ble_general.h>

Public Members

bleResult_t status
Command status

uint16_t responseMask
LL context bitmap indicating the Channel Sounding context

uint8_t llContextLength
Context data length

uint8_t llContext[(224U)]
LL Context

struct handoverAnchorMonitorEvent_tag
#include <ble_general.h>

Public Members

uint16_t connectionHandle
Connection identifier

uint16_t connEvent
Current connection event counter

int8_t rssiRemote
RSSI of the packet from the remote device (+127 if not available)

uint8_t lqiRemote
LQI (Link Quality Indicator) of the packet from the remote device

uint8_t statusRemote
Status of the packet from the remote device; b0: NESN; b1: SN; b2: CRC status valid (1) or invalid (0); b3: RSSI status valid (1) or invalid (0)

int8_t rssiActive

RSSI of the packet from the active device (+127 if not available)

uint8_t lqiActive

LQI (Link Quality Indicator) of the packet from the active device

uint8_t statusActive

Status of the packet from the active device; b0: NESN; b1: SN; b2: CRC status valid (1) or invalid (0); b3: RSSI status valid (1) or invalid (0)

uint32_t anchorClock625Us

Slot of the anchor point timing of the connection event

uint16_t anchorDelay

Slot offset of the anchor point timing of the connection event

uint8_t chIdx

BLE channel index

uint8_t ucNbReports

Number of remaining reports including the current one; 0: continuous reporting; 1-255: remaining number of reports to receive including the current one

struct handoverAnchorMonitorPacketEvent_tag

#include <ble_general.h>

Public Members

uint8_t packetCounter

Packet counter, incremented for each event

uint16_t connectionHandle

Connection identifier

uint8_t statusPacket

Status of the packet

uint8_t phy

PHY (0/1/2/3 for 1M/2M/LR S8/LR S2)

uint8_t chIdx

BLE channel index

int8_t rssiPacket

RSSI of the packet (+127 if not available)

uint8_t lqiPacket

LQI (Link Quality Indicator) of the packet

uint16_t connEvent

Current connection event counter

uint32_t anchorClock625Us

Slot value of packet start time (in 625us unit)

uint16_t anchorDelay

Slot offset value of packet start time (in 1us unit)

uint8_t ucNbConnIntervals

Number of remaining connection intervals to monitor including the current one; 0: continuous reporting; 1-255: remaining number of reports to receive including the current one

uint8_t pduSize

PDU length

uint8_t *pPdu

Full or first part of the data PDU including: PDU header, payload if present, MIC if present, CRC

struct handoverAnchorMonitorPacketContinueEvent_tag

#include <ble_general.h>

Public Members

uint8_t packetCounter

Packet counter, incremented for each event

uint16_t connectionHandle

Connection identifier

uint8_t pduSize

PDU length

uint8_t *pPdu

Full or first part of the data PDU including: PDU header, payload if present, MIC if present, CRC

struct handoverConnectionUpdateProcedureEvent_tag

#include <ble_general.h>

Public Members

uint16_t connectionHandle

Connection identifier

uint8_t winSize

Used to indicate the transmitWindowSize value as: transmitWindowSize = winSize * 1.25 ms

uint16_t winOffset

Used to indicate the transmitWindowOffset value as: transmitWindowOffset = winOffset * 1.25 ms

uint16_t interval

Used to indicate the connInterval value, as: connInterval = interval * 1.25 ms

uint16_t latency

Used to indicate the connPeripheralLatency value as: connPeripheralLatency = latency

uint16_t timeout

Used to indicate the connSupervisionTimeout value as: connSupervisionTimeout = timeout * 10 ms

uint16_t instant

Connection event counter value indicating when the new connection parameters are applied

uint16_t currentEventCounter

Current connection event counter

struct handoverTimeSyncEvent_tag

#include <ble_general.h>

Public Members

uint32_t txClkSlot

Transmitter packet start time in slot (625 us)

uint16_t txUs

Transmitter packet start time offset inside the slot (0 to 624 us)

uint32_t rxClkSlot

Receiver packet start time in slot (625 us)

uint16_t rxUs

Receiver packet start time offset inside the slot (0 to 624 us)

uint8_t rssi

RSSI

struct handoverConnParamUpdateEvent_tag

#include <ble_general.h>

Public Members

uint8_t status

Status indicating the event trigger source: bit 0: remote SN^NESN toggled; bit 1: local SN^NESN toggled; bit 2: RFU; bit 3: phy update procedure; bit 4: channel map update procedure

uint16_t connectionHandle

Connection identifier

uint32_t ulTxAccCode

Access address

uint8_t aCrcInitVal[3U]

CRC init

uint16_t uiConnInterval

Connection interval (unit 1.25ms)

uint16_t uiSuperTO

Supervision timeout (unit 10ms)

uint16_t uiConnLatency

Latency (unit connection interval)

uint8_t aChMapBm[5U]

Channel map (FF FF FF FF 1F if all channels are used)

uint8_t ucChannelSelection

Hop algorithm (0/1 for hop algorithm #1 or #2)

uint8_t ucHop

Hop increment

uint8_t ucUnMapChIdx

Unmapped channel index (used only for hop algorithm #1)

uint8_t ucCentralSCA

Sleep clock accuracy (0 to 7)

uint8_t ucRole

Role (0 for central and 1 for peripheral)

uint8_t aucRemoteMasRxPHY

TX/RX PHY

uint8_t seqNum

Sequence number; b1 and b0: the latest received SN and NESN; b5 and b4: the latest transmitted SN and NESN; other fields are reserved.

uint16_t uiConnEvent
Current connection event counter

uint32_t ulAnchorClk
Slot of the anchor point timing of the connection event

uint16_t uiAnchorDelay
Slot offset of the anchor point timing of the connection event

uint32_t ulRxInstant
Last successful access address reception instant (unit 625us)

struct handoverSuspendTransmitCompleteEvent_tag
#include <ble_general.h>

Public Members

uint16_t connectionHandle
Connection identifier

uint8_t noOfPendingAclPackets
Number of pending ACL packets (including the packet being transmitted partially or fully but not fully acked)

uint16_t sizeOfPendingAclPackets
Pending ACL packet data size including all the data in the oldest packet:

- data transmitted but not acked in the oldest ACL packet
- data transmitted and acked in the oldest ACL packet

uint16_t sizeOfDataTxInOldestPacket
Size of data transmitted and acked in the oldest ACL packet

uint8_t sizeOfDataNAckInOldestPacket
Size of data transmitted but not acked in the oldest ACL packet

struct handoverResumeTransmitCompleteEvent_tag
#include <ble_general.h>

Public Members

uint16_t connectionHandle
Connection identifier

struct handoverUpdateConnParams_tag
#include <ble_general.h>

Public Members

bleResult_t status
Command status

uint16_t connectionHandle
Connection identifier

struct handoverApplyConnectionUpdateProcedure_tag
#include <ble_general.h>

Public Members

uint16_t connectionHandle
Connection identifier

struct handoverAnchorNotificationStateChanged_tag
#include <ble_general.h>

Public Members

uint16_t connectionHandle
Connection identifier

struct handoverLlPendingDataIndication_tag
#include <ble_general.h>

Public Members

uint16_t dataSize
Pending HCI ACL data packet size

uint8_t *pData
Pointer to message containing the HCI ACL data packet. Should be free by the application using MSG_Free()

struct gapLEGenerateDhKeyCompleteEvent_t
#include <ble_general.h>

struct gapRemoteVersionInfoRead_tag
#include <ble_general.h>

struct gapLlSkdReport_tag
#include <ble_general.h>

Public Members

deviceId_t deviceId
Peer device identifier

uint8_t aSKD[(16U)]
LL SKD

struct vendorUnitaryTestEvent_tag
#include <ble_general.h>

struct vendorSpecificDbgEvent_tag
#include <ble_general.h>

struct gapGenericEvent_t
#include <ble_general.h> Generic Event Structure = type + data

Public Members

gapGenericEventType_t eventType
Event type.

union *gapGenericEvent_t* eventData
Event data, selected according to event type.

struct bleBondIdentityHeaderBlob_t
#include <ble_general.h>

struct bleBondDataDynamicBlob_t
#include <ble_general.h>

struct bleBondDataStaticBlob_t
#include <ble_general.h>

struct bleBondDataLegacyBlob_t
#include <ble_general.h>

struct bleBondDataDeviceInfoBlob_t
#include <ble_general.h>

struct bleBondDataDescriptorBlob_t
#include <ble_general.h>

struct bleLocalKeysBlob_t
#include <ble_general.h>

struct bleBondDataBlob_t
#include <ble_general.h>

struct bleBondDataRam_t

#include <ble_general.h>

struct bleCteAllowedTypesMap_t

#include <ble_general.h>

Public Members

uint8_t allowAoA

Bit to allow AoA CTE Response.

uint8_t allowAoD1us

Bit to allow AoD CTE Response with 1 microsecond slots.

uint8_t allowAoD2us

Bit to allow AoD CTE Response with 2 microsecond slots.

uint8_t reserved

Reserved for future use.

struct bleSyncCteType_tag

#include <ble_general.h>

Public Members

uint8_t doNotSyncWithAoA

Bit for CTE type AoA.

uint8_t doNotSyncWithAoD1us

Bit for CTE type AoD 1us.

uint8_t doNotSyncWithAoD2us

Bit for CTE type AoD 2us.

uint8_t doNotSyncWithType3

Bit for CTE type 3 (reserved).

uint8_t doNotSyncWithoutCte

Bit for no CTE type.

uint8_t reserved

Reserved for future use.

struct bleTxPowerLevelFlags_t

#include <ble_general.h>

Public Members

uint8_t minimum

Tx Power Level is at minimum level.

uint8_t maximum

Tx Power Level is at maximum level.

uint8_t reserved

Reserved for future use.

union eventData

Public Members

gapInternalError_t internalError

Data for the gInternalError_c event. The error that has occurred and the command that triggered it.

uint8_t filterAcceptListSize

Data for the gFilterAcceptListSizeRead_c event. The size of the Filter Accept List.

bleDeviceAddress_t aAddress

Data for the gPublicAddressRead_c event. Contains the requested device address.

gapAddrReadyEvent_t addrReady

Data for the gRandomAddressReady_c event. Contains the generated device address and advertising handle if applicable (0xFF otherwise).

uint8_t advHandle

Data for the gRandomAddressSet_c event. Contains the handle of the configured advertising set or 0xFF for legacy advertising.

bleResult_t setupFailError

Data for the gAdvertisingSetupFailed_c event. The error that occurred during the advertising setup.

int8_t advTxPowerLevel_dBm

Data for the gExtAdvertisingParametersSetupComplete_c and gAdvTxPowerLevelRead_c events. Value in dBm.

bool_t verified

Data for the gPrivateResolvableAddressVerified_c event. TRUE if the PRA was resolved with the given IRK.

gapLeScOobData_t localOobData

Data for the gLeScLocalOobData_c event. Contains local OOB data for LESC Pairing.

bool_t newHostPrivacyState

Data for the gHostPrivacyStateChanged_c event. TRUE if enabled, FALSE if disabled.

bool_t newControllerPrivacyState

Data for the gControllerPrivacyStateChanged_c event. TRUE if enabled, FALSE if disabled.

gapControllerTestEvent_t testEvent

Data for the gControllerTestEvent_c event. Contains test event type and received packets.

bleResult_t txPowerLevelSetStatus

Data for the gTxPowerLevelSetComplete_c event. Status of the set request.

gapPhyEvent_t phyEvent

Data for the gLePhyEvent_c event. Contains Tx and Rx Phy for a connection.

deviceId_t deviceId

Data for the gTxEntryAvailable_c event.

gapInitComplete_t initCompleteData

Data for the gInitializationComplete_c event. Contains the supported features, number of advertising sets and the size of the periodic advertiser list

bleNotificationEvent_t notifEvent

Data for the gControllerNotificationEvent_c event. Contains status and adv/scan/conn event data.

bleBondCreatedEvent_t bondCreatedEvent

Data for the gBondCreatedEvent_c event. Contains the NVM index and the address of the bond.

bleDeviceAddress_t aControllerLocalRPA

Data for the gControllerLocalRPARead_c event. Contains the requested device address.

getConnParams_t getConnParams

Data for the gGetConnParamsComplete_c event

uint16_t syncHandle

Data for the gConnectionlessIqSamplingStateChanged_c event. Contains the sync handle for the advertising train.

bleAntennaInformation_t antennaInformation

Data for the gAntennaInformationRead_c. Contains antenna information.

bleResult_t perAdvSyncTransferEnable

Data for the gPeriodicAdvRecvEnableComplete_c event

periodicAdvSyncTransferEvent_t perAdvSyncTransfer

Data for the gPeriodicAdvSyncTransferComplete_c event

periodicAdvSetInfoTransferEvent_t perAdvSetInfoTransfer

Data for the gPeriodicAdvSetInfoTransferComplete_c event

periodicAdvSetSyncTransferParamsEvent_t perAdvSetSyncTransferParams

Data for the gSetPeriodicAdvSyncTransferParamsComplete_c event

bleResult_t perAdvSetDefaultPerAdvSyncTransferParams

Data for the gSetDefaultPeriodicAdvSyncTransferParamsComplete_c event

gapSyncTransferReceivedEventData_t perAdvSyncTransferReceived

Data for the gPeriodicAdvSyncTransferReceived_c event

gapLEGenerateDhKeyCompleteEvent_t leGenerateDhKeyCompleteEvent

Data for gConnEvtLeGenerateDhKeyComplete_c: new key value

uint16_t pawrSyncHandle

Data for gPeriodicAdvSetResponseDataComplete_c and gPeriodicSyncSubeventComplete_c

uint8_t pawrAdvHandle

Data for gPeriodicAdvSetSubeventDataComplete_c

handoverGetData_t handoverGetData

Data for the gHandoverGetComplete_c event

handoverSetData_t handoverSetData

Data for the gHandoverSetComplete_c event

handoverGetCsLlContext_t handoverGetCsLlContext

Data for the gHandoverGetCsLlContextComplete_c event

handoverGetTime_t handoverGetTime

Data for the gHandoverGetTime_c event

handoverConnect_t handoverConnect

Data for the gHandoverConnect_c event

handoverAnchorSearchStart_t handoverAnchorSearchStart

Data for the gHandoverAnchorSearchStarted_c event

handoverAnchorSearchStop_t handoverAnchorSearchStop

Data for the gHandoverAnchorSearchStopped_c event

handoverAnchorMonitorEvent_t handoverAnchorMonitor

Data for the gHandoverAnchorMonitorEvent_c event

handoverTimeSyncEvent_t handoverTimeSync

Data for the gHandoverTimeSyncEvent_c event

handoverSuspendTransmitCompleteEvent_t handoverSuspendTransmitComplete

Data for the gHandoverSuspendTransmitComplete_c event

handoverResumeTransmitCompleteEvent_t handoverResumeTransmitComplete

Data for the gHandoverResumeTransmitComplete_c event

handoverAnchorNotificationStateChanged_t handoverAnchorNotificationStateChanged

Data for the gHandoverAnchorNotificationStateChanged_c event

handoverConnParamUpdateEvent_t handoverConnParamUpdate

Data for the gHandoverConnParamUpdateEvent_c event

gapRemoteVersionInfoRead_t gapRemoteVersionInfoRead

Data for the hciReadRemoteVersionInformationCompleteEvent_t event

gapLlSkdReport_t gapSkdReport

Data for the gLlSkdReportEvent_c event

handoverAnchorMonitorPacketEvent_t handoverAnchorMonitorPacket

Data for the gHandoverAnchorMonitorPacketEvent_c event. pPdu must be freed by the application

handoverAnchorMonitorPacketContinueEvent_t handoverAnchorMonitorPacketContinue

Data for the gHandoverAnchorMonitorPacketContinueEvent_c event. pPdu must be freed by the application

handoverUpdateConnParams_t handoverUpdateConnParams

Data for the gHandoverUpdateConnParamsComplete_c event

handoverLlPendingDataIndication_t handoverLlPendingDataIndication

Data for the gHandoverLlPendingData_c event

handoverConnectionUpdateProcedureEvent_t handoverConnectionUpdateProcedure

Data for the gHandoverConnectionUpdateProcedureEvent_c event

handoverApplyConnectionUpdateProcedure_t handoverApplyConnectionUpdateProcedure

Data for the gHandoverApplyConnectionUpdateProcedureComplete_c event

vendorUnitaryTestEvent_t unitaryTestData

Data for the gVendorUnitaryTestComplete_c event

uint8_t monAdvListSize

Data for the gMonAdvListSizeRead_c event. The size of the Monitored Advertisers List.

vendorSpecificDbgEvent_t vendorDebug

Data for the gVendorDebugEvent_c event.

union ___unnamed2___

Public Members

uint32_t raw[((40U) + 3U) / sizeof(uint32_t)]

uint8_t pKey[(40U)]

Bluetooth configuration

gcBleDeviceAddressSize_c
: Size of a BLE Device Address

gBleBondDataDynamicSize_c
: Size of bond data structures for a bonded device

gBleBondDataStaticSize_c

gBleBondDataLegacySize_c

gBleBondDataDeviceInfoSize_c

gBleBondDataDescriptorSize_c

gcSecureModeSavedLocalKeysNo_c

gcGapMaximumSavedCccds_c
: Maximum number of CCCDs

gcGapMaxAuthorizationHandles_c
: Maximum number of attributes that require authorization

gBleBondDataSize_c
: Bonding Data Size

gcBleLongUuidSize_c
: Size of long UUIDs

gcSmpMaxLtkSize_c
: Maximum Long Term Key size in bytes

gcSmpMaxBlobSize_c
: Maximum Long Term Key blob size in bytes

gcSmpIrkSize_c
: Identity Resolving Key size in bytes

gcSmpMaxIrkBlobSize_c
: Maximum Identity Resolving Key blob size in bytes

gcSmpCsrkSize_c

Connection Signature Resolving Key size in bytes - deprecated, not used

gcSmpMaxRandSize_c

Maximum Rand size in bytes

gcSmpOobSize_c

SMP OOB size in bytes

gSmpLeScRandomValueSize_c

SMP LE Secure Connections Pairing Random size in bytes

gSmpLeScRandomConfirmValueSize_c

SMP LE Secure Connections Pairing Confirm size in bytes

gcGapMaxDeviceNameSize_c

Maximum device name size

gcGapMaxAdvertisingDataLength_c

Maximum size of advertising and scan response data

gAttDefaultMtu_c

Default value of the ATT_MTU

gAttMaxMtu_c

Maximum possible value of the ATT_MTU for this device. This is used during the MTU Exchange.

gEattMaxMtu_c

Maximum possible value of the ATT_MTU for enhanced ATT bearers on this device.

gAttMaxValueLength_c

The maximum length of an attribute value shall be 512 octets.

gEattMinMtu_c

Minimum MTU size for enhanced ATT channels is 64.

gHciTransportUartChannel_c

Channel the number of the UART hardware module (For example, if UART1 is used, this value should be 1).

gcReservedFlashSizeForCustomInformation_c

Number of bytes reserved for storing application-specific information about a device

gcBleChannelMapSize_c

Size of a channel map in a connection

gBleMinTxOctets_c

gBleMinTxTime_c

gBleMaxTxOctets_c

gBleMaxTxTime_c

gBleMaxTxTimeCodedPhy_c

gBleCteMinTxOctets_c

gBleCteMinTxTime_c

gBleCteMaxTxOctets_c

gBleCteMaxTxTime_c

gBleExtAdvMaxSetId_c
: Maximum value of the advertising SID.

gBlePeriodicAdvMaxSyncHandle_c
: Maximum value of the periodic advertising handle.

gBleExtAdvLegacySetId_c
: SID of the legacy advertising set.

gBleExtAdvLegacySetHandle_c
: Handle of the legacy advertising set.

gBleExtAdvDefaultSetId_c
: Default SID for extended advertising.

gBleExtAdvDefaultSetHandle_c
: Default handle for extended advertising.

gBleAdvTxPowerNoPreference_c
: Host has no preference for Tx Power

gBleExtAdvNoDuration_c
: No advertising duration. Advertising to continue until the Host disables it.

gBleHighDutyDirectedAdvDuration
: Default advertising duration in high duty directed advertising 1.28s = 1280ms/10ms(unit) = 128

gBleExtAdvNoMaxEvents_c
: No maximum number of advertising events.

gBlePeriodicAdvDefaultHandle_c

Periodic advertising default handle.

gBlePeriodicAdvSyncTimeoutMin_c

Minimum value for the sync_timeout parameter

gBlePeriodicAdvSyncTimeoutMax_c

Maximum value for the sync_timeout parameter

gBlePeriodicAdvSkipMax_c

Maximum value for the skip parameter

gBleMaxADStructureLength_c

Maximum length of an AD structure

gBleMaxExtAdvDataLength_c

Maximum length of Extended Advertising Data

gBleExtAdvMaxAuxOffsetUsec_c

Maximum value in us of AUX Offset(13 bits) in AuxPtr in 300us units, i.e. ((1 « 13) - 1) * 300

gBleP256KeyLength_c

Length of P-256 key in octets

gcDecisionDataKeySize_c

decision data key size in bytes

gcDecisionInstructionsParamSize_c

decision instructions parameter size in bytes

gcDecisionInstructionsArbitraryDataMaskSize_c

decision instructions arbitrary data mask size in bytes

gcDecisionInstructionsArbitraryDataTargetSize_c

decision instructions arbitrary data target size in bytes

gcDecisionDataPrandSize_c

decision data random part size in bytes

gcDecisionDataMaxSize_c

decision data maximum size in bytes

gcDecisionDataResolvableTagSize_c

decision data maximum size in bytes

gcMaxNumTestsInDecisionInstruction_c

The maximum number of tests in the Gap_SetDecisionInstructions ((max param length) - sizeof(numTests))/((sizeof(TestFlag) + sizeof(TestField) + sizeof(TestParameters)) = (255-1)/(1 + 1 + 16)

gcEadRandomizerSize_c
: Size of the Encrypted Advertising Data randomizer in bytes

gcEadMicSize_c
: Size of the Encrypted Advertising Data MIC in bytes

gcEadIvSize_c
: Size of the Encrypted Advertising Data Initialization Vector in bytes

gcEadKeySize_c
: Size of the Encrypted Advertising Data Key in bytes

gcEadAadValue_c
: Value of the Additional Authentication Data field for the CCM algorithm used by EAD

gBleMonAdvRssiThresholdMin_c
: Minimum RSSI threshold value for Monitored Advertisers

gBleMonAdvRssiThresholdMax_c
: Maximum RSSI threshold value for Monitored Advertisers

gLeModulationTestMaxBleRfChannel_c
: LE Modulation Test - Maximum BLE RF channel number (0-39)

gLeModulationTestMaxGenericChannel_c
: LE Modulation Test - Maximum generic channel number (0-127)

gLeModulationTestMaxPowerIndex_c
: LE Modulation Test - Maximum power index (0-62)

L2CA

enum l2caLeCbConnectionRequestResult_t
: *Values:*

enumerator gSuccessful_c

enumerator gLePsmNotSupported_c

enumerator gNoResourcesAvailable_c

enumerator gInsufficientAuthentication_c

enumerator gInsufficientAuthorization_c

enumerator gInsufficientEncryptionKeySize_c

enumerator gInsufficientEncryption_c

enumerator gInvalidSourceCid_c

enumerator gSourceCidAreadyAllocated_c

enumerator gUnacceptableParameters_c

enumerator gInvalidParameters_c

enumerator gCommandRejected_c

enumerator gResponseTimeout_c

enum l2caErrorSource_t

Values:

enumerator gL2ca_CancelConnection_c

enumerator gL2ca_SendLeFlowControlCredit_c

enumerator gL2ca_DisconnectLePsm_c

enumerator gL2ca_HandleSendLeCbData_c

enumerator gL2ca_HandleRecvLeCbData_c

enumerator gL2ca_HandleLeFlowControlCredit_c

enumerator gL2ca_EnhancedReconfigureReq_c

enumerator gL2ca_EnhancedCancelConnection_c

enum l2capReconfigureResponse_t

Values:

enumerator gReconfigurationSuccessful_c

enumerator gMtuReductionNotAllowed_c

enumerator gMultipleChannelMpsReductionNotAllowed_c

enumerator gDestinationCidInvalid_c

enumerator gUnacceptableReconfParameters_c

enumerator gReconfigurationTimeout_c

enumerator gReconfReserved_c

enum l2caChannelStatus_t

Values:

enumerator gL2ca_ChannelStatusChannelIdle_c

enumerator gL2ca_ChannelStatusChannelBusy_c

enum l2capControlMessageType_t

Values:

enumerator gL2ca_LePsmConnectRequest_c

enumerator gL2ca_LePsmConnectionComplete_c

enumerator gL2ca_LePsmDisconnectNotification_c

enumerator gL2ca_NoPeerCredits_c

enumerator gL2ca_LowPeerCredits_c

enumerator gL2ca_LocalCreditsNotification_c

enumerator gL2ca_Error_c

enumerator gL2ca_ChannelStatusNotification_c

enumerator gL2ca_LePsmEnhancedConnectRequest_c

enumerator gL2ca_LePsmEnhancedConnectionComplete_c

enumerator gL2ca_EnhancedReconfigureRequest_c

enumerator gL2ca_EnhancedReconfigureResponse_c

enumerator gL2ca_HandoverConnectionComplete_c

typedef struct *l2caLeCbConnectionRequest_tag* l2caLeCbConnectionRequest_t

typedef struct *l2caLeCbConnectionComplete_tag* l2caLeCbConnectionComplete_t

typedef struct *l2caHandoverConnectionComplete_tag* l2caHandoverConnectionComplete_t

typedef struct *l2caLeCbDisconnection_tag* l2caLeCbDisconnection_t

typedef struct *l2caLeCbNoPeerCredits_tag* l2caLeCbNoPeerCredits_t

typedef struct *l2caLeCbLowPeerCredits_tag* l2caLeCbLowPeerCredits_t

typedef struct *l2caLeCbLocalCreditsNotification_tag* l2caLeCbLocalCreditsNotification_t

typedef struct *l2caLeCbError_tag* l2caLeCbError_t

typedef struct *l2caLeCbChannelStatusNotification_tag* l2caLeCbChannelStatusNotification_t

typedef struct *l2caEnhancedConnectionRequest_tag* l2caEnhancedConnectionRequest_t

typedef struct *l2caEnhancedConnectionComplete_tag* l2caEnhancedConnectionComplete_t

typedef struct *l2caEnhancedReconfigureRequest_tag* l2caEnhancedReconfigureRequest_t

typedef struct *l2caEnhancedReconfigureResponse_tag* l2caEnhancedReconfigureResponse_t

typedef struct *l2capControlMessage_tag* l2capControlMessage_t

typedef void (*l2caLeCbDataCallback_t)(*deviceId_t* deviceId, uint16_t channelId, uint8_t *pPacket, uint16_t packetLength)

typedef void (*l2caControlCallback_t)(*l2capControlMessage_t* *pMessage)

typedef *l2caControlCallback_t* l2caLeCbControlCallback_t

typedef void (*l2caGenericCallback_t)(*deviceId_t* deviceId, uint8_t *pPacket, uint16_t packetLength)

typedef void (*l2caEattCallback_t)(*deviceId_t* deviceId, uint16_t channelId, uint8_t *pPacket, uint16_t packetLength)

bleResult_t L2ca_RegisterLeCbCallbacks(*l2caLeCbDataCallback_t* pCallback, *l2caLeCbControlCallback_t* pCtrlCallback)

Registers callbacks for credit based data and control events on L2CAP.

Parameters

- pCallback – **[in]** Callback function for data plane messages.
- pCtrlCallback – **[in]** Callback function for control plane messages.

Return values

- gL2caSuccess_c – Successful callbacks installation.
- gL2caCallbackAlreadyInstalled_c – Callbacks already installed.

bleResult_t L2ca_RegisterLePsm(uint16_t lePsm, uint16_t lePsmMtu)

Registers the LE_PSM from the L2CAP.

Parameters

- lePsm – **[in]** Bluetooth SIG or Vendor LE_PSM.
- lePsmMtu – **[in]** MTU of the registered PSM.

Return values

- gL2caLePsmInsufficientResources_c – No free LE_PSM registration slot was found.
- gL2caLePsmInvalid_c – Invalid LE_PSM value.
- gL2caLePsmAlreadyRegistered_c – LE_PSM already registered.
- gL2caSuccess_c – Successful LE_PSM registration.

bleResult_t L2ca_DeregisterLePsm(uint16_t lePsm)

Unregisters the LE_PSM from the L2CAP.

Parameters

- lePsm – **[in]** Bluetooth SIG or Vendor LE_PSM.

Return values

- gL2caSuccess_c – Successful LE_PSM deregistration.
- gL2caLePsmNotRegistered_c – LE_PSM value not registered.
- gL2caInternalError_c – There are connected L2CAP channels for the LE_PSM

Pre

An LE_PSM must be registered a priori.

bleResult_t L2ca_ConnectLePsm(uint16_t lePsm, *deviceId_t* deviceId, uint16_t initialCredits)

Initiates a connection with a peer device for a registered LE_PSM.

Parameters

- lePsm – **[in]** Bluetooth SIG or Vendor LE_PSM.
- deviceId – **[in]** The DeviceID for which the command is intended.
- initialCredits – **[in]** Initial credits.

Return values

- gL2caSuccess_c – Successful application to host message sent.
- gBleOutOfMemory_c – Message allocation fail.
- gBleInvalidParameter_c – Invalid Device Id.

Pre

An LE_PSM must be registered a priori.

bleResult_t L2ca_DisconnectLeCbChannel(*deviceId_t* deviceId, uint16_t channelId)

Disconnects a peer device for a registered LE_PSM.

Remark

Once this command is issued, all incoming data in transit for this device shall be discarded and any new additional outgoing data shall be discarded.

Parameters

- deviceId – **[in]** The DeviceID for which the command is intended.
- channelId – **[in]** The L2CAP Channel Id assigned on the initiator.

Return values

- gL2caSuccess_c – Successful application to host message sent.
- gBleOutOfMemory_c – Message allocation fail.
- gBleInvalidParameter_c – Invalid Device Id.

Pre

A connection must have already been created.

bleResult_t L2ca_CancelConnection(uint16_t lePsm, *deviceId_t* deviceId, *l2caLeCbConnectionRequestResult_t* refuseReason)

Terminates an L2CAP channel.

Remark

This interface can be used for a connection pending creation.

Parameters

- lePsm – **[in]** Bluetooth SIG or Vendor LE_PSM.
- deviceId – **[in]** The DeviceID for which the command is intended.
- refuseReason – **[in]** Reason to refuse the channel creation.

Return values

- gL2caSuccess_c – Successful application to host message sent.
- gBleOutOfMemory_c – Message allocation fail.
- gBleInvalidParameter_c – Invalid Device Id.

bleResult_t L2ca_SendLeCbData(*deviceId_t* deviceId, uint16_t channelId, const uint8_t *pPacket, uint16_t packetLength)

Sends a data packet through a Credit-Based Channel.

Parameters

- deviceId – **[in]** The DeviceID for which the command is intended.
- channelId – **[in]** The L2CAP Channel Id assigned on the initiator.
- pPacket – **[in]** Data buffer to be transmitted.
- packetLength – **[in]** Length of the data buffer.

Return values

- gL2caSuccess_c – Successful application to host message sent.
- gBleInvalidParameter_c – Invalid Device Id.
- gBleOverflow_c – Tx queue of the device is not empty.
- gBleOutOfMemory_c – Message or packet allocation fail.

Pre

An L2CAP Credit Based connection must be in place.

bleResult_t L2ca_SendLeCredit(*deviceId_t* deviceId, uint16_t channelId, uint16_t credits)

Sends credits to a device when capable of receiving additional LE-frames.

Parameters

- deviceId – **[in]** The DeviceID to which credits are given.
- channelId – **[in]** The L2CAP Channel Id assigned on the initiator.
- credits – **[in]** Number of credits to be given.

Return values

- gL2caSuccess_c – Successful application to host message sent.
- gBleOutOfMemory_c – Message allocation fail.
- gBleInvalidParameter_c – Invalid device id.

Pre

An L2CAP Credit Based connection must be in place.

bleResult_t L2ca_EnhancedConnectLePsm(uint16_t lePsm, *deviceId_t* deviceId, uint16_t mtu, uint16_t initialCredits, uint8_t noOfChannels, uint16_t *aCids)

Initiates a connection with a peer device for a registered LE_PSM. Enhanced mode: up to 5 channels can be opened.

Parameters

- lePsm – **[in]** Bluetooth SIG or Vendor LE_PSM.
- deviceId – **[in]** The DeviceID for which the command is intended.
- initialCredits – **[in]** Initial credits.
- noOfChannels – **[in]** Number of channels to request for opening. Max 5.
- aCids – **[in]** List of cids if an Enhanced Connection Request has been previously received. NULL if not.

Return values

- gL2caSuccess_c – Successful application to host message sent.
- gBleOutOfMemory_c – Message allocation fail.
- gBleInvalidParameter_c – Invalid device id or invalid noOfChannels.

Pre

A LE_PSM must be registered a priori.

bleResult_t L2ca_EnhancedChannelReconfigure(*deviceId_t* deviceId, uint16_t newMtu, uint16_t newMps, uint8_t noOfChannels, uint16_t *aCids)

Reconfigures up to 5 channels with new values for MTU and/or MPS.

Parameters

- deviceId – **[in]** The DeviceID for which the command is intended.
- newMtu – **[in]** New MTU value to be configured.
- newMps – **[in]** New MPS value to be configured.
- noOfChannels – **[in]** Number of channels to reconfigure. Max 5.
- *aCids – **[in]** The list of CIDs (endpoints on local device).

Return values

- gL2caSuccess_c – Successful application to host message sent.
- gBleOutOfMemory_c – Message allocation fail.

- gBleInvalidParameter_c – Invalid device id or invalid noOfChannels.

bleResult_t L2ca_EnhancedCancelConnection(uint16_t lePsm, *deviceId_t* deviceId, *l2caLeCbConnectionRequestResult_t* refuseReason, uint8_t noOfChannels, uint16_t *aCids)

Termination of pending L2CAP channels.

Remark

This interface can be used for an enhanced connection pending creation.

Parameters

- lePsm – **[in]** Bluetooth SIG or Vendor LE_PSM.
- deviceId – **[in]** The DeviceID for which the command is intended.
- refuseReason – **[in]** Reason to refuse the channel creation.
- noOfChannels – **[in]** Number of channels to disconnect. Max 5.
- *aCids – **[in]** The list of CIDs (endpoints on local device).

Return values

- gL2caSuccess_c – Successful application to host message sent.
- gBleOutOfMemory_c – Message allocation fail.
- gBleInvalidParameter_c – Invalid device id, invalid noOfChannels or invalid aCids pointer.

gL2capCidNull_c

gL2capCidAtt_c

gL2capCidSignaling_c

gL2capCidSmp_c

gL2capCidSigAssignedFirst_c

gL2capCidSigAssignedLast_c

gL2capCidLePsmDynamicFirst_c

gL2capCidLePsmDynamicLast_c

gL2capCidNotApplicable_c

gL2caLePsmSigAssignedFirst_c

gL2caLePsmSigAssignedEatt_c

gL2caLePsmSigAssignedLast_c

gL2caLePsmDynamicFirst_c

gL2caLePsmDynamicLast_c

gL2capDefaultMtu_c

gL2capDefaultMps_c

gL2capMaximumMps_c

gEnhancedL2capMinimumMps_c

gL2capHeaderLength_c

gExpandAsEnum_m(a, b, c)

gExpandAsTable_m(a, b, c)

gL2capEnhancedMaxChannels_c

gLePsmSigAssignedNumbersTable_m(entry)

struct l2caLeCbConnectionRequest_tag

#include <l2ca_cb_interface.h>

struct l2caLeCbConnectionComplete_tag

#include <l2ca_cb_interface.h>

struct l2caHandoverConnectionComplete_tag

#include <l2ca_cb_interface.h>

struct l2caLeCbDisconnection_tag

#include <l2ca_cb_interface.h>

struct l2caLeCbNoPeerCredits_tag

#include <l2ca_cb_interface.h>

struct l2caLeCbLowPeerCredits_tag

#include <l2ca_cb_interface.h>

struct l2caLeCbLocalCreditsNotification_tag

#include <l2ca_cb_interface.h>

struct l2caLeCbError_tag

#include <l2ca_cb_interface.h>

struct l2caLeCbChannelStatusNotification_tag
#include <l2ca_cb_interface.h>

struct l2caEnhancedConnectionRequest_tag
#include <l2ca_cb_interface.h>

struct l2caEnhancedConnectionComplete_tag
#include <l2ca_cb_interface.h>

struct l2caEnhancedReconfigureRequest_tag
#include <l2ca_cb_interface.h>

struct l2caEnhancedReconfigureResponse_tag
#include <l2ca_cb_interface.h>

struct l2capControlMessage_tag
#include <l2ca_cb_interface.h>

union messageData

Public Members

l2caLeCbConnectionRequest_t connectionRequest

l2caLeCbConnectionComplete_t connectionComplete

l2caLeCbDisconnection_t disconnection

l2caLeCbNoPeerCredits_t noPeerCredits

l2caLeCbLowPeerCredits_t lowPeerCredits

l2caLeCbLocalCreditsNotification_t localCreditsNotification

l2caLeCbError_t error

l2caLeCbChannelStatusNotification_t channelStatusNotification

l2caEnhancedConnectionRequest_t enhancedConnRequest

l2caEnhancedConnectionComplete_t enhancedConnComplete

l2caEnhancedReconfigureRequest_t reconfigureRequest

l2caEnhancedReconfigureResponse_t reconfigureResponse

l2caHandoverConnectionComplete_t handoverConnectionComplete

Files

ble_constants.h

gap_interface.h

ble_general.h

struct bleIdentityAddress_t

Public Members

bleAddressType_t idAddressType

bleDeviceAddress_t idAddress

union bleUuid_t

Public Members

uint16_t uuid16

uint32_t uuid32

uint8_t uuid128[16]

struct bleAdvertisingChannelMap_t

Public Members

uint8_t enableChannel37

uint8_t enableChannel38

uint8_t enableChannel39

uint8_t reserved

struct gapLeScOobData_t

Public Members

uint8_t randomValue[(16U)]

uint8_t confirmValue[(16U)]

struct gapInternalError_t

Public Members

bleResult_t errorCode

gapInternalErrorSource_t errorSource

uint16_t hciCommandOpcode

struct gapControllerTestEvent_t

Public Members

gapControllerTestEventType_t testEventType

uint16_t receivedPackets

struct gapPhyEvent_t

Public Members

gapPhyEventType_t phyEventType

deviceId_t deviceId

uint8_t txPhy

uint8_t rxPhy

struct bleNotificationEvent_t

Public Members

bleNotificationEventType_t eventType

deviceId_t deviceId

int8_t rssi

uint8_t channel

uint16_t ce_counter

bleResult_t status

uint32_t timestamp

uint8_t adv_handle

bleDeviceAddress_t scanned_addr

struct gapInitComplete_t

Public Members

leSupportedFeatures_t supportedFeatures

uint8_t leExtendedFeatures[(2U)]

uint16_t maxAdvDataSize

uint8_t numOfSupportedAdvSets

uint8_t periodicAdvListSize

struct bleBondCreatedEvent_tag

Public Members

uint8_t nvmIndex

bleAddressType_t addressType

bleDeviceAddress_t address

struct gapAddrReadyEvent_t_tag

Public Members

bleDeviceAddress_t aAddress

uint8_t advHandle

struct bleSupportedSwitchingSamplingRates_t

Public Members

uint8_t switchingSupportedAodTransmission

uint8_t samplingSupportedAodReception

uint8_t switchingSamplingSupportedAoaReception

uint8_t reserved

struct bleAntennaInformation_tag

Public Members

bleSupportedSwitchingSamplingRates_t supportedSwitchingSamplingRates

uint8_t numAntennae

uint8_t maxSwitchingPatternLength

uint8_t maxCteLength

struct periodicAdvSyncTransferEvent_tag

Public Members

deviceId_t deviceId

bleResult_t status

struct periodicAdvSetInfoTransferEvent_tag

Public Members

deviceId_t deviceId

bleResult_t status

struct periodicAdvSetSyncTransferParamsEvent_tag

Public Members

deviceId_t deviceId

bleResult_t status

struct gapSyncTransferReceivedEventData_tag

Public Members

bleResult_t status

deviceId_t deviceId

uint16_t serviceData

uint16_t syncHandle

uint8_t advSID

bleAddressType_t advAddressType

bleDeviceAddress_t advAddress

gapLePhyMode_t advPhy

uint16_t periodicAdvInt

bleCentralClockAccuracy_t advClockAccuracy

uint8_t numSubevents

uint8_t subeventInterval

uint8_t responseSlotDelay

uint8_t responseSlotSpacing

struct getConnParams_tag

Public Members

uint16_t connectionHandle

uint32_t ulTxAccCode

uint8_t aCrcInitVal[3U]

uint16_t uiConnInterval

uint16_t uiSuperTO

uint16_t uiConnLatency

uint8_t aChMapBm[5U]

uint8_t ucChannelSelection

uint8_t ucHop

uint8_t ucUnMapChIdx

uint8_t ucCentralSCA

uint8_t ucRole

uint8_t aucRemoteMasRxPHY

uint8_t seqNum

uint16_t uiConnEvent

uint32_t ulAnchorClk

uint16_t uiAnchorDelay

uint32_t ulRxInstant

struct handoverGetTime_tag

Public Members

bleResult_t status

uint32_t slot

uint16_t us_offset

struct handoverAnchorSearchStart_tag

Public Members

bleResult_t status

uint16_t connectionHandle

struct handoverAnchorSearchStop_tag

Public Members

bleResult_t status

uint16_t connectionHandle

struct handoverConnect_tag

Public Members

bleResult_t status

uint16_t connectionHandle

struct handoverGetData_tag

Public Members

bleResult_t status

uint32_t *pData

struct handoverSetData_tag

Public Members

bleResult_t status

uint32_t *pData

struct handoverGetCsLlContext_tag

Public Members

bleResult_t status

uint16_t responseMask

uint8_t llContextLength

uint8_t llContext[(224U)]

struct handoverAnchorMonitorEvent_tag

Public Members

uint16_t connectionHandle

uint16_t connEvent

int8_t rssiRemote

uint8_t lqiRemote

uint8_t statusRemote

int8_t rssiActive

uint8_t lqiActive

uint8_t statusActive

uint32_t anchorClock625Us

uint16_t anchorDelay

uint8_t chIdx

uint8_t ucNbReports

struct handoverAnchorMonitorPacketEvent_tag

Public Members

uint8_t packetCounter

uint16_t connectionHandle

uint8_t statusPacket

uint8_t phy

uint8_t chIdx

int8_t rssiPacket

uint8_t lqiPacket

uint16_t connEvent

uint32_t anchorClock625Us

uint16_t anchorDelay

uint8_t ucNbConnIntervals

uint8_t pduSize

uint8_t *pPdu

struct handoverAnchorMonitorPacketContinueEvent_tag

Public Members

uint8_t packetCounter

uint16_t connectionHandle

uint8_t pduSize

uint8_t *pPdu

struct handoverConnectionUpdateProcedureEvent_tag

Public Members

uint16_t connectionHandle

uint8_t winSize

uint16_t winOffset

uint16_t interval

uint16_t latency

uint16_t timeout

uint16_t instant

uint16_t currentEventCounter

struct handoverTimeSyncEvent_tag

Public Members

uint32_t txClkSlot

uint16_t txUs

uint32_t rxClkSlot

uint16_t rxUs

uint8_t rssi

struct handoverConnParamUpdateEvent_tag

Public Members

uint8_t status

uint16_t connectionHandle

uint32_t ulTxAccCode

uint8_t aCrcInitVal[3U]

uint16_t uiConnInterval

uint16_t uiSuperTO

uint16_t uiConnLatency

uint8_t aChMapBm[5U]

uint8_t ucChannelSelection

uint8_t ucHop

uint8_t ucUnMapChIdx

uint8_t ucCentralSCA

uint8_t ucRole

uint8_t aucRemoteMasRxPHY

uint8_t seqNum

uint16_t uiConnEvent

uint32_t ulAnchorClk

uint16_t uiAnchorDelay

uint32_t ulRxInstant

struct handoverSuspendTransmitCompleteEvent_tag

Public Members

uint16_t connectionHandle

uint8_t noOfPendingAclPackets

uint16_t sizeOfPendingAclPackets

uint16_t sizeOfDataTxInOldestPacket

uint8_t sizeOfDataNAckInOldestPacket

struct handoverResumeTransmitCompleteEvent_tag

Public Members

uint16_t connectionHandle

struct handoverUpdateConnParams_tag

Public Members

bleResult_t status

uint16_t connectionHandle

struct handoverApplyConnectionUpdateProcedure_tag

Public Members

uint16_t connectionHandle

struct handoverAnchorNotificationStateChanged_tag

Public Members

uint16_t connectionHandle

struct handoverLlPendingDataIndication_tag

Public Members

uint16_t dataSize

uint8_t *pData

struct gapLEGenerateDhKeyCompleteEvent_t

Public Members

uint8_t aDHKey[(32U)]

struct gapRemoteVersionInfoRead_tag

Public Members

bleResult_t status

deviceId_t deviceId

uint8_t version

uint16_t manufacturerName

uint16_t subversion

struct gapLlSkdReport_tag

Public Members

deviceId_t deviceId

uint8_t aSKD[(16U)]

struct vendorUnitaryTestEvent_tag

Public Members

bleResult_t status

uint8_t paramLength

uint8_t aParam[(255U)]

struct vendorSpecificDbgEvent_tag

Public Members

uint8_t dataSize

uint8_t pData[(254U)]

struct gapGenericEvent_t

Public Members

gapGenericEventType_t eventType

union *gapGenericEvent_t* eventData

struct bleBondIdentityHeaderBlob_t

Public Members

uint32_t raw[(gBleBondIdentityHeaderSize_c + 3U) / sizeof(uint32_t)]

struct bleBondDataDynamicBlob_t

Public Members

uint32_t raw[((8U) + 3U) / sizeof(uint32_t)]

struct bleBondDataStaticBlob_t

Public Members

uint32_t raw[((30U) + 3U) / sizeof(uint32_t)]

struct bleBondDataLegacyBlob_t

Public Members

uint32_t raw[((12U) + 3U) / sizeof(uint32_t)]

struct bleBondDataDeviceInfoBlob_t

Public Members

uint32_t raw[((60U) + 3U) / sizeof(uint32_t)]

struct bleBondDataDescriptorBlob_t

Public Members

uint32_t raw[((4U) + 3U) / sizeof(uint32_t)]

struct bleLocalKeysBlob_t

Public Members

union bleLocalKeysBlob_t

bool_t keyGenerated

struct bleBondDataBlob_t

Public Members

bleBondIdentityHeaderBlob_t bondHeader

bleBondDataDynamicBlob_t bondDataBlobDynamic

bleBondDataStaticBlob_t bondDataBlobStatic

bleBondDataLegacyBlob_t bondDataBlobLegacy

bleBondDataDescriptorBlob_t bondDataDescriptors[(5U)]

bleBondDataDeviceInfoBlob_t bondDataBlobDeviceInfo

struct bleBondDataRam_t

Public Members

bleBondIdentityHeaderBlob_t bondHeader

bleBondDataDynamicBlob_t bondDataBlobDynamic

bleBondDataStaticBlob_t bondDataBlobStatic

bleBondDataLegacyBlob_t bondDataBlobLegacy

bleBondDataDeviceInfoBlob_t bondDataBlobDeviceInfo

bleBondDataDescriptorBlob_t bondDataDescriptors[1]

struct bleCteAllowedTypesMap_t

Public Members

uint8_t allowAoA

uint8_t allowAoD1us

uint8_t allowAoD2us

uint8_t reserved

struct bleSyncCteType_tag

Public Members

uint8_t doNotSyncWithAoA

uint8_t doNotSyncWithAoD1us

uint8_t doNotSyncWithAoD2us

uint8_t doNotSyncWithType3

uint8_t doNotSyncWithoutCte

uint8_t reserved

struct bleTxPowerLevelFlags_t

Public Members

uint8_t minimum

uint8_t maximum

uint8_t reserved

union eventData

Public Members

gapInternalError_t internalError

uint8_t filterAcceptListSize

bleDeviceAddress_t aAddress

gapAddrReadyEvent_t addrReady

uint8_t advHandle

bleResult_t setupFailError

int8_t advTxPowerLevel_dBm

bool_t verified

gapLeScOobData_t localOobData

bool_t newHostPrivacyState

bool_t newControllerPrivacyState

gapControllerTestEvent_t testEvent

bleResult_t txPowerLevelSetStatus

gapPhyEvent_t phyEvent

deviceId_t deviceId

gapInitComplete_t initCompleteData

bleNotificationEvent_t notifEvent

bleBondCreatedEvent_t bondCreatedEvent

bleDeviceAddress_t aControllerLocalRPA

getConnParams_t getConnParams

uint16_t syncHandle

bleAntennaInformation_t antennaInformation

bleResult_t perAdvSyncTransferEnable

periodicAdvSyncTransferEvent_t perAdvSyncTransfer

periodicAdvSetInfoTransferEvent_t perAdvSetInfoTransfer

periodicAdvSetSyncTransferParamsEvent_t perAdvSetSyncTransferParams

bleResult_t perAdvSetDefaultPerAdvSyncTransferParams

gapSyncTransferReceivedEventData_t perAdvSyncTransferReceived

gapLEGenerateDhKeyCompleteEvent_t leGenerateDhKeyCompleteEvent

uint16_t pawrSyncHandle

uint8_t pawrAdvHandle

handoverGetData_t handoverGetData

handoverSetData_t handoverSetData

handoverGetCsLlContext_t handoverGetCsLlContext

handoverGetTime_t handoverGetTime

handoverConnect_t handoverConnect

handoverAnchorSearchStart_t handoverAnchorSearchStart

handoverAnchorSearchStop_t handoverAnchorSearchStop

handoverAnchorMonitorEvent_t handoverAnchorMonitor

handoverTimeSyncEvent_t handoverTimeSync

handoverSuspendTransmitCompleteEvent_t handoverSuspendTransmitComplete

handoverResumeTransmitCompleteEvent_t handoverResumeTransmitComplete

handoverAnchorNotificationStateChanged_t handoverAnchorNotificationStateChanged

handoverConnParamUpdateEvent_t handoverConnParamUpdate

gapRemoteVersionInfoRead_t gapRemoteVersionInfoRead

gapLlSkdReport_t gapSkdReport

handoverAnchorMonitorPacketEvent_t handoverAnchorMonitorPacket

handoverAnchorMonitorPacketContinueEvent_t handoverAnchorMonitorPacketContinue

handoverUpdateConnParams_t handoverUpdateConnParams

handoverLlPendingDataIndication_t handoverLlPendingDataIndication

handoverConnectionUpdateProcedureEvent_t handoverConnectionUpdateProcedure

handoverApplyConnectionUpdateProcedure_t handoverApplyConnectionUpdateProcedure

vendorUnitaryTestEvent_t unitaryTestData

uint8_t monAdvListSize

vendorSpecificDbgEvent_t vendorDebug

union ___unnamed2___

Public Members

uint32_t raw[((40U) + 3U) / sizeof(uint32_t)]

uint8_t pKey[(40U)]

ble_host_tasks.h

ble_sig_defines.h

ble_utils.h

gap_types.h

struct gapSmpKeys_t

Public Members

uint8_t cLtkSize

uint8_t *aLtk

uint8_t *aIrk

uint8_t *aCsrk

uint8_t cRandSize

uint8_t *aRand

uint16_t ediv

bleAddressType_t addressType

uint8_t *aAddress

struct gapSecurityRequirements_t

Public Members

gapSecurityModeAndLevel_t securityModeLevel

bool_t authorization

uint16_t minimumEncryptionKeySize

struct gapServiceSecurityRequirements_t

Public Members

uint16_t serviceHandle

gapSecurityRequirements_t requirements

struct gapDeviceSecurityRequirements_t

Public Members

gapSecurityRequirements_t *pSecurityRequirements

uint8_t cNumServices

gapServiceSecurityRequirements_t *aServiceSecurityRequirements

struct gapHandleList_t

Public Members

uint8_t cNumHandles

uint16_t aHandles[(1U)]

struct gapConnectionSecurityInformation_t

Public Members

bool_t authenticated

gapHandleList_t authorizedToRead

gapHandleList_t authorizedToWrite

struct gapPairingParameters_t

Public Members

bool_t withBonding

gapSecurityModeAndLevel_t securityModeAndLevel

uint8_t maxEncryptionKeySize

gapIoCapabilities_t localIoCapabilities

bool_t oobAvailable

gapSmpKeyFlags_t centralKeys

gapSmpKeyFlags_t peripheralKeys

bool_t leSecureConnectionSupported

bool_t useKeypressNotifications

struct gapPeripheralSecurityRequestParameters_t

Public Members

bool_t bondAfterPairing

bool_t authenticationRequired

struct gapAdvertisingParameters_t

Public Members

uint16_t minInterval

uint16_t maxInterval

bleAdvertisingType_t advertisingType

bleAddressType_t ownAddressType

bleAddressType_t peerAddressType

bleDeviceAddress_t peerAddress

gapAdvertisingChannelMapFlags_t channelMap

gapAdvertisingFilterPolicy_t filterPolicy

struct gapExtAdvertisingParameters_tag

Public Members

uint8_t SID

uint8_t handle

uint32_t minInterval

uint32_t maxInterval

bleAddressType_t ownAddressType

bleDeviceAddress_t ownRandomAddr

bleAddressType_t peerAddressType

bleDeviceAddress_t peerAddress

gapAdvertisingChannelMapFlags_t channelMap

gapAdvertisingFilterPolicy_t filterPolicy

bleAdvRequestProperties_t extAdvProperties

int8_t txPower

gapLePhyMode_t primaryPHY

gapLePhyMode_t secondaryPHY

uint8_t secondaryAdvMaxSkip

bool_t enableScanReqNotification

gapLePhyOptionsFlags_t primaryAdvPhyOptions

gapLePhyOptionsFlags_t secondaryAdvPhyOptions

struct gapPeriodicAdvParameters_tag

Public Members

uint8_t handle

bool_t addTxPowerInAdv

uint16_t minInterval

uint16_t maxInterval

uint8_t numSubevents

uint8_t subeventInterval

uint8_t responseSlotDelay

uint8_t responseSlotSpacing

uint8_t numResponseSlots

struct gapPeriodicAdvSyncTransfer_tag

Public Members

deviceId_t deviceId

uint16_t serviceData

uint16_t syncHandle

struct gapPeriodicAdvSetInfoTransfer_tag

Public Members

deviceId_t deviceId

uint16_t serviceData

uint16_t advHandle

struct gapSetPeriodicAdvSyncTransferParams_tag

Public Members

deviceId_t deviceId

gapPeriodicAdvSyncMode_t mode

uint16_t skip

uint16_t syncTimeout

bleSyncCteType_t CTEType

struct gapScanningParameters_t

Public Members

bleScanType_t type

uint16_t interval

uint16_t window

bleAddressType_t ownAddressType

bleScanningFilterPolicy_t filterPolicy

gapLePhyFlags_t scanningPHYs

struct gapCreateSyncReqOptions_tag

Public Members

gapCreateSyncReqFilterPolicy_t filterPolicy

uint8_t reportingEnabled

uint8_t duplicateFilteringEnabled

struct gapPeriodicAdvSyncReq_tag

Public Members

gapCreateSyncReqOptions_t options

uint8_t SID

bleAddressType_t peerAddressType

bleDeviceAddress_t peerAddress

uint16_t skipCount

uint16_t timeout

bleSyncCteType_t cteType

struct gapConnectionRequestParameters_t

Public Members

uint16_t scanInterval

uint16_t scanWindow

bleInitiatorFilterPolicy_t filterPolicy

bleAddressType_t ownAddressType

bleAddressType_t peerAddressType

bleDeviceAddress_t peerAddress

uint16_t connIntervalMin

uint16_t connIntervalMax

uint16_t connLatency

uint16_t supervisionTimeout

uint16_t connEventLengthMin

uint16_t connEventLengthMax

bool_t usePeerIdentityAddress

gapLePhyFlags_t initiatingPHYs

struct gapConnectionFromPawrParameters_t

Public Members

uint16_t scanInterval

uint16_t scanWindow

bleInitiatorFilterPolicy_t filterPolicy

bleAddressType_t ownAddressType

bleAddressType_t peerAddressType

bleDeviceAddress_t peerAddress

uint16_t connIntervalMin

uint16_t connIntervalMax

uint16_t connLatency

uint16_t supervisionTimeout

uint16_t connEventLengthMin

uint16_t connEventLengthMax

bool_t usePeerIdentityAddress

gapLePhyFlags_t initiatingPHYs

uint8_t advHandle

uint8_t subevent

struct gapConnectionParameters_t

Public Members

uint16_t connInterval

uint16_t connLatency

uint16_t supervisionTimeout

bleCentralClockAccuracy_t centralClockAccuracy

struct gapConnectionSubrateParameters_t

Public Members

uint16_t subrateMin

uint16_t subrateMax

uint16_t latencyMax

uint16_t continuationNumber

uint16_t supervisionTimeout

struct gapGenerateDHKeyV2Params_t

Public Members

ecdhPublicKey_t remoteP256PublicKey

gapPrivateKeyType_t keyType

struct gapConnectionlessCteTransmitParams_t

Public Members

uint8_t handle

uint8_t cteLength

bleCteType_t cteType

uint8_t cteCount

uint8_t switchingPatternLength

uint8_t aAntennaIds[1]

struct gapConnectionlessIqSamplingParams_t

Public Members

bleIqSamplingEnable_t iqSamplingEnable

bleSlotDurations_t slotDurations

uint8_t maxSampledCtes

uint8_t switchingPatternLength

uint8_t aAntennaIds[1]

struct gapConnectionCteTransmitParams_t

Public Members

bleCteAllowedTypesMap_t cteTypes

uint8_t switchingPatternLength

uint8_t aAntennaIds[1]

struct gapConnectionCteReceiveParams_t

Public Members

bleIqSamplingEnable_t iqSamplingEnable

bleSlotDurations_t slotDurations

uint8_t switchingPatternLength

uint8_t aAntennaIds[1]

struct gapConnectionCteReqEnableParams_t

Public Members

bleCteReqEnable_t cteReqEnable

uint16_t cteReqInterval

uint8_t requestedCteLength

bleCteType_t requestedCteType

struct gapPathLossReportingParams_t

Public Members

uint8_t highThreshold

uint8_t highHysteresis

uint8_t lowThreshold

uint8_t lowHysteresis

uint16_t minTimeSpent

struct gapAdStructure_t

Public Members

uint8_t length

gapAdType_t adType

uint8_t *aData

struct gapAdvertisingData_t

Public Members

uint8_t cNumAdStructures

gapAdStructure_t *aAdStructures

struct gapAdvertisingDecisionData_t

Public Members

uint8_t *pKey

uint8_t *pPrand

uint8_t *pDecisionData

uint8_t dataLength

bool_t resolvableTagPresent

struct gapDecisionInstructionsData_tag

Public Members

gapDecisionInstructionsTestGroup_t testGroup

gapDecisionInstructionsTestPassCriteria_t passCriteria

gapDecisionInstructionsRelevantField_t relevantField

union *gapDecisionInstructionsData_tag* testParameters

struct gapSubeventDataStructure_t

Public Members

uint8_t subevent

uint8_t responseSlotStart

uint8_t responseSlotCount

gapAdvertisingData_t *pAdvertisingData

struct gapPeriodicAdvertisingSubeventData_t

Public Members

uint8_t cNumSubevents

gapSubeventDataStructure_t *aSubeventDataStructures

struct gapPeriodicAdvertisingResponseData_t

Public Members

uint16_t requestEvent

uint8_t requestSubevent

uint8_t responseSubevent

uint8_t responseSlot

gapAdvertisingData_t *pResponseData

struct gapPeriodicSyncSubeventParameters_t

Public Members

uint16_t perAdvProperties

uint8_t numSubevents

uint8_t aSubevents[1]

struct gapExtScanNotification_t

Public Members

uint8_t handle

bleAddressType_t scannerAddrType

bleDeviceAddress_t aScannerAddr

bool_t scannerAddrResolved

struct gapAdvertisingSetTerminated_t

Public Members

bleResult_t status

uint8_t handle

deviceId_t deviceId

uint8_t numCompletedExtAdvEvents

struct gapPerAdvSubeventDataRequest_t

Public Members

uint8_t handle

uint8_t subeventStart

uint8_t subeventDataCount

struct gapPerAdvResponse_t

Public Members

uint8_t advHandle

uint8_t subevent

int8_t txPower

int8_t rssi

uint8_t cteType

uint8_t responseSlot

uint8_t dataLength

uint8_t *pData

struct gapAdvertisingEvent_t

Public Members

gapAdvertisingEventType_t eventType

union *gapAdvertisingEvent_t* eventData

struct gapScannedDevice_t

Public Members

bleAddressType_t addressType

bleDeviceAddress_t aAddress

int8_t rssi

uint8_t dataLength

uint8_t *data

bleAdvertisingReportEventType_t advEventType

bool_t directRpaUsed

bleDeviceAddress_t directRpa

bool_t advertisingAddressResolved

struct gapExtScannedDevice_t

Public Members

bleAddressType_t addressType

bleDeviceAddress_t aAddress

uint8_t SID

bool_t advertisingAddressResolved

bleAdvReportEventProperties_t advEventProperties

int8_t rssi

int8_t txPower

uint8_t primaryPHY

uint8_t secondaryPHY

uint16_t periodicAdvInterval

bool_t directRpaUsed

bleAddressType_t directRpaType

bleDeviceAddress_t directRpa

uint16_t dataLength

uint8_t *pData

struct gapPeriodicScannedDevice_t

Public Members

uint16_t syncHandle

int8_t txPower

int8_t rssi

bleCteType_t cteType

uint16_t dataLength

uint8_t *pData

struct gapPeriodicScannedDeviceV2_t

Public Members

uint16_t syncHandle

int8_t txPower

int8_t rssi

bleCteType_t cteType

uint16_t periodicEventCounter

uint8_t subevent

uint16_t dataLength

uint8_t *pData

struct gapSyncEstbEventData_t

Public Members

bleResult_t status

uint16_t syncHandle

uint8_t SID

bleAddressType_t peerAddressType

bleDeviceAddress_t peerAddress

gapLePhyMode_t PHY

uint16_t periodicAdvInterval

bleAdvertiserClockAccuracy_t advertiserClockAccuracy

uint8_t numSubevents

uint8_t subeventInterval

uint8_t responseSlotDelay

uint8_t responseSlotSpacing

struct gapSyncLostEventData_t

Public Members

uint16_t syncHandle

struct gapConnectionlessIqReport_t

Public Members

uint16_t syncHandle

uint8_t channelIndex

int16_t rssi

uint8_t rssiAntennaId

bleCteType_t cteType

bleSlotDurations_t slotDurations

bleIqReportPacketStatus_t packetStatus

uint16_t periodicEventCounter

uint8_t sampleCount

int8_t *aI_samples

int8_t *aQ_samples

struct gapMonAdvReportReport_t

Public Members

bleAddressType_t peerAddressType

bleDeviceAddress_t peerAddress

bleMonAdvCondition_t condition

struct gapScanningEvent_t

Public Members

gapScanningEventType_t eventType

union *gapScanningEvent_t* eventData

struct gapConnectedEvent_t

Public Members

gapConnectionParameters_t connParameters

bleAddressType_t peerAddressType

bleDeviceAddress_t peerAddress

bool_t peerRpaResolved

bleDeviceAddress_t peerRpa

bool_t localRpaUsed

bleDeviceAddress_t localRpa

bleLlConnectionRole_t connectionRole

uint8_t advHandle

uint16_t syncHandle

struct gapKeyExchangeRequestEvent_t

Public Members

gapSmpKeyFlags_t requestedKeys

uint8_t requestedLtkSize

struct gapKeysReceivedEvent_t

Public Members

gapSmpKeys_t *pKeys

struct gapAuthenticationRejectedEvent_t

Public Members

gapAuthenticationRejectReason_t rejectReason

struct gapPairingCompleteEvent_t

Public Members

bool_t pairingSuccessful

union *gapPairingCompleteEvent_t* pairingCompleteData

struct gapLongTermKeyRequestEvent_t

Public Members

uint16_t ediv

uint8_t aRand[(8U)]

uint8_t randSize

struct gapEncryptionChangedEvent_t

Public Members

bool_t newEncryptionState

struct gapDisconnectedEvent_t

Public Members

gapDisconnectionReason_t reason

struct gapConnParamsUpdateReq_t

Public Members

uint16_t intervalMin

uint16_t intervalMax

uint16_t peripheralLatency

uint16_t timeoutMultiplier

struct gapConnParamsUpdateComplete_t

Public Members

bleResult_t status

uint16_t connInterval

uint16_t connLatency

uint16_t supervisionTimeout

struct gapConnLeDataLengthChanged_t

Public Members

uint16_t maxTxOctets

uint16_t maxTxTime

uint16_t maxRxOctets

uint16_t maxRxTime

struct gapConnIqReport_t

Public Members

gapLePhyMode_t rxPhy

uint8_t dataChannelIndex

int16_t rssi

uint8_t rssiAntennaId

bleCteType_t cteType

bleSlotDurations_t slotDurations

bleIqReportPacketStatus_t packetStatus

uint16_t connEventCounter

uint8_t sampleCount

int8_t *aI_samples

int8_t *aQ_samples

struct gapConnCteRequestFailed_t

Public Members

bleResult_t status

struct gapPathLossThresholdEvent_t

Public Members

uint8_t currentPathLoss

blePathLossThresholdZoneEntered_t zoneEntered

struct gapTransmitPowerReporting_t

Public Members

bleTxPowerReportingReason_t reason

blePowerControlPhyType_t phy

int8_t txPowerLevel

bleTxPowerLevelFlags_t flags

int8_t delta

struct gapSubrateChangeEvent_t

Public Members

bleResult_t status

uint16_t subrateFactor

uint16_t peripheralLatency

uint16_t continuationNumber

uint16_t supervisionTimeout

struct gapTransmitPowerInfo_t

Public Members

blePowerControlPhyType_t phy

int8_t currTxPowerLevel

int8_t maxTxPowerLevel

struct gapEattConnectionRequest_t

Public Members

uint16_t mtu

uint8_t cBearers

uint16_t initialCredits

struct gapEattConnectionComplete_t

Public Members

l2caLeCbConnectionRequestResult_t status

uint16_t mtu

uint8_t cBearers

bearerId_t aBearerIds[(5U)]

struct gapEattReconfigureResponse_t

Public Members

l2capReconfigureResponse_t status

uint16_t localMtu

uint8_t cBearers

bearerId_t aBearerIds[(5U)]

struct gapEattBearerStatusNotification_t

Public Members

bearerId_t bearerId

gapEattBearerStatus_t status

struct gapHandoverConnectedEvent_t

Public Members

bleAddressType_t peerAddressType

bleDeviceAddress_t peerAddress

bleLlConnectionRole_t connectionRole

struct gapHandoverDisconnectedEvent_t

Public Members

bleResult_t status

struct gapConnectionEvent_t

Public Members

gapConnectionEventType_t eventType

union *gapConnectionEvent_t* eventData

struct gapIdentityInformation_t

Public Members

bleIdentityAddress_t identityAddress

uint8_t irk[(16U)]

blePrivacyMode_t privacyMode

struct gapAutoConnectParams_t

Public Members

uint8_t cNumAddresses

bool_t writeInFilterAcceptList

gapConnectionRequestParameters_t *aAutoConnectData

struct gapHostVersion_tag

Public Members

uint8_t bleHostVerMajor

uint8_t bleHostVerMinor

uint8_t bleHostVerPatch

union testParameters

Public Members

uint8_t resolvableTagKey[(16U)]

struct *gapDecisionInstructionsData_tag* arbitraryData

struct *gapDecisionInstructionsData_tag* rssi

struct *gapDecisionInstructionsData_tag* pathLoss

struct *gapDecisionInstructionsData_tag* advA

gapDecisionInstructionsAdvMode_t advMode

struct arbitraryData

Public Members

uint8_t mask[(8U)]

uint8_t target[(8U)]

struct rssi

Public Members

int8_t min

int8_t max

struct pathLoss

Public Members

uint8_t min

uint8_t max

struct advA

Public Members

gapDecisionInstructionsAdvAChecks_t check

bleAddressType_t address1Type

bleDeviceAddress_t address1

bleAddressType_t address2Type

bleDeviceAddress_t address2

union eventData

Public Members

bleResult_t failReason

gapExtScanNotification_t scanNotification

gapAdvertisingSetTerminated_t advSetTerminated

gapPerAdvSubeventDataRequest_t subeventDataRequest

gapPerAdvResponse_t perAdvResponse

uint8_t advHandle

union eventData

Public Members

bleResult_t failReason

gapScannedDevice_t scannedDevice

gapExtScannedDevice_t extScannedDevice

gapPeriodicScannedDevice_t periodicScannedDevice

gapPeriodicScannedDeviceV2_t periodicScannedDeviceV2

gapSyncEstbEventData_t syncEstb

gapSyncLostEventData_t syncLost

gapConnectionlessIqReport_t iqReport

gapMonAdvReportReport_t monAdvReport

union pairingCompleteData

Public Members

bool_t withBonding

bleResult_t failReason

union eventData

Public Members

gapConnectedEvent_t connectedEvent

gapPairingParameters_t pairingEvent

gapAuthenticationRejectedEvent_t authenticationRejectedEvent

gapPeripheralSecurityRequestParameters_t peripheralSecurityRequestEvent

gapKeyExchangeRequestEvent_t keyExchangeRequestEvent

gapKeysReceivedEvent_t keysReceivedEvent

gapPairingCompleteEvent_t pairingCompleteEvent

gapLongTermKeyRequestEvent_t longTermKeyRequestEvent

gapEncryptionChangedEvent_t encryptionChangedEvent

gapDisconnectedEvent_t disconnectedEvent

int8_t rssi_dBm

int8_t txPowerLevel_dBm

bleResult_t failReason

uint32_t passkeyForDisplay

gapConnParamsUpdateReq_t connectionUpdateRequest

gapConnParamsUpdateComplete_t connectionUpdateComplete

gapConnLeDataLengthChanged_t leDataLengthChanged

gapKeypressNotification_t incomingKeypressNotification

uint32_t numericValueForDisplay

bleChannelMap_t channelMap

gapConnIqReport_t connIqReport

gapConnCteRequestFailed_t cteRequestFailedEvent

bleResult_t perAdvSyncTransferStatus

gapPathLossThresholdEvent_t pathLossThreshold

gapTransmitPowerReporting_t transmitPowerReporting

gapTransmitPowerInfo_t transmitPowerInfo

gapEattConnectionRequest_t eattConnectionRequest

gapEattConnectionComplete_t eattConnectionComplete

gapEattReconfigureResponse_t eattReconfigureResponse

gapEattBearerStatusNotification_t eattBearerStatusNotification

gapHandoverConnectedEvent_t handoverConnectedEvent

gapHandoverDisconnectedEvent_t handoverDisconnectedEvent

bleResult_t smError

gapSubrateChangeEvent_t gapSubrateChangeEvent

uint8_t remoteFeatures[8U]

att_errors.h

gatt_types.h

struct gattAttribute_t

Public Members

uint16_t handle

bleUuidType_t uuidType

bleUuid_t uuid

uint16_t valueLength

uint16_t maxValueLength

uint8_t *paValue

struct gattCharacteristic_t

Public Members

gattCharacteristicPropertiesBitFields_t properties

gattAttribute_t value

uint8_t cNumDescriptors

gattAttribute_t *aDescriptors

struct gattService_tag

Public Members

uint16_t startHandle

uint16_t endHandle

bleUuidType_t uuidType

bleUuid_t uuid

uint8_t cNumCharacteristics

gattCharacteristic_t *aCharacteristics

uint8_t cNumIncludedServices

struct *gattService_tag* *aIncludedServices

struct gattDbCharPresFormat_t

Public Members

uint8_t format

int8_t exponent

uint16_t unitUuid16

uint8_t ns

uint16_t description

struct gattHandleRange_t

Public Members

uint16_t startHandle

uint16_t endHandle

struct procStatus_t

Public Members

bool_t isOngoing

gattProcedureType_t ongoingProcedureType

gattProcedurePhase_t ongoingProcedurePhase

struct procDataStruct_t

Public Members

uint16_t index

uint16_t max

bleUuid_t charUuid

bleUuidType_t charUuidType

bool_t reliableLongWrite

bool_t bAllocatedArray

union *procDataStruct_t* pOutActualCount

union *procDataStruct_t* array

union *procDataStruct_t* reqParams

union pOutActualCount

Public Members

uint8_t *pCount8b

uint16_t *pCount16b

union array

Public Members

gattService_t *aServices

gattCharacteristic_t *aChars

gattAttribute_t *aDescriptors

uint8_t *aBytes

uint16_t *aHandles

union reqParams

Public Members

attReadByGroupTypeRequestParams_t rbgtParams

attFindByTypeValueRequestParams_t fbtvParams

attReadByTypeRequestParams_t rbtParams

attFindInformationRequestParams_t fiParams

attReadRequestParams_t rParams

attReadBlobRequestParams_t rbParams

attReadMultipleRequestParams_t rmParams

attVarWriteRequestAndCommandParams_t wParams

attPrepareWriteRequestResponseParams_t pwParams

attExecuteWriteRequestParams_t ewParams

gatt_interface.h

gatt_client_interface.h

gatt_server_interface.h

struct gattServerMtuChangedEvent_t

Public Members

uint16_t newMtu

struct gattServerAttributeWrittenEvent_t

Public Members

uint16_t handle

uint16_t cValueLength

uint8_t *aValue

bearerId_t bearerId

struct gattServerLongCharacteristicWrittenEvent_t

Public Members

uint16_t handle

uint16_t cValueLength

uint8_t *aValue

struct gattServerCccdWrittenEvent_t

Public Members

uint16_t handle

gattCccdFlags_t newCccd

struct gattServerAttributeReadEvent_t

Public Members

uint16_t handle

struct gattServerProcedureError_t

Public Members

gattServerProcedureType_t procedureType

bleResult_t error

struct gattServerInvalidPdu_t

Public Members

attOpcode_t attOpCode

struct gattServerEvent_t

Public Members

gattServerEventType_t eventType

union *gattServerEvent_t* eventData

union eventData

Public Members

gattServerMtuChangedEvent_t mtuChangedEvent

gattServerAttributeWrittenEvent_t attributeWrittenEvent

gattServerCccdWrittenEvent_t charCccdWrittenEvent

gattServerProcedureError_t procedureError

gattServerLongCharacteristicWrittenEvent_t longCharWrittenEvent

gattServerAttributeReadEvent_t attributeReadEvent

gattServerInvalidPdu_t attributeOpCode

gatt_database.h

struct gattDbAttribute_t

Public Members

uint16_t handle

uint16_t permissions

uint32_t uuid

uint8_t *pValue

uint16_t valueLength

uint16_t uuidType

uint16_t maxVariableValueLength

gatt_db_app_interface.h

l2ca_cb_interface.h

struct l2caLeCbConnectionRequest_tag

Public Members

deviceId_t deviceId

uint16_t lePsm

uint16_t peerMtu

uint16_t peerMps

uint16_t initialCredits

struct l2caLeCbConnectionComplete_tag

Public Members

deviceId_t deviceId

uint16_t cId

uint16_t peerMtu

uint16_t peerMps

uint16_t initialCredits

l2caLeCbConnectionRequestResult_t result

struct l2caHandoverConnectionComplete_tag

Public Members

deviceId_t deviceId

uint16_t cId

uint16_t peerMtu

uint16_t peerMps

uint16_t credits

struct l2caLeCbDisconnection_tag

Public Members

deviceId_t deviceId

uint16_t cId

struct l2caLeCbNoPeerCredits_tag

Public Members

deviceId_t deviceId

uint16_t cId

struct l2caLeCbLowPeerCredits_tag

Public Members

deviceId_t deviceId

uint16_t cId

struct l2caLeCbLocalCreditsNotification_tag

Public Members

deviceId_t deviceId

uint16_t cId

uint16_t localCredits

struct l2caLeCbError_tag

Public Members

deviceId_t deviceId

bleResult_t result

l2caErrorSource_t errorSource

struct l2caLeCbChannelStatusNotification_tag

Public Members

deviceId_t deviceId

uint16_t cId

l2caChannelStatus_t status

struct l2caEnhancedConnectionRequest_tag

Public Members

deviceId_t deviceId

uint16_t lePsm

uint16_t peerMtu

uint16_t peerMps

uint16_t initialCredits

uint8_t noOfChannels

uint16_t aCids[5]

struct l2caEnhancedConnectionComplete_tag

Public Members

deviceId_t deviceId

uint16_t peerMtu

uint16_t peerMps

uint16_t initialCredits

l2caLeCbConnectionRequestResult_t result

uint8_t noOfChannels

uint16_t aCids[5]

struct l2caEnhancedReconfigureRequest_tag

Public Members

deviceId_t deviceId

uint16_t newMtu

uint16_t newMps

l2capReconfigureResponse_t result

uint8_t noOfChannels

uint16_t aCids[5]

struct l2caEnhancedReconfigureResponse_tag

Public Members

deviceId_t deviceId

l2capReconfigureResponse_t result

struct l2capControlMessage_tag

Public Members

l2capControlMessageType_t messageType

union *l2capControlMessage_tag* messageData

union messageData

Public Members

l2caLeCbConnectionRequest_t connectionRequest

l2caLeCbConnectionComplete_t connectionComplete

l2caLeCbDisconnection_t disconnection

l2caLeCbNoPeerCredits_t noPeerCredits

l2caLeCbLowPeerCredits_t lowPeerCredits

l2caLeCbLocalCreditsNotification_t localCreditsNotification

l2caLeCbError_t error

l2caLeCbChannelStatusNotification_t channelStatusNotification

l2caEnhancedConnectionRequest_t enhancedConnRequest

l2caEnhancedConnectionComplete_t enhancedConnComplete

l2caEnhancedReconfigureRequest_t reconfigureRequest

l2caEnhancedReconfigureResponse_t reconfigureResponse

l2caHandoverConnectionComplete_t handoverConnectionComplete

l2ca_types.h

Bluetooth Low Energy Localization Application Note Introduction

NXP hardware platforms provide the features to deploy and operate Bluetooth Low Energy localization applications implementing the Channel Sounding functionality. This document provides an overview of the features and steps to run these applications on the supported NXP hardware platforms. It also describes the sample applications included in the SDK release package.

Audience

This document is a ready reference for firmware and system developers who create Bluetooth Low Energy localization enabled products. The document also provides a high-level description of Bluetooth Low Energy localization application scenarios that can be deployed on NXP development boards.

Parent topic:*Introduction*

Bluetooth Low Energy Localization Applications Overview

Localization Reader

The Localization Reader application implements the Ranging Service (RAS) client side. This application runs the localization algorithm using the local and remote measurement data received through RAS. Both the GAP central and peripheral roles, as well as the Channel Sounding initiator and reflector roles are supported and can be configured at compile time. In the default configuration, the application is a GAP central and Channel Sounding initiator.

Parent topic:*Bluetooth Low Energy Localization Applications Overview*

Localization User Device

The Localization User Device implements the RAS server side. Both the GAP central and peripheral roles, as well as the Channel Sounding initiator and reflector roles are supported and can be configured at compile time. In the default configuration, the application is a GAP peripheral and Channel Sounding reflector. The Localization User Device does not run the localization algorithms.

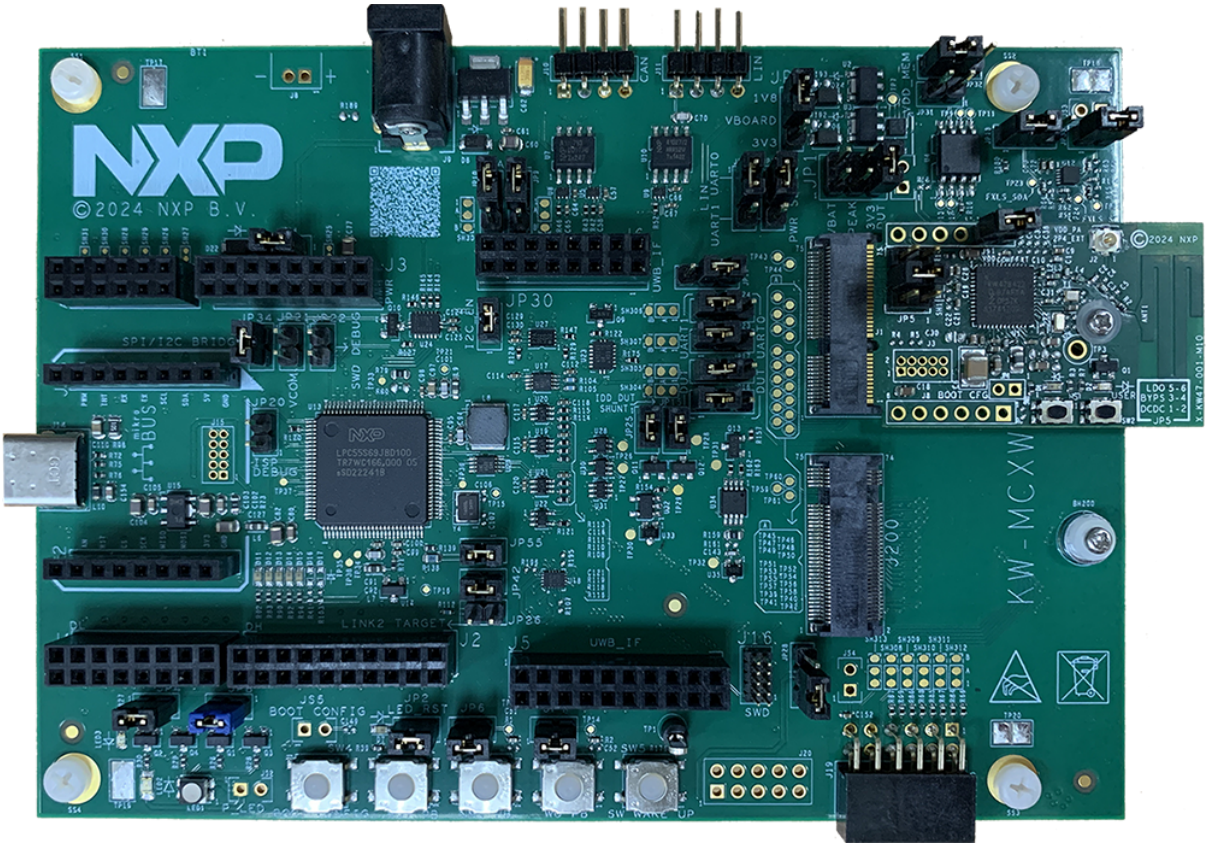
Parent topic:*Bluetooth Low Energy Localization Applications Overview*

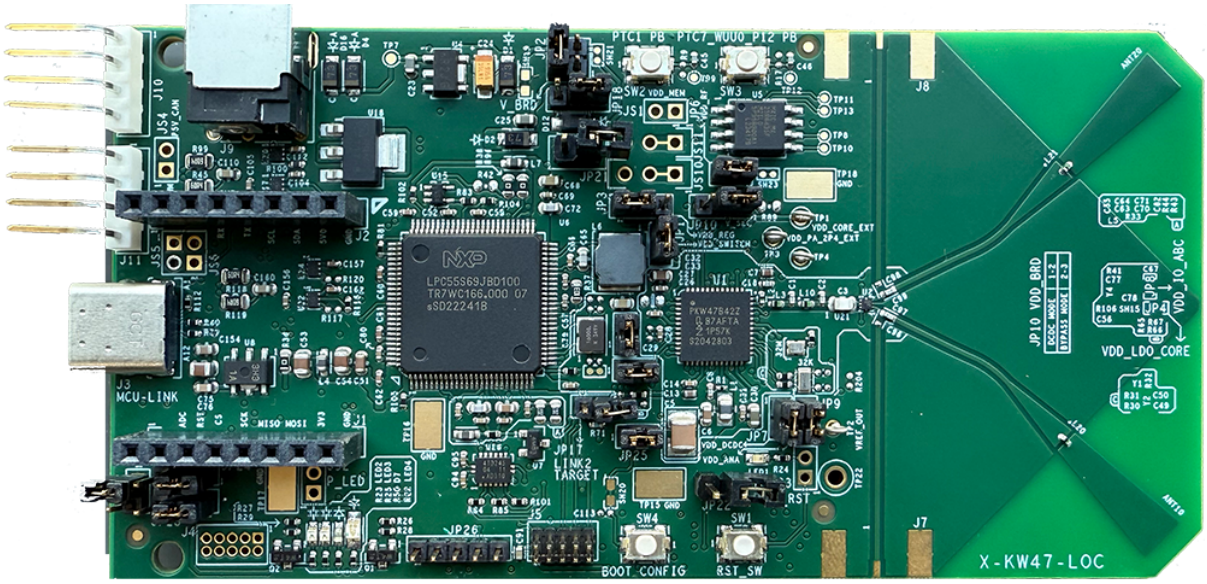
Bluetooth Low Energy Localization Hardware Platforms

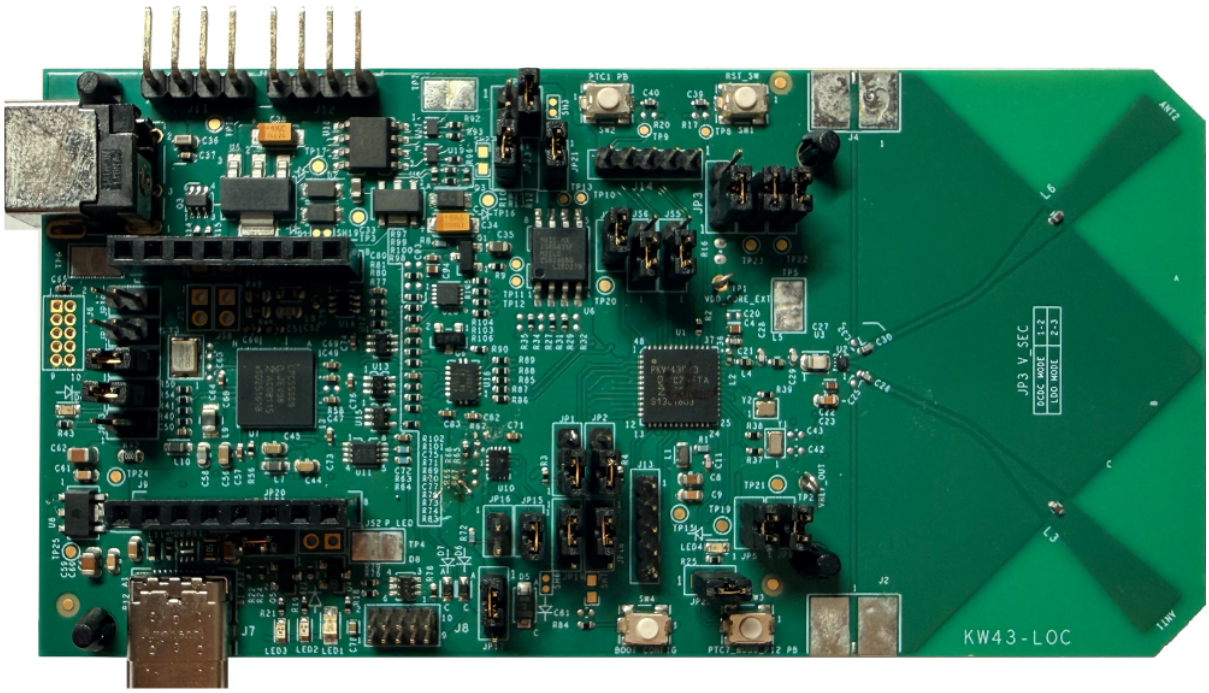
The Bluetooth Low Energy Localization demo applications support the following platforms that have Bluetooth Low Energy transceiver capabilities:

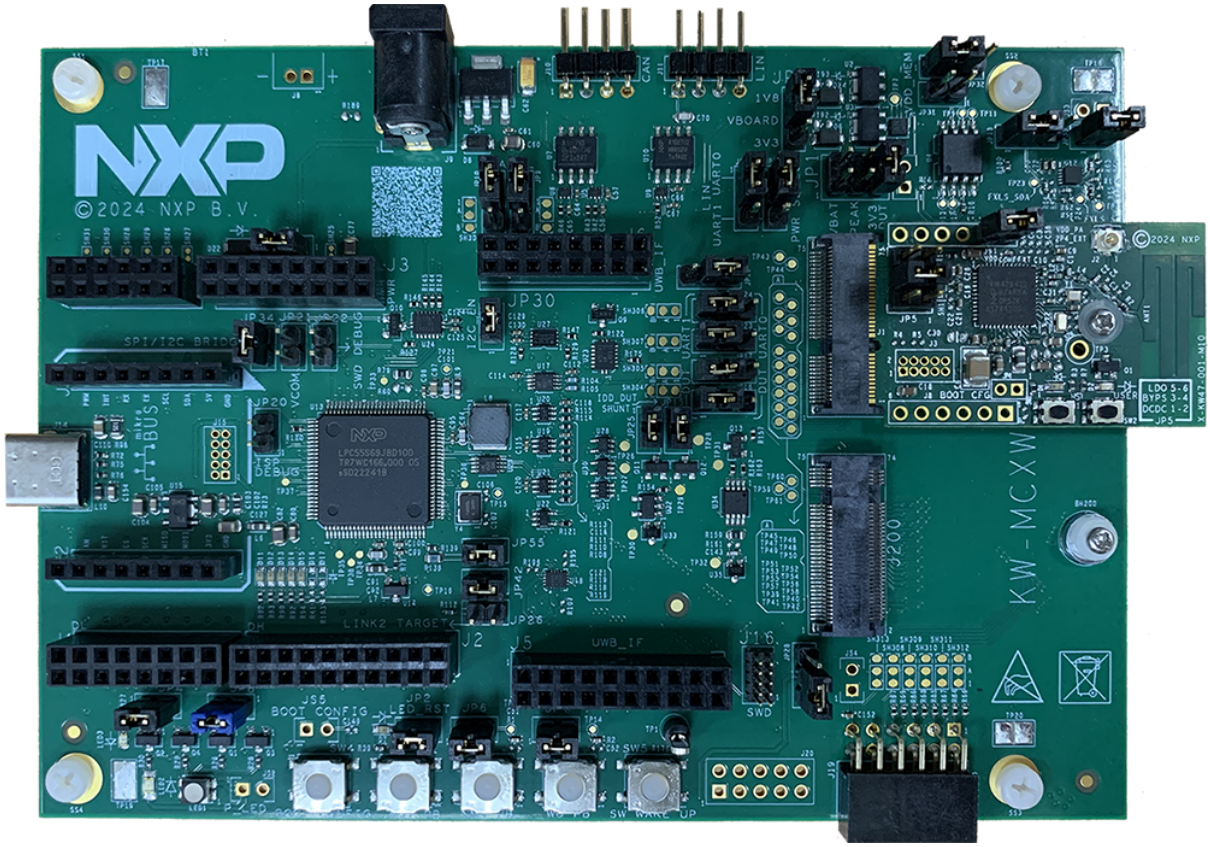
- KW47-EVK (see Figure 1)
- KW47-LOC (see Figure 2)
- KW43-LOC (see Figure 3)
- MCXW72-EVK (see Figure 4)
- FRDM-MCXW72 (see Figure 5)
- MCXW72-LOC (see Figure 6)
- FRDM-KW43 (see Figure 7)

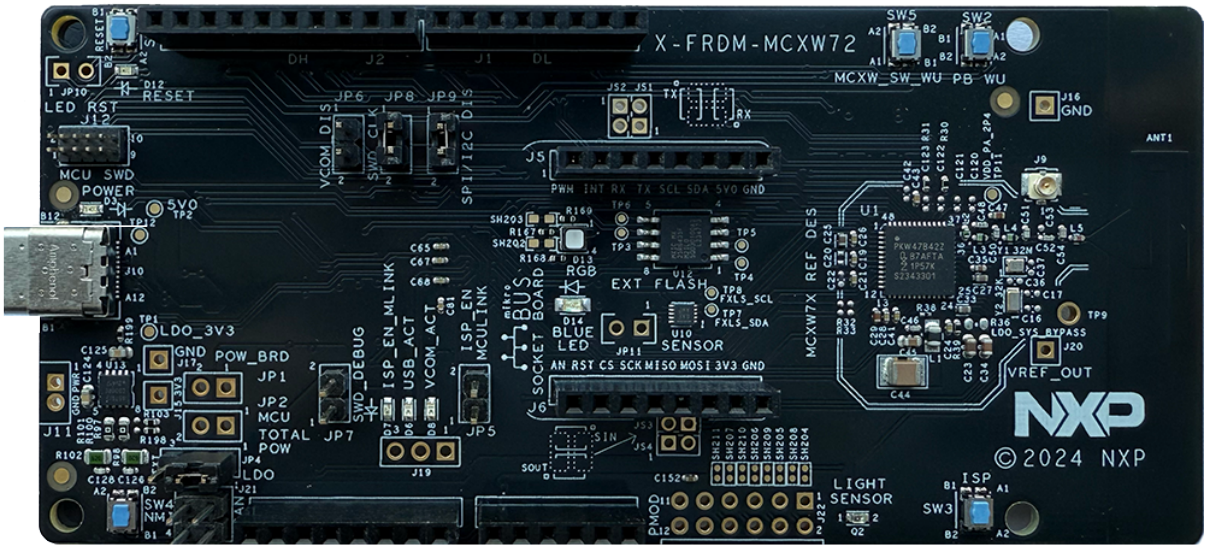
Note: The KW47/MCXW72/KW43 device family includes the LCE (Localization Compute Engine). This is a hardware accelerator for the localization algorithm.

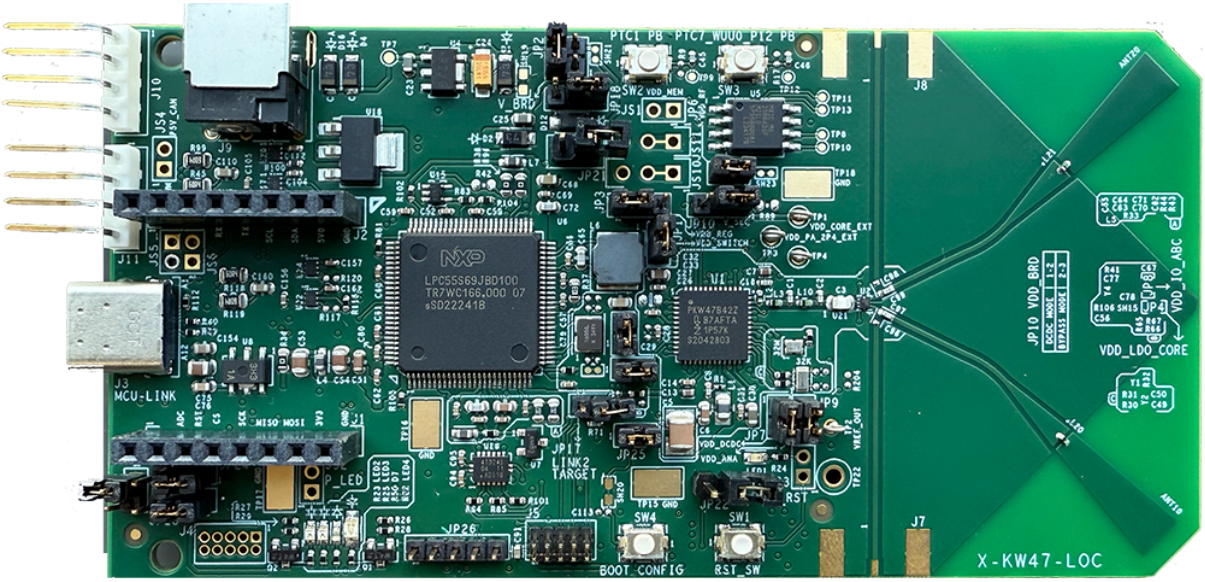

NXP
©2024 NXP B.V.
KW-MCXW

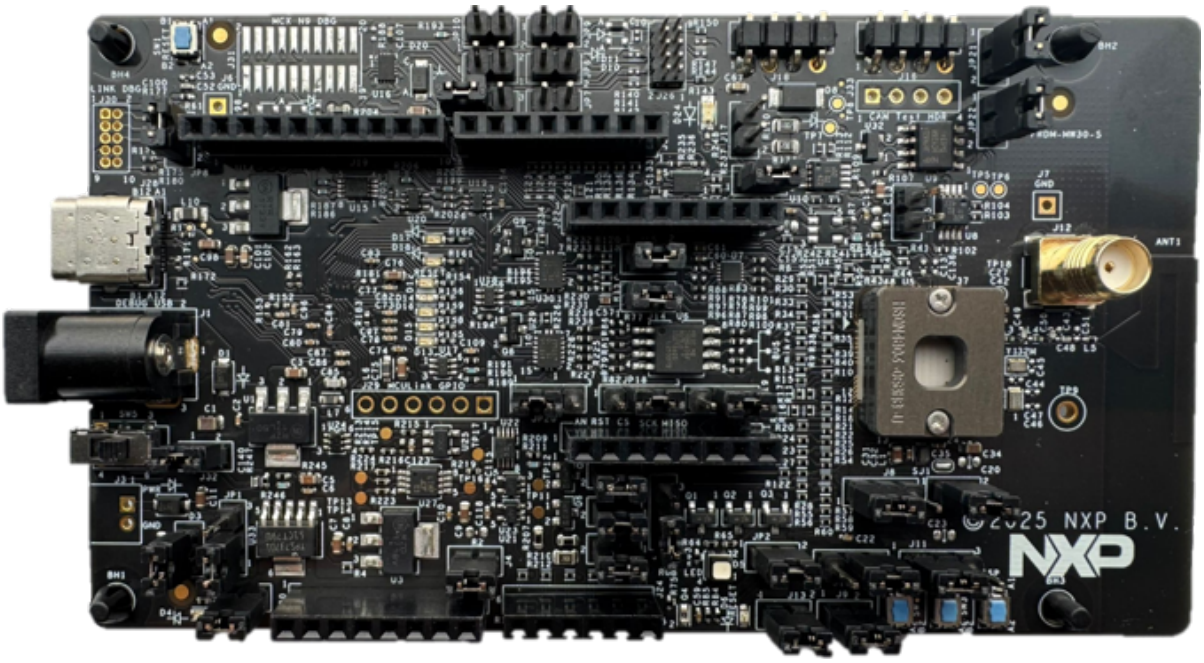

MCU-LINK
LINK2 TARGET
BOOT_CONFIG
RST_SW
VDD_LDO_CORE
X-KW47-LOC


KW43-LOC


NXP
©2024 NXP B.V.
KW-MCXW


X-FRDM-MCXW72
©2024 NXP


X-KW47-LOC


©2025 NXP B.V.

Deploying Bluetooth Low Energy Localization Applications Software

The software package includes the components necessary for Bluetooth Low Energy localization application development on the selected NXP platform. These components include:

- Bluetooth Low Energy Host software libraries
- NBU image binaries
- Channel Sounding and Localization Algorithm libraries
- Examples of sample application project corresponding to Channel Sounding roles, Initiator (Localization User Device), and Reflector (Localization Reader) roles
- Platform peripheral drivers, platform startup code, other generic platform software

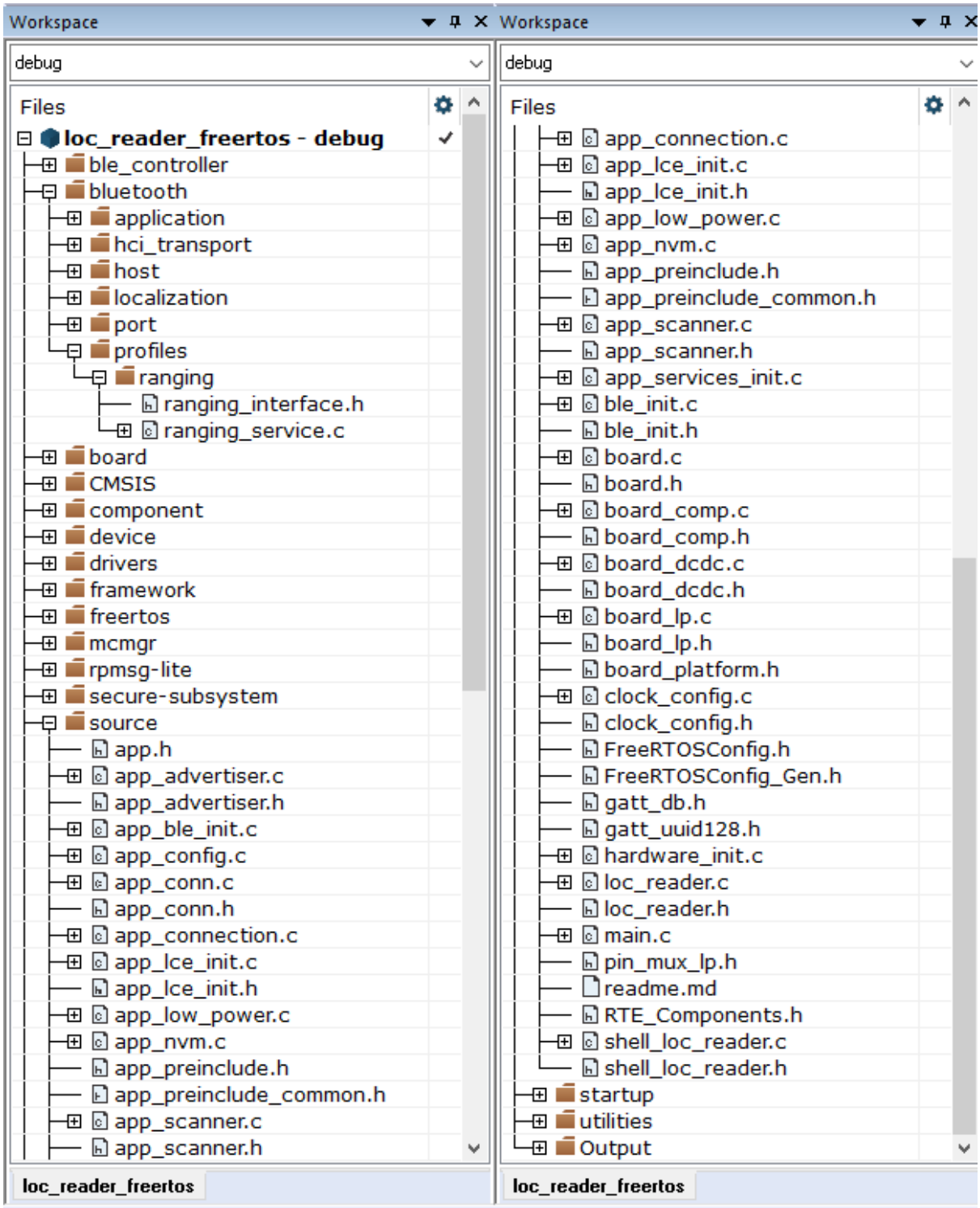
Note: Prior to loading any wireless SDK example, update your NBU image with the provided binaries, with channel sounding support, in the following folder of the SDK: ../middleware/wireless/ble-controller/bin.

The demo applications were compiled and tested with **IAR Embedded Workbench** for Arm and **ARMGCC**. Use one of these tools to deploy your sample application.

To open, build, and run any sample application, see the *Bluetooth Low Energy Quick Start Guide* document of the corresponding board. The Bluetooth Low Energy Localization examples are built and run in the same manner as the Bluetooth Low Energy examples.

Figure 1 shows the application folder structure for the loc_reader project for the zip package. The relevant files for Bluetooth Low Energy localization are the following:

- The main application source files are source/loc_reader.c and source/shell_loc_reader.c.
- The files in bluetooth/profiles/ranging implement the Ranging Service.
- The files in source folder contain the Bluetooth Low Energy localization application common code and localization algorithm implementation.



Demo Functionality Overview

The two demo applications provided showcase the following Bluetooth Low Energy localization features:

- Distance measurement through Channel Sounding
- Ranging Service (RAS) and Ranging Profile (RAP) implementation
- Distance measurement data transfer through RAS
- Complex-domain Distance Estimation (CDE) ranging algorithm

- Rapid Accurate Distance Estimator (RADE) ranging algorithm

Application configuration options:

- **Channel Sounding role:** Set the Channel Sounding role by using the gCsDefaultRole_c define in app_preinclude.h. In the default configuration, the **loc_reader** application is in the initiator role (gCsRoleInitiator_c) and the **loc_user_device** application is in the reflector role (gCsRoleReflector_c).
- **GATT CCCD type for RAS characteristics:** Configure whether RAS characteristics use INDICATION or NOTIFICATION for data transfer by using the gRasUseNotifOrInd_c define in app_preinclude.h. Set to gCccdIndication_c for reliable delivery with acknowledgment (slower, guaranteed) or gCccdNotification_c for best-effort delivery without acknowledgment (faster, no guarantee). This affects RAS Control Point, Data Ready, Data Overwritten, and On-Demand Data characteristics. Default: gCccdNotification_c.
- **Channel Sounding procedure repeat**: Use the Channel Sounding procedure repeat to specify the number of channel sounding procedures to be performed for each distance measurement triggered. This macro can be set through the gCsProcRepeatMaxNumProcedures_c define.
- **Ranging algorithm**: The ranging algorithm must be run once the local and remote measurement data is available. Both supported algorithms can be disabled at compile time for code size reasons. By default, both are enabled. To disable an algorithm at compile time, define its respective macro with value 0 in the file app_preinclude.h. Use the macros gAppUseCDEAlgorithm_d and gAppUseRADEAlgorithm_d. At runtime, the setalgo shell command can be used to configure the algorithm that runs on completion of the CS procedure and RAS transfer. Any combination of algorithms can be selected.
- **Timing information display**: To enable this feature, set the gAppCsTimeInfo_d macro to the value 1 in the app_preinclude.h file. The console then displays details of the timing information (such as the duration of the CS configuration phase, CS procedure, RAS transfer, and running of the algorithm(s)).
- **Data quality information display**: This feature can be enabled by setting the gAppParseQualityInfo_d macro to the value 1 in the file app_preinclude.h for the Anchor. Additional information (such as RSSI and Tone Quality Indicator values for the measurement data) is displayed in the console.
- **GAP role**: Set the GAP role using gAppIsPeripheral_d, 1 for peripheral role, 0 for central role. In the default configuration, the loc_reader application is central.
- **Maximum concurrent Channel Sounding procedures**: Configure the maximum number of Channel Sounding procedures that can run simultaneously across all connections using the gChannelSoundingMaxConcurrentProcedures_c define in app_preinclude.h.
- **Adaptive Channel Sounding procedure interval (experimental)**: This experimental feature is disabled by default. When enabled (and the procedure auto-restart loop is active), the CS procedure interval is dynamically adjusted between loop iterations based on the remote RSSI average: a good RSSI decreases the interval (faster distance report rate), a bad RSSI increases the interval (more time for the RAS transfer to complete on a weak link), and an RSSI within the hysteresis band leaves the interval unchanged. CS procedures that fail to complete during an iteration are treated as a very bad link. To enable it, set the gAppAdaptiveProcInterval_d macro to the value 1 in the app_preinclude.h file; this requires gAppParseRssiInfo_d to also be enabled (enforced at compile time). The RSSI thresholds and interval limits (gAdaptiveRssiGoodThreshold_d, gAdaptiveRssiBadThreshold_d, gAdaptiveRssiNotCompleted_d, gAdaptiveProcIntervalStep_d, gAdaptiveProcIntervalMin_d, and gAdaptiveProcIntervalMax_d) can be tuned in application/common/lcl/app_localization_config.h.

Running the Bluetooth Low Energy Localization Scenario

To run the Bluetooth Low Energy Localization scenario, flash the **loc_reader** and **loc_user_device** applications on two KW45B41Z-EVK development boards.

Note: The boards listed in *Bluetooth Low Energy Localization Hardware Platforms* can also be used to run the steps listed in this section.

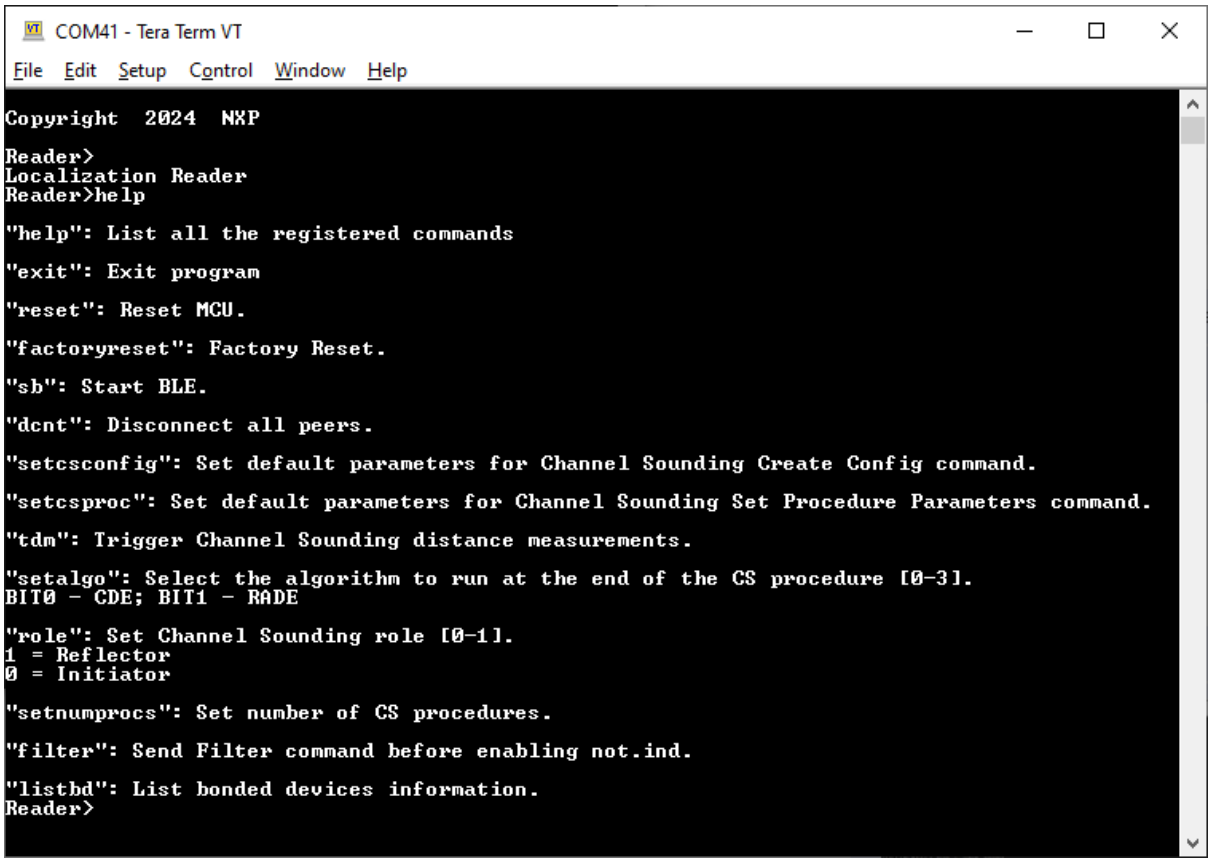
Running a Bluetooth Low Energy Localization Scenario Using the Shell Interface

The following Bluetooth Low Energy localization examples are provisioned by default with a shell command-line interface accessible via a Terminal application such as Tera Term:

- Localization Reader
- Localization User Device

After connecting with the Terminal application, pressing the **RST** switch on the board displays the application name.

To display the list of available commands and their description, enter the 'help' command as shown in Figure 1.



To run the application, perform the following steps:

- Enter the sb command on both terminals that are connected to the Localization Reader and Localization User Device applications. The devices connect, perform pairing if not previously bonded, and trigger distance measurement.

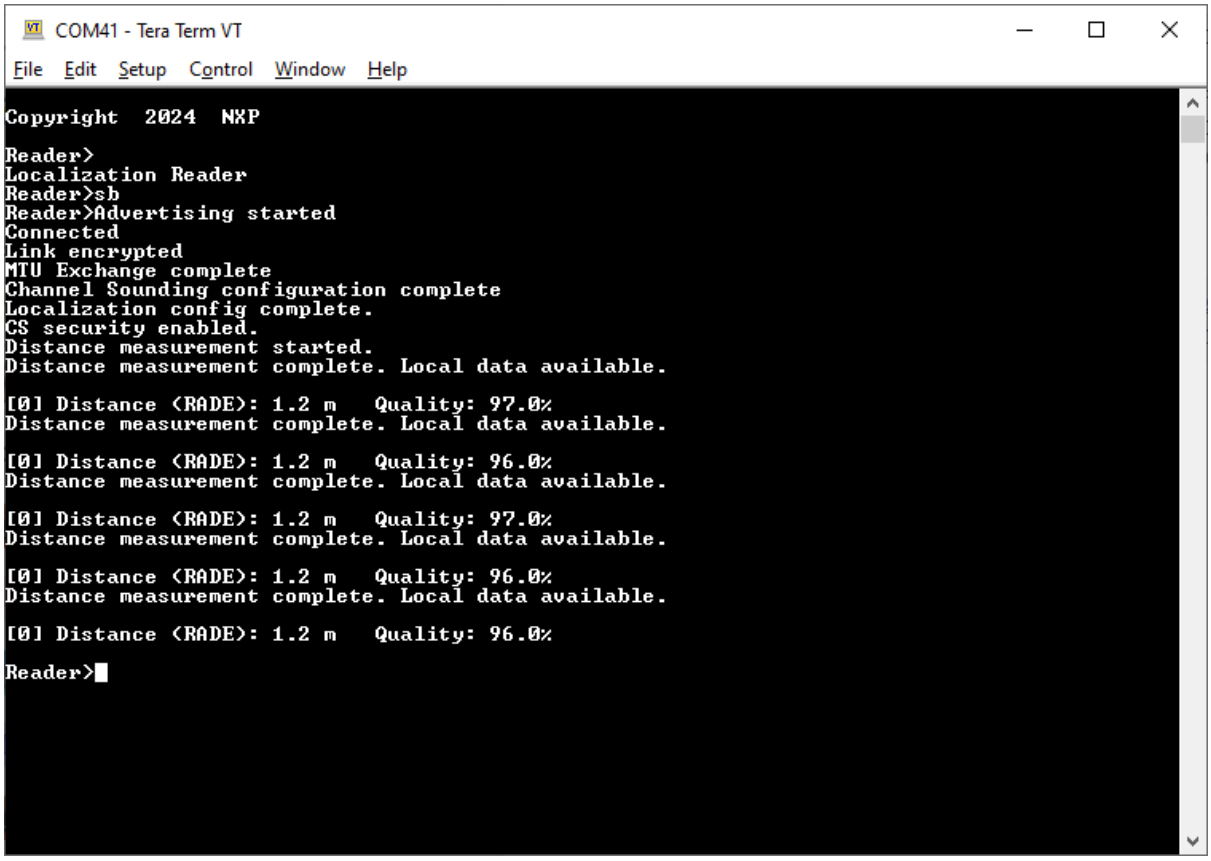
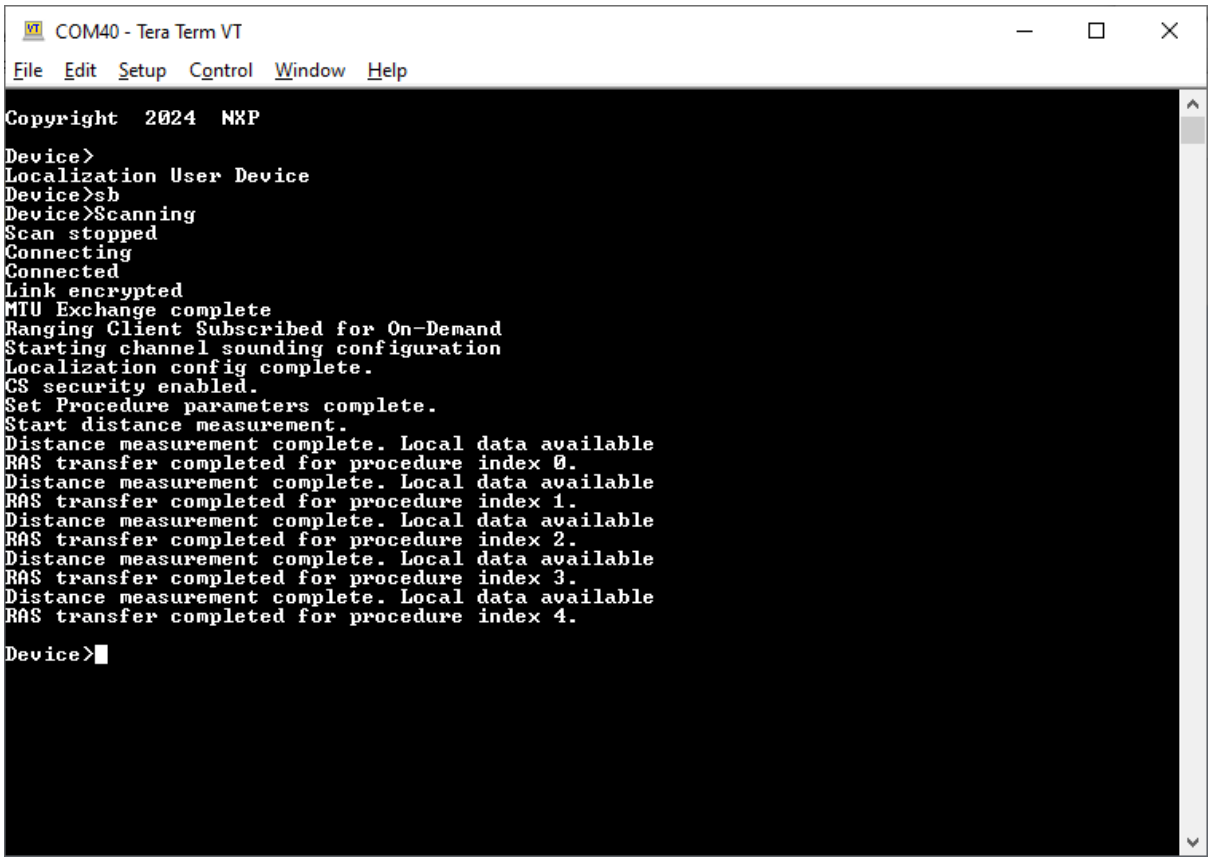


Figure 2 illustrates the start of the Localization Reader application.



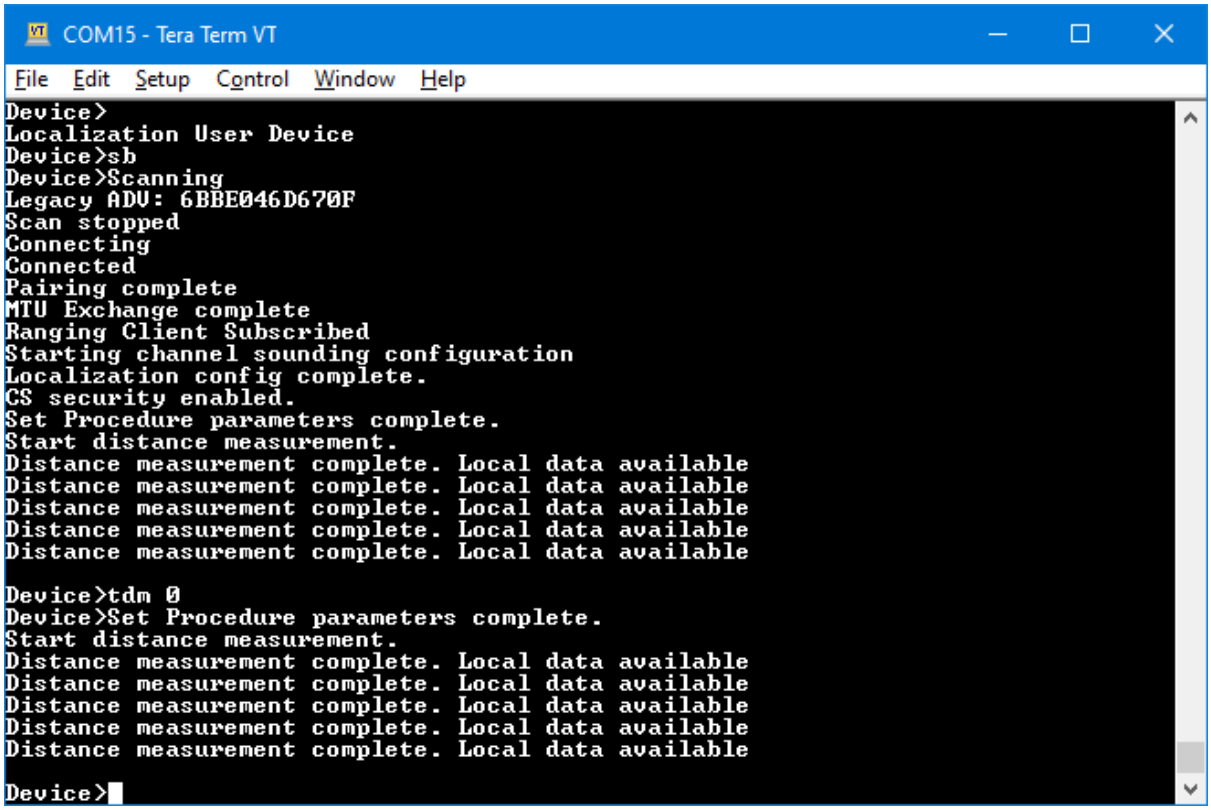
After the devices are connected, the link is encrypted, the Channel Sounding configuration is created, and distance measurement is triggered. This is shown in Figure 3. If two consecutive

distance measurements return a value lower or equal to 1 meter, the LED1 on the Localization Reader starts blinking. If the measured distance is higher than 1, LED1 turns off.

- To trigger a new distance measurement, use the "tdm" command:
  - tdm (without parameters): Triggers distance measurement on **all connected devices**
  - tdm <peerId>: Triggers distance measurement on the **specified peer device**

  Examples:
  - tdm - Start distance measurement on all connected peers
  - tdm 0 - Start distance measurement on peer device with ID 0
  - tdm 1 - Start distance measurement on peer device with ID 1

  This command can be executed on either the Localization User Device or the Localization Reader. Figure 4 shows an example of triggering the distance measurement from the Localization User Device.



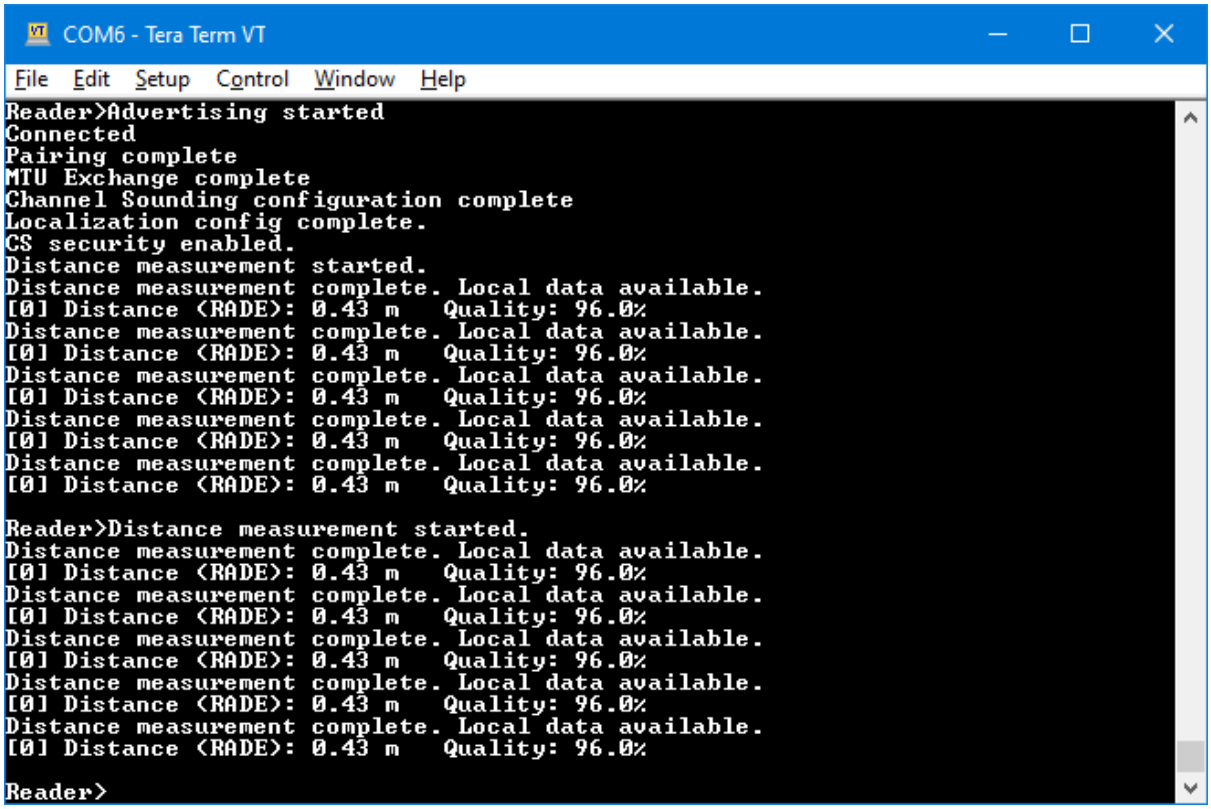


Figure 5 shows the available distance measurement data.

Other available shell commands are:

- setcsconfig: This command overwrites the default parameters for the *Channel Sounding Create Config* command. It updates only the parameters that are explicitly specified; any parameter that is not provided retains its current value. The syntax is:

  setcsconfig <peerId> <paramName> <value> [<paramName> <value> ...]

  where <peerId> is the device ID of the peer, followed by one or more <paramName> <value> pairs. The supported parameter names and their accepted values are:

  - mainmodetype: CS mode to be used as main mode during the CS procedure for this configuration. Possible values:
    - 1 (CS mode-1)
    - 2 (CS mode-2)
    - 3 (CS mode-3)
  - submodetype: CS mode to be used as submode during the CS procedure for this configuration. Possible values:
    - 1 (CS mode-1)
    - 2 (CS mode-2)
    - 3 (CS mode-3)
    - 255 (no submode used)
  - mainmodeminsteps: Lower bound of the range of main mode CS steps to be executed before a submode CS step is executed during the CS procedure. Range:
    - 1 to 255
  - mainmodemaxsteps: Higher bound of the range of main mode CS steps to be executed before a submode CS step is executed during the CS procedure. Range:

    * 1 to 255
  - mainmoderepetition: Number of main mode CS steps repeated from the previous CS subevent at the beginning of the current CS subevent. Range:
    * 0 to 3
  - mode0steps: Number of CS mode-0 steps to be included at the beginning of each CS subevent. Range:\
    * 1 to 3
  - role: CS role. Possible values:
    * 0 (initiator)
    * 1 (reflector)
  - rtttype: RTT variant to be used during the CS procedure. Possible values:
    * 0 (RTT AA only)
    * 1 (RTT with 32-bit sounding sequence)
    * 2 (RTT with 96-bit sounding sequence)
    * 3 (RTT with 32-bit random sequence)
    * 4 (RTT with 64-bit random sequence)
    * 5 (RTT with 96-bit random sequence)
    * 6 (RTT with 128-bit random sequence)
  - channelmap: 80-bit map in big endian order indicating channels to be used for the CS procedure. The default value is fcff7ffcffffffffff1f (all channels used except 0, 1, 23-25 and 77-78). Bit 79 is reserved for future use. At least 15 channels must be enabled.
  - channelmaprepetition: Number of times the channel map will be cyled through for non-mode-0 steps within a CS procedure. Range:
    * 1 to 255.
  - channelselectiontype: Channel selection type. Possible values:
    * 0 (Channel Selection Algorithm #3b)
    * 1 (Channel Selection Algorithm #3c)
  - cssyncphy: PHY to be used for CS_SYNC exchanges during the CS procedure for the specified CS configuration. Possible values:
    * 1 (LE 1M PHY)
    * 2 (LE 2M PHY)
    * 3 (LE 2M 2BT PHY) - if supported at Link Layer level

  Example: setcsconfig 0 mode0steps 3 cssyncphy 2 updates only the mode-0 steps and the CS_SYNC PHY, leaving all other parameters unchanged.

- setcsproc: This command overwrites the default parameters for the *Channel Sounding Set Procedure Parameters* command. It updates only the parameters that are explicitly specified; any parameter that is not provided retains its current value. The syntax is:

  setcsproc <peerId> <paramName> <value> [<paramName> <value> ...]

  where <peerId> is the device ID of the peer, followed by one or more <paramName> <value> pairs. The supported parameter names and their accepted values are:

  - maxprocedureduration: Maximum duration for each CS procedure. Range:
    * 1 to 65535 (units of 0.625ms)

  - minperiodbetweenprocedures: Minimum number of connection events between consecutive CS procedures. Range:
    * 1 to 65535
  - maxperiodbetweenprocedures: Maximum number of connection events between consecutive CS procedures. Range:
    * 1 to 65535
  - maxnumprocedures: Maximum number of CS procedures to be scheduled. Possible values:
    * 0 (CS procedures to continue until disabled)
    * 1 to 65535
  - minsubeventlen: Minimum suggested duration for each CS subevent in microseconds. Range:
    * 1250 microseconds to 3.999999 seconds
  - maxsubeventlen: Maximum suggested duration for each CS subevent in microseconds. Range:
    * 1250 microseconds to 3.999999 seconds
  - antcfgindex: Antenna Configuration Index as described in the Core specification. Range:
    * 0 to 7
  - snrcontrolinit: SNR Control to be applied to the Initiator in the CS procedure (if supported at Link Layer level). Possible values:
    * 0 (SNR control adjustment of 18 dB)
    * 1 (SNR control adjustment of 21 dB)
    * 2 (SNR control adjustment of 24 dB)
    * 3 (SNR control adjustment of 27 dB)
    * 4 (SNR control adjustment of 30 dB)
    * 255 (SNR control is not to be applied)
  - snrcontrolrefl: SNR Control to be applied to the Reflector in the CS procedure (if supported at Link Layer level). Possible values:
    * 0 (SNR control adjustment of 18 dB)
    * 1 (SNR control adjustment of 21 dB)
    * 2 (SNR control adjustment of 24 dB)
    * 3 (SNR control adjustment of 27 dB)
    * 4 (SNR control adjustment of 30 dB)
    * 255 (SNR control is not to be applied)

  Example: setcsproc 0 minsubeventlen 2500 antcfgindex 1 updates only the minimum subevent length and the antenna configuration index, leaving all other parameters unchanged.

- verbosity: Sets the verbosity level during the CS procedure.
- setnumprocs: This command overwrites the default value gCsProcRepeatMaxNumProcedures_c parameter for the CS Procedure Repeat. It takes the following parameters, in the order below:
  - peerId
  - maxnumprocedures: Number of procedures in hex format. Example: setnumprocs 0 0x0005. Value 0x0000 means infinite repeats.

- setmcp: This command overwrites the default value gChannelSoundingMaxConcurrentProcedures_c parameter for the number of maximum supported simultaneous CS procedures across all connections. It takes the following parameters:
  - maxNumConcurrentProcedures: Number of procedures in decimal format. Example: setmcp 2.
- loop: This command toggles the automatic restart of the CS procedure (whether with CS Procedure Repeat enabled or not) upon its end. By default, the CS procedure will not restart automatically.

Parent topic:*Running the Bluetooth Low Energy Localization Scenario*

Running the Bluetooth Low Energy Localization Scenario Using the Button Interface

To run the Bluetooth Low Energy Localization scenario using the button interface perform the following steps:

- Press **SW2** on both boards to connect the devices. After the devices are connected, the link is encrypted and the Channel Sounding configuration is created, distance measurement is triggered. When two consecutive distance measurements return a value lower or equal to 1, the LED1 on the Localization Reader starts blinking. When the measured distance is higher than 1, LED1 turns off.
- Press **SW3** on the *loc_user_device* board to trigger distance measurement. LED1 blinks or turns off, based on the results of the distance measurement.

Parent topic:*Running the Bluetooth Low Energy Localization Scenario*

Selecting the RADE Configuration

The localization examples support four RADE configurations: RADE1, RADE2, RADEX1, and RADEX2. Each configuration is provided as a dedicated Kconfig fragment and is selected at build time using the -DCONF_FILE= west build argument.

Configuration	CONF_FILE value
RADE1	examples/wireless_examples/bluetooth/config/channel_sounding_rade1.conf
RADE2 (default)	examples/wireless_examples/bluetooth/config/channel_sounding_rade2.conf
RADEX1	examples/wireless_examples/bluetooth/config/channel_sounding_radex1.conf
RADEX2	examples/wireless_examples/bluetooth/config/channel_sounding_radex2.conf

To build with a specific configuration, pass the corresponding path to -DCONF_FILE=. For example, to use RADE1:

```
west build -b <board> -- -DCONF_FILE=examples/wireless_examples/bluetooth/config/channel_sounding_
↪rade1.conf
```

Replace channel_sounding_rade1.conf with the desired configuration file name from the table above.

Notes:

- RADE1 and RADEX1 configurations do not use LCE acceleration. Running the RADE algorithm without LCE support increases stack size requirements. Ensure the stack size for the task on which the algorithm runs is at least 4000 octets.
- Make sure that conflicting Kconfig options are not duplicated across configuration files. If duplicates exist, remove them before building the project.

Transient Application Key (TAK)

Overview

The Transient Application Key (TAK) feature provides enhanced security for Bluetooth LE connections between the Localization Reader and Device applications. TAK is a one-time use symmetric key that must be pre-shared between devices before establishing a connection. Once used, the key is automatically erased per specification requirements. This guide provides out-of-the-box commands ready for copy-paste into TeraTerm.

Prerequisites

Ensure both applications are compiled with TAK enabled (gAppUseTAK_d = 1) and gAppTAKAdvID_c matches on both sides. The default configuration supports up to 4 TAK entries (gConnTakMaxEntries_c = 4). In order for Peripheral to allow passing LTK request (coming from an unbonded device) to the application, gBleHostAutoRejectLtkRequestForUnbondedDevices_c must be set to FALSE in the loc_user_device application. Because TAK is not considered sufficient method for an attribute which requires encryption, all encryption-related flags from the database must be removed. Check gatt_db.h, on the loc_user_device and remove the flags gPermissionFlagWriteWithEncryption_c and gPermissionFlagReadWithEncryption_c.

Quick Setup

Setup on Localization Reader

tak 0 00112233445566778899aabbccddeeff sb

Setup on Localization Device

tak 0 00112233445566778899aabbccddeeff sb

For Subsequent Connections

Since TAK is erased after use, set a new key before the next connection: tak 0 11223344556677889 9aabbccddeeff00

Typical Console Output

```
Reader>
Copyright  2024  NXP

Reader>
Localization Reader
Reader>tak 0 00112233445566778899aabbccddeeff
Reader>sb
Reader>Scanning
Legacy ADV: 4EA49933B8B4 (TAK)
Scan stopped
Connecting
Connected
Link encrypted
MTU Exchange complete
Channel Sounding configuration complete
Localization config complete.
CS security enabled.
Set Procedure parameters complete.
Start distance measurement.

Distance measurement complete. Local data available.

[0] Distance (RADE): 0.22 m   Quality: 90.0%

[0] RTT Distance: 0.0 m Success Rate: 100
```

```
Device>
Copyright  2024  NXP

Device>
Localization User Device
Device>tak 0 00112233445566778899aabbccddeeff
Device>sb
Device>Advertising started
Connected
Link encrypted
MTU Exchange complete
Ranging Client Subscribed for On-Demand
Starting channel sounding configuration
Localization config complete.
CS security enabled.
Distance measurement started.
Distance measurement complete. Local data available
RAS transfer completed for procedure index 0.
```

Testing RAS and RAP using Profile Tuning Suite (PTS)

When testing RAS and RAP using the PTS, the loc_reader and loc_user_device need to be used.

The test code support for testing RAS and RAP against PTS is located in the files:

- application/common/lcl/app_localization.c
- application/common/lcl/pts_test_vectors.c
- examples/loc_reader/loc_reader.c
- examples/loc_user_device/loc_user_device.c

This configuration must be enabled in the app_preinclude.h of either of the examples:

- gRasRapPtsTest_d is set to 1
- gAppUsePrivacy_d is set to 0
- gRandomStaticAddress_d is set to 0
- gAppMaxConnections_c is set to 2 (any value higher than 1)

The testing method for RAS and RAP in PTS is based on a test vector approach, as PTS does not support Channel Sounding (CS).

Usage

The PTS tool implements tests using test steps. These steps could vary from connects and disconnects, to generation and injection of CS data in the Implementation Under Test (IUT).

The test steps require user interaction with the IUT, as PTS will guide the user with actions necessary for the user to execute to complete the test.

Primarily, the user will use the:

- sb command to start Bluetooth LE advertising for the PTS to initiate a connection to the IUT.
- dcnt command to disconnect the IUT from the PTS.
- ptstest command, followed by the test_id that is the id of the test running, usage example: ptstest RAS/SR/RCO/BV-01-C.
- filter command set a filter. (used in reduced number of test)

A test could require no use of the ptstest, one use of the ptstest command or multiple uses of the ptstest command. When a test requires a user to perform several actions, ptstest test_id should be issued with the same test ID as the test being run. The test step logic can be found in the loc_reader.c and loc_user_device.c, under the BleApp_RunPtsTest function.

References

For more information, contact your local Field Application Engineer (FAE) or NXP representative.

Acronyms and abbreviations

The table below lists the acronyms used in this document.

Outer pipes Cell padding No sorting

Table 1. Acronyms and abbreviations

Bluetooth LE	Bluetooth Low Energy
CS	Channel Sounding
IDE	Integrated Development Environment
LCE	Localization Compute Engine
RAS	Ranging Service
RAP	Ranging Profile
CDE	Complex-domain Distance Estimation
RADE	Rapid Accurate Distance Estimator
SDK	Software Development Kit

Revision history

The table below summarizes the revisions to this document.

Document ID	Release date	Description
AN139 v1.5	06 April 2026	• Added support for FRDM-KW43 platform
AN139 v1.4	22 July 2025	• Remove support for KW45B41Z-EVK, KW45B41Z-LOC, FRDM-MCXW71, MCXW71-EVK
AN139 v1.3	26 November 2024	• Aligned to KW47 EAR 2.1 release• Added suppport for KW47-EVK, KW47-LOC, MCXW72-EVK, and FRDM- MCXW72 boards. See Section 3 "Bluetooth Low Energy localization hardware platforms"
AN139 v1.2	26 June 2024	• Aligned to SDK 2.16.000 release• Added FRDM-MCXW71 platform suppport. See Figure 4
AN139 v1.1	25 January 2024	Updated to add KW45B41Z-EVK and KW45B41Z-LOC hardware platform support
AN139 v.1.0	23 June 2023	Initial release

Legal information

Bluetooth Low Energy Channel Sounding Application Developer's Guide

Introduction

Channel Sounding (CS) is a Bluetooth Low Energy version 6.0 feature which, together with the Ranging Service (RAS)/Ranging Profile (RAP), enables two devices to securely estimate the distance between them.

The devices have the following roles:

- within GAP, one device is a Peripheral, the other is a Central.
- within CS, one device is the CS Initiator, the other is the CS Reflector.
- within RAS, one device is the Ranging Requester, the other is the Ranging Responder.

The role configurations above are described in detail in the following sections.

This document explains how to integrate CS and RAS/RAP functionality into an application and provides detailed explanation of the most commonly used APIs and code examples.

Prerequisites

Channel Sounding functionality can be integrated into an application that fulfills the following requirements:

- Supported platforms: KW47-EVK, KW47-LOC, KW43-LOC, FRDM-MCXW72, MCX-W72-EVK, MCX-W72-LOC and FRDM-KW43.
- Bluetooth Low Energy application that uses the lib_ble_OPT_host_cm33_iar/gcc.a Host library and the lib_channel_sounding_iar/gcc.a library.

Integration

The Channel Sounding functionality at Host level is implemented in the following libraries:

- lib_channel_sounding_iar.a (for IAR projects)
- lib_channel_sounding_gcc.a (for MCUX or armgcc projects)

All the Channel Sounding APIs can be found in middleware\wireless\bluetooth\host\interface\channel_sounding.h.

Application common modules

While the CS APIs are available to applications and can be used directly, common application modules are also provided. These modules, detailed below, offer a simplified API for performing distance measurements and allow for quick integration into existing applications.

app_localization common module

Introduction

The app_localization common module allows for quick integration of Channel Sounding functionality in a Bluetooth LE application.

Integration

All the APIs referenced in this section are available in the app_localization application common module.

Perform the following steps for integration into the application. Unless specified, each step applies to both the CS Initiator and the CS Reflector:

1. Add the following components to the application project via Kconfig:
   - CONFIG_MCUX_COMPONENT_middleware.wireless.lcl=y
   - CONFIG_MCUX_COMPONENT_middleware.wireless.ble_host_CS=y
   - CONFIG_MCUX_COMPONENT_middleware.wireless.ble_host_CS_component_lib=y
2. Begin the initialization of the common module by calling AppLocalization_Init(). One of the parameters of this API is the application callback where events will be received from the app_localization module.
3. Complete the initialization of the common module by calling AppLocalization_HostInitHandler() after the Host stack has been initialized.
4. The CS Config depends on the Bluetooth LE connection interval. Upon connection with the peer, the CS Config is retrieved by calling AppLocalization_ReadConfig() and updated by calling AppLocalization_WriteConfig().
5. If the application is the Ranging Requester, that is, it will run the distance measurement algorithm, the connection interval must also be saved by calling the function AppLocalization_SetConnectionInterval(). This is required by the algorithm.
6. Once the Ranging Requester has discovered and subscribed to the Ranging Service on the Ranging Responder (as detailed in the section below), the CS procedure can begin. The first step is to call AppLocalization_Config().
7. Upon receiving the gConfigComplete_c or the gLocalConfigWritten_c events, only the device who has the GAP Central role must call AppLocalization_SecurityEnable().
8. Upon receiving the gCsSecurityEnabled_c event, only one device must call AppLocalization_SetProcedureParameters().
9. Upon receiving the gSetProcParamsComplete_c event, the device from step 8 must call AppLocalization_StartMeasurement().
10. From this point onwards, the application will receive events informing it of the status of the distance measurement procedure, such as:

- gDistanceMeastStarted_c - the CS Procedure has started
- gLocalMeasurementComplete_c - the CS Procedure is complete and local data is available
- various error events

11. Upon peer disconnection, cleanup should be performed by calling AppLocalization_ResetPeer().

Parent topic:*Application common modules*

app_localization_algo common module

Introduction

The app_localization_algo module is responsible for unpacking the data resulted from the CS Procedure and the RAS transfer. The module then passes it to the distance measurement algorithm to obtain the distance in meters, which is then communicated to the application.‘

Integration

The app_localization_algo is only required on the application that has the role of Ranging Requester and computes the actual distance. Perform the following steps to integrate app_localization_algo common module:

1. Add the following component to the application project via Kconfig:
   - CONFIG_MCUX_COMPONENT_middleware.wireless.lcl_algo=y
2. Pass a callback of type pfAppDisplayResult_t as the third parameter of AppLocalization_Init. This callback receives the distance measurement result, which the application displays.
3. Call the function AppLocalizationAlgo_ResetPeer upon peer disconnection to clean up algorithm-related data structures.
4. There is no other direct interaction between the application and this module. Its APIs are mostly used by the app_localization module.

Ranging Service common module

Introduction

The Ranging Service is a GATT service used to transfer the data resulted from the CS Procedure from one device (the Ranging Responder) to the other (the Ranging Requester).

Integration

Perform the following steps to integrate the Ranging Service on the application that has the role of Ranging Requester (GATT client):

1. Add the following component to the application project via Kconfig:
   - CONFIG_MCUX_COMPONENT_middleware.wireless.ble_profiles_ranging_client=y
2. Ensure the gRasRREQ_d define is present and set to 1 in the application's app_preinclude.h file.
3. After the connection with the peer is encrypted, proceed to discover the RAS and its characteristics through the common Service Discovery module. Register a service discovery callback using BleServDisc_RegisterCallback and call BleServDisc_FindService() with the gBleSig_RangingService_d UUID as parameter. On the gServiceDiscovered_c event in the registered callback, store the handles of the RAS and its characteristics and characteristic descriptors for later use. Some handles also need to be registered with the localization component via the following APIs:

   1. RasClient_SetRasControlPointHandle
   2. RasClient_SetRasRealTimeHandle
   3. RasClient_SetRasSupportedFeatures

4. Proceed to use the regular GATT client APIs such as GattClient_ReadCharacteristicValue() and GattClient_WriteCharacteristicDescriptor() in order to read the peer's supported RAS features and enable notifications/indications on the RAS Control Point CCCD and on either the On-Demand or Real-Time CCCDs. If you choose On-Demand as a data transfer method, you must also enable notifications/indications on the RAS Data Ready and Data Overwritten CCCDs. The client is now considered subscribed to the RAS and the CS Procedure can start by following the steps described in the sections above.
5. Depending on the RAS settings chosen by the client application, communication from the server arrives as either GATT notifications or indications. In the GATT notification or indication callback, the following APIs should be used:
   1. RasClient_StorePeerMeasurementData, when the On-Demand or Real-Time handle is notified/indicated.
   2. RasClient_ProcessRasCPRsp, when the Control Point handle is notified/indicated.
   3. RasClient_ProcessRasDataReadyIndications, when the Data Ready handle is notified/indicated.
   4. RasClient_ProcessRasDataOverwrittenIndications, when the Data Overwritten handle is notified/indicated.

Perform the following steps to integrate the Ranging Service on the application that has the role of Ranging Responder (GATT server):

1. Add the following component to the application project via Kconfig:
   - CONFIG_MCUX_COMPONENT_middleware.wireless.ble_profiles_ranging=y
2. Ensure the gRasRRSP_d and gAppRasDataTransfer_d defines are present and set to 1 in the application's app_preinclude.h file.
3. Add the Ranging Service and its characteristics to the GATT database in the application project's gatt_db.h file as in the excerpt below:

```
PRIMARY_SERVICE(service_ranging, gBleSig_RangingService_d)
 CHARACTERISTIC(char_ras_ctrl_point, gBleSig_RasControlPoint_d,␣
↪(gGattCharPropWriteWithoutRsp_c | gGattCharPropIndicate_c) )
    VALUE_VARLEN(value_ras_ctrl_point, gBleSig_RasControlPoint_d, (gPermissionFlagWritable_
↪c | gPermissionFlagWriteWithEncryption_c ), 0x05, 0x01, 0x00)
    CCCD(cccd_ras_ctrl_point)
 CHARACTERISTIC(char_ras_stored_data, gBleSig_RasOnDemandProcData_d,␣
↪(gGattCharPropNotify_c | gGattCharPropIndicate_c) )
    VALUE_VARLEN(value_ras_stored_data, gBleSig_RasOnDemandProcData_d,␣
↪(gPermissionNone_c), gAttMaxNotifIndDataSize_d(gAttMaxMtu_c), 0x01, 0x00)
    CCCD(cccd_ras_stored_data)
 CHARACTERISTIC(char_ras_real_time_data, gBleSig_RasRealTimeProcData_d,␣
↪(gGattCharPropNotify_c | gGattCharPropIndicate_c) )
    VALUE_VARLEN(value_ras_real_time_data, gBleSig_RasRealTimeProcData_d,␣
↪(gPermissionNone_c), gAttMaxNotifIndDataSize_d(gAttMaxMtu_c), 0x01, 0x00)
    CCCD(cccd_ras_real_time_data)
 CHARACTERISTIC(char_ras_ranging_data_ready, gBleSig_RasProcDataReady_d,␣
↪(gGattCharPropNotify_c | gGattCharPropIndicate_c) )
    VALUE(value_ras_ranging_data_ready, gBleSig_RasProcDataReady_d, (gPermissionNone_c),␣
↪0x02, 0x00, 0x00)
    CCCD(cccd_ras_data_ready)
 CHARACTERISTIC(char_ras_ranging_data_overwritten, gBleSig_RasprocDataOverwritten_d,␣
↪(gGattCharPropNotify_c | gGattCharPropIndicate_c) )
    VALUE(value_ras_ranging_data_overwritten, gBleSig_RasprocDataOverwritten_d,␣
```

(continues on next page)

(continued from previous page)

```
↪(gPermissionNone_c), 0x02, 0x00, 0x00)
    CCCD(cccd_ras_data_overwritten)
CHARACTERISTIC(char_ras_feature, gBleSig_RasFeature_d, (gGattCharPropRead_c) )
    VALUE(value_ras_feature, gBleSig_RasFeature_d, (gPermissionFlagReadable_c |␣
↪gPermissionFlagReadWithEncryption_c), 4, 0x0F, 0x00, 0x00, 0x00)
```

4. Initialize RAS by calling Ras_Start() with the handles defined in the GATT database.
5. Upon the MTU Exchange procedure's completion, call Ras_SetMtuValue() to inform the module.
6. The client's interactions with the server are processed in the GATT server callback. Here the following APIs are used:
   1. Ras_Subscribe when the client subscribes to either On-Demand or Real-Time CCCDs.
   2. Ras_SetDataSendPreference to save the client's preference for the data transfer (either notifications or indications).
   3. Ras_ControlPointHandler when the client writes a command to the RAS Control Point.
   4. Ras_Unsubscribe when the client unsubscribes or disconnects.

Distance Measurement Algorithm

Introduction

The SDK provides the RADE (Rapid Accurate Distance Estimator) algorithm for precise distance measurement between devices. This algorithm leverages the LCE (Localization Compute Engine) accelerator to calculate distances with high accuracy and low latency.

Integration

Perform the following steps to integrate the RADE algorithm on the distance-measuring application:

1. Add the following components to the application project via Kconfig:
   - CONFIG_MCUX_COMPONENT_middleware.wireless.rade2_algo_cm33_lce_lib=y
   - CONFIG_MCUX_PRJSEG_module.board.wireless.app_lce=y
   - CONFIG_MCUX_PRJSEG_module.board.wireless.linker_script_ble=n
   - CONFIG_MCUX_PRJSEG_module.board.wireless.linker_script_ble_loc=y
2. APP_InitLce() will be automatically called by AppLocalization_Init() during application initialization.
3. Heap areas required by the algorithm are defined and registered in app_lce_init.c.

Bluetooth Channel Sounding technique for distance estimation

Introduction

The low-power 2.4-GHz ISM band wireless standards, such as Bluetooth 2 The channel sounding standard Low Energy (BLE), have broad adoption in consumer products such as smartphones, tablets, smartwatches, or edge nodes in Internet of Things (IoT) networks. As the application domains of the BLE technology expand into industrial and automotive markets, their mass adoption and omni-presence makes this technology usable for applications that go beyond data transfer between devices. The radios included in these products can be used for more complex use cases, such as distance estimation, angle (of arrival/departure) computations, and localization applications. They span the joint estimation of both the angle and the distance when used with specialized hardware enabled as anchors.

This document describes the distance measurement between 2 Bluetooth LE devices using NXP MCUs. This solution only applies to Bluetooth LE devices. This document refers to the KW47 and KW43 MCU and its associated SDK. The SDK also contains the Wireless Localization middleware, which uses a special distance-estimation technique called Channel Sounding (CS). By enabling the CS feature between Bluetooth devices, a measured channel response can be used to calculate the position of a Bluetooth device in a high accuracy mode (at a distance of 5 m with an accuracy of ± 0.5 m in ideal conditions). Channel sounding verifies whether a device is in a predefined area, which makes beacons easier to install and improves indoor localization. Deriving a distance estimate using CS requires 1 device to initiate the process (initiator) and a 2nd device to reciprocate the transmission (reflector). The Measuring Device (MD) and Reflector Device (RD) terms are also used. This solution does not require multiple antennas (but can benefit from their presence) to compute the distance between 2 nodes. There are many aspects to consider when creating a localization solution that meets unique requirements. This application note highlights the aspects that impact performance with a focus on how to use NXP enablement to create your own solution. This document also refers to the following documents for localization purposes:

- Channel Sounding Wireless Ranging Demo Application
- KW47 Ranging Performance Measurement
- KW47-LOC Board User Manual
- KW43-LOC Board User Manual
- FRDM-KW43 Board User Manual

This application note describes a method to estimate the distance partly based on Round-Trip Phase (RTP) measurements. The legacy name of the RTP is Phase-based Distance Estimation (PDE). The phase of RF signals is proportional to the frequency of the carrier and the distance traveled. The method is based on measuring the received signals' phases relative to their local oscillators. The measurements are then performed using tone signals at different frequencies and combined for a final distance estimation.

A part of the description incorporates a method of measuring the Round-Trip Time (RTT) of the RF signal. The legacy name of the RTT is the Time of Flight (ToF) distance estimation. This measurement has lower accuracy when compared to the phase measurement. However, it has an inherent advantage of adding security. The security aspect is described as well.

Bluetooth Channel Sounding high-level requirements

This chapter provides several aspects of CS requirements for localization demands. The distance-measurement accuracy is divided into 2 categories. The measured channel response can be used to calculate the position of a Bluetooth device with the high accuracy mode - defined at a distance of 5 m with an accuracy of ± 0.5 m.

The CS also defines conditions for non-stationary devices with the maximum relative speed between the initiator and the target. The maximum absolute speed of either device is less than or equal to 5 km/h (typical walking speed). The device should be capable to make at least 2 distance measurements per second to realize real-time distance estimation to an object moving at a speed of 5 km/h.

The immunity to relay attacks is also part of the CS specification. The mitigation of both packet and extender relay attacks is at least as cryptographically strong as a link layer encryption. The link layer security must be enabled as a prerequisite for channel-sounding operations. The protection against symbol manipulation attacks is covered by the resiliency to the Early Detect Late Commit (EDLC) and symbol attacks on the RTT packets. This is provided by symbol sharpening and shortening.

Another requirement defines the attack detection in terms of distance estimation. The CS should detect an attack in which the distance measurement is spoofed by more than:

- 3 m for distances of up to 15 m
- 20 % of the total distance for distances greater than 15 m

The channel sounding standard description

This chapter focuses on the Channel Sounding (CS) standard description. The CS feature consists of a combination of software link layer procedures and dedicated air interface protocol that creates a tightly interlocked exchange of RF signals at multiple RF channels. This mechanism is called a CS procedure. Before describing it, it is important to describe the high-level software architecture.

Software layers architecture description

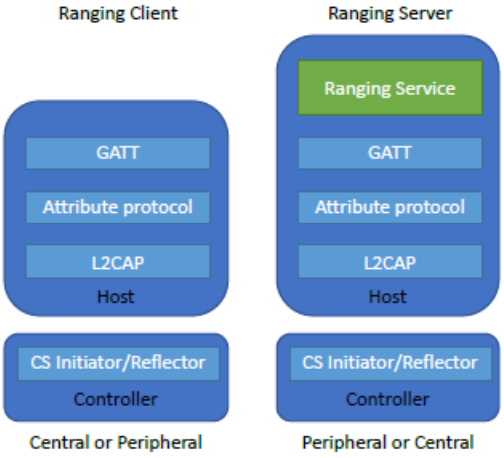
To enable the CS feature between 2 Bluetooth devices, a general service is proposed. In this service, the client and server roles are defined. Typically, the client role in the service is the CS initiator and the server role is the CS reflector. However, there is also a scenario, where the client role is the CS reflector and the server role is the CS initiator. The complete list of scenarios is shown in Table 1.

Table 1. Scenarios for ranging server and ranging client

	Scenario I		Scenario II	
Host Controller	Ranging Client Initiator	Ranging Server Reflector	Ranging Client Initiator	Ranging Server Reflector

The Ranging Server (RAS) and Ranging Profile (RAP) enable the CS configuration parameters and CS results transport toward the Distance Measurement Application. The complete exchange of unmodulated carrier signals and/or packets is controlled by lower Bluetooth protocol layers (Bluetooth controller).

The high-level SW architecture of the ranging server and ranging client



The ranging client and ranging server must exchange their capabilities via the RAS capabilities characteristic. The ranging client writes (or reads) the configuration to the Ranging Server through the RAS configuration characteristic. Then, the 2 controllers start the ranging procedure. Once the ranging results are received by the RAS from the controller, the results will be exchanged using GATT toward the Distance Measurement Application. Some implementations may optionally use the enhanced attribute protocol to make notifications more reliable. The roles defined in the ranging service and profiles are as follows:

- Ranging client: the device in this role supports the GATT client role and optionally the enhanced attribute protocol and the CS initiator or reflector roles The ranging client can have either the central or peripheral roles. The ranging client has a direct interconnection with the Distance Measurement Application.

- Ranging server: the device in this role supports the ranging service and profile. It implements the GATT server role and optionally the enhanced attribute protocol role. In the controller, it implements the CS reflector or initiator roles. The ranging server has either the central role or the peripheral role. The ranging server assists the ranging client by sending the local device capability, receiving the ranging client configurations, and sending the ranging result toward the ranging client.

The Distance Measurement Application will be implemented in the device that supports the ranging client. The hosts may support both the ranging client and ranging server implementations. The higher layer will choose which device runs the distance application and becomes the ranging client. Figure 1 illustrates the high-level architecture of the ranging server and ranging client for 2 scenarios listed in Table 1. In the asset-tracking and indoor-positioning applications, the distance measurement application may calculate the distance between:

- A ranging client and more than one ranging server
- A ranging server and more than one ranging client

The CS standard (the part of the ranging server and ranging profile) also mentions examples of the typical application use cases. Table 2 shows several typical localization applications.

Table 2. Localization user scenario examples

User Scenario	Ranging Client	Ranging Server
Vehicle access	Vehicle Bluetooth LE lock (single Bluetooth LE initiator, an initiator with multiple antennas, or multiple initiators each with a single antenna or multiple antennas)	Vehicle key (can be a phone)
Door access	Door Bluetooth LE lock	Door key (can be a phone)
Locate lamps, sensors, and actuators during commissioning or maintenance	Handhold commissioner	Other lamps/sensors
Locate items with a smartphone	Bluetooth LE localization system (1 initiator or multiple initiators)	Tags
Locate items with a handheld locator device	Handhold with Bluetooth LE	Tags
F: Automatically map mesh nodes	Some/all lamps must support dual roles (initiator and reflector devices)	Some/all lamps must support dual roles (initiator and target devices)
G: Cooktop control	Cooktop	Smartphone
H: Cooktop safety	Cooktop	Smartphone

Note: The CS standard itself is not responsible for computing the distance measurement

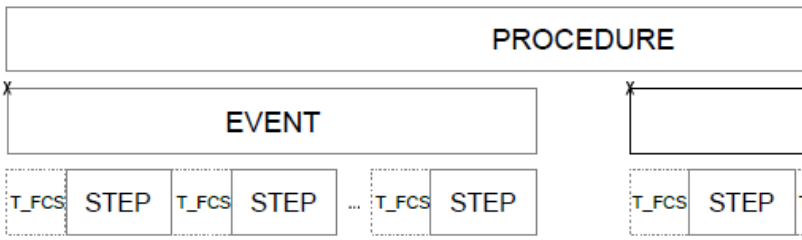
Parent topic:*The channel sounding standard description*

The channel sounding procedure description

The CS procedure consists of several events that are defined as a set of time and frequency slots, in which 2 devices agree to communicate and exchange a combination of RF signals. The main purpose of this exchange is to estimate the physical characteristics of the transmission channel. These exchanges are always bidirectional.

The CS procedure is divided into one or more events. The CS event can consist of one or more CS steps that are separated by a frequency change period called T_FCS. Time separation between

CS events avoids continuous allocation of the RF resources of a given device and facilitates coexistence with other RF technologies. Figure 2 shows the relationship between CS procedures, CS events, and CS steps.



Channel Sounding procedure composition

Each CS event has a separate anchor point labeled 'X'. The link layer configures devices to run this procedure. It defines all the steps for these events in both time and frequency relative to these anchor points. The output of the CS procedure is a table of phase correction terms for both the initiator and the reflector and/or a table of time stamps. The processing algorithms translate those tables into distance estimates. The processing algorithms are not part of the standard. The procedure is composed of one or more CS events. The CS event is composed of the multiple CS steps. There can be programmable gaps between CS events to allow other simultaneous connections.

Four CS step types (mode 0, mode 1, mode 2, and mode 3) are defined. Each mode is used for a specific purpose:

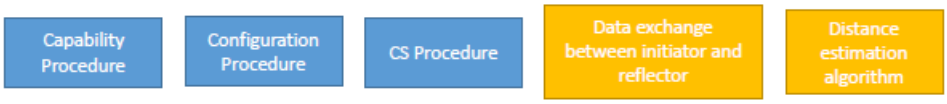
- Mode 0 is used to calibrate 1 side to the other in terms of frequency and timing.
- Mode 1 is used to exchange a Round Trip Timing (RTT) packet.
- Mode 2 is used to exchange Round Trip Phase (RTP) packets and to measure the phase and amplitude of the communication channel.
- Mode 3 is used to exchange both RTT and RTP measurement packets.

In the CS context, an initiator is the device that starts (initiates) the whole procedure and a reflector is the device that responds to (reflects) the CS procedure. The procedure's operating parameters are exchanged via link layer control messages. Once the procedure completes, each device has a set of information that describe the real communication channel. Mode 1 and mode 3 measure the time of arrival and the time of departure. Mode 2 and mode 3 measure the phase and amplitude information in the form of in-phase (I) and quadrature-phase (Q) components. The reflector communicates this information back to the initiator. The following chapters describe 4 different step types.

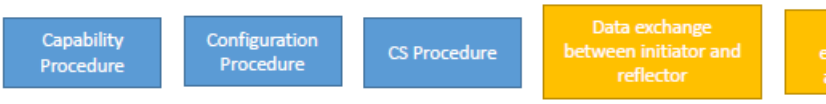
Parent topic:*The channel sounding standard description*

The CS high-level view

The high-level view of the several procedures needed for the distance estimation is shown in **Distance estimation - high-level view**

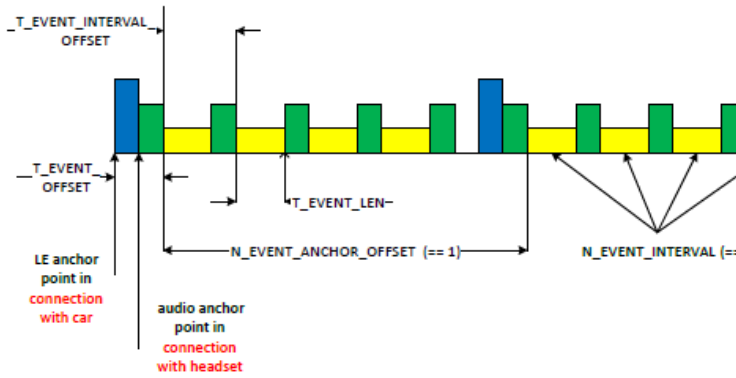


. A simple flowchart shows the procedures described in the CS standard. It also contains yellow boxes, which are not directly defined by the CS standard. The data exchange between the initiator and the reflector cover for example the IQ data, RTT timestamps, and so on. The data exchange is planned to be a part of the ranging service profile.



Distance estimation - high-level view

The example of the CS events interleaved with LE connection events is shown in Figure 4. This should improve the understanding how the CS events are interleaved with the LE connection events. The CS events are scheduled at an offset from the LE connection anchor points. Only 1 CS event is scheduled in this scheme. As an example, a smartphone in a key fob role may be connected to an automobile for localization purposes. The connection interval of this LE connection may be very long, because its main purpose is to start CS procedures.



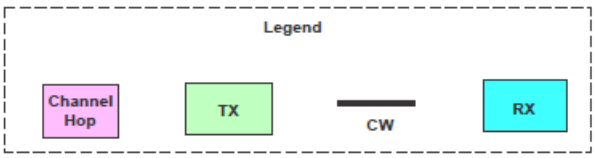
CS event scheduling in coexistence scenario

At the same time, the phone may be connected to a headset for audio connection purposes. To allow for flexible scheduling of CS events, multiple CS events can be scheduled offset from single LE connection events. A CS event offset is used to separate the multiple CS events in time. The number of CS events allowed in between LE connection events is selectable.

Parent topic:*The channel sounding standard description*

Channel Sounding steps description

The CS defines a set of interlocked transfers between initiator and reflector. Within a CS procedure, CSStepCount shall be set to the current CS step number. The CS steps are constructed by a combination of modulated and unmodulated RF signals. The modulated sections convey synchronization information and extract the round-trip time information. The unmodulated sections are used to measure the phase rotation, which is proportional to the distance between the devices. The CS steps use the dedicated packet formats. There are 4 different modes for the CS steps. Each of these modes has different usage goals: measuring frequency offset between devices (mode 0), measuring round-trip times (mode 1), measuring phase rotations due to distance (mode 2), and measuring both round-trip times and phase rotations (mode 3). The CS steps require a precise timing synchronization between the initiator and reflector devices. Figure 5 shows a legend of the diagrams used in the following chapters to describe individual CS step modes.



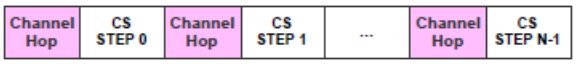
Legend for CS step type packet description

The colored time slots represent the following:

- Channel Hop time slot
- TX - transmitting time slot
- CW - unmodulated continuous wave signal
- RX - receiving time slot

The CS steps are separated by periods to perform a frequency hop. The structure is shown in Figure 6.

Channel hop period separates CS steps



The time for the frequency hop is known as T_FCS. The initiator and reflector devices exchange the list of T_FCS values that each one can support during the capability procedure. The initiator selects a common T_FCS value during the configuration procedure. The permitted values for T_FCS are given by the BLE standard. Within a CS procedure, the same T_FCS value should be used for each channel hop. Both devices use the T_FCS period to perform internal calibrations in addition to the frequency change. The initiator or reflector may use this time period to allow the settling of the transmitted RF signal. It means that it reached a stable state when the subsequent CS step begins.

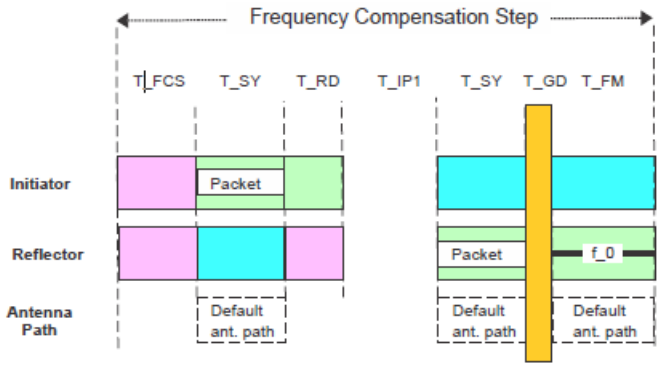
The set of variables used in the following chapters describes the CS step modes, as shown in Table 3.

Table 3. Channel Sounding step variables

Variable	Description	Time duration [us]
T_FC	Time for frequency change spacing. The RF circuitry locks to the measurement channel (f_i). Devices can also use this time for additional internal calibrations, if necessary. The initiator may start to ramp up its output signal toward the end of this step	15, 20, 30, 40, 50, 60, 80, 100, 120, 150
T_FM	Time for frequency measurement	80
T_SY	Time for synchronization sequence (CS packet)	26 for 2M PHY, 44 for 1M PHY
T_IP	Interlude period 1. The transition from TX to RX and the other way around when transmitting in mode 1 (Pk-Pk).	10, 20, 30, 40, 50, 60, 80, 145
T_IP	Interlude period 2. The transition from TX to RX and the other way around when transmitting in mode 2 (Tn-Tn).	10, 20, 30, 40, 50, 60, 80, 145
T_GI	Transition time between packet and tone	10
T_RI	Ramp-down time for transmission	5
T_PM	Single-antenna phase measurement period	10, 20, 40
N_A	Number of antenna paths	1, 2, 3, 4
T_SW	Antenna switching period	1, 2, 4, 10
N_A	Phase measurement period. This measurement must be performed for each antenna path. N_AP is the total number of antenna paths. Transmitted signals are expected to be stable during this period. When multiple antenna paths are used, an equivalent number of T_PM periods is used. Each T_PM reserves the first 2 us for antenna switching (T_SW).	-

CS step mode 0

TCS step mode 0 is used to measure the frequency offset between the initiator and the reflector at a given frequency. The step mode 0 is a basic step supported by all the devices with CS feature. Figure 7 shows the exchange defined for CS step mode 0.



CS step Mode 0 packet exchange

The frequency hop is created before each CS step. The initiator starts by transmitting the CS SYNC packet (T_SY). The duration of the packet is shown in Table 3. The initiator performs a measurement of the frequency offset from the transmitted RF signals from the reflector during CS step mode 0. The initiator shall use this information to compensate all further transmissions within the subsequent CS steps in a CS event. This is used for synchronization purposes.

After the transmission from the initiator completes, a defined ramp-down window of 5 us (T_RD) is allowed for the initiator to remove the transmitted energy from the RF channel. After T_RD, the output power shall be lower by 40 dB than the output power used during the transmission of the CS SYNC packet (measured at the initiator's antenna).

T_IP1 represents the idle time between the transmission from the initiator and the transmission from the reflector. Devices may use this time for internal calibrations, if needed. The reflector can use this time to ramp up its transmitted signal (if necessary) in advance of the transmission of the CS SYNC packet (T_SY) with the unmodulated carrier after synchronization. A transition time from packet to tone (T_GD) is needed. Then the reflector device makes the frequency-measurement period. The duration of this period (T_FM) for CS mode 0 shall be 80 us. The frequency measurement period ends the CS step mode 0 procedure.

It is important to mention 2 packet formats for channel sounding. Channel sounding uses a specific modulated bit sequence known as CS SYNC. The CS SYNC packet format is similar to a packet format for uncoded PHY, but it has no CRC. The CS SYNC packet has a specific format shown in Figure 8.

CS SYNC packet format

Preamble (1 or 2 octets)	CS Access Address (4 octets)	Trailer (4 bits)	Sounding Sequence (32 or 96 bits) Or Random bit sequence (32, 64 or 128 bits)

The preamble is 1 octet when transmitting or receiving on the LE 1M PHY and 2 octets when transmitting or receiving on the LE 2M PHY. The CS access address is 4 octets. The trailer is 4 bits. The sounding sequence or random bit sequence are optional fields. If present, the sounding sequence shall be 32 or 96 bits long. If the random bit sequence is present, it shall be 32, 64, or 128 bits long.

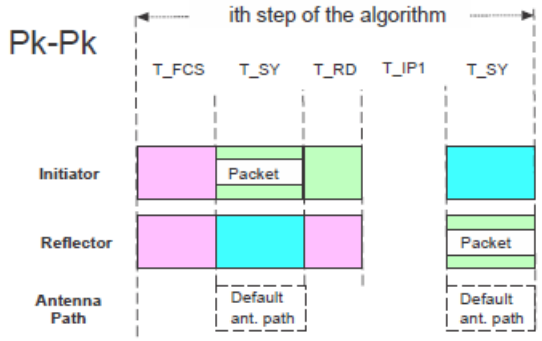
The variable length of the optional sounding sequence and random bit sequence allows an implementation to optimize the total packet length vs the correlation accuracy of the field. CS packets with no sounding or random bit sequence take 44 us to transmit when sent by LE 1M PHY and 26 us to transmit when sent by LE 2M PHY. CS packets that include a sounding or random bit sequence take proportionally longer to transmit, based on the length of the field and the PHY selection.

Parent topic: The channel sounding steps description

CS step mode 1

CS mode 1 is used to measure the round-trip time between the initiator and the reflector. Devices supporting the CS feature shall implement the CS step mode 1. This mode uses the theoretical background described in RTT estimation fundamentals. The packet structure in Figure 9 is defined for CS step mode 1. Before each CS step, a frequency hop is created

CS step mode 1 packet exchange



The initiator starts by transmitting the CS SYNC packet. The duration of the CS SYNC packet is marked as T_SY. In this case, CS step mode 1 may include a payload as part of the CS SYNC packet. After the transmission from the initiator completes, a defined ramp down window of 5 us is allowed. It helps the initiator to remove the transmitted energy from the RF channel. After the ramp down, the output power shall be lower by 40 dB than the output power used during the transmission of the packet, measured at the initiator's antenna.

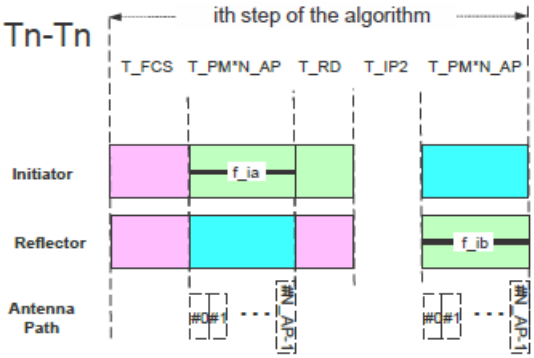
T_IP1 represents the idle time between the transmission from the initiator and the transmission from the reflector. Devices may use this time for internal calibrations, if needed. The reflector can use this time to ramp up its transmitted signal (if necessary) in advance of the transmission of the CS SYNC packet. The duration of T_IP1 is shown in Table 3. After T_IP1, the reflector device transmits its CS SYNC packet. The CS step mode 1 may have a payload as part of the CS SYNC packet. The reflector may transmit its CS SYNC packet even if it does not receive a previous CS SYNC packet from the initiator or if the CS SYNC packet was received with bit errors.

Parent topic: The channel sounding steps description

CS step mode 2

CS mode 2 is used to measure the phase rotations of the RF signal between the initiator and the reflector. The theoretical description of this mode is in RTP estimation fundamentals. The devices supporting this feature should fully support CS step mode 2. The structure of CS step mode 2 is shown in Figure 10.

CS step mode 2 tone exchange



Before each CS step, a frequency hop is created. The duration of the frequency hop is marked as T_FC. The initiator starts the transmission with an Unmodulated Carrier (UC) RF signal. The duration of the unmodulated carrier RF signal should be T_PM*N_AP, where T_PM is the phase-measurement period and N_AP is the number of antenna paths. Each individual T_PM reserves the first configurable time (1, 2, 4, 10 us) for antenna switching. The available values of T_PM are given by the standard and shown in Table 4.

Table 4. Permitted values for T_PM

T_PM Index	T_PM	Comment
1	10 us	Optional
2	20 us	Mandatory if any shorter T_PM extent is supported, else optional
3	40 us	Mandatory

For added security, each N_AP set of T_PM length transmissions shall be followed by a single tone-extension slot of the same T_PM extent. This tone-extension slot may or may not carry a transmission. The presence of the added security is controlled by random bit generation.

After the transmission from the initiator completes, a defined ramp-down window of 5 us is allowed for the initiator to remove the transmitted energy from the RF channel. The same rule is applied here. The output power shall be lower by 40 dB than the output power used during the transmission of the UC RF signal. The signal level is measured at the initiator's antenna. T_IP2 represents the idle time between the transmission including the tone extension slot from the initiator and the transmission from the reflector. Devices may use this time for internal calibrations (if needed). The reflector can use this time to ramp up its transmitted signal (if necessary) in advance of the transmission of the UC. The permitted values for T_IP2 are shown in Table 5. The T_IP1 value can be different from T_IP2.

Table 5. Permitted values for T_IP2

T_IP2 Index	T_IP2	Description
1	10 us	Optional
2	20 us	Optional
3	30 us	Optional
4	40 us	Mandatory if any shorter T_PM extent is supported, else optional
5	50 us	Optional
6	60 us	Optional
7	80 us	Mandatory if any shorter T_PM extent is supported, else optional
8	145 us	Mandatory

After T_IP2, the reflector device transmits its (UC) RF signal. The duration of the unmodulated carrier RF signal should be T_PM*N_AP, where T_PM is the phase-measurement period and N_AP is the number of antenna paths. Each individual T_PM reserves the first 2 us for antenna switching. The N_AP parameter is common to the entire CS procedure and shall range from 1 to 4. An

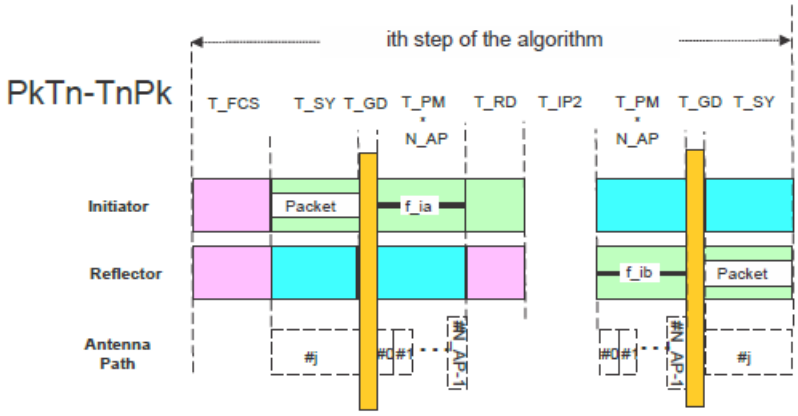
added extension transmission slot shall immediately follow the last valid N_AP transmission, as it was similarly present in the initiator-to-reflector direction. If a transmission is present, it shall be identical to that of the last T_PM slot and it shall use the same antenna element used in that slot. If a transmission is not present in the tone-extension slot, then that T_PM period shall still be present but not carry a transmission. The presence of the added security is controlled by the random bit generation.

Parent topic: The channel sounding steps description

CS step mode 3

CS step mode 3 is used to measure the phase rotations of the RF signal and the round-trip time between the initiator and the reflector. It uses a combination of the previous modes. All the devices supporting the CS feature may optionally implement also step mode 3. Figure 11 is defined for CS step mode 3.

CS step mode 3 packet exchange



A frequency hop is created before each CS step. The channel hop duration is marked as T_FCS. In the reflector to initiator direction, the duration is given by the CS SYNC synchronization packet (T_SY), transition time between the packet mode to tone (T_GD), and the unmodulated carrier RF signal T_PM*N_AP. T_PM is the phase-measurement period and N_AP is the number of antenna paths. Each individual T_PM reserves the first configurable time (1, 2, 4, 10 us) for the antenna switching (T_SW).

If a tone-extension slot is present, then the tone duration is (T_SW+T_PM)x(N_AP+1). If the tone-extension slot is not present, then the tone duration is (T_SW+T_PM) x N_AP. To compensate for the missing tone slot a “no tone” slot is added after the synchronization.

The initiator starts a transmission with the Unmodulated Carrier (UC) RF signal. The duration of the unmodulated carrier RF signal should be T_PM*(N_AP + 1) and it should also depend on the physical layer used. CS step mode 3 may use a payload as part of the synchronization.

For added security, each N_AP set of T_PM length transmissions shall be followed by a single tone-extension slot of the same T_PM extent. This tone-extension slot may or may not carry a transmission. If a transmission is present, it shall be identical to that of the last T_PM slot and it shall use the same antenna element used in that slot. If a transmission is not present in the tone-extension slot, then that T_PM period shall still be present, but it shall not carry a transmission. The presence of a physical transmission in the extension transmission slot is seeded by the random bit generation.

After the transmission from the initiator is completed, a defined ramp down window of 5 us (T_RD) is allowed for the initiator to remove the transmitted energy from the RF channel. After T_RD, the output power should be lower by 40 dB than the output power during the transmission of the synchronization. This is measured at the initiator’s antenna.

T_IP2 represents the idle time between the transmission from the initiator and the reflector. Devices may use this time for internal calibrations (if needed). The reflector can use this time

to ramp up its transmitted signal before the transmission. The duration of T_IP2 is defined the same way as the CS step mode 2 in Table 5.

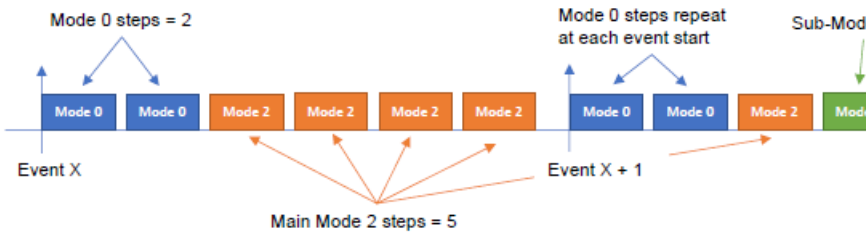
After T_IP2, the reflector device transmits a CS synchronization using the unmodulated carrier. The duration of the unmodulated carrier shall be either T_PMxN_AP or T_PMx(N_AP + 1). This depends on whether a transmission is selected for the tone extension slot. If a transmission is present, it should be identical to that of the last T_PM slot and it shall use the same antenna element as in that slot. If a transmission is not present in the tone-extension slot, then the T_PM period should not be present. The presence of a physical transmission in the extension is controlled by the random bit generation.

The reflector device may then transmit its CS synchronization even if it does not receive a CS synchronization from the initiator or if the synchronization was received with bit errors.

Parent topic: The channel sounding steps description

Channel Sounding mode sequencing

The CS mode sequencing describes specific time slots of the whole flow. The CS event will always start with one or more mode 0 steps, which is represented by blue rectangles in FFigure 12. A series of subsequent non-mode 0 steps shall follow in the same event. Any CS procedure shall only use at most two non-mode 0 modes. One of these two is known as the main mode and the other is known as the submode. An example of this sequencing is shown in Figure 12. Many other CS mode sequencing examples can be generated.



Channel Sounding mode sequencing

The submode insertion sequences the occurrence of the main mode steps and the submode steps and uses the following parameters:

- Main_Mode_Min_Steps – the minimum main-mode steps that shall occur before a single submode step.
- Main_Mode_Max_Steps – the maximum main-mode steps that shall occur before a single submode step.

The number of main-mode steps that occur before a submode step shall be between Main_Mode_Min_Steps and Main_Mode_Max_Steps, inclusive. The exact number of main-mode steps that occur before a submode step insertion is randomized for each sequence. The randomized function is defined by the BLE standard.

Parent topic: The channel sounding steps description

Parent topic:*The channel sounding standard description*

Wireless channels randomization

The whole ISM frequency band is used for the CS technique for distance estimation. There is an important aspect about the frequency channel switching. The ISM band is not swept linearly, but there is a procedure to create randomly selected channels. The BLE standard assumes the use of the channel-selection algorithm for two cases:

- Channel-selection algorithm for mode 0 steps
- Channel-selection algorithm for non-mode 0 steps

The channel-selection algorithm #3a is used to generate a randomized channel map with uniform distribution for mode 0 CS steps. The number of times that the mode 0 channel list is required to be regenerated in a CS procedure depends on several factors, including the size of the list of included filtered channels, as indicated by the filter-channel bitmap Chesterfield (the number of non-mode 0 steps included in a CS event and a CS procedure along with the runtime construction of CS events whose content may vary from event to event), depending on the CS mode sequencing.

The channel-selection algorithms #3b and #3c support channel selection for each non-mode 0 CS step within a CS procedure. Only one of the two algorithms shall be used within a single CS procedure. The number of times that the non-mode 0 channel list is required to be regenerated in a CS procedure depends on several factors, including the size of the list of included filtered channels, as indicated by the filter channel bitmap CSFilteredChM, the number of times the filtered channel map is repeated in a procedure for non-Mode 0 steps, along with the runtime construction of CS events whose content may vary from event to event (depending on the CS mode sequencing).

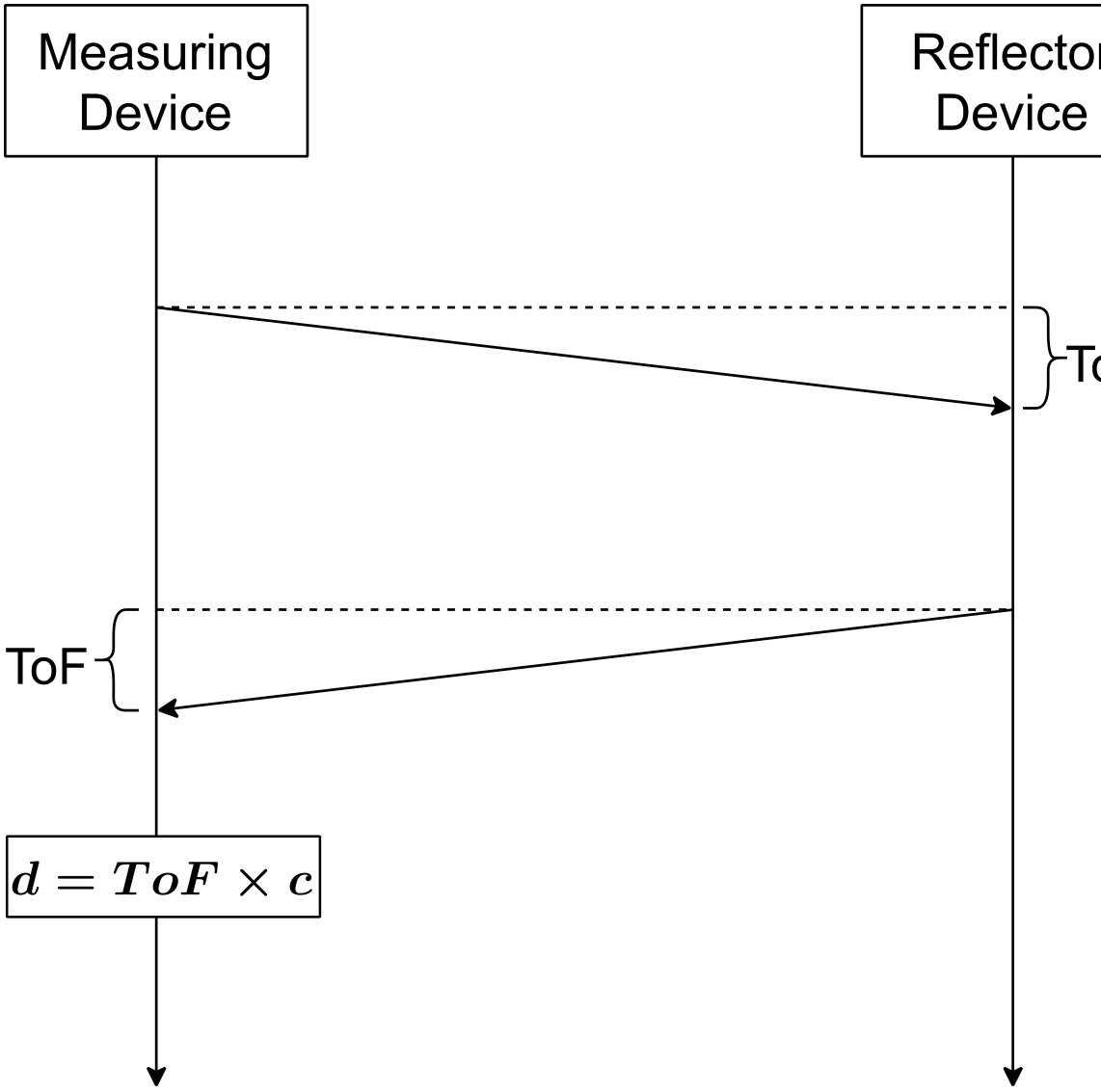
Detailed information about the wireless channels' randomization algorithms is in the BLE standard.

Parent topic:*The channel sounding standard description*

RTT estimation fundamentals

The round-trip time (previously known as the time of flight) is a method to measuring the distance between a sensor and an object, based on a precise measurement of the time difference between the transmission of a signal by the sensor and its return to the sensor after being reflected by an object. This approach is part of the RTT measurement. Wireless ToF applies the same concept to two wireless devices that can perform distance measurement between them by exchanging packets and accounting for radio latencies and turn-around time overheads.

The narrowband ToF requires at least two devices with NBW radio support to exchange packets over the air. As shown in Figure 13, it is based on a record of times at which the initial packet was transmitted by the MD and the time at which a response packet is received from the RD by the MD. The MD can calculate a ToF of the packet exchange. Using the calculated ToF, a distance estimate (or range) between the two devices can be computed. The determined distance (d) can be employed for a variety of functions, such as taking a specified action when the distance is above or below a threshold.



Basic ToF sequence

As shown in Figure 13, a method to capture the timestamps at which the packets are transmitted and received is required at both ends. Furthermore, these timestamps must be shared from RD to MD for the ToF calculation. The MD records a count from a timer (designated T1) in response to initiating transmission of the ranging packet. In response to receiving the ranging packet, the RD records a count from a local timer (designated T2). The RD takes a certain amount of time to process the incoming packet and send a response packet. This time is designated as "Processing Time", which is an overhead that must be eliminated to get an accurate ToF reading. In response to initiating the transmission of the response packet, the RD stores the count (designated T3). The response packet includes a data payload indicating the timestamps to calculate the RD processing time. Upon receiving the response packet, the MD records the count (designated T4). With these timestamps, a ToF value can be generated according to Equation 1..

Calculation of ToF

$$ToF = \frac{(T4 - T1) - (T3 - T2)}{2}$$

Once the ToF value is calculated, the distance can be estimated using Equation 2..

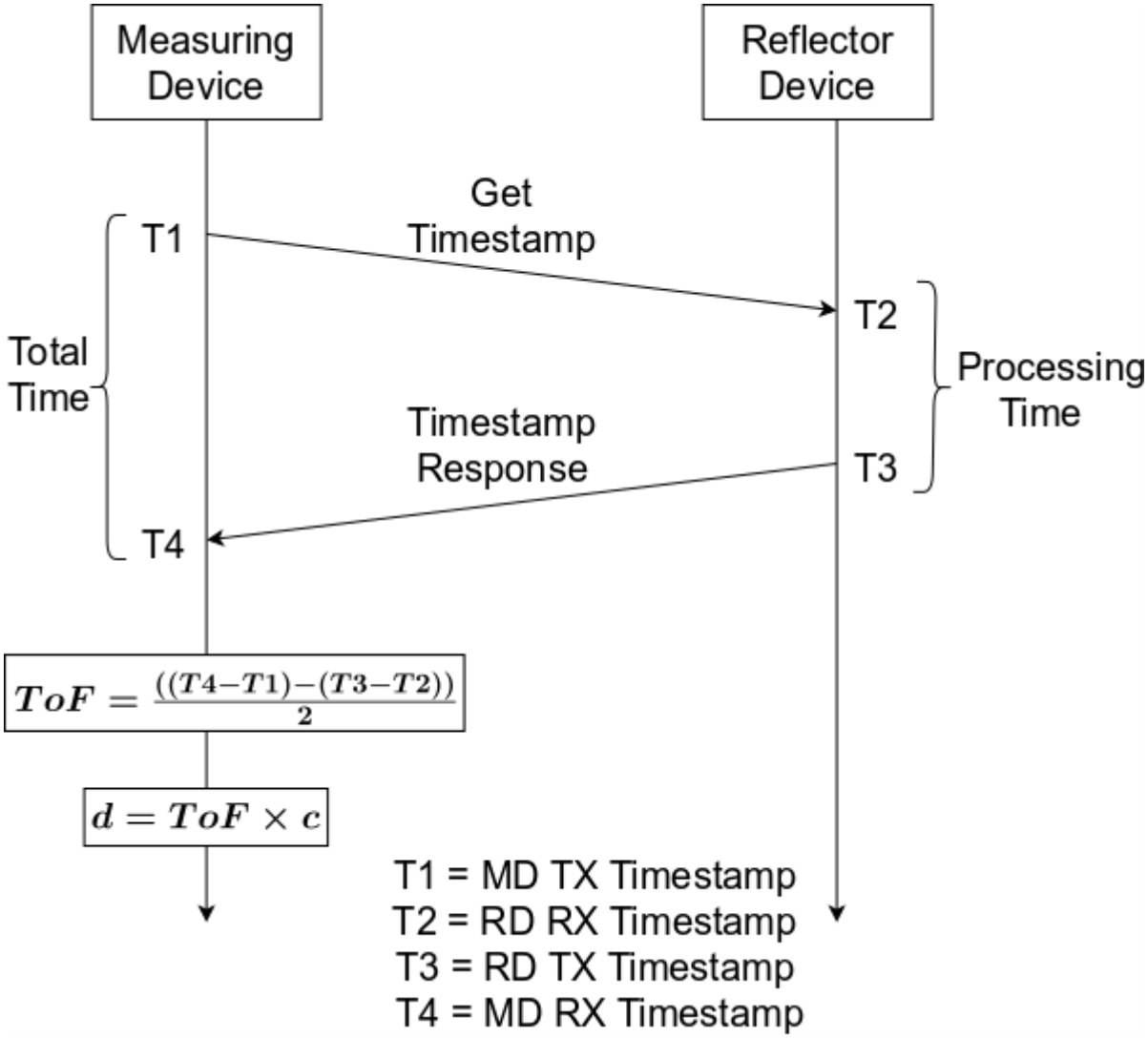
Distance estimation using ToF

$$Distance_m = ToF \times c$$

Figure 14 shows the sequence of packet exchange and timestamp collection for a single measure-

ment.

ToF calculation using timestamps



Narrowband ToF

Radio frequency waves travel at the speed of the light (c), which is a known constant (c 3×108m/s); 1 m is equivalent to ~3.3 ns. A system measuring the ToF would ideally require a high-speed clock (GHz) to generate the timestamps with 1-ns accuracy, which is not possible with today's low-cost deep submicron technologies. For mass-market applications, where low power consumption, low system complexity, and a tight cost budget are a constraint, a low-cost MCU with an integrated 2.4-GHz radio can be used. Typically, these devices work with a high-accuracy reference clock (Fref). As an example, using a reference clock of 32 MHz provides raw ToF accuracy of about 10 m, which may be adequate to validate a more precise but spoofable distance measurement. However, 10 m is too coarse and might not be useful as most applications require accuracy that is much finer. A typical target is to have accuracy of less than 1–2 m. To achieve such accuracy, a number of ToF measurements may be taken (approximating a normal distribution) and averaged. When using ToF distance estimation, there is a trade-off between accuracy and measurement time.

A narrowband ToF measurement systems consists of the following:

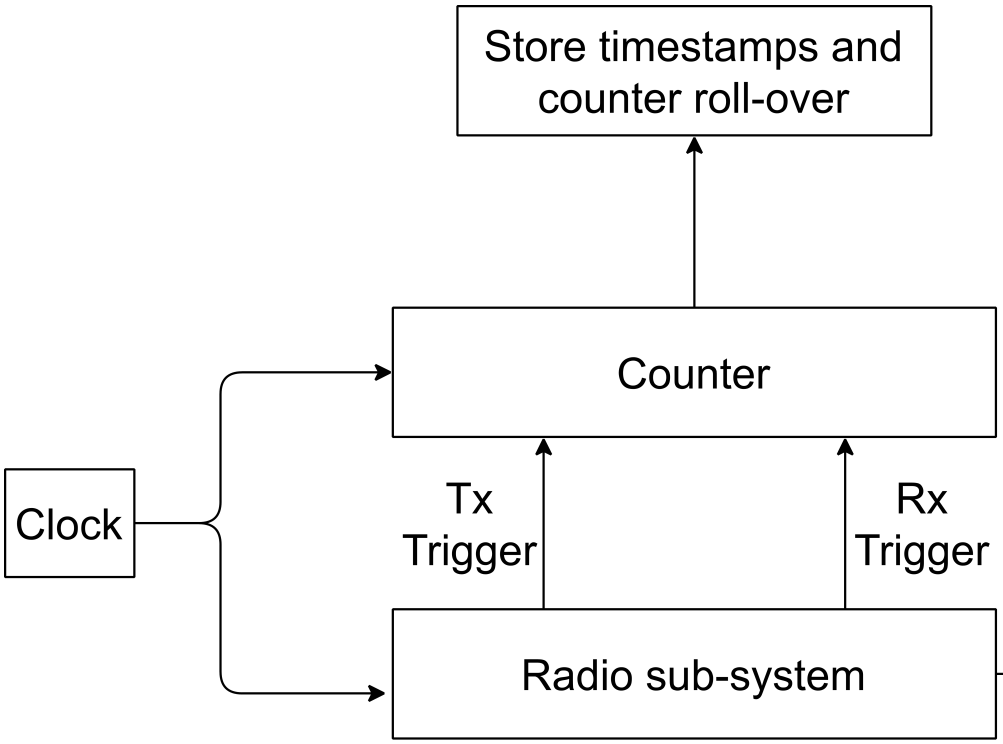
1. Counter for timestamping that is synchronized to the radio subsystem clock reference (Fref)
2. Precise hardware-based timestamping mechanism

3. Measurement protocol that includes the following steps:
    - Timestamp collection
    - Preprocessing
    - Postprocessing
    - Measurement report

Timestamping mechanism

A precise timestamping mechanism is required on both ends for the NBW ToF to achieve the targeted accuracy. At the system level, this not only imposes requirements on the stability and relative accuracy of the timestamping clocks used by both MD and RD, it is also desired that the timestamp-capture mechanism does not contribute to additional jitter. This requirement typically implies that timestamping must be hardware-based and utilize precise triggers for both packet transmission and reception.

To generate precise timestamps, a deterministic and accurate counter is required. For instance, a 16-bit MCU timer, clocked by the radio's reference clock, can be chosen. Given the counter-clock frequency and its size, the count overflows and roll-overs happen every few milliseconds, which must be accounted for as well. The timestamp trigger should be generated from the radio SoC to ensure a stable trigger at specific anchors of the packets. For instance, the transmission timestamp source could be taken from a signal asserted when the system is ready to transmit the first preamble bit. On the receiving end, a packet-reception trigger source can happen when the radio detects the synchronization delimiter (such as the access address in Bluetooth LE or Start-of-Frame Delimiter (SFD) for IEEE 802.15.4) arrival for the packet. Figure 15 shows the timestamp block diagram.



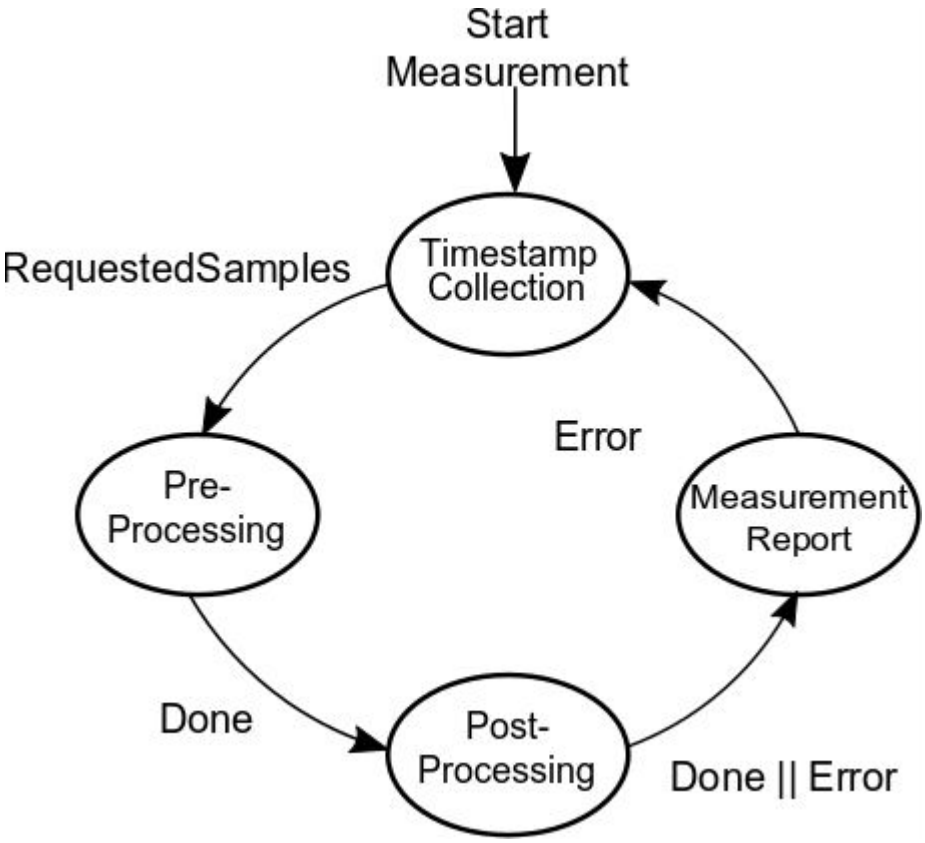
Timestamp block diagram

Parent topic: Narrowband ToF

Multi-stage ToF

The ToF measurement consists of four stages: timestamp collection, preprocessing, postprocessing, and measurement report

ToF stages

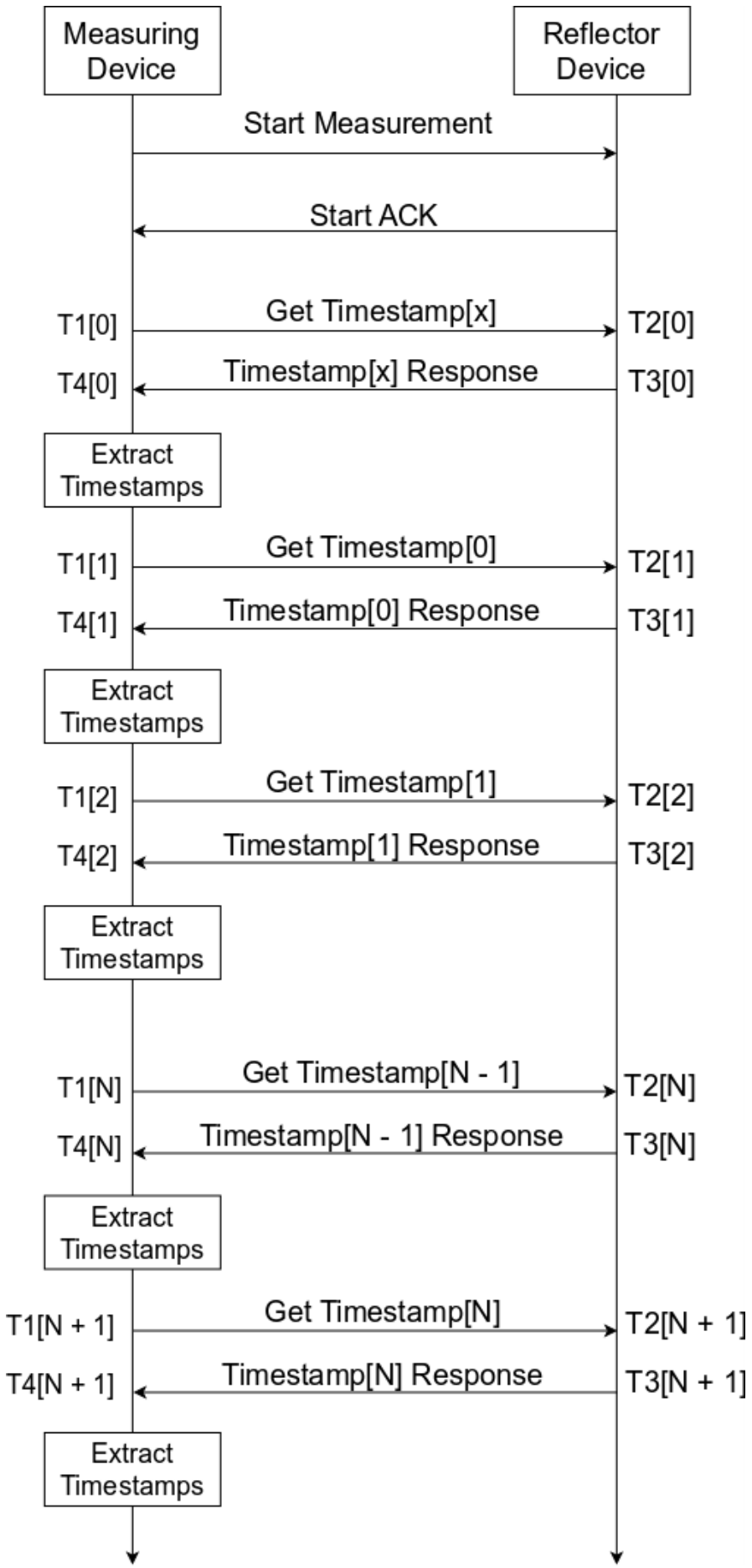


Timestamp collection

During this stage, the devices perform a series of packet exchanges. Each exchange creates a set of timestamps T1 – T4, which are made available to the MD for analysis. As T2 and T3 are generated at RD, they shall be shared on the response packet along with other information that might be useful, such as RSSI. As T3 (RD TX Timestamp) is generated by a transmission of the response packet, a causal implementation is possible by having the RD response to the MD contain timestamps corresponding to the previous packet exchange, as shown in Figure 17.

Wireless communication, especially in the shared Industrial, Scientific, and Medical (ISM) frequency band can be hampered by interference due to proximity of other radios utilizing the same frequency band and/or multi-path propagation artifacts. Using multiple carrier frequencies for ToF can reduce the impact of such artifacts and increases the likelihood of a successful ToF measurement. As explained in this document, a ToF distance measurement typically comprises of exchanging a set of packets for physical ToF measurements (by timestamping) as well as control communication between the two wireless nodes. Due to interference and artifacts, any of the packets containing either the timestamp requests, timestamped measurements, or measurement command packets might fail to arrive in either direction. For robust operation, a ToF measurement system should incorporate a robust mechanism that can recover from such events. Such a mechanism must be intelligent enough to identify and skip specific carrier frequencies if they have persistent interference and continue with the measurement on a different frequency, rather than just abort due to the missed packets.

Timestamping collection


Measuring Device
Reflector Device
Start Measurement
Start ACK
T1[0]
Get Timestamp[x]
T2[0]
T4[0]
Timestamp[x] Response
T3[0]
Extract Timestamps
T1[1]
Get Timestamp[0]
T2[1]
T4[1]
Timestamp[0] Response
T3[1]
Extract Timestamps
T1[2]
Get Timestamp[1]
T2[2]
T4[2]
Timestamp[1] Response
T3[2]
Extract Timestamps
T1[N]
Get Timestamp[N - 1]
T2[N]
T4[N]
Timestamp[N - 1] Response
T3[N]
Extract Timestamps
T1[N + 1]
Get Timestamp[N]
T2[N + 1]
T4[N + 1]
Timestamp[N] Response
T3[N + 1]
Extract Timestamps

Parent topic: Narrowband ToF

Preprocessing

With the timestamps already generated and stored at MD, the pre-processing stage calculates a ToF estimate for every T1 – T4 sample set. Such calculations can be performed using integer math. However, to improve the measurement accuracy, fixed-point or floating-point math can be used. To calculate each ToF estimate, the system performs the following steps:

1. Calculate the process time in function of timer counts. Any timer overflow must be accounted in Equation 3.

Equation 3. Process time calculation

$$ProcessTime = T3 - T2$$

2. Calculate the total time in function of timer counts. Any timer overflow must be accounted in Equation 4.

Equation 4. Total time calculation

$$TotalTime = T4 - T1$$

3. Translate the process time from TimerClockFrequencyHz counts to seconds, as shown in Equation 5.

Equation 5. Process time to seconds

$$ProcessTime_s = \frac{ProcessTime}{TimerClockFrequencyHz}$$

4. Translate the total time from TimerClockFrequencyHz counts to seconds, as shown in Equation 6.

Equation 6. Total time to seconds

$$TotalTime_s = \frac{TotalTime}{TimerClockFrequencyHz}$$

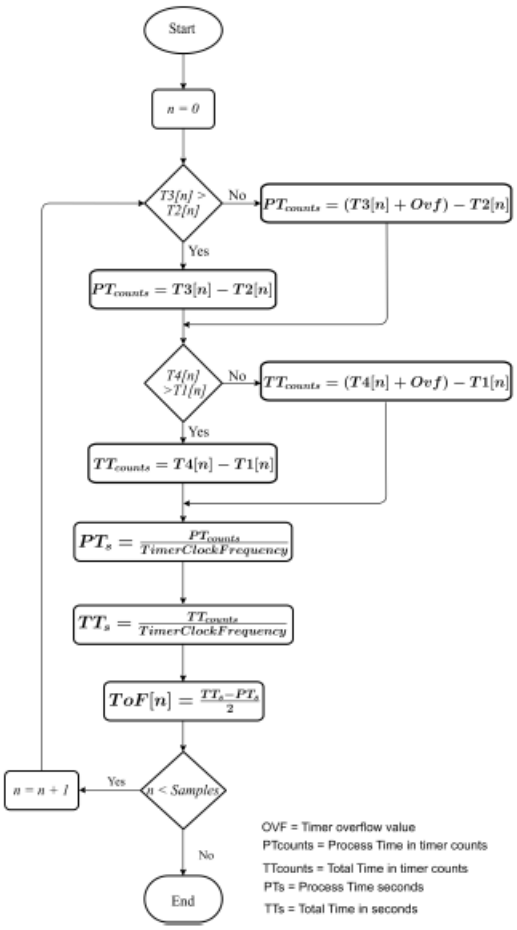
5. Calculate the ToF per sample, as shown in Equation 7.

Equation 7. ToF in seconds

$$ToF = \frac{TotalTime_s - ProcessTime_s}{2}$$

Figure 18 shows the previously discussed steps for n samples.

Individual ToF measurement and preprocessing flowchart



For a radio, it is typical to exercise the receiver's Automatic Gain Control (AGC) dynamic range to operate over the entire link budget or in response to the presence of interference. The AGC operation typically ensures that a certain level of down-converted signal is maintained at the receiver's Analog-to-Digital Conversion (ADC) over the receiver's entire dynamic range, until the receiver gain hits either a minimum or a maximum. Since the measurement resolution targeted in the ranging-distance estimates is in nano-seconds, any propagation time variability within the radio's analog front end due to gain adaptation must be carefully treated as a potential source of error, while performing the ToF distance estimation. This can be avoided by either performing a gain vs time propagation variability calibration. This can be addressed by choosing a robust method to choose an optimal AGC setting for a ToF measurement.

Parent topic: ToF Multistage

Postprocessing

The postprocessing step is typically applied over a set of individual ToF measurements with a goal to choose a reliable distance estimate from several single-shot measurements that may be individually lacking both accuracy and resolution. Statistical estimation techniques must be applied in this step to the collected data set. As an example, the postprocessing step can include data-processing techniques, calculation of a statistical average (ToFaverage, mean or median), weighted averages, and so on. It may include statistical tests for outlier rejection (among others) to improve ToF accuracy.

To cancel systematic delays in a ToF measurement, one option is to perform a calibration to estimate the bias that must be subtracted from ToFaverage. Such bias method can be carried out at a fixed distance between the two nodes. ToF is given as Equation 8.

Equation 8. ToF after bias

$$ToF_{final} = ToF_{average} - ToF_{bias}$$

When a ToFfinal value is calculated, the distance between the two wireless nodes may be estimated by dividing ToFfinal by the ideal ToF value expected for a reference distance of 1 m (Equation 9).

Equation 9. Distance estimation

$$Distance_m = \frac{ToF_{final}}{3.33ns}$$

The measurement report is created when the calculated distance estimate is ready. The measuring device reports the estimated distance and an indication that the measurement is complete.

Parent topic: ToF Multistage

Zero distance calibration (ToFbias)

The ToF calculation must account for any systematic offsets in the calculation, which may be contributed by the specific choice of receive and transmit trigger anchors in the exchanged packets, as well as radio-specific implementation, such as data path latencies, delays, and so on. This systematic offset in ToF computations may be eliminated via characterization or calibration.

For both the transmitted and received packets, the location of the anchor points in the packet that are timestamped are known. Combining this knowledge of anchor triggers to the ToF packet format and communication data rate, the theoretical time delay between a packet transmission by a device and its reception by another device for a known reference distance could be precisely computed if the data path latencies in the transceiver chain were negligible or precisely known. For ToF distance estimation, the timing resolution that must be resolved is in nanoseconds. This requires a reference physical measurement to have a precise measurement of the Radio Frequency (RF), analog and digital data path latencies, as well as other implementation delays in the control path.

For this purpose, a calibration method is implemented by placing both devices (MD and RD) at a known distance and perform average ToF measurements over a set of exchanged packets. This enables the computation of averaged ToFbias, which is calculated according to Equation 10.

Equation 10. ToF bias

$$ToF_{bias} = ToF_{calibration} - (CalibrationDistance*3.33)$$

The ToF measurement is also impacted by propagation delay variation in the RF/analog front end of the radio as a function of the Automatic Gain Control (AGC) step in use. It is recommended to perform the ToF zero-distance calibration at a known distance and gain the configuration of the radio. For a random distance measurement using ToF, the knowledge of the receiver's gain during the measurement and its relationship to the gain value used during the zero distance calibration may be used to compensate for the subsequent ToF measurements.

Parent topic: ToF Multistage

Parent topic: Narrowband ToF

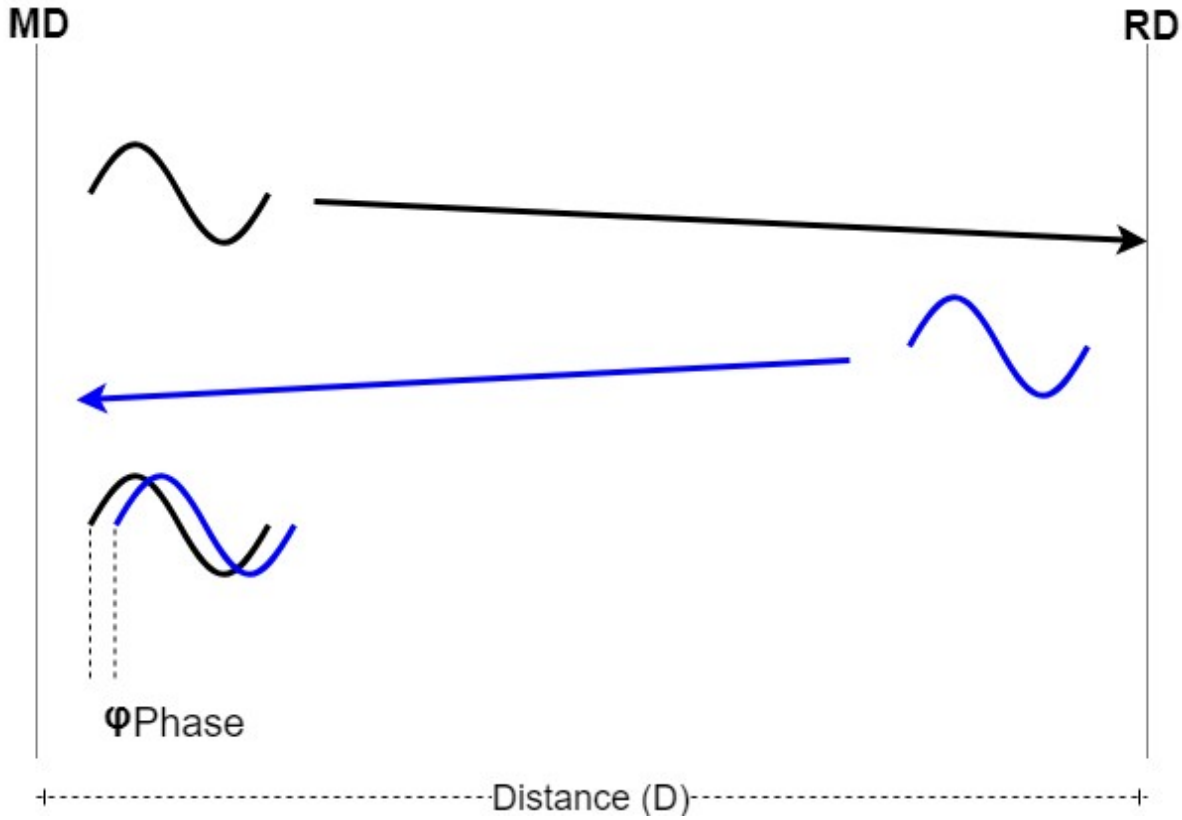
Parent topic:*RTT estimation fundamentals*

RTP estimation fundamentals

The round-trip phase (previously known as PDE) is a technique that utilizes the phase difference between an MD (which is the initiator) and determines the need to estimate distance and an active RD to determine the distance between them.

In its simplest form, the MD transmits a continuous wave to the RD, which synchronizes its phase and sends the signal back to the MD. The MD then compares the phase of the received signal with the phase of the signal it transmitted, obtaining a phase difference.

Phase difference using an MD and an RD



With this information, the distance can be measured using the following equation:

Equation 11. Measure distance using a single-phase difference

$$d = \frac{c}{2f} \cdot \left(\frac{\varphi}{2\pi} + n\right)$$

Where:

- φ is the phase difference, as measured by MD.
- c is the speed of light.
- f is the carrier frequency.
- n is the number of wraps.

Note that the number of wraps (n) depends on the wavelength of the carrier frequency.

Equation 12. Wavelength of the carrier frequency

$$\lambda = \frac{c}{f}$$

If the distance to measure is larger than λ, the phase wraps (creating a distance ambiguity) and the number of wraps (n) must be accounted for.

A sweep of multiple frequencies is used to reduce the above-mentioned ambiguity and to aid in analyzing multipath scenarios. If the phase difference is obtained at two different frequencies and the equations are combined, the following equation can be used to determine the distance:

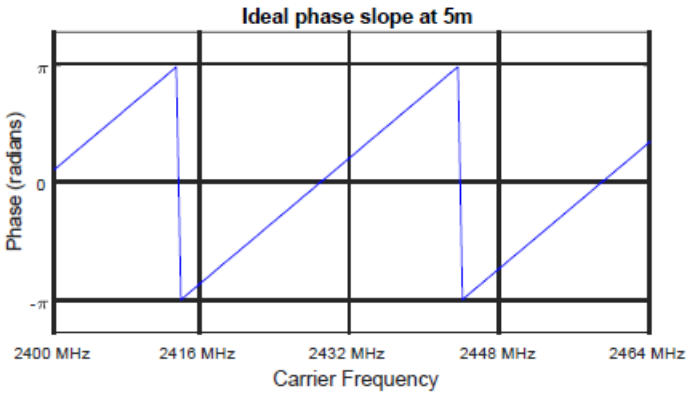
Equation 13. Distance estimation using two frequencies

$$d = \frac{c}{4\pi} \cdot \frac{\varphi_2 - \varphi_1}{f_2 - f_1}$$

Slope-based PDE

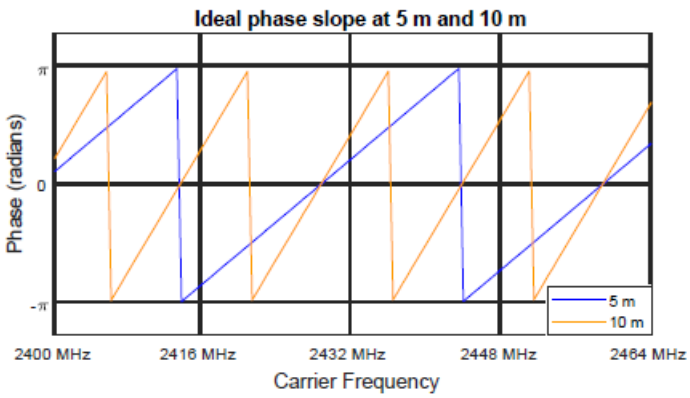
If a frequency sweep is done at a fixed distance and with constant spacing between the frequencies (Δf), the phases obtained at each frequency in the sweep produce a slope.

Phase slope at 5 m



There is a linear relationship between the phase slope observed and the distance to the object. The smaller the slope, the shorter the distance between RD and MD.

Phase slope difference at 5 m and 10 m



Using this slope, the equivalent distance can be obtained using the following equation:

Equation 14. Distance obtained from phase slope

$$d = \frac{c}{4\pi} \cdot slope$$

To get the slope-based distance using phase differences obtained at several frequencies, Equation 13 must be applied multiple times, as shown in the following equation:

Equation 15. Immediate distance from a single-phase difference

$$d_n = \frac{c}{4\pi} \cdot \left(\frac{\varphi_n - \varphi_{n-1}}{f_n - f_{n-1}} \right)$$

The final distance estimate can be obtained as follows:

Equation 16. Distance obtained from the average of immediate distances

$$d_{slope} = \frac{1}{n-1} \sum_{k=1}^{n-1} d_k$$

The maximum measurable distance depends on the spacing between the measurement frequencies (Δf) and it is given by the following equation:

Equation 17. Maximum distance measurable using the slope method

$$d_{max} = \frac{c}{4} \cdot \left(\frac{1}{\Delta f} \right)$$

Parent topic:*RTT estimation fundamentals*

Complex Distance Estimation (CDE)

In CDE, the phase measurements taken at each frequency are converted into a complex signal in the frequency domain. By transforming it to the time domain using the Inverse Fast Fourier Transform (IFFT), we obtain the distribution of the propagation delays that are in the signal.

The maximum propagation delay that can be measured without ambiguity is determined by the spacing between the measurement frequencies (Δf) and it is given by the following equation:

Equation 18. Largest unambiguous propagation delay measurable using the CDE method

$$t_{max} = \frac{1}{2 \cdot \Delta f}$$

The maximum measurable distance can be obtained by multiplying the maximum propagation speed by the speed of light (c):

Equation 19. Maximum distance measurable using the CDE method

$$d_{max} = t_{max} \cdot c$$

The distance estimate is obtained using the following equation:

Equation 20. CDE distance calculation

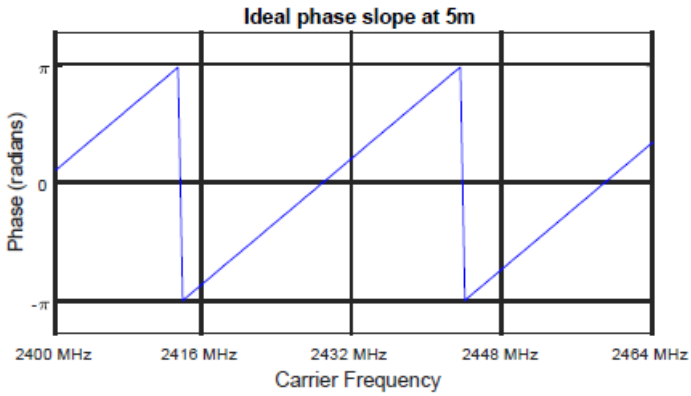
$$d_{cde} = d_{max} \cdot \frac{n_{peak} - 1}{M}$$

Where npeakis the bin with the highest peak, M is the number of bins used in the IFFT, and dmax is the largest unambiguous distance. Let's assume an ideal phase difference response for two boards separated by 5 m. The phase difference is measured using Δf = 500 kHz. The maximum measurable distance is given by the following equation:

Equation 21. Maximum distance measurable using Δf = 500 kHz

$$d_{max} = \frac{c}{2 \cdot \Delta f} = \frac{299792458 \frac{m}{s}}{2 \cdot 500000_{Hz}} = 299.7925_{m}$$

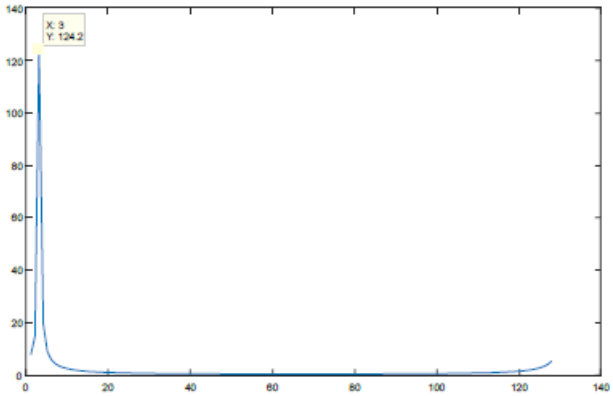
The wrapped phase response obtained from the frequency sweep at the same distance (see Figure 20) is shown in Figure 22.



Ideal phase slope at 5 m using Δf = 500 kHz

If a 128-bin FFT is applied to an array containing the wrapped phase points in their rectangular form, the output is shown in Figure 23.

Ideal 128-bin IFFT for the 5-m slope



The bin holding the biggest peak is number 3. Use the following equation to obtain the estimated distance:

Distance calculation for bin 3 using 128-bin FFTe

$$d_{cde} = 299.7925_m \cdot \frac{3-1}{128} = 4.6842_m$$

The resolution of the measured distance depends on the frequency step selected and the number of bins used in the IFFT.

Parent topic:*RTP estimation fundamentals*

PDE implementation

A two-way PDE estimation algorithm typically requires precise acquisition of phase changes across a span of frequencies by each participating device. The phases are derived from the captured I and Q samples in the receiver of a device for a tone transmitted by the peer device with a known raster.

The PDE implementation mainly consists of four steps: initialization and calibration, IQ data capture, phase processing, and distance estimation.

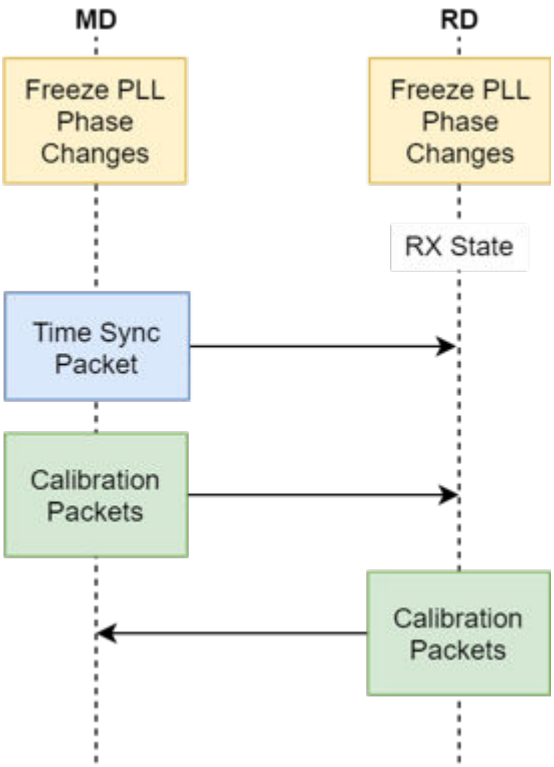
Initialization and calibration

Before capturing the IQ samples, four important conditions must be met:

1. All transceiver impairments that can impact the phase capture information (DC offsets, IQ mismatch, filtering transfer function contributions, PLL phase excursions, and so on) must be minimized.
2. Theoretically, the IQ signal amplitude does not affect the phase calculations. However, signal saturation or fading can affect the accuracy of phase calculations. Ensure a good dynamic range for the captured IQ samples.
3. The measurement of phase differences implies that changes in phase are only produced by the distance between the test devices. The PLLs in both devices must remain frequency-coherent throughout the capture process.
4. Some algorithms require changing the tone frequencies with a specific time interval. The devices must synchronize their transmit and receive cycles accordingly.

The initialization and calibration stages ensure that all of the above conditions are met before capturing the IQ samples.

Initialization and calibration sequence



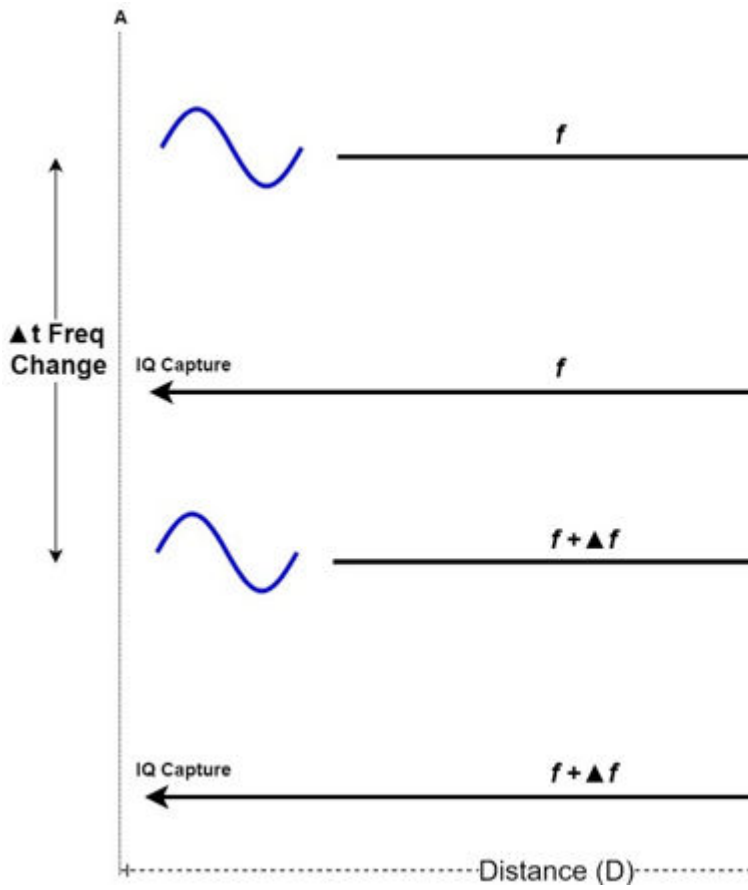
Parent topic:PDE

IQ data capture

Two-way PDE algorithms use multiple phase differences obtained at several frequencies. To achieve this, the MD and RD must generate Continuous-Wave (CW) tones and capture IQ samples in all the measurement frequencies.

The base mechanism consists of device A, which generates a tone at frequency f. At the same time, the peer device B sets its radio to the RX mode at f and starts to capture IQ samples. After a fixed amount of time, devices A and B change roles. Now device B generates a tone in f and device A starts the RX and IQ capture at f. This is a tone exchange sequence.

After the tone exchange sequence ends, both devices return to their original roles, but the frequency is updated by Δf. For this new tone exchange, device A generates a tone at f + Δf. Device B also sets its radio to the RX mode at f + Δf and captures IQ samples. The tone exchange sequence is repeated for all frequency values that are defined for this measurement



Multiple frequency IQ samples capture sequence

Parent topic:PDE

Phase processing

After the IQ samples are captured on each device, the phase difference is obtained using the following equation:

Equation 22. Phase from IQ samples

$$\varphi = arctan2\left(\frac{Q}{I}\right)$$

Each device holds the phase difference slope of the peer device against its own local oscillator. To obtain the phase combination result of the distance, the phase slope from the RD must be added to the phase slope from the MD.

Equation 23. Phase vector combination

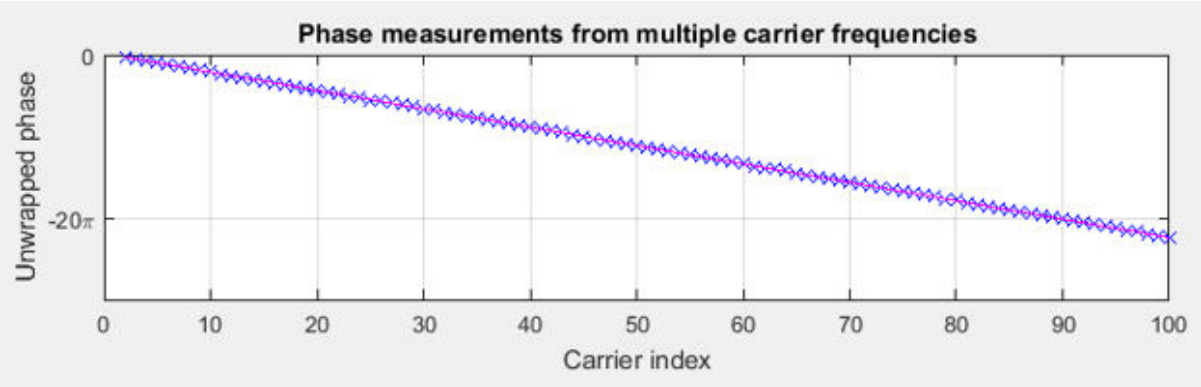
$$\Delta\varphi = \varphi_{MD} + \varphi_{RD}$$

This phase slope can now be used to perform distance estimation. As shown in Figure 21, depending on the frequencies used and the measured distance, phase wraps may occur.

Figure 27 shows how the measured phase values change as a function of the carrier-frequency

index. The data for this example is obtained using a wired setup, where the two devices are connected by a cable. A total of 99 carriers are used with the first carrier (index 2) corresponding to a frequency of 2400.5 MHz. The last carrier (with index 100) corresponds to a frequency of 2449.5 MHz. The frequency step size is 500 kHz.

Phase measurements (unwrapped) across multiple carrier frequencies

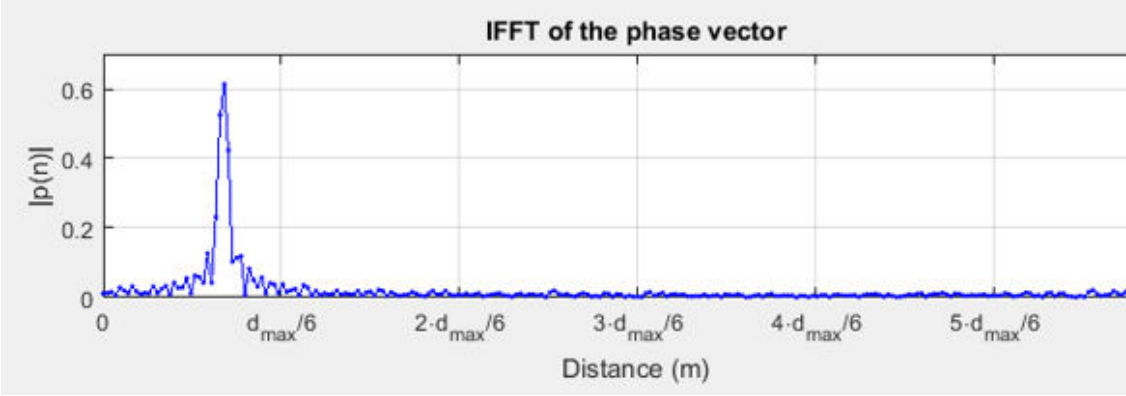


Parent topic:PDE

Distance estimation

After the IQ samples are captured on each device, the phase difference is obtained using the following equation:

The phase vector obtained as a result of phase processing is the input for the distance-estimation algorithm. It can be used directly with Equation 16 for the slope method or as an IFFT input for the CDE method.



IFFT of phase vector

Figure 28 shows the IFFT magnitude response of the phase vector illustrated in Figure 27. The labels on the x-axis show the distance in meters. The final estimate is generally obtained from the peak location of the IFFT. The zero-meter compensation may be necessary to account for the delays within the radio front end.

Parent topic:PDE

Parent topic:*RTP estimation fundamentals*

Test setup description

The basic test setup description is based on information provided in the following complementary application notes:

- Channel Sounding Wireless Ranging Demo Application
- KW47 Wireless Ranging Performance Measurement. The individual test setup is not part of this document.

The Channel Sounding Wireless Ranging Demo Application document contains simple out-of-box test setup description with all the important settings. The KW47 Wireless Ranging Performance Measurement is a validation report which uses more complex HW test setup to achieve precise and statistically evaluated results.

Conclusion

This application note describes the channel sounding standard and shows the step modes for distance estimation. The theoretical background of the RTT (ToF) and RTP (PDE) methods are also described.

Acronyms

The following acronyms are used in this document.

Acro	Description
CS	Channel Sounding
PDE	Phase-based Distance Estimation
RTP	Round-Trip Phase
RTT	Rount-Trip Time
ToF	Time of Flight
MD	Measuring Device
RD	Reflector Device
EDL(	Early Detect Late Commit
SFD	Start-of-Frame Delimiter
T_FC	The RF circuitry locks to the measurement channel (f_i). Devices can also use this time for additional internal calibrations, if necessary. The initiator may start to ramp up its output signal toward the end of this step.
T_FM	Time for Frequency Measurement
T_SY	Time for Synchronization sequence (CS packet)
T_IP1	Interlude Period 1 - transition from TX to RX and the other way around when transmitting in mode 1 (Tn-Tn)
T_IP2	Interlude Period 2 - transition from TX to RX and the other way around when transmitting in mode 2 (Tn-Tn)
T_GI	Transition time between packet and tone
T_RD	Ramp-Down time for transmission
T_PM	Single-antenna Phase Measurement period
N_AP	Number of Antenna Paths
T_SW	Antenna-Switching period
AGC	Automatic Gain Control
ADC	Analog-to-Digital Conversion
CDE	Complex Distance Estimation
IFFT	Inverse Fast Fourier Transform

Bluetooth Low Energy CCC Digital Key with Channel Sounding Introduction

NXP hardware platforms support the Bluetooth Low Energy applications implementing the Car Connectivity Consortium (CCC) Digital Key with channel sounding (CS) functionality. This document provides an overview of the deployment and operation of these applications, which also support the Bluetooth LE Channel Sounding localization functionality. This document is also a user guide for the sample applications included in the SDK release package.

Audience

The document also provides an overview of the procedure and steps to deploy and operate the Bluetooth Low Energy applications (enabling CCC Digital Key with CS). The NXP hardware platforms supporting these applications are the KW47-EVK, KW47-LOC, KW43-LOC and FRDM-KW43 boards.

This document is a useful reference for firmware and system developers who create CCC Digital Key-enabled products. The document also provides a high-level description of CCC Digital Key application scenarios that can be deployed on NXP development boards.

CCC Digital Key technology overview

The Car Connectivity Consortium (CCC) is a cross-industry organization advancing global technologies for smartphone-to-car connectivity solutions. CCC has developed Digital Key, a new open standard to allow smart devices such as smartphones to act as a vehicle key. Digital Key lets drivers lock and unlock their cars using their phone. It also allows them to start the engine and share access to friends or valets, using their phones.

The Digital Key Release 3.0 specification enhances Digital Key Release 2.0 by adding passive, location-aware keyless access. This feature allows vehicle users to access and/or start their vehicle while leaving their mobile device in their bag or pocket. They do not need to pull their mobile devices out to access the car. Passive access is not only vastly more convenient and a better overall user experience, it also allows vehicles to offer new location-aware features.

To support these new features, the CCC has developed a specification based on Bluetooth Low Energy with Ultra-Wideband (UWB). The specification enables passive keyless access and allows secure and accurate positioning.

CCC Digital Key applications overview

Note: The applications described below implement only the Bluetooth Low Energy functionality described by CCC Digital Key R3. They do not include any UWB or Secure Element functionality. The establishment of secure communication between the demos is simulated using dummy messages.

Digital Key Car Anchor application

The Digital Key Car Anchor demo application implements one of the multiple Bluetooth Low Energy Anchors that can reside inside the car. It is a Bluetooth Low Energy peripheral. Depending on the scenario, this peripheral can perform both Legacy (1M PHY) and Extended Long Range (500 kilobits per sec Coded PHY) advertising, searching for a CCC Digital Key-enabled device such as a smartphone.

There can be multiple Anchors inside a car. The Anchors must act as a single device and share information between them such as addresses, bonding data, or application-specific keys. The information must be shared such that a smartphone can connect to any one of the Anchors, depending on positioning and not detect any difference.

For the purposes of localization, the Digital Key Car Anchor acts either as a Channel Sounding Initiator or Reflector and as a Server/Client in the context of the Bluetooth LE Channel Sounding (BTCS). The Digital Key Car Anchor runs the localization algorithm using the local and remote distance measurement data.

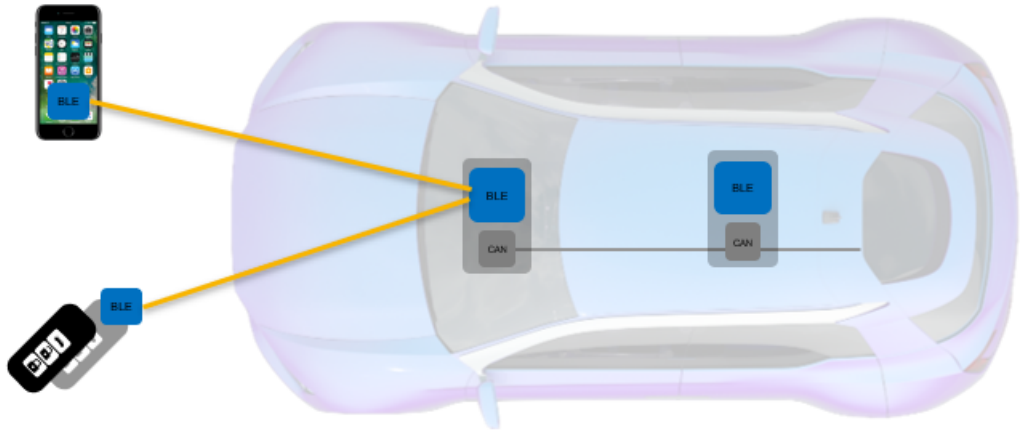
Parent topic:*CCC Digital Key applications overview*

Digital Key Device application

The Digital Key Device demo application emulates a CCC Digital Key-enabled smartphone. It acts as a Bluetooth Low Energy central device, scanning for advertising coming from a Digital Key Car Anchor.

The figure below shows an example deployment of CCC Digital Key R3.

Anchor communicating with a smartphone (CCC) and a key fob (non-CCC)



Inside the car, there are two Bluetooth Low Energy Anchors connected to each other through Controller Area Network (CAN) bus. These Anchors are used to exchange security data and other information. An Anchor can connect to a CCC Digital Key R3-enabled smartphone and to a non-CCC key fob and can perform both advertising and scanning.

For the purposes of localization, the Digital Key Device acts as either a Channel Sounding Initiator or Reflector and as a Client/Server in the context of the BTCS. The BTCS Server stores the local measurement data and provides it to the BTCS Client.

Parent topic:*CCC Digital Key applications overview*

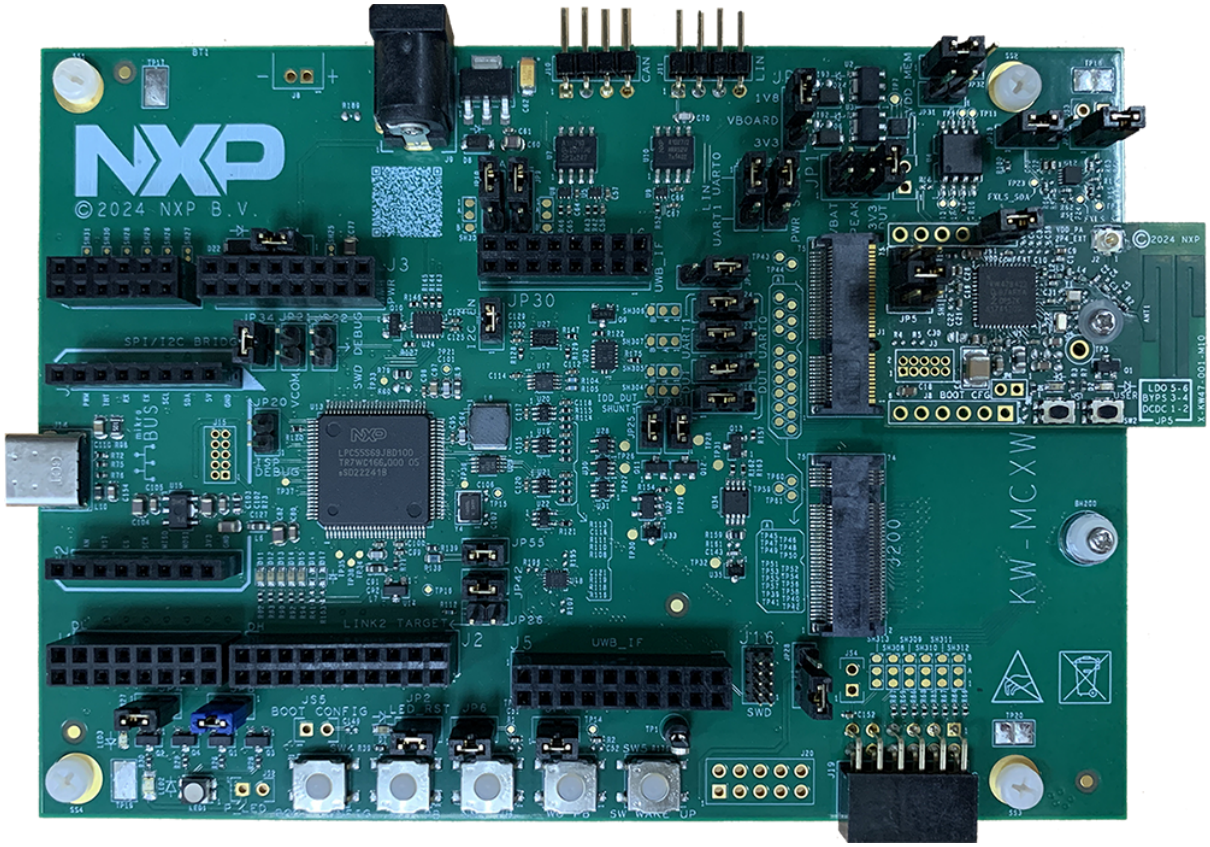
CCC Digital Key hardware platforms

The CCC Digital Key R3 demo applications with localization functionality support the following platforms that have Bluetooth Low Energy transceiver capabilities:

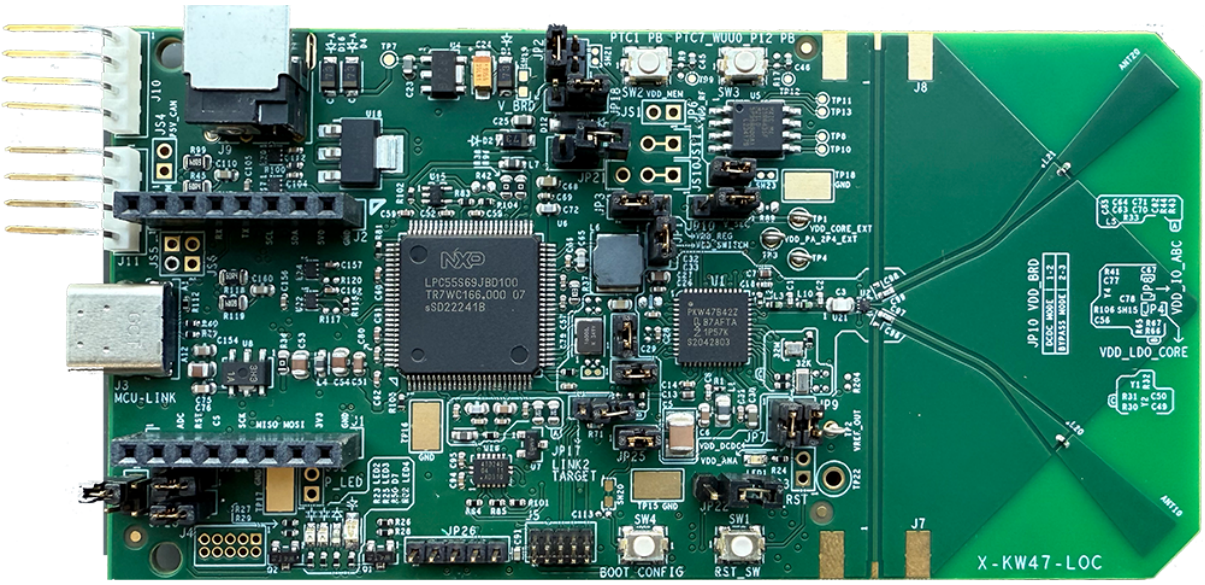
- KW47-EVK
- KW47-LOC board
- KW43-LOC board
- FRDM-KW43

The below figures show these boards.

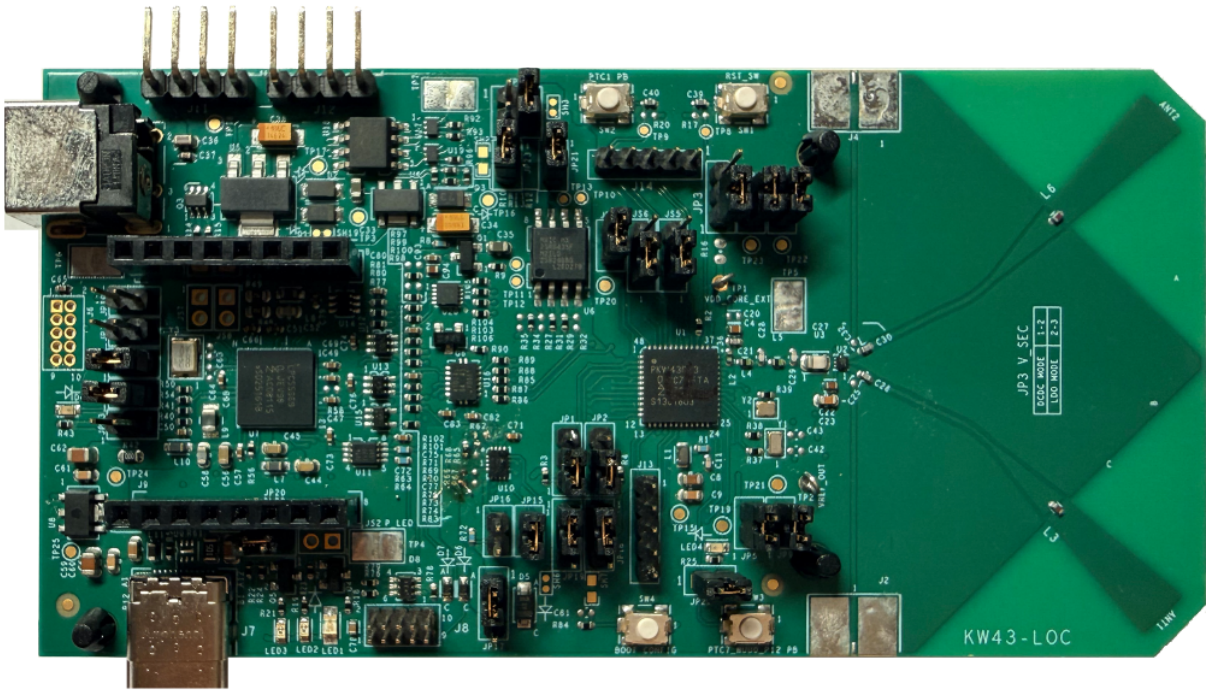
KW47-EVK board



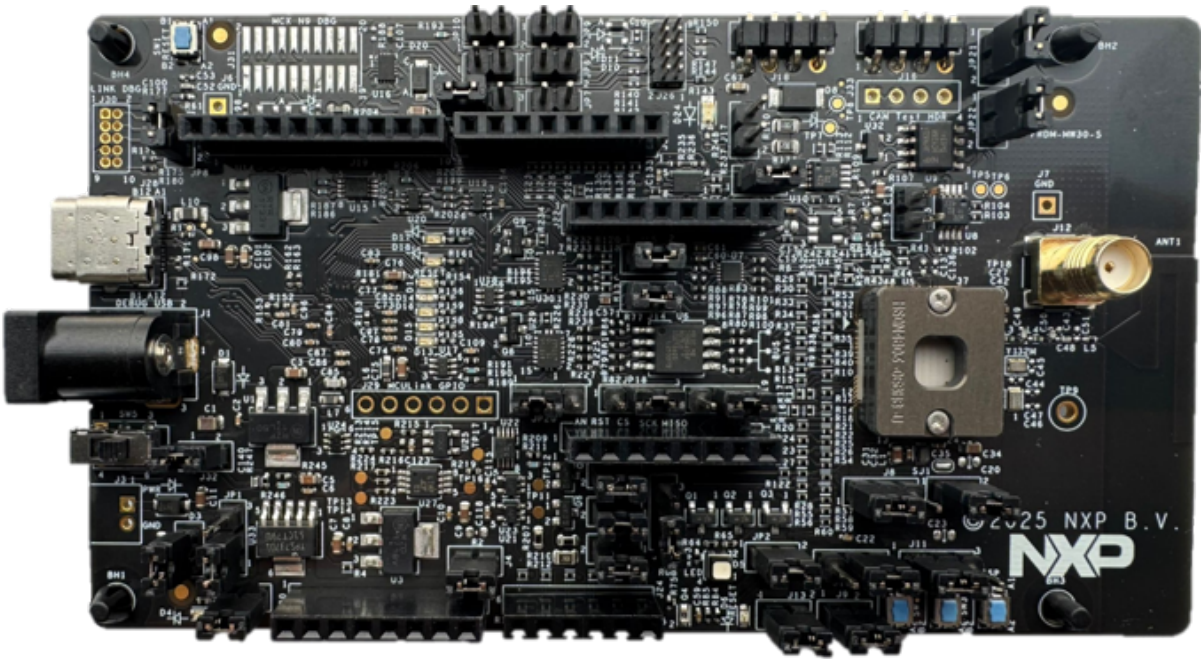
KW47-LOC board



KW43-LOC board



FRDM-KW43 board



Deploying CCC Digital Key application software

The software package includes the components necessary for CCC Digital Key R3 application development on the selected NXP platform. These components include:

- Bluetooth Low Energy Host software libraries
- Channel Sounding and Localization algorithm libraries
- NBU images binaries
- Examples of demo application projects corresponding to CCC Digital Key roles (Device and Car Anchor)
- Platform peripheral drivers, platform startup code, other generic platform software

Note: Prior to loading any wireless SDK example, update your NBU image with the provided binaries, with Channel Sounding (CS) support, in the following folder of the SDK:

../middleware/wireless/ble-controller/bin.

The demo applications were compiled and tested with **IAR Embedded Workbench for Arm** and **ARMGCC**. Users are recommended to use one of these tools.

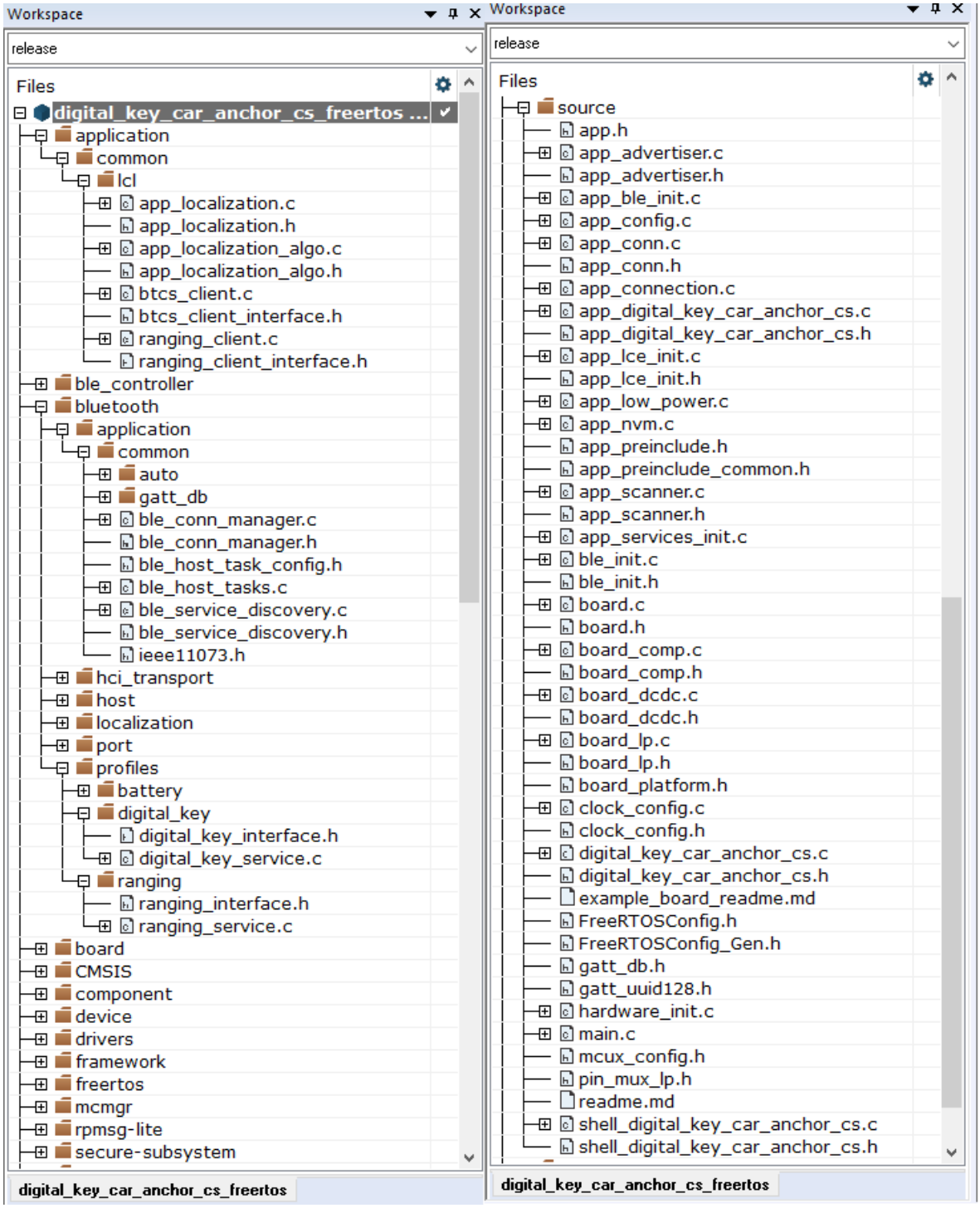
To open, build, and run any example application, see the Bluetooth Low Energy Quick Start Guide document of the corresponding board. The CCC Digital Key R3 examples are built and run in the same fashion as Bluetooth Low Energy examples.

The application folder structure for the *digital_key_car_anchor_cs* project can be described as the following:

- The main application source files are the source/digital_key_car_anchor_cs.c and source/app_digital_key_car_anchor_cs.c.
- The files in bluetooth/profiles/digital_key implement the Digital Key Service defined by the CCC Digital Key R3 specification.
- The application files for the localization functionality, are /application/common/lcl/app_localization.c and application/common/lcl/app_localization_algo.c.
- The files /application/common/lcl/btcs_server_interface.h and /application/common/lcl/btcs_server.c implement the BTCS Server functionality.
- The files /application/common/lcl/btcs_client_interface.h and /application/common/lcl/btcs_client.c implement the BTCS Client functionality.

The below figure shows the application folder structure for the *digital_key_car_anchor_cs* project.

Application folder structure in workspace



Demo functionality overview

The two demo applications provided showcase the following Digital Key R3 features:

- Owner Pairing between a Device and a Car Anchor
- Passive Entry
- Synchronizing bonding data between Car Anchors
- Connection between a Car Anchor and a non-CCC key fob

The two demo applications showcase the following localization features:

- Distance measurement through Channel Sounding

- Distance measurement data transfer through L2CAP
- Ranging algorithms
  - Complex-domain Distance Estimation (CDE)
  - Rapid Accurate Distance Estimator (RADE)

The demo applications showcase the NXP proprietary Connection Handover and Anchor Monitoring features, and the A2B feature (secured synchronization of bonding data).

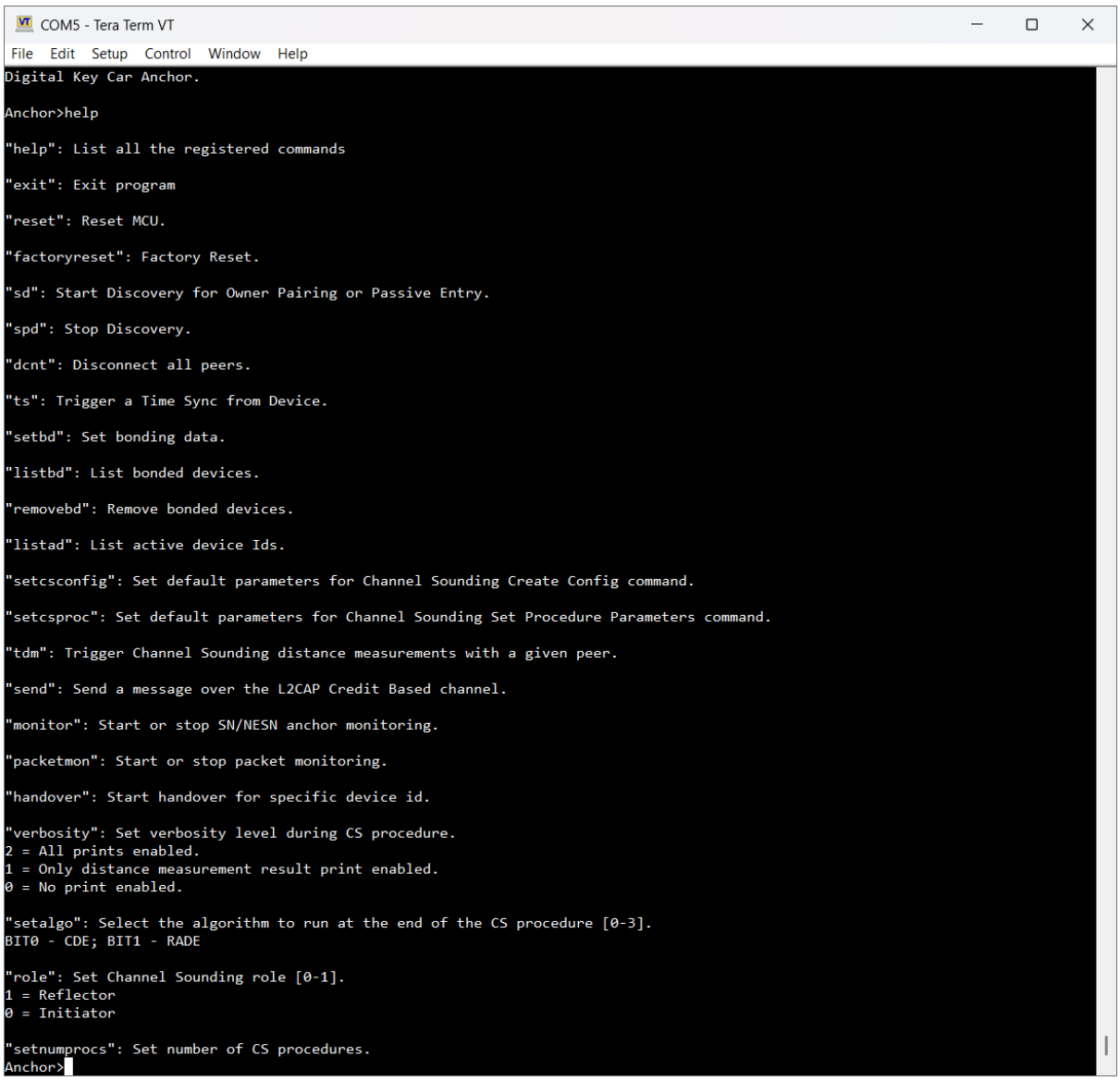
Running CCC Digital Key scenarios using the shell interface

The following CCC Digital Key R3 examples are provisioned by default with a shell command line interface accessible via a Terminal application such as Tera Term:

- Digital Key Car Anchor
- Digital Key Device

After you connect through the Terminal application and press the **RST** switch on the board, a welcome message appears. Enter the `help` command to display the list of available commands and their descriptions. See the figure below.

List of commands

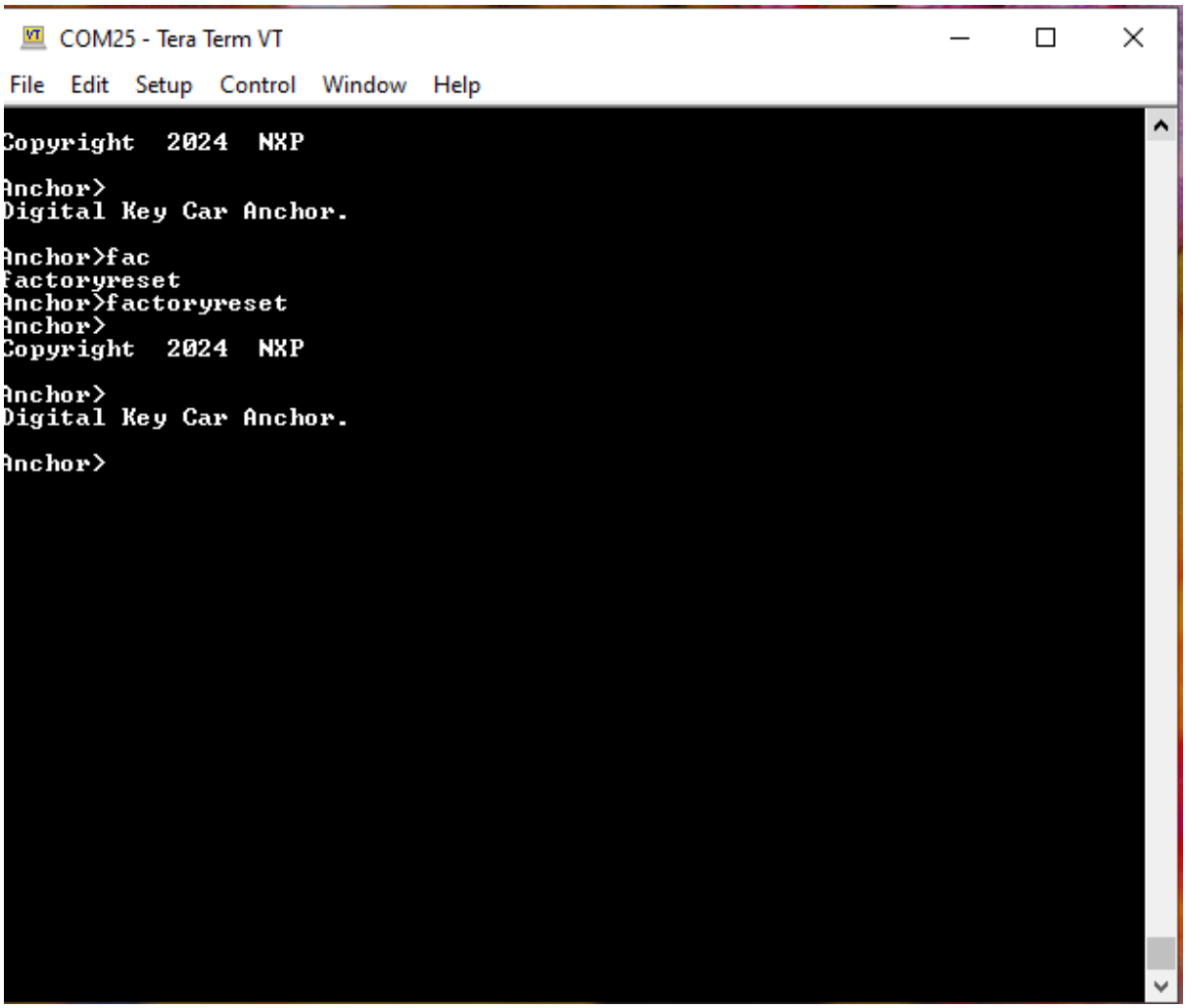


Owner Pairing scenario

Owner Pairing establishes a bond between the Digital Key Car Anchor and the Digital Key Device as per the CCC Digital Key R3 specification. The Car Anchor starts advertising using Legacy mode on the 1M PHY. The Device scans and connects. After establishing the connection, the Device performs service discovery. This step enables it to discover the DK Service and learn the PSM value it uses to open an L2CAP channel to the Car Anchor. This channel is used to exchange data and security information as part of the Bluetooth Low Energy Out-of-Band (OOB) pairing process. All security information exchanged consists of dummy messages. The applications do not currently implement any of the UWB and Secure Element functionality described by the CCC Digital Key R3 specifications (See *References*).

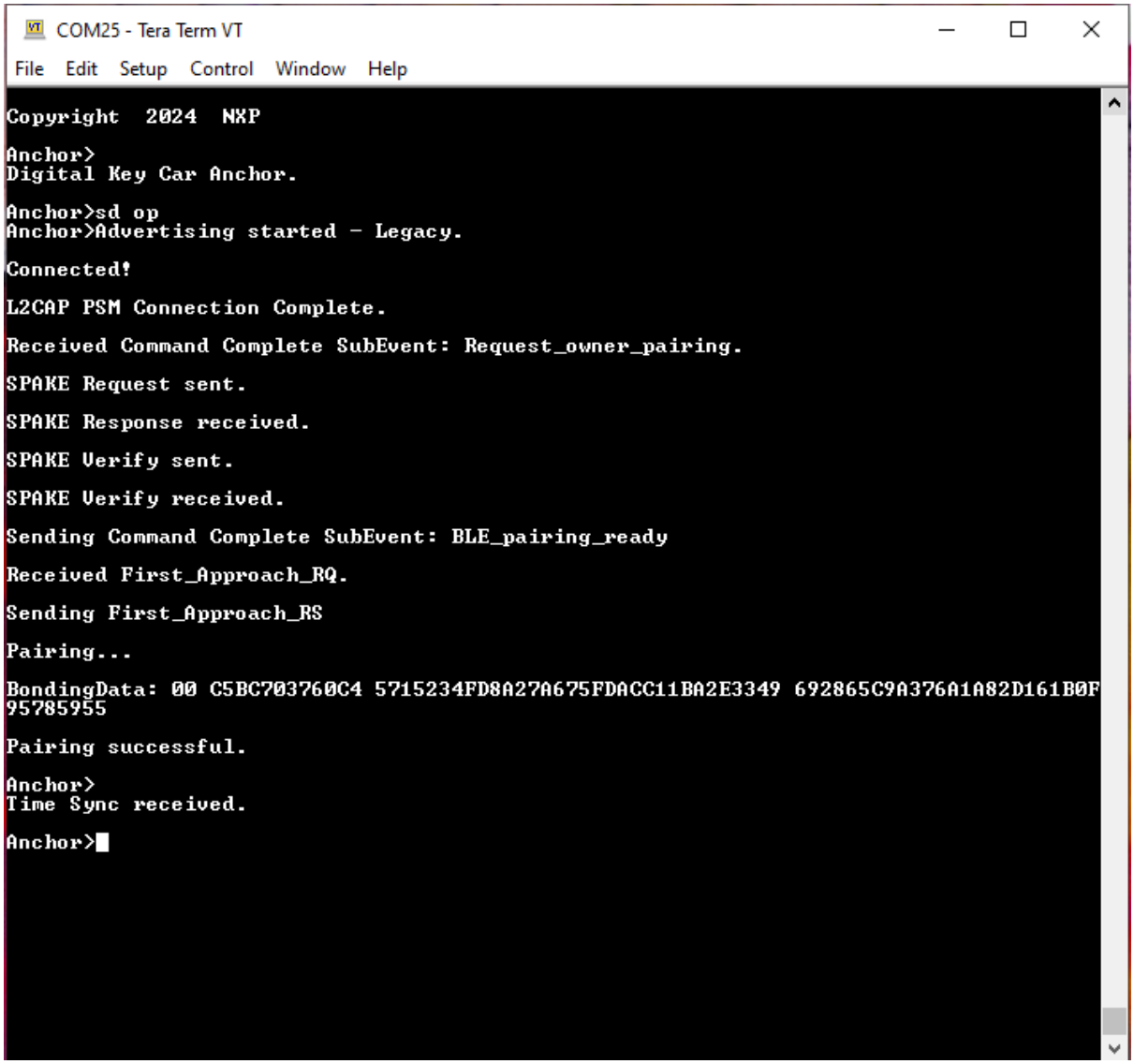
The first step is to run the "factoryreset" command on both boards in order to ensure no previous bonding data is present in non-volatile memory. Refer to the figure below.

Factory reset on Car Anchor

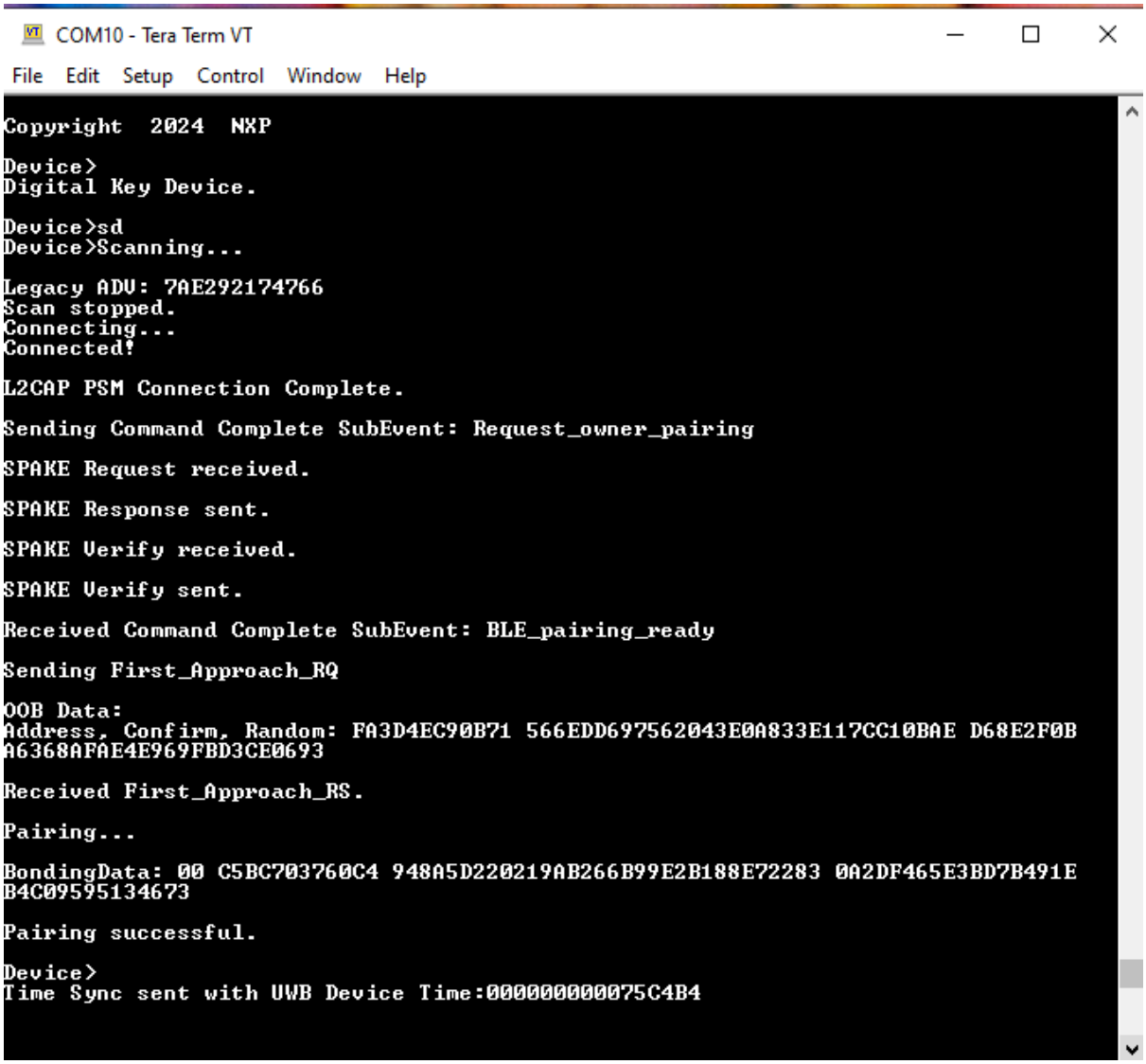


The next step is to run the "sd" command (Start Discovery) on the Device and the "sd op" command on the Car Anchor (Owner Pairing advertising differs from Passive Entry, using a single Legacy advertising set on the 1M PHY). The Car Anchor starts advertising while the Device scans. Since both boards have been factory reset, they follow the Owner Pairing flow. The command outputs are shown in the below figures.

Owner Pairing on Car Anchor



Owner Pairing on Device



Parent topic:*Running CCC Digital Key scenarios using the shell interface*

Passive Entry scenario

The Passive Entry scenario runs after owner pairing between a Car Anchor and a Device has successfully completed. First, reset the recently paired boards using either the RST switch or the "reset" command in the shell. Alternatively, disconnect using the dcnt shell command.

Note: Do not perform a factory reset as this step removes the bonding information from the non-volatile memory.

After reset or disconnect, run the "sd" command on the Device and the "sd pe" command on the Car Anchor. For the Passive Entry scenario, the Car Anchor advertises using two sets, one Legacy on the 1M PHY and one Extended on the Coded PHY. Since bonding data is present, they follow the Passive Entry flow, stopping after the successful creation of the L2CAP channel. The application is functional after this step. Similar to the Owner Pairing scenario, the applications implement only the Bluetooth Low Energy functionality.

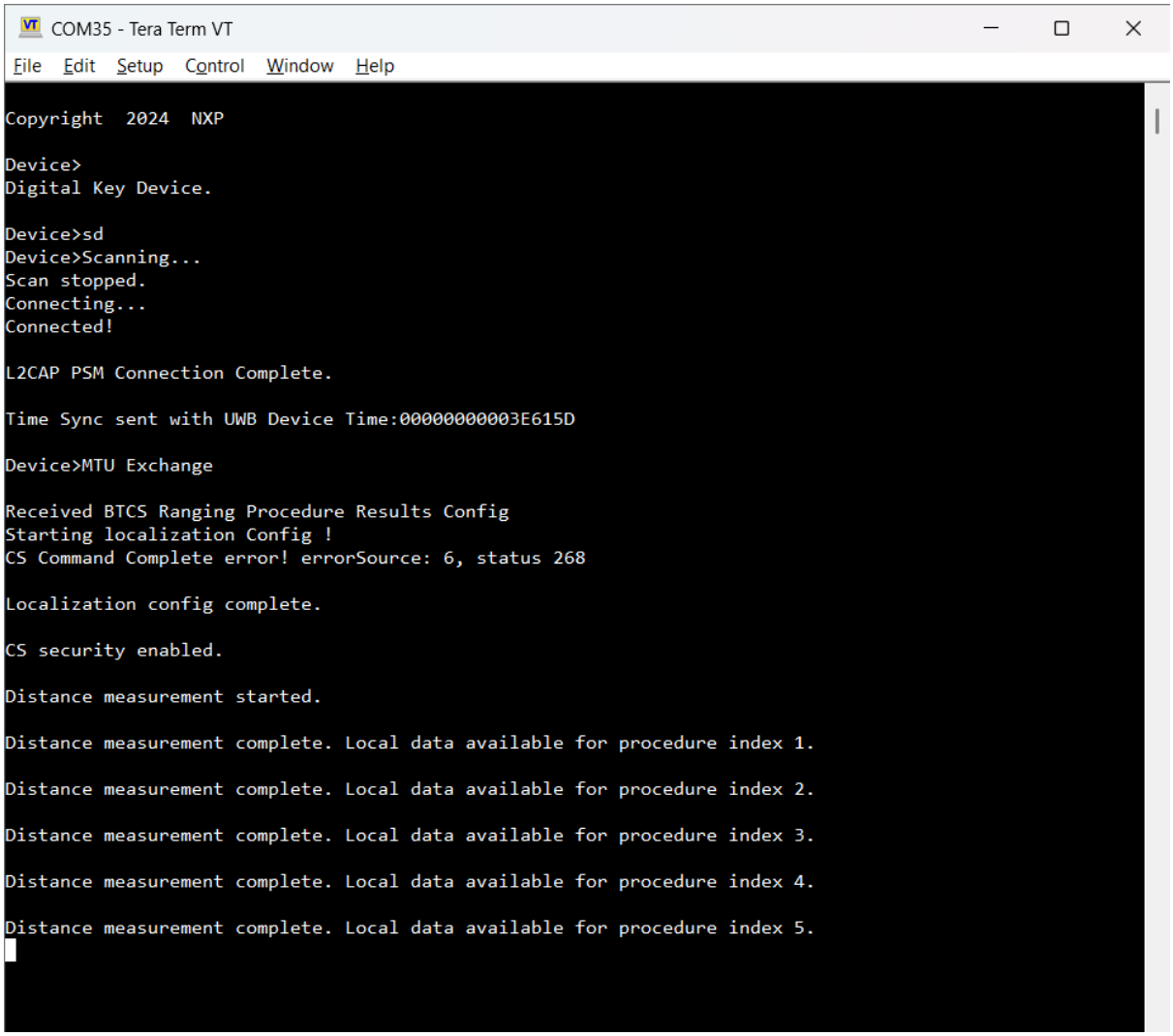
Note: The CCC specification requires using the coding scheme S=2 for Coded PHY advertising, equivalent to a data rate of 500 Kbits per second. To experiment with other coding schemes, change the advertising parameters in app_config.c for the *digital_key_car_anchor_cs* project.

The output of the sd and sd pe commands is shown in the figures below.

Passive Entry on Car Anchor

```
COM5 - Tera Term VT
File  Edit  Setup  Control  Window  Help
Copyright  2024  NXP

Anchor>
Digital Key Car Anchor.

Anchor>sd pe
Anchor>Advertising started - Legacy.
Advertising started - Extended LR.

Connected!
Advertising stopped - All PHYs.

L2CAP PSM Connection Complete.

Time Sync received.

Anchor>
Localization config complete.

CS security enabled.

Set Procedure parameters complete.

Start distance measurement.

[0] Distance measurement complete. Local data available for procedure index 1.

[0] Distance (RADE): 0.40 m   Quality: 93.0%

[0] RTT Distance: 0.0 m Success Rate: 90

[0] Distance measurement complete. Local data available for procedure index 2.

[0] Distance (RADE): 0.41 m   Quality: 93.0%

[0] RTT Distance: 0.0 m Success Rate: 100

[0] Distance measurement complete. Local data available for procedure index 3.

[0] Distance (RADE): 0.41 m   Quality: 93.0%

[0] RTT Distance: 0.0 m Success Rate: 100

[0] Distance measurement complete. Local data available for procedure index 4.

[0] Distance (RADE): 0.37 m   Quality: 91.0%

[0] RTT Distance: 0.0 m Success Rate: 100

[0] Distance measurement complete. Local data available for procedure index 5.

[0] Distance (RADE): 0.34 m   Quality: 89.0%

[0] RTT Distance: 0.0 m Success Rate: 100

Anchor>
```

Passive Entry on Device



Parent topic:*Running CCC Digital Key scenarios using the shell interface*

Synchronizing bonding data between Car Anchors

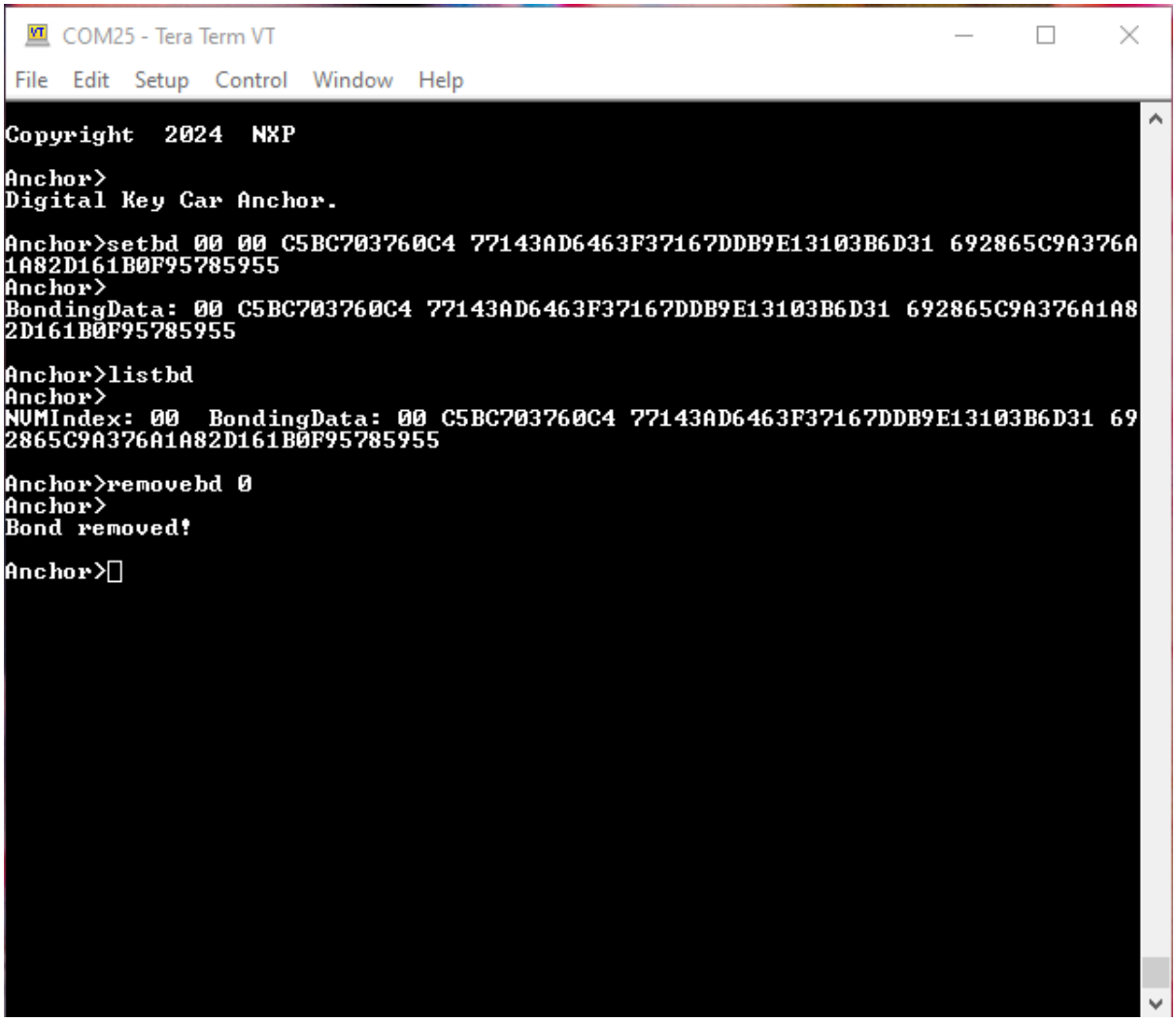
Note: The feature of Synchronizing bonding data between Car Anchors is only valid when the Advanced Secure Mode is disabled. This implies that the parameter (gAppSecureMode_d is set to 0 in the app_preinclude.h file. To synchronize bonding data when Advanced Secure Mode is enabled, refer to the section *Running the A2B scenario*.

Multiple Car Anchors can reside on a car, acting as a single Bluetooth Low Energy device as far as the Device is concerned. When the Device pairs and bonds with a Car Anchor, that bonding data must be shared with all other Anchors.

To showcase this functionality, the Car Anchor shell demo offers the "setbd" command. During owner pairing, the bonding data is displayed in the shell as seen in the Owner Pairing section. By passing this data to the "setbd" command on another Car Anchor, the Anchor is able to connect with the original Device and perform the Passive Entry flow. Currently only the Bluetooth Low Energy bonding data is transferred between Anchors. No CCC Digital Key R3-specific keys are exchanged. Saved bonding data can also be viewed using the "listbd" command and a specific bond can be removed using the "removebd" command. A bond can only be removed if a connection to that specific device is not currently active. The figure below also shows listing bonds and removing a bond.

The commands presented in this subsection are also supported on the Device for test purposes.

Adding bonding data to a Car Anchor



Note:

- To simulate Anchors residing on the same car, the Random Static Address required by the Digital Key protocol and the Identity Resolving Key are set at compile time for the *digital_key_car_anchor_cs* project. To simulate Anchors residing on different cars, change the values of the APP_BD_ADDR and APP_SMP_IRK macros in the *app_preinclude.h* file of the project.

Parent topic:*Running CCC Digital Key scenarios using the shell interface*

Running the Connection Handover scenario

Connection Handover is an NXP proprietary feature that enables seamless transfer of a Bluetooth Low Energy connection from one peripheral to another while the central device remains unaware. The CCC Digital Key use case best exemplifies this scenario but is not limited to it. As a person carrying a phone moves around the vehicle, the Bluetooth Low Energy connection is transferred between Car Anchors to ensure the best user experience. From the point of view of the Device, it stays in the same initial connection.

The CCC Digital Key demos showcase the Connection Handover feature. The handover is performed on demand via a button press. In a real scenario, the handover is performed based on criteria such as RSSI. The following hardware platforms support this feature:

- KW47-EVK
- KW47-LOC board
- KW43-LOC board

- FRDM-KW43

Prerequisites:

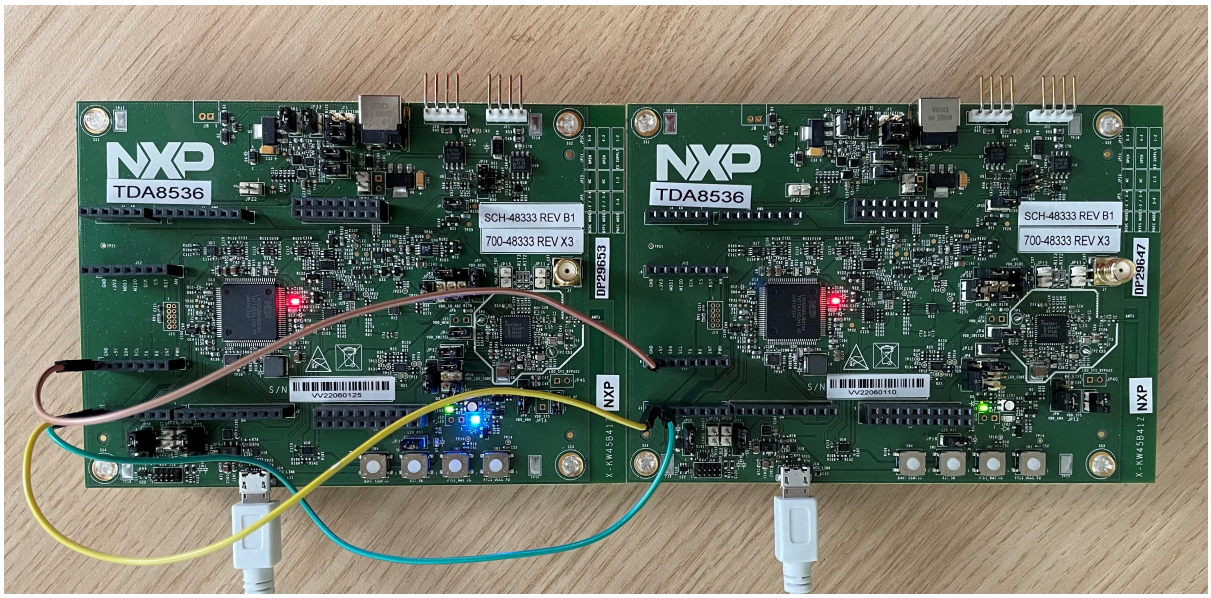
- Three boards (two boards act as Car Anchors, one as a Device). The demo is limited to a maximum of two Car Anchors.
- The two Car Anchors must have a serial connection via the secondary UART. The UART connection stands in for a real deployment solution such as a CAN bus. The table below describes the connections for each supported board.

Board	UART RX TX	UART TX RX	GND
KW47-EVK	J1-1 J1-3	J1-3 J1-1	J13-8 J13-8
KW47-LOC	J2-3 J2-4	J2-4 J2-3	J2-8 J2-8
KW43-LOC	JP19-3 (UART0_RX PTA18) JS6-3 (UART0_TX PTA17)	JS6-3 JP19-3	J9-8 J9-8
FRDM-KW43	J22-1 (UART0_RX PTA18) JP19-1 (UART0_TX PTA17)	JP19-1 J22-1	J21-8 J21-8

Note: For KW43-LOC and FRDM-KW43, the current firmware configuration uses PTA18 for UART0_RX instead of PTA16 as shown in the schematic.

- The *gHandoverIncluded_d* macro must be set to 1 in app_preinclude.h for the ***digital_key_car_anchor_cs*** project.
- The CS configuration must exist on the initial Car Anchor before the first handover is performed, as detailed in the Demo steps below.

Two Car Anchors connected via the secondary UART



Demo steps:

- Connect a Car Anchor and a Device using the Passive Entry scenario as described in the section Passive Entry scenario.
- Press the **SW3** switch on the Car Anchor which is connected to the Device.
- The connection is handed over to the second Car Anchor as seen in console messages. LED2 also turns solid blue on the Car Anchor that takes over the connection.

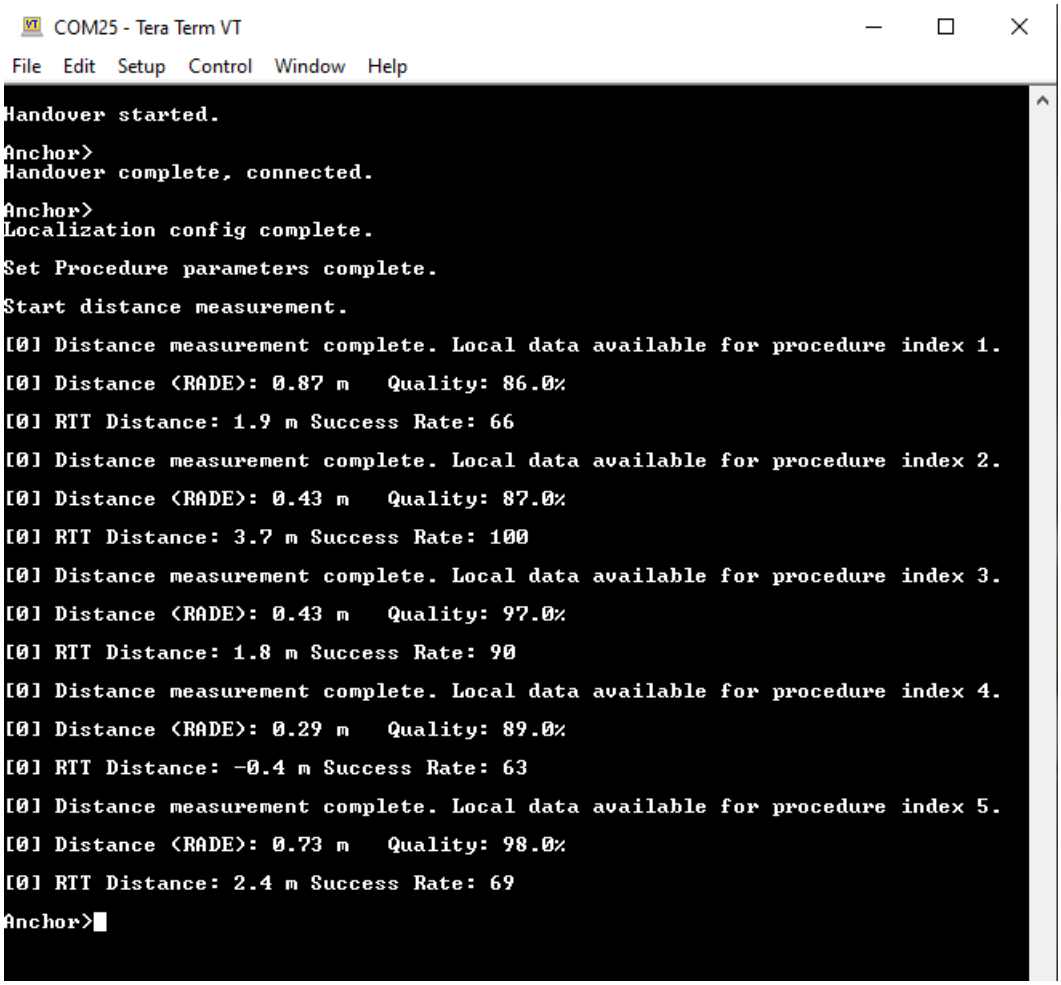
- **Optional**: Use the "send" shell command on the Car Anchor that has taken over the connection. The message is displayed in the console on the Device, after the Passive Entry scenario has completed and CS procedures have run.
- The connection can also be continually handed over back and forth between the two Car Anchors. To perform this step, press the **SW3** switch on the Car Anchor that currently has the connection.

As shown in the below figure, the first Car Anchor performs the Passive Entry scenario, followed by the CS procedures resulting in distance measurements. The handover is then triggered via button press.

First Car Anchor performs the Passive Entry scenario and distance measurements

The figure below shows that the second Car Anchor takes over the connection. The default role for the Car Anchor is CS Initiator. Therefore, it automatically triggers new CS procedures resulting in distance measurements. The "tdm" shell command can be used to trigger new distance measurements from either the Device or the Car Anchor.

Second Car Anchor takes over the connection



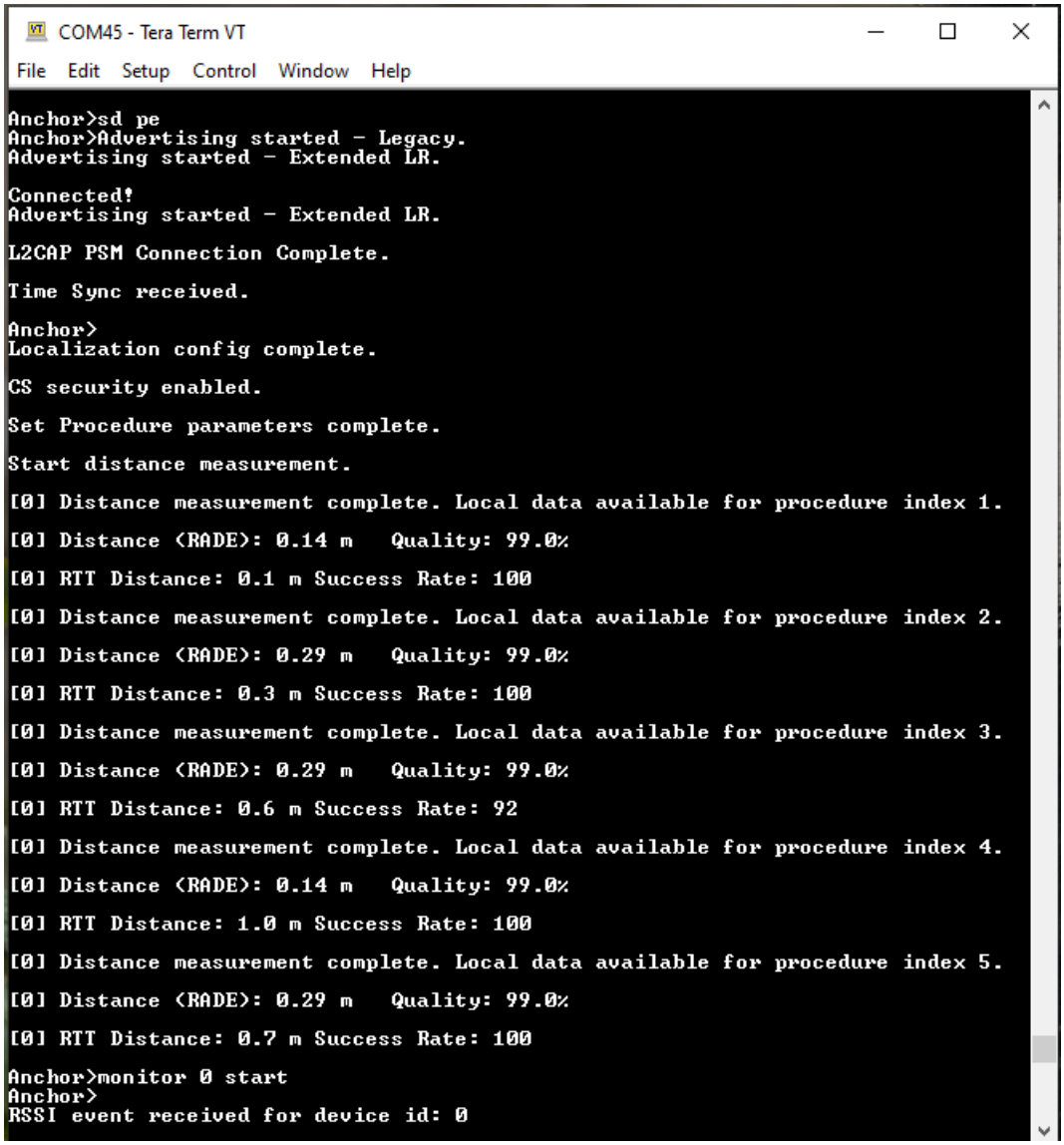
Running the Anchor Monitoring scenario

Anchor Monitoring is an NXP proprietary feature that allows a second Anchor to monitor the connection between the first Anchor and the Device. The feature requires the same hardware setup as used for the Connection Handover scenario. Note, however, that the Connection Handover scenario cannot be performed while Anchor Monitoring is in progress.

Demo steps:

1. Connect a Car Anchor and a Device using either the Owner Pairing or Passive Entry scenario as described in Owner Pairing scenario and Passive Entry scenario.

   Anchor has performed Owner Pairing and the `monitor start` **command is run**

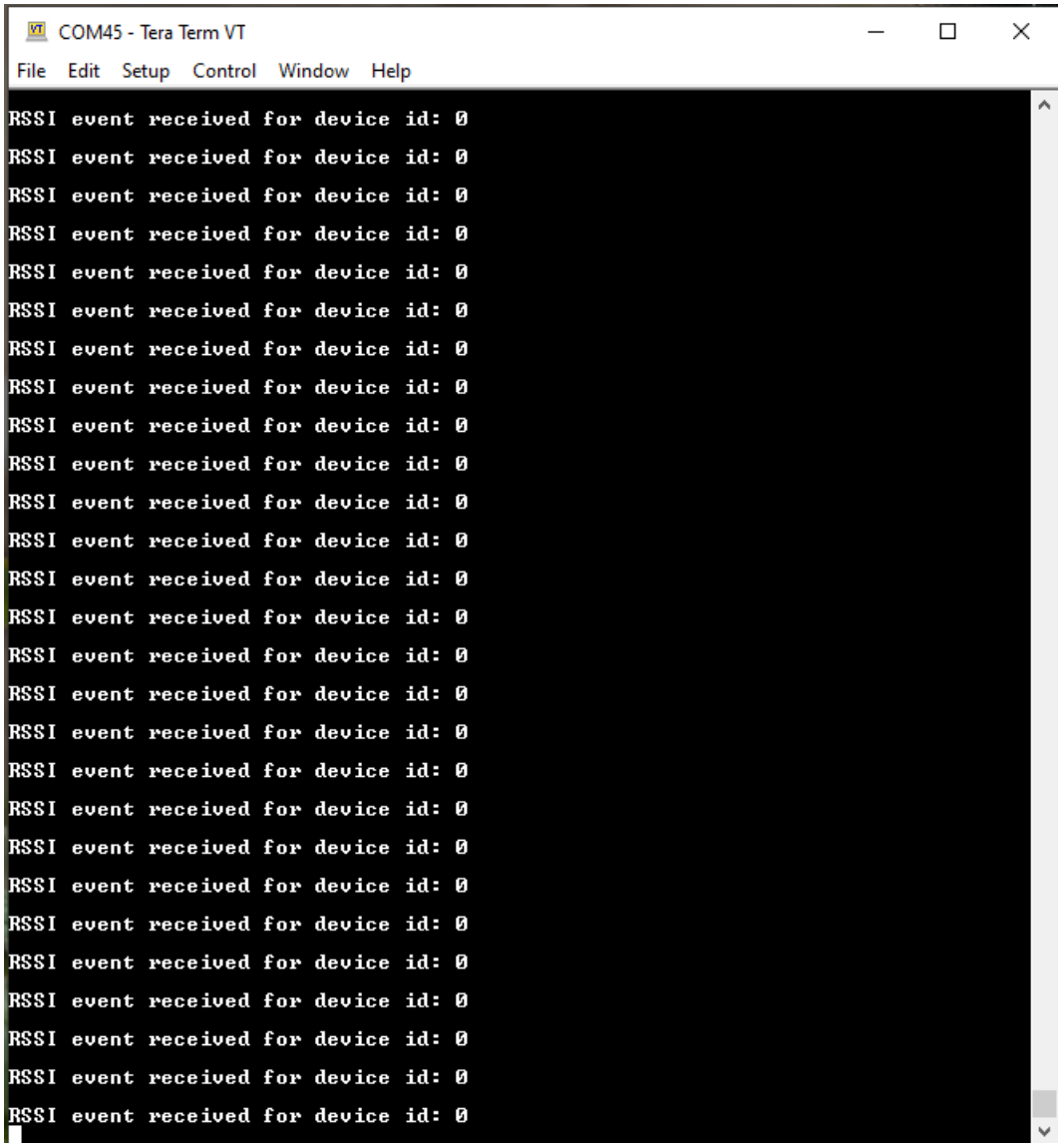


2. Run the "monitor start" shell command on the Car Anchor, as shown in the figure below

   The second Anchor has received the command over the serial interface and has started Anchor Monitoring

3. The second Car Anchor begins monitoring the connection and receives RSSI info events, which it forwards to the first Car Anchor. The received RSSI messages are then displayed on the console. This is shown in the figure below.

   The first Anchor displays RSSI events received from the second Anchor



4. To stop monitoring, run the `monitor stop` shell command on either of the Car Anchors (the second one is recommended, as its console is not flooded by RSSI event messages).

Parent topic:*Running the Connection Handover scenario*

Running the Packet Monitoring scenario

Packet Monitoring is an NXP proprietary feature that allows a second Anchor to receive the packets from the connection between the first Anchor and the Device. The prerequisites are the same hardware setup as used for the *Running the Connection Handover scenario*. Note, however, that the Connection Handover scenario cannot be performed while Packet Monitoring is in progress. Packets received by the second Anchor are sent over the serial interface to the first Anchor for processing. The first Anchor determines the source of the packet based on bit 1 of the `status` parameter, which indicates whether the packet was sent by the central (`1`) or the peripheral (`0`), and displays the RSSI.

Demo steps:

1. Connect the first Anchor and a Device using either the Owner Pairing or Passive Entry scenario as described in Owner Pairing scenario and Passive Entry scenario.

Anchor has performed Owner Pairing and the `packetmon start` command is run

```
COM45 - Tera Term VT
File  Edit  Setup  Control  Window  Help
Anchor>sd pe
Anchor>Advertising started - Legacy.
Advertising started - Extended LR.

Connected!
Advertising started - Extended LR.

L2CAP PSM Connection Complete.

Time Sync received.

Anchor>
Localization config complete.

CS security enabled.

Set Procedure parameters complete.

Start distance measurement.

[0] Distance measurement complete. Local data available for procedure index 1.

[0] Distance (RADE): 0.43 m   Quality: 99.0%

[0] RTT Distance: 1.0 m Success Rate: 100

[0] Distance measurement complete. Local data available for procedure index 2.

[0] Distance (RADE): 0.43 m   Quality: 99.0%

[0] RTT Distance: 0.9 m Success Rate: 100

[0] Distance measurement complete. Local data available for procedure index 3.

[0] Distance (RADE): 0.43 m   Quality: 99.0%

[0] RTT Distance: 0.9 m Success Rate: 100

[0] Distance measurement complete. Local data available for procedure index 4.

[0] Distance (RADE): 0.43 m   Quality: 99.0%

[0] RTT Distance: 0.9 m Success Rate: 100

[0] Distance measurement complete. Local data available for procedure index 5.

[0] Distance (RADE): 0.43 m   Quality: 99.0%

[0] RTT Distance: 1.2 m Success Rate: 100

Anchor>packetmon 0 start
Anchor>
```

Anchor has performed Owner Pairing and the "`packetmon start`" command is run.

2. Run the packetmon start command on the first Anchor, as shown in the figure below.

The second Anchor starts Packet Monitoring after receiving the command over the serial interface

```
COM25 - Tera Term VT
File  Edit  Setup  Control  Window  Help
Copyright  2024  NXP

Anchor>
Digital Key Car Anchor.

Anchor>
Anchor Monitor started.

Anchor>
```

3. The second Anchor begins monitoring the connection and receives packet information, which it forwards to the first Anchor. See the figure below.

The first Anchor displays the RSSI and source device of the received packets from the second Anchor

```
COM45 - Tera Term VT
File  Edit  Setup  Control  Window  Help
Packet monitor event received for device id 0, from central with RSSI: -74
Packet monitor event received for device id 0, from central with RSSI: -73
Packet monitor event received for device id 0, from central with RSSI: -72
Packet monitor event received for device id 0, from peripheral with RSSI: -61
Packet monitor event received for device id 0, from peripheral with RSSI: -60
Packet monitor event received for device id 0, from peripheral with RSSI: -61
Packet monitor event received for device id 0, from peripheral with RSSI: -61
Packet monitor event received for device id 0, from peripheral with RSSI: -61
Packet monitor event received for device id 0, from peripheral with RSSI: -60
Packet monitor event received for device id 0, from peripheral with RSSI: -59
Packet monitor event received for device id 0, from peripheral with RSSI: -61
Packet monitor event received for device id 0, from peripheral with RSSI: -60
Packet monitor event received for device id 0, from peripheral with RSSI: -61
Packet monitor event received for device id 0, from peripheral with RSSI: -59
Packet monitor event received for device id 0, from peripheral with RSSI: -61
Packet monitor event received for device id 0, from peripheral with RSSI: -59
Packet monitor event received for device id 0, from peripheral with RSSI: -60
Packet monitor event received for device id 0, from peripheral with RSSI: -61
Packet monitor event received for device id 0, from peripheral with RSSI: -59
Packet monitor event received for device id 0, from peripheral with RSSI: -59
Packet monitor event received for device id 0, from peripheral with RSSI: -60
Packet monitor event received for device id 0, from peripheral with RSSI: -61
Packet monitor event received for device id 0, from peripheral with RSSI: -61
Packet monitor event received for device id 0, from peripheral with RSSI: -59
Packet monitor event received for device id 0, from peripheral with RSSI: -61
Packet monitor event received for device id 0, from peripheral with RSSI: -59
Packet monitor event received for device id 0, from peripheral with RSSI: -59
```

4. The first Anchor receives the packet information from the second Anchor and determines the source of the packet based on bit 1 of the `status` parameter, which indicates whether the packet was sent by the central (1) or the peripheral (0). The source and the RSSI information is displayed in the console as shown in the preceding figure.
5. To stop monitoring, run the `packetmon stop` shell command on either Anchor. It is recommended to use the second Anchor, whose console is not flooded by packet information messages.

Parent topic: *Running the Connection Handover scenario*

Localization scenarios

Application configuration

This section describes configuration options available for the applications. The configuration can be done either at compile time, by using macros added in the `app_preinclude.h` file of the respective sample, or at runtime, via shell commands. The available configurations are:

- The Channel Sounding role can be set through the `role` shell command at runtime only if there are no active connections. The default role is set through the `gCsDefaultRole_c` define in `app_preinclude.h`. In the default configuration, the Anchor is in the CS Initiator role (`gCsRoleInitiator_c`) and the Device is in the CS Reflector role (`gCsRoleReflector_c`).

- The number of Channel Sounding procedures to be run via the CS Procedure Repeat mechanism, for each triggered distance measurement. The default value can be set through the gCsProcRepeatMaxNumProcedures_c define. At runtime, the value can be changed using the setnumprocs command.
- The ranging algorithm to run once the local and remote measurement data is available. Each of the two supported algorithms can be disabled at compile time for code size reasons. By default, both are enabled. To disable an algorithm at compile time, define its respective macro with the value 0 inside app_preinclude.h. The macros are gAppUseCDEAlgorithm_d, gAppUseRADEAlgorithm_d. At runtime, the setalgo shell command can be used to configure the algorithm that runs upon completion of the CS procedure and data transfer. Any combination of algorithms can be selected. By default, RADE is selected.
- The timing information display feature can be activated by setting the gAppCsTimeInfo_d macro to value 1 inside app_preinclude.h. Additional timing information such as the duration of the CS config phase, CS procedure, BTCS transfer, and running of the algorithms is displayed on the console.
- The BTCS L2CAP transfer watchdog is an **NXP proprietary extension** (the CCC BTCS specification does not define any timeout for the L2CAP measurement transfer). When the CS procedure auto-restart loop is enabled on the BTCS client, this watchdog ensures the loop does not stall if a BTCS L2CAP transfer is lost or never completes. The feature is client-side only and is always enabled. The watchdog timeout duration in seconds can be tuned through the gBtcsTimeoutSeconds_c macro (default 3). If a transfer does not complete before the timeout expires, the watchdog only restarts the CS procedure loop when the full procedure sequence (maxNumProcedures) has been reached; otherwise it simply returns to waiting for the next procedure's measurement data.

For better performance and user experience, the optional LE Coded PHY advertising should be disabled by setting gAppLeCodedAdvEnable_d to 0 inside app_preinclude.h on the Anchor.

- Maximum concurrent Channel Sounding procedures**: Configure the maximum number of Channel Sounding procedures that can run simultaneously across all connections using the gChannelSoundingMaxConcurrentProcedures_c define in app_preinclude.h.

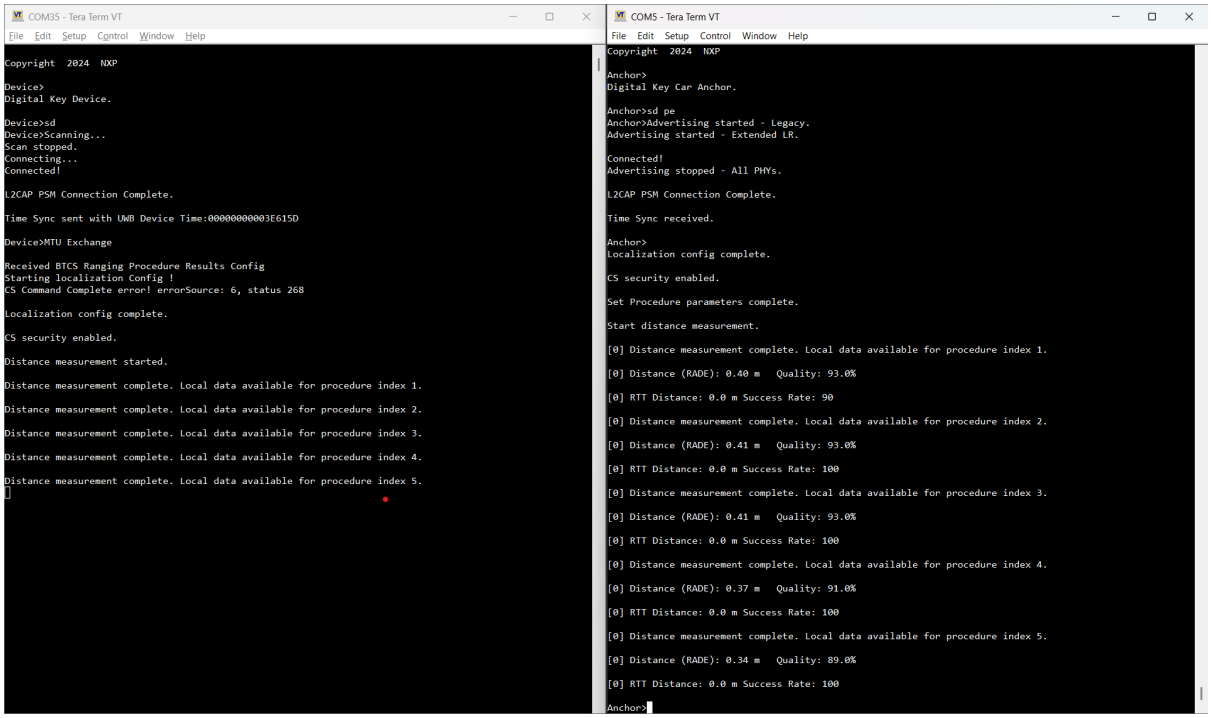
Parent topic:*Localization scenarios*

Localization after passive entry

The localization process starts automatically after the Passive Entry scenario is completed. The CS Initiator performs the CS setup and starts a number of gCsProcRepeatMaxNumProcedures_c procedures. After each procedure, the Device's local data is transmitted through BTCS to the Anchor. After the Anchor collects the data, it runs the selected algorithm(s) and displays the measured distance (meters) in the console.

The below figure shows the output of the localization process after each of the five measurements completed successfully and the distance is displayed.

Output of the localization process



Parent topic:*Localization scenarios*

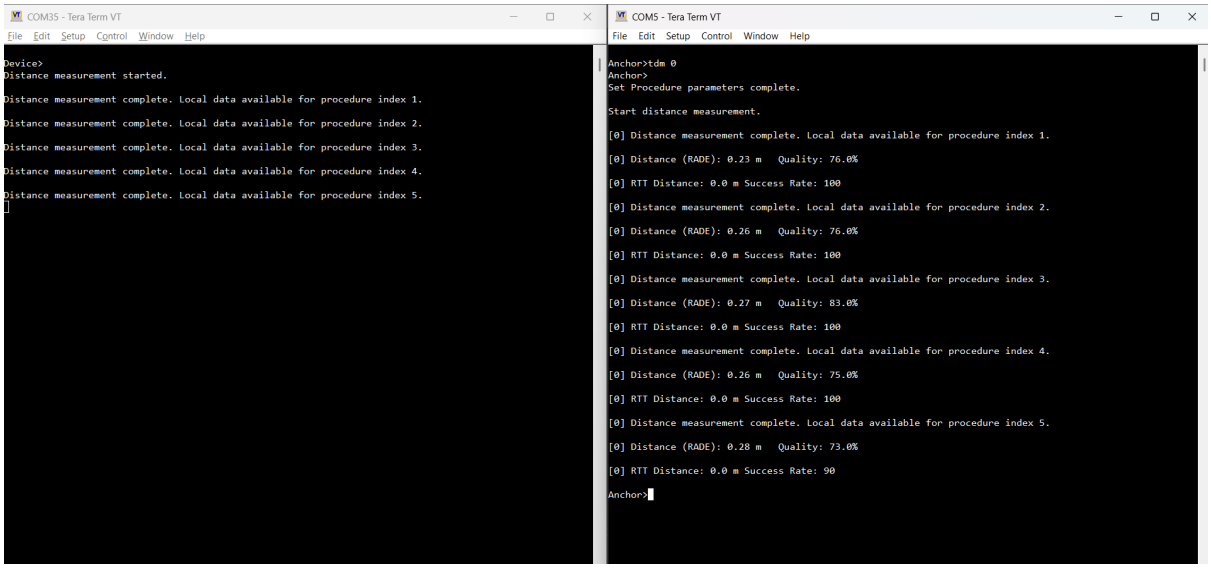
Localization on demand and other shell commands

Localization can also be performed on demand via the tdm shell command. The command can be used in two ways:

- tdm - Triggers distance measurement on all connected devices
- tdm <peerId> - Triggers distance measurement on the specified peer device (where peerId is the device ID of the peer)

As shown in the figure below, the Device triggers the localization process, and the default 5 measurements complete successfully.

Triggering the localization process



Other available shell commands are:

- setcsconfig: This command overwrites the default parameters for the *Channel Sounding Create Config* command. It updates only the parameters that are explicitly specified; any parameter that is not provided retains its current value. The syntax is:

  setcsconfig <peerId> <paramName> <value> [<paramName> <value> ...]

  where <peerId> is the device ID of the peer, followed by one or more <paramName> <value> pairs. The supported parameter names and their accepted values are:

  - mainmodetype: CS mode to be used as main mode during the CS procedure for this configuration. Possible values:
    - 1 (CS mode-1)
    - 2 (CS mode-2)
    - 3 (CS mode-3)
  - submodetype: CS mode to be used as submode during the CS procedure for this configuration. Possible values:
    - 1 (CS mode-1)
    - 2 (CS mode-2)
    - 3 (CS mode-3)
    - 255 (no submode used)
  - mainmodeminsteps: Lower bound of the range of main mode CS steps to be executed before a submode CS step is executed during the CS procedure. Range:
    - 1 to 255
  - mainmodemaxsteps: Higher bound of the range of main mode CS steps to be executed before a submode CS step is executed during the CS procedure. Range:
    - 1 to 255
  - mainmoderepetition: Number of main mode CS steps repeated from the previous CS subevent at the beginning of the current CS subevent. Range:
    - 0 to 3
  - mode0steps: Number of CS mode-0 steps to be included at the beginning of each CS subevent. Range:\
    - 1 to 3
  - role: CS role. Possible values:
    - 0 (initiator)
    - 1 (reflector)
  - rtttype: RTT variant to be used during the CS procedure. Possible values:
    - 0 (RTT AA only)
    - 1 (RTT with 32-bit sounding sequence)
    - 2 (RTT with 96-bit sounding sequence)
    - 3 (RTT with 32-bit random sequence)
    - 4 (RTT with 64-bit random sequence)
    - 5 (RTT with 96-bit random sequence)
    - 6 (RTT with 128-bit random sequence)
  - channelmap: 80-bit map in big endian order indicating channels to be used for the CS procedure. The default value is fcff7ffcffffffffff1f (all channels used except 0, 1, 23-25 and 77-78). Bit 79 is reserved for future use. At least 15 channels must be enabled.

  - channelmaprepetition: Number of times the channel map will be cyled through for non-mode-0 steps within a CS procedure. Range:
    * 1 to 255.
  - channelselectiontype: Channel selection type. Possible values:
    * 0 (Channel Selection Algorithm #3b)
    * 1 (Channel Selection Algorithm #3c)
  - cssyncphy: PHY to be used for CS_SYNC exchanges during the CS procedure for the specified CS configuration. Possible values:
    * 1 (LE 1M PHY)
    * 2 (LE 2M PHY)
    * 3 (LE 2M 2BT PHY) - if supported at Link Layer level

  Example: setcsconfig 0 mode0steps 3 cssyncphy 2 updates only the mode-0 steps and the CS_SYNC PHY, leaving all other parameters unchanged.

- setcsproc: This command overwrites the default parameters for the *Channel Sounding Set Procedure Parameters* command. It updates only the parameters that are explicitly specified; any parameter that is not provided retains its current value. The syntax is:

  setcsproc <peerId> <paramName> <value> [<paramName> <value> ...]

  where <peerId> is the device ID of the peer, followed by one or more <paramName> <value> pairs. The supported parameter names and their accepted values are:

  - maxprocedureduration: Maximum duration for each CS procedure. Range:
    * 1 to 65535 (units of 0.625ms)
  - minperiodbetweenprocedures: Minimum number of connection events between consecutive CS procedures. Range:
    * 1 to 65535
  - maxperiodbetweenprocedures: Maximum number of connection events between consecutive CS procedures. Range:
    * 1 to 65535
  - maxnumprocedures: Maximum number of CS procedures to be scheduled. Possible values:
    * 0 (CS procedures to continue until disabled)
    * 1 to 65535
  - minsubeventlen: Minimum suggested duration for each CS subevent in microseconds. Range:
    * 1250 microseconds to 3.999999 seconds
  - maxsubeventlen: Maximum suggested duration for each CS subevent in microseconds. Range:
    * 1250 microseconds to 3.999999 seconds
  - antcfgindex: Antenna Configuration Index as described in the Core specification. Range:
    * 0 to 7
  - snrcontrolinit: SNR Control to be applied to the Initiator in the CS procedure (if supported at Link Layer level). Possible values:
    * 0 (SNR control adjustment of 18 dB)
    * 1 (SNR control adjustment of 21 dB)

    * 2 (SNR control adjustment of 24 dB)
    * 3 (SNR control adjustment of 27 dB)
    * 4 (SNR control adjustment of 30 dB)
    * 255 (SNR control is not to be applied)
  - snrcontrolrefl: SNR Control to be applied to the Reflector in the CS procedure (if supported at Link Layer level). Possible values:
    * 0 (SNR control adjustment of 18 dB)
    * 1 (SNR control adjustment of 21 dB)
    * 2 (SNR control adjustment of 24 dB)
    * 3 (SNR control adjustment of 27 dB)
    * 4 (SNR control adjustment of 30 dB)
    * 255 (SNR control is not to be applied)

  Example: setcsproc 0 minsubeventlen 2500 antcfgindex 1 updates only the minimum subevent length and the antenna configuration index, leaving all other parameters unchanged.

- verbosity: Sets the verbosity level during the CS procedure.
- setnumprocs: This command overwrites the default value gCsProcRepeatMaxNumProcedures_c parameter for the CS Procedure Repeat. It takes the following parameters, in the order below:
  - peerId
  - maxnumprocedures: Number of procedures in hex format. Example: setnumprocs 0 0x0005. Value 0x0000 means infinite repeats.
- loop: This command toggles the automatic restart of the CS procedure (whether with CS Procedure Repeat enabled or not) upon its end. By default, the CS procedure will not restart automatically. **Parent topic:***Localization scenarios*

BTCS L2CAP data transfer

The CCC introduced the Bluetooth LE Channel Sounding (BTCS) measurement transfer feature. BTCS procedures are configured and executed using the Bluetooth Low Energy connection established for the Digital Key Service. The results of the BTCS procedures are transferred to the vehicle and used to estimate the distance. The data transfer is done through the L2CAP channel created during the passive entry scenario. The format of the data transferred through BTCS is defined by the CCC. The algorithms used on the receiver to estimate the distance are (RADE and CDE).

The default transfer direction enabled in the digital_key_car_anchor_cs and digital_key_device_cs is from Device to the Anchor. To reverse the transfer direction in BTCS, the following defines must be set in the app_preinclude.h file:

- digital_key_car_anchor_cs: gAppBtcsClient_d set to 0 and gAppBtcsServer_d set to 1
- digital_key_device_cs: gAppBtcsServer_d set to 0 and gAppBtcsClient_d set to 1

The figure below shows the car Anchor output after BTCS transfer direction is reversed.

BTCS transfer - car Anchor output

```
COM5 - Tera Term VT
File  Edit  Setup  Control  Window  Help

Copyright  2024  NXP

Anchor>
Digital Key Car Anchor.

Anchor>sd pe
Anchor>Advertising started - Legacy.
Advertising started - Extended LR.

Connected!
Advertising stopped - All PHYs.

L2CAP PSM Connection Complete.

Time Sync received.

Anchor>
Received BTCS Ranging Procedure Results Config

Localization config complete.

CS security enabled.

Set Procedure parameters complete.

Start distance measurement.

[0] Distance measurement complete. Local data available for procedure index 1.

[0] Distance measurement complete. Local data available for procedure index 2.

[0] Distance measurement complete. Local data available for procedure index 3.

[0] Distance measurement complete. Local data available for procedure index 4.

[0] Distance measurement complete. Local data available for procedure index 5.
```

The figure below shows the Device output after BTCS is enabled.

BTCS transfer - Device output

```
COM35 - Tera Term VT
File  Edit  Setup  Control  Window  Help

Copyright  2024  NXP

Device>
Digital Key Device.

Device>sd
Device>Scanning...
Scan stopped.
Connecting...
Connected!

L2CAP PSM Connection Complete.

Time Sync sent with UWB Device Time:0000000000427FC0

Device>MTU Exchange
Starting localization Config !

Localization config complete.

CS security enabled.

Distance measurement started.

Distance measurement complete. Local data available for procedure index 1.

[0] Distance (RADE): 0.60 m   Quality: 99.0%

[0] RTT Distance: 0.0 m Success Rate: 91

Distance measurement complete. Local data available for procedure index 2.

[0] Distance (RADE): 0.60 m   Quality: 99.0%

[0] RTT Distance: 0.0 m Success Rate: 100

Distance measurement complete. Local data available for procedure index 3.

[0] Distance (RADE): 0.60 m   Quality: 99.0%

[0] RTT Distance: 0.0 m Success Rate: 100

Distance measurement complete. Local data available for procedure index 4.

[0] Distance (RADE): 0.60 m   Quality: 99.0%

[0] RTT Distance: 0.0 m Success Rate: 100

Distance measurement complete. Local data available for procedure index 5.

[0] Distance (RADE): 0.58 m   Quality: 99.0%

[0] RTT Distance: 0.0 m Success Rate: 83

Device>
```

Both digital_key_car_anchor_cs and digital_key_device_cs applications support gAppBtcsClient_d and gAppBtcsServer_d configuration options at the same time. Also, the algorithm can be run on both (gAppRunAlgo_d can be set on both applicaitons at the same time). Here is a picture showing both applications running the algorithm.

```
Anchor>sd pe
Anchor>Advertising started - Legacy.
Advertising started - Extended LR.

Connected!
Advertising stopped - All PHYs.

L2CAP PSM Connection Complete.

Time Sync received.

Anchor>
Received BTCS Ranging Procedure Results Config

Localization config complete.

CS security enabled.

Set Procedure parameters complete.

Start distance measurement.

[0] Distance measurement complete. Local data available for procedure index 1.

[0] Distance (RADE): 0.05 m   Quality: 99.0%

[0] RTT Distance: 0.0 m Success Rate: 100

[0] Distance measurement complete. Local data available for procedure index 2.

[0] Distance (RADE): 0.05 m   Quality: 99.0%

[0] RTT Distance: 0.3 m Success Rate: 100

[0] Distance measurement complete. Local data available for procedure index 3.

[0] Distance (RADE): 0.05 m   Quality: 99.0%

[0] RTT Distance: 0.0 m Success Rate: 100

[0] Distance measurement complete. Local data available for procedure index 4.

[0] Distance (RADE): 0.05 m   Quality: 99.0%
```

```
Device>sd
Device>Scanning...
Scan stopped.
Connecting...
Connected!

L2CAP PSM Connection Complete.

Time Sync sent with UWB Device Time:000000000004A4818

Device>MTU Exchange
Starting localization Config !

Received BTCS Ranging Procedure Results Config

Localization config complete.

CS security enabled.

Distance measurement started.

Distance measurement complete. Local data available for procedure index 1.

[0] Distance (RADE): 0.05 m   Quality: 99.0%

[0] RTT Distance: 0.0 m Success Rate: 100

Distance measurement complete. Local data available for procedure index 2.

[0] Distance (RADE): 0.05 m   Quality: 99.0%

[0] RTT Distance: 0.3 m Success Rate: 100

Distance measurement complete. Local data available for procedure index 3.

[0] Distance (RADE): 0.05 m   Quality: 99.0%

[0] RTT Distance: 0.0 m Success Rate: 100

Distance measurement complete. Local data available for procedure index 4.

[0] Distance (RADE): 0.05 m   Quality: 99.0%

[0] RTT Distance: 0.0 m Success Rate: 100
```

BTCS L2CAP transfer watchdog (NXP proprietary extension)

The CCC BTCS specification does not define any timeout for the L2CAP measurement transfer. As an NXP proprietary extension, the BTCS client provides a watchdog timer that prevents the CS procedure auto-restart loop from stalling when a BTCS L2CAP transfer is lost or never completes. The watchdog is client-side only and is always enabled. Its duration in seconds is configured through gBtcsTimeoutSeconds_c (default 3). The timer is armed when the client starts waiting for the peer transfer and is stopped as soon as the transfer completes. If it expires, the loop is only restarted once the full procedure sequence (maxNumProcedures) has been reached; otherwise the client just returns to waiting for the next procedure's measurement data.

Parent topic:*Localization scenarios*

Running the A2B scenario

The A2B feature allows for the secure transfer of Bluetooth Low Energy security keys information between the Car Anchor devices. The feature is supported on the following platforms:

- KW47-EVK with a life cycle state of OEM-Open or higher, this feature is provided by th EdgeLock Secure Enclave.
- FRDM-KW43 with a life cycle state of OEM-Open or higher, this feature is provided by the EdgeLock Secure Enclave.

The Car Anchor demo application makes use of the A2B feature to securely synchronize the local Identity Resolving Key (IRK) and the Bonding Data (LTK Long Term Key and peer IRK) with another Car Anchor.

Prerequisites:

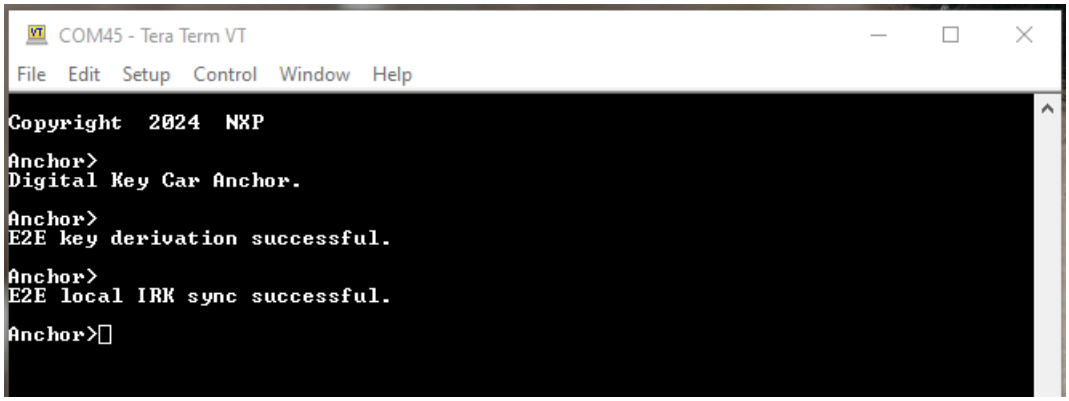
- Three boards (two act as Car Anchors, one as a Device). The demo is currently limited to a maximum of two Car Anchors.
- The two Car Anchors must be connected via a serial interface as described in the section above.
- Advanced Secure Mode must be enabled. The following macros must be defined and set to 1 in *app_preinclude.h* for the ***digital_key_car_anchor_cs*** project:
  - gAppSecureMode_d
  - gA2BEnabled_d

- One of the Car Anchors must be configured with the *gA2BInitiator_d* macro set to 1, this is called Car Anchor A, and the other must be configured with the *gA2BInitiator_d* macro set to 0, called Car Anchor B. Car Anchor A triggers the EdgeLock-to-EdgeLock (E2E) key derivation and local IRK synchronization. Therefore, it should be started up after Car Anchor B.

Demo steps:

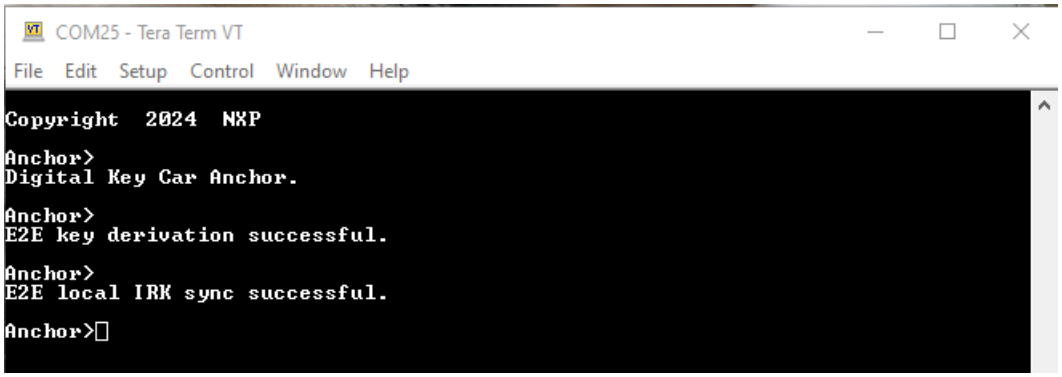
- Start Car Anchor B, then Car Anchor A. At initialization the E2E key is derived and the local IRK of Car Anchor A is sent as a secure blob to Car Anchor B in order for both Car Anchors to have the same local IRK. This process is shown in the figure below.

Car Anchor A starts up, derives the E2E key and syncs with Car Anchor B



- The below figure shows the Car Anchor B starting up and syncing with Car Anchor A.

Car Anchor B starts up and syncs with Car Anchor A



- Trigger the Owner Pairing scenario by sending the "sd" command on the Device and "sd op" on Car Anchor A. Once the bond is created, the Bonding Data containing secured blobs for the LTK and peer IRK is sent to Car Anchor B. The Car Anchors print the Bonding Data in shell with the LTK in blob form (the LTK is never available in plain text) and the peer IRK in plain text. Now both Car Anchors have the same Bonding Data. The figure below shows that the Car Anchor A pairs with the Device.

Car Anchor A pairs with Device

```
COM45 - Tera Term VT
File  Edit  Setup  Control  Window  Help

Copyright  2024  NXP

Anchor>
Digital Key Car Anchor.

Anchor>
E2E key derivation successful.

Anchor>
E2E local IRK sync successful.

Anchor>sd op
Anchor>Advertising started - Legacy.

Connected!

L2CAP PSM Connection Complete.

Received Command Complete SubEvent: Request_owner_pairing.

SPAKE Request sent.

SPAKE Response received.

SPAKE Verify sent.

SPAKE Verify received.

Sending Command Complete SubEvent: BLE_pairing_ready

Received First_Approach_RQ.

Sending First_Approach_RS

Pairing...

BondingData: 00 C5BC703760C4 2DBC592CE7C44537B5094F58DC992149FDE5D1F2F364267E561
F20DE0E5B9518B2A9EE94266E74B2 692865C9A376A1A82D161B0F95785955

Pairing successful.

Anchor>
Time Sync received.

Anchor>
```

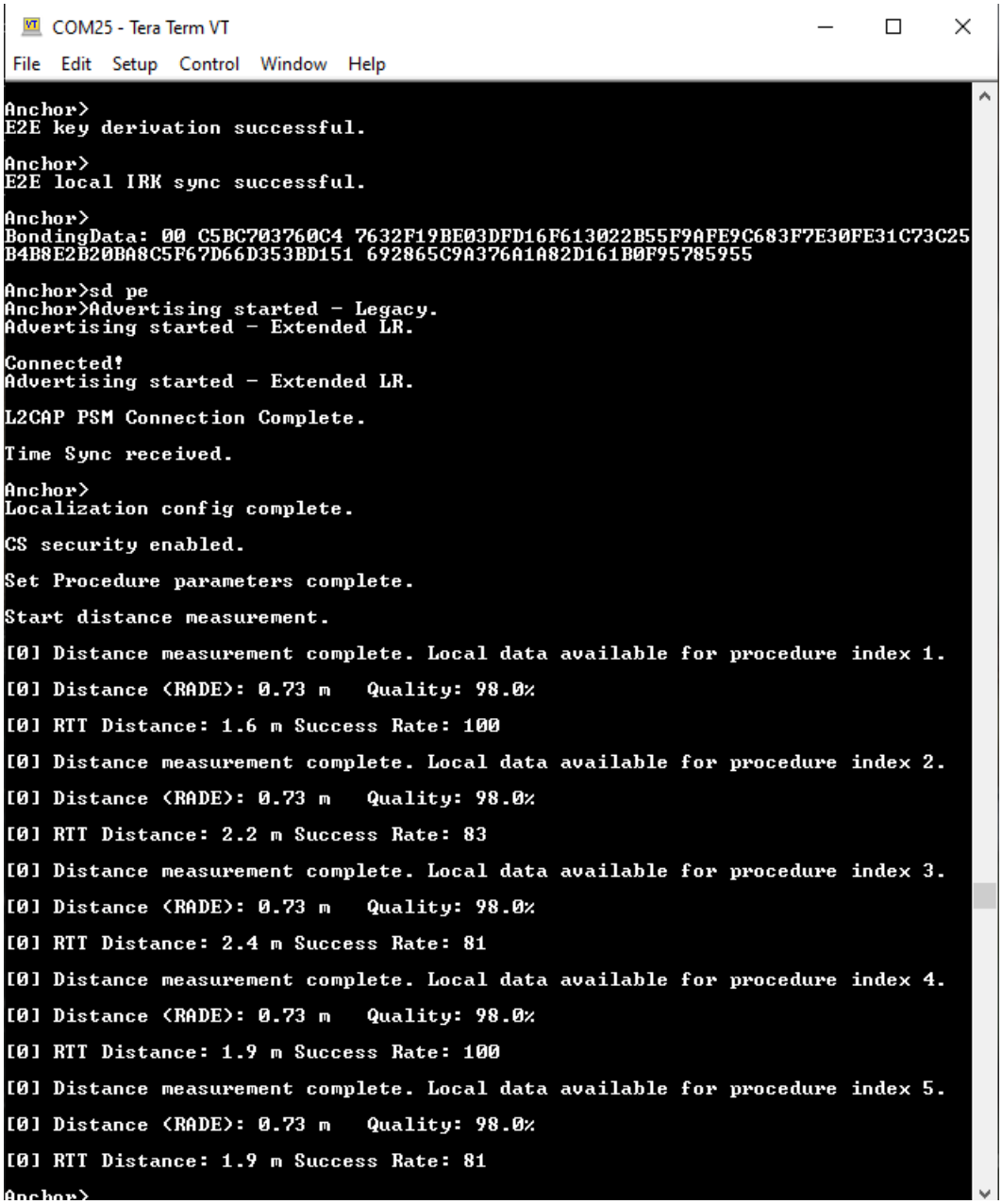
The figure below shows that the Car Anchor B receives the Bonding Data via the serial interface.

Car Anchor B has received the Bonding Data via the serial interface

```
COM25 - Tera Term VT
File  Edit  Setup  Control  Window  Help

Copyright  2024  NXP

Anchor>
Digital Key Car Anchor.

Anchor>
E2E key derivation successful.

Anchor>
E2E local IRK sync successful.

Anchor>
BondingData: 00 C5BC703760C4 7632F19BE03DFD16F613022B55F9AFE9C683F7E30FE31C73C25
B4B8E2B20BA8C5F67D66D353BD151 692865C9A376A1A82D161B0F95785955

Anchor>
```

- Disconnect Car Anchor A by sending the `dcnt` command.
- Once Car Anchor A is disconnected, run the `sd` command on the Device and run the `sd pe` command on Car Anchor B. Car Anchor B is now able to perform a Passive Entry connection, without pairing. The below figure shows the process.

Car Anchor B performs Passive Entry without pairing



Low-power mode

The CCC Owner Pairing and Passive Entry scenarios (with Channel Sounding support) can also run in Low-power mode. This feature is supported on the following platforms:

- KW47-EVK
- KW47-LOC
- KW43-LOC
- FRDM-KW43

To enable Low-power mode, set the `gAppLowpowerEnabled_d` macro to `1` in the `app_preinclude.h` file for the **digital_key_car_anchor_cs** and/or the **digital_key_device_cs** projects.

In Low-power mode, the shell interface is not available. Interaction with the demos is done through buttons. The button configuration is the same for all supported platforms.

The below table lists the buttons applicable for the car Anchor.

Car Anchor buttons

Button	Press	Description
SW2	(short press)	Starts Owner Pairing ADV
SW3	(long press)	Starts Passive Entry ADV

The below table lists the buttons applicable for the Device.

Device buttons

Button	Press	Description
SW2	(short press)	Starts Scanning

Acronyms and abbreviations

The following table lists the acronyms that have been used in this document.

Abbreviation	Description
Bluetooth LE	Bluetooth Low Energy
BTCS	Bluetooth LE Channel Sounding
CAN	Controller Area Network
CCC	Car Connectivity Consortium
CR	Change Request
CS	Channel Sounding
CDE	Complex-domain Distance Estimation
E2E	EdgeLock-to-EdgeLock
HCI	Host Controller Interface
IRK	Identity Resolving Key
L2CAP	Logical Link Control and Adaptation Protocol
LTK	Long Term Key
OOB	Out-of-Band
RADE	Rapid Accurate Distance Estimator
RSSI	Received Signal Strength Indicator
SDK	Software Development Kit
UWB	Ultra-Wideband

References

For more information, refer to the following documents:

- Bluetooth Low Energy Application Developer's Guide (KW47_BLEADG)
- Bluetooth Low Energy Host Stack API Reference Manual (KW47_BLEHSAPIRM)
- Bluetooth Low Energy Host Stack FSCI (Framework Serial Connectivity Interface) API Reference Manual(KW47_BLEHSFSCIRM)
- Connectivity Framework Reference Manual (KW47_CONNFWRM)
- Bluetooth Low Energy CCC Digital Key R3 Application Note (AN12791)

Refer to the the below specifications:

- Digital Key Release 3.0 Specifications: https://carconnectivity.org/digital-key-release-3-0-specification-download/
- Bluetooth LE Core Specification 6.0: https://www.bluetooth.com/specifications/specs/core-specification-6-0/

Note about the source code in the document

Example code shown in this document has the following copyright and BSD-3-Clause license:

Copyright 2023-2025 NXP Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Revision history

The table below summarizes the revisions to this document.

Document ID	Release date	Description
AN13979 v.2.2	06 April 2026	- Added support for FRDM-KW43 platform. See *CCC Digital Key hardware platforms*

- Added the FRDM-KW43 platform in documentation.

|AN13979 v.2.1|18 March 2025|- Added support for KW47-LOC platform. See *CCC Digital Key hardware platforms*

- Added the sections Exporting HCI data and BTCS L2CAP data transfer.
- Other updates.

| |AN13979 v.2.0|26 November 2024|- Added support for KW47-EVK platform. See *CCC Digital Key hardware platforms*

- Updated for KW47 EAR 2.1 release

| |AN13979 v.1.3|June 2024|Updated for SDK 2.16.000| |AN13979 v.1.2|25 January 2024|Updated for MR4Added Running the Packet Monitoring scenario and *Low-power mode*

| |AN13979 v.1.1|16 October 2023|Initial release for MR3- Added Running the Anchor Monitoring scenario

- Modified *Running the Connection Handover scenario*

| |AN13979 v.1.0|23 June 2023|Initial release for MR2|

CS Wireless Ranging Demo Application User Guide Introduction

The application presented in this document relies on channel sounding (CS) Bluetooth Low Energy implementation as defined by the Bluetooth Special Interest Group (SIG).

CS protocol performs secure and highly accurate distance estimation between two Bluetooth Low Energy devices. CS supports two different measurement modes: RTT and RTP. RTP is used for an accurate distance estimation and RTT to get a coarse but secure distance bound.

For more details about CS techniques, see *Bluetooth Channel Sounding Technique for Distance Estimation* (document AN13481).

The application relies on LE_CS_Create_Config, LE_CS_Set_Procedure_Parameters, and LE_CS_Set_Procedure_Enable HCI commands to perform CS measurements. CS measurement results are sent through host controller interface (HCI) using LE_CS_Subevent_Result_Continue and LE_CS_Subevent_Result_Continue HCI events. They are centralized on the client board by the wireless ranging application to be combined and passed to various distance computation algorithms, being embedded, or PC-based.

See Acronyms for details on which part of the standard is supported.

Wireless ranging application allows to:

- Configure most of the parameters required for a CS measurement (number of steps, type of steps, CS channels, CS timings (T_FCS, T_IP, T_PM), and so on)
- Select what type of measurement to be performed (RTT, RTP, or both) based on configured CS step modes (1 or 2)
- Initiate CS setup phase (exchange of capabilities, configuration, and so on)
- Trigger CS measurements
- Retrieve CS measurement results (RTP, RTT, and other debug information) from peer device over Bluetooth Low Energy connection
- Compute distance estimations by combining RTT and RTP measurements from both devices
- Log system debug information but also raw IQ data information in Matlab (*.mat), Excel (*.xls), or numpy (*.npz) format that can be used for further analysis or development of distance computation algorithms

Wireless ranging application is made of two parts:

- Embedded firmware that can be controlled manually via a serial connection
- Host application (Python) running on a PC and controlling the firmware using a serial link

The embedded firmware is in charge of:

- Initiate CS setup (CS capabilities, CS security)
- Handling CS configuration and measurement requests from the host application
- Synchronizing initiator and reflector devices
- Scheduling and executing CS measurements over the air
- Generate pseudo random sequences for channels, AAs, and so on, using CS DRBG
- Collecting measurement results from one device to the other
- Send back CS measurement results to the host application
- Perform distance computation using embedded algorithms (namely CDE and RADE)

While the host application:

- Automates measurement requests
- Logs measurement results in output files

- Performs two ways channel reconstruction from captured PCT data and runs the high-resolution phase-based ranging algorithm (RADE)

Requirements

This chapter provides information about hardware and software requirements.

Hardware and software requirements

For information on hardware setup and how to build and flash an application using the toolchain of your choice, see the *Getting Started Guide* for your selected board.

CS wireless ranging demo application is designed to run on KW47-EVK, KW47-LOC, KW43-LOC, MCX-W72-EVK, MCXW72-LOC, FRDM-MCXW72 and FRDM-KW43 platforms, referred to as "board" in the rest of this document.

Parent topic:*Requirements*

Toolchain requirements

WR demo application was compiled and tested with IAR Embedded Workbench and Visual Studio Code. Employing one of these IDEs is recommended.

Parent topic:*Requirements*

Wireless ranging demo application setup

This chapter provides information about hardware and software setup.

Hardware set up with single antenna configuration

Prerequisites

- Two boards (ranging service client and ranging service server)
- PC

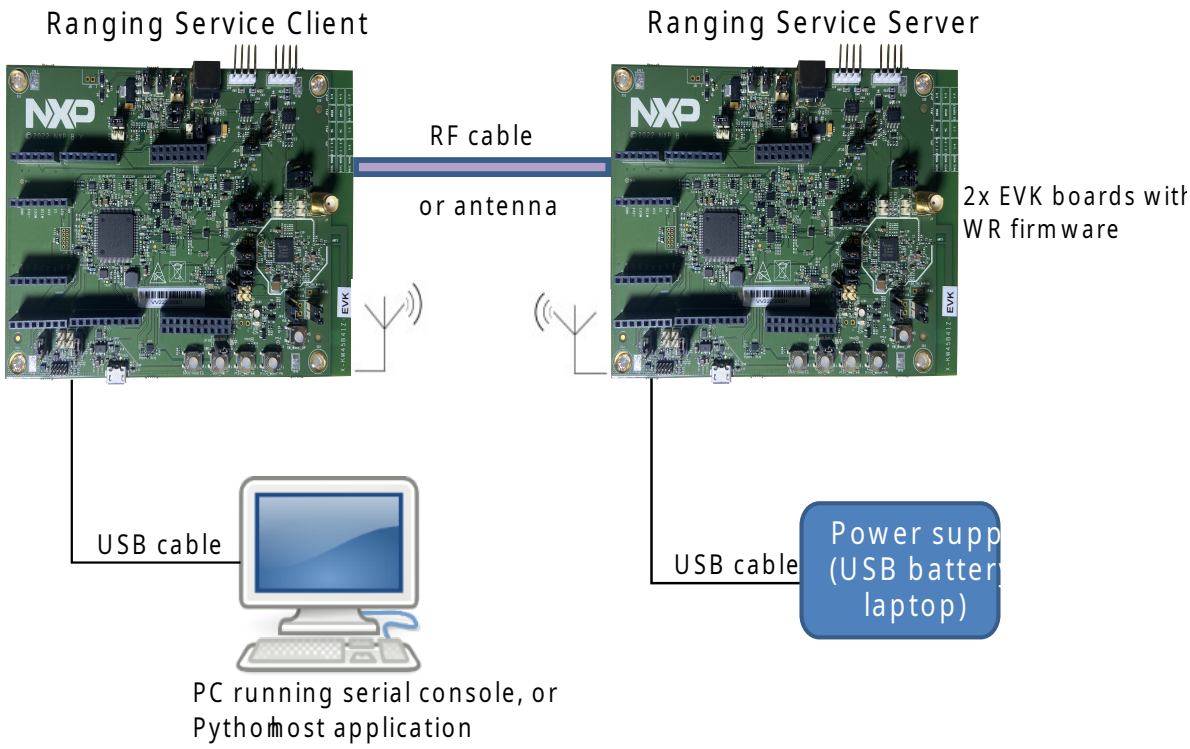
The first board (ranging service client) should be connected to the PC running the host application (see Python host application setup) using a USB cable. Once flashed, the second board (ranging service server), may be connected to a USB battery to facilitate movement.

Start the evaluation of this ranging system with a cabled configuration. Once the system operation is properly understood, the evaluation can be pursued in more challenging environments (with multi-path and interference).

To perform cabled measurements with KW47 boards, a small hardware modification must be done on the PCB. Rotate a capacitor to bypass the printed antenna and connect the RF path to an external SMA connector.

For further details, refer to the documentation corresponding to your board of choice. To perform radiated testing, connect an SMA-mounted antenna on each board instead of a cable.

Figure: Typical wireless ranging setup using KW47-EVK boards

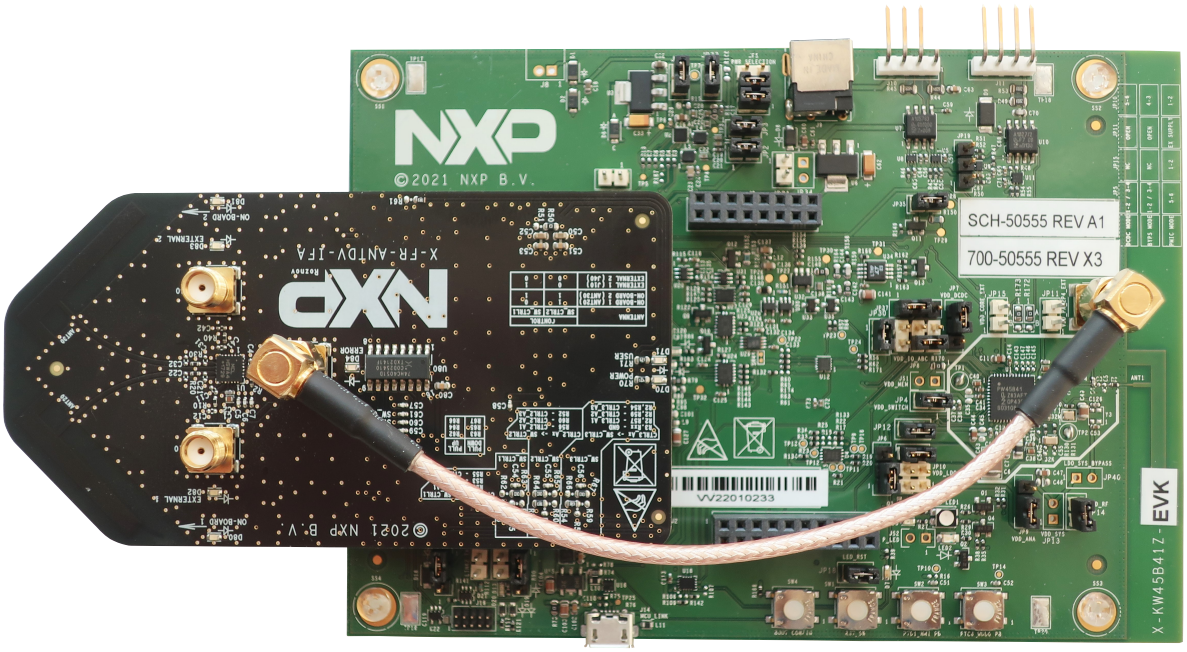


Parent topic: *Wireless ranging demo application setup*

Hardware set up with antenna diversity

To enable antenna diversity on KW47-EVK boards, antenna diversity boards (X-FR-ANTDV-IFA or X-FR-ANTDV-MPIFA) must be plugged to the EVKs using a similar setup as described in Hardware set up with single antenna configuration. The X-FR-ANTDV-IFA is a preferred option. Currently, only the X-FR-ANTDV-IFA is being distributed to the customers.

For a detailed setup description, see *Antenna Diversity Board User's Guide* (document ADBUG). **Figure: Antenna diversity board mounted on KW45 EVK board**



From antenna diversity boards, you must choose the pair of antennas, either the two SMA con-

nectors or the two printed antennas, to be used.

Another convenient way to enable antenna diversity testing is to use the KW47-LOC or KW43-LOC board which embeds two printed antennas on the same PCB. For a detailed description of these boards and how to set them up, refer to the *KW47-LOC Board User Manual* or the *KW43-LOC Board User Manual*.

To configure antenna diversity properly in the firmware, it is required to configure the following:

- Antenna diversity board type and antennas used (SMA connectors or printed antennas)
- Antenna configuration index
- Antenna switching time (T_SW)

Note: All platforms support only 1x1 or 2x2 antenna configurations, leading to either one or four antenna paths being used.

Parent topic:*Wireless ranging demo application setup*

Firmware setup

To install the firmware on the boards, perform the steps below:

Note: The firmware must be installed on the two boards following those two steps.

1. Update your NBU image with the provided binaries in the following folder of the SDK: ../middleware/wireless/ble_controller/bin (*_hadm_* for KW45 family).
2. Compile the application wireless_ranging_bluetooth_bm by referring to the quick start guide specific to your platform. The application can be found in the directory:

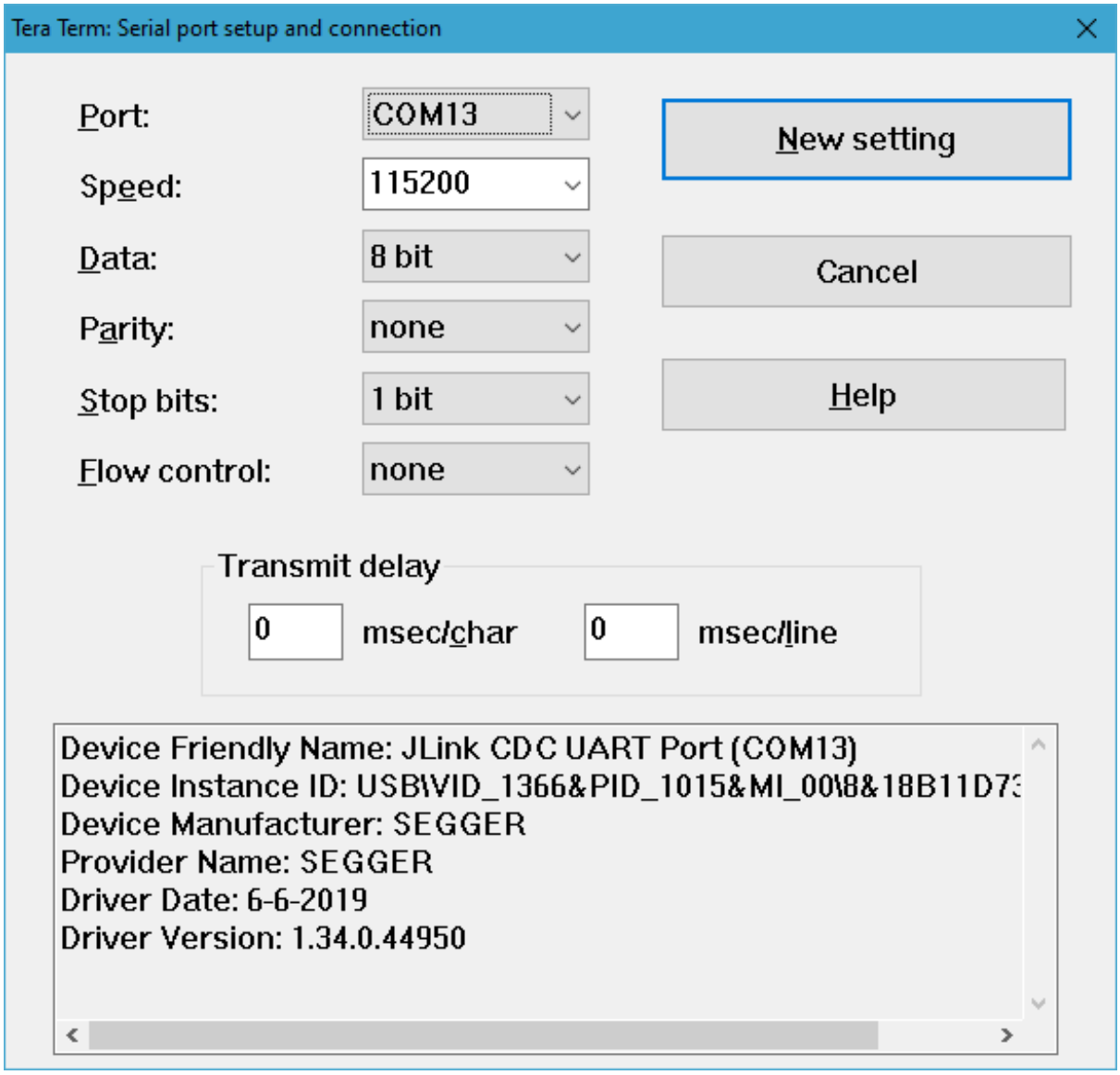
   \\boards\<board>\wireless_examples\bluetooth\wireless_ranging
3. Flash the binary on both devices. Flash the application using your preferred IDE.

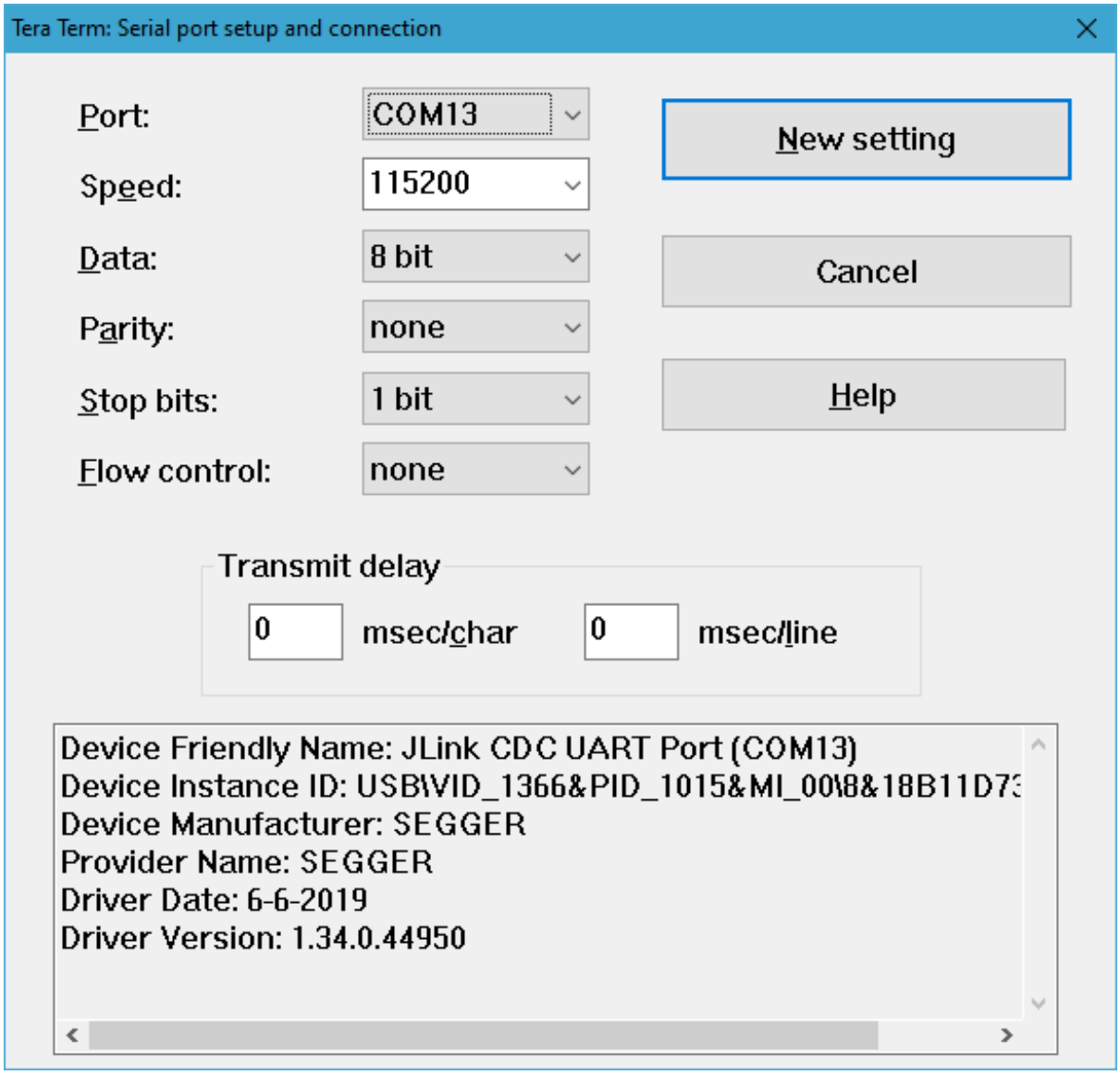
Parent topic:*Wireless ranging demo application setup*

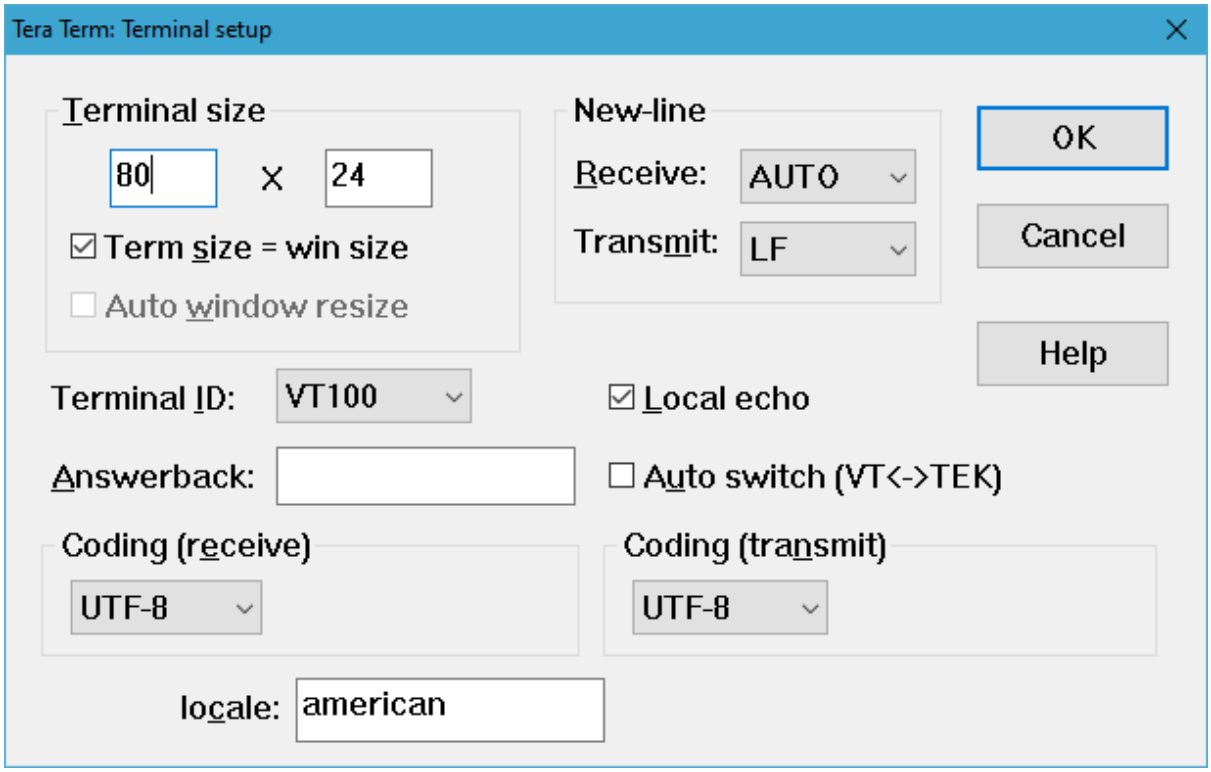
Serial console setup

Wireless ranging application can be used either by using a console or by using a Python script environment. To use the console, connect to each device COM port with Tera Term using the setup (putty is not supported), as shown in the figure below.

Figure: Tera Term configuration for operating with wireless ranging


Tera Term: Serial port setup and connection
Port:
COM13
Speed:
115200
Data:
8 bit
Parity:
none
Stop bits:
1 bit
Flow control:
none
New setting
Cancel
Help
Transmit delay
0
msec/char
0
msec/line
Device Friendly Name: JLink CDC UART Port (COM13)
Device Instance ID: USB\VID_1366&PID_1015&MI_00\8&18B11D73
Device Manufacturer: SEGGER
Provider Name: SEGGER
Driver Date: 6-6-2019
Driver Version: 1.34.0.44950


Tera Term: Serial port setup and connection
Port:
COM13
Speed:
115200
Data:
8 bit
Parity:
none
Stop bits:
1 bit
Flow control:
none
New setting
Cancel
Help
Transmit delay
0
msec/char
0
msec/line
Device Friendly Name: JLink CDC UART Port (COM13)
Device Instance ID: USB\VID_1366&PID_1015&MI_00\8&18B11D73
Device Manufacturer: SEGGER
Provider Name: SEGGER
Driver Date: 6-6-2019
Driver Version: 1.34.0.44950



Parent topic: *Wireless ranging demo application setup*

Python host application setup

To use the Python host application, go to tools\lcl_host. This folder is referred later in the document as <app_folder>.

This application is a Python application which requires several packages and Python 3.12 or newer. The supported OS is Windows. It has not been tested on other Python-capable OS.

The recommended approach is to install the required packages inside a Python virtual environment. This keeps the packages isolated from your system-wide Python installation and avoids conflicts with existing packages.

The list of required packages is provided in <app_folder>\python\requirements.txt. The same packages are also listed in <app_folder>\python\Pipfile.

To use the Python host application, follow the steps below:

Note: The following installation procedure is provided for Windows. Use Windows command prompt to execute commands below.

1. Install Python 3.12 or newer for 64-bit platform.

   Note: The "python" executable must point to Python 3.12 or newer (use python -V to check the version).

2. Go to the <app_folder>/python folder and create a virtual environment.

   ```
   cd <app_folder>/python
   python -m venv .venv
   ```

3. Activate the virtual environment.

   ```
   source .venv\Scripts\activate
   ```

   After activation, the command prompt is prefixed with (.venv).

4. Install the required packages inside the virtual environment.

```
pip install -r requirements.txt
```

This command may take several minutes to complete.

When you are done, you can leave the virtual environment by running `deactivate`. Remember to activate the virtual environment again (step 3) before running the host application in a new terminal session.

Parent topic: *Wireless ranging demo application setup*

Running wireless ranging demo application

This chapter provides information on how to run wireless application.

Using the console

Choose this option if you want to perform simple non-automated measurements without invoking any ranging algorithm.

If you want to run hosted algorithms or automated series of measurements, the host application is a better choice. See the following chapters for more detail.

Configuring the ranging service client

Connect to the board that acts as the ranging service client (the board that initiates the measurement request) using a serial console (see Firmware setup).

Run the following command:

```
communication role {central | peripheral}
```

Parent topic: Using the console

Triggering one CS measurement

First, ensure that you have configured the ranging service server and ranging service client boards as described above.

To trigger a measurement, run the following command:

```
range
```

Both devices synchronize to each other and perform a CS test procedure using default settings.

The expected output is a JSON-like structure similar to the following:

```
items:[{mciq:{cfg:{n_ap:1,n_stp:79},result:{vf:79,cde:8.76,cqi:1.000},},tof:{cfg:{n_stp:13},result:{ad:7.8,
↪sr:100},},},]
CRC32:cf49a670
marker:[DONE]
```

If verbosity is increased, the output may contain numerous additional information. It may contain all CS results output as well as debug information.

See section Command response for how to interpret these results.

Parent topic: Using the console

Changing CS configuration

The console configures numerous CS parameters.

See *Embedded firmware command reference* for the exhaustive firmware configuration commands that can be passed through the command line interface (CLI).

Parent topic: Using the console

Parent topic:*Running wireless ranging demo application*

Using Python host application

Using Python application to trigger CS measurements is the preferred option. It takes care of automatically interpreting the CS results, passing them into the selected algorithm, and presenting the output in user-friendly formats.

At this stage, we assume that the setup is properly connected and firmware and host application have been properly installed and configured.

Note: Be sure that you have configured the ranging service server board as Bluetooth Low Energy peripheral, as described above.

The application is in the form of a Python script located in: <app_folder>\python\Scripts\record_range_measurement.py.

To check that everything is installed properly, enter:

```
python record_range_measurement.py --help
```

It should display command usage. If any Python issue shows up, revisit the installation section.

Application parameters

The --help option provides most of the explanation regarding command-line options.

See additional information below:

- The application attempts to detect boards automatically. However, it is possible to force which COM port to connect to using --comport option.
- By default, the application attempts to detect two boards: one acts as a client, and another one acts as the server. Using -R option allows you to run the application with a single board (client) connected to the PC. Once the client is configured, it can be moved freely without being connected to the PC, as shown in the figure.
- The client always acts as the Bluetooth Low Energy central role. Additionally, by default, the client acts as the CS initiator (CS role can be changed using --role option).
- The application prints results on the screen. However, using --target option (can be specified multiple times), it is also possible to generate output files in the following formats:
  - Text file (*.txt) that contains only the raw data exchanged between firmware and Python script.
  - Numpy format (*.npz) that can be loaded using Python numpy library. It contains both raw data and results of distance estimations.
  - Matlab format (*.mat) that can be loaded directly into Matlab. It contains both raw data and results of distance estimations.
  - CSV file (*.csv) that contains distance estimation results dumped for each measurement.

Parent topic: Using Python host application

Performing CS measurements with only step mode 2's

The following command triggers a series of 5 CS measurements using RTP:

```
python record_range_measurement.py --mainmode 2 -n 5
```

The above command uses default CS parameters configured in the firmware.

To change some parameters, --dutconfig arguments can be added to the command.

For example to select 2 Mbit/s PHY, use:

```
--dutconfig "param rtt_phy 1"
```

To enable antenna diversity 2x2 with SMA connectors from antenna diversity board, use:

```
--dutconfig "param ant 7" --dutconfig "param ant_type 1" --dutconfig "param timings t_sw 2"
```

See *Embedded firmware command reference* for the exhaustive firmware configuration commands that can be passed through the --dutconfig option.

The output of the script shows:

- Several miscellaneous information, such as serial ports used, configuration, and so on:

```
2024-12-04 10:44:40 [INFO] __main__: Working on win32
2024-12-04 10:44:40 [INFO] ranging_platform_pkg.utils: Configuring RangingPlatform_kw45 as peer on port:␣
↪COM26 with unique_id: 0x000000000000
2024-12-04 10:44:41 [INFO] ranging_platform_pkg.utils: Configuring RangingPlatform_kw45 as dut on port:␣
↪COM27 with unique_id: 0x000000000000
2024-12-04 10:44:43 [INFO] ranging_platform_pkg.utils: Starting measurement with 1 parameters and 5␣
↪measurements
2024-12-04 10:44:43 [INFO] ranging_platform_pkg.utils: COM27:   central initiator
```

- A series of measurements returning estimated distance, initiator RSSI, reflector RSSI, and measurement duration:
- Measurements statistics (average and standard deviation):

```
===============Measurement stats =======================
Results for board pair (1, 1):
RTP: avg=0.29, stdev=0.00, min=0.29, max=0.29
RADE: avg=0.29, stdev=0.00, min=0.29, max=0.29
==========================================================
```

Depending on the --target option selected, different output files are generated in <app_folder>\python\Scripts\records\ folder.

Parent topic: Using Python host application

Performing distance estimation using mode 2's as main mode and mode 1 as submode

The following command triggers a series of 5 CS measurements using RTP/RTT:

```
python record_range_measurement.py --mainmode 2 --submode 1 -n5
```

Expected output is as follows where SRDE and RTT distance are computed:

```
2021-11-18 09:49:05 [INFO] record_range_measurement:␣
↪==============================================================
2021-11-18 09:49:05 [INFO] ranging_platform_pkg.utils: - parameter0 - meas_nr has 5 values
2021-11-18 09:49:05 [INFO] ranging_platform_pkg.utils: Iteration -   0/5   @(meas_nr = 00) => RTP/RTT-
↪distance = 12.47m/13.56m (100%), RSSI = -33dBm/-47dBm, Duration = 335ms
```

Parent topic: Using Python host application

Configuring embedded algorithms

The following command triggers CS measurements using the embedded CDE + RADE algorithms and RTP/RTT with minimal verbosity output:

```
python record_range_measurement.py --mainmode 2 --submode 1 --algo 0,5
```

```
2024-12-04 11:03:51 [INFO] __main__: Working on win32
2024-12-04 11:03:51 [INFO] ranging_platform_pkg.utils: Configuring RangingPlatform_kw45 as peer on port:␣
↪COM26 with unique_id: 0x000000000000
2024-12-04 11:03:52 [INFO] ranging_platform_pkg.utils: Configuring RangingPlatform_kw45 as dut on port:␣
↪COM27 with unique_id: 0x000000000000
2024-12-04 11:03:54 [INFO] ranging_platform_pkg.utils: Starting measurement with 1 parameters and 16␣
↪measurements
2024-12-04 11:03:54 [INFO] ranging_platform_pkg.utils: COM27:   central initiator
2024-12-04 11:03:54 [INFO] ranging_platform_pkg.utils: - parameter0 - meas_nr has 16 values
2024-12-04 11:03:55 [INFO] ranging_platform_pkg.utils: Iteration -   0/16   @(meas_nr = 00) => CDE-
↪distance =  0.71m RADE-distance =  0.87m RTT-distance =  0.70m (100%), Duration = 493ms
2024-12-04 11:03:55 [INFO] ranging_platform_pkg.utils: Iteration -   1/16   @(meas_nr = 01) => CDE-
↪distance =  0.71m RADE-distance =  0.87m RTT-distance =  1.30m (100%), Duration = 484ms
2024-12-04 11:03:56 [INFO] ranging_platform_pkg.utils: Iteration -   2/16   @(meas_nr = 02) => CDE-
↪distance =  0.70m RADE-distance =  0.87m RTT-distance =  0.30m (100%), Duration = 483ms
2024-12-04 11:03:56 [INFO] ranging_platform_pkg.utils: Iteration -   3/16   @(meas_nr = 03) => CDE-
↪distance =  0.70m RADE-distance =  0.87m RTT-distance =  0.10m (100%), Duration = 480ms
2024-12-04 11:03:57 [INFO] ranging_platform_pkg.utils: Iteration -   4/16   @(meas_nr = 04) => CDE-
↪distance =  0.70m RADE-distance =  0.87m RTT-distance = -0.60m ( 92%), Duration = 450ms
2024-12-04 11:03:57 [INFO] ranging_platform_pkg.utils: Iteration -   5/16   @(meas_nr = 05) => CDE-
↪distance =  0.70m RADE-distance =  0.87m RTT-distance = -0.30m (100%), Duration = 479ms
2024-12-04 11:03:58 [INFO] ranging_platform_pkg.utils: Iteration -   6/16   @(meas_nr = 06) => CDE-
↪distance =  0.71m RADE-distance =  0.87m RTT-distance =  0.60m (100%), Duration = 451ms
2024-12-04 11:03:58 [INFO] ranging_platform_pkg.utils: Iteration -   7/16   @(meas_nr = 07) => CDE-
↪distance =  0.71m RADE-distance =  0.87m RTT-distance =  0.70m (100%), Duration = 451ms
2024-12-04 11:03:59 [INFO] ranging_platform_pkg.utils: Iteration -   8/16   @(meas_nr = 08) => CDE-
↪distance =  0.72m RADE-distance =  0.87m RTT-distance =  0.70m (100%), Duration = 479ms
2024-12-04 11:03:59 [INFO] ranging_platform_pkg.utils: Iteration -   9/16   @(meas_nr = 09) => CDE-
↪distance =  0.71m RADE-distance =  0.87m RTT-distance = -0.40m (100%), Duration = 480ms
2024-12-04 11:04:00 [INFO] ranging_platform_pkg.utils: Iteration -  10/16   @(meas_nr = 10) => CDE-
↪distance =  0.69m RADE-distance =  0.87m RTT-distance =  0.70m (100%), Duration = 479ms
2024-12-04 11:04:00 [INFO] ranging_platform_pkg.utils: Iteration -  11/16   @(meas_nr = 11) => CDE-
↪distance =  0.72m RADE-distance =  0.87m RTT-distance = -1.00m (100%), Duration = 451ms
2024-12-04 11:04:01 [INFO] ranging_platform_pkg.utils: Iteration -  12/16   @(meas_nr = 12) => CDE-
↪distance =  0.72m RADE-distance =  0.87m RTT-distance = -0.60m (100%), Duration = 450ms
2024-12-04 11:04:01 [INFO] ranging_platform_pkg.utils: Iteration -  13/16   @(meas_nr = 13) => CDE-
↪distance =  0.72m RADE-distance =  0.87m RTT-distance = -0.00m (100%), Duration = 449ms
2024-12-04 11:04:01 [INFO] ranging_platform_pkg.utils: Iteration -  14/16   @(meas_nr = 14) => CDE-
↪distance =  0.72m RADE-distance =  0.87m RTT-distance = -0.10m (100%), Duration = 480ms
2024-12-04 11:04:02 [INFO] ranging_platform_pkg.utils: Iteration -  15/16   @(meas_nr = 15) => CDE-
↪distance =  0.72m RADE-distance =  0.87m RTT-distance =  1.00m (100%), Duration = 481ms
```

Parent topic: Using Python host application

Troubleshooting

The table below lists some errors that may result from incorrect installation or configuration.
Table: Python software errors and solutions

Error	Possible cause
[ERROR] main : bad magic number in 'range _ estimator_pkg': b'3\r\r\n'	You are not using the correct version of Python. See "Python host application setup".
[WARNING] RangingPlatform_Utils: Could not connect to RangingPlatform: kw47 on port: COM10	The COM port is busy with another software
[ERROR] main : DLL load failed: The specified module could not be found	Must install Visual C++ Redistributable for Visual Studio 2015, 2017 and 2019 (https://www.microsoft.com/en-us/download/details.aspx?id=48145)

Parent topic: Using Python host application

Parent topic:*Running wireless ranging demo application*

Embedded firmware command reference

This chapter provides details about CLI commands supported by the firmware.

Firmware shell commands

The firmware can be controlled with a shell accessible using the serial terminal. This chapter describes the command-line interface to secure proximity devices along with common command syntax and response format.

Note: The shell does not show any prompt.

Parent topic:*Embedded firmware command reference*

Command syntax

The command line interface (CLI) is text-based, that is, only printable characters are used. The user can interact with the board using some commands such as a regular command prompt. This process requires a serial terminal application such as Tera Term/Putty. Line processes commands, that is, all information needed to process a command is contained in one line. Lines are terminated with the "line-feed" character. A command consists of one or more space separated words where each word, except the first, can be a command, submenu, or value (of any type; integer/hex/character/string). The first word cannot be a value. All commands are constructed with menus, which consist of commands and/or submenus. Commands can have zero or more value arguments.

Implicit commands

The command-line processor has two implicit commands:

- "help" or if no <name> is specified – this command shows all available commands/submenus in the selected menu.
- "show" – this command shows the current values of all readable items in the selected menu.

Parent topic: Command syntax

Command response

The device does not echo back the commands given to it. However, the device can respond to each command received immediately and with a single line, except the "range", "help", and "show" commands. The "range" commands also have a delayed response.

The response for single-line commands is "<last-subcommand>: <value>". All responses are terminated with one "line-feed" character.

Parent topic: Command syntax

Boot message

When the device boots, it displays the following information:

- Software version number of the application
- Git hash and build date
- The reset reason of the device, for example, PIN, SW, WATCHOG

Parent topic: Command syntax

Parent topic:*Embedded firmware command reference*

Main menu

By sending the "help" command, a help menu gets listed (see below). Commands are provided to initiate a ranging measurement, display, and modify parameters.

```
-----------commands-------------
range [<ble-address>] | Run HADM procedure measurement with given peer
test [<tx,rx>] | Run HADM test mode measurement
setup [<ble-address>] | Run HADM setup (from Central only)
communication [options] | Communication commands
parameter [options] | Parameter commands
system [options] | System commands
misc [options] | Miscellaneous commands
-------------end----------------
```

The options displayed are explained in the table below:

Table: Wireless ranging top level commands

Command	Description
range	Execute a CS measurement using the Bluetooth Low Energy connection (implicitly set service role of the board as "client"). Limitation: should be issued from the same board after first invocation
test	Execute a CS measurement using CS test either as initiator (TX) or reflector (RX)
parameter	Submenu to set/display CS parameters
communication	Set Bluetooth Low Energy role (central/peripheral)
system	Submenu to set/display system-related parameters
misc	Submenu to set/display miscellaneous parameters

"parameter" - CS parameters

The *CS parameters menu* lists and allows modification of CS configuration. The options displayed here are explained in the table below.

```
-----------commands-------------
ch_list | Channel list commands (test mode only)
ch_map [<bitmask> [<repeat>]] | Set/show channel map as 10 hexa octets (79 bits used)
cs_algo [<0|1> [<0|1>] [<2...8>]] | Set/show channel selection algo type (0=3b, 1=3c), and for #3c: shape␣
↪(0=hat, 1=X) and jump
```

(continues on next page)

(continued from previous page)

```
main_mode_nb [<1...160> <1...160> <0...3>] | Set/show numbers of HADM main mode steps (min, max,
↪repetition)
mode0_nb [<1...3>] | Set/show number of HADM mode0 steps
mode_type [<1...3> [<1...3>]] | Set/show main mode and sub-mode types
role [<initiator,reflector>] | Set/show HADM role
rtt_phy [<0|1>] | Set/show RTT phy rate (0=1Mbps, 1=2Mbps)
tx_pwr [<-12...MAX>] | Set/show TX power in dBm of CS measurement. Range is -12dBm to max chip
↪capability
debug [<0-255>] | Set/show debug options (REPORT_DBG_INFO<2>, IQ_AVERAGE_DISABLE<1>,
↪IQ_DETAILS<0>)
ant_cfg [<0|7> [<0...23|255>]] | Set/show antenna config index and (test mode only) permutation index
ant_type [<0-5> ] | Set/show antenna board type: 0:none, 1:X-FR-ANTDIV SMA, 2:X-FR-ANTDIV printed,
↪3:LOC SMA, 4:LOC printed, 5:LOC printed SMA1 dummy
pn_seq [<AA initiator> [<AA reflector>]] | Set/show PN sequences (must be a hex number, test mode only)
tone_ext [<0...4>] | Set/show TP tone extension (test mode only)
timings | Timings commands
-------------end----------------
```

Table: Wireless ranging CS parameters commands

Command	Default	Description
ch_list	-	Used to define the CS channel list in test mode
ch_map	All ones	Used to define the CS channel map in connected modeUse “param ch_map 0x1FFFFFFFFFFFFC7FFFFC” to mask advertising channels
cs_algo	0	Used to define channel selection algorithm type (0=3b, 1=3c), and for #3c: shape (0=hat, 1=X) and jump
main_m	4, 4, 0	Numbers of CS main mode steps (min, max, repetition)
mode0_	2	Set/show numbers of CS mode 0 steps
mode_t	2,1	Set/show main mode and submode types as defined by CS**Note that the step mode 3 is only supported on KW47 and KW43 platforms.**
role	Initiator	Set/show CS role
rtt_phy	0	Set/show RTT PHY rate (0=1 Mbit/s, 1=2 Mbit/s)
tx_pwr	0	TX power in dBm used during the CS measurement;[-12...10] dBm
pn_seq	-	Set/show the PN sequences to be used in test mode
tone_ex	-	Set/show the tone extension behavior in test mode
timings	-	Timings submenu (see below); Only for test mode
debug	0	For internal debug purpose
ant_cfg	0, 0	Antenna configuration index as defined in CS specs (0:1x1, 7:2x2) followed by a permutation index (0-23 for fixed permutation index and 255 for rolling over all permutation indexes)
ant_typ	0	Antenna diversity board to be used (0:None, 1: X-FR-ANTDIV SMA connectors, 2: X-FR-ANTDIV printed antennas, 3: LOC SMA, 4: LOC printed, 5: LOC printed + SMA1 for dummy antenna)

“ch_list” parameter

The “ch_list” command parameter defines the CS channel list. It only applies in test mode. See for example, the code below:

```
set [<0...78>,..,<0...78> [<repeat>]] | Set/show channel list
generate <0...78> <0...78> [rand] | Generate channel list from first and last CS channels
```

Two modes can be used:

- The "set" mode to input a CS channel list that is input to the firmware.

  ```
  parameter ch_list set 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16
  ```

  This mode defines a CS sequence of 17 steps on CS channels in the given order.

- The "generate" mode makes the firmware generate the channel list based on a start and stop frequency.

  ```
  parameter ch_list generate 0 78
  ```

  This mode defines a CS sequence of 79 steps on CS channels 0 to 78 (linear sweep). This is the default behavior of the firmware.

There is also a possibility to randomize the generated frequency list.

```
parameter ch_list generate 0 78 rand
```

It defines a CS sequence of 79 steps on CS channels 0 to 78 in a pseudo-randomized order.

Parent topic:"parameter" - CS parameters

"timings" – CS timings configuration

```
-----------commands-------------
intervals <150|80|50> <145|80|40> [<145|80|40> ] | Set/show HADM interval timings T_FCS, T_IP1, T_
↪IP2 (us)
t_pm <20|40> | Set/show HADM T_PM timing (us)
t_sw <2|4|10> | Set/show HADM T_SW timing (us)
-------------end----------------
```

Table: Wireless ranging CS timings commands

Command	Default	Description
intervals	150, 145, 145 (test mode)50, 40, 40 (connected)	Set T_FCS, T_IP1 and T_IP2 interval in µs, as defined by CS. Allowed values are: 150 µs, 80 µs, and 50 µs for T_FCS and 145 µs, 80 µs, and 40 µs for T_IP1 and T_IP2.
t_pm	20	Set T_PM, duration in µs, as defined by CS Allowed values are: 20 µs and 40 µs
t_sw	0	When antenna diversity is used, this command controls the antenna switch time which can be set to 2 µs, 4 µs, or 10 µs. Otherwise 0 is used.

Note: The 'Timings' input is relevant for test mode only. In connected mode, the BLE controller chooses timings based on CS capability exchanged between the two devices. NXP controller implementation chooses the fastest timings supported by the two devices.

Parent topic:"parameter" - CS parameters

Parent topic: Main menu

"system" – system commands

```
-----------commands-------------
build_info | Show SW build information
debug [0|1] | Enable debug features (DTEST...)
verbosity [<0...255>] | Set/show verbosity flags (1=Info, 2=Debug, 4=MeasurementInfo, 8=MeasurementData,
↪ 16=MeasurementDebug, 32=BoardInfo, 64=Profiling)
```

(continues on next page)

(continued from previous page)

```
unique_id | Show Unique Id
version | Show SW version number
reset | Reset MCU
store | Store all parameters
factory | Reload and store factory defaults
events | Show events
timing | Show BLE timing
baudrate <value> | Set/show baudrate of serial interface (default: 115200, volatile)
output_format | Show output format
calibrate [<MCIQ dist> <ToF dist> [store]] | Set/Show zero-distance calibration, distances are in meters, Q10
-------------end----------------
```

Table. Wireless ranging system commands

Command	Default	Comment
verbosity	13	Level of details of the range response in the form of bits field. Add verbosity flag values to combine them (for instance 12 for Measurement Info and Measurement Data).
unique_	NA	6 bytes unique identifier. Reserved for future use; all zeros for now
version	NA	Application version. For example: v2.0.0
build_in	NA	Build version information (Git hash, and so on)
reset	NA	Reset the CPU
store	NA	Save settings to flash
factory	NA	Revert to default settings
events	NA	Reserved for future use
timing	NA	Reserved for future use
baudrate	1152	Change the baud rate of the UART interface. This setting is volatile, that is, it is restored to the default value after a power cycle or reset.
output_forı	4	Internal usage

Parent topic: Main menu

"misc" – miscellaneous commands

```
-----------commands-------------
xtal_trim [<0...63>] | Set/show XTAL trim value
cw_trim [0|1] | Start/Stop 2440MHz constant wave Tx (used for XTAL trimming)
rtp_algo [<0...7>] | Set/show RTP embedded algorithm bitmap to be run (bit0:CDE, bit2:RADE)
rtp_algo_config [<1(CDE)> [<param1> <param2>]] | Set/show RTP embedded algorithm configuration
```

Table: Wireless ranging miscellaneous commands

Command	Default	Comment
xtal_trir	0	Set/show 32 MHz crystal oscillator trimming value. Combined with cw_trim command, it allows you to adjust Xtal precisely.
cw_trim	-	Turn on/off a 2440 MHz constant wave transmission to tune the Xtal (using xtal_trim)
rtp_algc	1	Set RTP embedded algorithm to be run in bitmap format: 0: No embedded algorithm is runbit 0: The firmware runs the CDE algorithm bit 2: The firmware runs the RADE algorithm
rtp_algc	-	This command can be used to customize some parameters of CDE algorithm. It is for internal usage only.

Parent topic: Main menu

Firmware response interpretation

Each CS measurement produces results represented as a JSON-like structure terminated with "[DONE]".

The "system verbosity" command controls the verbosity of the output.

An example response of a "range" command with default (minimal) verbosity, showing mainly the distance computation results ('cde' and 'ad' fields in meters), without any additional data is shown below:

```
items:[{mciq:{cfg:{n_ap:4,n_stp:79},result:{vf:79,cde:0.64,cqi:0.842},},tof:{cfg:{n_stp:13},result:{ad:0.3,
↪sr:100},},},]
CRC32:1cebcbcb
marker:[DONE]
```

An example response of a "range" command is shown below:

```
items:[{hadm:{cfg:{rtyp:0,rphy:0,txpwr:0,fcs:150,ip1:145,ip2:145,tpm:20,ant:0},
sts:0,stp:{nb:101,md:'000222212222122221222212222122221222212222122221222212222122221222212
2221222212222122221222212222122221222122221222',
ch:'000102000102030304050607070809 0A0B0B0C0D0E0F0F101112131314151617171819 1A1B1B1C1D1E1
F1F2021222324252627272829 2A2B2B2C2D2E2F2F303132333334353637373839 3A3B3B3C3D3E3F3F40414243
4344454647474849 4A4B4B4C4D4E'}},
md0:{cfg:{n_stp:3},init:{r:'626262',c:'fjHQfkSQfh4g'},refl:{r:'808080',c:'gAAAgAAAgAAA'}},
mciq:{cfg:{n_ap:1,n_stp:79},init:{i:[
↪'n6i4YIY6oGncnylGm6agmQiSYKiYagmacGYedckSh+Y8nAmKmKdkhwaMae
meZelimKleaAaEbYkmacaihOkKa0lSk+lEjYiGcejgd0fqkEg8kmbWgodSiAfAcGcSjkj+dsdKjibkgMb8jcjOigdojadShkhof4
↪',],
q:[
↪'iYYUiakMeujYdumSkCaKbGnqfcYolgbwmme6nGmGnGeceijac2Zkmqcej6ecfOjYeedEfOfkjocihYhalecehcgMeah
Ij4kqjkjkbmk4dcbWe8gYkicYkCbuiAiQdoeacYc0imgMbeiQc8jUj8cEjIcSbubwki',],},},tof:{cfg:{n_stp:19},init:
{d:
↪'AyUOAyUVAyUbAyUdAyUkAyUlAyUsAyUwAyU2AyU+AyVBAyVGAyVKAyVOAyVQAyVWAyVcAyVnAyVn
↪',r:'62626161616160605F5F5E5E5D5D5C5C5C5D5D'},},
info:{init:{syn:0,syg:0,syr:0,syc:0,f:0x0000,x:0,ta:0,te:0,},
refl:{syn:0,syg:0,syr:0,syc:0,f:0x0000,x:0,ta:0,te:0,},},},]
marker:[DONE]
```

These responses have an albeit slightly modified YAML syntax. The modifications are: ": " is replaced with ":" and indentation/end-of-lines are removed by using the YAML data stream syntax. This reduces the output size to a minimum.

Online YAML beautifier tools can be used (after adding a space after double-colons) to convert them into a more readable format.

The above example is shown below:

```
items:
  [
    {
      hadm:
        {
          cfg:
            {
              rtyp: 0,
              rphy: 0,
              txpwr: 0,
              fcs: 150,
              ip1: 145,
              ip2: 145,
              tpm: 20,
              ant: 0,
            },
          sts: 0,
          stp:
            {
              nb: 101,
              md: "000222212222122221222212222122221222212222122221
```

Table 1 – conti

Hierarchy	Parameter	Unit	Description
mciq.init/refl	i	-	List of base64 encoded 12 bit PCT I values, that is, fo
mciq.init/refl	q	-	List of base64 encoded 12-bit PCT Q values, that is, f
mciq.init/refl	tqi	-	List of TQI values for each tone; Repeated per anten
mciq.result	cde, cqi	m, #	CDE distance estimation and quality indicator comp
mciq.result	vf	#	Number of valid frequencies retained by distance es
mciq.result	rade, rade_dqi	m, #	RADE distance estimation and quality indicator com
ToF (RTT) results			
tof.cfg	n_stp	#	Number of steps containing RTT packets (step mode
tof.init/refl	d	ns	List of base64 encoded 18 bits + 4 error bits ToA-ToD
tof.init/refl	r	dBm	Mode 1 or 3 packet RSSI array (required, length=n_s
tof.result	ad, sr	m, %	Average distance estimation and success rate as com
tof.init/refl	| nadm	| - |	Estimated chance of an attack on the received packe
CS measurement info			
info.init/refl	syn	#	Mode 0 index in sequence that was used to synchron
info.init/refl	syg	#	AGC index used during whole CS event (<= 11)
info.init/refl	syr	dBm	RSSI of mode 0 used for gain lock
info.init/refl	syc	Hz	CFO of mode 0 used for CFO compensation
info.init/refl	f	-	Error/warning flag (see description below)
info.init/refl	te	-	Reserved for future use

The table below shows the “info” error flags bit field description.

Table: Info error flags interpretation

Bit Mask	Comment
0x0001	PLL lock error
0x0002	CS sequence was aborted
0x0004	Error during AGC lock
0x0008	Error during IQ capture
0x0010	RSSI measured became too low
0x0020	Error happened during Mode 0 synchronization phase
0x0040	Timestamp reading on one or more RTT packets failed
0x0080	SW scheduler detected a desynchronization with RSM HW scheduler
0x0100	XCVR API reported an error

Use the script `record_range_measurement.py` to decode this data and obtain numerical values. We do not provide the encoding method of the compressed fields here.

Table: CSV fields description

Field	Description
meta.error_msg	Application error message.
meta.testcase.meas_nr	Ranging measurement number.
mciq.cfg.n_ap	Number of antenna paths used during CS mode 2 or mode 3 steps. Typically 1 for
mciq.cfg.n_stp	Number of CS RTP steps (mode2 or mode3) included in the CS procedure.
mciq.result.CDE_distance	CDE distance estimate in meters.
mciq.result.CDE_dqi	CDE distance quality indicator between 0 and 1.
mciq.result.RADE	RADE distance estimate in meters.
mciq.result.RADE_dqi	RADE distance quality indicator between 0 and 1.
mciq.result.RADE_error	RADE error status.
mciq.result.distance	Distance estimate in meters.
mciq.result.slope_rmse	IQ samples quality metric in degree. 2-way phase slope root mean square error.
tof.cfg.n_stp	Number of CS RTT steps (mode1 or mode3) included in the CS procedure.

Table 2 – continued from previo

Field	Description
tof.result.distance	RTT distance estimate in meters.
tof.result.successrate	How many CS RTT packets were sucessfully exchanged during the CS procedure, i
tof.result.init_nadm	Initiator Normalized Attack Detector Metric.
tof.result.refl_nadm	Reflector Normalized Attack Detector Metric.
tof.result.std	Standard deviation of the distance computed over all CS mode 1 steps.
md0.cfg.n_stp	Number of CS mode 0 steps (Synchronization) included in the CS procedure.
hadm.cfg.rtyp	Channel sounding RTT Type.
hadm.cfg.rphy	Channel sounding RTT PHY (0 = 1Mbps, 1 = 2Mbps).
hadm.cfg.txpwr	TX power used during the CS procedure, in dBm.
hadm.cfg.fcs	Channel sounding T_FCS, in µs. Frequency change period.
hadm.cfg.ip1	Channel sounding T_IP1, in µs. Defined for the CS mode 0 and CS mode 1 : idle tim
hadm.cfg.ip2	Channel sounding T_IP2, in µs. Defined for the CS mode 2 : idle time between the
hadm.cfg.tpm	Channel sounding T_PM, in µs. Phase measurement period (CS mode 2).
hadm.sts	Channel sounding measurement status.
hadm.stp.nb	Channel sounding number of executed steps.
hadm.stp.modes	The list of Channel Sounding modes used for each step.
hadm.stp.channels	The list of Channel Sounding channel numbers used for each step.
hadm.stp.event	Index of last event number.
hadm.stp.subevt	Index of last subevent number.
hadm.stp.m0_idx	The list of indexes for the steps that use mode 0 procedures.
hadm.stp.tof_idx	The list of indexes for the steps that use ToF procedures.
hadm.stp.mciq_idx	The list of indexes for the steps that use phase based procedures.
info.init.sync_step_id	Initiator side. Which step index has been used during the synchronization (CS mo
info.init.sync_agc	Initiator receiver gain index selected for the CS procedure.
info.init.sync_rssi	Initiator RSSI in dBm.
info.init.sync_cfo	Initiator measured Carrier Frequency Offset before compensation, in Hz.
info.init.flags	Initiator error flags (refer to SDK documentation 'CS wireless ranging demo applic
info.init.temperature	Temperature of initiator device during measurement.
info.refl.sync_step_id	Reflector side. Which step index has been used during the synchronization (CS m
info.refl.sync_agc	Reflector receiver gain index selected for the CS procedure.
info.refl.sync_rssi	Reflector RSSI in dBm.
info.refl.sync_cfo	Reflector measured Carrier Frequency Offset before compensation, in Hz.
info.refl.flags	Reflector error flags (refer to SDK documentation 'CS wireless ranging demo appli
info.refl.temperature	Temperature of reflector device during measurement.

Parent topic: Main menu

Parent topic:*Embedded firmware command reference*

Appendices

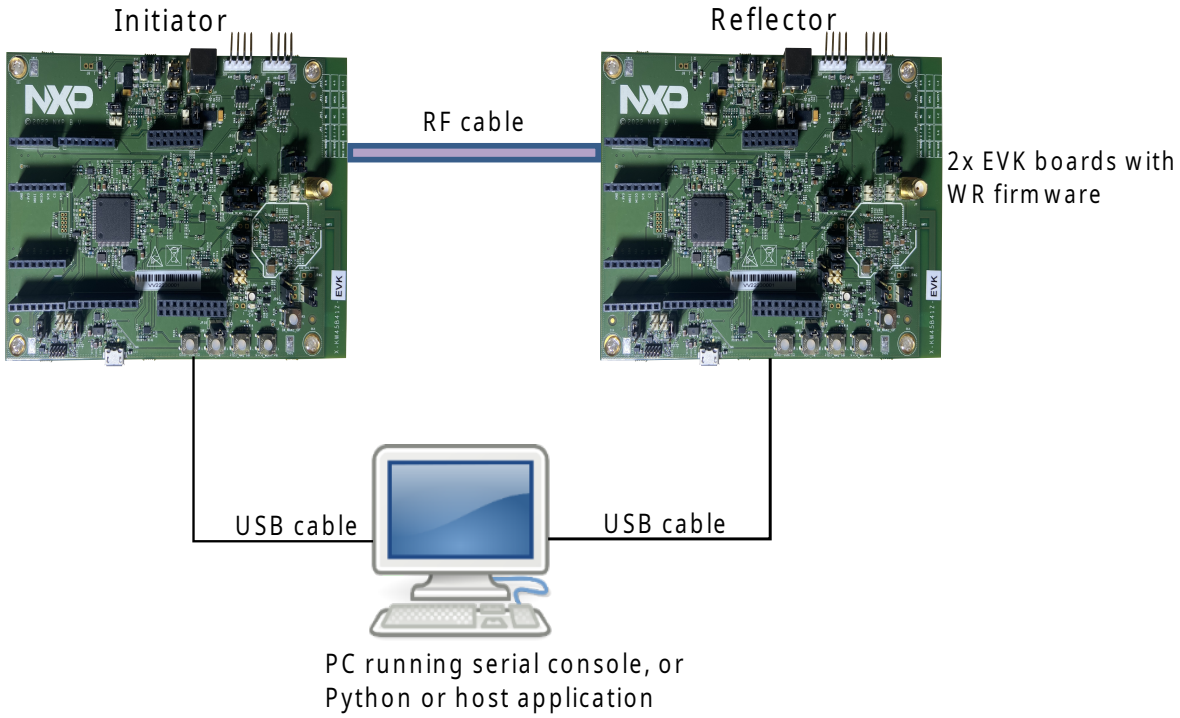
This chapter provides some additional information about the following: - Test mode hardware setup and operation- Wireless ranging and channel sounding standardization.

Test mode

There is an alternate way to test CS feature that does not require a BLE connection to be established between the boards. This is called "test mode", and it relies on the LE_CS_Test HCI command from the standard.

Hardware setup

This mode requires that both boards are connected to the controller PC to collect and combine data from both initiator and reflector.



Parent topic: Test mode

Operating test mode

This chapter provides information on how to use wireless application in test mode.

Using Python host application

When using test mode from Python host application, the following parameter must be added to the command:

```
--testmode
```

Parent topic: Operating test mode

Using the console

To start the measurement on the reflector side:

```
\> test rx
```

To start the measurement on the initiator side:

```
\> test tx
```

Both devices synchronize to each other and perform a CS test procedure using default settings.

The expected output (on both devices) is a JSON-like structure similar to the following:

```
items:[{hadm:{cfg:{rtyp:0,rphy:0,txpwr:0,fcs:150,ip1:145,ip2:145,tpm:20,ant:7},sts:0,stp:{nb:101,md:
↪'0002222122221222212222122221222212222122221222212222122221222212222122221222212222122221222212222122221222
↪',ch:
↪'000102000102030304050607070809 0A0B0B0C0D0E0F0F101112131314151617171819 1A1B1B1C1D1E1F1F202122232324252627
↪',evt:'00',se:'65'}},md0:{cfg:{n_stp:3},init:{r:'808080',c:'800080008000'},refl:{r:'5A555D',c:'7FFF7FFF7FFF
```

(continues on next page)

(continued from previous page)

```
↪'}},mciq:{cfg:{n_ap:4,n_stp:79},refl:{i:[
↪'lMjijwmvkNmoZOjCmxkglIcvkIZebvgqiMnAZGj1eFb+l1ZbjSkIljmMhDcgmEe9mQbNfTegk8azh1aWfFcvmEdobceLe8iEizfle1l
↪','kBgKgOj6gzkacUjpkUkJhmf4g4cEcGiKfQj0c9kChpgDh6b/kik4iJjjeDbakFhKlCdVeKgBkHdPiMb/
↪gqfOkJf/eIdVd0iogzgxeLkEd9djhHeTdtfLcmhNgafUcuh/fKfcjfd9fyjUcvgufVi/iyiudXgwiT',
↪'XVmVmLdZlUbKhgWvb+XfkDZcltiaohXLnMdHgiYiYzakifhdZcaIjRhtmolXiSZIgtaanBYicyaoZVheYTYofaYWZhmdm9Zon9X
↪nZeZYhZUYam3l3YVeYY6nVY8efaNmjZZgnksmMhDfQaRlvihe3e5hglOkE','w6X7X8oYaZr/
↪Y4xKrBwtcyoVZyX1PYvAVKpBa6wTr5m6fyX4vovEeyijTtRXiksimrkiRqsJs/
↪i0u+UtsPoWpfqRlMRzR9uQW5srSEq8mSkUtNSaSlpUaXsMW2n+bdr9VFoMmPU8WdlVaWp+WUimnXneYZY8dl
↪',],q:['bGmDlwgNlUegfHZ3exa1kfaMlMf5lHZSmlfnepaKZdaojTgVaWa/
↪jThwmXlYh9ZwffcCmLaDcYcwaIiYZ7ateQaJbtl+mIaHlpZzmFdAbVb4ZymIltaTgVZvluaxfiaGl7bBeQlbldeLiBailXfZcdcdjtjthX
↪',
↪'ezkUkdiSkfhFdcdYhUeUkNbhkceDh3cYkNh6dEebb6bdkVdZeKeukvj6lBiyjea2hhbpkobVeRc4cfgkcCcBgWbzcKjdj2cgkXbvj7e1
↪',
↪'bvY4YwW4YMX6pZeCXecAXmmpYnpAjeh1bBYCoSc4jXloY2nUdCcHZ9ZffTkGZpgGZDkiiNgZZNlYcYnifpieYRhNkJkUjpbU
↪cIiAg1ZikchXZ0mPdniYafaQadlRfxdI',
↪'kJvVvUvHwTsbQVfKtDjLwpRRvzRPdlYsslt6QViLUyRYwDSZhMjivNu1oYeVujYHtaRvkgXHnpRjePWZXmT5rIVzSmhuive6
↪',],tqi:[
↪'00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
↪',
↪'00000000000000000000000000000000000000000000000200000000000000000000000000000202020000000000000000000000000000000000
↪',
↪'00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
↪',
↪'00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
↪',],},},tof:{cfg:{n_stp:19},refl:{d:'AP+AAP+EAP+NAP+YAP+hAP+qAP+vAP/BAP/DAP/RAP/YAP/
↪kAP/oAP/yAAAAAAAIAAASAAAVAAAl',r:'625A565C6159585A6159565B6059545A5E5853',nadm:
↪'FFFFFFFFFFFFFFFFFF'},},info:{init:{},refl:{syn:0,syg:0,syr:0,syc:0,f:0x0000,x:0,ta:0,te:0,},},},]
CRC32:17002f48
marker:[DONE]
```

It contains all the CS results returned locally by HCI events and debug data, but does not show the final distance. In this mode, the host PC can only compute the final distance.

See Command response for how to interpret these results.

Parent topic: Operating test mode

Parent topic: Test mode

Parent topic:*Appendices*

Wireless ranging and channel sounding standardization

In this chapter, we are highlighting which part of wireless ranging application complies with a standard and which one does not.

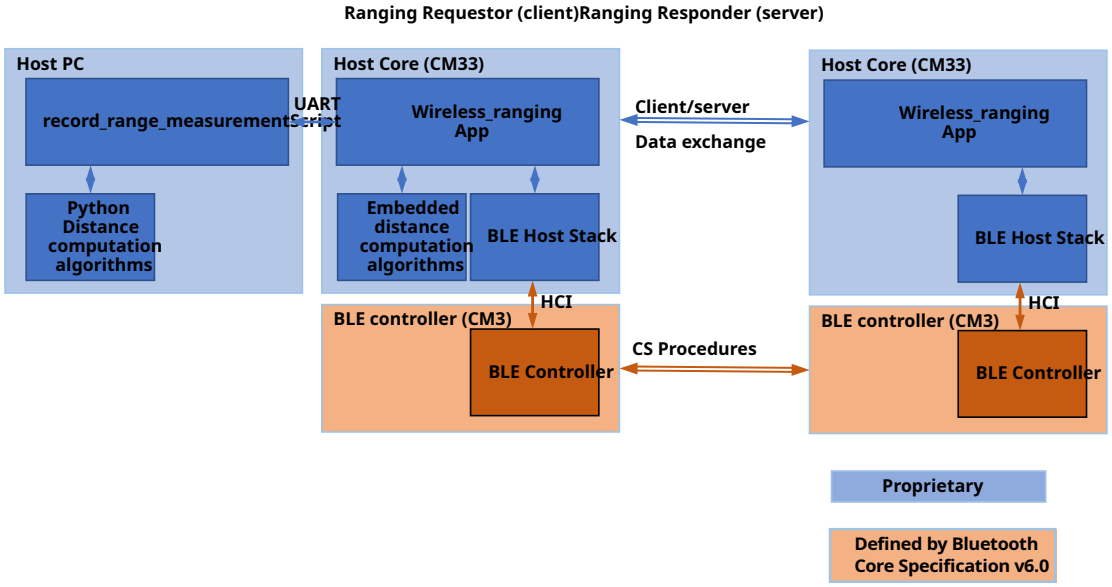
The Bluetooth Low Energy SIG defines two standards:

- Channel sounding is defined by the core working group. It defines the behavior of the Bluetooth Low Energy controller and how to interface it for CS features. The specification covers how the CS feature must be implemented in a Bluetooth Low Energy controller.
- Ranging service and profile (RASP) is defined by the “Direction Finding” working group. It defines an architecture (client/server) and a Bluetooth Low Energy profile for applications that require to operate Channel Sounding feature.

The Bluetooth Low Energy SIG *does not* define any sort of distance computation method. Wireless ranging application provides two sample algorithms to demonstrate the full ranging concept. The first is CDE, an embedded implementation with low complexity. The second one is RADE, which provides an essential compromise between complexity and performance. As mentioned previously, wireless ranging Python framework ensures that a user can develop their own signal processing algorithm using the real IQ datasets.

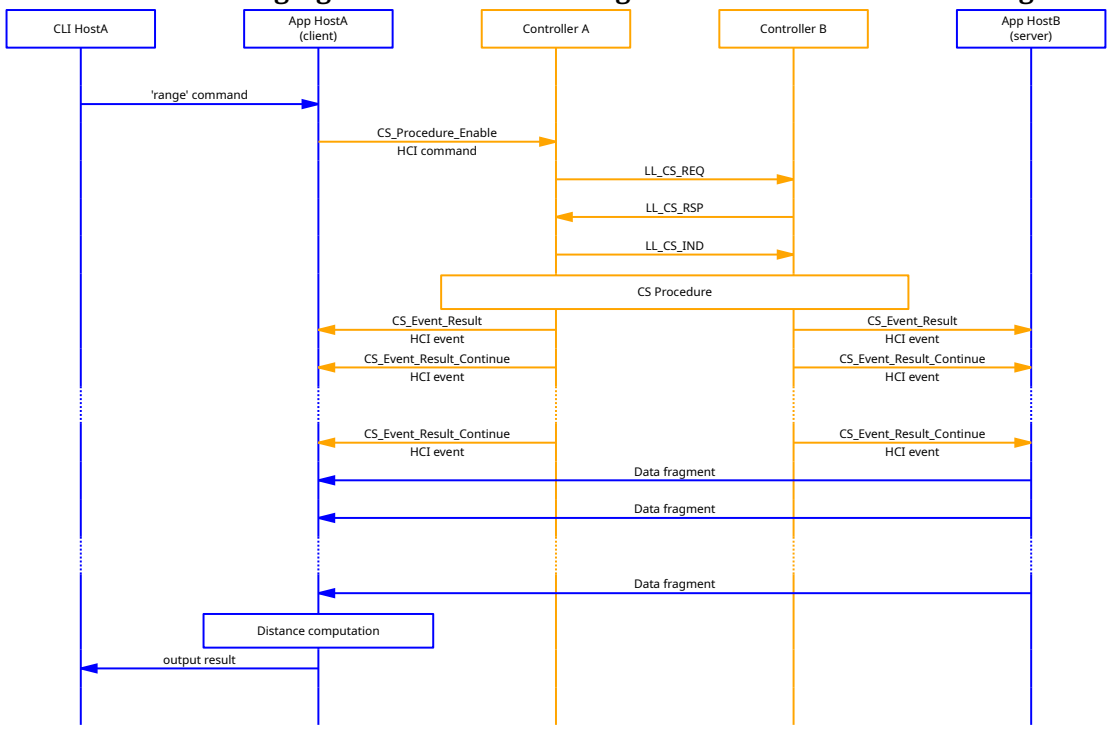
Wireless ranging is based on channel sounding core specifications, as mentioned in the associated SDK release note. It does not implement RASP specification. The figure below shows the application architecture with a colored overlay of what is defined by the standards.

Figure: Wireless ranging software architecture and channel sounding overlay



Similarly, the figure below shows a sequence chart highlighting a single measurement sequencing with colored overlay. The steps defined by the standard are highlighted in orange.

Figure: Wireless ranging software flow diagram and channel sounding overlay



Parent topic:*Appendices*

Acronyms

The table below lists the terms and acronyms used in this document.

Term	Definition
Measurement bias	Constant offset observed between estimated distance and real distance
Central/Peripheral	Refer to BLE connection roles – the central must be connected to a PC
Initiator	Device initiating the measurement and collecting data
Reflector	Device responding to the initiator during measurement phase
Multi channel IQ (MCIQ)	Method to capture IQ samples of data exchanged between two devices on multiple channels separated by a fixed increment
Time of flight (ToF)	Method to capture round-trip time between two devices and estimate distance based on speed of light
Complex-domain distance estimation (CDE)	Phase-based distance estimation algorithm
Multiple signal classification (MUSIC)	Phase-based distance estimation super-resolution algorithm
RTT	Round-trip time
RTP	Round-trip phase; synonym for MCIQ
HADM	High accuracy distance measurement
CS	New name for HADM
AGC	Automatic gain control
CFO	Carrier frequency offset
RSSI	Receive signal strength indicator
PCT	Phase correction term (I&Q)
WR	Wireless ranging application
RADE	Rapid and accurate distance estimation

Parent topic:*Appendices*

Parent topic:*CS Wireless Ranging Demo Application User Guide*

Related documentation

Table below lists the additional documents and resources that can be referred to for more information. The documents listed below are available only under a non-disclosure agreement (NDA). To request access to these documents, contact local field applications engineer (FAE) or sales representative.

Document	Link/how to access
Bluetooth Channel Sounding Technique for Distance Estimation (document AN13481)	Contact NXP local field applications engineer (FAE) or sales representative
KW47-EVK Board User Manual (document KW47-EVKUM)	Contact NXP local field applications engineer (FAE) or sales representative
Antenna Diversity Board User's Guide (document ADBUG)	Contact NXP local field applications engineer (FAE) or sales representative
KW47-EVK Updating Secure Binary NBU Image User's Guide	Contact NXP local field applications engineer (FAE) or sales representative
KW47-LOC Board User Manual (document KW47-LOCUM)	Contact NXP local field applications engineer (FAE) or sales representative
KW43-LOC Board User Manual (document KW43-LOCUM)	Contact NXP local field applications engineer (FAE) or sales representative

Note about the source code in the document

Example code shown in this document has the following copyright and BSD-3-Clause license:

Copyright 2023-2025 NXP Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Revision history

The table below summarizes the revisions to this document.

Revision number	Date	Substantive changes
2.2	22 July 2025	Removed KW45 platforms support
2.1	10 April 2025	Converted to Markdown format
2.0	13 January 2025	Updated for MCUXpresso SDK 24.12.00 release
1	26 June 2023	Updated multiple sections for KW45 MR2 SDK
0	10 October 2022	Initial release

Wireless Framework

Wireless Connectivity Framework Connectivity Framework repository provides both connectivity platform enablement with hardware abstraction layer and a set of Services for NXP connectivity stacks : BLE, Zigbee, OpenThread, Matter.

The connectivity framework repository consists of:

- Common folder to common header files for minimal type definition to be used in the repo
- Platform folder used for platform enablement with Hardware abstraction:
  - platform/include: common API header files used by several platforms
  - platform/common: common code for several platforms
  - specifics platform folders , See below the supported platform list
  - platform/../configs folder: configuration files for framework repository and other middlewares (rpmsg, mbedTls, etc.._)
- Services folder
- Zephyr folder for zephyr modules integrated in mcux SDK
- clang formatting script and script folder to format appropriately the source files of the repo

Supported platforms

The following devices/platforms are supported in platform folder for connectivity applications:

- kw45x, k32w1x, mcxw71x, under wireless_mcu, kw45_k32w1_mcxw71 folders.
- kw47x, mcxw72x families under wireless_mcu, kw47_mcxw72, kw47_mcxw72_nbu folders.
- rw61x
- RT1060 and RT1170 for Matter
- Other RT devices such as i.MX RT595s

Supported services

The supported services are provided for connectivity stacks and their demo application, and are usually dependent on PLATFORM API implementation:

- DBG: Light Debug Module, currently a stubbed header file
- FSCI: Framework Serial Communication Interface between BLE host stack and upper layer located on an other core/device
- FunctionLib: wrapper to toolchain memory manipulation functions (memcpy, memcmp, etc) or use its own implementation for code size reduction
- HWParameters: Store Factory hardware parameters and Application parameters in Flash or IFR
- LowPower: wrapper of SDK power manager for connectivity applications
- ModuleInfo: Store and handle connectivity component versions
- NVM: NXP proprietary File System used for KW45, KW47 automotive devices and RT1060/RT1170 platform for Matter
- OtaSupport: Handle OTA binary writes into internal or external flash.
- SecLib and RNG: Crypto and Random Number generator functions. It supports several ports:
  - Software algorithms
  - Secure subsystem interface to an HW enclave
  - MbedTls 2.x interface
- Sensors: Provides service for Battery and temperature measurements
- SFC: Smart Frequency Calibration to be run from KW47/MCXW71 from NBU core. Matter related modules:
- OTW: Over The Wire module for External Transceiver firmware update from RT platforms
- FactoryDataProvider to be used for Matter

Supported Zephyr modules integration in mcux SDK

Connectivity framework provides integration and port layers to the following Zephyr Modules located into zephyr/subsys:

- NVS: Zephyr File System used by Matter and Zigbee
- Settings: Over layer module that allows to store keys into NVS File System used by Matter Port layer and required libraries for these zephyr modules are located in port and lib folder in zephyr directory

Connectivity framework CHANGELOG 7.4.3 mcux SDK 26.09.00 RFP

Major Changes

- [platform][DBG] Moved the NBU software watchdog (handcheck based NBU stuck detection) into the common platform_dbg sources, making the feature available on both KW43/MCXW70 and KW47/MCXW72 without additional files.

Minor Changes

- [wireless_mcu][wireless_nbu][kw43_mcxw70] Moved P256 public key generation and ECDH DH key computation from the NBU controller to the host. Added two ICS message types (ReadP256PublicKey/GenerateDHKey) with their host and NBU handlers, the new PLATFORM_ReadLocalP256PublicKey() and PLATFORM_GenerateDHKey() APIs, and the new gPlatformIcsDeferDHKeyToHost_d feature flag gating all host side DH key code. APIs are kept but empty when the flag is disabled so callers can keep the call unconditional. Also fixed the duplicate ICS enum sentinel gFwkSrvNbu2HostLast_c. gPlatformIcsDeferDHKeyToHost_d is disabled by default on the host to avoid pulling SecLib and the deferred DH key path into all applications; applications requiring this feature (e.g. hci_bb controller qualification) must explicitly build with -DgPlatformIcsDeferDHKeyToHost_d=1. The NBU configuration is unchanged and an assert catches the host/NBU misconfiguration in debug builds.
- [platform][zephyr] Zephyr feature flag overrides are now owned by the framework repo in a per platform configs/fwk_config_zephyr.h, included at the top of fwk_config.h under __ZEPHYR__, removing the dual maintenance with the Zephyr integration. MCUXSDK defaults and package content are unchanged. Platforms: kw43_mcxw70, kw45_k32w1_mcxw71, kw47_mcxw72, mcxw23, rw61x.
- [wireless_mcu][kw43_mcxw70] Added gPlatformNbuDebugGpioDAccessEnabled_d support on kw43_mcxw70 to grant the NBU access to all GPIOs managed by GPIOD, through the new PLATFORM_InitNbuSpecific() API performing the NBU platform specific initialization.
- [DBG] The NBU debug path now preserves the first fault/assert information: NBUDBG_StateCheck() no longer consumes the NBU debug state before a consumer callback is registered, the sys_debug_panic_triggered guard is hoisted in the hard fault handler so capture, host indication and coredump only run for the first fault, and only the first fault/assert is recorded through the new NBUDBG_RECORD_CLAIMED() macro.
- [KW43-LOC][lcl] Updated PLATFORM_InitLcl() to support antenna diversity on the KW43-LOC board.
- [NVS] Replaced memcpy() by HAL_FlashRead() for internal flash reads so that HAL checks and Async Flash mode synchronization are not bypassed.
- [kw43_mcxw70] SecLib_psa_config.h is now only added to the build when the PSA SecLib port is selected, aligned with kw45_k32w1_mcxw71 and kw47_mcxw72.
- [wireless_mcu] Added the missing fwk_config.h as first include in the kw43_mcxw70, kw45_k32w1_mcxw71 and kw47_mcxw72 platform files so that feature flag overrides are taken into account.
- [kw43_mcxw70] Removed the usage of the m_lowpower_flag_(start|size|end) linker symbols.

Bug Fixes

- [WorkQ] Increased the default FWK_SYSWORKQ_STACK_SIZE from 608 to 640 bytes to fix a stack overflow on the system work queue thread.
- [rw61x] Fixed RTOS heap exhaustion when initializing BLE several times (bt init/bt disable loops): the mutex is now created only once in PLATFORM_InitBle(), protected against TOCTOU with interrupt masking, the flag is reset in PLATFORM_TerminateBle() to allow recreation after termination, and asserts are replaced by error code returns to remain correct in Release builds.

- [platform][TSTMR] Fixed the 56 bit version of the timestamp, now reading the TSTMR instance base and testing only the base pointer validity.
- [IW416][coex] Added the missing SD8978 firmware include guard in fwk_platform_coex.c, fixing the compilation error of IW416 coex_edgefast applications.
- [docs] Fixed Sphinx/docutils warnings and errors in the framework documentation: index title underline, leading transitions after README titles, and FSCI C code blocks that could not be lexed.
- [MISRA][CERT-C] Various MISRA, CERT-C and Coverity compliance fixes gathered across platform, LowPower, ICS (wireless_mcu and wireless_nbu), OTA, FSCI, SFC, SecLib and NVM modules: HCI packet length checks before field access, safe SWO prescaler computation, US_TO_TICK/PWR_ConvertUsecToTicks() saturation to TICK_TYPE_MAX, ICS message type re-validation in the Rx work handlers, OTA chunk length and CRC computation hardening, SFC unbalanced low power constraint release guard and 64 bit timestamp arithmetic, SecLib SHA-256 OVERRUN and padding fixes with AES block count overflow checks, NVM union initializer, plus in place annotations for the findings to be dismissed.
- [SFC] Prevent unnecessary SFA measurement retrieval when SFC is disabled. Added a check in the SFC ISR to verify that the SFC module is enabled before retrieving the SFA frequency measurement.

7.4.2 mcux SDK 26.09.00 pvw2

Major Changes

- [NVM] Refactored NVM handling to centralize address computation using offset-based flash access helpers (*_AtOffset) instead of stored flash addresses, reducing unsafe pointer arithmetic. NvUpdateSize() now returns uint16_t, added NV_PartitionBlankCheckAtOffset(), and deprecated NvIsMemoryAreaBlank(). Also fixed a data integrity issue in NvSaveAllDataSetEntry() and various Coverity/CERT-C findings.
- [platform] IFR BLE BD address is now reversed to match the BLE Host stack expectation. The PLATFORM_IFR_BD_ADDR_IS_MSB_FIRST option was removed with the reverse loop reworked accordingly.

Minor Changes

- [DBG] Added host-triggered NBU force fault for coredump capture through new PLATFORM_ForceNbuFault() and NBUDBG_ForceNbuFault() APIs, signaled to the NBU over MCMGR (MCMGR_REMOTE_APP_EVENT_COUNT set to 2 via Kconfig.defconfig for KW47/MCXW72).
- [DBG] Deferred the stall HCI event out of NBUDBG_StateCheck(), which now only sets the halted state and notifies the callback. Added new NBUDBG_SendStallEvent() API to emit the stall vendor event on demand.
- [DBG] Integrated Zephyr coredump capture into the app-core fault handler under CONFIG_DEBUG_COREDUMP.
- [kw43_mcxw70] Added PLATFORM_ReinitCrypto() function for low power support with PSA.
- [settings] Added IAR compiler support for iterable sections and disabled the SETTINGS_NAME_END IAR warning.
- [rw61x][zb] Backported hal_nxp patches using __ZEPHYR__ compile guards to disable dependencies not used in Zephyr.

Bug Fixes

- [wireless_mcu] Fixed FRO6M calibration by replacing FWK_MRCC_TSTMR0_CC/FWK_MRCC_TSTMR0 macros with MRCC_CC/MRCC_MUX from fsl_clock.h and preserving the MUX clock selection in PLATFORM_StartFro6MCalibration().
- [PSA] Fixed Kconfig warning by adding KW43 support in crypto Kconfig.
- [DBG] Guarded FreeRTOS sys_dump_callstack_ext() with INCLUDE_xTaskGetHandle to fix -Werror=unused-function build failures.
- [Coverity] Various Coverity compliance fixes in SecLib (DHKey handling in SecLib_GenerateBluetoothF5KeysSecure()) and OTA (replaced union with structure for callback/argument passing in message buffer).

7.4.1 mcux SDK 26.09.00 pvw1

Major Changes

- [SecLib_RNG] Enabled PSA by default on KW43/MCXW70 platforms.
- [OTA] Introduced gOtaEraseWholePartitionOnInit_d option (KW43 only, disabled by default) to erase the whole OTA partition at start of image transfer.
- [wireless_mcu] Replaced the critical section in PLATFORM_RemoteActiveReq() and PLATFORM_RemoteActiveRel() with a shared mutex to reduce critical section duration. **These APIs must not be called from ISR anymore.**
- [wireless_nbu] Updated the low power callback to check for pending RPMSG buffers so the NBU enters WFI only when a Tx message is pending, reducing main core latency.

Minor Changes

- [SecLib_RNG] Added to RNG_psa.c seed support including WorkQueue-based automatic reseeding gRngEnableAutoReseed_d, NBU seed forwarding via PLATFORM_SendRngSeed(), reseed counter logic gRngMaxRequests_d, and new APIs RNG_NotifyReseedNeeded() and RNG_IsReseedNeeded().
- [OTA] Added a blank check of the OTA partition to avoid unnecessary erase operations and refactored OTA_MakeHeadRoom().

Bug Fixes

- [zephyr][lib][crc] Fixed build conflict with EdgeFast OPN CRC by guarding CRC sources with the Zephyr common framework component check to avoid symbol redefinition.
- [NVM] Fixed offset validation in NvGetEntryFromDataPtr() and corrected the bottom record address calculation in NvGetPageFreeSpace() when gUnmirroredFeatureSet_d is undefined.

7.3.3 mcux SDK 26.06.00

New supported platform : kw43, mcxw70 (EAR: Engineering drop only)

Minor Changes

- [kw43_mcxw70][NVM] Added flash ECC option support for KW43 platforms.
- [kw45_k32w1_mcxw71][kw47_mcxw72] Removed use of SIRCCSR SDK definitions for wakeup by UART0.
- [platform] Added platform abstraction macros PLATFORM_GET_IPSR, PLATFORM_SET_INT_MASK, and PLATFORM_CLEAR_INT_MASK to allow platform-specific customization of IPSR read and interrupt mask functions while maintaining backward compatibility.

Bug Fixes

- [kw43_mcxw70] Enabled gPlatformUseHwParameter_d to prevent generation of a new Bluetooth device address at each reset.
- [kw43_mcxw70] Added gPlatformHasNbu_d compile flag to KW43 platform configuration to declare NBU domain presence.

7.3.2 mcux SDK 26.06.00 pvw2

Major Changes

- [NVM] Enhanced robustness of NVM MIT (Meta Information Tag) operations with improved validation and error handling. Added checksum validation feature controlled by gNvmMetaCheckSum_d compilation switch. Systematically validates MIT fields before use and triggers page switch if corruption detected. Fixed mNvTableSizeInFlash tracking when table entries are modified. Refactored NvWriteRecord() and added NvModuleSwitchPage() for better ECC fault handling. Added NvSetChecksumEnable() API to control feature at runtime. Feature disabled by default.

Minor Changes

- [wireless_mcu][ble] Refactored PLATFORM_SetBleMaxTxPower() API moved from platform file to fwk_platform_ble.c for Zephyr compatibility.
- [wireless_mcu] Modified PLATFORM_GetBDAddr() to return consistent address across calls when gPlatformUseHwParameter_d is disabled.
- [platform] Added FRDMKW43 to supported platform families.
- [Common] Enhanced external flash API with C++ compatibility by adding extern "C" guards.
- [kw43_mcxw70] Relocated PLATFORM_TM_CLK_SELECT macro to fwk_platform_definitions.h for platform-specific configuration.
- [kw43_mcxw70][sensors] Adapted temperature measurement to use FIFO0 instead of FIFO1 as only single FIFO available on this platform.
- [kw43_mcxw70] Updated framework component selections in configuration files.
- [wireless_nbu] Replaced FPGA_TARGET guard with FWK_KW43_MCXW70_NBU_FAMILIES for CPU clock configuration to better reflect target family.
- [kw43_mcxw70] Updated RF switch control and debug signal configurations for bringup support.
- [kw43_mcxw70] Hardware parameters placement switched to IFR mode. Compilation macro gHwParamsProdDataPlacement_c changed from gHwParamsProdDataMainFlash2IfrMode_c to gHwParamsProdDataIfrMode_c.
- [kw43_mcxw70_nbu] Enabled NBU Deep Sleep support. Removed PLATFORM_RemoteActiveReq() and PLATFORM_RemoteActiveRel() implementations as they are not required for this platform.
- [kw43_mcxw70] Enabled TRNG support on kw43_mcxw70 platform.
- [SecLib_RNG] Renamed TRNG_ISR to RNG_TrngIrqHandler() and mapped IRQ handler for platforms using TRNG to this implementation.

Bug Fixes

- [NVM] Fixed initialization procedure in InitNVMConfig() to validate start address and partition_size.

- [wireless_nbu] Fixed resource access issue by reverting TSTMR read restriction on NBU as underlying issue has been resolved.
- [wireless_mcu][wireless_nbu] Fixed timestamp initialization to ensure defined value when tstmrId is out of range.
- [SecLib_RNG] Corrected copyright header in seclib.c.
- [MISRA] Various MISRA and CERT-C compliance fixes in NVM module.

7.3.1 mcux SDK 26.06.00 pvw1

Major Changes

- [SecLib_RNG] Refactored SecLib mutex declaration and made Lock/Unlock functions public. Changed return type from osa_status_t to secResultType_t for SecLib mutex functions and moved mutex Lock/Unlock function declarations to SecLib.h.
- [SecLib_RNG][PSA] Activated PSA opaque execution with s200 and its secure key storage. Switched from PSA transparent mode to opaque mode for all functions except CMAC_LsbFirstInput() which is currently not supported in opaque version. Optimized SecLib_psa_config to fully accelerate all PSA_WANT_KEY_TYPE_ECC_KEY_PAIR functions.

Minor Changes

- [RW61X] Added new API PLATFORM_BlePriorityRemap() with HCI vendor command HCI_CMD_NXP_REMAP_BLE_PRIORITY 0x2A1 to configure BLE ADV and Connection Priority relative to Wi-Fi/OT.
- [DBG] Disabled DTEST signals and GPIO debug for debug target to prevent significant low power current consumption degradation.

Bug Fixes

- [OTA][Coverity] Sanitized pImageOffset parameter in OTA functions to avoid possible overflow.
- [SecLib] Fixed multiplication buffer pointer initialization for segmented ECDH operations. Fixed EC P256 multistep operations using SW legacy library. Fixed ECDH_P256_ComputeDhKeySeg() and ECDH_P256_GenerateKeysSeg() argument checking across SecLib variants. Fixed SecLib_AES_CMAC_PRF_128() behavior for SecLib sss variant that tolerated VK length to be 0. Removed unreachable code from SecLib_HMAC_SHA256_Finish().
- [Platform] Fixed TSTMR timestamp 64 bit read compilation failure when gPlatformTstmr32Bit_d is undefined.

7.2.3 mcux SDK 26.03.00 RFP

Major Changes

- [SecLib] Refactored SecLib functions with improved error handling and naming conventions. Added return status codes for better testing, added SecLib_ prefix for all functions with backward compatibility maintained via #define stubs. Added parameter checks on all functions and improved test coverage for all SecLib flavors.
- [SecLib] Updated CryptoLibSW APIs by removing unused pMultiplicationBuffer argument from Ecdh_ComputeDhKeyUltraFast(), ECP256_GeneratePublicKeyUltraFast(), and ECP256_GenerateKeyPairUltraFast(). Renamed ECP256_GeneratePublicKey() to ECP256_GeneratePublicKeySeg() in legacy implementation. Added new ECP256_GeneratePublicKey() API for SPAKE2+ ComputeL procedure.

Minor Changes

- [FunctionLib] Enhanced FLib_StrLen() to return error value when string size exceeds maximum limit (4096 bytes), adopting strnlen behavior.
- [Common] Cleaned deprecated mbedtls2x Kconfig configurations.
- [wireless_mcu][lcl] Updated FEM API to send configuration to NBU for proper XCVR timing adaptation during Channel Sounding activity. XCVR register backup and restore uses current mechanism with config sent through PLATFORM_NbuApiReq().
- [Kconfig] Fixed KCONFIG dependency loop related to MCUX_COMPONENT_component.wifi_bt_module.config by switching MCUX_COMPONENT_driver.conn_fwloader from select to depends on.
- [SecLib_RNG] Removed support for deprecated devices including FSL_FEATURE_SOC_SIM_COUNT RNG and QN908x platform.
- [FactoryDataProvider] Removed deprecated FactoryDataProvider service as it is no longer used following mbedtls2.x deprecation.

Bug Fixes

- [wireless_mcu] Fixed FRO6M calibration failure in OEM closed lifecycle by replacing inaccessible DWT cycle counter with SysTick timer. The implementation saves/restores SysTick state to maintain FreeRTOS compatibility.
- [wireless_mcu][ble] Removed redundant assert in PLATFORM_SendHciVendorEvent() as invalid parameters are already sanitized with error returned to caller.
- [SecLib] Fixed CMakeLists.txt lib_crypto variant dependency.
- [MISRA] Various MISRA compliance fixes in OTA, flash related files, FSCI, NVM, platform files, and ICS modules. Fixed potential flash blank check issue with unaligned pointers, split OTA_PostWriteToFlash() to match HIS_LEVEL constraint, prevented infinite loop in FLib_StrLen(), and ensured union has consistent non-zero size across compilers.

7.2.2 mcux SDK 26.03.00 pvw2

Major Changes

- [wireless_mcu][ble] Refactored BLE platform into separate core HCI and utilities modules for improved modularity and Zephyr compatibility. Core HCI transport functions remain in fwk_platform_ble.c while utility functions (BD address generation, security, TX power, timestamps) moved to new fwk_platform_ble_utils.c module. This enables minimal HCI-only builds without hardware parameter, RNG, or controller API dependencies.
- [wireless_mcu][ble] Aligned HCI platform APIs with Zephyr driver requirements. PLATFORM_SetHciRxCallback() now returns int instead of void, added new PLATFORM_StartHci() API, and enhanced PLATFORM_InitBle() with initialization guard to prevent multiple initialization attempts.
- [wireless_mcu][ble] Made optional initialization steps configurable in PLATFORM_InitBle() with new flags gPlatformSetBleMaxTxPowerAtInit_d, gPlatformSetSfcConfigAtInit_d, and gPlatformSetWakeUpDelayAtInit_d to enable cleaner integration with Zephyr and minimal configurations.
- [wireless_mcu][wireless_nbu] Added new APIs: PLATFORM_TSTMR_GetBase(), PLATFORM_TSTMR_Enable(), and modified PLATFORM_TSTMR_ReadTimeStamp() to use TSTMR instance ID rather than pointer. Removed dependencies on MCUX component tstmr.
- [SecLib_RNG] Removed legacy SecLib and RNG mbedtls framework implementations now replaced by PSA. Removed deprecated SecLib AES-MMO implementation.

- [RNG] Introduced gRngEnableAutoReseed_d feature flag to control automatic reseeding via WorkQ. Defaults to enabled for platforms with NBU, disabled otherwise. This allows single-core platforms to use RNG without WorkQ dependency when handling reseeding manually.

Minor Changes

- [wireless_mcu][ot] Conditionally exclude MCUXpresso-specific dependencies (board_platform.h, pin_mux.h, SecLib.h, RNG_Interface.h, HWParameter.h) when building for Zephyr, with stubbed IEEE 802.15.4 address generation functions.
- [wireless_mcu] Made TRDC driver dependency optional based on gPlatformNbuDebugGpioDAccessEnabled_d usage, changing from mandatory (select) to optional (imply) in Kconfig.
- [wireless_nbu] Added new NBU APIs PLATFORM_GetSharedMemConfig() and PLATFORM_GetDMemConfig() to retrieve size of SMU/DMEM.
- [NVM] Added Nv_GetPartitionAddressAndSize() API to retrieve NVM partition characteristics.
- [DBG] Added warning ID tracking support with enhanced NBUDBG_StateCheck() to monitor NBU warnings. Added PLATFORM_IsNbuWarningSet() API with callback support for proactive warning monitoring.
- [DBG] Added HCI vendor event transmission support with NBUDBG_ConfigureHciVendorEvent() API to enable/disable debug structure transmission over HCI.
- [DBG] Added coredump support on fault handler with gFaultHandlerCoredumpEnabled_d conditional compilation.
- [DBG] Initialized .debugbuf from flash into sqram_debug_region for GCC builds with INIT_BLE_DEBUG_DAT.
- [Sensors] Added initialization guard to prevent redundant sensor initialization calls with PLATFORM_InitSensors().
- [Platform] Added missing localization dependency to XCVR in Kconfig.
- [Platform] Fixed missing Kconfig dependencies for KW47 core1 variants (KW47B42ZB2, KW47B42ZB3, KW47B42ZB6, KW47B42Z96, KW47B42Z97).
- [MCXW23] Enabled gSecLibUseDspExtension_d to leverage CM33 DSP extensions for ultra-fast cryptographic library.
- [MCXW23] Changed temperature dummy value for sensors to support application enablement.
- [OTA] Improved OTA coverage by allowing static functions to be invoked from unit tests.
- [FSCI] Added UUID retrieval via PLATFORM_GetMCUUid() abstraction in FSCI_ReadNbuVer() API.
- [Common] Made services CMakeLists optional to allow integration of only required services into Zephyr's hal_nxp.
- [Platform] Made timer manager optional with configuration flag.
- [wireless_mcu][lcl] Corrected FEM & COEX API functions with proper NBU wake-up handling and RF_GPO pin conflict checks.
- [wireless_mcu] Consolidated KB() and MB() macro definitions by using fwk_hal_macros.h instead of platform-specific definitions to prevent conflicts and ensure Zephyr compatibility.
- [Common] Added Zephyr RTOS compatibility to fwk_hal_macros.h by including <zephyr/sys/util.h> when __ZEPHYR__ is defined and guarding _DO_CONCAT and CONTAINER_OF macros to prevent redefinition conflicts.

Bug Fixes

- [SecLib_RNG][PSA] Fixed RNG and PSA context release in RNG_Deinit() by invoking mbedtls_psa_crypto_free() and resetting rng_ctx.mRngInitialized to ensure proper reinitialization in RNG_ReInit().
- [NVM] Fixed GetFlashTableVersion() function to prevent recursion
- [NVM] Treated missing error case in NvCopyRecord() if failed to find record.
- [NVS] Fixed blank check procedure to return false (non-blank) when checking a 0 length area and corrected internal flash blank check of unaligned flash data.
- [SecLib] Initialized CMAC context before using it to prevent usage of uninitialized memory.
- [WorkQ] Increased workqueue stack size when coverage is enabled.
- [wireless_mcu] Fixed build issue when ICS WorkQ is disabled.
- [MISRA] Various MISRA and CERT-C compliance fixes in NVM, HWParameter, DBG (including platform dbg), LowPower, SecLib, Platform modules (platform_ics, platform_sensors), FSCI, FunctionLib, wireless_mcu, and Common modules.

7.2.1 mcux SDK 26.03.00 pvw1

Major Changes

- [wireless_mcu][ot] Removed gPlatformUseOuiFromIfr configuration flag. OUI will now be read from IFR by default on mcxw72 platform, with a fallback to the static OUI if no value is found in IFR.
- [wireless_mcu][ble] Enabled gPlatformUseUniqueDeviceIdForBdAddr_d = 2 on kw47/mcxw72 platforms to read the complete BD address from IFR. If no address is found in IFR, the system will fall back to generating one using the RNG method.
- [SecLib] Removed unused cryptographic implementations (SHA1, AES-EAX, AES-OFB) and obsolete FSL_FEATURE_SOC_AES_HW option to reduce code size and complexity.

Minor Changes

- [wireless_mcu][ble] Added callback registration for HCI log. Users can register a callback to log the HCI commands sent/received on the platform using new wrapper API.
- [wireless_nbu][CS][COEX] Added Channel Sounding (CS) coexistence enablement support.

Bug fixes

- [wireless_nbu][lowpower] Fixed NBU low-power timing issue by forcing LPO delay to 0 during active NBU window. The delay is restored to the default parameter right before re-entering low-power mode to guarantee accurate Link Layer timing calculations.
- [platform] Removed PLATFORM_GetClockFreq() API which contained MISRA violations, was unused, and duplicated PLATFORM_GetNbuFreq() functionality.
- [MISRA] Various MISRA compliance fixes in SFC, SecLib, Platform (including platform_ics and platform low power), FSCI, and PDUM modules.

7.1.3 mcux SDK 25.12.00 RFP

Major Changes

- [wireless_mcu][wireless_nbu] Replaced interrupt masking macros with static inline functions PLATFORM_SetInterruptMask() and PLATFORM_ClearInterruptMask() to ensure consistent BASEPRI value handling across all compilers. This addresses compiler-dependent behavior issues with the previous macro implementation.
- [wireless_mcu] Added BASEPRI-based interrupt masking in PLATFORM_RemoteActiveRel() to allow high-priority IRQs while ensuring only IMU0 or thread context can call this function.
- [wireless_mcu] Introduced gPlatformUseHwParameter_d compile flag to allow builds without HWParameter section. When undefined or set to 0, crystal trimming functions conditionally access HWParameters only when required.

Minor Changes

- [kw47_mcxw72] Introduced platform-specific library for KW47/MCXW72 platforms.
- [kw47_mcxw72] Support for additional KW47 phantoms including ZB2/ZB3/ZB6/ZB7 and Z83/Z96/Z97.
- [kw47_mcxw72] Removed SH_MEM_TOTAL_SIZE override as it is now automatically calculated to match rpmsg-lite configuration.
- [wireless_mcu] Set RL_BUFFER_PAYLOAD_SIZE to word-aligned value as expected by rpmsg-lite.
- [wireless_mcu] Added system-generated HCI vendor events capability for debug and diagnostic purposes.
- [DBG] Added debug structure transmission over HCI vendor events with NBUDBG_ConfigureHciVendorEvent() API to enable/disable the feature.
- [DBG] Added debug structure versioning and logging buffer address/size information to debug structure.
- [DBG] Removed 15.4 region from debug structure as not supported on KW47.
- [DBG] Added stack overflow detection for armv8_m_mainline architecture.
- [Platform] Simplified enablement of reset features via pin detection
  - Automatically selects gUseResetByLvdForce_c when gAppForceLvdResetOnResetPinDet_d is enabled.
  - Automatically select gUseResetByDeepPowerDown_c when gAppForceDeepPowerDownResetOnResetPinDet_d is enabled.
- [RNG] Replaced gRngHasSecLibDependency_d compilation switch with gRngUseSecLib_d.
- [mcxw23] Defined gRngIsrPrio_c on the preprocessor to make it global and avoid redefinition warnings.
- [mcxw23] Implemented PLATFORM_ResetCrypto() API called by RNG_reinit/SecLib_reinit.
- [mcxw23] Added support of Timer Manager timestamp with OSTIMER.

Bug fixes

- [wireless_mcu] Fixed race condition in PLATFORM_RemoteActiveRel() by adding verification loop to confirm NBU core execution before releasing power domain.
- [wireless_mcu] Added instruction synchronization barrier (__ISB()) after interrupt re-enable in PLATFORM_RemoteActiveRel() to ensure pending interrupts execute between critical sections.
- [wireless_mcu] Fixed external IO voltage isolation issue during low-power initialization - isolation is now cleared at init to ensure proper behavior.

- [wireless_mcu] Replaced spin-wait loops with event-based synchronization in NBU communication APIs. Added mutex protection to PLATFORM_NbuApiReq() and PLATFORM_GetNbuInfo() to prevent race conditions and deadlocks when multiple tasks call these APIs concurrently.
- [wireless_mcu] Fixed OSA bare metal event race condition in ICS where auto-clear event feature could cause tasks to become permanently stuck. Disabled auto-clear feature in bare metal builds and manually clear event flags after OSA_EventWait() returns successfully.
- [NVM] Fixed NvIdle() to prevent looping for more operations than the queue size.
- [NVS] Fixed blank check procedure to return false (non-blank) when checking a 0 length area.
- [NVS] Made external and internal flash ports consistent.
- [DBG] Fixed debug structure size and callback access issues - corrected memory placement overlap between reg_info and assert_info.
- [MISRA] Various MISRA compliance fixes in NVM, HWParameter, LowPower, SecLib, Platform modules and IFR offset definitions.

7.1.2 mcux SDK 25.12.00 pvw2

Major Changes

- [wireless_mcu] Reduced RPMSG buffer payload size from 496 to 270 bytes on KW43/KW47 platforms, saving 226 bytes per buffer (1808 bytes total with 4 buffers on each core). This optimization is possible as rpmsg-lite no longer requires buffer sizes to be powers of two.
- [configs] Introduced RL_ALLOW_CUSTOM_SHMEM_CONFIG flag in rpmsg_config.h to enable connectivity applications to use platform_set_static_shmem_config() and platform_get_custom_shmem_config().

Minor Changes

- [wireless_mcu] Updated radio power management configuration with PLATFORM_InitRadio() API on kw47/mcxw72 platforms
- [DBG] Added NBU assert indication support to host with line/file info using debug structure.
- [DBG] Enhanced NBU debug framework with warning detection and notification capabilities. Extended NBUDBG_StateCheck() to monitor NBU warnings via PLATFORM_IsNbuWarningSet() with callback support for proactive warning monitoring.
- [Sensors] Added periodic temperature measurement support allowing app/host to request periodic temperature measurement.
- [Sensors] Added markdown documentation explaining periodic measurement functionality upon NBU requests.
- [SecLib_RNG] Added documentation for asynchronous seed handling using RNG_NotifyReseedNeeded() and SecLib implementation flavors (Software, EdgeLock, PSA Crypto, MbedTLS).
- [PSA] Simplified PSA configuration files and reduced imports/definitions for wireless MCU platforms.
- [mcxw23] Changed temperature dummy value for sensors to progress on application enablement.
- [NVS] Enhanced debug capabilities by adding CONFIG_NVS_LOG_LEVEL and improved LOG macros to adapt to PRINTF limitations.

Bug fixes

- [wireless_mcu] Fixed FRO32K as 32 kHz clock source with deferred OSC32K switching to improve initialization performance after warm reset.
- [wireless_mcu] Added wait loop for NBU power domain readiness in PLATFORM_InitNbu() to prevent race conditions when accessing NBU power domain in applications without NBU images.
- [wireless_mcu] Fixed external flash blank check procedure for LSPI external NOR Flash by correcting PLATFORM_ExternalFlashAreaIsBlank() to read from RAM buffer and perform erase pattern comparison in RAM with optimized 4-byte step loops.
- [NVS] Removed mflash dependency from NVS external flash port and fixed internal flash blank check of unaligned flash data.
- [mcxw23] Ensured TX power does not exceed 6dBm maximum limit.

7.1.1 mcux SDK 25.12.00 pvw1

Minor Changes

- [wireless_mcu][wireless_nbu] Migrated TSTMR implementation to use SDK fsl_tstmr driver for better maintainability and consistency. This migration replaces custom TSTMR register definitions with official SDK driver APIs while maintaining existing PLATFORM_* API compatibility.

Bug fixes

- [SecLib_RNG][mbedtls] Enhanced ECDH context preservation across low-power transitions on KW45_MCXW71 and KW47_MCXW72 platforms using export/import APIs to ensure cryptographic context is retained when hardware accelerator loses internal memory during power-down mode.
- [wireless_nbu] Fixed incorrect FRODIV values that were causing reduced peripheral clocks by updating PLATFORM_FroDiv[] mapping array to prevent over-division of flash APB and RF_CMC clocks.
- [rw61x/rt1060/rt1170] Added missing common files for external and OTA support on rw61x/rt1060/rt1170 platforms.

7.0.3 RFP mcux SDK 25.09.00

Major Changes

- [wireless_mcu] Replaced ICS RX linked list with message queue to eliminate memory allocation in ISR context and enable user callbacks to run in thread context where memory allocation is permitted.
- [wireless_mcu] Added HCI RX workqueue processing support to reduce ISR execution time and system impact. Feature controlled by gPlatformHciUseWorkqueueRxProcessing_d configuration option (enabled by default on kw45_k32w1_mcxw71 and kw47_mcxw72 platforms in freertos applications). When enabled, HCI transport processes received data in system workqueue thread, allowing user callbacks to run in thread context.
- [wireless_nbu] Introduced PLATFORM_ConfigureSmuDmemMapping() API to configure SMU and DMEM sharing on NBU for kw47_mcxw72 platform, using linker file symbols for correct configuration.
- [mcxw23] Implemented HCI interface using PLATFORM API as preliminary requirement for Zephyr enablement, introducing PLATFORM_SendHciMessageAlt() alternative API.
- [wireless_mcu][wireless_nbu] Added NBU2Host event manager for status indications to host (Information, Warning, Error) sent over RPMSG.

- [wireless_mcu] Added a call to PLATFORM_IcsRxWorkHandler() within PLATFORM_NbuApiReq() for baremetal applications to prevent potential deadlocks.

Minor Changes

- [wireless_mcu] Fixed variable underflow issue in PLATFORM_RemoteActiveRel().
- [SecLib_RNG] Fixed escaping local HashKeyBuffer address issue and added missing cast in RNG_GetTrueRandomNumber() function.
- [Common] Fixed heap memory manager return values and added missing include to fwk_freertos_utils.h.
- [rw61x] Prevented array out of bounds in PLATFORM_RegisterRtcHandle().
- [FSCI] Fixed memory leak in FSCI module.
- [NVM] Enhanced debug facilitation by restricting variable scope, assigning return statuses to variables, and fixing display format in NV_ShowFlashTable().
- [wireless_mcu] Added new chip revision A2.1 support in PLATFORM_SendChipRevision() API.
- [kw47_mcxw72] Implemented BLE BD address retrieval from IFR memory with fallback to OUI + RNG.
- [DBG] Added ThreadX support to fault handlers and reworked fault handler structure with dedicated RTOS files.
- [DBG][Common] Added NBU debug support on host side to detect faults and system errors, with debug info extraction capability (limited to MCXW72/KW47).
- [Common] Platform CMake rework and Kconfig renaming, removing unneeded checks and renaming PRJSEG platform Kconfigs to COMPONENT.
- [mcxw23] Added experimental SecLib PSA support with additional configuration for MBEDTLS_ECP_C and MBEDTLS_BIGNUM_C.
- [wireless_mcu] Cleaned CMakeLists.txt to avoid wrong inclusions of files and folders from incorrect platforms.
- [wireless_mcu][wireless_nbu] Added NBU2Host warning when 32MHz crystal is unready on low power exit.
- [wireless_mcu][ot] Introduced gPlatformUseOuiFromIfr to use OUI from IFR for the extended address (disabled by default). When enabled and IFR is not blank, copies first three bytes to OUI field of extended address, otherwise uses static OUI as fallback.
- [General] Removed useless warning about TSTMR_CLOCK_FREQUENCY_MHZ definition.
- [General] Updated framework license and SBOM for 25.09 RFP release.
- [wireless_mcu] Fixed unused variable warning when gPlatformIcsUseWorkqueueRxProcessing_d and gPlatformHciUseWorkqueueRxProcessing_dare disable

7.0.3 revB mcux SDK 25.09.00

Major Changes

- [wireless_mcu] Adjusted default value of BOARD_RADIO_DOMAIN_WAKE_UP_DELAY from 0x16 to 0x10 to address stability issues observed with the previous setting. This change enhances system reliability but will reduce low-power performance.

Minor Changes (bug fixes)

- [Common] Added MDK compatibility for the errno framework header.

- [mcxw23] Implemented missing PLATFORM_OtaClearBootInterface() API.
- [mcxw23] Refactored fwk_platform.c to separate BLE-specific logic into fwk_platform_ble.c.
- [OTA] Corrected definition of gEepromParams_WriteAlignment_c flag for mcxw23
- [OTA] Enabled calling OTA_GetImgState() prior to OTA_Initialize().
- [wireless_mcu] Fixed PLATFORM_IsExternalFlashSectorBlank() to check the entire sector instead of just one page.
- [mcxw23] Added support for OTA using external flash.
- [mcxw23] Introduced PLATFORM_GetRadioIdleDuration32K() to estimate time until next radio event.
- [OTA] Removed gUseInternalStorageLink_d linker flag definition when external OTA storage is used.
- [mcxw23] Extended CopyAndReboot() to support external flash OTA.
- [wireless_mcu] Resolved counter wrap issue in PLATFORM_GetDeltaTimeStamp().
- [kw43_mcxw70] Defined LPTMR frequency constants in fwk_platform_definitions.h.
- [kw47_mcxw72] Updated shared memory allocation for RPMsg adapter.
- [mcxw23] Implement PLATFORM_IsExternalFlashBusy() API.
- [kw45_mcxw71][kw47_mcxw72] Moved RAM bank definitions from the connectivity framework to device-specific definitions.

7.0.3 revA mcux SDK 25.09.00

Major Changes

- [wireless_nbu] Enhanced XTAL32M trimming handling: updates are applied when requested by the application core and the NBU enters low-power mode, ensuring no interference from ongoing radio activity. Introduced new APIs to lock (PLATFORM_LockXtal32MTrim()) and unlock XTAL32M (PLATFORM_UnlockXtal32MTrim()) trimming updates using a counter-based mechanism. Also added a reset API (PLATFORM_ResetContext()) for platform-specific variables (currently limited to the trimming lock).
- [wireless_mcu] Introduced a new API, PLATFORM_SetLdoCoreNormalDriveVoltage(), to enable support for NBU clock frequency at 64 MHz, as required by BLE channel sounding applications.
- [wireless_mcu][wireless_nbu] Increased delayLpoCycle default from 2 to 3 to address link layer instabilities in low-power NBU use cases. Adjusted BOARD_RADIO_DOMAIN_WAKE_UP_DELAY from 0x10 to 0x16 to balance power consumption and stability. □ NBU may malfunction if delayLpoCycle (or BOARD_LL_32MHz_WAKEUP_ADVANCE_HSLOT) is set to 2 while BOARD_RADIO_DOMAIN_WAKE_UP_DELAY is 0x16.

Minor Changes (bug fixes)

- [WorkQ] Increased stack size when RNG use mbedtls port and coverage is enabled.
- [FSCI] Resolved an issue where messages remained unprocessed in the queue by ensuring OSA_EventSet() is triggered when pending messages are detected.
- [OTA] Fixed a bug in in OTA_PullImageChunk() that prevented retrieval of data previously received via OTA_PushImageChunk() when still buffered in RAM during posted operations.
- [OTA] Various MISRA and coverity fixes.

- [mcxw23] Fixed an unused variable warning in PLATFORM_RegisterNbuTemperatureRequestEventCb() API.
- [SFC] Remove obsolete flag gNbuJtagCapability.
- [wireless_mcu] Introduced new API PLATFORM_GetRadioIdleDuration32K(). Deprecated PLATFORM_CheckNextBleConnectivityActivity() API.
- [mcxw23] Aligned platform-specific implementations with the corresponding prototypes defined in wireless_mcu.
- [DBG] Cleaned up fwk_fault_handler.c.

7.0.2 RFP mcux SDK 25.06.00

Major Changes

- [wireless_mcu][wireless_nbu] Introduced PLATFORM_Get32KTimeStamp() API, available on platforms that support it.
- [RNG] Switched to using a workqueue for scheduling seed generation tasks.
- [Sensors] Integrated workqueue to trigger temperature readings on periodic timer expirations.
- [wireless_nbu] Removed outdated configuration files from wireless_nbu/configs.
- [SecLib_RNG][PSA] Added a PSA-compliant implementation for SecLib_RNG. ⚠ This is an experimental feature and should be used with caution.
- [wireless_mcu][wireless_nbu] Implemented PLATFORM_SendNBUXtal32MTrim() API to transmit XTAL32M trimming values to the NBU.

Minor Changes (bug fixes)

- [MWS] Migrated the Mobile Wireless Standard (MWS) service to the public repository. This service manages coexistence between connectivity protocols such as BLE, 802.15.4, and GenFSK.
- [HWParameter][NVM][SecLib_RNG][Sensors] Addressed various MISRA compliance issues across multiple modules.
- [Sensors] Applied a filtering mechanism to temperature data measured by the application core before forwarding it to the NBU, improving data reliability.
- [Common] Relocated the GetPowerOfTwoShift() function to a shared module for broader accessibility across components.
- [RNG] Resolved inconsistencies in RNG behavior when using the fsl_adapter_rng HAL by aligning it with other API implementations.
- [SecLib] Updated the AES CMAC block counter in AES_128_CMAC() and AES_128_CMAC_LsbFirstInput() to support data segments larger than 4KB.
- [SecLib] Utilized sss_sscp_key_object_free() with kSSS_keyObjFree_KeysStoreDefragment to avoid key allocation failures.
- [MCXW23] Removed redundant NVIC_SetPriority() call for the ctimer IRQ in the platform file, as it's already handled by the driver.
- [WorkQ] Increased workqueue stack size to accommodate RNG usage with mbedtls.
- [wireless_mcu][ot] Suppressed chip revision transmission when operating with nbu_15_4.
- [platform][mflash] Ensured proper address alignment for external flash reads in PLATFORM_ReadExternalFlash() when required by platform constraints.

- [RNG] Corrected reseed flag behavior in RNG_GetPseudoRandomData() after reaching gRngMaxRequests_d threshold.
- [platform][mflash] Fixed uninitialized variable issue in PLATFORM_ReadExternalFlash().
- [platform][wireless_nbu] Fixed an issue on KW47 where PLATFORM_InitFro192M incorrectly reads IFR1 from a hardcoded flash address (0x48000), leading to unstable FRO192M trimming. The function is now conditionally compiled for KW45 only.

7.0.2 revB mcux SDK 25.06.00

Major Changes

- [RNG][wireless_mcu][wireless_nbu] Rework RNG seeding on NBU request
- [wireless_mcu] [LowPower] Add gPlatformEnableFro6MCalLowpower_d macro to enable FRO6M frequency verification on exit of Low Power
  - add PLATFORM_StartFro6MCalibration() and PLATFORM_EndFro6MCalibration() new function for FRO6M calibration (6MHz or 2Mhz) on wake-up from low power mode.
  - Enabled by default in fwk_config.h
- [wireless_nbu][LowPower] Clear pending interrupt status of the systick before going in low-power - Reduce NBU active time
- [wireless_nbu] Fix impossibility to go to WFI in combo mode (15.4/BLE)
- [wireless_mcu] Implement XTAL32M temperature compensation mechanism. 2 new APIs:
  - PLATFORM_RegisterXtal32MTempCompLut(): register the temperature compensation table for XTAL32M.
  - PLATFORM_CalibrateXtal32M(): apply XTAL32M temperature compensation depending on current temperature.
- [Sensors][wireless_mcu] Add support for periodic temperature measurement. new API:
  - SENSORS_TriggerTemperatureMeasurementUnsafe(): to be called from Interrupt masked critical section, from ISR or when scheduler is stopped
- [SFC] Change default maximal ppm target of the SFC algorithm from 200 to 360ppm. Impact the SFC algorith of kw45 and mcxw71 platforms, 360ppm was already the default setting for kw47 and mcxw72 platforms

Minor Changes (bug fixes)

- [DBG] Fix FWK_DBG_PERF_DWT_CYCLE_CNT_STOP macro
- [wireless_nbu] Add gPlatformIsNbu_d compile Macro set to 1
- [wireless_nbu][ics] gFwkSrvHostChipRevision_c can be processed in the system workqueue
- [kw45_mcxw71][kw47_mcxw72]
  - Remove LTC dependency from platform in kconfig
  - gPlatformShutdownEccRamInLowPower moved from fwk_platform_definition.h to fwk_confg.h as this is a configuration flag.
- [wireless_mcu][sensors] Rework and remove unnecessary ADC APIs
- [wireless_nbu] Add PLATFORM_GetMCUUid() function from Chip UID
- [SecLib] Change AES_MMO_BlockUpdate() function from private to public for zigbee.

7.0.2 revA mcux SDK 25.06.00

Supported platforms:

- Same as 25.03.00 release

Major Changes

- [KW45/MCXW71] HW parameters placement now located in IFR section. Flash storage is not longer used:
  - Compilation: Macro gHwParamsProdDataPlacement_c changed from gHwParamsProdDataMainFlash2IfrMode_c to gHwParamsProdDataIfrMode_c
- [KW47] NBU: Add new fwk_platform_dcdc.[ch] files to allow DCDC stepping by using SPC high power mode. This requires new API in board_dcdc.c files. Please refer to new compilation MACROs gBoardDcdcRampTrim_c and gBoardDcdcEnableHighPowerModeOnNbu_d in board_platform.h files located in kw47evk, kw47loc, frdmmcxw72 board folders.
- [KW45/MCXW71/KW47/MCXW72] Trigger an interrupt each time App core calls PLATFORM_RemoteActiveReq() to access NBU power domain in order to restart NBU core for domain low power process

Minor Changes (bug fixes)

Services

- [SecLib_RNG]
  - Rename mSecLibMutexId mutex to mSecLibSssMutexId in SecLib_sss.c
  - Remove MEM_TRACKING flag from RNG.c
  - Implement port to fsl_adapter_rng.h API using gRngUseRngAdapter_c compil Macro from RNG.c
  - Add support for BLE debug Keys in SecLi and SecLin_sss.c with gSecLibUseBleDebugKeys_d - for Debug only
- [FSCI] Add queue mechanism to prevent corruption of FSCI global variableAllow the application to override the trig sample number parameter when gFsciOverRpmsg_c is set to 1
- [DBG][btsnoop] Add a mechanism to dump raw HCI data via UART using SBTSNOOP_MODE_RAW
- [OTA]
  - OtaInternalFlash.c: Take into account chunks smaller than a flash phrase worth
  - fwk_platform_ot.c: dependencies and include files to gpio, port, pin_mux removed

Platform specific

- [kw45_mcxw71][kw47_mcxw72]
  - fwk_platform_reset.h : add compil Macro gUseResetByLvdForce_c and gUseResetByDeepPowerDown_c to avoid compile the code if not supported on some platforms
  - New compile Flag gPlatformHasNbu_d
  - Rework FRO32K notification service for MISRA fix

7.0.1 RFP mcux SDK 25.03.00

Supported platforms:

- KW45x, KW47x, MCXW71, MCXW72, K32W1x
- RW61x

- RT595, RT1060, RT1170
- MCXW23

Minor Changes (bug fixes)

- [General] Various MISRA/Coverity fixes in framework: NVM, RNG, LowPower, SecLib and platform files

Services

- [SecLib_RNG] fix return status from RNG_GetTrueRandomNumber() function: return correctly gRngSuccess_d when RNG_entropy_func() function is successful
- [SFC] Allow the application to override the trig sample number parameter
- [Settings] Re-define the framework settings API name to avoid double definition when gSettingsRedefineApiName_c flag is defined

Platform specific

- [wireless_mcu] fwk_platform_sensors update :
  - Enable temperature measurement over ADC ISR
  - Enable temperature handling requested by NBU
- [wireless_mcu] fwk_platform_lcl coex config update for KW45
- [kw47_mcxw72] Change the default ppm_target of SFC algorithm from 200 to 360ppm

7.0.1 revB mcux SDK 25.03.00

Supported platforms:

- KW45x, KW47x, MCXW71, MCXW72, K32W1x
- RW61x
- RT595, RT1060, RT1170
- MCXW23

Minor Changes (bug fixes)

General

- [General] Various MISRA/Coverity fixes in framework: NVM, RNG, LowPower, FunctionLib and platform files

Services

- [SecLib_RNG] AES-CBC evolution:
  - added AES_CBC_Decrypt() API for sw, SSS and mbedtls variants.
  - Made AES-CBC SW implementation reentrant avoiding use of static storage of AES block.
  - fixed SSS version to update Initialization Vector within SecLib, simplifying caller's implementation.
  - modified AES_128_CBC_Encrypt_And_Pad() so as to avoid the constraint mandating that 16 byte headroom be available at end of input buffer.
- [SecLib_RNG] RNG modifications:

  - RNG_GetPseudoRandomData() could return 0 in some error cases where caller expected a negative status.
    * Explicited RNG error codes
    * Added argument checks for all APIs and return gRngBadArguments_d (-2) when wrong
    * added checks of RNG initalization and return gRngNotInitialized_d (-3) when not done
    * fixed correcteness of RNG_GetPrngFunc() and RNG_GetPrngContext() relative to API description.
    * Added RNG_DeInit() function mostly for test and coverage purposes.
    * Improved RNG description in README.md
    * Unified the APIs behaviour between mbedtls and non mbedtls variants.
  - RNG/mbedtls : Prevent RNG_Init() from corrupting RNG entropy context if called more than once.
  - RNG/mbedtls: fixed RNG_GetTrueRandomNumber() to return a proper mbedtls_entropy_func() result.
  - Use defragmetation option when freeing key object in SecLib_sss to avoid leak in S200 memory
  - Add new API ECP256_IsKeyValid() to check whether a public key is valid
- [OtaSupport] Update return status to OTA_Flash_Success when success at the end of InternalFlash_WriteData() and InternalFlash_FlushWriteBuffer() APIs
- [WorQ] Implementing a simple workqueue service to the framework
- [SFC] Keep using immediate measurement for some measurement before switching to configuration trig to confirm the calibration made
- [DBG] Adding modules to framework DBG :
  - sbtsnoop
  - SWO
- [Common] Fix HAL_CTZ and HAL_RBIT IAR versions
- [LowPower] Fix wrong tick error calculation in case of infinite timeout
- [Settings] Add new macro gSettingsRedefineApiName_c to avoid multiple definition of settings API when using connectivity framework repo

Platform specific

- [KW47/MCXW72] Change xtal cload default value from 4 to 8 in order to increase the precision of the link layer timebase in NBU
- [wireless_mcu] [wireless_nbu] Use new WorkQ service to process framework intercore messages
- [rw61x] Fix HCI message sending failure in some corner case by releasing controller wakes up after that the host has send its HCI message
- [MCXW23] Adding the initial support of MCXW23 into the framework

7.0.0 mcux SDK 24.12.00

Supported platforms:

- KW45x, KW47x, MCXW71, MCXW72, K32W1x

- RW61x
- RT595, RT1060, RT1170

Minor Changes (bug fixes)

Platform specific

- [RW61X]
  - Add MCUX_COMPONENT_middleware.wireless.framework.platform.rng to the platform to fix a warning at generation
  - Retrieve IEEE 64 bits address from OTP memory
- [KW45x, MCXW71x, KW47x, MCXW72x]
  - Ignore the secure bit from RAM addresses when comparing used ram bank in bank retention mechanism
  - Add gPlatformNbuDebugGpioDAccessEnabled_d Compile Macro (enabled by default). Can be used to disable the NBU debug capability using IOs in case Trustzone is enabled ("PLATFORM_InitNbu()' code executed from unsecure world).
  - Fix in NBU firmware when sending ICS messages gFwkSrvNbuApiRequest_c (from controller_api.h API functions)

Services

- [OTA]
  - Add choice name to OtaSupport flash selection in Kconfig
- [NVM]
  - Add gNvmErasePartitionWhenFlashing_c feature support to gcc toolchain
- [SecLib_RNG]
  - Misra fixes

7.0.0 revB mcux SDK 24.12.00

Supported platforms: KW45x, KW47x, MCXW71, MCXW72, K32W1x, RW61x, RT595, RT1060, RT1170

Major Changes (User Applications may be impacted)

- mcux github support with cmake/Kconfig from sdk3 user shall now use CmakeLists.txt and Kconfig files from root folder. Compilation should be done using west build command. In order to see the Framework Kconfig, use command >west build -t guiconfig
- Board files and linker scripts moved to examples repository

Bugfixes

- [platform lowpower]
  - Entering Deep down power mode will no longer call PLATFORM_EnterPowerDown(). This API is now called only when going to Power down mode

Platform specific

- [KW47/MCXW72]: Early access release only

  - Deep sleep power mode not fully tested. User can experiment deep sleep and deep down modes using low power reference design applications
  - XTAL32K-less support using FRO32K not tested
- [KW45/MCXW71/K32W148]
  - Deep sleep mode is supported. Power down mode is supported in low power reference design applications as experimental only
  - XTAL32K-less support using FRO32K is experimental - FRO32K notifications callback is debug only and should not be used for mass production firmware builds

Minor Changes (no impact on application)

- Overall folder restructuring for SDK3
  - [Platform]:
    * Rename platform_family from connected_mcu/nbu to wireless_mcu/nbu
    * platform family have now a dedicated fwk_config.h, rpmsg_config.h and SecLib_mbedtls_config.h
  - [Services]
    * Move all framework services in a common directory "services/"

7.0.0 revA: KW45/KW47/MCX W71/MCX W72/K32W148

Experimental Features only

- Power down on application power domain: Some tests have shown some failure. Power consumption higher than Deep Sleep. => This feature is not fully supported in this release
- XTAL32K less board with FRO32K support: Some additional stress tests are under progress.
- FRO32K notifications callback is for debug only and shall not be used for production. User shall not execute long processing (such as PRINTF) as it is executed in ISR context.

Main Changes

- Cmake/Kconfig support for SDK3.0
- [Sensors] API renaming:
  - SENSORS_InitAdc() renamed to SENSORS_Init()
  - SENSORS_DeinitAdc() remamed to SENSORS_Deinit()
- [HWparams]
  - Repair PROD_DATA sector in case of ECC error (implies loss of previous contents of sector)
- [NVM] Linker script modification for armgcc whenever gNvTableKeptInRam_d option is used:
  - placement of NVM_TABLE_RW in data initialized section, providing start and end address symbols. For details see NVM_Interface.h comments.
- [OtaSupport]
  - OTA_Initialize(): now transitions the image state from RunCandidate to Permanent if not done by the application. OTA module shall always be initialized on a Permanent image, this change ensures it is the case.
  - OTA_MakeHeadRoomForNextBlock(): now erases the OTA partition up to the image total size (rounded to the sector) if known.

Minor changes

- [Platform]
  - Updated macro values: -kw47: BOARD_32MHZ_XTAL_CDAC_VALUE from 12U to 16U, BOARD_32MHZ_XTAL_ISEL_VALUE from 7U to 11U, BOARD_32KHZ_XTAL_CLOAD_DEFAULT from 8U to 4U, BOARD_32KHZ_XTAL_COARSE_ADJ_DEFAULT from 1U to 3U
    * MCX W72 (low-power reference design applications only): BOARD_32MHZ_XTAL_CDAC_VALUE from 12U to 10U, BOARD_32MHZ_XTAL_ISEL_VALUE from 7U to 11U, BOARD_32KHZ_XTAL_CLOAD_DEFAULT from 8U to 4U, BOARD_32KHZ_XTAL_COARSE_ADJ_DEFAULT from 1U to 3U
  - New PLATFORM_RegisterNbuTemperatureRequestEventCb() API: register a function callback when NBU request new temperature measurement. API provides the interval request for the temperature measurement
  - Update PLATFORM_IsNbuStarted() API to return true only if the NBU firmware has been started.
- [platform lowpower]
  - Move RAM layout values in fwk_platform_definition.h and update RAM retention API for KW47/MCXW72

Bugfixes

- [OtaSupport]
  - OTA_MakeHeadRoomForNextBlock(): fixed a case where the function could try to erase outside the OTA partition range.

6.2.4: KW45/K32W1x/MCXW71/RX61x SDK 2.16.100

This release does not contain the changes from 6.2.3 release.

This release contains changes from 6.2.2 release.

Main Change

- armgcc support for Cmake sdk2 support and VS code integration

Minor changes

- [NBU]
  - Optimize some critical sections on nbu firmware
- [Platform]
  - Optimize PLATFORM_RemoteActiveReq() execution time.

6.2.3: KW47 EAR1.0

Initial Connectivity Framework enablement for KW47 EAR1.0 support.

New features

- OpenNBU feature : nbu_ble project is available for modification and building

Supported features

- Deep sleep mode

Unsuported features

- Power down mode
- FRO32K support (XTAL32K less boards)

Main changes

- [NBU]
  - LPTMR2 available and TimerManager initialization with Compile Macro: gPlatformUseLptmr_d
  - NBU can now have access to GPIOD
  - SW RNG and SW SecLib ported to NBU (Software implementation only)
- [RNG]
  - Obsoleted API removed : FWK_RNG_DEPRECATED_API
  - RNG can be built without SecLib for NBU, using gRngUseSecLib_d in fwk_config.h
  - Some API updates:
    * RNG_IsReseedneeded() renamed to RNG_IsReseedNeeded,
    * RNG_TriggerReseed() renamed to RNG_NotifyReseedNeeded(),
    * RNG_SetSeed() and RNG_SetExternalSeed() return status code.
  - Optimized Linear Congruential modulus computation to reduce cycle count.

Minor changes

- [NVM]
  - Optimize NvIsRecordErased() procedure for faster garbage collection
  - MISRA fix : Remove externs and weaks from NVM module - Make RNG and timer manager dependencies conditional
- [Platform]
  - Allow the debugger to wakeup the KW47/MCXW72 target

6.2.2: KW45/K32W1 MR6 SDK 2.16.000

Experimental Features only:

- Power down on application power domain : Some tests have shown some failure. Power consumption higher than Deep Sleep. => This feature is not fully supported in this release
- XTAL32K less board with FRO32K support : Some additional stress tests are under progress.
- FRO32K notifications callback is for debug only and shall not be used for production. User shall not execute long processing (such as PRINTF) as it is executed in ISR context.

Changes

- [Board] Support for freedom board FRDM-MCX W7X
- [HWparams]
  - Support for location of HWParameters and Application Factory Data IFR in IFR1
  - Default is still to use HWparams in Flash to keep backward compatibility
- [RNG]: API updates:

  - New APIS RNG_IsReseedneeded(), RNG_SetSeed() to provide See to PRNG on NBU/App core - See BluetoothLEHost_ProcessIdleTask() in app_conn.c
  - New APIs RNG_SetExternalSeed() : User can provide external seed. Typically used on NBU firmrware for App core to set a seed to RNG. RNG_TriggerReseed() : Not required on App core. Used on NBU only.
- [NVS] Wear statistics counters added - Fix nvs_file_stat() function
- [NVM] fix Nv_Shutdown() API
- [SecLib] New feature AES MMO supported for Zigbee

6.2.2: RW61x RFP4 SDK 2.16.000

- [Platform] Support Zigbee stack
- [OTA] Add support for RW61x OTA with remap feature.
  - Required modifications to prevent direct access to flash logical addresses when remap is active.
  - Image trailers expected at different offset with remap enabled (see gPlatformMcuBootUseRemap_d in fwk_config.h)
  - fixed image state assessment procedure when in RunCandidate.
- [NVS] Wear statistics counters added
- [SecLib] New feature AES MMO supported for Zigbee
- [Misra] various fixes

6.2.1: KW45/K32W1 MR5 SDK 2.15.000

Experimental Features only:

- Power down on application power domain : Some tests have shown some failure. This feature is not fully supported in this release
- XTAL32K less board with FRO32K support : Some additional stress tests are under progress. Timing variation of the timebase are being analyzed

Major changes

- [RNG]: API updates
  - New compile flag to keep deprecated API: FWK_RNG_DEPRECATED_API
  - change return error code to int type for RNG_Init(), RNG_ReInit()
  - New APIs RNG_GetTrueRandomNumber(), RNG_GetPseudoRandomData()
- [Platform]
  - fwk_platform_sensors
    * Change default temperature value from -1 to 999999 when unknown
  - fwk_platform_genfsk
    * rename from platform_genfsk.c/h to fwk_platform_genfsk.c/h
  - platform family
    * Rename the framework platform folder from kw45_k32w1 to connected_mcu to support other platform from the same family
  - fwk_platform_intflash

    * Moved from fwk_platform files to the new fwk_platform_intflash files the internal flash dependant API

- [NBU]
  - BOARD_LL_32MHz_WAKEUP_ADVANCE_HSLOT changed from 2 to 3 by default
  - BOARD_RADIO_DOMAIN_WAKE_UP_DELAY changed from 0x10 to 0x0F
- [gcc linker]
  - Exclude k32w1_nbu_ble_15_4_dyn.bin from .data section

Minor Changes

- [Platform]
  - PLATFORM_GetTimeStamp(0 has an important fix for reading the Timestamp in TSTMR0
  - New API PLATFORM_TerminateCrypto(), PLATFORM_ResetCrypto() called from SecLib for lowpower exit
  - Fix when enable fro debug callback on nbu
- [DBG]
  - SWO
    * Add new files fwk_debug_swo.c/h to use SWO for debug purpose
    * Two new flags has been added:
      · BOARD_DBG_SWO_CORE_FUNNEL to chose on which core you want to use SWO
      · BOARD_DBG_SWO_PIN_ENABLE to enable SWO on a pin
- [NVS]
  - Add support of NVS and Settings in framework
- [NBU]
  - Fix power down issues and reduce critical section on NBU side:
    * new API PLATFORM_RemoteActiveReqWithoutDelay() called from NBU functions where waiting delay is not required
    * Increase delay needed in power down for OEM part to request the SOC to be active
    * Remove unnecessary code to PLATFORM_RemoteActiveReqWithoutDelay() from PLATFORM_HciRpmsgRxCallback()
    * Improve nbu memory allocation failure debug messages
- [SDK]
  - Multicore: remove critical section in HAL_RpmsgSendTimeout() (only required in FPGA HDI mode)
  - Flash drivers: update for ECC detection
- [Platform]
  - fwk_platform_sensors
    * Fix temperature reporting to NBU
  - fwk_platform_extflash
    * Align .c and .h prototype of PLATFORM_ExternalFlashAreaIsBlank() function
- [NVM]

  - Keep Mutex in NvModuleDeInit(). In Bare metal OS, Mutex can not be destroyed
  - New API NvRegisterEccFaultNotificationCb() to register Notification callback when Ecc error happens in FileSystem
- [MISRA] fixes
  - SecLib_sss.c: ECDH_P256_ComputeDhKey()
  - fwk_platform_extflash.c: PLATFORM_IsExternalFlashPageBlank()
  - fwk_fs_abstraction.c: Various fixes
- [HWparams]
  - Fix on if condition when gHwParamsProdDataPlacementLegacy2IfrMode_c mode is selected
- [OTA]
  - Enable gOtaCheckEccFaults_d by default to avoid bus in case of ECC error during OTA
  - Fix OTA partition overflow during OTA stop and resume transfer
- [BOARD]
  - Place code button or led specific under correct defines in board_comp.c/h
  - Bring back MACROs BOARD_INITRFSWITCHCONTROLPINS in pin_mux header file of the loc board
- [SecLib]
  - Add some undefinition in SecLib_mbedtls_config as new dependency has been added in mbedtls repo:
    * MBEDTLS_SSL_CBC_RECORD_SPLITTING, MBEDTLS_SSL_PROTO_TLS1, MBEDTLS_SSL_PROTO_TLS1_1
- [FRO32K]
  - FRO32K notification callback PLATFORM_FroDebugCallback_t() has new parameter to report he fro_trim value
  - maxCalibrationIntervalMs value can be provided to NBU using PLATFORM_FwkSrvSetRfSfcConfig()
- [Sensors]
  - fix: PLATFORM_GetTemperatureValue() shall have NBU started to send temperature to NBU

6.2.1: RW61x RFP3

- [NVS]
  - Add support of NVS and Settings in framework
- [MISRA] fixes
  - board_lp.c BOARD_UninitDebugConsole() and BOARD_ReinitDebugConsole()
  - fwk_platform_ble.c: Various fixes
- [OTA]
  - Fix OTA partition overflow during OTA stop and resume transfer

6.2.0: RT1060/RT1170 SDK2.15 Major

6.1.8: KW45/K32W1 MR4

- [BOARD PLATFORM]
  - Move gBoardUseFro32k_d to board_platform.h file
  - Offer the possibility to change the source clock accuracy to gain in power consumption
- [BOARD LP]
  - Move PLATFORM_SetRamBanksRetained() at end of BOARD_EnterLowPowerCb() in case a memory allocation is done previously in this function
  - fix low power, increase BOARD_RADIO_DOMAIN_WAKE_UP_DELAY from 0 to 0x10 - Skip this delay when App requesting NBU wakeup
- [PLATFORM]
  - fwk_platform_ble.c/h: New timestamp API that returns the difference between the current value of the LL clock and the argument of the function
  - fwk_platform.c/h:
    * New PLATFORM_EnableEccFaultsAPI_d compile flag: Enable APIs for interception of ECC Fault in bus fault handler
    * New gInterceptEccBusFaults_d compile flag: Provide FaultRecovery() demo code for bus fault handler to Intercept bus fault from Flash Ecc error
- [LOC]
  - Incorrect behavior for set_dtest_page (DqTEST11 overridden)
  - Fix SW1 button wake able on Localization board
  - Fix yellow led not properly initialized
  - Format localization pin_mux.c/h files
- [Inter Core]
  - Affect values to enumeration giving the inter core service message ids
  - Shared memory settings shared between both cores
  - Add callback to register when NBU has unrecoverable Radio issue
- [NVM]
  - Add NV_STORAGE_MAX_SECTORS, NV_STORAGE_SIZE as linker symbol for alignment with other toolchain
  - ECC detection and recovery. New gNvSalvageFromEccFault_d and gNvVerifyReadBackAfterProgram_d compile flags. Please refer to ECC Fault detection section in README.md file located in NVM folder
- [OTA]
  - Prevent bus fault in case of ECC error when reading back OTA_CFR update status (disable by default)
- [SecLib]
  - Shared mutex for RNG and SecLib as they share same hardware resource
- [Key storage]
  - Fix to ignore the garbage at the end of buffers
  - Detect when buffers are too small in KS_AddKey() functions
- [FileCache]
  - Fix deadlock in Filecache FC_Process()
- [SDK]

  - Applications: remove definition of stack location and use default from linker script, fix warmboot stack in freertos at 0x20004000
  - Memory Manager Light:
    * fix Null pointer harfault when MEM_STATISTICS_INTERNAL enable
    * Fix MemReinitBank() on wakeup from lowpower when Ecc banks are turned off

6.1.7: KW45/K32W1 MR3

- [OTA]
  - New API OTA_SetNewImageFlagWithOffset()
  - Fix StorageBitmapSize calculation
  - OTA clean up: Removed OTA_ValidateImage()
- [Low Power]
  - New linker Symbol m_lowpower_flag_start in linker file.
    * Flag is used to indicate NBU that Application domain goes to power down mode. Keep this flag to 0 if only Deep sleep is supported
    * This flag will be set to 1 if Application domain goes to power down mode
  - Re-introduce PWR_AllowDeviceToSleep()/PWR_DisallowDeviceToSleep(), PWR_IsDeviceAllowedToSleep() API
  - Implement tick compensation mechanism for idle hook in a dedicated freertos utils file fwk_freertos_utils.[ch], new functions: FWK_PreIdleHookTickCompensation() and FWK_PostIdleHookTickCompensation
  - Rework timestamping on K4W1
    * PLATFORM_GetMaxTimeStamp() based on TSTMR
    * Rename PLATFORM_GetTimestamp() to PLATFORM_GetTimeStamp()
    * Update PLATFORM_Delay(): Rework to use TSTMR instead of LPTMR for platform_delay
    * Update PLATFORM_WaitTimeout(): Fixed a bug in PLATFORM_WaitTimeout() related to timer wrap
    * Add PLATFORM_IsTimeoutExpired() API
  - Fix race condition in PWR_EnterLowPower(), masking interrupts in case not done at upper layer
  - Low power timer split in new files fwk_platform_lowpower_timer.[ch]
  - New PWR_systicks_bm.c file for bare metal usage: implement SysTick suspend/resume functionality, New weak PWR_SysTicksLowPowerInit()
- [FRO32K]
  - Improve FRO32K calibration in NBU
  - create PLATFORM_InitFro32K() to initialize FRO32K instead of XTAL32K (to be called from hardware_init())
  - update FRO32K README.md file in SFC module
  - Debug:
  - Add Notification callback feature for SFC module FRO32K
  - Linker script update to support m_sfc_log_start in SMU2
- [SecLib]

  - Remove gSecLibSssUseEncryptedKeys_d compile option, split Secure/Unsecure APIs
  - RNG update to use same mutex than SecLib
  - Fix AES_128_CBC_Encrypt_And_Pad length
  - Implement RNG_ReInit() for lowpower
  - Fix issue in ECDH_P256_GenerateKeys() when waking up from power down
  - Call CRYPTO_ELEMU_reset() from SecLib_reInit() for power down support
- [BOARD]
  - Create new board_platform.h file for all Board characteristics settings (32Mhz XTAL, 32KHZ XTAL, etc..)
  - TM_EnterLowpower() TM_EnterLowpower() to be called from LP callbacks
  - Support Localization boards, Only BUTTON0 supported
    * New compile flag BOARD_LOCALIZATION_REVISION_SUPPORT
    * New pin_mux.[ch] files
  - Offer the possibility to override CDAC and ISEL 32MHz settings before the initialization of the crystal in board_platform.h
    * new BOARD_32MHZ_XTAL_CDAC_VALUE, BOARD_32MHZ_XTAL_ISEL_VALUE
    * BOARD_32MHZ_XTAL_TRIM_DEFAULT obsoleted
- [NVM file system]
  - Look ahead in pending save queue - Avoid consuming space to save outdated record
  - Fix NVM gNvDualImageSupport feature in NvIsRecordCopied
- [Inter Core]
  - Change PLATFORM_NbuApiReq() API return parameters granularity from uint32 to uint8
  - MAX_VARIANT_SZ change from 20 to 25
  - Set lp wakeup delay to 0 to reduce time of execution on host side, NBU waits XTAL to be ready before starting execution
  - Update inter core config rpmsg_config.h
  - Add timeout to while loops that relies on hardware in RemoteActiveReq(), Application can register Callbacks when timeout
  - Return non-0 status when calling PLATFORM_FwkSrvSendPacket when NBU non started
  - Let PLATFORM_GetNbuInfo return -10 if response not received on timeout - Doxygen platform_ics APIs
- [HW params]
  - New compile Macro for HW params placement in IFR - Save 8K in FLash: gHwParamsProdDataPlacement_c . 3 modes:
  - Legacy placement, move from legacy to IFR, IFR only placement
  - New compile Macro for Application data to be stored with HW params (in shared flash sector): gHwParamsAppFactoryDataExtension_d, New APIs:
    * Nv_WriteAppFactoryData(), Nv_GetAppFactoryData()
  - See HWParameter.h
- [Platform]

  - Implement PLATFORM_GetIeee802_15_4Addr() API in fwk_platform_ot.c - New gPlatformUseUniqueDeviceIdFor15_4Addr_d compile Macro
  - Wakeup NBU domain when reading RADIO_CTRL UID_LSB register in PLATFORM_GenerateNewBDAddr()
- [Reset]
  - New reset Implementations using Deep power down mode or LVD:
    * new files fwk_platform_reset.[ch]
    * new APIs: PLATFORM_ForceDeepPowerDownReset(), PLATFORM_ForceLvdReset() + reset on ext pins
    * new compile flags: gAppForceDeepPowerDownResetOnResetPinDet_d and gAppForceLvdResetOnResetPinDet_d to reset on external pins
- [FSCI]
  - fix when gFsciRxAck_c enabled
  - integrate new reset APIs

6.1.4: RW610/RW612 RFP1

- [Low Power]
  - Added support of low power for OpenThread stack.
  - Added PWR_AllowDeviceToSleep/PWR_DisallowDeviceToSleep/PWR_IsDeviceAllowedToSleep APIs.
- [platform]
  - Added PLATFORM_GetMaxTimeStamp API.
  - Fixed high impact Coverity.
- [FreeRTOS]
  - Created a new utilities module for FreeRTOS: fwk_freertos_utils.c/h.
  - Implemented a tick compensation mechanism to be used in FreeRTOS idle hook, likely around flash operations. This mechanism aims to estimate the number of ticks missed by FreeRTOS in case the interrupts are masked for a long time.

6.1.4: KW45/K32W1 MR2

- [Low power]
  - Powerdown mode tested and enabled on Low Power Reference Design applications
  - XTAL32K removal functionality using FRO32K, supported from NBU firmwares - limitation: Application domain supports Deep Sleep only (not power down)
  - NBU low power improvement: low power entry sequence improvement and system clock reduction to 16Mhz during WFI
  - Wake up time from cold boot, reset, power switch greatly improved. Device starts on FRO32K, switch to XTAL32K when ready if gBoardUseFro32k_d not set
  - Bug fixes:
    * Move PWR LowPower callback to PLATFORM layers
    * Fix wrong compensation of SysTicks
    * Reinit system clocks when exiting power down mode: BOARD_ExitPowerDownCb(), restore 96MHz clock is set before going to low power

    * Call Timermanager lowpower entry exit callbacks from PLATFORM_EnterLowPower()
    * Update PLATFORM_ShutdownRadio() function to force NBU for Deep power down mode
  - K32W1:
    * Support lowpower mode for 15.4 stacks
- [NVM]
  - New Compilation MACRO gNvDualImageSupport to support multiple firmware image with different register dataset
  - Change default configuration gNvStorageIncluded_d to 1, gNvFragmentation_Enabled_d to 1, gUnmirroredFeatureSet_d to TRUE
  - Some MISRA issues for this new configuration.
  - Remove deprecated functionality gNvUseFlexNVM_d
- [SecLib]
  - New NXP Ultrafast ecp256 security library:
    * New optimized API for ecdh DhKey/ecp256 key pair computation: Ecdh_ComputeDhKeyUltraFast(), ECP256_GenerateKeyPairUltraFast().
    * New macro gSecLibUseDspExtension_d.
    * Improved software version of Seclib with Ultrafast library for ECP256_LePointValid()
  - Bug fixes:
    * Share same mutex between Seclib and RNG to prevent concurrent access to S200
    * Optimized S200 re-initialization, restore ecdh key pair after power down
    * Fixed race condition when power down low power entry is aborted
    * Endianness function updates and clean up
- [OTA]
  - OTASupport improvements:
    * New API OTA_GetImgState(), OTA_UpdateImgState()
    * OTASupport and fwk_platform_extflash API updates for external flash: OTA_SelectExternalStoragePartition(), PLATFORM_IsExternalFlashSectorBlank(), PLATFORM_IsExternalFlashPageBlank(), PLATFORM_OtaGetOtaPartitionConfig()
    * Updated OtaExternalFlash.c, 2 new APIs in fwk_platform_extflash.c
    * Removed unused FLASH_op_type and FLASH_TransactionOpNode_t definitions from public API
    * Removed unused InternalFlash_EraseBlock() from OtaInternalFlash.c
- [NBU firmware]
  - Mechanism to set frequency constraint to controller from the host PLATFORM_SetNbuConstraintFrequency()
  - NbuInfo has one more digit in versionBuildNo field
- [Board]
  - Support Extflash low power mode, add BOARD_UninitExternalFlash(), PLATFORM_UninitExternalFlash(), PLATFORM_ReinitExternalFlash()

  - Support XTAL32K removal functionatity, use FRO32K instead by setting gBoardUseFro32k_d to 1 in board.h file
  - Support localization boards KW45B41Z-LOC Rev C
  - Low power improvement: New BOARD_InitPins() and BOARD_InitPinButtonBootConfig() called from hardware_init.c
  - Removed KW45_A0_SUPPORT support (dcdc)
  - Bug fixes:
    * Fixed glitches on the serial manager RX when exiting from power down
    * Fixed ADC not deinitialized in clock gated modes in BOARD_EnterLowPowerCb()
    * Fixed UART output flush when going to low power: BOARD_UninitAppConsole()
- [platform]
  - PLATFORM_InitBle(), PLATFORM_SendHci() can now block with timeout if NBU does not answer. Application can register callback function to be notified when it occurs: PLATFORM_RegisterBleErrorCallback()
  - Added API to set and get 32Khz XTAL capacitance values: PLATFORM_GetOscCap32KValue() and PLATFORM_SetOscCap32KValue()
  - Added new Service FWK call gFwkSrvNbuMemFullIndication_c to get NBU mem full indication, register with PLATFORM_RegisterNbuMemErrorCallback()
  - Added support negative value in platform intercore service
- [linker script]
  - Realigned gcc linker script with IAR linker script.
  - Added possibility to redefine cstack_start position
  - Added Possibility to change gNvmSectors in gcc linker script
  - Added dedicated reserved Section in shared memory for LL debugging
- [FreeRTOSConfig.h]
  - Removed unused MACRO configFRTOS_MEMORY_SCHEME and configTOTAL_HEAP_SIZE
- [HW Param]
  - Added xtalCap32K field to store XTAL32K triming value
- [fwk_hal_macros.h]
  - Added MACRO for KB, MB and set, clear bits in bit fields
- [Debug]
  - Added MACROs for performance measurement using DWT: DBG_PERF_MEAS

6.1.3 KW45 MR1 QP1

- [Initialization] Delay the switch to XTAL32K source clock until the BLE host stack is initialized
- [lowpower] NBU wakeup from lowpower: configuration can now be programmed with BOARD_NBU_WAKEUP_DELAY_LPO_CYCLE, BOARD_RADIO_DOMAIN_WAKE_UP_DELAY in board.h file
- [NBU firmware] Major fix for NBU system clock accuracy
- [clock_config]
  - Update SRAM margin and flash config when switching system frequency

  - Trim FIRC in HSRUN case
- [XTAL 32K trim] XTAL 32K configuration can be tuned in board.h file with BOARD_32MHZ_XTAL_TRIM_DEFAULT, BOARD_32KHZ_XTAL_CLOAD_DEFAULT, BOARD_32KHZ_XTAL_COARSE_ADJ_DEFAULT
- [MAC address] Add OUI field in PLATFORM_GenerateNewBDAddr() when using Unique Device Id

6.1.2: RW610/RW612 PRC1

- [Low Power]
  - Updates after SDK Power Manager files renaming.
  - Moved PWR LowPower callback to PLATFORM layers.
  - Bug fixes:
    * Fixed wrong compensation of SysTicks during tickless idle.
    * Reinit RTC bus clock after exit from PM3 (power down).
- [OTA]
  - Initial support for OTA using the external flash.
- [platform]
  - Implemented platform specific time stamp APIs over OSTIMER.
  - Implemented platform specific APIs for OTA and external flash support.
  - Removed PLATFORM_GetLowpowerMode API.
  - Added support of CPU2 wake up over Spinel for OpenThread stack.
  - Bug fixes:
    * Fixed issues related to handling CPU2 power state.
- [board]
  - Updated flash_config to support 64MB range.
- [linker script]
  - Fixed wrong assert.

6.1.1: KW45/K32W1 MR1

- [platform] Use new FLib_MemSet32Aligned() to write in ECC RAM bank to force ECC calculation in the MEM_ReinitRamBank() function
- [FunctionLib] Implement new API to set a word aligned
- [platform] Set coarse amplifier gain of the oscilattor 32k to 3
- [platform] Switch back to RNG for MAC Adress generation
- [SecLib] Get rid of the lowpower constraint of deep sleep in ECDH API
- [DCDC] Set DCDC output voltage to 1.35V in case LDO core is set to 1.1V to ensure a drop of 250mV between them
- [NVM] NvIdle() is now returning the number of operations that has been executed
- [documentation] Add markdown of each framework module by default on all package
- [LowPower] Add a delay advised by hardware team on exit of lowpower for SPC
- [SecLib] Rework of SecLib_mbedTLS ECDH functions

- [OTA] Make OTA_IsTransactionPending() public API
- [FunctionLib] Change prototype of FLib_MemCpyWord(), pDst is now a void* to permit more flexibility
- [NVM] Add an API to know if there is a pending operation in the queue
- [FSCI] Fix wrong error case handling in FSCI_Monitor()

6.1.0: KW45/K32W1 RFP

- [LowPower] Do not call PLATFORM_StopWakeUpTimer() in PWR_EnterLowPower() if PLATFORM_StartWakeUpTimer() was not previously called
- [boards] Add the possibility to wakeup on UART 0 even if it is not the default UART
- [boards] Add support for Hardware flow control for UART0, Enable with gBoardUseUart0HwFlowControl, Pin mux update with two additional API for RTS, CTS pins
- [Sensors] Improve ADC wakeup time from deep sleep state: use save and restore API for ADC context before/after deep sleep state.
- [linker script] update SMU2 shared memory region layout with NBU: increase sqram_btblebuf_size to support 24 connections. Shared memory region moved to the end
- [SecLib] SecLib_DeriveBluetoothSKD() API update to support if EdgeLock key shall be regenerated

6.0.11: KW45/K32W1 PRC3.1

FSCI: Framework Serial Communication Interface Overview

The Framework Serial Communication Interface (FSCI) is both a software module and a protocol that allows monitoring and extensive testing of the protocol layers. It also allows separation of the protocol stack between two protocol layers in a two processing entities setup, the host processor (typically running the upper layers of a protocol stack) and the Black Box application (typically containing the lower layers of the stack, serving as a modem). The Test Tool software is an example of a host processor, which can interact with FSCI Black Boxes at various layers. In this setup, the user can run numerous commands to test the Black Box application services and interfaces.

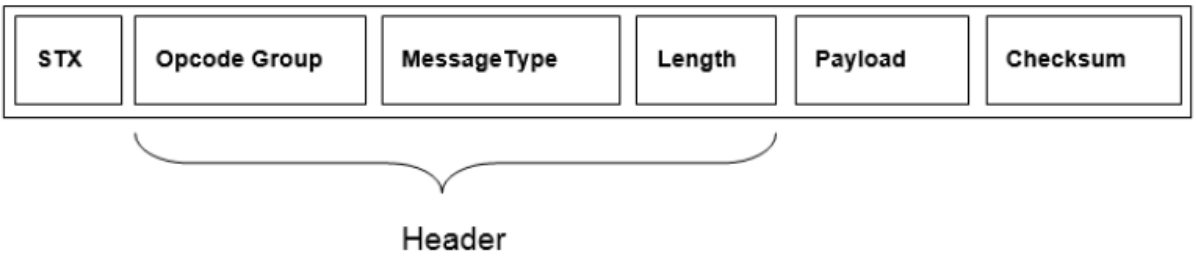
The FSCI enables common service features for each device enables monitoring of specific interfaces and API calls. Additionally, the FSCI injects or calls specific events and commands into the interfaces between layers.

An entity which needs to be interfaced to the FSCI module can use the API to register opcodes to specific interfaces. After doing so, any packet coming from that interface with the same opcode triggers a callback execution. Two or more entities cannot register the same opcode on the same interface, but they can do so on different interfaces. For example, two MAC instances can register the same opcodes, one over UARTA, and the other over UARTB. This way, Test Tool can communicate with each MAC layer over two UART interfaces.

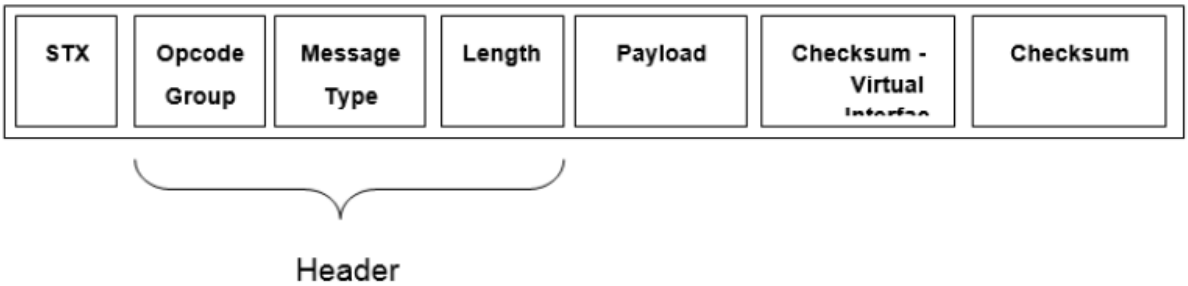
The FSCI module executes either in the context of the Serial Manager task or owns its dedicated task if the compilation Macro *gFsciUseDedicatedTask_c* is set to 1.

FSCI packet structure

The FSCI module sends and receives messages as shown in the figure below. This structure is not specific to a serial interface and is designed to offer the best communication reliability. The Black Box device expects messages in little-endian format. It also responds with messages in little-endian format.



Below is an illustration of the FSCI packet structure when a virtual interface is used instead :



Field name	Length (bytes)	Description
STX	1	Used for synchronization over the serial interface. The value is always 0x02.
Opcode Group	1	Distinguishes between different Service Access Primitives (for example MLME or MCPS).
Message Type	1	Specifies the exact message opcode that is contained in the packet.
Length	1 or 2	The length of the packet payload, excluding the header and FCS. The length field content must be provided in little-endian format.
Payload	variable	Payload of the actual message.
Checksum	1	Checksum field used to check the data integrity of the packet.
Checksum2	0 or 1	The second CRC field appears only for virtual interfaces.

NOTE : When virtual interfaces are used, the first checksum is decremented with the ID of the interface. The second checksum is used for error detection.

constant definition

The following Macro configurs the FSCI module

```
#define gFsciIncluded_c 0 /* Enable/Disable FSCI module */
#define gFsciUseDedicatedTask_c 1 /* Enable Fsci task to avoid recursivity in Fsci module (Misra␣
↪compliant) */
#define gFsciMaxOpGroups_c 8
#define gFsciMaxInterfaces_c 1
#define gFsciMaxVirtualInterfaces_c 0
#define gFsciMaxPayloadLen_c 245 /* bytes */
#define gFsciTimestampSize_c 0 /* bytes */
#define gFsciLenHas2Bytes_c 0 /* boolean */
#define gFsciUseEscapeSeq_c 0 /* boolean */
#define gFsciUseFmtLog_c 0 /* boolean */
#define gFsciUseFileDataLog_c 0 /* boolean */
#define gFsciLoggingInterface_c 1 /* [0..gFsciMaxInterfaces_c) */
#define gFsciHostMacSupport_c 0 /* Host support at MAC layer */
```

The following provides the OpGroups values reserved by MAC, application, and FSCI.

FSCI Host

FSCI Host is a functionality that allows separation at a certain stack layer between two entities, usually two boards running separate layers of a stack.

Support is provided for functionality at the MAC layer, for example, MAC/PHY layers of a stack are running as a Black Box on a board, and MAC higher layers are running on another. The higher layers send and receive serial commands to and from the MAC Black Box using the FSCI set of operation codes and groups.

The protocol of communication between the two is the same. The current level of support is provided for:

- FSCI_MsgResetCPUReqFunc – sends a CPU reset request to black box
- FSCI_MsgWriteExtendedAdrReqFunc – configures MAC extended address to the Black Box
- FSCI_MsgReadExtendedAdrReqFunc – N/A

The approach on the Host interfacing a Black Box using synchronous primitives is by default the polling of the FSCI_receivePacket function, until the response is received from the Black Box. The calling task polls whenever the task is being scheduled. This is required because a stack synchronous primitive requires that the response of that request is available in the context of the caller right after the SAP call has been executed.

The other option, available for RTOS environments, is using an event mechanism. The calling task blocks waiting for the event that is sent from the Serial Manager task when the response is available from the Black Box. This option is disabled by default. The disadvantage of this option is that the primitive cannot be received from another Black Box through a serial interface because the blocked task is the Serial Manager task, which reaches a deadlock as cannot be released again.

FSCI ACK

ACK transmission is enabled through the gFsciTxAck_c macro definition. Each FSCI valid packet received triggers an FSCI ACK packet transmission on the same FSCI interface that the packet was received on. The serial write call is performed synchronously to send the ACK packet before any other FSCI packet. Only then the registered handler is called to process the received packet. The ACK is represented by the gFSCI_CnfOpcodeGroup_c and mFsciMsgAck_c Opcode. An additional byte is left empty in the payload so that it can be used optionally as a packet identifier to correlate packets and ACKs. ACK reception is the other component that is enabled through gFsciRxAck_c. The behavior is such that every FSCI packet sent through a serial interface triggers an FSCI ACK packet reception on the same interface after the packet is sent. If an ACK packet is received, the transmission is considered successful. Otherwise, the packet is resent a number of times. The ACK wait period is configurable through mFsciRxAckTimeoutMs_c and the number of transmission retries through mFsciTxRetryCnt_c. The ACK mechanism described above can also be coupled with a FSCI packet reception timeout enabled through gFsciRxTimeout_c and configurable through mFsciRxRestartTimeoutMs_c. Whenever there are no more bytes to be read from a serial interface, a timeout is configured at the predefined value if no other bytes are received. If new bytes are received, the timer is stopped and eventually canceled at successful reception. However, if, for any reason, the timeout is triggered, the FSCI module considers that the current packet is invalid, drops it, and searches for a new start marker.

FSCI usage example

Detailed data types and APIs are described in ConnFWK API documentation.

Initialization

```
/* Configure the number of interfaces and virtual interfaces used */
#define gFsciMaxInterfaces_c 4
```

(continues on next page)

(continued from previous page)

```
#define gFsciMaxVirtualInterfaces_c 2
/* ... */
/* Define the interfaces used */
static const gFsciSerialConfig_t myFsciSerials[] = {
    /* Baudrate, interface type, channel No, virtual interface */ {gUARTBaudRate115200_c, gSerialMgrUart_
↪c, 1, 0}, {gUARTBaudRate115200_c, gSerialMgrUart_c, 1, 1}, {0 , gSerialMgrIICSlave_c, 1, 0}, {0 ,␣
↪gSerialMgrUSB_c, 0, 0},
};
/* ... */
/* Call init function to open all interfaces */
FSCI_Init( (void*)mFsciSerials );
```

Registering operation groups

```
myOpGroup = 0x12; // Operation Group used
myParam = NULL; // pointer to a parameter to be passed to the handler function (myHandlerFunc)
myInterface = 1; // index of entry from myFsciSerials
/* ... */
FSCI_RegisterOpGroup( myOpGroup, gFsciMonitorMode_c, myHandlerFunc, myParam, myInterface );
```

Implementing handler function

```
void fsciMcpsReqHandler(void *pData, void* param, uint32_t interfaceId)
{
    clientPacket_t *pClientPacket = ((clientPacket_t*)pData);
    fsciLen_t myNewLen;
    switch( pClientPacket->structured.header.opCode )
    {
        case 0x01:
        {
            /* Reuse packet received over the serial interface The OpCode remains the same. The length of the␣
↪response must be <= that the length of the received packet */
            pClientPacket->structured.header.opGroup = myResponseOpGroup;/* Process packet */
            /* ... */
            pClientPacket->structured.header. len = myNewLen;
            FSCI_transmitFormatedPacket(pClientPacket, interfaceId);
            return;
        }
        case 0x02:
        {
            /* Alocate a new message for the response. The received packet is Freed */
            clientPacket_t *pResponsePkt = MEM_BufferAlloc( sizeof(clientPacketHdr_t) + myPayloadSize_d␣
↪+ sizeof(uint8_t) // CRC);
            if(pResponsePkt)
            {
                /* Process received data and fill the response packet */
                pResponsePkt->structured.header. len = myPayloadSize_d;
                FSCI_transmitFormatedPacket(pClientPacket, interfaceId);
            }
            break;
        }
        default:
            MEM_BufferFree( pData );
            FSCI_Error( gFsciUnknownOpcode_c, interfaceId );
            return;
    }
    /* Free message received over the serial interface */
    MEM_BufferFree( pData );
}
```

Helper Functions Library Overview

This framework provides a collection of features commonly used in embedded software centered on memory manipulation.

HWParameter: Hardware parameter Production Data Storage

Hardware parameters provide production data storage

Overview

Different platforms/boards need board/network node-specific settings to function according to the design. (Examples of such settings are IEEE® addresses and radio calibration values specific to the node.) For this purpose, the last flash sector is reserved and contains hardware-specific parameters for production data storage. These parameters pertain to the network node as a distinct entity. For example, a silicon mounted on a PCB in a specific configuration, rather than to just the silicon itself. This sector is reserved by the linker file, through the PROD_DATA section and it should be read/written only through the API described below.

> Note : This sector is not erased/written at code download time and it is not updated via over-the-air firmware update procedures to preserve the respective node-specific data, regardless of the firmware running on it.

Constant Definitions

Name :

```
extern uint32_t PROD_DATA_BASE_ADDR[];
```

Description :

This symbol is defined in the linker script. It specifies the start address of the PROD_DATA section.

Name :

```
static const uint8_t mProdDataIdentifier[10] = {"PROD_DATA:"};
```

Description :

The value of this constant is copied as identification word (header) at the beginning of the PROD_DATA area and verified by the dedicated read function.

Note: the length of mProdDataIdentifier imposes the definition of PROD_DATA_ID_STRING_SZ as 10. The legacy HW parameters structure provides headroom for future usage. There are currently 63 bytes available.

Data type definitions

Name :

```
typedef PACKED_STRUCT HwParameters_tag
{
    uint8_t identificationWord[PROD_DATA_ID_STRING_SZ]; /* internal usage only: valid data present */
    /*@{*/
    uint8_t bluetooth_address[BLE_MAC_ADDR_SZ];       /*!< Bluetooth address */
    uint8_t ieee_802_15_4_address[IEEE_802_15_4_SZ]; /*!< IEEE 802.15.4 MAC address - K32W1 only␣
↪*/
    uint8_t xtalTrim;                                 /*!< XTAL 32MHz Trim value */
    uint8_t xtalCap32K;                               /*!< XTAL 32kHz capacitance value */
    /* For forward compatibility additional fields may be added here
       Existing data in flash will not be compatible after modifying the hardwareParameters_t typedef.
```

(continues on next page)

(continued from previous page)

```
    In this case the size of the padding has to be adjusted.
  */
  uint8_t reserved[1];
  /* first byte of padding : actual size if 63 for legacy HwParameters but
    complement to 128 bytes in the new structure */
}
hardwareParameters_t;
```

Description:

Defines the structure of the hardware-dependent information.

> Note : Some members of this structure may be ignored on a specific board/silicon configuration. Also, new members may be added for implementation-specific purposes and the backward compatibility must be maintained.

The CRC calculation starts from the reserved field of the hardwareParameters_t and ends before the hardwareParamsCrc field. Additional members to this structure may be added using the following method :

Add new fields before the reserved field. This method does not cause a CRC fail, but you must keep in mind to subtract the total size of the new fields from the size of the reserved field. For example, if a field of uint8_t size is added using this method, the size of the reserved field shall be changed to 63.

Co-locating application factory data in HW Parameters flash sector.

The sector containing the Hardware parameter structure may be located in the internal flash, usually at its last sector. The actual Hardware parameter structure has a size of 128 bytes - including padding reserved for future use. Since there is plenty of room available in a flash sector (4kB or 8kB), co-locating Application Factory Data in the same structure prevents from reserving another flash sector for these data. The application designer may adopt this solution by defining gHwParamsAppFactoryDataExtension_d as 1. A total of 2kB is alloted to this purpose.

If this option was chosen, whenever any of the Hardware parameter fields is modified, its CRC16 will change so the sector will need erasing. The gHwParamsAppFactoryDataPreserveOnHwParamUpdate_d compilation option deals with restoring the contents of the App Factory Data. Nonetheless this requires a temporary allocation a 2kB buffer to preserve the previous content and restore then on completion of the Hw Parameter update.

Special reserved area at start of IFR1 in range [0x02002000..0x02002600[

On development boards a 1536 byte area is reserved and the actual Hardware parameter area begins at offset 0x600. Preserving this area on a HW parameter update also requires a temporary 1.5kB dynamic allocation (in addition to the App Factory 2kB allocation), to be able to restore on completion of update operation.

HW Parameters Production Data placement options

The placement of production data (PROD_DATA) can be selected based on the definition of gHwParamsProdDataPlacement_c (see fwk_config.h). The productions data seldom need update for final products, once calibration data, MAC addresses or others have been programmed. Two cases exist, plus a transition mode :

1) gHwParamsProdDataMainFlashMode_c (0) :
   - PROD_DATA are located at top of Main Flash. Hardware parameters section is placed in the last sector of internal flash [0xfe000..0x100000[.
   - The linker script must reserve this area explicitly so as to prevent placement of NVM or text sections at that location by setting gUseProdInfoMainFlash_d.

2) gHwParamsProdDataMainFlash2IfrMode_c(1) : - PROD_DATA are located in IFR1, but MainFlash version still exists during interim period. - If the contents of the PROD_DATA section in MainFlash is valid (not blank and correct CRC) but the IFR PROD_DATA is still blank, copy the contents of MainFlash PROD_DATA to IFR location. - When done PROD_DATA in IFR are used. Once the transition is done, an application using (2: gHwParamsProdDataPlacementIfrMode_c) may be programmed.

3) gHwParamsProdDataIfrMode_c (2) :
   - PROD_DATA section dwells in the IFR1 sector [0x02002000..0x02004000[
   - in development phase the area comprised between [0x02002000..0x02002600[must be reserved for internal purposes.
   - This allows to free up the top sector of Main Flash by linking with gUseProdInfoMainFlash_d unset.

Low Power Low power reference user guide

This Readme file describes the connectivity software architecture and provides the general low power enablement user guide.

Connectivity Low Power Software architecture

The connectivity low power software architecture is composed of various components. These are described from the lower layer to the application layer:

1. The SDK power manager in component/power_manager. This component provides the basic low power framework. It is not specific to the connectivity but generic across devices. It performs the following functions:
   - Gathers the low power constraints for upper layer and takes the decision on the best suitable low power state the device is allowed to go to fullfill the constraints.
   - Calls the low power entry and exit function callbacks.
   - Calls the appropriate SW routines to switch the device into the suitable low power state.
2. Connectivity Low power module in the connectivity framework. This module is composed of:
   - The low power service called PWR inside framework/LowPower (this folder), This module is generic to all connectivity devices.
   - The platform lowpower: fwk_platform_lowpower.[ch] located in framework\platform\<platform_name>. These files are a collection of low power routines functions for the PWR module and upper layer. These are specific to the device.

   Both PWR and platform lowpower files are detailed in the section below.
3. Low power Application modules, which consists of three parts:
   - Application initialization file app_services_init.c where the application initializes the low power framework. See the next section 'Demo example for typical usage of low power framework'.
   - Application Idle task from application to call the main low power entry function PWR_EnterLowPower() to switch the device into low power. This function is application specific, one example is given in the section 1.3.3.
   - Low power board files : board_lp.[ch] located in board/lowpower. These files implement the low power entry and exit functions related to the application and board. Users must modify these files for their own needs. Example code is given for the connectivity applications.

User guide is provided in section 1.3 below.

Note : Linker script may also be impacted for power down mode support in order to provide an RAM area for ROM warm boot (depends on the platform) and application warmboot stack

The Low power central and master reference design applications provide an example of Low power implementation for BLE. Customer can also refer to the associated document 'low power connectivity reference design user guide'.

1.1 - SDK power manager

This module provides the main low power functionalities such as:

- Decide the best low-power mode dependent on the constraints set by upper layers by using PWR_SetLowPowerModeConstraints() API function.
- Handle the sequences to enter and exit low-power mode.
- Enable and configure wake up sources, call the application callbacks on low power entry/exit sequences.

The SDK power manager provides the capability for application and all components to receive low power constraints to the power. The Application does not set the low-power mode the device shall go into. When going to low power, the SDK power manager selects the best low-power mode that fits all the constraints.

As an example, if the low power constraint set from Application is Power Down mode, and no other constraint is set, the SDK power manager selects Power down mode, the next time the device enters low power. However, if a new constraint is set by another component, such as the SecLib module that operates Hardware encryption, the SecLib module would select WFI as additional low power constraint. Also, the SDK power manager selects this last low-power mode until the constraint is released by the SecLib module. It then reselects Power Down mode for further low power entry modes.

1.2 - PWR Low power module

The PWR module in the connectivity framework provides additional services for the connectivity stacks and applications on top of the SDK power manager.

It also provides a simple API for Connectivity Stack and Connectivity applications.

However, more advanced features such as configuring the wake-up sources are only accessible from the SDK Power Manager API.

In addition to the SDK Power Manager, the PWR module uses the software resources from lower level drivers but is independent of the platform used.

1.2.1 - Functional description

Initialization of the PWR module should be done through the $\mathrm{PWR_Init()}$ function. This is mainly to initialize the SDK Power Manager and the platform for low power. It also registers the PWR low power entry or exit callback function $\mathrm{PWR_LowpowerCb()}$ to the SDK Power Manager. This function is called back while entering and exiting the low power modes to perform mandatory save or restore operations for connectivity stacks. The application can perform extra optional save/restore operations in the $\mathrm{board_lp}$ file where it can register to the SDK Power Manager its own callback. This method is usually used to handle optional peripherals such as serial interfaces, GPIOs, and so on.The main entry function is $\mathrm{PWR_EnterLowPower()}$. It must be called from Idle task when no SW activity is required. The maximum duration for low power is given as argument $\mathrm{timeoutUs}$ in microseconds. This function checks the next Hardware event in the connectivity stack, typically the next Radio activity. A wakeup timer is programmed if the $\mathrm{timeoutUs}$ value is shorter than the next radio event timing. Passing a timeout of $\mathrm{0us}$ is interpreted as no timeout on the application side.

On device wakeup from low power state, the function returns the time duration the device has been in low power state.

Two APIs are provided to set and release low power state constraints: PWR_SetLowPowerModeConstraint() and PWR_ReleaseLowPowerModeConstraint(). These are helper functions. Users can use the SDK power manager directly if needed.

The PWR module also provides some API to be set as callbacks into other components to prevent from going to low power state. It can be used in following examples :

1. If a DMA is running, the module in charge of the DMA would need to set a constraint to avoid the system from going to a low power state when the RAM and system bus are no longer available.
2. If transfer is going on a peripheral, the drivers shall set a constraint to forbid low power mode.
3. If encryption is on going through an Hardware accelerator, the HW accelerator and the required ressources (clocks, etc), shall be kept active also by setting a constraints.

1.2.2 - Tickless mode support

This module also provides some routines functions PWR_SysticksPreProcess() and PWR_SysticksPostProcess() from PWR_systicks.c in order to support the tickless mode when using FreeRTOS. The tickless mode is the capability to suspend the periodic system ticks from FreeRTOS and keep timebase tracking using another low power counter. In this implementation, the Timer Manager and time_stamp component are used for this purpose.

Idle task shall call these functions PWR_SysticksPreProcess() and PWR_SysticksPostProcess()‘ before and after the call to the main low power entry function PWR_EnterLowPower().

Refer to the *framework/LowPower/PWR_systicks.c* file or Section 2.1 below for more information.

1.3 - Low power platform submodule

Low power platform module file fwk_platform_lowpower.c provides the necessary helper functions to support low power device initialization, device entry, and exit routines. These are platform and device specific. Typically, the PWR module uses the low power platform submodule for all low power specific routines.

The low power platform submodule is documented in the Connectivity Framework Reference Manual document and in the Connectivity Framework API document.

1.4 - Low power board files

Low power board files board_lp.[ch] are both application and board specific. Users should update this file to add new functions to include new used peripherals that require low power support. In the current SDK package, only Serial Manager over UART and button (IO toggle wake up source) are supported and demonstrated in the Bluetooth LE demo application.

Other peripherals that require specific action on low power entry and restore on low power exit should be added to low power board files. For more details, refer to section Low power board file update

2 - Low power Application user guide

This section provides a user guide to enable Low power on a connectivity application, It gives example of typical implementation for the initialization, Idle task function and low power entry/exit functions.

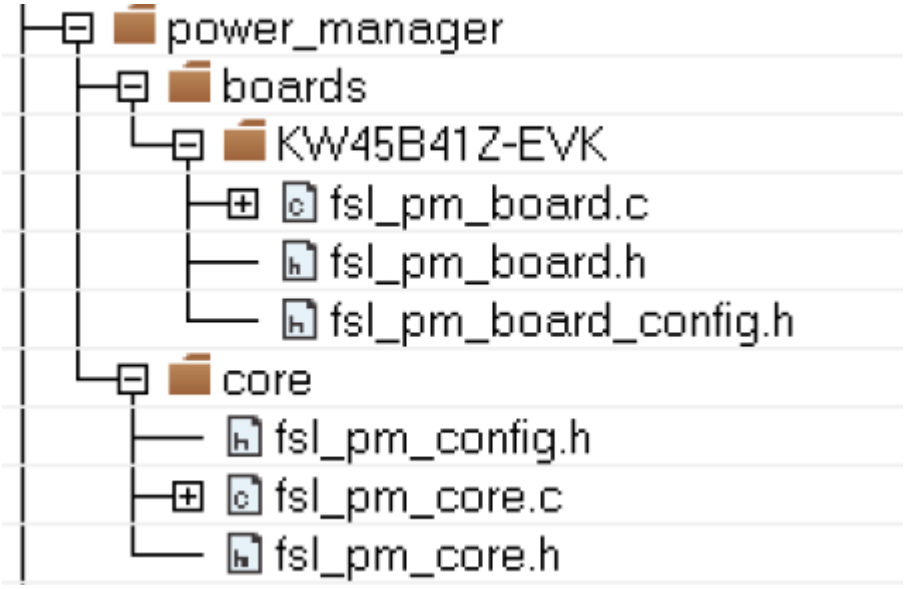
2.1 - Application Project updates

It is recommended to reuse the low-power peripheral/central reference design application projects as a start. This ensures that everything is in place for the low-power optimization feature. Then, application files may be added to one of the two projects.

However, users can start directly from the application project and implement low power in it, by performing the steps described in the following sections.

2.1.1 - SDK Power Manager

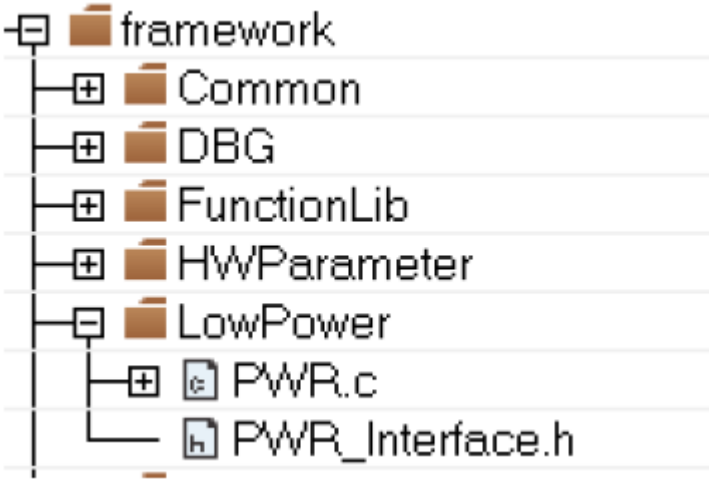
Most of the Low power functionality is implemented in the SDK Power Manager. The files to add into the project SDK power_manager module are listed in the figure below:



You need to use the files located in the folder that match your device.

2.1.2 - PWR connectivity framework module

PWR.c PWR_Interface.h shall be added to your application projects :

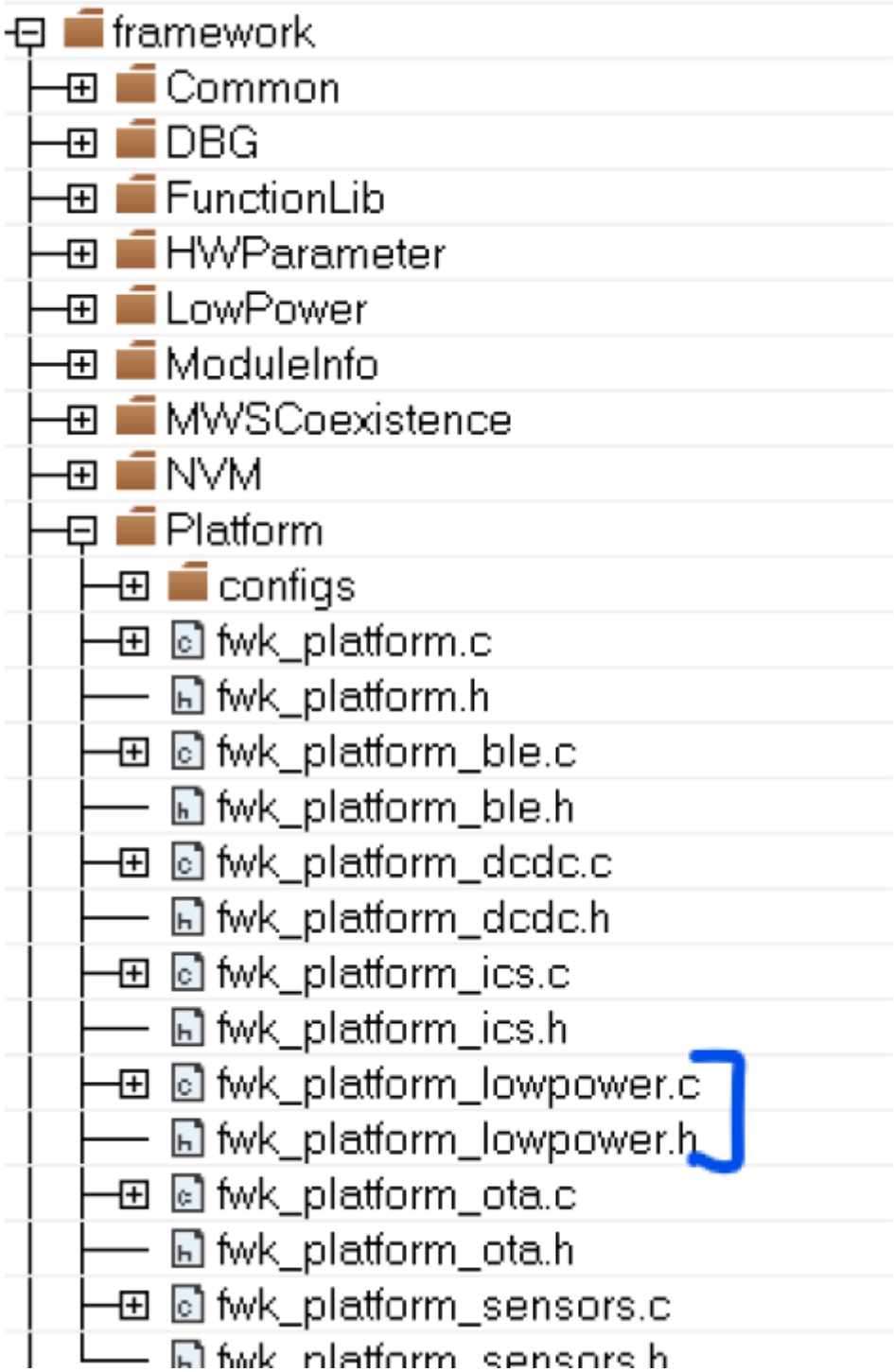


Optionally, in order to support Systick less mode, PWR_systicks.c or PWR_systicks_bm.c could also be added.

The include path to add is: middleware/wireless/framework/LowPower

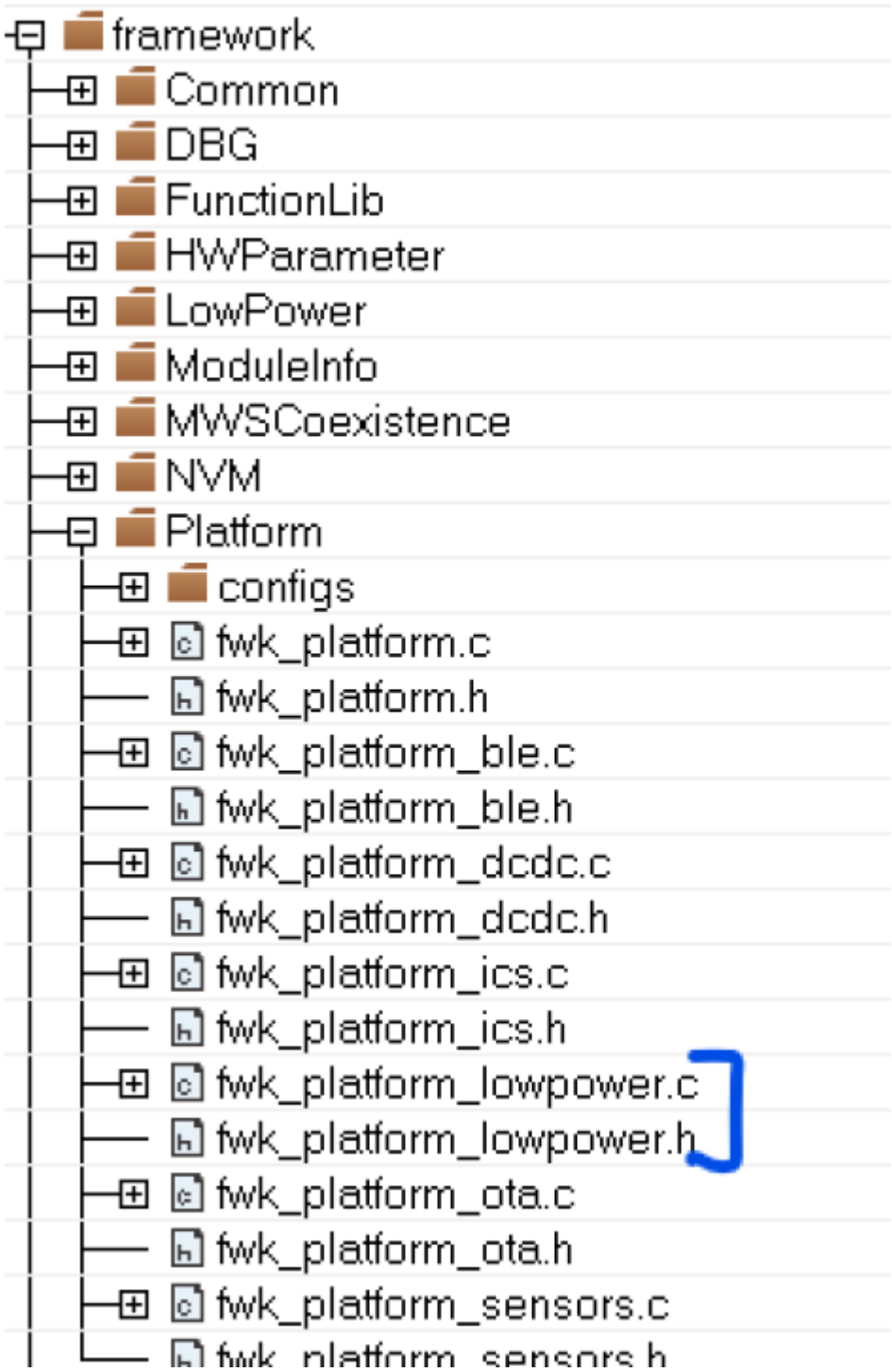
2.1.3 -Low power platform submodule

Low power platform files can be found in the 'Platform' module in the connectivity framework:



2.1.4 - Low power board files

These files are located in the same folder that the other board files board.[ch]. Hence, it is not required to add any new include path at compiler command line.



2.1.5 - Application RTOS Idle hook and tickeless hook functions

See section 2.4.3 Idle task implementation example

2.2 - Low power and wake up sources Initialization

Low power initialization and configuration are performed in APP_ServiceInitLowpower()function. This is called from APP_InitServices() function called from the main() function so all is already set up when calling the main application entry point, typically BluetoothLEHost_AppInit() function in the Bluetooth LE demo applications.

The default Low Power mode configured in APP_InitServices() is Deep Sleep mode. In Bluetooth

LE, (or any other stack technology), Deep Sleep mode fits for all use cases. For instance, for Bluetooth LE states: Advertising, Connected, Scanning states. This mode already performs a very good level of power saving and likely, this is not required to optimize more if the device is powered from external supply.

APP_ServiceInitLowpower() function performs the following initialization and configuration:

- Initialize the Connectivity framework Low power module PWR_Init(), this function initialized the SDK Power Manager.
- Configure the wakeup sources such as serial manager wake up source for UART, or button for IO wake up configuration. These are typical wakeup sources used in the connectivity application. Developer may want to add additional wake up sources here specific for the application.

  Note : The low power timer wakeup source and wakeup from Radio domain are directly enabled from the Connectivity framework Low power module PWR as it is mandatory for the connectivity stack. If your application supports other peripherals (such as i2c, spi, and others) that require wake sources from low power, developer should add additional wake up sources setting in this function APP_ServiceInitLowpower(). The complete list of wakeup sources are available from the SDK Power Manager component, see file fsl_pm_board.h in component/boards/<device_name>/.
- Initialize and register the Low power board file used to register and implement low power entry and exit callback function used for peripheral. This is done by calling the BOARD_LowPowerInit() function.
- Register low power Enter and exit critical function to driver component to enable / disable low power when the Hardware is active. Example is given for serial manager that needs to disable low power when the TX ring buffer contains data so the device does not enter low power until the buffer is empty.

Finally, APP_ServiceInitLowpower() function configures the Deep Sleep mode as the default low power constraint for the application. It is recommended to keep this level of low power constraint during all the connectivity stack initialization.

Example of low power framework initialization can be found in app_services_init.c file. Below is some code example for initializing the low power framework and wake up sources:

```
static void APP_ServiceInitLowpower(void)
{
    PWR_ReturnStatus_t status = PWR_Success;

    /* It is required to initialize PWR module so the application
     * can call PWR API during its init (wake up sources...) */
    PWR_Init();

    /* Initialize board_lp module, likely to register the enter/exit
     * low power callback to Power Manager */
    BOARD_LowPowerInit();

    /* Set Deep Sleep constraint by default (works for All application)
     *   Application will be allowed to release the Deep Sleep constraint
     *   and set a deepest lowpower mode constraint such as Power down if it needs
     *   more optimization */
    status = PWR_SetLowPowerModeConstraint(PWR_DeepSleep);
    assert(status == PWR_Success);

#if (defined(gAppButtonCnt_c) && (gAppButtonCnt_c > 0))

    /* Init and enable button0 as wake up source
     * BOARD_WAKEUP_SOURCE_BUTTON0 can be customized based on board configuration
```

(continues on next page)

(continued from previous page)

```
    * On EVK we use the SW2 mapped to GPIOD */
   PM_InitWakeupSource(&button0WakeUpSource, BOARD_WAKEUP_SOURCE_BUTTON0, NULL,␣
↪true);
#endif

#if (gAppButtonCnt_c > 1)
   /* Init and enable button1 as wake up source
    * BOARD_WAKEUP_SOURCE_BUTTON1 can be customized based on board configuration
    * On EVK we use the SW3 mapped to PTC6 */
   PM_InitWakeupSource(&button1WakeUpSource, BOARD_WAKEUP_SOURCE_BUTTON1, NULL,␣
↪true);
#endif

#if (defined(gAppUseSerialManager_c) && (gAppUseSerialManager_c > 0))

#if defined(gAppLpuart0WakeUpSourceEnable_d) && (gAppLpuart0WakeUpSourceEnable_d > 0)
   /* To be able to wake up from LPUART0, we need to keep the FRO6M running
    * also, we need to keep the WAKE domain is SLEEP.
    * We can't put the WAKE domain in DEEP SLEEP because the LPUART0 is not mapped
    * to the WUU as wake up source */
   (void)PM_SetConstraints(PM_LP_STATE_NO_CONSTRAINT, APP_LPUART0_WAKEUP_
↪CONSTRAINTS);
#endif

   /* Register PWR functions into SerialManager module in order to disable device lowpower
      during SerialManager processing. Typically, allow only WFI instruction when
      uart data are processed by serail manager  */
   SerialManager_SetLowpowerCriticalCb(&gSerMgr_LowpowerCriticalCBs);
#endif

#if defined(gAppUseSensors_d) && (gAppUseSensors_d > 0)
   Sensors_SetLowpowerCriticalCb(&app_LowpowerSensorsCriticalCBs);
#endif

   (void)status;
}
```

2.3 - low power entry/exit sequences : board files updates

Board Files that handles low-power are board_lp.[ch] files.

Low power board files implement the low-power callbacks of the peripherals to be notified when entering or exiting Low Power mode. This module also registers these low-power callbacks to the SDK Power Manager component to get the notifications when the device is about to enter low-power or exit Low Power mode. The Low-power callbacks are registered from BOARD_LowPowerInit() function. This function is called from app_services_init.c file after PWR module initialization.

The low power callback functions can be categorized in two groups:

- Entry Low power call back functions: These are usually used to prepare the peripherals to enter low-power. For example, they can be used for flushing FIFOs, switching off some clocks, and reconfiguring pin mux to avoid leakage on pins. In case of Power Down mode, these functions could be used to save the Hardware peripheral context.
- Exit Low power call back functions: These are typically used to restore the peripherals to functionality. Therefore, they perform the reverse of what is done by the entry callback functions: restoring the pin mux, re-enabling the clock, in case of Power Down mode, restoring the Hardware peripheral context, and so on.

Note that distinction can be done between clock gating mode (Deep Sleep mode), and power gated mode (Power down mode) when entering and exiting Low Power mode. The

BOARD_EnterLowPowerCb() and BOARD_ExitLowPowerCb() functions provide the code to call the various peripheral entry and exit functions to go and exit Deep Sleep mode: serial manager, button, debug console, and others.

However, the processing to save and restore the Hardware peripheral is implemented in different functions BOARD_EnterPowerDownCb() and BOARD_ExitPowerDownCb(). These two functions must be called when exiting the power gated modes of the power domain. These two should implement specific code for such case (likely the complete reinitialization of each peripheral). In order to know the Low Power mode that the wake up domain, or main domain has been entered, the low-power platform API PLATFORM_GetLowpowerMode() can be called.

> **Note** : The BOARD_ExitPowerDownCb() API is called before BOARD_ExitLowPowerCb() as it is generally required to restore the Hardware peripheral contexts before reconfiguring the pin mux to avoid any signal glitches on the pads.

Also, it is important to know whether the location of the hardware peripheral is in the main domain or wake up domain. The two power domains can go into different power modes with the limitation that the wakeup domain cannot go to a deepest Low Power mode than the main domain. Depending on the constraint set on SDK Power Manager, the wake up domain could remain in active while the main domain can go to deep sleep or power down modes. In this case, the peripherals in the wake up domain does not required to be restored, as explained in the **Power Down** section. Likely, only pin mux reconfiguration is required in this case.

Example The Low power entry and exit functions must be registered with the SDK Power Manager so that these functions are called when the device enters and exits low power mode. This is done by BOARD_LowPowerInit() typically called from the application source code in the app_services_init.c file.

```
static pm_notify_element_t boardLpNotifyGroup = {
    .notifyCallback = BOARD_LowpowerCb,
    .data           = NULL,
};

void BOARD_LowPowerInit(void)
{
    status_t status;

    status = PM_RegisterNotify(kPM_NotifyGroup2, &boardLpNotifyGroup);
    assert(status == kStatus_Success);
    (void)status;
}
```

The BOARD_LowpowerCb() callback function handles both the entry and exit sequences. An argument is passed to the function to indicate the low power state the device enters or exits. A typical implementation is given below. Users must ensure to differentiate low power entry and exit, and the various low power states.

Typically, nothing is expected to be done if low power state is WFI or Sleep mode. These modes are some light low power states and the system can be woken up by interrupt trigger.

In Deep sleep mode, the clock tree and the source clocks are off, and the system needs to be woken up from an event from the WUU module.

In Power down mode, some peripherals are likely to be powered off, context save and restore may need to be done in these functions.

```
static status_t BOARD_LowpowerCb(pm_event_type_t eventType, uint8_t powerState, void *data)
{
    status_t ret = kStatus_Success;
    if (powerState < PLATFORM_DEEP_SLEEP_STATE)
    {
        /* Nothing to do when entering WFI or Sleep low power state
```

(continues on next page)

(continued from previous page)

```
        NVIC fully functionnal to trigger upcoming interrupts */
    }
    else
    {
        if (eventType == kPM_EventEnteringSleep)
        {
            BOARD_EnterLowPowerCb();

            if (powerState >= PLATFORM_POWER_DOWN_STATE)
            {
                /* Power gated low power modes often require extra specific
                 * entry/exit low power procedures, those should be implemented
                 * in the following BOARD API */
                BOARD_EnterPowerDownCb();
            }
        }
        else
        {
            /* Check if Main power domain domain really went to Power down,
             *   powerState variable is just an indication, Lowpower mode could have been skipped by an␣
↪immediate wakeup
             */
            PLATFORM_PowerDomainState_t main_pd_state = PLATFORM_NO_LOWPOWER;
            PLATFORM_status_t           status;

            status = PLATFORM_GetLowpowerMode(PLATFORM_MainDomain, &main_pd_state);
            assert(status == PLATFORM_Successful);
            (void)status;

            if (main_pd_state == PLATFORM_POWER_DOWN_MODE)
            {
                /* Process wake up from power down mode on Main domain
                 *  Note that Wake up domain has not been in power down mode */
                BOARD_ExitPowerDownCb();
            }

            BOARD_ExitLowPowerCb();
        }
    }
    return ret;
}
```

2.4 - Low power constraint updates and optimization

Except for the board file update as seen in previous section, the application does not need any other changes for low-power support in Deep Sleep mode. It shall work as if no low-power is supported. However, If more aggressive power saving is required, this constraint can be changed in your application in order to further reduce the power consumption in Low Power mode.

2.4.1 - Changing the Default Application low power constraint after firmware initialization

The Low power reference design applications (central or peripheral) provides demonstration on how to change the Application low power constraint. In the Application main entry point BluetoothLEHost_AppInit(), Deep Sleep mode is configured by default from APP_ServiceInitLowpower() function.

> **Note :** It is recommended to keep Deep Sleep mode as default during all the stack initialization phase until BluetoothLEHost_Initialized() and BleApp_StartInit() functions are called. In case of Bonded device with privacy, it is recommended to wait for gControllerPrivacyStateChanged_c event to be called.

The BleApp_LowpowerInit() function provides an example of code on how to release the default Deep sleep low-power constraint and set a new constraint such as Power down mode for the application. This deeper low-power mode is used when no Bluetooth LE activity is ongoing, and if no other higher Low-power constraint is set by another component or layer. For instance, if some serial transmission is ongoing by the serial manager, or if the SecLib module has on going activity on the HW crypto accelerator, the low-power mode could be less deep.

```
static void BleApp_LowpowerInit(void)
{
#if defined(gAppLowpowerEnabled_d) && (gAppLowpowerEnabled_d>0)
    PWR_ReturnStatus_t status;

    /*
     * Optionally, Allow now Deepest lowpower mode constraint given by gAPP_
↪LowPowerConstraintInNoBleActivity_c
     *    rather than DeepSleep mode.
     * Deep Sleep mode constraint has been set in APP_InitServices(), this is fine
     *    to keep this constraint for typical lowpower application but we want the
     *     lowpower reference design application to be more agressive in term of power saving.

     *   To apply a lower lowpower mode than Deep Sleep mode, we need to
     *     - 1) First, release the Deep sleep mode constraint previously set  by default in app_services_init()
     *     - 2) Apply new lowpower constraint when No BLE activity
     *   In the various BLE states (advertising, scanning, connected mode), a new Lowpower
     *     mode constraint will be applied depending of Application Compilation macro set in app_preinclude.
↪h :
     *     gAppPowerDownInAdvertising, gAppPowerDownInConnected, gAppPowerDownInScanning
     */

    /*  1) Release the Deep sleep mode constraint previously set  by default in app_services_init() */
    status = PWR_ReleaseLowPowerModeConstraint(PWR_DeepSleep);
    assert(status == PWR_Success);
    (void)status;

    /*  2) Apply new Lowpower mode constraint gAppLowPowerConstraintInNoBleActivity_c *
     *       The BleAppStart() call above has already set up the new lowpower constraint
     *         when Advertising request has been sent to controller        */
    BleApp_SetLowPowerModeConstraint(gAppLowPowerConstraintInNoBleActivity_c);
#endif
}
```

2.4.2 - Changing the Application lowest low power constraint during application execution

In the various application use cases, (in the various Bluetooth LE activity states, advertising, connected, scanning), some lower low-power constraint can be set, as Power down for advertising, Deep Sleep for connected, or Scanning. Customer can change the level of Low Power mode in the various use case mainly depending of the time duration the device is supposed to remain in low-power. The longer the time that the device remains in low power, the higher the benefit for a deeper Low Power mode such as Power down mode. However, please note that the wake up from power down mode takes significantly more time than deep sleep as ROM code is re executed and the hardware logic needs to be restored. Sections Deep Sleep and Power Down provide some guidance on when to use Deep Sleep mode or Power Down modes respectively.

In the low power reference design applications, four application compilations macros are defined to adjust the low power mode into advertising, scanning, connected, or no Bluetooth LE activity. Other use cases can be added as desired. For instance, If application needs to run a DMA transfer, or if application needs to wakeup regularly to process data from external device, it may be useful to set WFI constraint (in case of DMA transfer), or Deep Sleep constraint (in case of regular wake up to process external data), rather than power down or a even lower low-power mode.

The four application compilation macros can be found in app_preinclude.h file of the project. See

app_preinclude.h for low power reference design peripheral application:

```
/*! Lowpower Constraint setting for various BLE states (Advertising, Scanning, and Connected mode)
   The value shall map with the type defintion PWR_LowpowerMode_t in PWR_Interface.h
     0 : no LowPower, WFI only
     1 : Reserved
     2 : Deep Sleep
     3 : Power Down
     4 : Deep Power Down
   Note that if a Bluetooth LE State is configured to Power Down mode, you must ensure that the
      gLowpowerPowerDownEnable_d variable is set to 1 in the Linker Script.
   The PowerDown mode allows lowest power consumption but the wake-up time is longer and the first 16K␣
↪in SRAM is reserved to ROM code (this section will be corrupted on        each power down wake-up so␣
↪only temporary data could be stored there.)
       Power down feature not supported. */

#define gAppLowPowerConstraintInAdvertising_c`          3
/* Scanning not supported on peripheral */
//#define gAppLowPowerConstraintInScanning_c            2
#define gAppLowPowerConstraintInConnected_c             2
#define gAppLowPowerConstraintInNoBleActivity_c         4
```

In lowpower_central.c and the lowpower_preripheral.c files, the application sets and releases the low power constraint from BleApp_SetLowPowerModeConstraint() and BleApp_ReleaseLowPowerModeConstraint() functions. These functions are called with the macro value passed as argument.

> **Important Note :** Setting the application low power constraint must be done on new Bluetooth LE state request so the new constraint is applied immediately. In contrast, the application low-power mode constraint must be released when the Bluetooth LE state is exited. For example, setting the new low power constraint for Advertising must be done when the application requests advertising to start. Releasing the low power constraint must be done in the advertising stop callback (advertising has been stopped).

After releasing the low power constraint, the previous low power constraint, (likely the one that has been set during firmware initialization in APP_ServiceInitLowpower() function, or the updated low power constraint in BleApp_StartInit() function) applies again.

2.4.3 - Idle task implementation example

2.4.3.1 Tickless mode support and Low power entry function

Idle task configuration from FreeRTOS must be enabled by using the parameter configUSE_TICKLESS_IDLE in FreeRTOSConfig.h. This has the effect to have vPortSuppressTicksAndSleep() called from Idle task created by FreeRTOS. Here is a typical implementation of this function:

```
void vPortSuppressTicksAndSleep(TickType_t xExpectedIdleTime)
{

    bool abortIdle = false;
    uint64_t actualIdleTimeUs, expectedIdleTimeUs;

    /* The OSA_InterruptDisable() API will prevent us to wakeup so we use
     * OSA_DisableIRQGlobal() */
    OSA_DisableIRQGlobal();

    /* Disable and prepare systicks for low power */
    abortIdle = PWR_SysticksPreProcess((uint32_t)xExpectedIdleTime, &expectedIdleTimeUs);
```

(continues on next page)

(continued from previous page)

```
    if (abortIdle == false)
    {
        /* Enter low power with a maximal timeout */
        actualIdleTimeUs = PWR_EnterLowPower(expectedIdleTimeUs);

        /* Re enable systicks and compensate systick timebase */
        PWR_SysticksPostProcess(expectedIdleTimeUs, actualIdleTimeUs);
    }

    /* Exit from critical section */
    OSA_EnableIRQGlobal();
}
```

2.4.3.2 Connectivity background tasks and Idle hook function example

A few processes need to be run in the background before going into low power. This is the case for writing in NVM, or firmware update OTA to be writen in Flash. If so, configUSE_IDLE_HOOK must be enabled in FreeRTOSCOnfig.h. This enables vApplicationIdleHook() to be called prior to vPortSuppressTicksAndSleep(). Typical implementation of vApplicationIdleHook() function can be found here :

```
void vApplicationIdleHook(void)
{
    /* call some background tasks required by connectivity */
#if ((gAppUseNvm_d) || \
    (defined gAppOtaASyncFlashTransactions_c && (gAppOtaASyncFlashTransactions_c > 0)))

    if (PLATFORM_CheckNextBleConnectivityActivity() == true)
    {
        BluetoothLEHost_ProcessIdleTask();
    }
#endif
}
```

PLATFORM_CheckNextBleConnectivityActivity() function implemented in low power platform file fwk_platform_lowpower.c typically checks the next connectivity event and returns true if there is enough time to perform time consuming tasks such as flash erase/write operations (These operations can be defined by the compile macro depending on the platform).

2. Low power features

2.1 - FreeRTOS systicks

Low power module in the framework supports the systick generation for FreeRTOS. Systicks in FreeRTOS are generally not required in the Bluetooth LE demos applications. The framework already supports timers through the timer manager component, so the application can use the timers from this module. The systicks in FreeRTOS are useful for all internal timer service provided by FreeRTOS (through OSA) such as OSA_TimeDelay(), OSA_TimeGetMsec(), OSA_EventWait(). When systicks are enabled, an interrupt (systick interrupt) is triggered and executed on a periodic basis. In order to save power, periodic systick interrupts are undesirable and thus disabled when going to low-power mode. This feature is called low power FreeRTOS tickless mode. When entering the low power state, the system ticks must be disabled and switch to a low power timer. On wake-up, the module retrieves the time passed in low power and compensates the ticks count accordingly. This feature does not apply on bare metal scheduler.

On FreeRTOS, the vPortSuppressTicksAndSleep() function implemented in the app_low_power.c file is called when going to idle state. FreeRTOS provides to this function the xExpectedIdleTime, which is the time in tick periods before a task is due to be moved into the Ready state. This function manages the systicks (disable/enable) through PWR_SysticksPreProcess() and

PWR_SysticksPostProcess() calls. Then, when calling PWR_EnterLowPower(), a time out duration in micro seconds is given and the function sets a timer before entering low power. In addition, this function returns the low power period duration, used to compensate the ticks count.

In our example low power reference design peripheral application, a function OSA_EventWait() has been added to demonstrate the tickless mode feature. You can adjust the timeout with the gAppTaskWaitTimeout_ms_c flag in the app_preinclude.h file, its value in our demo is 8000ms. So 8 seconds after stopping any activity we will wake up from low power. If the flag is not defined in the application, its value is osaWaitForever_c and there will be no OS wake up.

2.2 - Selective RAM bank retention

To optimize the consumption in low power, the linker script specific function PLATFORM_GetDefaultRamBanksRetained() is implemented. This function obtains the RAM banks that need to be retained when the device goes in low power, in order to set them with PLATFORM_SetRamBanksRetained() function. The RAM banks that are not needed are set in power off state, when the device goes in low power mode.

The function PLATFORM_GetDefaultRamBanksRetained() is linker script specific. Hence, it cannot be adapted for a different application. If these functions are called from the board_lp.c, it is possible to give to PLATFORM_SetRamBanksRetained() a different bank_mask adapted to your specific application.

In deep power down mode, this feature does not have any impact because in this power mode, all RAM banks are already powered off.

3 - Low power modes overview

PWR module API provides the capability to set low power mode constraints from various components or from the application. These constraints are provided to the SDK Power Manager. Upper layer (such as all Application code, connectivity stacks) can call directly the SDK Power Manger if it requires more advanced tuning. The PWR API can be found in PWR_Interface.h.

> **Note :** 'Upper layer' signifies all layers, applications, components, or modules that are above the connectivity framework in the Software architecture.
>
> **Note :** Each power domain has its own low power mode capability. The low power modes described below are for the main domain and it is supposed that the wake up domain goes to the same low power mode. This is not always true as the wake up domain that contains some wake up peripheral can go to a lower low power mode state than the main domain so the peripherals in the wake up domain can remain operational when the main domain is in low power mode (deep sleep or power down modes). In this case, the context of the Hardware peripheral located in the wake up domain does not need to be saved and restored as for the peripherals located in the main domain

3.1 Wait for Interrupt (WFI)

Definition

In the Wait for Interrupt (WFI) state, the CPU core is powered on, but is in an idle mode with the clock turned OFF.

Wake up time and typical use case

The wakeup time from this low power mode is insignificant because the Fast clock from FRO is still running.

The WFI low power mode is mainly used when there is an hardware activity while the Software runs the Idle task. This allows the code execution to be temporarily suspende, thus reducing a bit the power consumption of the device by switching off the processor clock. When an interrupt fires, the processor clock is instantaneously restored to process the Interrupt Service Routine (ISR).

Usage

In order to prevent the software from programming the device to go to a lower low power mode (such as Deep Sleep, Power Down mode or Deep Power Down mode), the component responsible for the hardware drivers must call PWR_SetLowPowerModeConstraint(PWR_WFI) function. When the Hardware activity is completed, the component shall release the constraint by calling PWR_ReleaseLowPowerModeConstraint(PWR_WFI).

Alternatively, the component can call PWR_LowPowerEnterCritical() and then PWR_LowPowerExitCritical() functions.

For fine tuning of the low power mode allowing more power saving, the component can call directly the SDK Power Manager API with PM_SetConstraints() function using the appropriate low power mode and low power constraint. However, this is reserved for more advanced users who know the device very well. It is not recommended to do so.

The PWR module has no external dependencies, so the low-power entry and exit callback functions must be defined by the user for each peripheral that has specific low power constraints It is consequently convenient to register to the component the low power callbacks structure that is used for entering and exit low power critical sections. In Bluetooth LE, you can take the example in the app_conn.c file as shown below:

```
#if defined(gAppLowpowerEnabled_d) && (gAppLowpowerEnabled_d>0)
static const Seclib_LowpowerCriticalCBs_t app_LowpowerCriticalCBs =
{
  .SeclibEnterLowpowerCriticalFunc = &PWR_LowPowerEnterCritical,
  .SeclibExitLowpowerCriticalFunc  = &PWR_LowPowerExitCritical,
};
#endif

void BluetoothLEHost_Init(..)
{
 ...
      /* Cryptographic hardware initialization */
      SecLib_Init();
#if defined(gAppLowpowerEnabled_d) && (gAppLowpowerEnabled_d>0)
      /* Register PWR functions into SecLib module in order to disable device lowpower
         during Seclib processing. Typically, allow only WFI instruction when
         commands (key generation, encryption) are processed by Seclib  */
      SecLib_SetLowpowerCriticalCb(&app_LowpowerCriticalCBs);
#endif
 ...
}
```

Limitations

No limitation when using the WFI mode.

3.2 Sleep mode

Sleep mode is similar to WFI low power mode but with some additional clock gating. The Sleep mode is device-specific. Therefore, you may refer to the hardware Reference manual of the device for more information.

3.2 Deep Sleep mode

Definition

In Deep Sleep mode, the fast clock is turned off, and the CPU along with the main power domain are placed into a retention state, with the voltage being scaled down to support state retention only. Because no high frequency clock is running, the voltage applied on the power domain can be reduced to reduce leakage on the hardware logic. This reduces the overall power consumption in the Deep Sleep mode. When waking up from Deep sleep mode, the core voltage is

increased back to nominal voltage and the fast clock (FRO) is turned back on, the peripheral in this domain can be reused as normal.

To same more additional power, Some unused RAM banks can be powered off. this prevents from having current leakage and consequently, allow to reduce even more the power consumption in Deep SLeep mode. This is achieved by calling PLATFORM_SetRamBanksRetained() from low power entry function from the board_lp.c file.

Usage

All firmware is able to implement Deep Sleep mode transparently to the application due to the PWR module, low power platform submodule, and the low power board file. This is described in the section Low-power implementation.

When entering this mode, it is recommended to turn the output pins into input mode, or high impedance to reduce leakage on the pads. This is typically done in pin_mux.c file, called from board.c file and executed from the low power callback in board_lp.c file. As an example, the TX line of the UART peripheral can be turned to disabled so it prevents the current from being drawn by the pad in low power mode.

Wake up time and typical use case

The wake up time is very fast, it takes mostly the time for the Fast FRO to start up again (couple of hundreds of microseconds). Therefore, this mode is a very good balance between power consumption in low-power mode and wake up latency. This mode is used extensively in most of the use cases of the application.

Limitations

In Deep Sleep mode, the clock is disabled to the CPU and the main peripheral domain, so peripheral activity (for example, an ongoing DMA transfer) is not possible in Deep Sleep mode.

3.3 Power down mode

Definition

In power-down mode, both the clock, and power are shut off to the CPU and the main peripheral domain. The SRAM is retained but register values are lost. The SDK Power Manager handles the restore of the processor registers and dependencies such as interrupt controller and similar ones transparently from the application.

Usage

The application, with the help of the low power board files, saves and restores the peripherals that were located in the power domain during the entry and exit of the power down mode. This is done from low power board_lp files in the entry/exit low power callbacks. An example is given for the serial manager and debug console in board_lp.c file in the function BOARD_ExitPowerDownCb().

If the device contains a dedicated wake up power domain where some wake up peripherals are located, if this wake up domain is not turned into power down mode but only Deep sleep mode or active mode, this peripheral does not need for a save and restore on low power entry/exit. For instance, on KW45, This is basically achieved when enabling the wakeup source of the peripheral *PWR_EnableWakeUpSource()* from APP_ServiceInitLowpower() function. Alternatively, this can be directly achieved by setting the constraint to the SDK power manager by calling PM_SetConstraints(), (use APP_LPUART0_WAKEUP_CONSTRAINTS for wakeup from UART constraint).

On exit from low power, The low power state of power domain can be retrieved by Platform API PLATFORM_GetLowpowerMode(). This API shall be called from low power exit callback function only.

As for Deep Sleep mode, software shall configure the output pins into input or high impedance during the Low Power mode to avoid leakage on the pads.

Wake up time and typical use case

The wake up time is significantly longer than wake up time from Deep Sleep (from several hundreds of micro-seconds to a couple of milliseconds depending on the platform). On some platform, it can takes longer, for instance, if ROM code is implemented and perform authentication checks for security and hardware logic in power domain needs to be restored (case for KW45). However, After ROM code execution, the SDK power manager resumes the Idle task execution from where it left before entering low-power mode. Hence, the wakeup time from this mode is still significantly lower that the initialization time from a power on reset or any other reset.

Depending on the wakeup time of the platform and the low power time duration, This mode is recommended when no Software activity is expected to happen for the next several seconds. In Bluetooth LE, this mode is preferred in advertising or without Bluetooth LE activity. However, in scanning or connected mode, Regular wakes up happens regularly for instance to retrieve HCI message responses from the Link layer, the Deep Sleep mode is rather recommended.

Limitations

In addition to the Deep Sleep limitation (no Hardware processing on going when going to Power down mode) and the significant increase of the wake time, the Power Down mode requires the ROM code to execute and this last uses a significant amount of memory in the SRAM.

Typically, The first SRAM bank (16 KBytes) is used by the ROM code during execution so the Application firmware can use this section of SRAM for storing bss, rw data, or stacks. Only temporary data could be stored here and this location is overwritten on every Power Down exit sequence.

In order to avoid placing firmware data section (bss, rw, etc.) in the first SRAM bank, the linker script variable *gLowpowerPowerDownEnable_d* should be set to 1. Setting the linker script variable to avoid placing firmware data section in the first SRAM bank, The effect of setting this flag is to prevent the firmware from using the first 16 KB in SRAM.

> *Note* : This setting is ONLY required if the application implements Power Down mode.
> If Application uses other low-power mode, this is not required.

3.4 Deep Power-down mode

Definition

In Deep Power Down mode, the SRAM memory is not retained. This power mode consumes the lowest power and is exited through a reset sequence.

Usage

In addition to the Power Down limitation, the Deep Power Down mode shuts down all the memory in the SRAM. Since it is exited through a reset sequence, the wake up time is also longer.

Wake up time and typical use case

This low-power mode is exited through a reset sequence. Consequently, the wake up time is longer than in any other mode. In Bluetooth LE, using this mode is possible when there is no Bluetooth LE activity. It is preferable to use this mode if there is no Bluetooth LE activity before a certain time interval.

Limitations

All the memory in SRAM is shut down in the Deep Power Down mode. The main limitation in going to this low-power mode is that the context is not saved.

ModuleInfo Overview

The ModuleInfo is a small Connectivity Framework module that provides a mechanism that allows stack components to register information about themselves.

The information comprises :

- Component or module name (for example: Bootloader, IEEE 802.15.4 MAC, and Bluetooth LE Host) and associated version string
- Component or module ID
- Version number
- Build number

The information can be retrieved using shell commands or FSCI commands.

Detailed data types and APIs used in ConnFWK_APIs_documentation.pdf.

NVM: Non-volatile memory module Overview

In a standard Harvard-architecture-based MCU, the flash memory is used to store the program code and program constant data. Modern processors have a built-in flash memory controller that can be used under user program execution to store non-volatile data. The flash memories have individually erasable segments (sectors) and each segment has a limited number of erase cycles. If the same segments are used to store various kinds of data all the time, those segments quickly become unreliable. Therefore, a wear-leveling mechanism is necessary to prolong the service life of the memory. The NVM module in the connectivity framework provides a file system with a wear-leveling mechanism, described in the subsequent sections. The *NvIdle()* function handles the program and erase memory operations. Before resetting the MCU, *NvShutdown()* must be called to ensure that all save operations have been processed.

NVM boundaries and linker script requirement

Most of the MCUs have only a standard flash memory that the non-volatile (NV) storage system uses. The amount of memory that the NV system uses for permanent storage and its boundaries are defined in the linker configuration file though the following linker symbols :

- NV_STORAGE_START_ADDRESS
- NV_STORAGE_END_ADDRESS
- NV_STORAGE_MAX_SECTORS
- NV_STORAGE_SECTOR_SIZE

The reserved memory consists of two virtual pages. The virtual pages are equally sized and each page is using one or more physical flash sectors. Therefore, the smallest configuration is using two physical sectors, one sector per virtual page.

NVM Table

The Flash Management and Non-Volatile Storage Module holds a pointer to a RAM table. The upper layers of this table register information about data that the storage system should save and restore. An example of NVM table entry list is given below.

pData	ElemCount	ElemSize	EntryId	EntryType
0x1FFF9000	3	8	0xF1F4	MirroredInRam
0x1FFF7640	5	4	0xA2A6	NotMirroredInRam
0x1FFF1502	6	1	0x4212	NotMirroredInRam AutoRestore
0x1FFFF200	2	6	0x118F	MirroredInRam

NVM Table entry

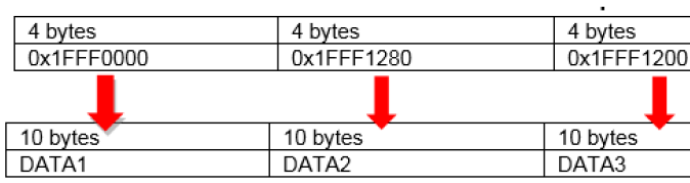
As show above, A NVM table entry contains a generic pointer to a contiguous RAM data structure, the number of elements the structure contains, the size of a single element, a table entry ID, and an entry type.

A RAM table entry has the following structure:

- pData (4 bytes) is a pointer to the RAM memory location where the dataset elements are stored.
- elemCnt (2 bytes) represents how many elements the dataset has.
- elemSz (2 bytes) is the size of a single element.
- entryID is a 16-bit unique ID of the dataset.
- dataEntryType is a 16-bit value representing the type of entry (mirrored/unmirrored/unmirrored auto restore).

For mirrored datasets, pData must point directly to the RAM data. For unmirrored datasets, it must be a double pointer to a vector of pointers. Each pointer in this table points to a RAM/FLASH area. Mirrored datasets require the data to be permanently kept in RAM, while unmirrored datasets have dataset entries either in flash or in RAM. If the unmirrored entries must be restored at the initialization, NotMirroredInRamAutoRestore should be used. The entryID gUnmirroredFeatureSet_d should be set to 1 for enabling unmirrored entries in the application. The last entry in the RAM table must have the entryID set to gNvEndOfTableId_c.

pData	0x1FFF8000
elemCnt	4
elemSz	10
entryID	1
dataEntryType	gNVM_NotM gNVM_NotM



The figure below provides an example of table entry :

When the data pointed to by the table entry pointer (pData) has changed (entirely or just a single element), the upper layers call the appropriate API function that requests the storage system to save the modified data. All the save operations (except for the synchronous save and atomic save) and the page erase and page copy operations are performed on system idle task. The application must create a task that calls NvIdle in an infinite loop. It should be created with OSA_PRIORITY_IDLE. However, the application may choose another priority. The save operations are done in one virtual page, which is the active page. After a save operation is performed on an unmirrored dataset, pData points to a flash location and the RAM pointer is freed. As a result, the effective data should always be allocated using the memory management module.

Active page

The active page contains information about the records and the records. The storage system can save individual elements of a table entry or the entire table entry. Unmirrored datasets can only have individual saves. On mirrored datasets, the save/restore functions must receive the pointer to RAM data. For example, if the application must save the third element in the above vector, it should send 0x1FFF8000 + 2 * elemSz. For unmirrored datasets, the application must send the pointer that points to the area where the data is located. For example, if the application must save the third element in the above vector, it should send 0x1FFF8000 + 2 * sizeof(void*).

The page validity is guaranteed by the page counter. The page counter is a 32-bit value and

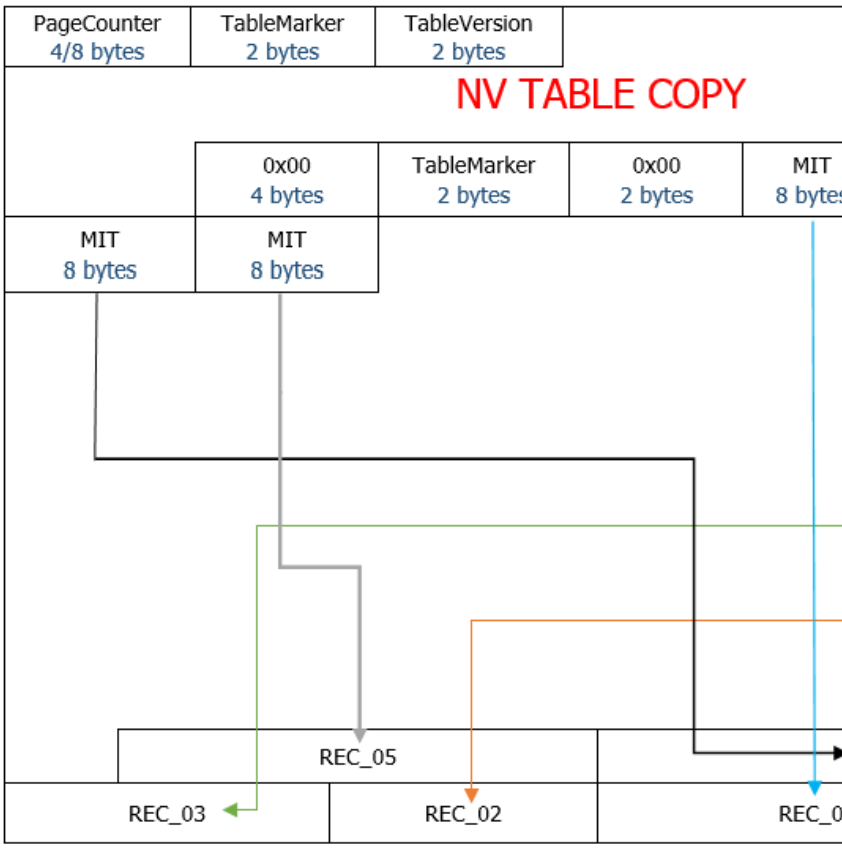
is written at the beginning and at the end of the active page. The values need to be equal to consider the page a valid one. The value of the page counter is incremented after each page copy operation. A page erase operation is performed when the system is formatted. It is also performed when the page is full and a new record cannot be written into that page. Before being erased, the full page is first copied (only the most recent saves) and erased afterward.

The validity of the Meta Information Tag (MIT), and, therefore, of a record, is guaranteed by the MIT start and stop validation bytes. These two bytes must be equal to consider the record referred by the MIT valid. Furthermore, the value of these bytes indicates the type of the record, whether it is a single element or an entire table entry. The nonvolatile storage system allows dynamic changes of the table within the RAM memory, as follows:

- Remove table entry
- Register table entry

A new table entry can be successfully registered if there is at least one entry previously removed or if the NV table contains uninitialized table entries, declared explicitly to register new table entries at run time. A new table entry can also replace an existing one if the register table entry is called with an overwrite set to true. This functionality is disabled by default and must be enabled by the application by setting gNvUseExtendedFeatureSet_d to 1.

The layout of an active page is shown below:



As shown above, the table stored in the RAM memory is copied into the flash active page, just after the table version. The “table start” and “table end” are marked by the table markers. The data pointers from RAM are not copied. A flash copy of a RAM table entry has the following structure:

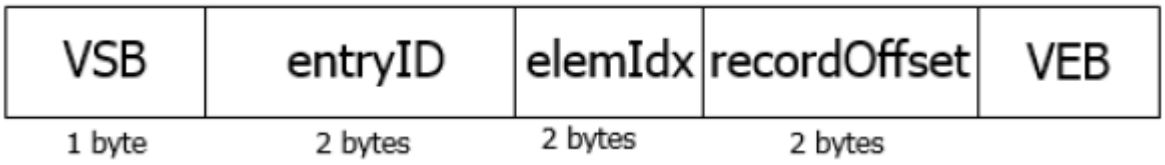
Where:

- entryID is the ID of the table entry

- entryType represents the type of the entry (mirrored/unmirrored/unmirrored auto restore)
- elemCnt is the elements count of that entry
- elemSz is the size of a single element

This copy of the RAM table in flash is used to determine whether the RAM table has changed. The table marker has a value of 0x4254 ("TB" if read as ASCII codes) and marks the beginning and end of the NV table copy.

After the end of the RAM table copy, the Meta Information Tags (MITs) follow. Each MIT is used to store information related to one record. An MIT has the following structure:

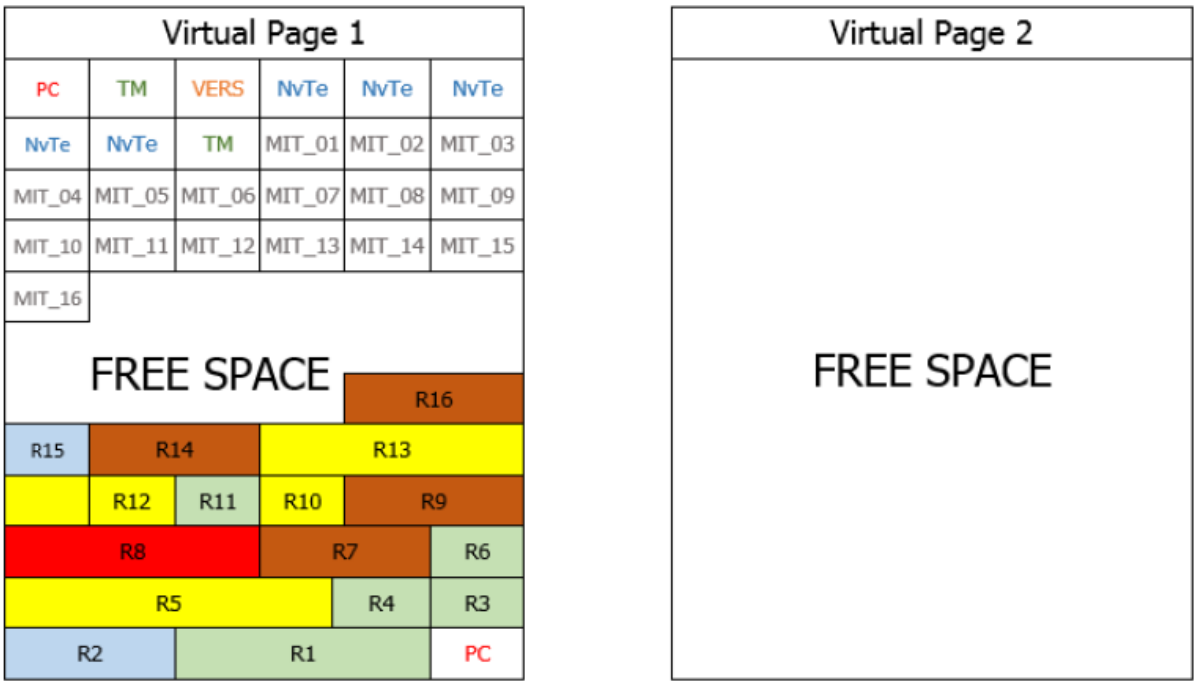


Where:

- VSB is the validation start byte.
- entryID is the ID of the NV table entry.
- elemIdx is the element index.
- recordOffset is the offset of the record related to the start address of the virtual page.
- VEB is the validation end byte.

A valid MIT has a VSB equal to a VEB. If the MIT refers to a single-element record type, VSB=VEB=0xAA. If the MIT refers to a full table entry record type (all elements from a table entry), VSB=VEB=0x55. Because the records are written to the flash page, the available page space decreases. As a result, the page becomes full and a new record does not have enough free space to be copied into that page.

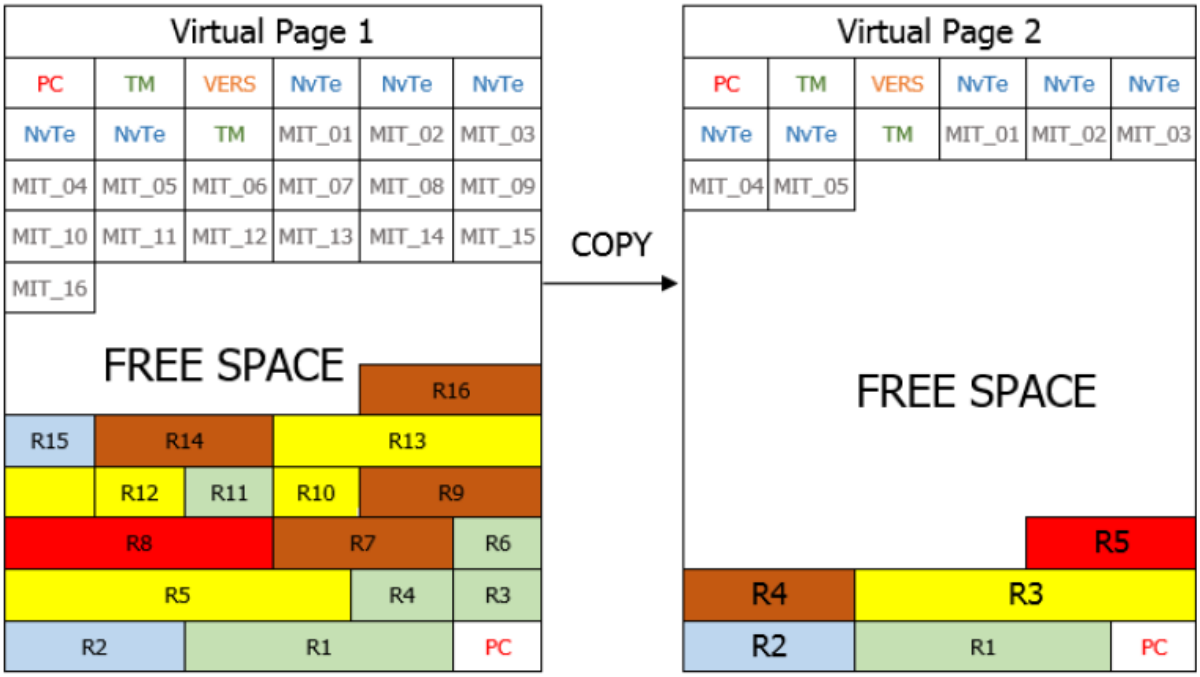
In the example given below, the virtual page 1 is considered to be full if a new save request is pending and the page free space is not sufficient to copy the new record and the additional MIT. In this case, the latest saved datasets (table entries) are copied to virtual page 2.



In this example, there are five datasets (one color for each dataset) with both 'full' and 'single' record types.

- R1 is a 'full' record type (contains all the NV table entry elements), whereas R3, R4, R6 and R11 are 'single' record types.
- R2 – full record type; R15 – single record type
- R5, R13 – full record type; R10, R12 – single record type
- R8 – full record type
- R7, R9, R14, R16 – full record type

As shown above, the R3, R4, R6, and R11 are 'single' record types, while R1 is a 'full' record type of the same dataset. When copied to virtual page 2, a defragmentation process takes place. As a result, the record copied to virtual page 2 has as much elements as R1, but individual elements are taken from R3, R4, R6, and R11. After the copy process completes, the virtual page 2 has five 'full' record types, one for each dataset. |This is illustrated below:



Finally, the virtual page 2 is validated by writing the PC value and a request to erase virtual page 1 is performed. The page is erased on an idle task, sector by sector where only one sector is erased at a time when idle task is executed.

If there is any difference between the RAM and flash tables, the application must call RecoverNvEntry for each entry that is different from its RAM copy to recover the entry data (ID, Type, ElemSz, ElemCnt) from flash before calling NvInit. The application must allocate the pData and change the RAM entry. It can choose to ignore the flash entry if the entry is not desired. If any entry from RAM differs from its flash equivalent at initialization, a page copy is triggered that ignores the entries that are different. In other words, data stored in those entries is lost.

The application can check if the RAM table was updated. In other words, if the MCU program was changed and the RAM table was updated, using the function GetFlashTableVersion and compare the result with the constant gNvFlashTableVersion_c. If the versions are different, NvInit detects the update and automatically upgrades the flash table. The upgrade process triggers a page copy that moves the flash data from the active page to the other one. It keeps the entries that were not modified intact and it moves the entries that had their elements count changed as follows:

- If the RAM element count is smaller than the flash element count, the upgrade only copies as many elements as are in RAM.
- If the RAM element count is larger than the flash element count, the upgrade copies all data from flash and fills the remaining space with data from RAM. If the entry size is changed, the entry is not copied. Any entryIds that are present in flash and not present in RAM are

also not copied. This functionality is not supported if gNvUseExtendedFeatureSet_d is not set to 1.

ECC Fault detection

The KW45/K32W1 internal flash is organized in 16 byte phrases and 8kB sectors (minimal erase unit). Its flash controller is synthesized so that it generates ECC information and an ECC generator / checker. During the programming of internal flash, errors may accidentally happen and cause ECC errors as a flash phrase is being written. These may happen due to multiple reasons:

- programmatic errors such as overwriting an already programmed phrase (transitioning bits from 0b to 1b). These are evitable by performing a blank check verification over phrase to be programmed, at the expense of processing power.
- occurrence of power drop or glitches during a programming operation.
- excessive wear of flash sector. The flash controller is capable of correcting one single ECC error but raises a bud fault whenever reading a phrase containing more than one ECC fault. Once an ECC error has 'infected' a flash phrase, the fault will remain and raise again at each read operation over the same phrase including blank check and prefetch. It can only be rid of by erasing the whole flash sector that contained the faulty phrase. In order to recover from situations where an ECC fault has occurred a gNvSalvageFromEccFault_d option has been added, which forces gNvVerifyReadBackAfterProgram_d to be defined to TRUE. If defined, the gNvVerifyReadBackAfterProgram_d option of the NVM module, causes the program to read back the programmed area after every flash programming operation. The verification is performed in safe mode if gNvSalvageFromEccFault_d is also defined. This is so as to detect ECC faults as early as possible as they appear, indeed when verifying a programming operation, one cannot be certain of the absence of ECC fault and avoid the bus fault. The safe API is thence used to perform the read back operation is performed using this safe API, so that we can tread in the flash and detect potential errors. The defects are detected on the fly whereas in the absence of safe read back, the error would cause a fault, potentially much later. During normal operation, assuming that no chip reset was provoked, this will consist in a single ECC fault either in the last record data or its meta information. Detecting such a fault calls for an immediate page copy to the other virtual page, so that the currently active page gets erased and the error gets cleared. Should the ECC fault occurs in the middle of a page copy operation, the switch of active page is postponed so that the fault page can be erased again and the copy can be restarted.

If the system underwent a power drop during a flash programming operation, sufficient to provoke a reset, at the ensuing reboot, ECC fault(s) may be present in the NVM area at the location that was being written. The detection is performed by an NVM sweeping mechanism, using the safe read API. That marks the faulty virtual page so that all subsequent reads within this virtual page are done with the safe API. If this case arises, a copy of the valid contents of the faulty page is attempted to the other virtual page. At NVM initialization, faults should be detected, either at the top of the meta data or at the bottom of the record area within the previous active page. This should guarantee that only the latest record write operation may be impaired. When the page copy has taken place, the faulty page is erased and the execution may resume. During NvCopyPage, when 'garbage collecting' occurs or whenever the current virtual active page needs to be transferred to the other virtual page, ECC errors are intercepted so that the operation can be attempted again in case of error. In case of NVM contents clobbering by programming errors, the salvage operation does its best to rescue as many records as possible but data will inevitably be lost.

An additional option -namely *gInterceptEccBusFaults_d* - was introduced in order to catch and correct ECC faults at Bus Fault handler level. Indeed, should an ECC bus fault fire, in spite of the precautions taken with NVM's gNvSalvageFromEccFault_d, we verify if the fault belongs to the NV storage. If so, a drastic policy can be adopted consisting in an erasure of the faulty sector. The corresponding Bus Fault handling is not part of the NVM, but dwells in the framework platform specific sources. Alternative handling could be implemented by the customer.

Save policy:

Execution of program and erase operations on a flash an MCU core fetches code from cause perturbations of the core activity or requires to place critical code in RAM so that real-time ISR can still be served. The penalty of a sector erase is much higher than a simple program operation. The NVM is designed so as to limit the erase operations at 'garbage collecting' time, so that flash wear is limited and no time is wasted. Several write policies are implemented to cope with the application constraints, one synchronous mode API and several posted write APIs. Among the posted write policies, the gNvmSaveOnIdleTimerPolicy_d compilation option selects a mode where flash write operations occur at time interval within the Idle task. Another option exists to 'randomize' the time interval with some jitter.

1) NvSyncSave performs a write synchronously with the disadvantage of stalling processor activity until comp
2) NvSaveOnCount posts a pending write operation and postpones the actual flash operation until number of record updates has reached a maximum. The actual write happens during Idle Task execution.see NvSetCountsBetweenSaves related API.
3) NvSaveOnInterval: posts a pending write operation and postpones the actual flash operation until the predefined number of ticks has elapsed. Optional mode - Active if (gNvmSaveOnIdleTimerPolicy_d & gNvmUseSaveOnTimerOn_c). see NvSetMinimumTicksBetweenSaves related API. Note that gNvmUseSaveIntervalJitter_c policy is a sub-option of gNvmSaveOnIdleTimerPolicy_d used to randomize slightly the time at which the write operation will happen.

Constant macro definition

- *gNvStorageIncluded_d* : If set to TRUE, it enables the whole functionality of the nonvolatile storage system. By default, it is set to FALSE (no code or data is generated for this module).
- *gNvUseFlexNVM_d* : If set to TRUE, it enables the FlexNVM functionality of the nonvolatile storage system. By default, it is set to FALSE. If FlexNVM is used, the standard nonvolatile storage system is disabled.
- *gNvFragmentation_Enabled_d* : Macro used to enable/disable the fragmented saves/restores (a particular element from a table entry can be saved or restored). It is set to FALSE by default.
- *gNvUseExtendedFeatureSet_d* : Macro used to enable/disable the extended feature set of the module:
  - Remove existing NV table entries
  - Register new NV table entries
  - Table upgrade

    It is set to FALSE by default.
- *gUnmirroredFeatureSet_d* : Macro used to enable unmirrored datasets. It is set to 0 by default.
- *gNvTableEntriesCountMax_c* : This constant defines the maximum count of the table entries (datasets) that the application is going to use. It is set to 32 by default.
- *gNvRecordsCopiedBufferSize_c* : This constant defines the size of the buffer used by the page copy function, when the copy operation performs defragmentation. The chosen value must be bigger than the maximum number of elements stored in any of the table entries. It is set by default to 64.
- *gNvCacheBufferSize_c* : This constant defines the size of the cache buffer used by the page copy function, when the copy operation does not perform defragmentation. The chosen value must be a multiple of 8. It is set by default to 64.

- *gNvMinimumTicksBetweenSaves_c* : This constant defines the minimum timer ticks between dataset saves (in seconds). It is set to 4 by default.
- *gNvCountsBetweenSaves_c* : This constant defines the number of calls to 'NvSaveOnCount' between dataset saves. It is set to 256 by default.
- *gNvInvalidDataEntry_c* : Macro used to mark a table entry as invalid in the NV table. The default value is 0xFFFFU.
- *gNvFormatRetryCount_c* : Macro used to define the maximum retries count value for the format operation. It is set to 3 by default.
- *gNvPendingSavesQueueSize_c* : Macro used to define the size of the pending saves queue. It is set to 32 by default.
- *gFifoOverwriteEnabled_c* : Macro used to enable overwriting older entries in the pending saves queue (if it is full). If it is FALSE and the queue is full, the module tries to process the oldest save in the queue to free a position. It is set to FALSE by default.
- *gNvMinimumFreeBytesCountStart_c* : Macro used to define the minimum free space at initialization. If the free space is smaller than this value, a page copy is triggered. It is set by default to 128.
- *gNvEndOfTableId_c* : Macro used to define the ID of the end-of-table entry. It is set to 0xFFFEU by default. No valid entry should use this ID.
- *gNvTableMarker_c* : Macro used to define the table marker value. The table marker is used to indicate the start and the end of the flash copy of the NV table. It is set to 0x4254U by default.
- *gNvFlashTableVersion_c* : Macro used to define the flash table version. It is used to determine if the NVM table was updated. It is set to 1 by default. The application should modify this every time the NVM table is updated and the data from NVM is still required.
- *gNvTableKeptInRam_d* : Set gNvTableKeptInRam_d to FALSE, if the NVM table is stored in FLASH memory (default). If the NVM table is stored in RAM memory, set the macro to TRUE.
- *gNvVerifyReadBackAfterProgram_d* : set by default force verification of NVM programming operations. Is forced implicitly when gNvSalvageFromEccFault_d is defined.
- *gNvSalvageFromEccFault_d* : use safe flash API to read from flash, and provide corrective action when ECC fault is met.

OtaSupport: Over-the-Air Programming Support Overview

This module includes APIs for the over-the-air image upgrade process. A Server device receives an image over the serial interface from a PC or other device thorough FSCI commands. If the Server has an image storage, the image is saved locally. If not, the image is requested chunk by chunk: With image storage

- OTA_RegisterToFsci()
- OTA_InitExternalMemory()
- OTA_WriteExternalMemory()
- ...
- OTA_WriteExternalMemory()

Without image storage:

- OTA_RegisterToFsci()
- OTA_QueryImageReq()
- OTA_ImageChunkReq()

- ...
- OTA_ImageChunkReq()

A Client device processes the received image by computing the CRC and filter unused data and stores the received image into a non-volatile storage. After the entire image has been transferred and verified, the Client device informs the Bootloader application that a new image is available, and that the MCU must be reset to start the upgrade process. See the following command sequence:

- OTA_StartImage()
- OTA_PushImageChunk() and OTA_CrcCompute ()
- ...
- OTA_PushImageChunk() and OTA_CrcCompute ()
- OTA_CommitImage()
- OTA_SetNewImageFlag()
- ResetMCU()

SecLib_RNG: Security library and random number generator SecLib Implementation Flavors

Overview

The NXP Connectivity Framework provides multiple SecLib implementations (flavors) to support different hardware capabilities, security requirements, and performance needs. Each flavor offers the same high-level cryptographic APIs but with different underlying implementations optimized for specific use cases.

Available SecLib Flavors

1. Software SecLib (SecLib.c)(RNG.c)

Description: Pure software implementation providing cryptographic operations if no hardware is available.

Platforms: all platforms

Characteristics:

- Platform-independent software implementation
- No hardware acceleration requirements
- Consistent behavior across all platforms
- Higher CPU usage and power consumption
- Suitable for platforms without crypto hardware

Files: SecLib.c, RNG.c

2. EdgeLock SecLib (SecLib_sss.c)(RNG.c)

Description: Hardware-accelerated implementation using NXP's EdgeLock secure subsystem s200.

Platforms: devices with S200: KW45, KW47, MCXW71, MCXW72

Characteristics:

- Hardware-backed security operations
- Secure key storage in hardware
- Hardware random number generation
- Lower CPU usage, better performance
- Enhanced security through hardware isolation
- Platform-specific (requires Secure Subsystem API)

Files: SecLib_sss.c, RNG.c

3. PSA Crypto SecLib (SecLib_psa.c)(RNG_psa.c)

Description: Implementation based on ARM's Platform Security Architecture (PSA) Crypto API standard.

Platforms: MCXW23, (MCXW71 and MCXW72 for evaluation)

Characteristics:

- Industry-standard API
- Hardware acceleration when available
- Secure key management
- Future-proof and portable
- Compliance-ready

Files: SecLib_psa.c, SecLib_psa_config.h RNG_psa.c

Flavor Selection

SecLib flavor is described in 'Kconfig' and selected inside prj.conf or defconfig. Define one of this line to override the default configuration for your board:

```
CONFIG_MCUX_COMPONENT_middleware.wireless.framework.seclib_rng_port.sw=y
CONFIG_MCUX_COMPONENT_middleware.wireless.framework.seclib_rng_port.secure_subsystem=y
CONFIG_MCUX_COMPONENT_middleware.wireless.framework.seclib_rng_port.psa=y
```

The necessary configuration files are automatically added inside of CmakeLists.txt

Random number generator

Overview

The RNG module is part of the framework used for random number generation. It uses hardware RNG peripherals as entropy sources (TRNG, Secure Subsystem, ...) to provide a true random number generator interface. A Pseudo-Random number generator (PRNG) implementation is available. The PRNG may depend of SecLib services (thus requiring a common mutex) to perform HMAC-SHA256, SHA256, AES-CTR, or alternateively a Lehmer Linear Congruential generator. A prerequisite for the PRNG to function with desired randomness is to be seeded using a proper source of entropy. If no hardware acceleration is present, the RNG may fallback to lesser quality ad-hoc source e.g if present SIM_UID registers, the UIDL is used as the initial seed for the random number generator.

Initialization

The RNG module requires an initialization via a call to RNG_Init. The RNG initialization involves a call to RNG_SetSeed.

In the case of a dual core system consisting of a Host core and an NBU core, the Secure Subsystem is owned by the Host core. The Host core then has a direct access to its TRNG embedded in its secure subssystem. On the NBU code side, a request is emitted via RPMSG to the Host to provide a seed. On receipt of this request, the Host sets a 'reseed needed' flag (from the ISR context) If the core running the RNG service owns the TRNG entropy hardware (if any), it can get the seed directly form this hardware synchronously. In the case of an NBU that does not control the devices entropy source, that is owned by the Host, it request a seed from the Host processor via RPMSG exchange. On receipt of this request the Host sets a flag notifying of this request from the RPMSG ISR context.

RNG_ReInit API is to be used at wake up time in the context of LowPower. RNG_DeInit is used for unit tests and coverage purposes but has no useful role in a real application.

Asynchronous Seed Handling Enhancement

The RNG module has been enhanced to use the framework's work queue system instead of polling from the IDLE thread.

Key Changes:

- **Work Queue Integration**: RNG_NotifyReseedNeeded() now submits seed operations to the work queue on Host cores
- **Non-blocking**: Replaces previous IDLE thread polling with asynchronous work queue scheduling
- **ISR-Safe**: Seed requests from ISR context are safely deferred to work queue execution

Benefits:

- Improved system responsiveness by removing IDLE thread dependency
- Better resource management through work queue scheduling

Seed handling

RNG_SetSeed: RNG_SetExternalSeed may be used to inject application entropy to RNG context seed using a supplied array of bytes. RNG_IsReseedNeeded used from task in Host core to check whether seed must be sent to NBU core.

RNG_GetTrueRandomNumber is the API used to generate a Random 32 bit number from a HW source of entropy. It is essential if only to seed the pseudo random number generator.

RNG_GetPseudoRandomData is used to generate arrays of random bytes.

RNG_NotifyReseedNeeded Enhanced to use work queue submission on Host cores instead of requiring IDLE thread polling. Sets reseed flag and schedules asynchronous seed generation.

Security Library

Overview

The framework provides support for cryptography in the security module. It supports both software and hardware encryption. Depending on the device, the hardware encryption uses either the S200, MMCAU, LTC, or CAU3 module instruction set or dedicated AES and SHA hardware blocks.

Software implementation is provided in a library format.

Support for security algorithms

	SW Seclib : SecLib.c	EdgeLock SecLib_sss.c	Seclib_ps	nccl (part of SecLib.c)	Usage example
AES_128	SecLib_aes.c	x	x		
AES_128_ECB		x	x		
AES_128_CBC	x	x			
AES_128_CTR encryption	x	x			
AES_128_OFB Encryption	x				
AES_128_CMAC	x	x	x		BLE connection, ieee 15.4
AES_128_EAX	x				
AES_128_CCM	x	x	x		BLE, ieee 15.4
SHA1	SecLib_sha.c	x			
SHA256	x	x	x		
HMAC_SHA256	x	x			PRNG, Digest for Matter
ECDH_P256 shared secret generation	x (by 15 incremental steps) -> SecLib_ecdh.c	x with MACRO SecLibECDHUseSSS	x	x	BLE pairing,
EC_P256 key pair generation	x	x	x	x	
EC_P256 public key generation from private key				x	Matter (ECDSA)
ECDSA_P256 hash and msg signature generation / verification		only if owner of the key pair		x	Matter
SPAKE2+ P256 arithmetics				x	Matter

SecLib_psa

The framework provides PSA (Platform Security Architecture) Crypto API support through the SecLib_psa.c implementation. This provides a standardized cryptographic interface that can leverage hardware acceleration when available.

Configuration

PSA support is configured through the SecLib_psa_config.h file, which defines:

- Hardware acceleration capabilities via MBEDTLS_PSA_ACCEL_* macros
- Memory management integration with the framework's memory manager
- External RNG integration

Advantages of PSA Crypto API

Standardization and Portability

- **Vendor Independence**: Applications using PSA APIs can be easily ported between different hardware platforms without changing cryptographic code
- **Industry Standard**: PSA Crypto API is an ARM-defined industry standard (PSA Certified), ensuring consistent behavior across different platforms and vendors
- **Future-Proof**: As an evolving standard, PSA provides long-term compatibility and support for emerging cryptographic requirements

Security Benefits

- **Secure Key Management**: PSA provides built-in secure key storage and lifecycle management, reducing the risk of key exposure
- **Hardware Security Integration**: Seamless integration with hardware security modules (HSMs) and secure elements when available
- **Isolation**: Keys and cryptographic operations can be isolated from application code, reducing attack surface
- **Tamper Resistance**: Hardware-backed implementations provide protection against physical attacks

BLE advanced secure mode

New elements in existing structures:

computeDhKeyParam_t::keepInternalBlob - boolean telling if the shared blob is kept in this structure(in .outpoint) after ECDH_P256_ComputeDhKey() or ECDH_P256_ComputeDhKeySeg() call.

New arguments in existing functions:

ECDH_P256_ComputeDhKey keepBlobDhKey - boolean telling ECDH_P256_ComputeDhKey() or ECDH_P256_ComputeDhKeySeg() to keep the shared object after computation for later use (it is required by the SecLib_GenerateBluetoothF5KeysSecure).

New macros:

gSecLibSssUseEncryptedKeys_d - Enable or disable S200 blobs SecLib support. 0 - the Bluetooth Keys are available in plaintext, 1 - the Bluetooth Keys are not available in plaintext, but in secured blobs. Default is disabled.

New functions:

LE Secure connections pairing:

void ECDH_P256_FreeDhKeyDataSecure

This is a function used to free the shared object stored in computeDhKeyParam_t. When user calls ECDH_P256_ComputeDhKeySeg() with keepBlobDhKey set to 1, it should also call **ECDH_P256_FreeDhKeyDataSecure** .

SecLib_GenerateBluetoothF5Keys

This function is extracted from the Bluetooth LE Host Stack implementation. This corresponds to the legacy implementation without key blobs.

SecLib_GenerateBluetoothF5KeysSecure

Similar to **SecLib_GenerateBluetoothF5Keys** this function is modified to work with key blobs, the reason is to not use SSS inside the Bluetooth LE Host Stack.

SecLib_DeriveBluetoothSKD

This is a helper function used by the Bluetooth LE Host Stack in the pairing procedure, when receiving the vendor HCI command specifying that the ESK needs to be provided to LL.

ELKE_BLE_SM_F5_DeriveKeys

This is a private function, helper for **SecLib_GenerateBluetoothF5KeysSecure**. It was provided by the STEC team.

Privacy:

SecLib_ObfuscateKeySecure

This is a function used by the Bluetooth LE Host Stack to obfuscate the IRK before setting it to Bluetooth LE Controller or before saving it to NVM

SecLib_DeobfuscateKeySecure

This is a function used by the Bluetooth LE Host Stack to extract the plaintext IRK key from the saved NVM blob.

SecLib_VerifyBluetoothAh

This function is extracted from the legacy Bluetooth LE Host Stack implementation using plaintext keys.

SecLib_VerifyBluetoothAhSecure

Similar to **SecLib_VerifyBluetoothAh** with modification to work with S200 key blob.

SecLib_GenerateSymmetricKey

This is a function used by the application to generate the local IRK and local CSRK.

SecLib_GenerateBluetoothEIRKBlobSecure

This is a function used by the application to generate the EIRK needed by Bluetooth LE Controller from the IRK blob.

A2B feature

ECDH_P256_ComputeA2BKey

This function is used to compute the EdgeLock to EdgeLock key. pInPeerPublicKey points to the peer public key, pOutE2EKey is the pointer to where the E2E key object will be stored, this will be freed by the application when it is no longer required by calling ECDH_P256_FreeE2EKeyData().

ECDH_P256_FreeE2EKeyData

This function is used to free the key object given as a parameter. It is used by the application to free the E2E key when is no longer needed.

SecLib_ExportA2BBlobSecure

This function is used to import an ELKE blob or plain text symmetric key in s200 and export an E2E key blob. The input type is identified by the keyType parameter.

SecLib_ImportA2BBlobSecure

This function is used to import an E2E key blob in s200 and export an ELKE blob or plain text symmetric key. The output type is identified by the keyType parameter.

LE Secure connections Pairing flow and SecLib usage:

1. Each device needs to generate locally the public+private keypair. This is done using **ECDH_P256_GenerateKeys**.
2. Devices exchange their public keys.
3. Upon receiving the peer device's public key, local device is computing DH key using **ECDH_P256_ComputeDhKey**.
4. Each device sends DHKeyCheck packet
5. Upon receiving DhKeyCheck each device computes LTK blob using **SecLib_GenerateBluetoothF5Keys**
6. After computing the each device sends HCI_LeStartEnc (on initiator), HCI_Le_Provide_Long_Term_Key (on responder)
7. Bluetooth LE Controller sends back SKD report custom event
8. Bluetooth LE Host Stack computes ESKD based on LTK blob using **SecLib_DeriveBluetoothSKD** and sends it to Bluetooth LE Controller
9. Bluetooth LE Controller encrypts the link

IRK flow and SecLib usage:

1. At startup, when gInitializationComplete_c event is received:

- the local IRK is generated using **SecLib_GenerateSymmetricKey**
- the local EIRK is generated using **SecLib_GenerateBluetoothEIRKBlobSecure**
- local CSRK is generated using **SecLib_GenerateSymmetricKey**

2. During legacy pairing when receiving bonding keys, IRK is obfuscated using **SecLib_ObfuscateKeySecure** and stored
3. When app wants to set the OOB keys using Gap_SaveKeys the IRK is obfuscated using **SecLib_ObfuscateKeySecure**
4. When application calls API Gap_VerifyPrivateResolvableAddress IRK is obfuscated using **SecLib_ObfuscateKeySecure** and verified using **SecLib_VerifyBluetoothAhSecure**
5. When a new connection is received in Host with RPA address not resolved by the Bluetooth LE Controller, the Host tries to resolve it by obfuscating it using **SecLib_ObfuscateKeySecure** and verifying it using **SecLib_VerifyBluetoothAhSecure**
6. When adding a peer in Bluetooth LE Controller resolving list, the peer's IRK is obfuscated using **SecLib_ObfuscateKeySecure** before setting it using **HCI_Le_Add_Device_To_Resolving_List**.
7. When an IRK plaintext is requested by the application using Gap_LoadKeys it is obtained using **SecLib_DeobfuscateKeySecure**
8. When legacy pairing completes and LTK needs to be send in the pairing complete event (gConnEvtPairingComplete_c) the **SecLib_DeobfuscateKey** is used to extract the plaintext.

A2B flow and SecLib usage:

1. At startup, when gInitializationComplete_c event is received, the application will call **ECDH_P256_GenerateKeys** to generate the public/private key pair required for the E2E key derivation and send the public key to the peer device.
2. When the public key is received from the peer device, the application will call **ECDH_P256_ComputeA2BKeySecure** to generate the EdgeLock to EdgeLock key.
3. The application will obtain an E2E IRK blob by calling **SecLib_ExportA2BBlobSecure** with key type gSecElkeBlob_c. The obtained blob is sent to the peer anchor. The peer anchor will call **SecLib_ImportA2BBlob** with keyType gSecElkeBlob_c and save the resulting ELKE blob in NVM, for Digital Key both anchors must have the same IRK.
4. After pairing, in order to send the LTK and IRK contained in the bonding data securely, the application will call **SecLib_ExportA2BBlobSecure** with keyType gSecLtkElkeBlob_c for the LTK, and **SecLib_ExportA2BBlobSecure** with keyType gSecPlainText_c for the IRK. The E2E blobs obtained are sent along with the rest of the bonding data to the peer anchor device.
5. After the bonding data is trasfered the E2E key is no longer needed and **ECDH_P256_FreeE2EKeyData** is called with the key object obtained at step 2 when **ECDH_P256_ComputeA2BKeySecure** was called.

Sensors Overview

The Sensors module provides an API to communicate with the ADC. Two values can be obtained by this module :

- Temperature value
- Battery level

The temperature is given in tenths of degrees Celsius and the battery in percentage.

This module is multi-caller, the ADC is protected by a mutex on the resource and by prevententing lowpower (only WFI) during its processing. Platform specific code can be find in fwk_platform_sensors.c/h.

Features

Manual Measurement

Calling “SENSORS_GetTemperature()” or “SENSORS_GetBatteryLevel()” function will give the last measured value that was copied to RAM. If you want to refresh this value you must first call the trigger function (SENSORS_TriggerTemperatureMeasurement() / SENSORS_TriggerBatteryMeasurement()) that configures the ADC to perform temperature or battery measurement, then a call to the refresh function (SENSORS_RefreshTemperatureValue() / SENSORS_RefreshBatteryLevel()) will read the returned value from the ADC and save it in RAM. You can run code between these two functions because the ADC won’t give you the value until it’s fully ready.

> **Important**: When a measurement is ongoing, new measurement requests will be **dropped**. Measurement triggers are only accepted when the measurement state is **IDLE**. This applies to both temperature and battery measurements to prevent ADC resource conflicts.

Periodic Temperature Measurement

The Sensors module supports automatic periodic temperature measurements that can be requested from two sources:

1. HOST Requests

Applications can request periodic temperature measurements using:

```
SENSORS_TriggerPeriodicTemperatureMeasurement(uint32_t temperature_meas_interval_ms);
```

2. NBU Requests (wireless_mcu platforms only)

The NBU (Narrow Band Unit) can request periodic temperature measurements. For more details about NBU temperature requests check NBU Integration Features section bellow.

Periodic Measurement Behavior

Immediate Trigger: Every periodic request (HOST or NBU) triggers an **immediate measurement** before starting the interval timer.

Interval Management:

- **Minimum Interval Selection**: When both HOST and NBU request periodic measurements, the system uses the **shorter interval**
- **Dynamic Updates**: Changing the interval from either source immediately triggers a new measurement and restarts the timer with the new effective interval
- **Independent Control**: HOST and NBU can independently start/stop their periodic requests

One-shot Mode: Setting interval to 0 triggers a single measurement and stops periodic operation for that source.

Usage Examples

```
// HOST requests periodic measurement every 100ms
SENSORS_TriggerPeriodicTemperatureMeasurement(100);

// HOST requests one-shot measurement and stops periodic operation
SENSORS_TriggerPeriodicTemperatureMeasurement(0);
```

Concurrent Requests Behavior

HOST Interval	NBU Interval	Effective Interval	Notes
200ms	100ms	100ms	NBU interval is shorter
100ms	300ms	100ms	HOST interval is shorter
0 (stopped)	150ms	150ms	Only NBU active
200ms	0 (stopped)	200ms	Only HOST active
0 (stopped)	0 (stopped)	No periodic	Both stopped

Measurement State Management

The Sensors module uses a state machine to manage ADC resource access:

States:

- **MEASUREMENT_IDLE**: No measurement in progress, ready to accept new requests
- **TMP_MEASUREMENT_ONGOING**: Temperature measurement in progress
- **BAT_MEASUREMENT_ONGOING**: Battery measurement in progress

Behavior:

- **Request Acceptance**: New measurement requests are only processed when state is MEASUREMENT_IDLE
- **Request Dropping**: If a measurement request is made while another measurement is ongoing, the new request is silently dropped
- **State Transitions**: State changes from IDLE ONGOING when measurement starts, and back to IDLE when measurement completes
- **Recovery Mechanism**: For periodic measurements, if a trigger fails due to ongoing measurement, a 1ms retry timer is started

NBU Integration Features

> **Platform Support**: NBU integration features are only supported on **wireless_mcu** platforms. These features are **not available** on RT, mcxw23, and rw61x platforms.

Periodic Temperature Measurement from NBU

The Sensors module supports automatic periodic temperature measurements triggered by NBU requests. This feature allows the system to automatically measure temperature at regular intervals without manual intervention. This temperature data is used for XTAL32M calibration and Channel Sounding calibration.

How it works:

1. **NBU Request**: The NBU can request periodic temperature measurements by calling the registered callback with a measurement interval
2. **Timer-based Scheduling**: A timer manager is used to schedule periodic measurements based on the requested interval
3. **Workqueue Processing**: Temperature measurement triggers are processed through the system workqueue to avoid blocking operations
4. **Automatic Refresh**: Once a measurement is complete, the next measurement is automatically scheduled

Key Components:

- **Temperature Request Callback**: Sensors_TemperatureReqCb() - Handles NBU requests for periodic measurements
- **Timer Callback**: Sensors_TempMeasTimerCallback() - Triggers measurements on timer expiration
- **Work Handler**: Sensors_PeriodicTempWorkHandler() - Processes temperature measurement requests in workqueue context
- **Ready Callback**: Sensors_TemperatureReadyCb() - Called when temperature measurement is complete

Usage:

- **One-shot measurement**: NBU requests with interval = 0
- **Periodic measurement**: NBU requests with interval > 0 (in milliseconds)
- The periodic measurement continues until a new request changes the interval or stops it

Temperature Filtering and NBU Communication

Modified Moving Average Filter

On wireless_mcu platforms, temperature readings are processed through a Modified Moving Average (MMA) filter to reduce noise and prevent unnecessary NBU notifications. This filter helps to:

- **Limit XTAL32M calibration changes**: Prevents excessive calibration adjustments that could degrade RF performances
- **Optimize power consumption**: Reduces NBU wake-up events by limiting notifications to significant temperature changes
- **Reduce noise**: Smooths out temperature reading fluctuations for more stable system behavior

The XTAL32M calibration process uses a LUT (Look-Up Table) that contains pre-characterized calibration values mapping temperature ranges to optimal XTAL32M frequency adjustments, enabling precise crystal oscillator tuning based on thermal conditions.

The LUT is registered using `PLATFORM_RegisterXtal32MTempCompLut()` and this is currently done in `examples/_common/project_segments/wireless/wireless_mcu/app_common/hardware_init.c` and requires `gBoardUseXtal32MTempComp` to be enabled.

Configuration: Filter size is controlled by `PLATFORM_TEMPERATURE_FILTER_SIZE` (default: 4 samples)

NBU Temperature Threshold

Temperature values are only sent to the NBU when they exceed a configurable threshold:

Threshold is Controlled by `PLATFORM_TEMP_SENT_NBU_THRESHOLD` (default: 10 tenths of degrees Celsius = 1°C)

Temperature Processing Flow:

1. **Raw Measurement**: ADC interrupt captures temperature reading
2. **Sensors Module Informed** Sensors referesh the temperature value
3. **Filtering**: Raw value is processed through MMA filter
4. **Threshold Check**: Filtered value is compared against last sent value ± threshold
5. **NBU Communication**: If threshold is exceeded:
    - XTAL32M calibration is performed (if LUT is available)
    - Temperature value is sent to NBU
    - Last sent value is updated

```
Temperature Request
        ↓
ADC Interrupt - Measurement is ready
        ↓
Raw Temperature Reading
        ↓
Sensors Module Informed - Temperature refresh
        ↓
MMA Filter Processing
        ↓
Threshold Check: |filtered - last_sent| > threshold?
    ↓                ↓
   YES              NO
    ↓
LUT Available?
  ↓      ↓
 YES     NO
```

(continues on next page)

(continued from previous page)

```
  ↓       ↓
XTAL32M     ↓
Calibration ↓
  ↓       ↓

     ↓
Send Temperature to NBU
     ↓
Update Last Value Sent to NBU
```

Platform Configuration Macros:

```
// Temperature filter size (number of samples for averaging)
#define PLATFORM_TEMPERATURE_FILTER_SIZE 4U

// Threshold for sending temperature to NBU (in tenths of degrees Celsius)
#define PLATFORM_TEMP_SENT_NBU_THRESHOLD 10  // 1°C
```

API Usage Examples

Manual Temperature Measurement

```
int32_t temperature1, temperature2;
SENSORS_TriggerTemperatureMeasurement();
...
temperature1 = (SENSORS_RefreshTemperatureValue())/10;
temperature2 = (SENSORS_GetTemperature())/10;
```

In this case temperature1 is equal to temperature2, both are in degree celsius.

Battery Level Measurement

```
uint8_t batteryLevel;
SENSORS_TriggerBatteryMeasurement();
...
batteryLevel = SENSORS_RefreshBatteryLevel();
// batteryLevel is now in percentage (0-100%)
```

Handling Concurrent Requests

```
// First request - will be accepted
SENSORS_TriggerTemperatureMeasurement();

// Second request while first is ongoing - will be dropped silently
SENSORS_TriggerBatteryMeasurement();  // This will be ignored

// Wait for first measurement to complete
int32_t temp = SENSORS_RefreshTemperatureValue();

// Now battery measurement can be triggered
SENSORS_TriggerBatteryMeasurement();  // This will be accepted
```

Low Power Optimization - Wake-up Temperature Measurement

```
// In low power exit callback (interrupts masked, scheduler disabled)
/* Trig temperature measurement on wake-up.
 * If a temperature change is detected,
 * the new value will be sent to the NBU for appropriate XTAL32M trimming.
 * During channel sounding measurements, this temperature is also used for RTT compensation.
 * Since this function runs with interrupts masked and the scheduler disabled,
 * the unsafe variant must be used to avoid mutex acquisition.
 */
SENSORS_TriggerTemperatureMeasurementUnsafe();
```

This approach optimizes power consumption and provides additional benefits:

- **XTAL32M Trimming**: Temperature changes are sent to NBU for crystal oscillator calibration only if they exceed the configured threshold after filtering
- **RTT Compensation**: This temperature data is used by the NBU for RTT compensation during channel sounding measurements
- **Safe Execution**: Uses the unsafe variant to avoid mutex acquisition in interrupt context with scheduler disabled In case of periodic measurement:
- **Power Optimization**: Triggers temperature measurements when the system is already awake, avoiding dedicated timer wake-ups
- **Timer Reset**: Resets the periodic measurement timer to prevent unnecessary wake-ups

  Note: This optimization is implemented by default in wireless platform examples when gAppUseSensors_d is enabled. The temperature measurement is automatically triggered in the low power exit callback (BOARD_ExitLowPowerCb()) in examples/_common/project_segments/wireless/wireless_mcu/board_lp.c.

Constant macro definitions

Name :

```
#define VALUE_NOT_AVAILABLE_8 0xFFu
#define VALUE_NOT_AVAILABLE_32 0xFFFFFFFFu
```

Description :

Defines the error value that can be compared to the value obtain on the ADC.

SFC : Smart Frequency Calibration Overview

The Smart Frequency Calibration module provides operations and calibration for the FRO32K source clock. This module is split between main core and Radio core:

- fwk_rf_sfc.[ch]: RF_SFC module on Radio core that provides Main FRO32K measurement/calibration and state machine in synchornization with Radio domain activities. See details below.
- fwk_sfc.h: SFC module on host core that provides type definition for usage with fwk_platform_ics.[ch] with PLATFORM_FwkSrvSetRfSfcConfig() API and fwk_platform_ble.c for received callback from the NBU core

Host SFC Module

Algorithm parametrization

This module provides ability to configure the RF_SFC module by sending message to Radio core through fwk_platform_ics.c PLATFORM_FwkSrvSetRfSfcConfig():

- Filter size
- Maximum ppm threshold
- Maximum calibration interval
- Number of sample in filter to swicth from convergence to monitor mode

Ppm target

The ppm target is the deviation from the target clock accepted by the algorithm. When the deviation is larger than the ppm target. The algorithm will update the trimming value and reset the filter. The ppm target cannot be more aggressive RF_SFC_MAXIMAL_PPM_TARGET in order to avoid having to update trimming value at each measurement.

Filter size

Filter size must be included between RF_SFC_MINIMAL_FILTER_SIZE and RF_SFC_MAXIMAL_FILTER_SIZE. See *Filtering and Frequency estimation* section for more details on the parameter.

Maximum calibration interval

In monitor mode, new measurement are triggered by lowpower entry/exit. If the NBU core has a lot of radio activity it could never enter lowpower. The maximum calibration interval is here to ensure a measurement is done regularly. When executing idle the SFC module checks when the last measurement has been done, if it has been too long, it reset the filter and forces a new measurement

Trig sample number

The trig sample number is the number of samples needed by the algorithm in its filter to switch from convergence to monitor mode. Having more than one sample in convergence mode allows to confirm the trimming value that we have set.

SFC debug information

On the other way, the RF_SFC from Radio core sends back notifications to SFC module on main core using RX callback PLATFORM_RegisterFroNotificationCallback() from fwk_platform_ics.h and such information:

- last measured frequency
- average ppm from 32768Khz frequency
- last ppm measured from 32768Khz frequency
- FRO trimming value

RF_SFC module

The RF_SFC module provides the functionality to calibrate the FRO32K source clock during Initialization and radio activity.

The RF_SFC is mostly used on XTAL32K less solution when no 32Khz crystal is soldered on the board. It allows to calibrate the FRO32K source clock to the desired frequency to keep Radio time base within the allowed tolerance given by the connectivity standards. However, even on a XTAL32K solution, the RF_SFC is also used during Initialization until the XTAL32K is up and running in the system. The system firstly runs on the FRO32K clock source then switch to the XTAL32K clock source when it is ready with enough accuracy. This allows to save significant boot time as the FRO32K start up (including calibration) is much faster compared to XTAL32K .

This module will handle:

- FRO32K clock frequency measurement against 32Mhz crystal. It schedules appropriately the start of the measurement and gets the result when completed,
- Filter and estimate the 32Khz frequency value and error by averaging from the last measurements,
- FRO32K calibration in order to update the trimming value to reduce the frequency error on the clock.

The targeted frequency offset shall be within 200ppm. The RF_SFC will handle two modes of operation:

- Convergence mode: when frequency estimation is above 200pm,
- Monitor mode: when frequency estimation is below 200pm.

The RF_SFC module works in active and all low power modes on NBU domain, or on host application domain except power down mode. Power down mode on host application domain is not supported with the FRO32K configuration as clock source.

Feature enablement

Enabling the FRO32K is done by calling the PLATFORM_InitFro32K() function during application initialization in hardware_init.c file, in BOARD_InitHardware() function. If FRO32K is not enabled, Oscillator XTAL32K shall be called instead by calling PLATFORM_InitOsc32K() function. The call to PLATFORM_InitFro32K() from BOARD_InitHardware() can be done by setting the Compilation flag gBoardUseFro32k_d to 1 in hardware_init.c or any header files included from this file.

```
#define gBoardUseFro32k_d        1
```

Detailed description

Frequency measurements

When NBU low power is enabled, the frequency measurements are triggered on Low power wake-up by HW signal. The SFC process called from Idle task will check regularly the completion of the frequency measurement. When the measurement is done, it goes to filtering and frequency estimation process. The frequency measurement duration depends on monitor mode or convergence mode: In convergence mode, the frequency measurement duration is 0.5ms while it is 2ms in monitor mode. In monitor mode, the duration value remains less than the minimal radio activity duration so it does not impact the low power consumption in monitoring mode.

Filtering and Frequency estimation

The FRO32KHz frequency measurement values are noisy because of thermal noise on the FRO32K itself. Also, the frequency measurement can introduce some error. In monitoring mode, it is required to filter the measurements by applying an exponential filter. new_estimation = (new_measurement + ((1 « n) -1) * last_estimation) » n

Default value for n is 7 (meaning 128 samples in the averaging window).

Frequency calibration

When the frequency estimation gets higher than the targeted 200ppm target, the RF_SFC updates the trimming value for one positive or negative increment. For this purpose, it requires to:

- wake up the host application domain and keep the domain active,
- update the trim register of the FRO32K , this register is used to trim the capacitance value of the FRO32K,
- re-allow the host application domain to enter low power.

A slight power impact is expected during a calibration update due to host domain wake-up.

Operational modes

When the low power mode is enabled on NBU power domain, RF SFC handles two modes of operation: convergence and monitor modes. However, when low power is disabled on NBU power domain, only convergence mode is supported.

Convergence mode

Convergence mode is used when the estimated FRO32K frequency is above 200ppm or when the filter has been reset. Typically this occurs :

- During Power ON reset or other reset when NBU is switched OFF
- When temperature varies and FRO32K frequency deviates outside 200ppm threshold target
- When no calibration has been done during some time as we discard old values that could influence the algorithm

The convergence mode process typically starts with a FRO32K trim register update, performs a frequency measurement and the FRO32K trim register is updated until the measured frequency gets below 200ppm. These operations are repeated in a loop until the estimated frequency value gets below 200ppm. When below 200ppm during multiple measurements, the RC SFC switches to Monitoring mode. The convergence mode is only a transition mode to monitoring mode. In convergence mode, the NBU power domain does not go to low power. The convergence mode time duration depends on the initial frequency error of the FR032K. Default frequency measurement duration is 0.5ms so 20 measurements (given as example only) will require less than 10 ms to converge.

Monitoring mode

Monitoring mode is used when the estimated FRO32K frequency is below 200ppm. In this mode, the measurement is triggered on NBU domain wake up from low power mode using an internal hardware signal. The exponential filter is applied to compute the frequency estimation. If the frequency estimation value is still within 200ppm, the NBU power domain is allowed to go to low power. If the estimated value gets above the 200ppm threshold, the RF SFC switch back to convergence mode. The trim register is updated by one increment (positive or negative) and because the frequency has been adjusted and changed, the estimated filtered frequency is reset to discard all previous measurements. Going back to convergence mode typically happens during a temperature gradient. If the temperature is constant, it is not expected to have the estimated value to go beyond 200ppm so no calibration should be required.

Initialization and configuration

During initialization, the RF SFC module will block the Radio Software until monitoring mode is reached. This is to prevent the radio from running with an inaccurate time base due to an important 32k clock frequency error.

Initialization and configuration is done by the NBU core. The configuration parameters can set up:

- The 200ppm target threshold. This value shall be 200ppm or higher.
- The filtering number n (see section above), It shall be between 0 and 8. Default is 7 which is similar to an averaging filter of 128 samples. A higher value will be more robust against noise. A lower value will track temperature variation more faster.

In order to prevent the host application domain from going into power down mode (power down mode not supported with FRO32K as clock source), the fwkSrvLowPowerConstraintCallbacks functions structure is registered to the Framework service on host application core from fwk_platform_lowpower.c file, PLATFORM_LowPowerInit() function. The NBU code applies a

low power Deep Sleep constraint to the application core. This constraint is released when the NBU firmware has no activity to do and re-applied when a new activity starts.

Lowpower impact

Power impact during active mode:

In monitoring mode (this should be 99.9% of the time if temperature does not vary), the FRO32KHz frequency measurements are performed during a Radio activity so it does not increase the active current as the sources clocks are already active. Also, it does not increase the active time as the measurement takes less time than an advertising event or connection event so no impact on power consumption.

The main power impact will be in convergence mode. In this case, measurements/calibrations are done in loop until the monitoring mode is reached (frequency error less than 200ppm). This could happen:

- During power ON reset,
- When temperature varies: The frequency will deviate from 32768Hz and FRO32K trimming register correction will need to be updated for that,
- When no measurement has been done during some time as we cannot predict if the FRO has drifted, so we discard older values and start convergence mode.

When FRO32K frequency needs to be adjusted, the NBU core will wake-up the main power domain and will update the FRO32K trimming register.

Power impact during low power mode:

The power consumption in low power mode will increase slightly due to running FRO32K compared to XTAL32K. The power consumption of FRO32K typically consumes 350nA while it is only 100nA with XTAL32K. Refer to the product datasheet for the exact numbers.

Chapter 4

RTOS

4.1 FreeRTOS

4.1.1 FreeRTOS kernel

Open source RTOS kernel for small devices.

FreeRTOS kernel for MCUXpresso SDK Readme

MCUXpresso SDK: FreeRTOS Kernel NXP Integration This repository is the NXP integration layer for the FreeRTOS kernel (https://github.com/FreeRTOS/FreeRTOS-Kernel). It allows original FreeRTOS kernel upstream sources build in MCUXpresso SDK. It is part of the MCUXpresso SDK overall delivery which is composed of several sub-repositories/projects. Navigate to the top/parent repository mcuxsdk-manifests(https://github.com/nxp-mcuxpresso/mcuxsdk-manifests) for the complete delivery of MCUXpresso SDK.

For more information about the FreeRTOS kernel repo adoption see README_mcuxsdk.md: FreeRTOS kernel for MCUXpresso SDK Readme document.

FreeRTOS kernel for MCUXpresso SDK Overview

The purpose of this document is to describes the FreeRTOS kernel repo integration into the NXP MCUXpresso Software Development Kit: mcuxsdk. MCUXpresso SDK provides a comprehensive development solutions designed to optimize, ease, and help accelerate embedded system development of applications based on MCUs from NXP. This project involves the FreeRTOS kernel repo fork with:

- cmake and Kconfig support to allow the configuration and build in MCUXpresso SDK ecosystem
- FreeRTOS OS additions, such as FreeRTOS driver wrappers, RTOS ready FatFs file system, and the implementation of FreeRTOS tickless mode

The history of changes in FreeRTOS kernel repo for MCUXpresso SDK are summarized in *CHANGELOG.md* file.

The MCUXpresso SDK framework also contains a set of FreeRTOS examples which show basic FreeRTOS OS features. This makes it easy to start a new FreeRTOS project or begin experimenting with FreeRTOS OS. Selected drivers and middleware are RTOS ready with related FreeRTOS adaptation layer.

FreeRTOS example applications

The FreeRTOS examples are written to demonstrate basic FreeRTOS features and the interaction between peripheral drivers and the RTOS.

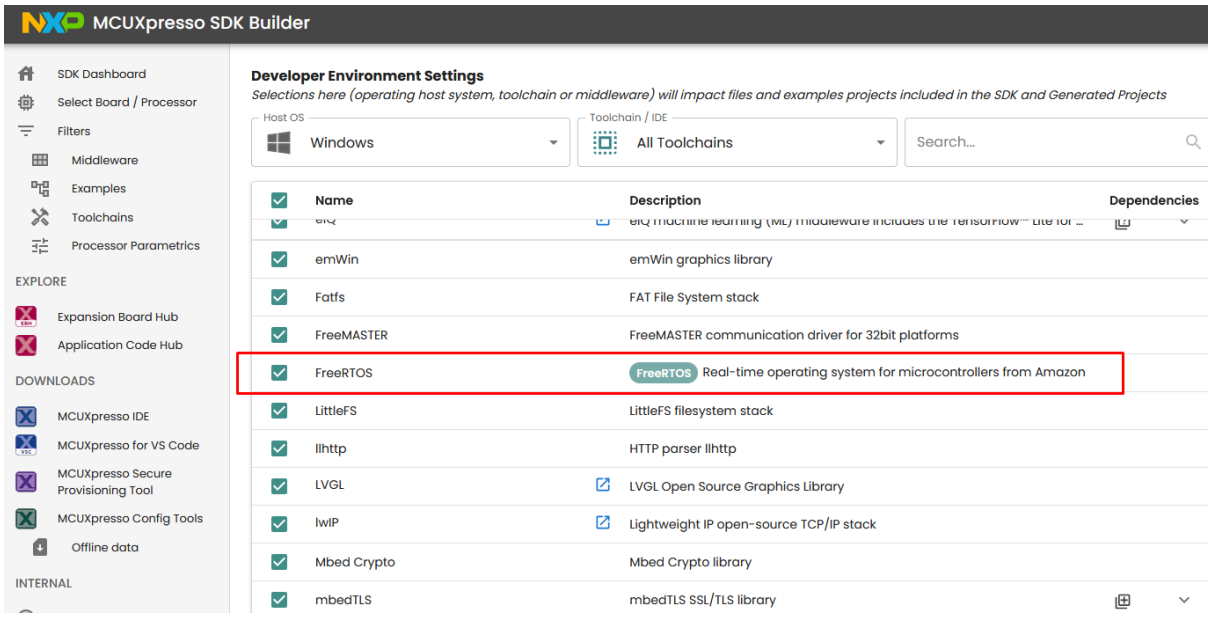
List of examples

The list of freertos_examples, their description and availability for individual supported MCUXpresso SDK development boards can be obtained here: https://mcuxpresso.nxp.com/mcuxsdk/latest/html/examples/freertos_examples/index.html

Location of examples

The FreeRTOS examples are located in mcuxsdk-examples repository, see the freertos_examples folder.

Once using MCUXpresso SDK zip packages created via the MCUXpresso SDK Builder the FreeRTOS kernel library and associated freertos_examples are added into final zip package once FreeRTOS components is selected on the Developer Environment Settings page:



The FreeRTOS examples in MCUXpresso SDK zip packages are located in <MCUXpressoSDK_install_dir>/boards/<board_name>/freertos_examples/ subfolders.

Building a FreeRTOS example application

For information how to use the cmake and Kconfig based build and configuration system and how to build freertos_examples visit: MCUXpresso SDK documentation for Build And Configuration MCUXpresso SDK Getting Start Guide

Tip: To list all FreeRTOS example projects and targets that can be built via the west build command, use this west list_project command in mcuxsdk workspace:

```
west list_project -p examples/freertos_examples
```

FreeRTOS aware debugger plugin

NXP provides FreeRTOS task aware debugger for GDB. The plugin is compatible with Eclipse-based (MCUXpressoIDE) and is available after the installation.

Task List (FreeRTOS)

TCB#	Task Name	Task Handle	Task State	Priority	Stack Usage	Event Object	Runtime
1	task_one	0x1fffecc8	Blocked	1 (1)	0 B / 880 B	MyCountingSemaphore (Rx)	0x0 (0.0%)
2	task_two	0x1ffff130	Blocked	2 (2)	0 B / 888 B	MyCountingSemaphore (Rx)	0x1 (0.1%)
3	IDLE	0x1ffff330	Running	0 (0)	0 B / 296 B		0x3e5 (99.6%)
4	Tmr Svc	0x1ffff6b8	Blocked	17 (17)	28 B / 672 B	TmrQ (Rx)	0x3 (0.3%)

FreeRTOS kernel for MCUXpresso SDK ChangeLog

Changelog FreeRTOS kernel for MCUXpresso SDK All notable changes to this project will be documented in this file.

The format is based on Keep a Changelog, and this project adheres to Semantic Versioning.

[Unreleased]

Fixed

- Temporary workaround for FreeRTOS-Kernel upstream bug #1480: stray URL (http://files.iar.com/ftp/pub/box/bxarm-9.60.3.deb) accidentally placed outside /* */ comment delimiters on line 466 of the IAR CM33/CM55/CM85 TrustZone non-secure portasm.s files (introduced in upstream commit 78e0cc778a, PR #1195). Fixed copies are placed under portable/IAR/ARM_CM{33,55,85}/non_secure/portasm.s and the CMakeLists.txt SOURCES entries for the cm33/cm55/cm85_trustzone.non_secure IAR toolchain blocks are redirected to those copies. Revert the three CMakeLists.txt lines once the upstream fix is merged and the freertos-kernel-upstream revision is bumped past it, then delete portable/IAR/ARM_CM{33,55,85}/ from this repo.

Added

[11.2.0_rev0]

- update amazon freertos version

[11.1.0_rev0]

- update amazon freertos version

[11.0.1_rev0]

- update amazon freertos version

[10.5.1_rev0]

- update amazon freertos version

[10.4.3_rev1]

- Apply CM33 security fix from 10.4.3-LTS-Patch-2. See rtos\freertos\freertos_kernel\History.txt
- Apply CM33 security fix from 10.4.3-LTS-Patch-1. See rtos\freertos\freertos_kernel\History.txt

[10.4.3_rev0]

- update amazon freertos version.

[10.4.3_rev0]

- update amazon freertos version.

[9.0.0_rev3]

- New features:
  - Tickless idle mode support for Cortex-A7. Add fsl_tickless_epit.c and fsl_tickless_generic.h in portable/IAR/ARM_CA9 folder.
  - Enabled float context saving in IAR for Cortex-A7. Added configUSE_TASK_FPU_SUPPORT macros. Modified port.c and portmacro.h in portable/IAR/ARM_CA9 folder.
- Other changes:
  - Transformed ARM_CM core specific tickless low power support into generic form under freertos/Source/portable/low_power_tickless/.

[9.0.0_rev2]

- New features:
  - Enabled MCUXpresso thread aware debugging. Add freertos_tasks_c_additions.h and configINCLUDE_FREERTOS_TASK_C_ADDITIONS_H and configFRTOS_MEMORY_SCHEME macros.

[9.0.0_rev1]

- New features:
  - Enabled -flto optimization in GCC by adding **attribute**((used)) for vTaskSwitchContext.
  - Enabled KDS Task Aware Debugger. Apply FreeRTOS patch to enable configRECORD_STACK_HIGH_ADDRESS macro. Modified files are task.c and FreeRTOS.h.

[9.0.0_rev0]

- New features:
  - Example freertos_sem_static.
  - Static allocation support RTOS driver wrappers.
- Other changes:
  - Tickless idle rework. Support for different timers is in separated files (fsl_tickless_systick.c, fsl_tickless_lptmr.c).
  - Removed configuration option configSYSTICK_USE_LOW_POWER_TIMER. Low power timer is now selected by linking of apropriate file fsl_tickless_lptmr.c.
  - Removed configOVERRIDE_DEFAULT_TICK_CONFIGURATION in RVDS port. Use of **attribute**((weak)) is the preferred solution. Not same as _weak!

[8.2.3]

- New features:
  - Tickless idle mode support.
  - Added template application for Kinetis Expert (KEx) tool (template_application).
- Other changes:
  - Folder structure reduction. Keep only Kinetis related parts.

FreeRTOS kernel Readme

4.1.2 FreeRTOS drivers

This is set of NXP provided FreeRTOS reentrant bus drivers.

4.1.3 backoffalgorithm

Algorithm for calculating exponential backoff with jitter for network retry attempts.

Readme

MCUXpresso SDK: backoffAlgorithm Library NXP Integration This repository is the NXP integration layer for the backoffAlgorithm library (https://github.com/FreeRTOS/backoffalgorithm). It allows original backoffAlgorithm upstream sources build in MCUXpresso SDK. It is part of the MCUXpresso SDK overall delivery which is composed of several sub-repositories/projects. Navigate to the top/parent repository mcuxsdk-manifests(https://github.com/nxp-mcuxpresso/mcuxsdk-manifests) for the complete delivery of MCUXpresso SDK.

4.1.4 corehttp

C language HTTP client library designed for embedded platforms.

MCUXpresso SDK: coreHTTP Client Library NXP Integration

This repository is the NXP integration layer for the coreHTTP Client library (https://github.com/FreeRTOS/corehttp). It allows original coreHTTP Client upstream sources build in MCUXpresso SDK. It is part of the MCUXpresso SDK overall delivery which is composed of several sub-repositories/projects. Navigate to the top/parent repository mcuxsdk-manifests(https://github.com/nxp-mcuxpresso/mcuxsdk-manifests) for the complete delivery of MCUXpresso SDK.

4.1.5 corejson

JSON parser.

Readme

MCUXpresso SDK: coreJSON Library NXP Integration This repository is the NXP integration layer for the coreJSON library (https://github.com/FreeRTOS/corejson). It allows original coreJSON upstream sources build in MCUXpresso SDK. It is part of the MCUXpresso SDK overall delivery which is composed of several sub-repositories/projects. Navigate to the top/parent repository mcuxsdk-manifests(https://github.com/nxp-mcuxpresso/mcuxsdk-manifests) for the complete delivery of MCUXpresso SDK.

4.1.6 coremqtt

MQTT publish/subscribe messaging library.

MCUXpresso SDK: coreMQTT Library NXP Integration

This repository is the NXP integration layer for the coreMQTT library (https://github.com/FreeRTOS/coremqtt). It allows original coreMQTT upstream sources build in MCUXpresso SDK. It is part of the MCUXpresso SDK overall delivery which is composed of several sub-repositories/projects. Navigate to the top/parent repository mcuxsdk-manifests(https://github.com/nxp-mcuxpresso/mcuxsdk-manifests) for the complete delivery of MCUXpresso SDK.

4.1.7 corepkcs11

PKCS #11 key management library.

Readme

MCUXpresso SDK: corePKCS11 Library NXP Integration This repository is the NXP integration layer for the PKCS #11 key management library (https://github.com/FreeRTOS/corePKCS11). It allows original PKCS #11 key management upstream sources build in MCUXpresso SDK. It is part of the MCUXpresso SDK overall delivery which is composed of several sub-repositories/projects. Navigate to the top/parent repository mcuxsdk-manifests(https://github.com/nxp-mcuxpresso/mcuxsdk-manifests) for the complete delivery of MCUXpresso SDK.

4.1.8 freertos-plus-tcp

Open source RTOS FreeRTOS Plus TCP.

Readme

MCUXpresso SDK: FreeRTOS-Plus-TCP Library NXP Integration This repository is the NXP integration layer for the FreeRTOS-Plus-TCP library (https://github.com/FreeRTOS/freertos-plus-tcp). It allows original FreeRTOS-Plus-TCP upstream sources build in MCUXpresso SDK. It is part of the MCUXpresso SDK overall delivery which is composed of several sub-repositories/projects. Navigate to the top/parent repository mcuxsdk-manifests(https://github.com/nxp-mcuxpresso/mcuxsdk-manifests) for the complete delivery of MCUXpresso SDK.

Chapter 5

Downloads

This page is generated automatically by the "downloads_index" Sphinx extension.

4 downloadable file(s) referenced across 3 page(s), grouped into 2 categories:

5.1 boards

- mcuxsdk-mcxw72loc.pdf — from *boards/MCX/mcxw72loc/index*

5.2 develop

- evkbmimxrt1170_custom (.zip) — from develop/sdk/custom_board_development
- frdmmcxa346_custom (.zip) — from develop/sdk/custom_board_development
- example (.zip) — from develop/sdk/integration

Index

Non-alphabetical

__noreturn (*C macro*), 1518

A

addDifferentSecurityModes (*C macro*), 1362
addMode1AndMode2 (*C macro*), 1362
addSameSecurityModes (*C macro*), 1362
attErrorCode_t (*C enum*), 1404
attErrorCode_t.gAttErrCodeAttributeNotFound_c (*C enumerator*), 1405
attErrorCode_t.gAttErrCodeAttributeNotLong_c (*C enumerator*), 1405
attErrorCode_t.gAttErrCodeCccdImproperlyConfigured_c (*C enumerator*), 1406
attErrorCode_t.gAttErrCodeDatabaseOutOfSync_c (*C enumerator*), 1405
attErrorCode_t.gAttErrCodeInsufficientAuthentication_c (*C enumerator*), 1405
attErrorCode_t.gAttErrCodeInsufficientAuthorization_c (*C enumerator*), 1405
attErrorCode_t.gAttErrCodeInsufficientEncryption_c (*C enumerator*), 1405
attErrorCode_t.gAttErrCodeInsufficientEncryptionKeySize_c (*C enumerator*), 1405
attErrorCode_t.gAttErrCodeInsufficientResources_c (*C enumerator*), 1405
attErrorCode_t.gAttErrCodeInvalidAttributeValueLength_c (*C enumerator*), 1405
attErrorCode_t.gAttErrCodeInvalidHandle_c (*C enumerator*), 1404
attErrorCode_t.gAttErrCodeInvalidOffset_c (*C enumerator*), 1405
attErrorCode_t.gAttErrCodeInvalidPdu_c (*C enumerator*), 1405
attErrorCode_t.gAttErrCodeNoError_c (*C enumerator*), 1404
attErrorCode_t.gAttErrCodeOutOfRange_c (*C enumerator*), 1406
attErrorCode_t.gAttErrCodePrepareQueueFull_c (*C enumerator*), 1405
attErrorCode_t.gAttErrCodeProcedureAlreadyInProgress_c (*C enumerator*), 1406
attErrorCode_t.gAttErrCodeReadNotPermitted_c (*C enumerator*), 1404
attErrorCode_t.gAttErrCodeRequestNotSupported_c (*C enumerator*), 1405
attErrorCode_t.gAttErrCodeUnlikelyError_c (*C enumerator*), 1405
attErrorCode_t.gAttErrCodeUnsupportedGroupType_c (*C enumerator*), 1405
attErrorCode_t.gAttErrCodeValueNotAllowed_c (*C enumerator*), 1405
attErrorCode_t.gAttErrCodeWriteNotPermitted_c (*C enumerator*), 1404
attErrorCode_t.gAttErrCodeWriteRequestRejected_c (*C enumerator*), 1406

B

bearerId_t (*C type*), 1499
bearerStatus_t (*C type*), 1499
bearerStatus_tag (*C enum*), 1469
bearerStatus_tag.gEattBearerActive_c (*C enumerator*), 1469
bearerStatus_tag.gEattBearerAllocated_c (*C enumerator*), 1470
bearerStatus_tag.gEattBearerAlreadyAllocated_c (*C enumerator*), 1470
bearerStatus_tag.gEattBearerFree_c (*C enumerator*), 1470
bearerStatus_tag.gEattBearerReconfInProgress_c (*C enumerator*), 1469
bearerStatus_tag.gEattBearerSuspended_c (*C enumerator*), 1470
Ble_CheckMemoryStorage (*C function*), 1504
Ble_CopyDeviceAddress (*C macro*), 1506
Ble_DeviceAddressesMatch (*C macro*), 1505

Ble_HciRecv (*C function*), 1503
Ble_HciRecvFromIsr (*C function*), 1503
Ble_HostDeInitialize (*C function*), 1503
Ble_HostInitialize (*C function*), 1502
Ble_IsPrivateNonresolvableDeviceAddress (*C macro*), 1505
Ble_IsPrivateResolvableDeviceAddress (*C macro*), 1505
Ble_IsRandomStaticDeviceAddress (*C macro*), 1505
bleAddressType_t (*C type*), 1499
bleAdvertiserClockAccuracy_t (*C type*), 1499
bleAdvertiserClockAccuracy_tag (*C enum*), 1473
bleAdvertiserClockAccuracy_tag.gBleAdvertiserClkAcc20ppm_c (*C enumerator*), 1473
bleAdvertiserClockAccuracy_tag.gBleAdvertiserClkAcc30ppm_c (*C enumerator*), 1473
bleAdvertiserClockAccuracy_tag.gBleAdvertiserClkAcc50ppm_c (*C enumerator*), 1473
bleAdvertiserClockAccuracy_tag.gBleAdvertiserClkAcc75ppm_c (*C enumerator*), 1473
bleAdvertiserClockAccuracy_tag.gBleAdvertiserClkAcc100ppm_c (*C enumerator*), 1473
bleAdvertiserClockAccuracy_tag.gBleAdvertiserClkAcc150ppm_c (*C enumerator*), 1473
bleAdvertiserClockAccuracy_tag.gBleAdvertiserClkAcc250ppm_c (*C enumerator*), 1473
bleAdvertiserClockAccuracy_tag.gBleAdvertiserClkAcc500ppm_c (*C enumerator*), 1473
bleAdvertisingChannelMap_t (*C struct*), 1520, 1554
bleAdvertisingChannelMap_t.enableChannel37 (*C var*), 1520, 1554
bleAdvertisingChannelMap_t.enableChannel38 (*C var*), 1520, 1554
bleAdvertisingChannelMap_t.enableChannel39 (*C var*), 1520, 1554
bleAdvertisingChannelMap_t.reserved (*C var*), 1520, 1554
bleAdvertisingFilterPolicy_t (*C enum*), 1471
bleAdvertisingFilterPolicy_t.gBleAdvFilterAllowScanFromAnyAllowConnFromAny_c (*C enumerator*), 1471
bleAdvertisingFilterPolicy_t.gBleAdvFilterAllowScanFromAnyAllowConnFromWL_c (*C enumerator*), 1471
bleAdvertisingFilterPolicy_t.gBleAdvFilterAllowScanFromWLAllowConnFromAny_c (*C enumerator*), 1471
bleAdvertisingFilterPolicy_t.gBleAdvFilterAllowScanFromWLAllowConnFromWL_c (*C enumerator*), 1471
bleAdvertisingReportEventType_t (*C enum*), 1473
bleAdvertisingReportEventType_t.gBleAdvRepAdvDirectInd_c (*C enumerator*), 1473
bleAdvertisingReportEventType_t.gBleAdvRepAdvInd_c (*C enumerator*), 1473
bleAdvertisingReportEventType_t.gBleAdvRepAdvNonconnInd_c (*C enumerator*), 1473
bleAdvertisingReportEventType_t.gBleAdvRepAdvScanInd_c (*C enumerator*), 1473
bleAdvertisingReportEventType_t.gBleAdvRepScanRsp_c (*C enumerator*), 1473
bleAdvertisingType_t (*C enum*), 1470
bleAdvertisingType_t.gAdvConnectableUndirected_c (*C enumerator*), 1470
bleAdvertisingType_t.gAdvDirectedHighDutyCycle_c (*C enumerator*), 1470
bleAdvertisingType_t.gAdvDirectedLowDutyCycle_c (*C enumerator*), 1470
bleAdvertisingType_t.gAdvNonConnectable_c (*C enumerator*), 1470
bleAdvertisingType_t.gAdvScannable_c (*C enumerator*), 1470
bleAdvIndexType_t (*C enum*), 1498
bleAdvIndexType_t.gAdvIndexAscend_c (*C enumerator*), 1498
bleAdvIndexType_t.gAdvIndexDescend_c (*C enumerator*), 1498
bleAdvIndexType_t.gAdvIndexRandom_c (*C enumerator*), 1499
bleAdvIndexType_t.gAdvIndexUser_c (*C enumerator*), 1498
bleAdvReportEventProperties_t (*C type*), 1499
bleAdvReportEventProperties_tag (*C enum*), 1470
bleAdvReportEventProperties_tag.gAdvEventAnonymous_c (*C enumerator*), 1470
bleAdvReportEventProperties_tag.gAdvEventConnectable_c (*C enumerator*), 1470
bleAdvReportEventProperties_tag.gAdvEventDirected_c (*C enumerator*), 1470
bleAdvReportEventProperties_tag.gAdvEventLegacy_c (*C enumerator*), 1470
bleAdvReportEventProperties_tag.gAdvEventScannable_c (*C enumerator*), 1470
bleAdvReportEventProperties_tag.gAdvEventScanResponse_c (*C enumerator*), 1470
bleAdvRequestProperties_t (*C type*), 1499

bleAdvRequestProperties_tag (*C enum*), 1471
bleAdvRequestProperties_tag.gAdvIncludeADIinDecisionPDU_c (*C enumerator*), 1471
bleAdvRequestProperties_tag.gAdvIncludeAdvAinDecisionPDU_c (*C enumerator*), 1471
bleAdvRequestProperties_tag.gAdvIncludeTxPower_c (*C enumerator*), 1471
bleAdvRequestProperties_tag.gAdvReqAnonymous_c (*C enumerator*), 1471
bleAdvRequestProperties_tag.gAdvReqConnectable_c (*C enumerator*), 1471
bleAdvRequestProperties_tag.gAdvReqDirected_c (*C enumerator*), 1471
bleAdvRequestProperties_tag.gAdvReqHighDutyCycle_c (*C enumerator*), 1471
bleAdvRequestProperties_tag.gAdvReqLegacy_c (*C enumerator*), 1471
bleAdvRequestProperties_tag.gAdvReqScannable_c (*C enumerator*), 1471
bleAdvRequestProperties_tag.gAdvUseDecisionPDU_c (*C enumerator*), 1471
bleAntennaInformation_t (*C type*), 1500
bleAntennaInformation_tag (*C struct*), 1523, 1557
bleAntennaInformation_tag.maxCteLength (*C var*), 1557
bleAntennaInformation_tag.maxSwitchingPatternLength (*C var*), 1557
bleAntennaInformation_tag.numAntennae (*C var*), 1557
bleAntennaInformation_tag.supportedSwitchingSamplingRates (*C var*), 1557
bleBondCreatedEvent_t (*C type*), 1500
bleBondCreatedEvent_tag (*C struct*), 1522, 1556
bleBondCreatedEvent_tag.address (*C var*), 1522, 1556
bleBondCreatedEvent_tag.addressType (*C var*), 1522, 1556
bleBondCreatedEvent_tag.nvmIndex (*C var*), 1522, 1556
bleBondDataBlob_t (*C struct*), 1534, 1568
bleBondDataBlob_t.bondDataBlobDeviceInfo (*C var*), 1568
bleBondDataBlob_t.bondDataBlobDynamic (*C var*), 1568
bleBondDataBlob_t.bondDataBlobLegacy (*C var*), 1568
bleBondDataBlob_t.bondDataBlobStatic (*C var*), 1568
bleBondDataBlob_t.bondDataDescriptors (*C var*), 1568
bleBondDataBlob_t.bondHeader (*C var*), 1568
bleBondDataDescriptorBlob_t (*C struct*), 1534, 1567
bleBondDataDescriptorBlob_t.raw (*C var*), 1568
bleBondDataDeviceInfoBlob_t (*C struct*), 1534, 1567
bleBondDataDeviceInfoBlob_t.raw (*C var*), 1567
bleBondDataDynamicBlob_t (*C struct*), 1534, 1567
bleBondDataDynamicBlob_t.raw (*C var*), 1567
bleBondDataLegacyBlob_t (*C struct*), 1534, 1567
bleBondDataLegacyBlob_t.raw (*C var*), 1567
bleBondDataRam_t (*C struct*), 1534, 1568
bleBondDataRam_t.bondDataBlobDeviceInfo (*C var*), 1568
bleBondDataRam_t.bondDataBlobDynamic (*C var*), 1568
bleBondDataRam_t.bondDataBlobLegacy (*C var*), 1568
bleBondDataRam_t.bondDataBlobStatic (*C var*), 1568
bleBondDataRam_t.bondDataDescriptors (*C var*), 1568
bleBondDataRam_t.bondHeader (*C var*), 1568
bleBondDataStaticBlob_t (*C struct*), 1534, 1567
bleBondDataStaticBlob_t.raw (*C var*), 1567
bleBondIdentityHeaderBlob_t (*C struct*), 1534, 1567
bleBondIdentityHeaderBlob_t.raw (*C var*), 1567
bleCentralClockAccuracy_t (*C type*), 1499
bleCentralClockAccuracy_tag (*C enum*), 1472
bleCentralClockAccuracy_tag.gBleCentralClkAcc20ppm_c (*C enumerator*), 1473
bleCentralClockAccuracy_tag.gBleCentralClkAcc30ppm_c (*C enumerator*), 1473
bleCentralClockAccuracy_tag.gBleCentralClkAcc50ppm_c (*C enumerator*), 1472
bleCentralClockAccuracy_tag.gBleCentralClkAcc75ppm_c (*C enumerator*), 1472
bleCentralClockAccuracy_tag.gBleCentralClkAcc100ppm_c (*C enumerator*), 1472
bleCentralClockAccuracy_tag.gBleCentralClkAcc150ppm_c (*C enumerator*), 1472
bleCentralClockAccuracy_tag.gBleCentralClkAcc250ppm_c (*C enumerator*), 1472
bleCentralClockAccuracy_tag.gBleCentralClkAcc500ppm_c (*C enumerator*), 1472

bleChannelFrequency_t (*C enum*), 1474
bleChannelFrequency_t.gBleFreq2402MHz_c (*C enumerator*), 1475
bleChannelFrequency_t.gBleFreq2404MHz_c (*C enumerator*), 1475
bleChannelFrequency_t.gBleFreq2406MHz_c (*C enumerator*), 1475
bleChannelFrequency_t.gBleFreq2408MHz_c (*C enumerator*), 1475
bleChannelFrequency_t.gBleFreq2410MHz_c (*C enumerator*), 1475
bleChannelFrequency_t.gBleFreq2412MHz_c (*C enumerator*), 1475
bleChannelFrequency_t.gBleFreq2414MHz_c (*C enumerator*), 1475
bleChannelFrequency_t.gBleFreq2416MHz_c (*C enumerator*), 1475
bleChannelFrequency_t.gBleFreq2418MHz_c (*C enumerator*), 1475
bleChannelFrequency_t.gBleFreq2420MHz_c (*C enumerator*), 1475
bleChannelFrequency_t.gBleFreq2422MHz_c (*C enumerator*), 1475
bleChannelFrequency_t.gBleFreq2424MHz_c (*C enumerator*), 1475
bleChannelFrequency_t.gBleFreq2426MHz_c (*C enumerator*), 1475
bleChannelFrequency_t.gBleFreq2428MHz_c (*C enumerator*), 1475
bleChannelFrequency_t.gBleFreq2430MHz_c (*C enumerator*), 1475
bleChannelFrequency_t.gBleFreq2432MHz_c (*C enumerator*), 1475
bleChannelFrequency_t.gBleFreq2434MHz_c (*C enumerator*), 1475
bleChannelFrequency_t.gBleFreq2436MHz_c (*C enumerator*), 1475
bleChannelFrequency_t.gBleFreq2438MHz_c (*C enumerator*), 1475
bleChannelFrequency_t.gBleFreq2440MHz_c (*C enumerator*), 1475
bleChannelFrequency_t.gBleFreq2442MHz_c (*C enumerator*), 1475
bleChannelFrequency_t.gBleFreq2444MHz_c (*C enumerator*), 1475
bleChannelFrequency_t.gBleFreq2446MHz_c (*C enumerator*), 1476
bleChannelFrequency_t.gBleFreq2448MHz_c (*C enumerator*), 1476
bleChannelFrequency_t.gBleFreq2450MHz_c (*C enumerator*), 1476
bleChannelFrequency_t.gBleFreq2452MHz_c (*C enumerator*), 1476
bleChannelFrequency_t.gBleFreq2454MHz_c (*C enumerator*), 1476
bleChannelFrequency_t.gBleFreq2456MHz_c (*C enumerator*), 1476
bleChannelFrequency_t.gBleFreq2458MHz_c (*C enumerator*), 1476
bleChannelFrequency_t.gBleFreq2460MHz_c (*C enumerator*), 1476
bleChannelFrequency_t.gBleFreq2462MHz_c (*C enumerator*), 1476
bleChannelFrequency_t.gBleFreq2464MHz_c (*C enumerator*), 1476
bleChannelFrequency_t.gBleFreq2466MHz_c (*C enumerator*), 1476
bleChannelFrequency_t.gBleFreq2468MHz_c (*C enumerator*), 1476
bleChannelFrequency_t.gBleFreq2470MHz_c (*C enumerator*), 1476
bleChannelFrequency_t.gBleFreq2472MHz_c (*C enumerator*), 1476
bleChannelFrequency_t.gBleFreq2474MHz_c (*C enumerator*), 1476
bleChannelFrequency_t.gBleFreq2476MHz_c (*C enumerator*), 1476
bleChannelFrequency_t.gBleFreq2478MHz_c (*C enumerator*), 1476
bleChannelFrequency_t.gBleFreq2480MHz_c (*C enumerator*), 1476
bleChannelMap_t (*C type*), 1499
bleChannelOverrideMode_t (*C enum*), 1474
bleChannelOverrideMode_t.gOverrideAdvChannels_c (*C enumerator*), 1474
bleChannelOverrideMode_t.gOverrideInitChannels_c (*C enumerator*), 1474
bleChannelOverrideMode_t.gOverrideScanChannels_c (*C enumerator*), 1474
bleCteAllowedTypesMap_t (*C struct*), 1535, 1568
bleCteAllowedTypesMap_t.allowAoA (*C var*), 1535, 1569
bleCteAllowedTypesMap_t.allowAoD1us (*C var*), 1535, 1569
bleCteAllowedTypesMap_t.allowAoD2us (*C var*), 1535, 1569
bleCteAllowedTypesMap_t.reserved (*C var*), 1535, 1569
bleCteReqEnable_t (*C enum*), 1495
bleCteReqEnable_t.gCteReqDisable_c (*C enumerator*), 1495
bleCteReqEnable_t.gCteReqEnable_c (*C enumerator*), 1495
bleCteRspEnable_t (*C enum*), 1495
bleCteRspEnable_t.gCteRspDisable_c (*C enumerator*), 1495
bleCteRspEnable_t.gCteRspEnable_c (*C enumerator*), 1495
bleCteTransmitEnable_t (*C enum*), 1495

bleCteTransmitEnable_t.gCteTransmitDisable_c (*C enumerator*), 1495
bleCteTransmitEnable_t.gCteTransmitEnable_c (*C enumerator*), 1495
bleCteType_t (*C type*), 1501
bleCteType_tag (*C enum*), 1495
bleCteType_tag.gCteTypeAoA_c (*C enumerator*), 1495
bleCteType_tag.gCteTypeAoD1us_c (*C enumerator*), 1495
bleCteType_tag.gCteTypeAoD2us_c (*C enumerator*), 1495
bleCteType_tag.gCteTypeNoCte_c (*C enumerator*), 1496
bleDeviceAddress_t (*C type*), 1499
bleGetConnParamsMode_t (*C enum*), 1498
bleGetConnParamsMode_t.gSendEventAfterLLProcUpdate_c (*C enumerator*), 1498
bleGetConnParamsMode_t.gSendEventAfterRemoteSNChange_c (*C enumerator*), 1498
bleHandoverAnchorNotificationEnable_t (*C enum*), 1498
bleHandoverAnchorNotificationEnable_t.gAnchorNotificationDisable_c (*C enumerator*), 1498
bleHandoverAnchorNotificationEnable_t.gAnchorNotificationEnable_c (*C enumerator*), 1498
bleHandoverAnchorSearchMode_t (*C enum*), 1498
bleHandoverAnchorSearchMode_t.gPacketMode_c (*C enumerator*), 1498
bleHandoverAnchorSearchMode_t.gRssiSniffingMode_c (*C enumerator*), 1498
bleHandoverAnchorSearchMode_t.gSuspendTxMode_c (*C enumerator*), 1498
bleHandoverSuspendTransmitMode_t (*C enum*), 1497
bleHandoverSuspendTransmitMode_t.gDoNotUseEventCounter_c (*C enumerator*), 1497
bleHandoverSuspendTransmitMode_t.gSafeStopHostLlTx_c (*C enumerator*), 1497
bleHandoverSuspendTransmitMode_t.gSafeStopLlTx_c (*C enumerator*), 1497
bleHandoverSuspendTransmitMode_t.gUseEventCounter_c (*C enumerator*), 1497
bleHandoverTimeSyncEnable_t (*C enum*), 1498
bleHandoverTimeSyncEnable_t.gTimeSyncDisable_c (*C enumerator*), 1498
bleHandoverTimeSyncEnable_t.gTimeSyncEnable_c (*C enumerator*), 1498
bleHandoverTimeSyncStopWhenFound_t (*C enum*), 1498
bleHandoverTimeSyncStopWhenFound_t.gTimeSyncDoNotStopWhenFound_c (*C enumerator*), 1498
bleHandoverTimeSyncStopWhenFound_t.gTimeSyncStopWhenFound_c (*C enumerator*), 1498
bleHardwareErrorCode_t (*C enum*), 1477
bleHardwareErrorCode_t.bleHwErrCodeNoError_c (*C enumerator*), 1477
bleIdentityAddress_t (*C struct*), 1519, 1554
bleIdentityAddress_t.idAddress (*C var*), 1520, 1554
bleIdentityAddress_t.idAddressType (*C var*), 1520, 1554
bleInitiatorFilterPolicy_t (*C type*), 1499
bleIqReportPacketStatus_t (*C type*), 1502
bleIqReportPacketStatus_tag (*C enum*), 1496
bleIqReportPacketStatus_tag.gIqReportPacketStatusCorrectCrc_c (*C enumerator*), 1496
bleIqReportPacketStatus_tag.gIqReportPacketStatusCrcIncorrectUsedLength_c (*C enumerator*), 1496
bleIqReportPacketStatus_tag.gIqReportPacketStatusCrcIncorrectUsedOther_c (*C enumerator*), 1496
bleIqReportPacketStatus_tag.gIqReportPacketStatusInsufficientResources_c (*C enumerator*), 1496
bleIqSamplingEnable_t (*C enum*), 1495
bleIqSamplingEnable_t.gIqSamplingDisable_c (*C enumerator*), 1495
bleIqSamplingEnable_t.gIqSamplingEnable_c (*C enumerator*), 1495
bleLlConnectionRole_t (*C enum*), 1472
bleLlConnectionRole_t.gBleLlConnectionCentral_c (*C enumerator*), 1472
bleLlConnectionRole_t.gBleLlConnectionPeripheral_c (*C enumerator*), 1472
bleLocalKeysBlob_t (*C struct*), 1534, 1568
bleLocalKeysBlob_t.___unnamed2___ (*C union*), 1539, 1572
bleLocalKeysBlob_t.___unnamed2___.pKey (*C var*), 1540, 1572
bleLocalKeysBlob_t.___unnamed2___.raw (*C var*), 1540, 1572
bleLocalKeysBlob_t.keyGenerated (*C var*), 1568
bleMonAdvCondition_t (*C enum*), 1293
bleMonAdvCondition_t.gBleMonAdvConditionRssiHighThreshold_c (*C enumerator*), 1293
bleMonAdvCondition_t.gBleMonAdvConditionRssiLowThreshold_c (*C enumerator*), 1293
bleNotificationEvent_t (*C struct*), 1521, 1555
bleNotificationEvent_t.adv_handle (*C var*), 1522, 1556

bleNotificationEvent_t.ce_counter (*C var*), 1521, 1556
bleNotificationEvent_t.channel (*C var*), 1521, 1556
bleNotificationEvent_t.deviceId (*C var*), 1521, 1555
bleNotificationEvent_t.eventType (*C var*), 1521, 1555
bleNotificationEvent_t.rssi (*C var*), 1521, 1556
bleNotificationEvent_t.scanned_addr (*C var*), 1522, 1556
bleNotificationEvent_t.status (*C var*), 1521, 1556
bleNotificationEvent_t.timestamp (*C var*), 1522, 1556
bleNotificationEventType_t (*C type*), 1500
bleNotificationEventType_tag (*C enum*), 1494
bleNotificationEventType_tag.gNotifAdvConnReqRx_c (*C enumerator*), 1494
bleNotificationEventType_tag.gNotifAdvEventOver_c (*C enumerator*), 1494
bleNotificationEventType_tag.gNotifAdvScanReqRx_c (*C enumerator*), 1494
bleNotificationEventType_tag.gNotifAdvTx_c (*C enumerator*), 1494
bleNotificationEventType_tag.gNotifChannelMatrix_c (*C enumerator*), 1494
bleNotificationEventType_tag.gNotifConnChannelMapUpdate_c (*C enumerator*), 1495
bleNotificationEventType_tag.gNotifConnCreated_c (*C enumerator*), 1494
bleNotificationEventType_tag.gNotifConnEventOver_c (*C enumerator*), 1494
bleNotificationEventType_tag.gNotifConnInd_c (*C enumerator*), 1495
bleNotificationEventType_tag.gNotifConnRxPdu_c (*C enumerator*), 1494
bleNotificationEventType_tag.gNotifEventNone_c (*C enumerator*), 1494
bleNotificationEventType_tag.gNotifPhyReq_c (*C enumerator*), 1494
bleNotificationEventType_tag.gNotifPhyUpdateInd_c (*C enumerator*), 1495
bleNotificationEventType_tag.gNotifScanAdvPktRx_c (*C enumerator*), 1494
bleNotificationEventType_tag.gNotifScanEventOver_c (*C enumerator*), 1494
bleNotificationEventType_tag.gNotifScanReqTx_c (*C enumerator*), 1494
bleNotificationEventType_tag.gNotifScanRspRx_c (*C enumerator*), 1494
blePathLossReportingEnable_t (*C enum*), 1497
blePathLossReportingEnable_t.gPathLossReportingDisable_c (*C enumerator*), 1497
blePathLossReportingEnable_t.gPathLossReportingEnable_c (*C enumerator*), 1497
blePathLossThresholdZoneEntered_t (*C type*), 1502
blePathLossThresholdZoneEntered_tag (*C enum*), 1496
blePathLossThresholdZoneEntered_tag.gPathLossThresholdHighZone_c (*C enumerator*), 1496
blePathLossThresholdZoneEntered_tag.gPathLossThresholdLowZone_c (*C enumerator*), 1496
blePathLossThresholdZoneEntered_tag.gPathLossThresholdMiddleZone_c (*C enumerator*), 1496
blePowerControlPhyType_t (*C type*), 1502
blePowerControlPhyType_tag (*C enum*), 1497
blePowerControlPhyType_tag.gPowerControlLePhy1M_c (*C enumerator*), 1497
blePowerControlPhyType_tag.gPowerControlLePhy2M_c (*C enumerator*), 1497
blePowerControlPhyType_tag.gPowerControlLePhyCodedS2_c (*C enumerator*), 1497
blePowerControlPhyType_tag.gPowerControlLePhyCodedS8_c (*C enumerator*), 1497
blePrivacyMode_t (*C type*), 1499
bleResult_t (*C type*), 1499
bleResult_tag (*C enum*), 1456
bleResult_tag.gAttStatusBase_c (*C enumerator*), 1467
bleResult_tag.gAttSuccess_c (*C enumerator*), 1467
bleResult_tag.gBleAlreadyInitialized_c (*C enumerator*), 1456
bleResult_tag.gBleFeatureNotSupported_c (*C enumerator*), 1456
bleResult_tag.gBleInvalidParameter_c (*C enumerator*), 1456
bleResult_tag.gBleInvalidState_c (*C enumerator*), 1456
bleResult_tag.gBleNVMError_c (*C enumerator*), 1457
bleResult_tag.gBleOsError_c (*C enumerator*), 1456
bleResult_tag.gBleOutOfMemory_c (*C enumerator*), 1456
bleResult_tag.gBleOverflow_c (*C enumerator*), 1456
bleResult_tag.gBleReassemblyInProgress_c (*C enumerator*), 1457
bleResult_tag.gBleRngError_c (*C enumerator*), 1457
bleResult_tag.gBleSecLibError_c (*C enumerator*), 1457
bleResult_tag.gBleStatusBase_c (*C enumerator*), 1456

bleResult_tag.gBleSuccess_c (*C enumerator*), 1456
bleResult_tag.gBleTimerError_c (*C enumerator*), 1457
bleResult_tag.gBleUnavailable_c (*C enumerator*), 1456
bleResult_tag.gBleUnexpectedError_c (*C enumerator*), 1456
bleResult_tag.gCsCallbackAlreadyInstalled_c (*C enumerator*), 1469
bleResult_tag.gCsSecurityCheckFail_c (*C enumerator*), 1469
bleResult_tag.gCsStatusBase_c (*C enumerator*), 1469
bleResult_tag.gCsSuccess_c (*C enumerator*), 1469
bleResult_tag.gCtrlStatusBase_c (*C enumerator*), 1462
bleResult_tag.gCtrlSuccess_c (*C enumerator*), 1462
bleResult_tag.gDevDbCccdLimitReached_c (*C enumerator*), 1468
bleResult_tag.gDevDbCccdNotFound_c (*C enumerator*), 1469
bleResult_tag.gDevDbStatusBase_c (*C enumerator*), 1468
bleResult_tag.gDevDbSuccess_c (*C enumerator*), 1468
bleResult_tag.gGapAdvDataTooLong_c (*C enumerator*), 1468
bleResult_tag.gGapAnotherProcedureInProgress_c (*C enumerator*), 1468
bleResult_tag.gGapDeviceNotBonded_c (*C enumerator*), 1468
bleResult_tag.gGapScanRspDataTooLong_c (*C enumerator*), 1468
bleResult_tag.gGapStatusBase_c (*C enumerator*), 1468
bleResult_tag.gGapSuccess_c (*C enumerator*), 1468
bleResult_tag.gGattAnotherProcedureInProgress_c (*C enumerator*), 1467
bleResult_tag.gGattClientConfirmationTimeout_c (*C enumerator*), 1468
bleResult_tag.gGattConnectionSecurityRequirementsNotMet_c (*C enumerator*), 1468
bleResult_tag.gGattDbCccdNotFound_c (*C enumerator*), 1469
bleResult_tag.gGattDbCharacteristicNotFound_c (*C enumerator*), 1469
bleResult_tag.gGattDbDescriptorNotFound_c (*C enumerator*), 1469
bleResult_tag.gGattDbInvalidHandle_c (*C enumerator*), 1469
bleResult_tag.gGattDbServiceNotFound_c (*C enumerator*), 1469
bleResult_tag.gGattDbServiceOrCharAlreadyDeclared_c (*C enumerator*), 1469
bleResult_tag.gGattDbStatusBase_c (*C enumerator*), 1469
bleResult_tag.gGattDbSuccess_c (*C enumerator*), 1469
bleResult_tag.gGattIndicationAlreadyInProgress_c (*C enumerator*), 1468
bleResult_tag.gGattInvalidPduReceived_c (*C enumerator*), 1468
bleResult_tag.gGattInvalidValueLength_c (*C enumerator*), 1467
bleResult_tag.gGattLongAttributePacketsCorrupted_c (*C enumerator*), 1467
bleResult_tag.gGattMtuExchangeInProgress_c (*C enumerator*), 1468
bleResult_tag.gGattMultipleAttributesOverflow_c (*C enumerator*), 1467
bleResult_tag.gGattOutOfSyncProceduresOngoing_c (*C enumerator*), 1468
bleResult_tag.gGattPeerDisconnected_c (*C enumerator*), 1468
bleResult_tag.gGattServerTimeout_c (*C enumerator*), 1467
bleResult_tag.gGattStatusBase_c (*C enumerator*), 1467
bleResult_tag.gGattSuccess_c (*C enumerator*), 1467
bleResult_tag.gGattUnexpectedReadMultipleResponseLength_c (*C enumerator*), 1467
bleResult_tag.gHciAclConnectionAlreadyExists_c (*C enumerator*), 1458
bleResult_tag.gHciAlreadyInit_c (*C enumerator*), 1462
bleResult_tag.gHciAuthenticationFailure_c (*C enumerator*), 1457
bleResult_tag.gHciCallbackAlreadyInstalled_c (*C enumerator*), 1462
bleResult_tag.gHciCallbackNotInstalled_c (*C enumerator*), 1462
bleResult_tag.gHciChannelClassificationNotSupported_c (*C enumerator*), 1460
bleResult_tag.gHciCoarseClockAdjustmentRejected_c (*C enumerator*), 1461
bleResult_tag.gHciCommandDisallowed_c (*C enumerator*), 1458
bleResult_tag.gHciCommandNotSupported_c (*C enumerator*), 1462
bleResult_tag.gHciConnectionAcceptTimeoutExceeded_c (*C enumerator*), 1458
bleResult_tag.gHciConnectionFailedToBeEstablishedOrSyncTimeout_c (*C enumerator*), 1461
bleResult_tag.gHciConnectionLimitExceeded_c (*C enumerator*), 1457
bleResult_tag.gHciConnectionRejectedDueToLimitedResources_c (*C enumerator*), 1458
bleResult_tag.gHciConnectionRejectedDueToNoSuitableChannelFound_c (*C enumerator*), 1461
bleResult_tag.gHciConnectionRejectedDueToSecurityReasons_c (*C enumerator*), 1458

bleResult_tag.gHciConnectionRejectedDueToUnacceptableBdAddr_c (*C enumerator*), 1458
bleResult_tag.gHciConnectionTerminatedByLocalHost_c (*C enumerator*), 1458
bleResult_tag.gHciConnectionTerminatedDueToMicFailure_c (*C enumerator*), 1461
bleResult_tag.gHciConnectionTimeout_c (*C enumerator*), 1457
bleResult_tag.gHciControllerBusy_c (*C enumerator*), 1461
bleResult_tag.gHciDifferentTransactionCollision_c (*C enumerator*), 1460
bleResult_tag.gHciDirectedAdvertisingTimeout_c (*C enumerator*), 1461
bleResult_tag.gHciEncryptionModeNotAcceptable_c (*C enumerator*), 1459
bleResult_tag.gHciEventNotSupported_c (*C enumerator*), 1462
bleResult_tag.gHciExtendedInquiryResponseTooLarge_c (*C enumerator*), 1460
bleResult_tag.gHciHardwareFailure_c (*C enumerator*), 1457
bleResult_tag.gHciHostBusyPairing_c (*C enumerator*), 1461
bleResult_tag.gHciInstantPassed_c (*C enumerator*), 1460
bleResult_tag.gHciInsufficientSecurity_c (*C enumerator*), 1460
bleResult_tag.gHciInvalidHciCommandParameters_c (*C enumerator*), 1458
bleResult_tag.gHciInvalidLpmParameters_c (*C enumerator*), 1459
bleResult_tag.gHciInvalidParameter_c (*C enumerator*), 1462
bleResult_tag.gHciLimitReached_c (*C enumerator*), 1461
bleResult_tag.gHciLinkKeyCannotBeChanged_c (*C enumerator*), 1459
bleResult_tag.gHciLLResponseTimeout_c (*C enumerator*), 1459
bleResult_tag.gHciLmpErrorTransactionCollision_c (*C enumerator*), 1459
bleResult_tag.gHciLmpPduNotAllowed_c (*C enumerator*), 1459
bleResult_tag.gHciMacConnectionFailed_c (*C enumerator*), 1461
bleResult_tag.gHciMemoryCapacityExceeded_c (*C enumerator*), 1457
bleResult_tag.gHciOperationCancelledByHost_c (*C enumerator*), 1461
bleResult_tag.gHciPacketTooEarly_c (*C enumerator*), 1462
bleResult_tag.gHciPacketTooLate_c (*C enumerator*), 1462
bleResult_tag.gHciPacketTooLong_c (*C enumerator*), 1461
bleResult_tag.gHciPageTimeout_c (*C enumerator*), 1457
bleResult_tag.gHciPairingNotAllowed_c (*C enumerator*), 1458
bleResult_tag.gHciPairingWithUnitKeyNotSupported_c (*C enumerator*), 1460
bleResult_tag.gHciParameterOutOfMandatoryRange_c (*C enumerator*), 1460
bleResult_tag.gHciPinOrKeyMissing_c (*C enumerator*), 1457
bleResult_tag.gHciQosNotAcceptableParameter_c (*C enumerator*), 1460
bleResult_tag.gHciQosRejected_c (*C enumerator*), 1460
bleResult_tag.gHciRemoteDeviceTerminatedConnectionLowResources_c (*C enumerator*), 1458
bleResult_tag.gHciRemoteDeviceTerminatedConnectionPowerOff_c (*C enumerator*), 1458
bleResult_tag.gHciRemoteUserTerminatedConnection_c (*C enumerator*), 1458
bleResult_tag.gHciRepeatedAttempts_c (*C enumerator*), 1458
bleResult_tag.gHciRequestedQosNotSupported_c (*C enumerator*), 1459
bleResult_tag.gHciReserved_0x2B_c (*C enumerator*), 1460
bleResult_tag.gHciReserved_0x31_c (*C enumerator*), 1460
bleResult_tag.gHciReserved_0x33_c (*C enumerator*), 1460
bleResult_tag.gHciReservedSlotViolation_c (*C enumerator*), 1460
bleResult_tag.gHciRoleChangeNotAllowed_c (*C enumerator*), 1459
bleResult_tag.gHciRoleSwitchFailed_c (*C enumerator*), 1460
bleResult_tag.gHciRoleSwitchPending_c (*C enumerator*), 1460
bleResult_tag.gHciScoAirModeRejected_c (*C enumerator*), 1459
bleResult_tag.gHciScoIntervalRejected_c (*C enumerator*), 1459
bleResult_tag.gHciScoOffsetRejected_c (*C enumerator*), 1459
bleResult_tag.gHciSecureSimplePairingNotSupportedByHost_c (*C enumerator*), 1460
bleResult_tag.gHciStatusBase_c (*C enumerator*), 1457
bleResult_tag.gHciSuccess_c (*C enumerator*), 1457
bleResult_tag.gHciSynchronousConnectionLimitToADeviceExceeded_c (*C enumerator*), 1458
bleResult_tag.gHciTransportError_c (*C enumerator*), 1462
bleResult_tag.gHciType0SubmapNotDefined_c (*C enumerator*), 1461
bleResult_tag.gHciUnacceptableConnectionParameters_c (*C enumerator*), 1461
bleResult_tag.gHciUnknownAdvertisingIdentifier_c (*C enumerator*), 1461

bleResult_tag.gHciUnknownConnectionIdentifier_c (*C enumerator*), 1457
bleResult_tag.gHciUnknownHciCommand_c (*C enumerator*), 1457
bleResult_tag.gHciUnknownLpmPdu_c (*C enumerator*), 1459
bleResult_tag.gHciUnspecifiedError_c (*C enumerator*), 1459
bleResult_tag.gHciUnsupportedFeatureOrParameterValue_c (*C enumerator*), 1458
bleResult_tag.gHciUnsupportedLpmParameterValue_c (*C enumerator*), 1459
bleResult_tag.gHciUnsupportedRemoteFeature_c (*C enumerator*), 1459
bleResult_tag.gL2caAlreadyInit_c (*C enumerator*), 1462
bleResult_tag.gL2caCallbackAlreadyInstalled_c (*C enumerator*), 1463
bleResult_tag.gL2caCallbackNotInstalled_c (*C enumerator*), 1462
bleResult_tag.gL2caChannelAlreadyConnected_c (*C enumerator*), 1463
bleResult_tag.gL2caChannelBusy_c (*C enumerator*), 1463
bleResult_tag.gL2caChannelClosed_c (*C enumerator*), 1463
bleResult_tag.gL2caChannelInvalid_c (*C enumerator*), 1463
bleResult_tag.gL2caConnectionParametersRejected_c (*C enumerator*), 1463
bleResult_tag.gL2caInsufficientResources_c (*C enumerator*), 1462
bleResult_tag.gL2caInternalError_c (*C enumerator*), 1463
bleResult_tag.gL2caInvalidParameter_c (*C enumerator*), 1463
bleResult_tag.gL2caLePsmAlreadyRegistered_c (*C enumerator*), 1463
bleResult_tag.gL2caLePsmInsufficientResources_c (*C enumerator*), 1463
bleResult_tag.gL2caLePsmInvalid_c (*C enumerator*), 1463
bleResult_tag.gL2caLePsmNotRegistered_c (*C enumerator*), 1463
bleResult_tag.gL2caStatusBase_c (*C enumerator*), 1462
bleResult_tag.gL2caSuccess_c (*C enumerator*), 1462
bleResult_tag.gSmCommandNotSupported_c (*C enumerator*), 1463
bleResult_tag.gSmInproperKeyDistributionField_c (*C enumerator*), 1464
bleResult_tag.gSmInsufficientResources_c (*C enumerator*), 1465
bleResult_tag.gSmInvalidCommandCode_c (*C enumerator*), 1464
bleResult_tag.gSmInvalidCommandLength_c (*C enumerator*), 1464
bleResult_tag.gSmInvalidCommandParameter_c (*C enumerator*), 1464
bleResult_tag.gSmInvalidConnectionHandle_c (*C enumerator*), 1464
bleResult_tag.gSmInvalidDeviceId_c (*C enumerator*), 1464
bleResult_tag.gSmInvalidHciEventParameter_c (*C enumerator*), 1465
bleResult_tag.gSmInvalidInternalOperation_c (*C enumerator*), 1464
bleResult_tag.gSmInvalidPeerPublicKey_c (*C enumerator*), 1466
bleResult_tag.gSmInvalidSmpPacketLength_c (*C enumerator*), 1464
bleResult_tag.gSmInvalidSmpPacketParameter_c (*C enumerator*), 1465
bleResult_tag.gSmKeySessionKeyDerivationFailed_c (*C enumerator*), 1467
bleResult_tag.gSmLlConnectionEncryptionFailure_c (*C enumerator*), 1465
bleResult_tag.gSmLlConnectionEncryptionInProgress_c (*C enumerator*), 1465
bleResult_tag.gSmNullCBFunction_c (*C enumerator*), 1463
bleResult_tag.gSmOobDataAddressMismatch_c (*C enumerator*), 1465
bleResult_tag.gSmPairingAlreadyStarted_c (*C enumerator*), 1466
bleResult_tag.gSmPairingErrorAuthenticationRequirements_c (*C enumerator*), 1466
bleResult_tag.gSmPairingErrorBusy_c (*C enumerator*), 1467
bleResult_tag.gSmPairingErrorCommandNotSupported_c (*C enumerator*), 1466
bleResult_tag.gSmPairingErrorConfirmValueFailed_c (*C enumerator*), 1466
bleResult_tag.gSmPairingErrorDhKeyCheckFailed_c (*C enumerator*), 1466
bleResult_tag.gSmPairingErrorEncryptionKeySize_c (*C enumerator*), 1466
bleResult_tag.gSmPairingErrorInvalidParameters_c (*C enumerator*), 1466
bleResult_tag.gSmPairingErrorKeyRejected_c (*C enumerator*), 1466
bleResult_tag.gSmPairingErrorNumericComparisonFailed_c (*C enumerator*), 1466
bleResult_tag.gSmPairingErrorOobNotAvailable_c (*C enumerator*), 1467
bleResult_tag.gSmPairingErrorPairingNotSupported_c (*C enumerator*), 1466
bleResult_tag.gSmPairingErrorPasskeyEntryFailed_c (*C enumerator*), 1466
bleResult_tag.gSmPairingErrorRepeatedAttempts_c (*C enumerator*), 1467
bleResult_tag.gSmPairingErrorTimeout_c (*C enumerator*), 1466
bleResult_tag.gSmPairingErrorUnknownReason_c (*C enumerator*), 1466

bleResult_tag.gSmReceivedHciEventFromUnknownDevice_c (*C enumerator*), 1465
bleResult_tag.gSmReceivedSmpPacketFromUnknownDevice_c (*C enumerator*), 1465
bleResult_tag.gSmReceivedTimerEventForUnknownDevice_c (*C enumerator*), 1465
bleResult_tag.gSmReceivedUnexpectedHciEvent_c (*C enumerator*), 1465
bleResult_tag.gSmReceivedUnexpectedSmpPacket_c (*C enumerator*), 1465
bleResult_tag.gSmSmpPacketReceivedAfterTimeoutOccurred_c (*C enumerator*), 1465
bleResult_tag.gSmSmpTimeoutOccurred_c (*C enumerator*), 1464
bleResult_tag.gSmStatusBase_c (*C enumerator*), 1463
bleResult_tag.gSmSuccess_c (*C enumerator*), 1463
bleResult_tag.gSmTbResolvableAddressDoesNotMatchIrk_c (*C enumerator*), 1467
bleResult_tag.gSmUnattainableLocalDeviceMinKeySize_c (*C enumerator*), 1465
bleResult_tag.gSmUnattainableLocalDeviceSecRequirements_c (*C enumerator*), 1465
bleResult_tag.gSmUnattainablePeripheralSecReqRequirements_c (*C enumerator*), 1466
bleResult_tag.gSmUnexpectedCommand_c (*C enumerator*), 1464
bleResult_tag.gSmUnexpectedKeyset_c (*C enumerator*), 1464
bleResult_tag.gSmUnexpectedKeyType_c (*C enumerator*), 1464
bleResult_tag.gSmUnexpectedPairingTerminationReason_c (*C enumerator*), 1464
bleResult_tag.gSmUnknownSmpPacketType_c (*C enumerator*), 1464
bleScanningFilterPolicy_t (*C type*), 1499
bleScanType_t (*C enum*), 1474
bleScanType_t.gScanTypeActive_c (*C enumerator*), 1474
bleScanType_t.gScanTypePassive_c (*C enumerator*), 1474
BleSig_IsGroupingAttributeUuid16 (*C macro*), 1518
BleSig_IsServiceDeclarationUuid16 (*C macro*), 1518
bleSlotDurations_t (*C type*), 1501
bleSlotDurations_tag (*C enum*), 1496
bleSlotDurations_tag.gSlotDurations1us_c (*C enumerator*), 1496
bleSlotDurations_tag.gSlotDurations2us_c (*C enumerator*), 1496
bleSupportedSwitchingSamplingRates_t (*C struct*), 1522, 1557
bleSupportedSwitchingSamplingRates_t.reserved (*C var*), 1523, 1557
bleSupportedSwitchingSamplingRates_t.samplingSupportedAodReception (*C var*), 1523, 1557
bleSupportedSwitchingSamplingRates_t.switchingSamplingSupportedAoaReception (*C var*), 1523, 1557
bleSupportedSwitchingSamplingRates_t.switchingSupportedAodTransmission (*C var*), 1523, 1557
bleSyncCteType_t (*C type*), 1501
bleSyncCteType_tag (*C struct*), 1535, 1569
bleSyncCteType_tag.doNotSyncWithAoA (*C var*), 1535, 1569
bleSyncCteType_tag.doNotSyncWithAoD1us (*C var*), 1535, 1569
bleSyncCteType_tag.doNotSyncWithAoD2us (*C var*), 1535, 1569
bleSyncCteType_tag.doNotSyncWithoutCte (*C var*), 1535, 1569
bleSyncCteType_tag.doNotSyncWithType3 (*C var*), 1535, 1569
bleSyncCteType_tag.reserved (*C var*), 1535, 1569
bleTransmitPowerChannelType_t (*C enum*), 1474
bleTransmitPowerChannelType_t.gTxPowerAdvChannel_c (*C enumerator*), 1474
bleTransmitPowerChannelType_t.gTxPowerConnChannel_c (*C enumerator*), 1474
bleTransmitPowerLevelType_t (*C enum*), 1474
bleTransmitPowerLevelType_t.gReadCurrentTxPowerLevel_c (*C enumerator*), 1474
bleTransmitPowerLevelType_t.gReadMaximumTxPowerLevel_c (*C enumerator*), 1474
bleTxPowerLevelFlags_t (*C struct*), 1535, 1569
bleTxPowerLevelFlags_t.maximum (*C var*), 1536, 1569
bleTxPowerLevelFlags_t.minimum (*C var*), 1536, 1569
bleTxPowerLevelFlags_t.reserved (*C var*), 1536, 1569
bleTxPowerReportingEnable_t (*C enum*), 1497
bleTxPowerReportingEnable_t.gTxPowerReportingDisable_c (*C enumerator*), 1497
bleTxPowerReportingEnable_t.gTxPowerReportingEnable_c (*C enumerator*), 1497
bleTxPowerReportingReason_t (*C type*), 1502
bleTxPowerReportingReason_tag (*C enum*), 1496
bleTxPowerReportingReason_tag.gLocalTxPowerChanged_c (*C enumerator*), 1496

bleTxPowerReportingReason_tag.gReadRemoteTxPowerLevelCommandCompleted_c (*C enumerator*), 1497
bleTxPowerReportingReason_tag.gRemoteTxPowerChanged_c (*C enumerator*), 1497
bleTxTestPacketPayload_t (*C enum*), 1476
bleTxTestPacketPayload_t.gBleTestPacketPayloadPattern00001111_c (*C enumerator*), 1477
bleTxTestPacketPayload_t.gBleTestPacketPayloadPattern01010101_c (*C enumerator*), 1477
bleTxTestPacketPayload_t.gBleTestPacketPayloadPattern10101010_c (*C enumerator*), 1476
bleTxTestPacketPayload_t.gBleTestPacketPayloadPattern11110000_c (*C enumerator*), 1476
bleTxTestPacketPayload_t.gBleTestPacketPayloadPatternAllBits0_c (*C enumerator*), 1477
bleTxTestPacketPayload_t.gBleTestPacketPayloadPatternAllBits1_c (*C enumerator*), 1477
bleTxTestPacketPayload_t.gBleTestPacketPayloadPrbs9_c (*C enumerator*), 1476
bleTxTestPacketPayload_t.gBleTestPacketPayloadPrbs15_c (*C enumerator*), 1477
bleUuid_t (*C union*), 1520, 1554
bleUuid_t.uuid16 (*C var*), 1520, 1554
bleUuid_t.uuid32 (*C var*), 1520, 1554
bleUuid_t.uuid128 (*C var*), 1520, 1554
bleUuidType_t (*C type*), 1499

D

deviceId_t (*C type*), 1499
drivers_MCX_MCXW_MCXW727C._ccm32k_clock_monitor_config (*C struct*), 166
drivers_MCX_MCXW_MCXW727C._ccm32k_clock_monitor_config.divideTrimValue (*C var*), 166
drivers_MCX_MCXW_MCXW727C._ccm32k_clock_monitor_config.enableClockMonitor (*C var*), 166
drivers_MCX_MCXW_MCXW727C._ccm32k_clock_monitor_config.freqTrimValue (*C var*), 166
drivers_MCX_MCXW_MCXW727C._ccm32k_clock_monitor_divide_trim_value (*C enum*), 164
drivers_MCX_MCXW_MCXW727C._ccm32k_clock_monitor_divide_trim_value.kCCM32K_ClockMonitor_1kHzFr (*C enumerator*), 164
drivers_MCX_MCXW_MCXW727C._ccm32k_clock_monitor_divide_trim_value.kCCM32K_ClockMonitor_1KHzF (*C enumerator*), 164
drivers_MCX_MCXW_MCXW727C._ccm32k_clock_monitor_divide_trim_value.kCCM32K_ClockMonitor_64HzFr (*C enumerator*), 164
drivers_MCX_MCXW_MCXW727C._ccm32k_clock_monitor_divide_trim_value.kCCM32K_ClockMonitor_64HzFr (*C enumerator*), 164
drivers_MCX_MCXW_MCXW727C._ccm32k_clock_monitor_freq_trim_value (*C enum*), 164
drivers_MCX_MCXW_MCXW727C._ccm32k_clock_monitor_freq_trim_value.kCCM32K_ClockMonitor2CycleAsse (*C enumerator*), 164
drivers_MCX_MCXW_MCXW727C._ccm32k_clock_monitor_freq_trim_value.kCCM32K_ClockMonitor4CycleAsse (*C enumerator*), 164
drivers_MCX_MCXW_MCXW727C._ccm32k_clock_monitor_freq_trim_value.kCCM32K_ClockMonitor6CycleAsse (*C enumerator*), 164
drivers_MCX_MCXW_MCXW727C._ccm32k_clock_monitor_freq_trim_value.kCCM32K_ClockMonitor8CycleAsse (*C enumerator*), 164
drivers_MCX_MCXW_MCXW727C._ccm32k_clock_output_peripheral (*C enum*), 165
drivers_MCX_MCXW_MCXW727C._ccm32k_clock_output_peripheral.kCCM32K_ClockOutToNbu (*C enumerator*), 165
drivers_MCX_MCXW_MCXW727C._ccm32k_clock_output_peripheral.kCCM32K_ClockOutToOtherModules (*C enumerator*), 165
drivers_MCX_MCXW_MCXW727C._ccm32k_clock_output_peripheral.kCCM32K_ClockOutToRfmc (*C enumerator*), 165
drivers_MCX_MCXW_MCXW727C._ccm32k_clock_output_peripheral.kCCM32K_ClockOutToRtc (*C enumerator*), 165
drivers_MCX_MCXW_MCXW727C._ccm32k_clock_output_peripheral.kCCM32K_ClockOutToWuuRmcPortD (*C enumerator*), 165
drivers_MCX_MCXW_MCXW727C._ccm32k_clock_source (*C enum*), 163
drivers_MCX_MCXW_MCXW727C._ccm32k_clock_source_select (*C enum*), 164
drivers_MCX_MCXW_MCXW727C._ccm32k_clock_source_select.kCCM32K_ClockSourceSelectFro32k (*C enumerator*), 165

drivers_MCX_MCXW_MCXW727C._ccm32k_clock_source_select.kCCM32K_ClockSourceSelectOsc32k (*C enumerator*), 165
drivers_MCX_MCXW_MCXW727C._ccm32k_clock_source.kCCM32K_ClockSource32kFro (*C enumerator*), 164
drivers_MCX_MCXW_MCXW727C._ccm32k_clock_source.kCCM32K_ClockSource32kOsc (*C enumerator*), 164
drivers_MCX_MCXW_MCXW727C._ccm32k_clock_source.kCCM32K_ClockSourceNone (*C enumerator*), 164
drivers_MCX_MCXW_MCXW727C._ccm32k_osc_coarse_adjustment_value (*C enum*), 162
drivers_MCX_MCXW_MCXW727C._ccm32k_osc_coarse_adjustment_value.kCCM32K_OscCoarseAdjustmentRang (*C enumerator*), 162
drivers_MCX_MCXW_MCXW727C._ccm32k_osc_coarse_adjustment_value.kCCM32K_OscCoarseAdjustmentRang (*C enumerator*), 162
drivers_MCX_MCXW_MCXW727C._ccm32k_osc_coarse_adjustment_value.kCCM32K_OscCoarseAdjustmentRang (*C enumerator*), 162
drivers_MCX_MCXW_MCXW727C._ccm32k_osc_coarse_adjustment_value.kCCM32K_OscCoarseAdjustmentRang (*C enumerator*), 162
drivers_MCX_MCXW_MCXW727C._ccm32k_osc_config (*C struct*), 166
drivers_MCX_MCXW_MCXW727C._ccm32k_osc_config.coarseAdjustment (*C var*), 167
drivers_MCX_MCXW_MCXW727C._ccm32k_osc_config.enableInternalCapBank (*C var*), 167
drivers_MCX_MCXW_MCXW727C._ccm32k_osc_config.extalCap (*C var*), 167
drivers_MCX_MCXW_MCXW727C._ccm32k_osc_config.fineAdjustment (*C var*), 167
drivers_MCX_MCXW_MCXW727C._ccm32k_osc_config.xtalCap (*C var*), 167
drivers_MCX_MCXW_MCXW727C._ccm32k_osc_extal_cap (*C enum*), 161
drivers_MCX_MCXW_MCXW727C._ccm32k_osc_extal_cap.kCCM32K_OscExtal0pFCap (*C enumerator*), 161
drivers_MCX_MCXW_MCXW727C._ccm32k_osc_extal_cap.kCCM32K_OscExtal2pFCap (*C enumerator*), 161
drivers_MCX_MCXW_MCXW727C._ccm32k_osc_extal_cap.kCCM32K_OscExtal4pFCap (*C enumerator*), 161
drivers_MCX_MCXW_MCXW727C._ccm32k_osc_extal_cap.kCCM32K_OscExtal6pFCap (*C enumerator*), 161
drivers_MCX_MCXW_MCXW727C._ccm32k_osc_extal_cap.kCCM32K_OscExtal8pFCap (*C enumerator*), 161
drivers_MCX_MCXW_MCXW727C._ccm32k_osc_extal_cap.kCCM32K_OscExtal10pFCap (*C enumerator*), 161
drivers_MCX_MCXW_MCXW727C._ccm32k_osc_extal_cap.kCCM32K_OscExtal12pFCap (*C enumerator*), 161
drivers_MCX_MCXW_MCXW727C._ccm32k_osc_extal_cap.kCCM32K_OscExtal14pFCap (*C enumerator*), 162
drivers_MCX_MCXW_MCXW727C._ccm32k_osc_extal_cap.kCCM32K_OscExtal16pFCap (*C enumerator*), 162
drivers_MCX_MCXW_MCXW727C._ccm32k_osc_extal_cap.kCCM32K_OscExtal18pFCap (*C enumerator*), 162
drivers_MCX_MCXW_MCXW727C._ccm32k_osc_extal_cap.kCCM32K_OscExtal20pFCap (*C enumerator*), 162
drivers_MCX_MCXW_MCXW727C._ccm32k_osc_extal_cap.kCCM32K_OscExtal22pFCap (*C enumerator*), 162
drivers_MCX_MCXW_MCXW727C._ccm32k_osc_extal_cap.kCCM32K_OscExtal24pFCap (*C enumerator*), 162
drivers_MCX_MCXW_MCXW727C._ccm32k_osc_extal_cap.kCCM32K_OscExtal26pFCap (*C enumerator*), 162
drivers_MCX_MCXW_MCXW727C._ccm32k_osc_extal_cap.kCCM32K_OscExtal28pFCap (*C enumerator*), 162
drivers_MCX_MCXW_MCXW727C._ccm32k_osc_extal_cap.kCCM32K_OscExtal30pFCap (*C enumerator*), 162
drivers_MCX_MCXW_MCXW727C._ccm32k_osc_fine_adjustment_value (*C enum*), 162
drivers_MCX_MCXW_MCXW727C._ccm32k_osc_fine_adjustment_value.kCCM32K_OscFineAdjustmentRange0

(*C enumerator*), 162
drivers_MCX_MCXW_MCXW727C._ccm32k_osc_mode (*C enum*), 163
drivers_MCX_MCXW_MCXW727C._ccm32k_osc_mode.kCCM32K_Bypass32kHzCrystalOsc (*C enumerator*), 163
drivers_MCX_MCXW_MCXW727C._ccm32k_osc_mode.kCCM32K_Disable32kHzCrystalOsc (*C enumerator*), 163
drivers_MCX_MCXW_MCXW727C._ccm32k_osc_mode.kCCM32K_Enable32kHzCrystalOsc (*C enumerator*), 163
drivers_MCX_MCXW_MCXW727C._ccm32k_osc_xtal_cap (*C enum*), 160
drivers_MCX_MCXW_MCXW727C._ccm32k_osc_xtal_cap.kCCM32K_OscXtal0pFCap (*C enumerator*), 160
drivers_MCX_MCXW_MCXW727C._ccm32k_osc_xtal_cap.kCCM32K_OscXtal2pFCap (*C enumerator*), 160
drivers_MCX_MCXW_MCXW727C._ccm32k_osc_xtal_cap.kCCM32K_OscXtal4pFCap (*C enumerator*), 160
drivers_MCX_MCXW_MCXW727C._ccm32k_osc_xtal_cap.kCCM32K_OscXtal6pFCap (*C enumerator*), 160
drivers_MCX_MCXW_MCXW727C._ccm32k_osc_xtal_cap.kCCM32K_OscXtal8pFCap (*C enumerator*), 160
drivers_MCX_MCXW_MCXW727C._ccm32k_osc_xtal_cap.kCCM32K_OscXtal10pFCap (*C enumerator*), 160
drivers_MCX_MCXW_MCXW727C._ccm32k_osc_xtal_cap.kCCM32K_OscXtal12pFCap (*C enumerator*), 160
drivers_MCX_MCXW_MCXW727C._ccm32k_osc_xtal_cap.kCCM32K_OscXtal14pFCap (*C enumerator*), 160
drivers_MCX_MCXW_MCXW727C._ccm32k_osc_xtal_cap.kCCM32K_OscXtal16pFCap (*C enumerator*), 161
drivers_MCX_MCXW_MCXW727C._ccm32k_osc_xtal_cap.kCCM32K_OscXtal18pFCap (*C enumerator*), 161
drivers_MCX_MCXW_MCXW727C._ccm32k_osc_xtal_cap.kCCM32K_OscXtal20pFCap (*C enumerator*), 161
drivers_MCX_MCXW_MCXW727C._ccm32k_osc_xtal_cap.kCCM32K_OscXtal22pFCap (*C enumerator*), 161
drivers_MCX_MCXW_MCXW727C._ccm32k_osc_xtal_cap.kCCM32K_OscXtal24pFCap (*C enumerator*), 161
drivers_MCX_MCXW_MCXW727C._ccm32k_osc_xtal_cap.kCCM32K_OscXtal26pFCap (*C enumerator*), 161
drivers_MCX_MCXW_MCXW727C._ccm32k_osc_xtal_cap.kCCM32K_OscXtal28pFCap (*C enumerator*), 161
drivers_MCX_MCXW_MCXW727C._ccm32k_osc_xtal_cap.kCCM32K_OscXtal30pFCap (*C enumerator*), 161
drivers_MCX_MCXW_MCXW727C._ccm32k_state (*C enum*), 163
drivers_MCX_MCXW_MCXW727C._ccm32k_state.kCCM32K_Both32kFro32kOscDisabled (*C enumerator*), 163
drivers_MCX_MCXW_MCXW727C._ccm32k_state.kCCM32K_Both32kFro32kOscEnabled (*C enumerator*), 163
drivers_MCX_MCXW_MCXW727C._ccm32k_state.kCCM32K_Only32kFroEnabled (*C enumerator*), 163
drivers_MCX_MCXW_MCXW727C._ccm32k_state.kCCM32K_Only32kOscEnabled (*C enumerator*), 163
drivers_MCX_MCXW_MCXW727C._ce_copy_image (*C struct*), 170
drivers_MCX_MCXW_MCXW727C._clock_clkout_src (*C enum*), 188
drivers_MCX_MCXW_MCXW727C._clock_clkout_src.kClockClkoutSelFirc (*C enumerator*), 188
drivers_MCX_MCXW_MCXW727C._clock_clkout_src.kClockClkoutSelScgRtcOsc (*C enumerator*), 188
drivers_MCX_MCXW_MCXW727C._clock_clkout_src.kClockClkoutSelScgSlow (*C enumerator*), 188
drivers_MCX_MCXW_MCXW727C._clock_clkout_src.kClockClkoutSelSosc (*C enumerator*), 188

drivers_MCX_MCXW_MCXW727C._clock_ip_control (*C enum*), 182
drivers_MCX_MCXW_MCXW727C._clock_ip_control.kCLOCK_IpClkControl_fun0 (*C enumerator*), 182
drivers_MCX_MCXW_MCXW727C._clock_ip_control.kCLOCK_IpClkControl_fun1 (*C enumerator*), 182
drivers_MCX_MCXW_MCXW727C._clock_ip_control.kCLOCK_IpClkControl_fun2 (*C enumerator*), 182
drivers_MCX_MCXW_MCXW727C._clock_ip_control.kCLOCK_IpClkControl_fun3 (*C enumerator*), 182
drivers_MCX_MCXW_MCXW727C._clock_ip_name (*C enum*), 183
drivers_MCX_MCXW_MCXW727C._clock_ip_name.kCLOCK_Can0 (*C enumerator*), 184
drivers_MCX_MCXW_MCXW727C._clock_ip_name.kCLOCK_Can1 (*C enumerator*), 185
drivers_MCX_MCXW_MCXW727C._clock_ip_name.kCLOCK_Crc0 (*C enumerator*), 183
drivers_MCX_MCXW_MCXW727C._clock_ip_name.kCLOCK_Data_stream_2p4 (*C enumerator*), 184
drivers_MCX_MCXW_MCXW727C._clock_ip_name.kCLOCK_Dma0 (*C enumerator*), 185
drivers_MCX_MCXW_MCXW727C._clock_ip_name.kCLOCK_DSP0 (*C enumerator*), 185
drivers_MCX_MCXW_MCXW727C._clock_ip_name.kCLOCK_DSP0_MUA (*C enumerator*), 185
drivers_MCX_MCXW_MCXW727C._clock_ip_name.kCLOCK_Ewm0 (*C enumerator*), 183
drivers_MCX_MCXW_MCXW727C._clock_ip_name.kCLOCK_Flexio0 (*C enumerator*), 184
drivers_MCX_MCXW_MCXW727C._clock_ip_name.kCLOCK_GpioA (*C enumerator*), 185
drivers_MCX_MCXW_MCXW727C._clock_ip_name.kCLOCK_GpioB (*C enumerator*), 185
drivers_MCX_MCXW_MCXW727C._clock_ip_name.kCLOCK_GpioC (*C enumerator*), 185
drivers_MCX_MCXW_MCXW727C._clock_ip_name.kCLOCK_I3c0 (*C enumerator*), 184
drivers_MCX_MCXW_MCXW727C._clock_ip_name.kCLOCK_Lpadc0 (*C enumerator*), 184
drivers_MCX_MCXW_MCXW727C._clock_ip_name.kCLOCK_Lpcmp0 (*C enumerator*), 184
drivers_MCX_MCXW_MCXW727C._clock_ip_name.kCLOCK_Lpcmp1 (*C enumerator*), 184
drivers_MCX_MCXW_MCXW727C._clock_ip_name.kCLOCK_Lpi2c0 (*C enumerator*), 183
drivers_MCX_MCXW_MCXW727C._clock_ip_name.kCLOCK_Lpi2c1 (*C enumerator*), 183
drivers_MCX_MCXW_MCXW727C._clock_ip_name.kCLOCK_Lpit0 (*C enumerator*), 183
drivers_MCX_MCXW_MCXW727C._clock_ip_name.kCLOCK_Lpspi0 (*C enumerator*), 184
drivers_MCX_MCXW_MCXW727C._clock_ip_name.kCLOCK_Lpspi1 (*C enumerator*), 184
drivers_MCX_MCXW_MCXW727C._clock_ip_name.kCLOCK_Lptmr0_wake2vsys (*C enumerator*), 186
drivers_MCX_MCXW_MCXW727C._clock_ip_name.kCLOCK_Lptmr1_wake2vsys (*C enumerator*), 186
drivers_MCX_MCXW_MCXW727C._clock_ip_name.kCLOCK_Lptmr2_wake2vsys (*C enumerator*), 186
drivers_MCX_MCXW_MCXW727C._clock_ip_name.kCLOCK_Lpuart0 (*C enumerator*), 184
drivers_MCX_MCXW_MCXW727C._clock_ip_name.kCLOCK_Lpuart1 (*C enumerator*), 184
drivers_MCX_MCXW_MCXW727C._clock_ip_name.kCLOCK_Mtr_master (*C enumerator*), 185
drivers_MCX_MCXW_MCXW727C._clock_ip_name.kCLOCK_NOGATE (*C enumerator*), 183
drivers_MCX_MCXW_MCXW727C._clock_ip_name.kCLOCK_Pflexnvm (*C enumerator*), 185
drivers_MCX_MCXW_MCXW727C._clock_ip_name.kCLOCK_PortA (*C enumerator*), 184
drivers_MCX_MCXW_MCXW727C._clock_ip_name.kCLOCK_PortB (*C enumerator*), 184
drivers_MCX_MCXW_MCXW727C._clock_ip_name.kCLOCK_PortC (*C enumerator*), 184
drivers_MCX_MCXW_MCXW727C._clock_ip_name.kCLOCK_Rf_2p4ghz_bist (*C enumerator*), 185
drivers_MCX_MCXW_MCXW727C._clock_ip_name.kCLOCK_Secsubsys (*C enumerator*), 183
drivers_MCX_MCXW_MCXW727C._clock_ip_name.kCLOCK_Sema0 (*C enumerator*), 184
drivers_MCX_MCXW_MCXW727C._clock_ip_name.kCLOCK_Sfa0 (*C enumerator*), 183
drivers_MCX_MCXW_MCXW727C._clock_ip_name.kCLOCK_Sram0 (*C enumerator*), 185
drivers_MCX_MCXW_MCXW727C._clock_ip_name.kCLOCK_Sram0_NOECC (*C enumerator*), 185
drivers_MCX_MCXW_MCXW727C._clock_ip_name.kCLOCK_Sram1 (*C enumerator*), 185
drivers_MCX_MCXW_MCXW727C._clock_ip_name.kCLOCK_Sram1_NOECC (*C enumerator*), 185
drivers_MCX_MCXW_MCXW727C._clock_ip_name.kCLOCK_Sram2 (*C enumerator*), 185

drivers_MCX_MCXW_MCXW727C._clock_ip_name.kCLOCK_Sram3 (*C enumerator*), 185
drivers_MCX_MCXW_MCXW727C._clock_ip_name.kCLOCK_Syspm0 (*C enumerator*), 183
drivers_MCX_MCXW_MCXW727C._clock_ip_name.kCLOCK_Tpm0 (*C enumerator*), 183
drivers_MCX_MCXW_MCXW727C._clock_ip_name.kCLOCK_Tpm1 (*C enumerator*), 183
drivers_MCX_MCXW_MCXW727C._clock_ip_name.kCLOCK_Vref0 (*C enumerator*), 184
drivers_MCX_MCXW_MCXW727C._clock_ip_name.kCLOCK_Wdog0 (*C enumerator*), 183
drivers_MCX_MCXW_MCXW727C._clock_ip_name.kCLOCK_Wdog1 (*C enumerator*), 183
drivers_MCX_MCXW_MCXW727C._clock_ip_src (*C enum*), 182
drivers_MCX_MCXW_MCXW727C._clock_ip_src.kCLOCK_IpSrc32kClk (*C enumerator*), 182
drivers_MCX_MCXW_MCXW727C._clock_ip_src.kCLOCK_IpSrcFro6M (*C enumerator*), 182
drivers_MCX_MCXW_MCXW727C._clock_ip_src.kCLOCK_IpSrcFro192M (*C enumerator*), 182
drivers_MCX_MCXW_MCXW727C._clock_ip_src.kCLOCK_IpSrcSoscClk (*C enumerator*), 182
drivers_MCX_MCXW_MCXW727C._clock_name (*C enum*), 181
drivers_MCX_MCXW_MCXW727C._clock_name.kCLOCK_BusClk (*C enumerator*), 181
drivers_MCX_MCXW_MCXW727C._clock_name.kCLOCK_CoreSysClk (*C enumerator*), 181
drivers_MCX_MCXW_MCXW727C._clock_name.kCLOCK_PlatClk (*C enumerator*), 181
drivers_MCX_MCXW_MCXW727C._clock_name.kCLOCK_RtcOscClk (*C enumerator*), 181
drivers_MCX_MCXW_MCXW727C._clock_name.kCLOCK_ScgFircClk (*C enumerator*), 181
drivers_MCX_MCXW_MCXW727C._clock_name.kCLOCK_ScgSysOscClk (*C enumerator*), 181
drivers_MCX_MCXW_MCXW727C._clock_name.kCLOCK_SlowClk (*C enumerator*), 181
drivers_MCX_MCXW_MCXW727C._clock_name.kCLOCK_SysClk (*C enumerator*), 181
drivers_MCX_MCXW_MCXW727C._cmc_clock_mode (*C enum*), 209
drivers_MCX_MCXW_MCXW727C._cmc_clock_mode.kCMC_GateAllSystemClocks (*C enumerator*), 209
drivers_MCX_MCXW_MCXW727C._cmc_clock_mode.kCMC_GateAllSystemClocksEnterLowPowerMode (*C enumerator*), 210
drivers_MCX_MCXW_MCXW727C._cmc_clock_mode.kCMC_GateCoreClock (*C enumerator*), 209
drivers_MCX_MCXW_MCXW727C._cmc_clock_mode.kCMC_GateCorePlatformClock (*C enumerator*), 209
drivers_MCX_MCXW_MCXW727C._cmc_clock_mode.kCMC_GateNoneClock (*C enumerator*), 209
drivers_MCX_MCXW_MCXW727C._cmc_core_clock_gate_status (*C enum*), 209
drivers_MCX_MCXW_MCXW727C._cmc_core_clock_gate_status.kCMC_CoreClockGated (*C enumerator*), 209
drivers_MCX_MCXW_MCXW727C._cmc_core_clock_gate_status.kCMC_CoreClockNotGated (*C enumerator*), 209
drivers_MCX_MCXW_MCXW727C._cmc_low_power_mode (*C enum*), 210
drivers_MCX_MCXW_MCXW727C._cmc_low_power_mode.kCMC_ActiveMode (*C enumerator*), 210
drivers_MCX_MCXW_MCXW727C._cmc_low_power_mode.kCMC_DeepPowerDown (*C enumerator*), 210
drivers_MCX_MCXW_MCXW727C._cmc_low_power_mode.kCMC_DeepSleepMode (*C enumerator*), 210
drivers_MCX_MCXW_MCXW727C._cmc_low_power_mode.kCMC_PowerDownMode (*C enumerator*), 210
drivers_MCX_MCXW_MCXW727C._cmc_low_power_mode.kCMC_SleepMode (*C enumerator*), 210
drivers_MCX_MCXW_MCXW727C._cmc_power_domain_config (*C struct*), 211
drivers_MCX_MCXW_MCXW727C._cmc_power_domain_config.clock_mode (*C var*), 211
drivers_MCX_MCXW_MCXW727C._cmc_power_domain_config.main_domain (*C var*), 211
drivers_MCX_MCXW_MCXW727C._cmc_power_domain_config.wake_domain (*C var*), 211
drivers_MCX_MCXW_MCXW727C._cmc_power_mode_protection (*C enum*), 205
drivers_MCX_MCXW_MCXW727C._cmc_power_mode_protection.kCMC_AllowAllLowPowerModes (*C enumerator*), 205
drivers_MCX_MCXW_MCXW727C._cmc_power_mode_protection.kCMC_AllowDeepPowerDownMode (*C enumerator*), 205
drivers_MCX_MCXW_MCXW727C._cmc_power_mode_protection.kCMC_AllowDeepSleepMode

(*C enumerator*), 205
drivers_MCX_MCXW_MCXW727C._cmc_power_mode_protection.kCMC_AllowPowerDownMode (*C enumerator*), 205
drivers_MCX_MCXW_MCXW727C._cmc_power_mode_protection.kCMC_AllowSleepMode (*C enumerator*), 205
drivers_MCX_MCXW_MCXW727C._cmc_reset_pin_config (*C struct*), 210
drivers_MCX_MCXW_MCXW727C._cmc_reset_pin_config.lowpowerFilterEnable (*C var*), 211
drivers_MCX_MCXW_MCXW727C._cmc_reset_pin_config.resetFilterEnable (*C var*), 211
drivers_MCX_MCXW_MCXW727C._cmc_reset_pin_config.resetFilterWidth (*C var*), 211
drivers_MCX_MCXW_MCXW727C._cmc_system_reset_interrupt_enable (*C enum*), 206
drivers_MCX_MCXW_MCXW727C._cmc_system_reset_interrupt_enable.kCMC_DAPResetInterruptEnable (*C enumerator*), 206
drivers_MCX_MCXW_MCXW727C._cmc_system_reset_interrupt_enable.kCMC_LockupResetInterruptEnable (*C enumerator*), 206
drivers_MCX_MCXW_MCXW727C._cmc_system_reset_interrupt_enable.kCMC_LowPowerAcknowledgeTimeoutR (*C enumerator*), 206
drivers_MCX_MCXW_MCXW727C._cmc_system_reset_interrupt_enable.kCMC_PinResetInterruptEnable (*C enumerator*), 206
drivers_MCX_MCXW_MCXW727C._cmc_system_reset_interrupt_enable.kCMC_SoftwareResetInterruptEnable (*C enumerator*), 206
drivers_MCX_MCXW_MCXW727C._cmc_system_reset_interrupt_enable.kCMC_SystemClockGenerationResetInte (*C enumerator*), 206
drivers_MCX_MCXW_MCXW727C._cmc_system_reset_interrupt_enable.kCMC_Watchdog0ResetInterruptEnable (*C enumerator*), 206
drivers_MCX_MCXW_MCXW727C._cmc_system_reset_interrupt_enable.kCMC_Watchdog1ResetInterruptEnable (*C enumerator*), 207
drivers_MCX_MCXW_MCXW727C._cmc_system_reset_interrupt_flag (*C enum*), 207
drivers_MCX_MCXW_MCXW727C._cmc_system_reset_interrupt_flag.kCMC_DAPResetInterruptFlag (*C enumerator*), 207
drivers_MCX_MCXW_MCXW727C._cmc_system_reset_interrupt_flag.kCMC_LockupResetInterruptFlag (*C enumerator*), 207
drivers_MCX_MCXW_MCXW727C._cmc_system_reset_interrupt_flag.kCMC_LowPowerAcknowledgeTimeoutRese (*C enumerator*), 207
drivers_MCX_MCXW_MCXW727C._cmc_system_reset_interrupt_flag.kCMC_PinResetInterruptFlag (*C enumerator*), 207
drivers_MCX_MCXW_MCXW727C._cmc_system_reset_interrupt_flag.kCMC_SoftwareResetInterruptFlag (*C enumerator*), 207
drivers_MCX_MCXW_MCXW727C._cmc_system_reset_interrupt_flag.kCMC_Watchdog0ResetInterruptFlag (*C enumerator*), 207
drivers_MCX_MCXW_MCXW727C._cmc_system_reset_interrupt_flag.kCMC_Watchdog1ResetInterruptFlag (*C enumerator*), 207
drivers_MCX_MCXW_MCXW727C._cmc_system_reset_sources (*C enum*), 208
drivers_MCX_MCXW_MCXW727C._cmc_system_reset_sources.kCMC_DAPReset (*C enumerator*), 208
drivers_MCX_MCXW_MCXW727C._cmc_system_reset_sources.kCMC_FatalReset (*C enumerator*), 208
drivers_MCX_MCXW_MCXW727C._cmc_system_reset_sources.kCMC_HVDReset (*C enumerator*), 208
drivers_MCX_MCXW_MCXW727C._cmc_system_reset_sources.kCMC_LockUpReset (*C enumerator*), 209
drivers_MCX_MCXW_MCXW727C._cmc_system_reset_sources.kCMC_LowPowerAcknowledgeTimeoutReset (*C enumerator*), 209
drivers_MCX_MCXW_MCXW727C._cmc_system_reset_sources.kCMC_LVDReset (*C enumerator*), 208
drivers_MCX_MCXW_MCXW727C._cmc_system_reset_sources.kCMC_PinReset (*C enumerator*), 208
drivers_MCX_MCXW_MCXW727C._cmc_system_reset_sources.kCMC_PORReset (*C enumerator*), 208

drivers_MCX_MCXW_MCXW727C._cmc_system_reset_sources.kCMC_ResetTimeout (*C enumerator*), 209
drivers_MCX_MCXW_MCXW727C._cmc_system_reset_sources.kCMC_SCGReset (*C enumerator*), 209
drivers_MCX_MCXW_MCXW727C._cmc_system_reset_sources.kCMC_SoftwareReset (*C enumerator*), 209
drivers_MCX_MCXW_MCXW727C._cmc_system_reset_sources.kCMC_WakeUpReset (*C enumerator*), 208
drivers_MCX_MCXW_MCXW727C._cmc_system_reset_sources.kCMC_WarmReset (*C enumerator*), 208
drivers_MCX_MCXW_MCXW727C._cmc_system_reset_sources.kCMC_Watchdog0Reset (*C enumerator*), 209
drivers_MCX_MCXW_MCXW727C._cmc_system_reset_sources.kCMC_Watchdog1Reset (*C enumerator*), 209
drivers_MCX_MCXW_MCXW727C._cmc_system_sram_arrays (*C enum*), 207
drivers_MCX_MCXW_MCXW727C._cmc_system_sram_arrays.kCMC_AllSramArrays (*C enumerator*), 208
drivers_MCX_MCXW_MCXW727C._cmc_system_sram_arrays.kCMC_SRAMBank0 (*C enumerator*), 207
drivers_MCX_MCXW_MCXW727C._cmc_system_sram_arrays.kCMC_SRAMBank1 (*C enumerator*), 207
drivers_MCX_MCXW_MCXW727C._cmc_system_sram_arrays.kCMC_SRAMBank2 (*C enumerator*), 207
drivers_MCX_MCXW_MCXW727C._cmc_system_sram_arrays.kCMC_SRAMBank3 (*C enumerator*), 207
drivers_MCX_MCXW_MCXW727C._cmc_system_sram_arrays.kCMC_SRAMBank4 (*C enumerator*), 207
drivers_MCX_MCXW_MCXW727C._cmc_system_sram_arrays.kCMC_SRAMBank5 (*C enumerator*), 208
drivers_MCX_MCXW_MCXW727C._cmc_system_sram_arrays.kCMC_SRAMBank6 (*C enumerator*), 208
drivers_MCX_MCXW_MCXW727C._cmc_system_sram_arrays.kCMC_SRAMBank7 (*C enumerator*), 208
drivers_MCX_MCXW_MCXW727C._cmc_system_sram_arrays.kCMC_SRAMBank8 (*C enumerator*), 208
drivers_MCX_MCXW_MCXW727C._cmc_system_sram_arrays.kCMC_SRAMBank9 (*C enumerator*), 208
drivers_MCX_MCXW_MCXW727C._cmc_system_sram_arrays.kCMC_SRAMBank10 (*C enumerator*), 208
drivers_MCX_MCXW_MCXW727C._cmc_wakeup_sources (*C enum*), 206
drivers_MCX_MCXW_MCXW727C._cmc_wakeup_sources.kCMC_WakeupFromBusMaster (*C enumerator*), 206
drivers_MCX_MCXW_MCXW727C._cmc_wakeup_sources.kCMC_WakeupFromDebugRequest (*C enumerator*), 206
drivers_MCX_MCXW_MCXW727C._cmc_wakeup_sources.kCMC_WakeupFromDMAWakeup (*C enumerator*), 206
drivers_MCX_MCXW_MCXW727C._cmc_wakeup_sources.kCMC_WakeupFromInterrupt (*C enumerator*), 206
drivers_MCX_MCXW_MCXW727C._cmc_wakeup_sources.kCMC_WakeupFromResetInterruptOrPowerDown (*C enumerator*), 206
drivers_MCX_MCXW_MCXW727C._cmc_wakeup_sources.kCMC_WakeupFromWUURequest (*C enumerator*), 206
drivers_MCX_MCXW_MCXW727C._crc_bits (*C enum*), 212
drivers_MCX_MCXW_MCXW727C._crc_bits.kCrcBits16 (*C enumerator*), 212
drivers_MCX_MCXW_MCXW727C._crc_bits.kCrcBits32 (*C enumerator*), 212
drivers_MCX_MCXW_MCXW727C._crc_config (*C struct*), 213
drivers_MCX_MCXW_MCXW727C._crc_config.complementChecksum (*C var*), 214
drivers_MCX_MCXW_MCXW727C._crc_config.crcBits (*C var*), 214

drivers_MCX_MCXW_MCXW727C._crc_config.crcResult (*C var*), 214
drivers_MCX_MCXW_MCXW727C._crc_config.polynomial (*C var*), 214
drivers_MCX_MCXW_MCXW727C._crc_config.reflectIn (*C var*), 214
drivers_MCX_MCXW_MCXW727C._crc_config.reflectOut (*C var*), 214
drivers_MCX_MCXW_MCXW727C._crc_config.seed (*C var*), 214
drivers_MCX_MCXW_MCXW727C._crc_result (*C enum*), 212
drivers_MCX_MCXW_MCXW727C._crc_result.kCrcFinalChecksum (*C enumerator*), 212
drivers_MCX_MCXW_MCXW727C._crc_result.kCrcIntermediateChecksum (*C enumerator*), 212
drivers_MCX_MCXW_MCXW727C._edma_bandwidth (*C enum*), 229
drivers_MCX_MCXW_MCXW727C._edma_bandwidth.kEDMA_BandwidthStall4Cycle (*C enumerator*), 229
drivers_MCX_MCXW_MCXW727C._edma_bandwidth.kEDMA_BandwidthStall8Cycle (*C enumerator*), 229
drivers_MCX_MCXW_MCXW727C._edma_bandwidth.kEDMA_BandwidthStallNone (*C enumerator*), 229
drivers_MCX_MCXW_MCXW727C._edma_channel_link_type (*C enum*), 229
drivers_MCX_MCXW_MCXW727C._edma_channel_link_type.kEDMA_LinkNone (*C enumerator*), 229
drivers_MCX_MCXW_MCXW727C._edma_channel_link_type.kEDMA_MajorLink (*C enumerator*), 229
drivers_MCX_MCXW_MCXW727C._edma_channel_link_type.kEDMA_MinorLink (*C enumerator*), 229
drivers_MCX_MCXW_MCXW727C._edma_channel_Preemption_config (*C struct*), 234
drivers_MCX_MCXW_MCXW727C._edma_channel_Preemption_config.channelPriority (*C var*), 234
drivers_MCX_MCXW_MCXW727C._edma_channel_Preemption_config.enableChannelPreemption (*C var*), 234
drivers_MCX_MCXW_MCXW727C._edma_channel_Preemption_config.enablePreemptAbility (*C var*), 234
drivers_MCX_MCXW_MCXW727C._edma_channel_protection_level (*C enum*), 230
drivers_MCX_MCXW_MCXW727C._edma_channel_protection_level.kEDMA_ChannelProtectionLevelPrivileged (*C enumerator*), 230
drivers_MCX_MCXW_MCXW727C._edma_channel_protection_level.kEDMA_ChannelProtectionLevelUser (*C enumerator*), 230
drivers_MCX_MCXW_MCXW727C._edma_channel_security_level (*C enum*), 229
drivers_MCX_MCXW_MCXW727C._edma_channel_security_level.kEDMA_ChannelSecurityLevelNonSecure (*C enumerator*), 229
drivers_MCX_MCXW_MCXW727C._edma_channel_security_level.kEDMA_ChannelSecurityLevelSecure (*C enumerator*), 230
drivers_MCX_MCXW_MCXW727C._edma_config (*C struct*), 233
drivers_MCX_MCXW_MCXW727C._edma_config.enableDebugMode (*C var*), 233
drivers_MCX_MCXW_MCXW727C._edma_config.enableHaltOnError (*C var*), 233
drivers_MCX_MCXW_MCXW727C._edma_config.enableMasterIdReplication (*C var*), 233
drivers_MCX_MCXW_MCXW727C._edma_config.enableRoundRobinArbitration (*C var*), 233
drivers_MCX_MCXW_MCXW727C._edma_handle (*C struct*), 235
drivers_MCX_MCXW_MCXW727C._edma_handle.base (*C var*), 236
drivers_MCX_MCXW_MCXW727C._edma_handle.callback (*C var*), 236
drivers_MCX_MCXW_MCXW727C._edma_handle.channel (*C var*), 236
drivers_MCX_MCXW_MCXW727C._edma_handle.flags (*C var*), 236
drivers_MCX_MCXW_MCXW727C._edma_handle.header (*C var*), 236
drivers_MCX_MCXW_MCXW727C._edma_handle.tail (*C var*), 236
drivers_MCX_MCXW_MCXW727C._edma_handle.tcdPool (*C var*), 236
drivers_MCX_MCXW_MCXW727C._edma_handle.tcdSize (*C var*), 236
drivers_MCX_MCXW_MCXW727C._edma_handle.tcdUsed (*C var*), 236
drivers_MCX_MCXW_MCXW727C._edma_handle.userData (*C var*), 236
drivers_MCX_MCXW_MCXW727C._edma_interrupt_enable (*C enum*), 231
drivers_MCX_MCXW_MCXW727C._edma_interrupt_enable.kEDMA_ErrorInterruptEnable (*C enumerator*), 231

drivers_MCX_MCXW_MCXW727C._edma_interrupt_enable.kEDMA_HalfInterruptEnable (*C enumerator*), 231
drivers_MCX_MCXW_MCXW727C._edma_interrupt_enable.kEDMA_MajorInterruptEnable (*C enumerator*), 231
drivers_MCX_MCXW_MCXW727C._edma_minor_offset_config (*C struct*), 234
drivers_MCX_MCXW_MCXW727C._edma_minor_offset_config.enableDestMinorOffset (*C var*), 235
drivers_MCX_MCXW_MCXW727C._edma_minor_offset_config.enableSrcMinorOffset (*C var*), 234
drivers_MCX_MCXW_MCXW727C._edma_minor_offset_config.minorOffset (*C var*), 235
drivers_MCX_MCXW_MCXW727C._edma_modulo (*C enum*), 227
drivers_MCX_MCXW_MCXW727C._edma_modulo.kEDMA_Modulo1Gbytes (*C enumerator*), 229
drivers_MCX_MCXW_MCXW727C._edma_modulo.kEDMA_Modulo1Kbytes (*C enumerator*), 227
drivers_MCX_MCXW_MCXW727C._edma_modulo.kEDMA_Modulo1Mbytes (*C enumerator*), 228
drivers_MCX_MCXW_MCXW727C._edma_modulo.kEDMA_Modulo2bytes (*C enumerator*), 227
drivers_MCX_MCXW_MCXW727C._edma_modulo.kEDMA_Modulo2Gbytes (*C enumerator*), 229
drivers_MCX_MCXW_MCXW727C._edma_modulo.kEDMA_Modulo2Kbytes (*C enumerator*), 228
drivers_MCX_MCXW_MCXW727C._edma_modulo.kEDMA_Modulo2Mbytes (*C enumerator*), 228
drivers_MCX_MCXW_MCXW727C._edma_modulo.kEDMA_Modulo4bytes (*C enumerator*), 227
drivers_MCX_MCXW_MCXW727C._edma_modulo.kEDMA_Modulo4Kbytes (*C enumerator*), 228
drivers_MCX_MCXW_MCXW727C._edma_modulo.kEDMA_Modulo4Mbytes (*C enumerator*), 228
drivers_MCX_MCXW_MCXW727C._edma_modulo.kEDMA_Modulo8bytes (*C enumerator*), 227
drivers_MCX_MCXW_MCXW727C._edma_modulo.kEDMA_Modulo8Kbytes (*C enumerator*), 228
drivers_MCX_MCXW_MCXW727C._edma_modulo.kEDMA_Modulo8Mbytes (*C enumerator*), 228
drivers_MCX_MCXW_MCXW727C._edma_modulo.kEDMA_Modulo16bytes (*C enumerator*), 227
drivers_MCX_MCXW_MCXW727C._edma_modulo.kEDMA_Modulo16Kbytes (*C enumerator*), 228
drivers_MCX_MCXW_MCXW727C._edma_modulo.kEDMA_Modulo16Mbytes (*C enumerator*), 228
drivers_MCX_MCXW_MCXW727C._edma_modulo.kEDMA_Modulo32bytes (*C enumerator*), 227
drivers_MCX_MCXW_MCXW727C._edma_modulo.kEDMA_Modulo32Kbytes (*C enumerator*), 228
drivers_MCX_MCXW_MCXW727C._edma_modulo.kEDMA_Modulo32Mbytes (*C enumerator*), 228
drivers_MCX_MCXW_MCXW727C._edma_modulo.kEDMA_Modulo64bytes (*C enumerator*), 227
drivers_MCX_MCXW_MCXW727C._edma_modulo.kEDMA_Modulo64Kbytes (*C enumerator*), 228
drivers_MCX_MCXW_MCXW727C._edma_modulo.kEDMA_Modulo64Mbytes (*C enumerator*), 228
drivers_MCX_MCXW_MCXW727C._edma_modulo.kEDMA_Modulo128bytes (*C enumerator*), 227
drivers_MCX_MCXW_MCXW727C._edma_modulo.kEDMA_Modulo128Kbytes (*C enumerator*), 228
drivers_MCX_MCXW_MCXW727C._edma_modulo.kEDMA_Modulo128Mbytes (*C enumerator*), 229
drivers_MCX_MCXW_MCXW727C._edma_modulo.kEDMA_Modulo256bytes (*C enumerator*), 227
drivers_MCX_MCXW_MCXW727C._edma_modulo.kEDMA_Modulo256Kbytes (*C enumerator*), 228
drivers_MCX_MCXW_MCXW727C._edma_modulo.kEDMA_Modulo256Mbytes (*C enumerator*), 229
drivers_MCX_MCXW_MCXW727C._edma_modulo.kEDMA_Modulo512bytes (*C enumerator*), 227
drivers_MCX_MCXW_MCXW727C._edma_modulo.kEDMA_Modulo512Kbytes (*C enumerator*), 228
drivers_MCX_MCXW_MCXW727C._edma_modulo.kEDMA_Modulo512Mbytes (*C enumerator*), 229
drivers_MCX_MCXW_MCXW727C._edma_modulo.kEDMA_ModuloDisable (*C enumerator*), 227
drivers_MCX_MCXW_MCXW727C._edma_tcd (*C struct*), 235
drivers_MCX_MCXW_MCXW727C._edma_transfer_config (*C struct*), 233
drivers_MCX_MCXW_MCXW727C._edma_transfer_config.destAddr (*C var*), 234
drivers_MCX_MCXW_MCXW727C._edma_transfer_config.destOffset (*C var*), 234

drivers_MCX_MCXW_MCXW727C._edma_transfer_config.destTransferSize (*C var*), 234
drivers_MCX_MCXW_MCXW727C._edma_transfer_config.majorLoopCounts (*C var*), 234
drivers_MCX_MCXW_MCXW727C._edma_transfer_config.minorLoopBytes (*C var*), 234
drivers_MCX_MCXW_MCXW727C._edma_transfer_config.srcAddr (*C var*), 234
drivers_MCX_MCXW_MCXW727C._edma_transfer_config.srcOffset (*C var*), 234
drivers_MCX_MCXW_MCXW727C._edma_transfer_config.srcTransferSize (*C var*), 234
drivers_MCX_MCXW_MCXW727C._edma_transfer_size (*C enum*), 226
drivers_MCX_MCXW_MCXW727C._edma_transfer_size.kEDMA_TransferSize1Bytes (*C enumerator*), 226
drivers_MCX_MCXW_MCXW727C._edma_transfer_size.kEDMA_TransferSize2Bytes (*C enumerator*), 226
drivers_MCX_MCXW_MCXW727C._edma_transfer_size.kEDMA_TransferSize4Bytes (*C enumerator*), 226
drivers_MCX_MCXW_MCXW727C._edma_transfer_size.kEDMA_TransferSize8Bytes (*C enumerator*), 227
drivers_MCX_MCXW_MCXW727C._edma_transfer_size.kEDMA_TransferSize16Bytes (*C enumerator*), 227
drivers_MCX_MCXW_MCXW727C._edma_transfer_size.kEDMA_TransferSize32Bytes (*C enumerator*), 227
drivers_MCX_MCXW_MCXW727C._edma_transfer_size.kEDMA_TransferSize64Bytes (*C enumerator*), 227
drivers_MCX_MCXW_MCXW727C._edma_transfer_type (*C enum*), 231
drivers_MCX_MCXW_MCXW727C._edma_transfer_type.kEDMA_MemoryToMemory (*C enumerator*), 231
drivers_MCX_MCXW_MCXW727C._edma_transfer_type.kEDMA_MemoryToPeripheral (*C enumerator*), 232
drivers_MCX_MCXW_MCXW727C._edma_transfer_type.kEDMA_PeripheralToMemory (*C enumerator*), 232
drivers_MCX_MCXW_MCXW727C._edma_transfer_type.kEDMA_PeripheralToPeripheral (*C enumerator*), 232
drivers_MCX_MCXW_MCXW727C._ewm_config (*C struct*), 243
drivers_MCX_MCXW_MCXW727C._ewm_config.clockSource (*C var*), 243
drivers_MCX_MCXW_MCXW727C._ewm_config.compareHighValue (*C var*), 243
drivers_MCX_MCXW_MCXW727C._ewm_config.compareLowValue (*C var*), 243
drivers_MCX_MCXW_MCXW727C._ewm_config.enableEwm (*C var*), 243
drivers_MCX_MCXW_MCXW727C._ewm_config.enableEwmInput (*C var*), 243
drivers_MCX_MCXW_MCXW727C._ewm_config.enableInterrupt (*C var*), 243
drivers_MCX_MCXW_MCXW727C._ewm_config.prescaler (*C var*), 243
drivers_MCX_MCXW_MCXW727C._ewm_config.setInputAssertLogic (*C var*), 243
drivers_MCX_MCXW_MCXW727C._ewm_interrupt_enable_t (*C enum*), 242
drivers_MCX_MCXW_MCXW727C._ewm_interrupt_enable_t.kEWM_InterruptEnable (*C enumerator*), 242
drivers_MCX_MCXW_MCXW727C._ewm_lpo_clock_source (*C enum*), 242
drivers_MCX_MCXW_MCXW727C._ewm_lpo_clock_source.kEWM_LpoClockSource0 (*C enumerator*), 242
drivers_MCX_MCXW_MCXW727C._ewm_lpo_clock_source.kEWM_LpoClockSource1 (*C enumerator*), 242
drivers_MCX_MCXW_MCXW727C._ewm_lpo_clock_source.kEWM_LpoClockSource2 (*C enumerator*), 242
drivers_MCX_MCXW_MCXW727C._ewm_lpo_clock_source.kEWM_LpoClockSource3 (*C enumerator*), 242
drivers_MCX_MCXW_MCXW727C._ewm_status_flags_t (*C enum*), 243
drivers_MCX_MCXW_MCXW727C._ewm_status_flags_t.kEWM_RunningFlag (*C enumerator*), 243
drivers_MCX_MCXW_MCXW727C._flash_config (*C struct*), 249
drivers_MCX_MCXW_MCXW727C._flash_driver_api_keys (*C enum*), 244
drivers_MCX_MCXW_MCXW727C._flash_driver_api_keys.kFLASH_ApiEraseKey (*C enumerator*), 244

drivers_MCX_MCXW_MCXW727C._flash_driver_version_constants (*C enum*), 247
drivers_MCX_MCXW_MCXW727C._flash_driver_version_constants.kFLASH_DriverVersionBugfix (*C enumerator*), 247
drivers_MCX_MCXW_MCXW727C._flash_driver_version_constants.kFLASH_DriverVersionMajor (*C enumerator*), 247
drivers_MCX_MCXW_MCXW727C._flash_driver_version_constants.kFLASH_DriverVersionMinor (*C enumerator*), 247
drivers_MCX_MCXW_MCXW727C._flash_driver_version_constants.kFLASH_DriverVersionName (*C enumerator*), 247
drivers_MCX_MCXW_MCXW727C._flash_ifr_desc (*C struct*), 249
drivers_MCX_MCXW_MCXW727C._flash_mem_descriptor (*C struct*), 249
drivers_MCX_MCXW_MCXW727C._flash_mem_descriptor.blockBase (*C var*), 249
drivers_MCX_MCXW_MCXW727C._flash_mem_descriptor.blockCount (*C var*), 249
drivers_MCX_MCXW_MCXW727C._flash_mem_descriptor.totalSize (*C var*), 249
drivers_MCX_MCXW_MCXW727C._flash_property_tag (*C enum*), 247
drivers_MCX_MCXW_MCXW727C._flash_property_tag.kFLASH_PropertyFlexRamBlockBaseAddr (*C enumerator*), 248
drivers_MCX_MCXW_MCXW727C._flash_property_tag.kFLASH_PropertyFlexRamTotalSize (*C enumerator*), 248
drivers_MCX_MCXW_MCXW727C._flash_property_tag.kFLASH_PropertyPflash0AccessSegmentCount (*C enumerator*), 248
drivers_MCX_MCXW_MCXW727C._flash_property_tag.kFLASH_PropertyPflash0AccessSegmentSize (*C enumerator*), 247
drivers_MCX_MCXW_MCXW727C._flash_property_tag.kFLASH_PropertyPflash0BlockBaseAddr (*C enumerator*), 247
drivers_MCX_MCXW_MCXW727C._flash_property_tag.kFLASH_PropertyPflash0BlockCount (*C enumerator*), 247
drivers_MCX_MCXW_MCXW727C._flash_property_tag.kFLASH_PropertyPflash0BlockSize (*C enumerator*), 247
drivers_MCX_MCXW_MCXW727C._flash_property_tag.kFLASH_PropertyPflash0FacSupport (*C enumerator*), 247
drivers_MCX_MCXW_MCXW727C._flash_property_tag.kFLASH_PropertyPflash0SectorSize (*C enumerator*), 247
drivers_MCX_MCXW_MCXW727C._flash_property_tag.kFLASH_PropertyPflash0TotalSize (*C enumerator*), 247
drivers_MCX_MCXW_MCXW727C._flash_property_tag.kFLASH_PropertyPflash1AccessSegmentCount (*C enumerator*), 248
drivers_MCX_MCXW_MCXW727C._flash_property_tag.kFLASH_PropertyPflash1AccessSegmentSize (*C enumerator*), 248
drivers_MCX_MCXW_MCXW727C._flash_property_tag.kFLASH_PropertyPflash1BlockBaseAddr (*C enumerator*), 248
drivers_MCX_MCXW_MCXW727C._flash_property_tag.kFLASH_PropertyPflash1BlockCount (*C enumerator*), 248
drivers_MCX_MCXW_MCXW727C._flash_property_tag.kFLASH_PropertyPflash1BlockSize (*C enumerator*), 248
drivers_MCX_MCXW_MCXW727C._flash_property_tag.kFLASH_PropertyPflash1FacSupport (*C enumerator*), 248
drivers_MCX_MCXW_MCXW727C._flash_property_tag.kFLASH_PropertyPflash1SectorSize (*C enumerator*), 248
drivers_MCX_MCXW_MCXW727C._flash_property_tag.kFLASH_PropertyPflash1TotalSize (*C enumerator*), 248
drivers_MCX_MCXW_MCXW727C._flexcan_byte_error_syndrome (*C enum*), 282
drivers_MCX_MCXW_MCXW727C._flexcan_byte_error_syndrome.kFLEXCAN_AllOneError (*C enumerator*), 283
drivers_MCX_MCXW_MCXW727C._flexcan_byte_error_syndrome.kFLEXCAN_AllZeroError (*C enumerator*), 283
drivers_MCX_MCXW_MCXW727C._flexcan_byte_error_syndrome.kFLEXCAN_DataBits0Error (*C enumerator*), 282

drivers_MCX_MCXW_MCXW727C._flexcan_byte_error_syndrome.kFLEXCAN_DataBits1Error (*C enumerator*), 283
drivers_MCX_MCXW_MCXW727C._flexcan_byte_error_syndrome.kFLEXCAN_DataBits2Error (*C enumerator*), 283
drivers_MCX_MCXW_MCXW727C._flexcan_byte_error_syndrome.kFLEXCAN_DataBits3Error (*C enumerator*), 283
drivers_MCX_MCXW_MCXW727C._flexcan_byte_error_syndrome.kFLEXCAN_DataBits4Error (*C enumerator*), 283
drivers_MCX_MCXW_MCXW727C._flexcan_byte_error_syndrome.kFLEXCAN_DataBits5Error (*C enumerator*), 283
drivers_MCX_MCXW_MCXW727C._flexcan_byte_error_syndrome.kFLEXCAN_DataBits6Error (*C enumerator*), 283
drivers_MCX_MCXW_MCXW727C._flexcan_byte_error_syndrome.kFLEXCAN_DataBits7Error (*C enumerator*), 283
drivers_MCX_MCXW_MCXW727C._flexcan_byte_error_syndrome.kFLEXCAN_NoError (*C enumerator*), 282
drivers_MCX_MCXW_MCXW727C._flexcan_byte_error_syndrome.kFLEXCAN_NonCorrectableErrors (*C enumerator*), 283
drivers_MCX_MCXW_MCXW727C._flexcan_byte_error_syndrome.kFLEXCAN_ParityBits0Error (*C enumerator*), 282
drivers_MCX_MCXW_MCXW727C._flexcan_byte_error_syndrome.kFLEXCAN_ParityBits1Error (*C enumerator*), 282
drivers_MCX_MCXW_MCXW727C._flexcan_byte_error_syndrome.kFLEXCAN_ParityBits2Error (*C enumerator*), 282
drivers_MCX_MCXW_MCXW727C._flexcan_byte_error_syndrome.kFLEXCAN_ParityBits3Error (*C enumerator*), 282
drivers_MCX_MCXW_MCXW727C._flexcan_byte_error_syndrome.kFLEXCAN_ParityBits4Error (*C enumerator*), 282
drivers_MCX_MCXW_MCXW727C._flexcan_capture_point (*C enum*), 273
drivers_MCX_MCXW_MCXW727C._flexcan_capture_point.kFLEXCAN_CANFDFrameRes (*C enumerator*), 274
drivers_MCX_MCXW_MCXW727C._flexcan_capture_point.kFLEXCAN_CANFrameEnd (*C enumerator*), 274
drivers_MCX_MCXW_MCXW727C._flexcan_capture_point.kFLEXCAN_CANFrameID2ndBit (*C enumerator*), 273
drivers_MCX_MCXW_MCXW727C._flexcan_capture_point.kFLEXCAN_CANFrameStart (*C enumerator*), 274
drivers_MCX_MCXW_MCXW727C._flexcan_clock_source (*C enum*), 272
drivers_MCX_MCXW_MCXW727C._flexcan_clock_source.kFLEXCAN_ClkSrc0 (*C enumerator*), 273
drivers_MCX_MCXW_MCXW727C._flexcan_clock_source.kFLEXCAN_ClkSrc1 (*C enumerator*), 273
drivers_MCX_MCXW_MCXW727C._flexcan_clock_source.kFLEXCAN_ClkSrcOsc (*C enumerator*), 272
drivers_MCX_MCXW_MCXW727C._flexcan_clock_source.kFLEXCAN_ClkSrcPeri (*C enumerator*), 273
drivers_MCX_MCXW_MCXW727C._flexcan_config (*C struct*), 291
drivers_MCX_MCXW_MCXW727C._flexcan_config.__unnamed34__ (*C union*), 302
drivers_MCX_MCXW_MCXW727C._flexcan_config.__unnamed34__.__unnamed36__ (*C struct*), 302
drivers_MCX_MCXW_MCXW727C._flexcan_config.__unnamed34__.__unnamed36__.baudRate (*C var*), 302
drivers_MCX_MCXW_MCXW727C._flexcan_config.__unnamed34__.__unnamed36__.baudRateFD (*C var*), 302
drivers_MCX_MCXW_MCXW727C._flexcan_config.__unnamed34__.__unnamed38__ (*C struct*), 302
drivers_MCX_MCXW_MCXW727C._flexcan_config.__unnamed34__.__unnamed38__.bitRate (*C var*), 302

drivers_MCX_MCXW_MCXW727C._flexcan_config.__unnamed34__.__unnamed38__.bitRateFD (*C var*), 302
drivers_MCX_MCXW_MCXW727C._flexcan_config.capturePoint (*C var*), 293
drivers_MCX_MCXW_MCXW727C._flexcan_config.captureTimeBase (*C var*), 293
drivers_MCX_MCXW_MCXW727C._flexcan_config.clkSrc (*C var*), 292
drivers_MCX_MCXW_MCXW727C._flexcan_config.disableBusOffAutoRecovery (*C var*), 293
drivers_MCX_MCXW_MCXW727C._flexcan_config.disableSelfReception (*C var*), 292
drivers_MCX_MCXW_MCXW727C._flexcan_config.enableDoze (*C var*), 292
drivers_MCX_MCXW_MCXW727C._flexcan_config.enableEntireArbitrationCompare (*C var*), 293
drivers_MCX_MCXW_MCXW727C._flexcan_config.enableExternalTimeTick (*C var*), 293
drivers_MCX_MCXW_MCXW727C._flexcan_config.enableIndividMask (*C var*), 292
drivers_MCX_MCXW_MCXW727C._flexcan_config.enableListenOnlyMode (*C var*), 292
drivers_MCX_MCXW_MCXW727C._flexcan_config.enableLoopBack (*C var*), 292
drivers_MCX_MCXW_MCXW727C._flexcan_config.enableMemoryErrorControl (*C var*), 292
drivers_MCX_MCXW_MCXW727C._flexcan_config.enableNonCorrectableErrorEnterFreeze (*C var*), 292
drivers_MCX_MCXW_MCXW727C._flexcan_config.enablePretendedeNetworking (*C var*), 292
drivers_MCX_MCXW_MCXW727C._flexcan_config.enableRemoteRequestFrameStored (*C var*), 292
drivers_MCX_MCXW_MCXW727C._flexcan_config.enableThreeSamplingMode (*C var*), 293
drivers_MCX_MCXW_MCXW727C._flexcan_config.enableTimerSync (*C var*), 292
drivers_MCX_MCXW_MCXW727C._flexcan_config.enableTransceiverDelayMeasure (*C var*), 292
drivers_MCX_MCXW_MCXW727C._flexcan_config.maxMbNum (*C var*), 292
drivers_MCX_MCXW_MCXW727C._flexcan_config.payloadEndianness (*C var*), 293
drivers_MCX_MCXW_MCXW727C._flexcan_config.txArbitrationStartDelay (*C var*), 293
drivers_MCX_MCXW_MCXW727C._flexcan_config.wakeupSrc (*C var*), 292
drivers_MCX_MCXW_MCXW727C._flexcan_edma_handle (*C struct*), 307
drivers_MCX_MCXW_MCXW727C._flexcan_edma_handle.callback (*C var*), 307
drivers_MCX_MCXW_MCXW727C._flexcan_edma_handle.framefd (*C var*), 307
drivers_MCX_MCXW_MCXW727C._flexcan_edma_handle.frameNum (*C var*), 307
drivers_MCX_MCXW_MCXW727C._flexcan_edma_handle.rxFifoEdmaHandle (*C var*), 307
drivers_MCX_MCXW_MCXW727C._flexcan_edma_handle.rxFifoState (*C var*), 307
drivers_MCX_MCXW_MCXW727C._flexcan_edma_handle.userData (*C var*), 307
drivers_MCX_MCXW_MCXW727C._flexcan_efifo_dma_per_read_length (*C enum*), 276
drivers_MCX_MCXW_MCXW727C._flexcan_efifo_dma_per_read_length.kFLEXCAN_1WordPerRead (*C enumerator*), 276
drivers_MCX_MCXW_MCXW727C._flexcan_efifo_dma_per_read_length.kFLEXCAN_2WordPerRead (*C enumerator*), 276
drivers_MCX_MCXW_MCXW727C._flexcan_efifo_dma_per_read_length.kFLEXCAN_3WordPerRead (*C enumerator*), 276
drivers_MCX_MCXW_MCXW727C._flexcan_efifo_dma_per_read_length.kFLEXCAN_4WordPerRead (*C enumerator*), 276
drivers_MCX_MCXW_MCXW727C._flexcan_efifo_dma_per_read_length.kFLEXCAN_5WordPerRead (*C enumerator*), 276
drivers_MCX_MCXW_MCXW727C._flexcan_efifo_dma_per_read_length.kFLEXCAN_6WordPerRead (*C enumerator*), 276
drivers_MCX_MCXW_MCXW727C._flexcan_efifo_dma_per_read_length.kFLEXCAN_7WordPerRead (*C enumerator*), 276
drivers_MCX_MCXW_MCXW727C._flexcan_efifo_dma_per_read_length.kFLEXCAN_8WordPerRead (*C enumerator*), 276
drivers_MCX_MCXW_MCXW727C._flexcan_efifo_dma_per_read_length.kFLEXCAN_9WordPerRead (*C enumerator*), 276
drivers_MCX_MCXW_MCXW727C._flexcan_efifo_dma_per_read_length.kFLEXCAN_10WordPerRead (*C enumerator*), 276
drivers_MCX_MCXW_MCXW727C._flexcan_efifo_dma_per_read_length.kFLEXCAN_11WordPerRead (*C enumerator*), 276
drivers_MCX_MCXW_MCXW727C._flexcan_efifo_dma_per_read_length.kFLEXCAN_12WordPerRead (*C enumerator*), 276
drivers_MCX_MCXW_MCXW727C._flexcan_efifo_dma_per_read_length.kFLEXCAN_13WordPerRead

(*C enumerator*), 276
drivers_MCX_MCXW_MCXW727C._flexcan_efifo_dma_per_read_length.kFLEXCAN_14WordPerRead (*C enumerator*), 276
drivers_MCX_MCXW_MCXW727C._flexcan_efifo_dma_per_read_length.kFLEXCAN_15WordPerRead (*C enumerator*), 276
drivers_MCX_MCXW_MCXW727C._flexcan_efifo_dma_per_read_length.kFLEXCAN_16WordPerRead (*C enumerator*), 277
drivers_MCX_MCXW_MCXW727C._flexcan_efifo_dma_per_read_length.kFLEXCAN_17WordPerRead (*C enumerator*), 277
drivers_MCX_MCXW_MCXW727C._flexcan_efifo_dma_per_read_length.kFLEXCAN_18WordPerRead (*C enumerator*), 277
drivers_MCX_MCXW_MCXW727C._flexcan_efifo_dma_per_read_length.kFLEXCAN_19WordPerRead (*C enumerator*), 277
drivers_MCX_MCXW_MCXW727C._flexcan_efifo_dma_per_read_length.kFLEXCAN_20WordPerRead (*C enumerator*), 277
drivers_MCX_MCXW_MCXW727C._flexcan_endianness (*C enum*), 273
drivers_MCX_MCXW_MCXW727C._flexcan_endianness.kFLEXCAN_bigEndian (*C enumerator*), 273
drivers_MCX_MCXW_MCXW727C._flexcan_endianness.kFLEXCAN_littleEndian (*C enumerator*), 273
drivers_MCX_MCXW_MCXW727C._flexcan_enhanced_rx_fifo_config (*C struct*), 296
drivers_MCX_MCXW_MCXW727C._flexcan_enhanced_rx_fifo_config.dmaPerReadLength (*C var*), 296
drivers_MCX_MCXW_MCXW727C._flexcan_enhanced_rx_fifo_config.extendIdFilterNum (*C var*), 296
drivers_MCX_MCXW_MCXW727C._flexcan_enhanced_rx_fifo_config.fifoWatermark (*C var*), 296
drivers_MCX_MCXW_MCXW727C._flexcan_enhanced_rx_fifo_config.idFilterPairNum (*C var*), 296
drivers_MCX_MCXW_MCXW727C._flexcan_enhanced_rx_fifo_config.idFilterTable (*C var*), 296
drivers_MCX_MCXW_MCXW727C._flexcan_enhanced_rx_fifo_config.priority (*C var*), 296
drivers_MCX_MCXW_MCXW727C._flexcan_enhanced_rx_fifo_ext_id_filter (*C struct*), 295
drivers_MCX_MCXW_MCXW727C._flexcan_enhanced_rx_fifo_ext_id_filter.filterType (*C var*), 296
drivers_MCX_MCXW_MCXW727C._flexcan_enhanced_rx_fifo_ext_id_filter.rtr1 (*C var*), 296
drivers_MCX_MCXW_MCXW727C._flexcan_enhanced_rx_fifo_ext_id_filter.rtr2 (*C var*), 296
drivers_MCX_MCXW_MCXW727C._flexcan_enhanced_rx_fifo_ext_id_filter.std1 (*C var*), 296
drivers_MCX_MCXW_MCXW727C._flexcan_enhanced_rx_fifo_ext_id_filter.std2 (*C var*), 296
drivers_MCX_MCXW_MCXW727C._flexcan_enhanced_rx_fifo_std_id_filter (*C struct*), 295
drivers_MCX_MCXW_MCXW727C._flexcan_enhanced_rx_fifo_std_id_filter.filterType (*C var*), 295
drivers_MCX_MCXW_MCXW727C._flexcan_enhanced_rx_fifo_std_id_filter.rtr1 (*C var*), 295
drivers_MCX_MCXW_MCXW727C._flexcan_enhanced_rx_fifo_std_id_filter.rtr2 (*C var*), 295
drivers_MCX_MCXW_MCXW727C._flexcan_enhanced_rx_fifo_std_id_filter.std1 (*C var*), 295
drivers_MCX_MCXW_MCXW727C._flexcan_enhanced_rx_fifo_std_id_filter.std2 (*C var*), 295
drivers_MCX_MCXW_MCXW727C._flexcan_error_flags (*C enum*), 280
drivers_MCX_MCXW_MCXW727C._flexcan_error_flags.kFLEXCAN_AckError (*C enumerator*), 281
drivers_MCX_MCXW_MCXW727C._flexcan_error_flags.kFLEXCAN_Bit0Error (*C enumerator*), 281
drivers_MCX_MCXW_MCXW727C._flexcan_error_flags.kFLEXCAN_Bit1Error (*C enumerator*), 281
drivers_MCX_MCXW_MCXW727C._flexcan_error_flags.kFLEXCAN_CrcError (*C enumerator*), 281
drivers_MCX_MCXW_MCXW727C._flexcan_error_flags.kFLEXCAN_FDBit0Error (*C enumerator*), 280
drivers_MCX_MCXW_MCXW727C._flexcan_error_flags.kFLEXCAN_FDBit1Error (*C enumerator*), 280
drivers_MCX_MCXW_MCXW727C._flexcan_error_flags.kFLEXCAN_FDCrcError (*C enumera-*

tor), 280
drivers_MCX_MCXW_MCXW727C._flexcan_error_flags.kFLEXCAN_FDFormError (*C enumerator*), 280
drivers_MCX_MCXW_MCXW727C._flexcan_error_flags.kFLEXCAN_FDStuffingError (*C enumerator*), 280
drivers_MCX_MCXW_MCXW727C._flexcan_error_flags.kFLEXCAN_FormError (*C enumerator*), 281
drivers_MCX_MCXW_MCXW727C._flexcan_error_flags.kFLEXCAN_RxErrorWarningFlag (*C enumerator*), 281
drivers_MCX_MCXW_MCXW727C._flexcan_error_flags.kFLEXCAN_StuffingError (*C enumerator*), 281
drivers_MCX_MCXW_MCXW727C._flexcan_error_flags.kFLEXCAN_TxErrorWarningFlag (*C enumerator*), 281
drivers_MCX_MCXW_MCXW727C._flexcan_fd_frame (*C struct*), 290
drivers_MCX_MCXW_MCXW727C._flexcan_fd_frame.__unnamed24__ (*C struct*), 300
drivers_MCX_MCXW_MCXW727C._flexcan_fd_frame.__unnamed24__.brs (*C var*), 300
drivers_MCX_MCXW_MCXW727C._flexcan_fd_frame.__unnamed24__.edl (*C var*), 300
drivers_MCX_MCXW_MCXW727C._flexcan_fd_frame.__unnamed24__.esi (*C var*), 300
drivers_MCX_MCXW_MCXW727C._flexcan_fd_frame.__unnamed24__.format (*C var*), 300
drivers_MCX_MCXW_MCXW727C._flexcan_fd_frame.__unnamed24__.length (*C var*), 300
drivers_MCX_MCXW_MCXW727C._flexcan_fd_frame.__unnamed24__.srr (*C var*), 300
drivers_MCX_MCXW_MCXW727C._flexcan_fd_frame.__unnamed24__.timestamp (*C var*), 300
drivers_MCX_MCXW_MCXW727C._flexcan_fd_frame.__unnamed24__.type (*C var*), 300
drivers_MCX_MCXW_MCXW727C._flexcan_fd_frame.__unnamed26__ (*C struct*), 300
drivers_MCX_MCXW_MCXW727C._flexcan_fd_frame.__unnamed26__.__pad0__ (*C var*), 301
drivers_MCX_MCXW_MCXW727C._flexcan_fd_frame.__unnamed26__.id (*C var*), 301
drivers_MCX_MCXW_MCXW727C._flexcan_fd_frame.__unnamed28__ (*C union*), 301
drivers_MCX_MCXW_MCXW727C._flexcan_fd_frame.__unnamed28__.__unnamed30__ (*C struct*), 301
drivers_MCX_MCXW_MCXW727C._flexcan_fd_frame.__unnamed28__.__unnamed30__.dataWord (*C var*), 301
drivers_MCX_MCXW_MCXW727C._flexcan_fd_frame.__unnamed28__.__unnamed32__ (*C struct*), 301
drivers_MCX_MCXW_MCXW727C._flexcan_fd_frame.__unnamed28__.__unnamed32__.dataByte0 (*C var*), 301
drivers_MCX_MCXW_MCXW727C._flexcan_fd_frame.__unnamed28__.__unnamed32__.dataByte1 (*C var*), 301
drivers_MCX_MCXW_MCXW727C._flexcan_fd_frame.__unnamed28__.__unnamed32__.dataByte2 (*C var*), 301
drivers_MCX_MCXW_MCXW727C._flexcan_fd_frame.__unnamed28__.__unnamed32__.dataByte3 (*C var*), 301
drivers_MCX_MCXW_MCXW727C._flexcan_fd_frame.__unnamed28__.__unnamed32__.dataByte4 (*C var*), 302
drivers_MCX_MCXW_MCXW727C._flexcan_fd_frame.__unnamed28__.__unnamed32__.dataByte5 (*C var*), 302
drivers_MCX_MCXW_MCXW727C._flexcan_fd_frame.__unnamed28__.__unnamed32__.dataByte6 (*C var*), 301
drivers_MCX_MCXW_MCXW727C._flexcan_fd_frame.__unnamed28__.__unnamed32__.dataByte7 (*C var*), 301
drivers_MCX_MCXW_MCXW727C._flexcan_fd_frame_length (*C enum*), 274
drivers_MCX_MCXW_MCXW727C._flexcan_fd_frame_length.kFLEXCAN_0BperFrame (*C enumerator*), 275
drivers_MCX_MCXW_MCXW727C._flexcan_fd_frame_length.kFLEXCAN_1BperFrame (*C enumerator*), 275
drivers_MCX_MCXW_MCXW727C._flexcan_fd_frame_length.kFLEXCAN_2BperFrame (*C enumerator*), 275
drivers_MCX_MCXW_MCXW727C._flexcan_fd_frame_length.kFLEXCAN_3BperFrame (*C enumerator*), 275

drivers_MCX_MCXW_MCXW727C._flexcan_fd_frame_length.kFLEXCAN_4BperFrame (*C enumerator*), 275
drivers_MCX_MCXW_MCXW727C._flexcan_fd_frame_length.kFLEXCAN_5BperFrame (*C enumerator*), 275
drivers_MCX_MCXW_MCXW727C._flexcan_fd_frame_length.kFLEXCAN_6BperFrame (*C enumerator*), 275
drivers_MCX_MCXW_MCXW727C._flexcan_fd_frame_length.kFLEXCAN_7BperFrame (*C enumerator*), 275
drivers_MCX_MCXW_MCXW727C._flexcan_fd_frame_length.kFLEXCAN_8BperFrame (*C enumerator*), 275
drivers_MCX_MCXW_MCXW727C._flexcan_fd_frame_length.kFLEXCAN_12BperFrame (*C enumerator*), 275
drivers_MCX_MCXW_MCXW727C._flexcan_fd_frame_length.kFLEXCAN_16BperFrame (*C enumerator*), 275
drivers_MCX_MCXW_MCXW727C._flexcan_fd_frame_length.kFLEXCAN_20BperFrame (*C enumerator*), 275
drivers_MCX_MCXW_MCXW727C._flexcan_fd_frame_length.kFLEXCAN_24BperFrame (*C enumerator*), 275
drivers_MCX_MCXW_MCXW727C._flexcan_fd_frame_length.kFLEXCAN_32BperFrame (*C enumerator*), 275
drivers_MCX_MCXW_MCXW727C._flexcan_fd_frame_length.kFLEXCAN_48BperFrame (*C enumerator*), 275
drivers_MCX_MCXW_MCXW727C._flexcan_fd_frame_length.kFLEXCAN_64BperFrame (*C enumerator*), 275
drivers_MCX_MCXW_MCXW727C._flexcan_fd_frame.hrtimestamp (*C var*), 291
drivers_MCX_MCXW_MCXW727C._flexcan_fd_frame.idhit (*C var*), 290
drivers_MCX_MCXW_MCXW727C._flexcan_fifo_transfer (*C struct*), 297
drivers_MCX_MCXW_MCXW727C._flexcan_fifo_transfer.frame (*C var*), 297
drivers_MCX_MCXW_MCXW727C._flexcan_fifo_transfer.framefd (*C var*), 297
drivers_MCX_MCXW_MCXW727C._flexcan_fifo_transfer.frameNum (*C var*), 297
drivers_MCX_MCXW_MCXW727C._flexcan_flags (*C enum*), 278
drivers_MCX_MCXW_MCXW727C._flexcan_flags.kFLEXCAN_AllMemoryErrorFlag (*C enumerator*), 280
drivers_MCX_MCXW_MCXW727C._flexcan_flags.kFLEXCAN_AllMemoryErrorIntFlag (*C enumerator*), 280
drivers_MCX_MCXW_MCXW727C._flexcan_flags.kFLEXCAN_BusoffDoneIntFlag (*C enumerator*), 278
drivers_MCX_MCXW_MCXW727C._flexcan_flags.kFLEXCAN_BusOffIntFlag (*C enumerator*), 279
drivers_MCX_MCXW_MCXW727C._flexcan_flags.kFLEXCAN_CorrectableErrorIntFlag (*C enumerator*), 280
drivers_MCX_MCXW_MCXW727C._flexcan_flags.kFLEXCAN_CorrectableErrorOverrunFlag (*C enumerator*), 280
drivers_MCX_MCXW_MCXW727C._flexcan_flags.kFLEXCAN_ErrorFlag (*C enumerator*), 279
drivers_MCX_MCXW_MCXW727C._flexcan_flags.kFLEXCAN_ErrorIntFlag (*C enumerator*), 279
drivers_MCX_MCXW_MCXW727C._flexcan_flags.kFLEXCAN_ErrorOverrunFlag (*C enumerator*), 278
drivers_MCX_MCXW_MCXW727C._flexcan_flags.kFLEXCAN_ERxFifoDataAvlIntFlag (*C enumerator*), 279
drivers_MCX_MCXW_MCXW727C._flexcan_flags.kFLEXCAN_ERxFifoEmptyFlag (*C enumerator*), 279
drivers_MCX_MCXW_MCXW727C._flexcan_flags.kFLEXCAN_ERxFifoFullFlag (*C enumerator*), 280
drivers_MCX_MCXW_MCXW727C._flexcan_flags.kFLEXCAN_ERxFifoOverflowIntFlag (*C enumerator*), 279
drivers_MCX_MCXW_MCXW727C._flexcan_flags.kFLEXCAN_ERxFifoUnderflowIntFlag (*C enumerator*), 279
drivers_MCX_MCXW_MCXW727C._flexcan_flags.kFLEXCAN_ERxFifoWatermarkIntFlag (*C enu-*

merator), 279
drivers_MCX_MCXW_MCXW727C._flexcan_flags.kFLEXCAN_FaultConfinementFlag (*C enumerator*), 279
drivers_MCX_MCXW_MCXW727C._flexcan_flags.kFLEXCAN_FDErrorIntFlag (*C enumerator*), 278
drivers_MCX_MCXW_MCXW727C._flexcan_flags.kFLEXCAN_FlexCanAccessNonCorrectableErrorIntFlag (*C enumerator*), 280
drivers_MCX_MCXW_MCXW727C._flexcan_flags.kFLEXCAN_FlexCanAccessNonCorrectableErrorOverrunFlag (*C enumerator*), 280
drivers_MCX_MCXW_MCXW727C._flexcan_flags.kFLEXCAN_HostAccessNonCorrectableErrorIntFlag (*C enumerator*), 280
drivers_MCX_MCXW_MCXW727C._flexcan_flags.kFLEXCAN_HostAccessNonCorrectableErrorOverrunFlag (*C enumerator*), 280
drivers_MCX_MCXW_MCXW727C._flexcan_flags.kFLEXCAN_IdleFlag (*C enumerator*), 279
drivers_MCX_MCXW_MCXW727C._flexcan_flags.kFLEXCAN_PNMatchIntFlag (*C enumerator*), 279
drivers_MCX_MCXW_MCXW727C._flexcan_flags.kFLEXCAN_PNTimeoutIntFlag (*C enumerator*), 279
drivers_MCX_MCXW_MCXW727C._flexcan_flags.kFLEXCAN_ReceivingFlag (*C enumerator*), 279
drivers_MCX_MCXW_MCXW727C._flexcan_flags.kFLEXCAN_RxWarningIntFlag (*C enumerator*), 279
drivers_MCX_MCXW_MCXW727C._flexcan_flags.kFLEXCAN_SynchFlag (*C enumerator*), 279
drivers_MCX_MCXW_MCXW727C._flexcan_flags.kFLEXCAN_TransmittingFlag (*C enumerator*), 279
drivers_MCX_MCXW_MCXW727C._flexcan_flags.kFLEXCAN_TxWarningIntFlag (*C enumerator*), 279
drivers_MCX_MCXW_MCXW727C._flexcan_frame (*C struct*), 290
drivers_MCX_MCXW_MCXW727C._flexcan_frame.__unnamed14__ (*C struct*), 298
drivers_MCX_MCXW_MCXW727C._flexcan_frame.__unnamed14__.__pad0__ (*C var*), 298
drivers_MCX_MCXW_MCXW727C._flexcan_frame.__unnamed14__.format (*C var*), 298
drivers_MCX_MCXW_MCXW727C._flexcan_frame.__unnamed14__.idhit (*C var*), 299
drivers_MCX_MCXW_MCXW727C._flexcan_frame.__unnamed14__.length (*C var*), 298
drivers_MCX_MCXW_MCXW727C._flexcan_frame.__unnamed14__.timestamp (*C var*), 298
drivers_MCX_MCXW_MCXW727C._flexcan_frame.__unnamed14__.type (*C var*), 298
drivers_MCX_MCXW_MCXW727C._flexcan_frame.__unnamed16__ (*C struct*), 299
drivers_MCX_MCXW_MCXW727C._flexcan_frame.__unnamed16__.__pad0__ (*C var*), 299
drivers_MCX_MCXW_MCXW727C._flexcan_frame.__unnamed16__.id (*C var*), 299
drivers_MCX_MCXW_MCXW727C._flexcan_frame.__unnamed18__ (*C union*), 299
drivers_MCX_MCXW_MCXW727C._flexcan_frame.__unnamed18__.__unnamed20__ (*C struct*), 299
drivers_MCX_MCXW_MCXW727C._flexcan_frame.__unnamed18__.__unnamed20__.dataWord0 (*C var*), 299
drivers_MCX_MCXW_MCXW727C._flexcan_frame.__unnamed18__.__unnamed20__.dataWord1 (*C var*), 299
drivers_MCX_MCXW_MCXW727C._flexcan_frame.__unnamed18__.__unnamed22__ (*C struct*), 299
drivers_MCX_MCXW_MCXW727C._flexcan_frame.__unnamed18__.__unnamed22__.dataByte0 (*C var*), 300
drivers_MCX_MCXW_MCXW727C._flexcan_frame.__unnamed18__.__unnamed22__.dataByte1 (*C var*), 299
drivers_MCX_MCXW_MCXW727C._flexcan_frame.__unnamed18__.__unnamed22__.dataByte2 (*C var*), 299
drivers_MCX_MCXW_MCXW727C._flexcan_frame.__unnamed18__.__unnamed22__.dataByte3 (*C var*), 299
drivers_MCX_MCXW_MCXW727C._flexcan_frame.__unnamed18__.__unnamed22__.dataByte4 (*C var*), 300
drivers_MCX_MCXW_MCXW727C._flexcan_frame.__unnamed18__.__unnamed22__.dataByte5

(*C var*), 300
drivers_MCX_MCXW_MCXW727C._flexcan_frame.___unnamed18___.___unnamed22___.dataByte6 (*C var*), 300
drivers_MCX_MCXW_MCXW727C._flexcan_frame.___unnamed18___.___unnamed22___.dataByte7 (*C var*), 300
drivers_MCX_MCXW_MCXW727C._flexcan_frame_format (*C enum*), 272
drivers_MCX_MCXW_MCXW727C._flexcan_frame_format.kFLEXCAN_FrameFormatExtend (*C enumerator*), 272
drivers_MCX_MCXW_MCXW727C._flexcan_frame_format.kFLEXCAN_FrameFormatStandard (*C enumerator*), 272
drivers_MCX_MCXW_MCXW727C._flexcan_frame_type (*C enum*), 272
drivers_MCX_MCXW_MCXW727C._flexcan_frame_type.kFLEXCAN_FrameTypeData (*C enumerator*), 272
drivers_MCX_MCXW_MCXW727C._flexcan_frame_type.kFLEXCAN_FrameTypeRemote (*C enumerator*), 272
drivers_MCX_MCXW_MCXW727C._flexcan_handle (*C struct*), 297
drivers_MCX_MCXW_MCXW727C._flexcan_handle.callback (*C var*), 297
drivers_MCX_MCXW_MCXW727C._flexcan_handle.mbFDFrameBuf (*C var*), 297
drivers_MCX_MCXW_MCXW727C._flexcan_handle.mbFrameBuf (*C var*), 297
drivers_MCX_MCXW_MCXW727C._flexcan_handle.mbState (*C var*), 298
drivers_MCX_MCXW_MCXW727C._flexcan_handle.rxFifoFDFrameBuf (*C var*), 298
drivers_MCX_MCXW_MCXW727C._flexcan_handle.rxFifoFrameBuf (*C var*), 297
drivers_MCX_MCXW_MCXW727C._flexcan_handle.rxFifoFrameNum (*C var*), 298
drivers_MCX_MCXW_MCXW727C._flexcan_handle.rxFifoState (*C var*), 298
drivers_MCX_MCXW_MCXW727C._flexcan_handle.rxFifoTransferTotalNum (*C var*), 298
drivers_MCX_MCXW_MCXW727C._flexcan_handle.timestamp (*C var*), 298
drivers_MCX_MCXW_MCXW727C._flexcan_handle.userData (*C var*), 297
drivers_MCX_MCXW_MCXW727C._flexcan_interrupt_enable (*C enum*), 277
drivers_MCX_MCXW_MCXW727C._flexcan_interrupt_enable.kFLEXCAN_BusOffInterruptEnable (*C enumerator*), 277
drivers_MCX_MCXW_MCXW727C._flexcan_interrupt_enable.kFLEXCAN_ErrorInterruptEnable (*C enumerator*), 277
drivers_MCX_MCXW_MCXW727C._flexcan_interrupt_enable.kFLEXCAN_ERxFifoDataAvlInterruptEnable (*C enumerator*), 278
drivers_MCX_MCXW_MCXW727C._flexcan_interrupt_enable.kFLEXCAN_ERxFifoOverflowInterruptEnable (*C enumerator*), 278
drivers_MCX_MCXW_MCXW727C._flexcan_interrupt_enable.kFLEXCAN_ERxFifoUnderflowInterruptEnable (*C enumerator*), 278
drivers_MCX_MCXW_MCXW727C._flexcan_interrupt_enable.kFLEXCAN_ERxFifoWatermarkInterruptEnable (*C enumerator*), 278
drivers_MCX_MCXW_MCXW727C._flexcan_interrupt_enable.kFLEXCAN_FDErrorInterruptEnable (*C enumerator*), 278
drivers_MCX_MCXW_MCXW727C._flexcan_interrupt_enable.kFLEXCAN_FlexCanAccessNCErrorInterruptEnable (*C enumerator*), 278
drivers_MCX_MCXW_MCXW727C._flexcan_interrupt_enable.kFLEXCAN_HostAccessNCErrorInterruptEnable (*C enumerator*), 278
drivers_MCX_MCXW_MCXW727C._flexcan_interrupt_enable.kFLEXCAN_HostOrFlexCanCErrorInterruptEnable (*C enumerator*), 278
drivers_MCX_MCXW_MCXW727C._flexcan_interrupt_enable.kFLEXCAN_PNMatchWakeUpInterruptEnable (*C enumerator*), 278
drivers_MCX_MCXW_MCXW727C._flexcan_interrupt_enable.kFLEXCAN_PNTimeoutWakeUpInterruptEnable (*C enumerator*), 278
drivers_MCX_MCXW_MCXW727C._flexcan_interrupt_enable.kFLEXCAN_RxWarningInterruptEnable (*C enumerator*), 277
drivers_MCX_MCXW_MCXW727C._flexcan_interrupt_enable.kFLEXCAN_TxWarningInterruptEnable (*C enumerator*), 277
drivers_MCX_MCXW_MCXW727C._flexcan_mb_size (*C enum*), 274
drivers_MCX_MCXW_MCXW727C._flexcan_mb_size.kFLEXCAN_8BperMB (*C enumerator*), 274

drivers_MCX_MCXW_MCXW727C._flexcan_mb_size.kFLEXCAN_16BperMB (*C enumerator*), 274
drivers_MCX_MCXW_MCXW727C._flexcan_mb_size.kFLEXCAN_32BperMB (*C enumerator*), 274
drivers_MCX_MCXW_MCXW727C._flexcan_mb_size.kFLEXCAN_64BperMB (*C enumerator*), 274
drivers_MCX_MCXW_MCXW727C._flexcan_MB_timestamp_base (*C enum*), 273
drivers_MCX_MCXW_MCXW727C._flexcan_MB_timestamp_base.kFLEXCAN_CANTimer (*C enumerator*), 273
drivers_MCX_MCXW_MCXW727C._flexcan_MB_timestamp_base.kFLEXCAN_Lower16bitsHRTimer (*C enumerator*), 273
drivers_MCX_MCXW_MCXW727C._flexcan_MB_timestamp_base.kFLEXCAN_Upper16bitsHRTimer (*C enumerator*), 273
drivers_MCX_MCXW_MCXW727C._flexcan_mb_transfer (*C struct*), 297
drivers_MCX_MCXW_MCXW727C._flexcan_mb_transfer.frame (*C var*), 297
drivers_MCX_MCXW_MCXW727C._flexcan_mb_transfer.mbIdx (*C var*), 297
drivers_MCX_MCXW_MCXW727C._flexcan_memory_access_type (*C enum*), 282
drivers_MCX_MCXW_MCXW727C._flexcan_memory_access_type.kFLEXCAN_MoveInAccess (*C enumerator*), 282
drivers_MCX_MCXW_MCXW727C._flexcan_memory_access_type.kFLEXCAN_MoveOutFlexCanAccess (*C enumerator*), 282
drivers_MCX_MCXW_MCXW727C._flexcan_memory_access_type.kFLEXCAN_MoveOutHostAccess (*C enumerator*), 282
drivers_MCX_MCXW_MCXW727C._flexcan_memory_access_type.kFLEXCAN_RxMatchingAccess (*C enumerator*), 282
drivers_MCX_MCXW_MCXW727C._flexcan_memory_access_type.kFLEXCAN_TxArbitrationAccess (*C enumerator*), 282
drivers_MCX_MCXW_MCXW727C._flexcan_memory_error_report_status (*C struct*), 290
drivers_MCX_MCXW_MCXW727C._flexcan_memory_error_report_status.accessAddress (*C var*), 290
drivers_MCX_MCXW_MCXW727C._flexcan_memory_error_report_status.accessType (*C var*), 290
drivers_MCX_MCXW_MCXW727C._flexcan_memory_error_report_status.byteStatus (*C struct*), 298
drivers_MCX_MCXW_MCXW727C._flexcan_memory_error_report_status.byteStatus.byteIsRead (*C var*), 298
drivers_MCX_MCXW_MCXW727C._flexcan_memory_error_report_status.errorData (*C var*), 290
drivers_MCX_MCXW_MCXW727C._flexcan_memory_error_report_status.errorType (*C var*), 290
drivers_MCX_MCXW_MCXW727C._flexcan_memory_error_type (*C enum*), 281
drivers_MCX_MCXW_MCXW727C._flexcan_memory_error_type.kFLEXCAN_CorrectableError (*C enumerator*), 281
drivers_MCX_MCXW_MCXW727C._flexcan_memory_error_type.kFLEXCAN_NonCorrectableError (*C enumerator*), 282
drivers_MCX_MCXW_MCXW727C._flexcan_pn_config (*C struct*), 294
drivers_MCX_MCXW_MCXW727C._flexcan_pn_config.__unnamed40__ (*C union*), 302
drivers_MCX_MCXW_MCXW727C._flexcan_pn_config.__unnamed40__.__unnamed44__ (*C struct*), 303
drivers_MCX_MCXW_MCXW727C._flexcan_pn_config.__unnamed40__.__unnamed44__.lowerWord0 (*C var*), 303
drivers_MCX_MCXW_MCXW727C._flexcan_pn_config.__unnamed40__.__unnamed44__.lowerWord1 (*C var*), 303
drivers_MCX_MCXW_MCXW727C._flexcan_pn_config.__unnamed40__.__unnamed46__ (*C struct*), 303
drivers_MCX_MCXW_MCXW727C._flexcan_pn_config.__unnamed40__.__unnamed46__.lowerByte0 (*C var*), 303
drivers_MCX_MCXW_MCXW727C._flexcan_pn_config.__unnamed40__.__unnamed46__.lowerByte1 (*C var*), 303
drivers_MCX_MCXW_MCXW727C._flexcan_pn_config.__unnamed40__.__unnamed46__.lowerByte2 (*C var*), 303

drivers_MCX_MCXW_MCXW727C._flexcan_pn_config.__unnamed40__.__unnamed46__.lowerByte3 (*C var*), 303
drivers_MCX_MCXW_MCXW727C._flexcan_pn_config.__unnamed40__.__unnamed46__.lowerByte4 (*C var*), 303
drivers_MCX_MCXW_MCXW727C._flexcan_pn_config.__unnamed40__.__unnamed46__.lowerByte5 (*C var*), 303
drivers_MCX_MCXW_MCXW727C._flexcan_pn_config.__unnamed40__.__unnamed46__.lowerByte6 (*C var*), 303
drivers_MCX_MCXW_MCXW727C._flexcan_pn_config.__unnamed40__.__unnamed46__.lowerByte7 (*C var*), 303
drivers_MCX_MCXW_MCXW727C._flexcan_pn_config.__unnamed42__ (*C union*), 303
drivers_MCX_MCXW_MCXW727C._flexcan_pn_config.__unnamed42__.__unnamed48__ (*C struct*), 304
drivers_MCX_MCXW_MCXW727C._flexcan_pn_config.__unnamed42__.__unnamed48__.upperWord0 (*C var*), 304
drivers_MCX_MCXW_MCXW727C._flexcan_pn_config.__unnamed42__.__unnamed48__.upperWord1 (*C var*), 304
drivers_MCX_MCXW_MCXW727C._flexcan_pn_config.__unnamed42__.__unnamed50__ (*C struct*), 304
drivers_MCX_MCXW_MCXW727C._flexcan_pn_config.__unnamed42__.__unnamed50__.upperByte0 (*C var*), 304
drivers_MCX_MCXW_MCXW727C._flexcan_pn_config.__unnamed42__.__unnamed50__.upperByte1 (*C var*), 304
drivers_MCX_MCXW_MCXW727C._flexcan_pn_config.__unnamed42__.__unnamed50__.upperByte2 (*C var*), 304
drivers_MCX_MCXW_MCXW727C._flexcan_pn_config.__unnamed42__.__unnamed50__.upperByte3 (*C var*), 304
drivers_MCX_MCXW_MCXW727C._flexcan_pn_config.__unnamed42__.__unnamed50__.upperByte4 (*C var*), 304
drivers_MCX_MCXW_MCXW727C._flexcan_pn_config.__unnamed42__.__unnamed50__.upperByte5 (*C var*), 304
drivers_MCX_MCXW_MCXW727C._flexcan_pn_config.__unnamed42__.__unnamed50__.upperByte6 (*C var*), 304
drivers_MCX_MCXW_MCXW727C._flexcan_pn_config.__unnamed42__.__unnamed50__.upperByte7 (*C var*), 304
drivers_MCX_MCXW_MCXW727C._flexcan_pn_config.dataMatchMode (*C var*), 294
drivers_MCX_MCXW_MCXW727C._flexcan_pn_config.enableMatch (*C var*), 294
drivers_MCX_MCXW_MCXW727C._flexcan_pn_config.enableTimeout (*C var*), 294
drivers_MCX_MCXW_MCXW727C._flexcan_pn_config.idLower (*C var*), 294
drivers_MCX_MCXW_MCXW727C._flexcan_pn_config.idMatchMode (*C var*), 294
drivers_MCX_MCXW_MCXW727C._flexcan_pn_config.idUpper (*C var*), 294
drivers_MCX_MCXW_MCXW727C._flexcan_pn_config.lengthLower (*C var*), 294
drivers_MCX_MCXW_MCXW727C._flexcan_pn_config.lengthUpper (*C var*), 294
drivers_MCX_MCXW_MCXW727C._flexcan_pn_config.matchNum (*C var*), 294
drivers_MCX_MCXW_MCXW727C._flexcan_pn_config.matchSrc (*C var*), 294
drivers_MCX_MCXW_MCXW727C._flexcan_pn_config.timeoutValue (*C var*), 294
drivers_MCX_MCXW_MCXW727C._flexcan_pn_match_mode (*C enum*), 283
drivers_MCX_MCXW_MCXW727C._flexcan_pn_match_mode.kFLEXCAN_PNMatModeEqual (*C enumerator*), 283
drivers_MCX_MCXW_MCXW727C._flexcan_pn_match_mode.kFLEXCAN_PNMatModeGreater (*C enumerator*), 284
drivers_MCX_MCXW_MCXW727C._flexcan_pn_match_mode.kFLEXCAN_PNMatModeRange (*C enumerator*), 284
drivers_MCX_MCXW_MCXW727C._flexcan_pn_match_mode.kFLEXCAN_PNMatModeSmaller (*C enumerator*), 284
drivers_MCX_MCXW_MCXW727C._flexcan_pn_match_source (*C enum*), 283
drivers_MCX_MCXW_MCXW727C._flexcan_pn_match_source.kFLEXCAN_PNMatSrcID (*C enumerator*), 283

drivers_MCX_MCXW_MCXW727C._flexcan_pn_match_source.kFLEXCAN_PNMatSrcIDAndData (*C enumerator*), 283
drivers_MCX_MCXW_MCXW727C._flexcan_rx_fifo_config (*C struct*), 295
drivers_MCX_MCXW_MCXW727C._flexcan_rx_fifo_config.idFilterNum (*C var*), 295
drivers_MCX_MCXW_MCXW727C._flexcan_rx_fifo_config.idFilterTable (*C var*), 295
drivers_MCX_MCXW_MCXW727C._flexcan_rx_fifo_config.idFilterType (*C var*), 295
drivers_MCX_MCXW_MCXW727C._flexcan_rx_fifo_config.priority (*C var*), 295
drivers_MCX_MCXW_MCXW727C._flexcan_rx_fifo_filter_type (*C enum*), 274
drivers_MCX_MCXW_MCXW727C._flexcan_rx_fifo_filter_type.kFLEXCAN_RxFifoFilterTypeA (*C enumerator*), 274
drivers_MCX_MCXW_MCXW727C._flexcan_rx_fifo_filter_type.kFLEXCAN_RxFifoFilterTypeB (*C enumerator*), 274
drivers_MCX_MCXW_MCXW727C._flexcan_rx_fifo_filter_type.kFLEXCAN_RxFifoFilterTypeC (*C enumerator*), 274
drivers_MCX_MCXW_MCXW727C._flexcan_rx_fifo_filter_type.kFLEXCAN_RxFifoFilterTypeD (*C enumerator*), 274
drivers_MCX_MCXW_MCXW727C._flexcan_rx_fifo_priority (*C enum*), 277
drivers_MCX_MCXW_MCXW727C._flexcan_rx_fifo_priority.kFLEXCAN_RxFifoPrioHigh (*C enumerator*), 277
drivers_MCX_MCXW_MCXW727C._flexcan_rx_fifo_priority.kFLEXCAN_RxFifoPrioLow (*C enumerator*), 277
drivers_MCX_MCXW_MCXW727C._flexcan_rx_mb_config (*C struct*), 293
drivers_MCX_MCXW_MCXW727C._flexcan_rx_mb_config.format (*C var*), 293
drivers_MCX_MCXW_MCXW727C._flexcan_rx_mb_config.id (*C var*), 293
drivers_MCX_MCXW_MCXW727C._flexcan_rx_mb_config.type (*C var*), 294
drivers_MCX_MCXW_MCXW727C._flexcan_timing_config (*C struct*), 291
drivers_MCX_MCXW_MCXW727C._flexcan_timing_config.fphaseSeg1 (*C var*), 291
drivers_MCX_MCXW_MCXW727C._flexcan_timing_config.fphaseSeg2 (*C var*), 291
drivers_MCX_MCXW_MCXW727C._flexcan_timing_config.fpreDivider (*C var*), 291
drivers_MCX_MCXW_MCXW727C._flexcan_timing_config.fpropSeg (*C var*), 291
drivers_MCX_MCXW_MCXW727C._flexcan_timing_config.frJumpwidth (*C var*), 291
drivers_MCX_MCXW_MCXW727C._flexcan_timing_config.phaseSeg1 (*C var*), 291
drivers_MCX_MCXW_MCXW727C._flexcan_timing_config.phaseSeg2 (*C var*), 291
drivers_MCX_MCXW_MCXW727C._flexcan_timing_config.preDivider (*C var*), 291
drivers_MCX_MCXW_MCXW727C._flexcan_timing_config.propSeg (*C var*), 291
drivers_MCX_MCXW_MCXW727C._flexcan_timing_config.rJumpwidth (*C var*), 291
drivers_MCX_MCXW_MCXW727C._flexcan_wake_up_source (*C enum*), 273
drivers_MCX_MCXW_MCXW727C._flexcan_wake_up_source.kFLEXCAN_WakeupSrcFiltered (*C enumerator*), 273
drivers_MCX_MCXW_MCXW727C._flexcan_wake_up_source.kFLEXCAN_WakeupSrcUnfiltered (*C enumerator*), 273
drivers_MCX_MCXW_MCXW727C._flexio_config_ (*C struct*), 325
drivers_MCX_MCXW_MCXW727C._flexio_config_.enableFastAccess (*C var*), 325
drivers_MCX_MCXW_MCXW727C._flexio_config_.enableFlexio (*C var*), 325
drivers_MCX_MCXW_MCXW727C._flexio_config_.enableInDebug (*C var*), 325
drivers_MCX_MCXW_MCXW727C._flexio_config_.enableInDoze (*C var*), 325
drivers_MCX_MCXW_MCXW727C._flexio_gpio_config (*C struct*), 327
drivers_MCX_MCXW_MCXW727C._flexio_gpio_config.inputConfig (*C var*), 327
drivers_MCX_MCXW_MCXW727C._flexio_gpio_config.outputLogic (*C var*), 327
drivers_MCX_MCXW_MCXW727C._flexio_gpio_config.pinDirection (*C var*), 327
drivers_MCX_MCXW_MCXW727C._flexio_gpio_direction (*C enum*), 322
drivers_MCX_MCXW_MCXW727C._flexio_gpio_direction.kFLEXIO_DigitalInput (*C enumerator*), 322
drivers_MCX_MCXW_MCXW727C._flexio_gpio_direction.kFLEXIO_DigitalOutput (*C enumerator*), 322
drivers_MCX_MCXW_MCXW727C._flexio_i2c_direction (*C enum*), 341
drivers_MCX_MCXW_MCXW727C._flexio_i2c_direction.kFLEXIO_I2C_Read (*C enumerator*), 341

drivers_MCX_MCXW_MCXW727C._flexio_i2c_direction.kFLEXIO_I2C_Write (*C enumerator*), 341
drivers_MCX_MCXW_MCXW727C._flexio_i2c_master_config (*C struct*), 342
drivers_MCX_MCXW_MCXW727C._flexio_i2c_master_config.baudRate_Bps (*C var*), 342
drivers_MCX_MCXW_MCXW727C._flexio_i2c_master_config.enableFastAccess (*C var*), 342
drivers_MCX_MCXW_MCXW727C._flexio_i2c_master_config.enableInDebug (*C var*), 342
drivers_MCX_MCXW_MCXW727C._flexio_i2c_master_config.enableInDoze (*C var*), 342
drivers_MCX_MCXW_MCXW727C._flexio_i2c_master_config.enableMaster (*C var*), 342
drivers_MCX_MCXW_MCXW727C._flexio_i2c_master_handle (*C struct*), 343
drivers_MCX_MCXW_MCXW727C._flexio_i2c_master_handle.completionCallback (*C var*), 343
drivers_MCX_MCXW_MCXW727C._flexio_i2c_master_handle.needRestart (*C var*), 343
drivers_MCX_MCXW_MCXW727C._flexio_i2c_master_handle.state (*C var*), 343
drivers_MCX_MCXW_MCXW727C._flexio_i2c_master_handle.transfer (*C var*), 343
drivers_MCX_MCXW_MCXW727C._flexio_i2c_master_handle.transferSize (*C var*), 343
drivers_MCX_MCXW_MCXW727C._flexio_i2c_master_handle.userData (*C var*), 343
drivers_MCX_MCXW_MCXW727C._flexio_i2c_master_interrupt (*C enum*), 340
drivers_MCX_MCXW_MCXW727C._flexio_i2c_master_interrupt.kFLEXIO_I2C_RxFullInterruptEnable (*C enumerator*), 340
drivers_MCX_MCXW_MCXW727C._flexio_i2c_master_interrupt.kFLEXIO_I2C_TxEmptyInterruptEnable (*C enumerator*), 340
drivers_MCX_MCXW_MCXW727C._flexio_i2c_master_status_flags (*C enum*), 340
drivers_MCX_MCXW_MCXW727C._flexio_i2c_master_status_flags.kFLEXIO_I2C_ReceiveNakFlag (*C enumerator*), 341
drivers_MCX_MCXW_MCXW727C._flexio_i2c_master_status_flags.kFLEXIO_I2C_RxFullFlag (*C enumerator*), 341
drivers_MCX_MCXW_MCXW727C._flexio_i2c_master_status_flags.kFLEXIO_I2C_TxEmptyFlag (*C enumerator*), 341
drivers_MCX_MCXW_MCXW727C._flexio_i2c_master_transfer (*C struct*), 342
drivers_MCX_MCXW_MCXW727C._flexio_i2c_master_transfer.data (*C var*), 343
drivers_MCX_MCXW_MCXW727C._flexio_i2c_master_transfer.dataSize (*C var*), 343
drivers_MCX_MCXW_MCXW727C._flexio_i2c_master_transfer.direction (*C var*), 343
drivers_MCX_MCXW_MCXW727C._flexio_i2c_master_transfer.flags (*C var*), 342
drivers_MCX_MCXW_MCXW727C._flexio_i2c_master_transfer.slaveAddress (*C var*), 343
drivers_MCX_MCXW_MCXW727C._flexio_i2c_master_transfer.subaddress (*C var*), 343
drivers_MCX_MCXW_MCXW727C._flexio_i2c_master_transfer.subaddressSize (*C var*), 343
drivers_MCX_MCXW_MCXW727C._flexio_i2c_type (*C struct*), 341
drivers_MCX_MCXW_MCXW727C._flexio_i2c_type.baudrate (*C var*), 342
drivers_MCX_MCXW_MCXW727C._flexio_i2c_type.flexioBase (*C var*), 342
drivers_MCX_MCXW_MCXW727C._flexio_i2c_type.SCLPinIndex (*C var*), 342
drivers_MCX_MCXW_MCXW727C._flexio_i2c_type.SDAPinIndex (*C var*), 342
drivers_MCX_MCXW_MCXW727C._flexio_i2c_type.shifterIndex (*C var*), 342
drivers_MCX_MCXW_MCXW727C._flexio_i2c_type.timerIndex (*C var*), 342
drivers_MCX_MCXW_MCXW727C._flexio_i2s_config (*C struct*), 353
drivers_MCX_MCXW_MCXW727C._flexio_i2s_config.bclkPinPolarity (*C var*), 354
drivers_MCX_MCXW_MCXW727C._flexio_i2s_config.enableI2S (*C var*), 353
drivers_MCX_MCXW_MCXW727C._flexio_i2s_config.fsPinPolarity (*C var*), 354
drivers_MCX_MCXW_MCXW727C._flexio_i2s_config.masterSlave (*C var*), 353
drivers_MCX_MCXW_MCXW727C._flexio_i2s_config.rxPinPolarity (*C var*), 354
drivers_MCX_MCXW_MCXW727C._flexio_i2s_config.rxTimerPolarity (*C var*), 354
drivers_MCX_MCXW_MCXW727C._flexio_i2s_config.txPinPolarity (*C var*), 353
drivers_MCX_MCXW_MCXW727C._flexio_i2s_config.txTimerPolarity (*C var*), 354
drivers_MCX_MCXW_MCXW727C._flexio_i2s_format (*C struct*), 354
drivers_MCX_MCXW_MCXW727C._flexio_i2s_format.bitWidth (*C var*), 354
drivers_MCX_MCXW_MCXW727C._flexio_i2s_format.sampleRate_Hz (*C var*), 354
drivers_MCX_MCXW_MCXW727C._flexio_i2s_handle (*C struct*), 354
drivers_MCX_MCXW_MCXW727C._flexio_i2s_handle.bitWidth (*C var*), 355
drivers_MCX_MCXW_MCXW727C._flexio_i2s_handle.callback (*C var*), 355
drivers_MCX_MCXW_MCXW727C._flexio_i2s_handle.queue (*C var*), 355

drivers_MCX_MCXW_MCXW727C._flexio_i2s_handle.queueDriver (*C var*), 355
drivers_MCX_MCXW_MCXW727C._flexio_i2s_handle.queueUser (*C var*), 355
drivers_MCX_MCXW_MCXW727C._flexio_i2s_handle.state (*C var*), 355
drivers_MCX_MCXW_MCXW727C._flexio_i2s_handle.transferSize (*C var*), 355
drivers_MCX_MCXW_MCXW727C._flexio_i2s_handle.userData (*C var*), 355
drivers_MCX_MCXW_MCXW727C._flexio_i2s_master_slave (*C enum*), 350
drivers_MCX_MCXW_MCXW727C._flexio_i2s_master_slave.kFLEXIO_I2S_Master (*C enumerator*), 350
drivers_MCX_MCXW_MCXW727C._flexio_i2s_master_slave.kFLEXIO_I2S_Slave (*C enumerator*), 350
drivers_MCX_MCXW_MCXW727C._flexio_i2s_sample_rate (*C enum*), 351
drivers_MCX_MCXW_MCXW727C._flexio_i2s_sample_rate.kFLEXIO_I2S_SampleRate8KHz (*C enumerator*), 351
drivers_MCX_MCXW_MCXW727C._flexio_i2s_sample_rate.kFLEXIO_I2S_SampleRate12KHz (*C enumerator*), 351
drivers_MCX_MCXW_MCXW727C._flexio_i2s_sample_rate.kFLEXIO_I2S_SampleRate16KHz (*C enumerator*), 351
drivers_MCX_MCXW_MCXW727C._flexio_i2s_sample_rate.kFLEXIO_I2S_SampleRate24KHz (*C enumerator*), 351
drivers_MCX_MCXW_MCXW727C._flexio_i2s_sample_rate.kFLEXIO_I2S_SampleRate32KHz (*C enumerator*), 351
drivers_MCX_MCXW_MCXW727C._flexio_i2s_sample_rate.kFLEXIO_I2S_SampleRate48KHz (*C enumerator*), 351
drivers_MCX_MCXW_MCXW727C._flexio_i2s_sample_rate.kFLEXIO_I2S_SampleRate96KHz (*C enumerator*), 351
drivers_MCX_MCXW_MCXW727C._flexio_i2s_sample_rate.kFLEXIO_I2S_SampleRate11025Hz (*C enumerator*), 351
drivers_MCX_MCXW_MCXW727C._flexio_i2s_sample_rate.kFLEXIO_I2S_SampleRate22050Hz (*C enumerator*), 351
drivers_MCX_MCXW_MCXW727C._flexio_i2s_sample_rate.kFLEXIO_I2S_SampleRate44100Hz (*C enumerator*), 351
drivers_MCX_MCXW_MCXW727C._flexio_i2s_transfer (*C struct*), 354
drivers_MCX_MCXW_MCXW727C._flexio_i2s_transfer.data (*C var*), 354
drivers_MCX_MCXW_MCXW727C._flexio_i2s_transfer.dataSize (*C var*), 354
drivers_MCX_MCXW_MCXW727C._flexio_i2s_type (*C struct*), 353
drivers_MCX_MCXW_MCXW727C._flexio_i2s_type.bclkPinIndex (*C var*), 353
drivers_MCX_MCXW_MCXW727C._flexio_i2s_type.bclkTimerIndex (*C var*), 353
drivers_MCX_MCXW_MCXW727C._flexio_i2s_type.flexioBase (*C var*), 353
drivers_MCX_MCXW_MCXW727C._flexio_i2s_type.fsPinIndex (*C var*), 353
drivers_MCX_MCXW_MCXW727C._flexio_i2s_type.fsTimerIndex (*C var*), 353
drivers_MCX_MCXW_MCXW727C._flexio_i2s_type.rxPinIndex (*C var*), 353
drivers_MCX_MCXW_MCXW727C._flexio_i2s_type.rxShifterIndex (*C var*), 353
drivers_MCX_MCXW_MCXW727C._flexio_i2s_type.txPinIndex (*C var*), 353
drivers_MCX_MCXW_MCXW727C._flexio_i2s_type.txShifterIndex (*C var*), 353
drivers_MCX_MCXW_MCXW727C._flexio_i2s_word_width (*C enum*), 352
drivers_MCX_MCXW_MCXW727C._flexio_i2s_word_width.kFLEXIO_I2S_WordWidth8bits (*C enumerator*), 352
drivers_MCX_MCXW_MCXW727C._flexio_i2s_word_width.kFLEXIO_I2S_WordWidth16bits (*C enumerator*), 352
drivers_MCX_MCXW_MCXW727C._flexio_i2s_word_width.kFLEXIO_I2S_WordWidth24bits (*C enumerator*), 352
drivers_MCX_MCXW_MCXW727C._flexio_i2s_word_width.kFLEXIO_I2S_WordWidth32bits (*C enumerator*), 352
drivers_MCX_MCXW_MCXW727C._flexio_pin_config (*C enum*), 316
drivers_MCX_MCXW_MCXW727C._flexio_pin_config.kFLEXIO_PinConfigBidirectionOutputData (*C enumerator*), 316
drivers_MCX_MCXW_MCXW727C._flexio_pin_config.kFLEXIO_PinConfigOpenDrainOrBidirection (*C enumerator*), 316

drivers_MCX_MCXW_MCXW727C._flexio_pin_config.kFLEXIO_PinConfigOutput (*C enumerator*), 316
drivers_MCX_MCXW_MCXW727C._flexio_pin_config.kFLEXIO_PinConfigOutputDisabled (*C enumerator*), 316
drivers_MCX_MCXW_MCXW727C._flexio_pin_input_config (*C enum*), 322
drivers_MCX_MCXW_MCXW727C._flexio_pin_input_config.kFLEXIO_FlagFallingEdgeEnable (*C enumerator*), 322
drivers_MCX_MCXW_MCXW727C._flexio_pin_input_config.kFLEXIO_FlagRisingEdgeEnable (*C enumerator*), 322
drivers_MCX_MCXW_MCXW727C._flexio_pin_input_config.kFLEXIO_InputInterruptDisabled (*C enumerator*), 322
drivers_MCX_MCXW_MCXW727C._flexio_pin_input_config.kFLEXIO_InputInterruptEnable (*C enumerator*), 322
drivers_MCX_MCXW_MCXW727C._flexio_pin_polarity (*C enum*), 316
drivers_MCX_MCXW_MCXW727C._flexio_pin_polarity.kFLEXIO_PinActiveHigh (*C enumerator*), 316
drivers_MCX_MCXW_MCXW727C._flexio_pin_polarity.kFLEXIO_PinActiveLow (*C enumerator*), 316
drivers_MCX_MCXW_MCXW727C._flexio_shifter_buffer_type (*C enum*), 321
drivers_MCX_MCXW_MCXW727C._flexio_shifter_buffer_type.kFLEXIO_ShifterBuffer (*C enumerator*), 321
drivers_MCX_MCXW_MCXW727C._flexio_shifter_buffer_type.kFLEXIO_ShifterBufferBitByteSwapped (*C enumerator*), 321
drivers_MCX_MCXW_MCXW727C._flexio_shifter_buffer_type.kFLEXIO_ShifterBufferBitSwapped (*C enumerator*), 321
drivers_MCX_MCXW_MCXW727C._flexio_shifter_buffer_type.kFLEXIO_ShifterBufferByteSwapped (*C enumerator*), 321
drivers_MCX_MCXW_MCXW727C._flexio_shifter_buffer_type.kFLEXIO_ShifterBufferHalfWordSwapped (*C enumerator*), 322
drivers_MCX_MCXW_MCXW727C._flexio_shifter_buffer_type.kFLEXIO_ShifterBufferNibbleByteSwapped (*C enumerator*), 322
drivers_MCX_MCXW_MCXW727C._flexio_shifter_buffer_type.kFLEXIO_ShifterBufferNibbleSwapped (*C enumerator*), 322
drivers_MCX_MCXW_MCXW727C._flexio_shifter_config (*C struct*), 326
drivers_MCX_MCXW_MCXW727C._flexio_shifter_config.inputSource (*C var*), 327
drivers_MCX_MCXW_MCXW727C._flexio_shifter_config.parallelWidth (*C var*), 327
drivers_MCX_MCXW_MCXW727C._flexio_shifter_config.pinConfig (*C var*), 326
drivers_MCX_MCXW_MCXW727C._flexio_shifter_config.pinPolarity (*C var*), 327
drivers_MCX_MCXW_MCXW727C._flexio_shifter_config.pinSelect (*C var*), 326
drivers_MCX_MCXW_MCXW727C._flexio_shifter_config.shifterMode (*C var*), 327
drivers_MCX_MCXW_MCXW727C._flexio_shifter_config.shifterStart (*C var*), 327
drivers_MCX_MCXW_MCXW727C._flexio_shifter_config.shifterStop (*C var*), 327
drivers_MCX_MCXW_MCXW727C._flexio_shifter_config.timerPolarity (*C var*), 326
drivers_MCX_MCXW_MCXW727C._flexio_shifter_config.timerSelect (*C var*), 326
drivers_MCX_MCXW_MCXW727C._flexio_shifter_input_source (*C enum*), 320
drivers_MCX_MCXW_MCXW727C._flexio_shifter_input_source.kFLEXIO_ShifterInputFromNextShifterOutput (*C enumerator*), 321
drivers_MCX_MCXW_MCXW727C._flexio_shifter_input_source.kFLEXIO_ShifterInputFromPin (*C enumerator*), 320
drivers_MCX_MCXW_MCXW727C._flexio_shifter_mode (*C enum*), 320
drivers_MCX_MCXW_MCXW727C._flexio_shifter_mode.kFLEXIO_ShifterDisabled (*C enumerator*), 320
drivers_MCX_MCXW_MCXW727C._flexio_shifter_mode.kFLEXIO_ShifterModeLogic (*C enumerator*), 320
drivers_MCX_MCXW_MCXW727C._flexio_shifter_mode.kFLEXIO_ShifterModeMatchContinuous (*C enumerator*), 320
drivers_MCX_MCXW_MCXW727C._flexio_shifter_mode.kFLEXIO_ShifterModeMatchStore (*C enumerator*), 320

drivers_MCX_MCXW_MCXW727C._flexio_shifter_mode.kFLEXIO_ShifterModeReceive (*C enumerator*), 320
drivers_MCX_MCXW_MCXW727C._flexio_shifter_mode.kFLEXIO_ShifterModeState (*C enumerator*), 320
drivers_MCX_MCXW_MCXW727C._flexio_shifter_mode.kFLEXIO_ShifterModeTransmit (*C enumerator*), 320
drivers_MCX_MCXW_MCXW727C._flexio_shifter_start_bit (*C enum*), 321
drivers_MCX_MCXW_MCXW727C._flexio_shifter_start_bit.kFLEXIO_ShifterStartBitDisabledLoadDataOnEnable (*C enumerator*), 321
drivers_MCX_MCXW_MCXW727C._flexio_shifter_start_bit.kFLEXIO_ShifterStartBitDisabledLoadDataOnShift (*C enumerator*), 321
drivers_MCX_MCXW_MCXW727C._flexio_shifter_start_bit.kFLEXIO_ShifterStartBitHigh (*C enumerator*), 321
drivers_MCX_MCXW_MCXW727C._flexio_shifter_start_bit.kFLEXIO_ShifterStartBitLow (*C enumerator*), 321
drivers_MCX_MCXW_MCXW727C._flexio_shifter_stop_bit (*C enum*), 321
drivers_MCX_MCXW_MCXW727C._flexio_shifter_stop_bit.kFLEXIO_ShifterStopBitDisable (*C enumerator*), 321
drivers_MCX_MCXW_MCXW727C._flexio_shifter_stop_bit.kFLEXIO_ShifterStopBitHigh (*C enumerator*), 321
drivers_MCX_MCXW_MCXW727C._flexio_shifter_stop_bit.kFLEXIO_ShifterStopBitLow (*C enumerator*), 321
drivers_MCX_MCXW_MCXW727C._flexio_shifter_timer_polarity (*C enum*), 320
drivers_MCX_MCXW_MCXW727C._flexio_shifter_timer_polarity.kFLEXIO_ShifterTimerPolarityOnNegitive (*C enumerator*), 320
drivers_MCX_MCXW_MCXW727C._flexio_shifter_timer_polarity.kFLEXIO_ShifterTimerPolarityOnPositive (*C enumerator*), 320
drivers_MCX_MCXW_MCXW727C._flexio_spi_clock_phase (*C enum*), 364
drivers_MCX_MCXW_MCXW727C._flexio_spi_clock_phase.kFLEXIO_SPI_ClockPhaseFirstEdge (*C enumerator*), 364
drivers_MCX_MCXW_MCXW727C._flexio_spi_clock_phase.kFLEXIO_SPI_ClockPhaseSecondEdge (*C enumerator*), 364
drivers_MCX_MCXW_MCXW727C._flexio_spi_data_bitcount_mode (*C enum*), 364
drivers_MCX_MCXW_MCXW727C._flexio_spi_data_bitcount_mode.kFLEXIO_SPI_8BitMode (*C enumerator*), 364
drivers_MCX_MCXW_MCXW727C._flexio_spi_data_bitcount_mode.kFLEXIO_SPI_16BitMode (*C enumerator*), 364
drivers_MCX_MCXW_MCXW727C._flexio_spi_data_bitcount_mode.kFLEXIO_SPI_32BitMode (*C enumerator*), 364
drivers_MCX_MCXW_MCXW727C._flexio_spi_dma_enable (*C enum*), 365
drivers_MCX_MCXW_MCXW727C._flexio_spi_dma_enable.kFLEXIO_SPI_DmaAllEnable (*C enumerator*), 365
drivers_MCX_MCXW_MCXW727C._flexio_spi_dma_enable.kFLEXIO_SPI_RxDmaEnable (*C enumerator*), 365
drivers_MCX_MCXW_MCXW727C._flexio_spi_dma_enable.kFLEXIO_SPI_TxDmaEnable (*C enumerator*), 365
drivers_MCX_MCXW_MCXW727C._flexio_spi_interrupt_enable (*C enum*), 364
drivers_MCX_MCXW_MCXW727C._flexio_spi_interrupt_enable.kFLEXIO_SPI_RxFullInterruptEnable (*C enumerator*), 364
drivers_MCX_MCXW_MCXW727C._flexio_spi_interrupt_enable.kFLEXIO_SPI_TxEmptyInterruptEnable (*C enumerator*), 364
drivers_MCX_MCXW_MCXW727C._flexio_spi_master_config (*C struct*), 367
drivers_MCX_MCXW_MCXW727C._flexio_spi_master_config.baudRate_Bps (*C var*), 367
drivers_MCX_MCXW_MCXW727C._flexio_spi_master_config.dataMode (*C var*), 367
drivers_MCX_MCXW_MCXW727C._flexio_spi_master_config.enableFastAccess (*C var*), 367
drivers_MCX_MCXW_MCXW727C._flexio_spi_master_config.enableInDebug (*C var*), 367
drivers_MCX_MCXW_MCXW727C._flexio_spi_master_config.enableInDoze (*C var*), 367
drivers_MCX_MCXW_MCXW727C._flexio_spi_master_config.enableMaster (*C var*), 367

drivers_MCX_MCXW_MCXW727C._flexio_spi_master_config.phase (*C var*), 367
drivers_MCX_MCXW_MCXW727C._flexio_spi_master_edma_handle (*C struct*), 330
drivers_MCX_MCXW_MCXW727C._flexio_spi_master_edma_handle.callback (*C var*), 331
drivers_MCX_MCXW_MCXW727C._flexio_spi_master_edma_handle.nbytes (*C var*), 330
drivers_MCX_MCXW_MCXW727C._flexio_spi_master_edma_handle.rxHandle (*C var*), 331
drivers_MCX_MCXW_MCXW727C._flexio_spi_master_edma_handle.rxInProgress (*C var*), 331
drivers_MCX_MCXW_MCXW727C._flexio_spi_master_edma_handle.transferSize (*C var*), 330
drivers_MCX_MCXW_MCXW727C._flexio_spi_master_edma_handle.txHandle (*C var*), 331
drivers_MCX_MCXW_MCXW727C._flexio_spi_master_edma_handle.txInProgress (*C var*), 330
drivers_MCX_MCXW_MCXW727C._flexio_spi_master_edma_handle.userData (*C var*), 331
drivers_MCX_MCXW_MCXW727C._flexio_spi_master_handle (*C struct*), 368
drivers_MCX_MCXW_MCXW727C._flexio_spi_master_handle.bytePerFrame (*C var*), 369
drivers_MCX_MCXW_MCXW727C._flexio_spi_master_handle.callback (*C var*), 369
drivers_MCX_MCXW_MCXW727C._flexio_spi_master_handle.direction (*C var*), 369
drivers_MCX_MCXW_MCXW727C._flexio_spi_master_handle.isCsContinuous (*C var*), 369
drivers_MCX_MCXW_MCXW727C._flexio_spi_master_handle.rxData (*C var*), 369
drivers_MCX_MCXW_MCXW727C._flexio_spi_master_handle.rxRemainingBytes (*C var*), 369
drivers_MCX_MCXW_MCXW727C._flexio_spi_master_handle.state (*C var*), 369
drivers_MCX_MCXW_MCXW727C._flexio_spi_master_handle.timer1Cfg (*C var*), 369
drivers_MCX_MCXW_MCXW727C._flexio_spi_master_handle.transferSize (*C var*), 369
drivers_MCX_MCXW_MCXW727C._flexio_spi_master_handle.txData (*C var*), 369
drivers_MCX_MCXW_MCXW727C._flexio_spi_master_handle.txRemainingBytes (*C var*), 369
drivers_MCX_MCXW_MCXW727C._flexio_spi_master_handle.userData (*C var*), 369
drivers_MCX_MCXW_MCXW727C._flexio_spi_shift_direction (*C enum*), 364
drivers_MCX_MCXW_MCXW727C._flexio_spi_shift_direction.kFLEXIO_SPI_LsbFirst (*C enumerator*), 364
drivers_MCX_MCXW_MCXW727C._flexio_spi_shift_direction.kFLEXIO_SPI_MsbFirst (*C enumerator*), 364
drivers_MCX_MCXW_MCXW727C._flexio_spi_slave_config (*C struct*), 368
drivers_MCX_MCXW_MCXW727C._flexio_spi_slave_config.dataMode (*C var*), 368
drivers_MCX_MCXW_MCXW727C._flexio_spi_slave_config.enableFastAccess (*C var*), 368
drivers_MCX_MCXW_MCXW727C._flexio_spi_slave_config.enableInDebug (*C var*), 368
drivers_MCX_MCXW_MCXW727C._flexio_spi_slave_config.enableInDoze (*C var*), 368
drivers_MCX_MCXW_MCXW727C._flexio_spi_slave_config.enableSlave (*C var*), 368
drivers_MCX_MCXW_MCXW727C._flexio_spi_slave_config.phase (*C var*), 368
drivers_MCX_MCXW_MCXW727C._flexio_spi_status_flags (*C enum*), 364
drivers_MCX_MCXW_MCXW727C._flexio_spi_status_flags.kFLEXIO_SPI_RxBufferFullFlag (*C enumerator*), 365
drivers_MCX_MCXW_MCXW727C._flexio_spi_status_flags.kFLEXIO_SPI_TxBufferEmptyFlag (*C enumerator*), 365
drivers_MCX_MCXW_MCXW727C._flexio_spi_transfer (*C struct*), 368
drivers_MCX_MCXW_MCXW727C._flexio_spi_transfer_flags (*C enum*), 365
drivers_MCX_MCXW_MCXW727C._flexio_spi_transfer_flags.kFLEXIO_SPI_8bitLsb (*C enumerator*), 365
drivers_MCX_MCXW_MCXW727C._flexio_spi_transfer_flags.kFLEXIO_SPI_8bitMsb (*C enumerator*), 365
drivers_MCX_MCXW_MCXW727C._flexio_spi_transfer_flags.kFLEXIO_SPI_16bitLsb (*C enumerator*), 365
drivers_MCX_MCXW_MCXW727C._flexio_spi_transfer_flags.kFLEXIO_SPI_16bitMsb (*C enumerator*), 365
drivers_MCX_MCXW_MCXW727C._flexio_spi_transfer_flags.kFLEXIO_SPI_32bitLsb (*C enumerator*), 365
drivers_MCX_MCXW_MCXW727C._flexio_spi_transfer_flags.kFLEXIO_SPI_32bitMsb (*C enumerator*), 365
drivers_MCX_MCXW_MCXW727C._flexio_spi_transfer_flags.kFLEXIO_SPI_csContinuous (*C enumerator*), 365
drivers_MCX_MCXW_MCXW727C._flexio_spi_transfer.dataSize (*C var*), 368
drivers_MCX_MCXW_MCXW727C._flexio_spi_transfer.flags (*C var*), 368

drivers_MCX_MCXW_MCXW727C._flexio_spi_transfer.rxData (*C var*), 368
drivers_MCX_MCXW_MCXW727C._flexio_spi_transfer.txData (*C var*), 368
drivers_MCX_MCXW_MCXW727C._flexio_spi_type (*C struct*), 366
drivers_MCX_MCXW_MCXW727C._flexio_spi_type.CSnPinIndex (*C var*), 367
drivers_MCX_MCXW_MCXW727C._flexio_spi_type.flexioBase (*C var*), 367
drivers_MCX_MCXW_MCXW727C._flexio_spi_type.SCKPinIndex (*C var*), 367
drivers_MCX_MCXW_MCXW727C._flexio_spi_type.SDIPinIndex (*C var*), 367
drivers_MCX_MCXW_MCXW727C._flexio_spi_type.SDOPinIndex (*C var*), 367
drivers_MCX_MCXW_MCXW727C._flexio_spi_type.shifterIndex (*C var*), 367
drivers_MCX_MCXW_MCXW727C._flexio_spi_type.timerIndex (*C var*), 367
drivers_MCX_MCXW_MCXW727C._flexio_timer_config (*C struct*), 325
drivers_MCX_MCXW_MCXW727C._flexio_timer_config.pinConfig (*C var*), 325
drivers_MCX_MCXW_MCXW727C._flexio_timer_config.pinPolarity (*C var*), 325
drivers_MCX_MCXW_MCXW727C._flexio_timer_config.pinSelect (*C var*), 325
drivers_MCX_MCXW_MCXW727C._flexio_timer_config.timerCompare (*C var*), 326
drivers_MCX_MCXW_MCXW727C._flexio_timer_config.timerDecrement (*C var*), 326
drivers_MCX_MCXW_MCXW727C._flexio_timer_config.timerDisable (*C var*), 326
drivers_MCX_MCXW_MCXW727C._flexio_timer_config.timerEnable (*C var*), 326
drivers_MCX_MCXW_MCXW727C._flexio_timer_config.timerMode (*C var*), 326
drivers_MCX_MCXW_MCXW727C._flexio_timer_config.timerOutput (*C var*), 326
drivers_MCX_MCXW_MCXW727C._flexio_timer_config.timerReset (*C var*), 326
drivers_MCX_MCXW_MCXW727C._flexio_timer_config.timerStart (*C var*), 326
drivers_MCX_MCXW_MCXW727C._flexio_timer_config.timerStop (*C var*), 326
drivers_MCX_MCXW_MCXW727C._flexio_timer_config.triggerPolarity (*C var*), 325
drivers_MCX_MCXW_MCXW727C._flexio_timer_config.triggerSelect (*C var*), 325
drivers_MCX_MCXW_MCXW727C._flexio_timer_config.triggerSource (*C var*), 325
drivers_MCX_MCXW_MCXW727C._flexio_timer_decrement_source (*C enum*), 317
drivers_MCX_MCXW_MCXW727C._flexio_timer_decrement_source.kFLEXIO_TimerDecSrcOnFlexIOClockShiftT
(*C enumerator*), 317
drivers_MCX_MCXW_MCXW727C._flexio_timer_decrement_source.kFLEXIO_TimerDecSrcOnPinInputShiftPinIn
(*C enumerator*), 317
drivers_MCX_MCXW_MCXW727C._flexio_timer_decrement_source.kFLEXIO_TimerDecSrcOnTriggerInputShiftT
(*C enumerator*), 317
drivers_MCX_MCXW_MCXW727C._flexio_timer_decrement_source.kFLEXIO_TimerDecSrcOnTriggerInputShiftT
(*C enumerator*), 317
drivers_MCX_MCXW_MCXW727C._flexio_timer_disable_condition (*C enum*), 318
drivers_MCX_MCXW_MCXW727C._flexio_timer_disable_condition.kFLEXIO_TimerDisableNever
(*C enumerator*), 318
drivers_MCX_MCXW_MCXW727C._flexio_timer_disable_condition.kFLEXIO_TimerDisableOnPinBothEdge
(*C enumerator*), 318
drivers_MCX_MCXW_MCXW727C._flexio_timer_disable_condition.kFLEXIO_TimerDisableOnPinBothEdgeTrigg
(*C enumerator*), 318
drivers_MCX_MCXW_MCXW727C._flexio_timer_disable_condition.kFLEXIO_TimerDisableOnPreTimerDisable
(*C enumerator*), 318
drivers_MCX_MCXW_MCXW727C._flexio_timer_disable_condition.kFLEXIO_TimerDisableOnTimerCompare
(*C enumerator*), 318
drivers_MCX_MCXW_MCXW727C._flexio_timer_disable_condition.kFLEXIO_TimerDisableOnTimerCompareTrig
(*C enumerator*), 318
drivers_MCX_MCXW_MCXW727C._flexio_timer_disable_condition.kFLEXIO_TimerDisableOnTriggerFallingEdge
(*C enumerator*), 318
drivers_MCX_MCXW_MCXW727C._flexio_timer_enable_condition (*C enum*), 318
drivers_MCX_MCXW_MCXW727C._flexio_timer_enable_condition.kFLEXIO_TimerEnabledAlways
(*C enumerator*), 318
drivers_MCX_MCXW_MCXW727C._flexio_timer_enable_condition.kFLEXIO_TimerEnableOnPinRisingEdge
(*C enumerator*), 319
drivers_MCX_MCXW_MCXW727C._flexio_timer_enable_condition.kFLEXIO_TimerEnableOnPinRisingEdgeTrigg
(*C enumerator*), 319
drivers_MCX_MCXW_MCXW727C._flexio_timer_enable_condition.kFLEXIO_TimerEnableOnPrevTimerEnable

(*C enumerator*), 318
drivers_MCX_MCXW_MCXW727C._flexio_timer_enable_condition.kFLEXIO_TimerEnableOnTriggerBothEdge (*C enumerator*), 319
drivers_MCX_MCXW_MCXW727C._flexio_timer_enable_condition.kFLEXIO_TimerEnableOnTriggerHigh (*C enumerator*), 319
drivers_MCX_MCXW_MCXW727C._flexio_timer_enable_condition.kFLEXIO_TimerEnableOnTriggerHighPinHigh (*C enumerator*), 319
drivers_MCX_MCXW_MCXW727C._flexio_timer_enable_condition.kFLEXIO_TimerEnableOnTriggerRisingEdge (*C enumerator*), 319
drivers_MCX_MCXW_MCXW727C._flexio_timer_mode (*C enum*), 316
drivers_MCX_MCXW_MCXW727C._flexio_timer_mode.kFLEXIO_TimerModeDisabled (*C enumerator*), 316
drivers_MCX_MCXW_MCXW727C._flexio_timer_mode.kFLEXIO_TimerModeDual8BitBaudBit (*C enumerator*), 316
drivers_MCX_MCXW_MCXW727C._flexio_timer_mode.kFLEXIO_TimerModeDual8BitPWM (*C enumerator*), 316
drivers_MCX_MCXW_MCXW727C._flexio_timer_mode.kFLEXIO_TimerModeDual8BitPWMLow (*C enumerator*), 317
drivers_MCX_MCXW_MCXW727C._flexio_timer_mode.kFLEXIO_TimerModeSingle16Bit (*C enumerator*), 317
drivers_MCX_MCXW_MCXW727C._flexio_timer_output (*C enum*), 317
drivers_MCX_MCXW_MCXW727C._flexio_timer_output_state (*C enum*), 319
drivers_MCX_MCXW_MCXW727C._flexio_timer_output_state.kFLEXIO_PwmHigh (*C enumerator*), 320
drivers_MCX_MCXW_MCXW727C._flexio_timer_output_state.kFLEXIO_PwmLow (*C enumerator*), 320
drivers_MCX_MCXW_MCXW727C._flexio_timer_output.kFLEXIO_TimerOutputOneAffectedByReset (*C enumerator*), 317
drivers_MCX_MCXW_MCXW727C._flexio_timer_output.kFLEXIO_TimerOutputOneNotAffectedByReset (*C enumerator*), 317
drivers_MCX_MCXW_MCXW727C._flexio_timer_output.kFLEXIO_TimerOutputZeroAffectedByReset (*C enumerator*), 317
drivers_MCX_MCXW_MCXW727C._flexio_timer_output.kFLEXIO_TimerOutputZeroNotAffectedByReset (*C enumerator*), 317
drivers_MCX_MCXW_MCXW727C._flexio_timer_reset_condition (*C enum*), 317
drivers_MCX_MCXW_MCXW727C._flexio_timer_reset_condition.kFLEXIO_TimerResetNever (*C enumerator*), 317
drivers_MCX_MCXW_MCXW727C._flexio_timer_reset_condition.kFLEXIO_TimerResetOnTimerPinEqualToTime (*C enumerator*), 317
drivers_MCX_MCXW_MCXW727C._flexio_timer_reset_condition.kFLEXIO_TimerResetOnTimerPinRisingEdge (*C enumerator*), 318
drivers_MCX_MCXW_MCXW727C._flexio_timer_reset_condition.kFLEXIO_TimerResetOnTimerTriggerBothEdge (*C enumerator*), 318
drivers_MCX_MCXW_MCXW727C._flexio_timer_reset_condition.kFLEXIO_TimerResetOnTimerTriggerEqualToT (*C enumerator*), 318
drivers_MCX_MCXW_MCXW727C._flexio_timer_reset_condition.kFLEXIO_TimerResetOnTimerTriggerRisingEdg (*C enumerator*), 318
drivers_MCX_MCXW_MCXW727C._flexio_timer_start_bit_condition (*C enum*), 319
drivers_MCX_MCXW_MCXW727C._flexio_timer_start_bit_condition.kFLEXIO_TimerStartBitDisabled (*C enumerator*), 319
drivers_MCX_MCXW_MCXW727C._flexio_timer_start_bit_condition.kFLEXIO_TimerStartBitEnabled (*C enumerator*), 319
drivers_MCX_MCXW_MCXW727C._flexio_timer_stop_bit_condition (*C enum*), 319
drivers_MCX_MCXW_MCXW727C._flexio_timer_stop_bit_condition.kFLEXIO_TimerStopBitDisabled (*C enumerator*), 319
drivers_MCX_MCXW_MCXW727C._flexio_timer_stop_bit_condition.kFLEXIO_TimerStopBitEnableOnTimerCon (*C enumerator*), 319
drivers_MCX_MCXW_MCXW727C._flexio_timer_stop_bit_condition.kFLEXIO_TimerStopBitEnableOnTimerCon

(*C enumerator*), 319
drivers_MCX_MCXW_MCXW727C._flexio_timer_stop_bit_condition.kFLEXIO_TimerStopBitEnableOnTimerDisa
(*C enumerator*), 319
drivers_MCX_MCXW_MCXW727C._flexio_timer_trigger_polarity (*C enum*), 315
drivers_MCX_MCXW_MCXW727C._flexio_timer_trigger_polarity.kFLEXIO_TimerTriggerPolarityActiveHigh
(*C enumerator*), 315
drivers_MCX_MCXW_MCXW727C._flexio_timer_trigger_polarity.kFLEXIO_TimerTriggerPolarityActiveLow
(*C enumerator*), 315
drivers_MCX_MCXW_MCXW727C._flexio_timer_trigger_source (*C enum*), 316
drivers_MCX_MCXW_MCXW727C._flexio_timer_trigger_source.kFLEXIO_TimerTriggerSourceExternal
(*C enumerator*), 316
drivers_MCX_MCXW_MCXW727C._flexio_timer_trigger_source.kFLEXIO_TimerTriggerSourceInternal
(*C enumerator*), 316
drivers_MCX_MCXW_MCXW727C._flexio_uart_bit_count_per_char (*C enum*), 377
drivers_MCX_MCXW_MCXW727C._flexio_uart_bit_count_per_char.kFLEXIO_UART_7BitsPerChar
(*C enumerator*), 377
drivers_MCX_MCXW_MCXW727C._flexio_uart_bit_count_per_char.kFLEXIO_UART_8BitsPerChar
(*C enumerator*), 377
drivers_MCX_MCXW_MCXW727C._flexio_uart_bit_count_per_char.kFLEXIO_UART_9BitsPerChar
(*C enumerator*), 377
drivers_MCX_MCXW_MCXW727C._flexio_uart_config (*C struct*), 379
drivers_MCX_MCXW_MCXW727C._flexio_uart_config.baudRate_Bps (*C var*), 379
drivers_MCX_MCXW_MCXW727C._flexio_uart_config.bitCountPerChar (*C var*), 379
drivers_MCX_MCXW_MCXW727C._flexio_uart_config.enableFastAccess (*C var*), 379
drivers_MCX_MCXW_MCXW727C._flexio_uart_config.enableInDebug (*C var*), 379
drivers_MCX_MCXW_MCXW727C._flexio_uart_config.enableInDoze (*C var*), 379
drivers_MCX_MCXW_MCXW727C._flexio_uart_config.enableUart (*C var*), 379
drivers_MCX_MCXW_MCXW727C._flexio_uart_edma_handle (*C struct*), 333
drivers_MCX_MCXW_MCXW727C._flexio_uart_edma_handle.callback (*C var*), 333
drivers_MCX_MCXW_MCXW727C._flexio_uart_edma_handle.nbytes (*C var*), 334
drivers_MCX_MCXW_MCXW727C._flexio_uart_edma_handle.rxDataSizeAll (*C var*), 333
drivers_MCX_MCXW_MCXW727C._flexio_uart_edma_handle.rxEdmaHandle (*C var*), 333
drivers_MCX_MCXW_MCXW727C._flexio_uart_edma_handle.rxState (*C var*), 334
drivers_MCX_MCXW_MCXW727C._flexio_uart_edma_handle.txDataSizeAll (*C var*), 333
drivers_MCX_MCXW_MCXW727C._flexio_uart_edma_handle.txEdmaHandle (*C var*), 333
drivers_MCX_MCXW_MCXW727C._flexio_uart_edma_handle.txState (*C var*), 334
drivers_MCX_MCXW_MCXW727C._flexio_uart_edma_handle.userData (*C var*), 333
drivers_MCX_MCXW_MCXW727C._flexio_uart_handle (*C struct*), 379
drivers_MCX_MCXW_MCXW727C._flexio_uart_handle.callback (*C var*), 380
drivers_MCX_MCXW_MCXW727C._flexio_uart_handle.rxData (*C var*), 380
drivers_MCX_MCXW_MCXW727C._flexio_uart_handle.rxDataSize (*C var*), 380
drivers_MCX_MCXW_MCXW727C._flexio_uart_handle.rxDataSizeAll (*C var*), 380
drivers_MCX_MCXW_MCXW727C._flexio_uart_handle.rxRingBuffer (*C var*), 380
drivers_MCX_MCXW_MCXW727C._flexio_uart_handle.rxRingBufferHead (*C var*), 380
drivers_MCX_MCXW_MCXW727C._flexio_uart_handle.rxRingBufferSize (*C var*), 380
drivers_MCX_MCXW_MCXW727C._flexio_uart_handle.rxRingBufferTail (*C var*), 380
drivers_MCX_MCXW_MCXW727C._flexio_uart_handle.rxState (*C var*), 380
drivers_MCX_MCXW_MCXW727C._flexio_uart_handle.txData (*C var*), 380
drivers_MCX_MCXW_MCXW727C._flexio_uart_handle.txDataSize (*C var*), 380
drivers_MCX_MCXW_MCXW727C._flexio_uart_handle.txDataSizeAll (*C var*), 380
drivers_MCX_MCXW_MCXW727C._flexio_uart_handle.txState (*C var*), 380
drivers_MCX_MCXW_MCXW727C._flexio_uart_handle.userData (*C var*), 380
drivers_MCX_MCXW_MCXW727C._flexio_uart_interrupt_enable (*C enum*), 377
drivers_MCX_MCXW_MCXW727C._flexio_uart_interrupt_enable.kFLEXIO_UART_RxDataRegFullInterruptEnal
(*C enumerator*), 377
drivers_MCX_MCXW_MCXW727C._flexio_uart_interrupt_enable.kFLEXIO_UART_TxDataRegEmptyInterruptE
(*C enumerator*), 377
drivers_MCX_MCXW_MCXW727C._flexio_uart_status_flags (*C enum*), 378

drivers_MCX_MCXW_MCXW727C._flexio_uart_status_flags.kFLEXIO_UART_RxDataRegFullFlag (*C enumerator*), 378
drivers_MCX_MCXW_MCXW727C._flexio_uart_status_flags.kFLEXIO_UART_RxOverRunFlag (*C enumerator*), 378
drivers_MCX_MCXW_MCXW727C._flexio_uart_status_flags.kFLEXIO_UART_TxDataRegEmptyFlag (*C enumerator*), 378
drivers_MCX_MCXW_MCXW727C._flexio_uart_transfer (*C struct*), 379
drivers_MCX_MCXW_MCXW727C._flexio_uart_transfer.__unnamed78__ (*C union*), 380
drivers_MCX_MCXW_MCXW727C._flexio_uart_transfer.__unnamed78__.data (*C var*), 380
drivers_MCX_MCXW_MCXW727C._flexio_uart_transfer.__unnamed78__.rxData (*C var*), 381
drivers_MCX_MCXW_MCXW727C._flexio_uart_transfer.__unnamed78__.txData (*C var*), 381
drivers_MCX_MCXW_MCXW727C._flexio_uart_transfer.dataSize (*C var*), 379
drivers_MCX_MCXW_MCXW727C._flexio_uart_type (*C struct*), 378
drivers_MCX_MCXW_MCXW727C._flexio_uart_type.flexioBase (*C var*), 378
drivers_MCX_MCXW_MCXW727C._flexio_uart_type.RxPinIndex (*C var*), 378
drivers_MCX_MCXW_MCXW727C._flexio_uart_type.shifterIndex (*C var*), 379
drivers_MCX_MCXW_MCXW727C._flexio_uart_type.timerIndex (*C var*), 379
drivers_MCX_MCXW_MCXW727C._flexio_uart_type.TxPinIndex (*C var*), 378
drivers_MCX_MCXW_MCXW727C._fro192m_rf_clk_config (*C struct*), 198
drivers_MCX_MCXW_MCXW727C._fro192m_rf_clk_config.apb_rfcmc_div (*C var*), 198
drivers_MCX_MCXW_MCXW727C._fro192m_rf_clk_config.range (*C var*), 198
drivers_MCX_MCXW_MCXW727C._fro192m_rf_clk_div (*C enum*), 190
drivers_MCX_MCXW_MCXW727C._fro192m_rf_clk_div.kFro192M_ClkDivBy1 (*C enumerator*), 190
drivers_MCX_MCXW_MCXW727C._fro192m_rf_clk_div.kFro192M_ClkDivBy2 (*C enumerator*), 190
drivers_MCX_MCXW_MCXW727C._fro192m_rf_clk_div.kFro192M_ClkDivBy4 (*C enumerator*), 190
drivers_MCX_MCXW_MCXW727C._fro192m_rf_clk_div.kFro192M_ClkDivBy8 (*C enumerator*), 191
drivers_MCX_MCXW_MCXW727C._fro192m_rf_range (*C enum*), 190
drivers_MCX_MCXW_MCXW727C._fro192m_rf_range.kFro192M_Range16M (*C enumerator*), 190
drivers_MCX_MCXW_MCXW727C._fro192m_rf_range.kFro192M_Range24M (*C enumerator*), 190
drivers_MCX_MCXW_MCXW727C._fro192m_rf_range.kFro192M_Range32M (*C enumerator*), 190
drivers_MCX_MCXW_MCXW727C._fro192m_rf_range.kFro192M_Range48M (*C enumerator*), 190
drivers_MCX_MCXW_MCXW727C._fro192m_rf_range.kFro192M_Range64M (*C enumerator*), 190
drivers_MCX_MCXW_MCXW727C._gpio_checker_attribute (*C enum*), 381
drivers_MCX_MCXW_MCXW727C._gpio_checker_attribute.kGPIO_IgnoreAttributeCheck (*C enumerator*), 382
drivers_MCX_MCXW_MCXW727C._gpio_checker_attribute.kGPIO_UsernonsecureNUsersecureNPrivilegedsecureN (*C enumerator*), 382
drivers_MCX_MCXW_MCXW727C._gpio_checker_attribute.kGPIO_UsernonsecureNUsersecureNPrivilegedsecureR (*C enumerator*), 381
drivers_MCX_MCXW_MCXW727C._gpio_checker_attribute.kGPIO_UsernonsecureNUsersecureNPrivilegedsecureR (*C enumerator*), 381
drivers_MCX_MCXW_MCXW727C._gpio_checker_attribute.kGPIO_UsernonsecureNUsersecureRPrivilegedsecureR (*C enumerator*), 381
drivers_MCX_MCXW_MCXW727C._gpio_checker_attribute.kGPIO_UsernonsecureNUsersecureRWPrivilegedsecure (*C enumerator*), 381
drivers_MCX_MCXW_MCXW727C._gpio_checker_attribute.kGPIO_UsernonsecureRUsersecureRPrivilegedsecureR (*C enumerator*), 381
drivers_MCX_MCXW_MCXW727C._gpio_checker_attribute.kGPIO_UsernonsecureRUsersecureRWPrivilegedsecure (*C enumerator*), 381
drivers_MCX_MCXW_MCXW727C._gpio_checker_attribute.kGPIO_UsernonsecureRWUsersecureRWPrivilegedsecu (*C enumerator*), 381
drivers_MCX_MCXW_MCXW727C._gpio_interrupt_config (*C enum*), 382
drivers_MCX_MCXW_MCXW727C._gpio_interrupt_config.kGPIO_ActiveHighTriggerOutputEnable (*C enumerator*), 382

drivers_MCX_MCXW_MCXW727C._gpio_interrupt_config.kGPIO_ActiveLowTriggerOutputEnable (*C enumerator*), 383
drivers_MCX_MCXW_MCXW727C._gpio_interrupt_config.kGPIO_DMAEitherEdge (*C enumerator*), 382
drivers_MCX_MCXW_MCXW727C._gpio_interrupt_config.kGPIO_DMAFallingEdge (*C enumerator*), 382
drivers_MCX_MCXW_MCXW727C._gpio_interrupt_config.kGPIO_DMARisingEdge (*C enumerator*), 382
drivers_MCX_MCXW_MCXW727C._gpio_interrupt_config.kGPIO_FlagEitherEdge (*C enumerator*), 382
drivers_MCX_MCXW_MCXW727C._gpio_interrupt_config.kGPIO_FlagFallingEdge (*C enumerator*), 382
drivers_MCX_MCXW_MCXW727C._gpio_interrupt_config.kGPIO_FlagRisingEdge (*C enumerator*), 382
drivers_MCX_MCXW_MCXW727C._gpio_interrupt_config.kGPIO_InterruptEitherEdge (*C enumerator*), 382
drivers_MCX_MCXW_MCXW727C._gpio_interrupt_config.kGPIO_InterruptFallingEdge (*C enumerator*), 382
drivers_MCX_MCXW_MCXW727C._gpio_interrupt_config.kGPIO_InterruptLogicOne (*C enumerator*), 382
drivers_MCX_MCXW_MCXW727C._gpio_interrupt_config.kGPIO_InterruptLogicZero (*C enumerator*), 382
drivers_MCX_MCXW_MCXW727C._gpio_interrupt_config.kGPIO_InterruptRisingEdge (*C enumerator*), 382
drivers_MCX_MCXW_MCXW727C._gpio_interrupt_config.kGPIO_InterruptStatusFlagDisabled (*C enumerator*), 382
drivers_MCX_MCXW_MCXW727C._gpio_interrupt_selection (*C enum*), 383
drivers_MCX_MCXW_MCXW727C._gpio_interrupt_selection.kGPIO_InterruptOutput0 (*C enumerator*), 383
drivers_MCX_MCXW_MCXW727C._gpio_interrupt_selection.kGPIO_InterruptOutput1 (*C enumerator*), 383
drivers_MCX_MCXW_MCXW727C._gpio_pin_config (*C struct*), 384
drivers_MCX_MCXW_MCXW727C._gpio_pin_config.outputLogic (*C var*), 384
drivers_MCX_MCXW_MCXW727C._gpio_pin_config.pinDirection (*C var*), 384
drivers_MCX_MCXW_MCXW727C._gpio_pin_direction (*C enum*), 381
drivers_MCX_MCXW_MCXW727C._gpio_pin_direction.kGPIO_DigitalInput (*C enumerator*), 381
drivers_MCX_MCXW_MCXW727C._gpio_pin_direction.kGPIO_DigitalOutput (*C enumerator*), 381
drivers_MCX_MCXW_MCXW727C._gpio_version_info (*C struct*), 384
drivers_MCX_MCXW_MCXW727C._gpio_version_info.feature (*C var*), 384
drivers_MCX_MCXW_MCXW727C._gpio_version_info.major (*C var*), 384
drivers_MCX_MCXW_MCXW727C._gpio_version_info.minor (*C var*), 384
drivers_MCX_MCXW_MCXW727C._i3c_baudrate (*C struct*), 423
drivers_MCX_MCXW_MCXW727C._i3c_baudrate.i2cBaud (*C var*), 423
drivers_MCX_MCXW_MCXW727C._i3c_baudrate.i3cOpenDrainBaud (*C var*), 423
drivers_MCX_MCXW_MCXW727C._i3c_baudrate.i3cPushPullBaud (*C var*), 423
drivers_MCX_MCXW_MCXW727C._i3c_bus_request (*C enum*), 416
drivers_MCX_MCXW_MCXW727C._i3c_bus_request.kI3C_RequestAutoIbi (*C enumerator*), 416
drivers_MCX_MCXW_MCXW727C._i3c_bus_request.kI3C_RequestEmitStartAddr (*C enumerator*), 416
drivers_MCX_MCXW_MCXW727C._i3c_bus_request.kI3C_RequestEmitStop (*C enumerator*), 416
drivers_MCX_MCXW_MCXW727C._i3c_bus_request.kI3C_RequestForceExit (*C enumerator*), 416
drivers_MCX_MCXW_MCXW727C._i3c_bus_request.kI3C_RequestIbiAckNack (*C enumerator*), 416
drivers_MCX_MCXW_MCXW727C._i3c_bus_request.kI3C_RequestNone (*C enumerator*), 416
drivers_MCX_MCXW_MCXW727C._i3c_bus_request.kI3C_RequestProcessDAA (*C enumerator*), 416
drivers_MCX_MCXW_MCXW727C._i3c_bus_type (*C enum*), 417

drivers_MCX_MCXW_MCXW727C._i3c_bus_type.kI3C_TypeI2C (*C enumerator*), 417
drivers_MCX_MCXW_MCXW727C._i3c_bus_type.kI3C_TypeI3CDdr (*C enumerator*), 417
drivers_MCX_MCXW_MCXW727C._i3c_bus_type.kI3C_TypeI3CSdr (*C enumerator*), 417
drivers_MCX_MCXW_MCXW727C._i3c_config (*C struct*), 394
drivers_MCX_MCXW_MCXW727C._i3c_config.baudRate_Hz (*C var*), 394
drivers_MCX_MCXW_MCXW727C._i3c_config.bcr (*C var*), 395
drivers_MCX_MCXW_MCXW727C._i3c_config.dcr (*C var*), 395
drivers_MCX_MCXW_MCXW727C._i3c_config.disableTimeout (*C var*), 394
drivers_MCX_MCXW_MCXW727C._i3c_config.enableMaster (*C var*), 394
drivers_MCX_MCXW_MCXW727C._i3c_config.enableOpenDrainHigh (*C var*), 394
drivers_MCX_MCXW_MCXW727C._i3c_config.enableOpenDrainStop (*C var*), 394
drivers_MCX_MCXW_MCXW727C._i3c_config.enableSlave (*C var*), 395
drivers_MCX_MCXW_MCXW727C._i3c_config.hdrMode (*C var*), 395
drivers_MCX_MCXW_MCXW727C._i3c_config.hKeep (*C var*), 394
drivers_MCX_MCXW_MCXW727C._i3c_config.ignoreS0S1Error (*C var*), 395
drivers_MCX_MCXW_MCXW727C._i3c_config.masterDynamicAddress (*C var*), 395
drivers_MCX_MCXW_MCXW727C._i3c_config.matchSlaveStartStop (*C var*), 395
drivers_MCX_MCXW_MCXW727C._i3c_config.maxReadLength (*C var*), 395
drivers_MCX_MCXW_MCXW727C._i3c_config.maxWriteLength (*C var*), 395
drivers_MCX_MCXW_MCXW727C._i3c_config.nakAllRequest (*C var*), 395
drivers_MCX_MCXW_MCXW727C._i3c_config.offline (*C var*), 395
drivers_MCX_MCXW_MCXW727C._i3c_config.partNumber (*C var*), 395
drivers_MCX_MCXW_MCXW727C._i3c_config.restartSclDelay (*C var*), 395
drivers_MCX_MCXW_MCXW727C._i3c_config.startSclDelay (*C var*), 395
drivers_MCX_MCXW_MCXW727C._i3c_config.staticAddr (*C var*), 395
drivers_MCX_MCXW_MCXW727C._i3c_config.vendorID (*C var*), 395
drivers_MCX_MCXW_MCXW727C._i3c_device_info (*C struct*), 392
drivers_MCX_MCXW_MCXW727C._i3c_device_info.bcr (*C var*), 392
drivers_MCX_MCXW_MCXW727C._i3c_device_info.dcr (*C var*), 392
drivers_MCX_MCXW_MCXW727C._i3c_device_info.dynamicAddr (*C var*), 392
drivers_MCX_MCXW_MCXW727C._i3c_device_info.hdrMode (*C var*), 393
drivers_MCX_MCXW_MCXW727C._i3c_device_info.maxReadLength (*C var*), 393
drivers_MCX_MCXW_MCXW727C._i3c_device_info.maxWriteLength (*C var*), 393
drivers_MCX_MCXW_MCXW727C._i3c_device_info.partNumber (*C var*), 393
drivers_MCX_MCXW_MCXW727C._i3c_device_info.staticAddr (*C var*), 392
drivers_MCX_MCXW_MCXW727C._i3c_device_info.vendorID (*C var*), 392
drivers_MCX_MCXW_MCXW727C._i3c_direction (*C enum*), 418
drivers_MCX_MCXW_MCXW727C._i3c_direction.kI3C_Read (*C enumerator*), 418
drivers_MCX_MCXW_MCXW727C._i3c_direction.kI3C_Write (*C enumerator*), 418
drivers_MCX_MCXW_MCXW727C._i3c_hdr_mode (*C enum*), 392
drivers_MCX_MCXW_MCXW727C._i3c_hdr_mode.kI3C_HDRModeDDR (*C enumerator*), 392
drivers_MCX_MCXW_MCXW727C._i3c_hdr_mode.kI3C_HDRModeNone (*C enumerator*), 392
drivers_MCX_MCXW_MCXW727C._i3c_hdr_mode.kI3C_HDRModeTSL (*C enumerator*), 392
drivers_MCX_MCXW_MCXW727C._i3c_hdr_mode.kI3C_HDRModeTSP (*C enumerator*), 392
drivers_MCX_MCXW_MCXW727C._i3c_ibi_response (*C enum*), 417
drivers_MCX_MCXW_MCXW727C._i3c_ibi_response.kI3C_IbiRespAck (*C enumerator*), 417
drivers_MCX_MCXW_MCXW727C._i3c_ibi_response.kI3C_IbiRespAckMandatory (*C enumerator*), 417
drivers_MCX_MCXW_MCXW727C._i3c_ibi_response.kI3C_IbiRespManual (*C enumerator*), 417
drivers_MCX_MCXW_MCXW727C._i3c_ibi_response.kI3C_IbiRespNack (*C enumerator*), 417
drivers_MCX_MCXW_MCXW727C._i3c_ibi_state (*C enum*), 417
drivers_MCX_MCXW_MCXW727C._i3c_ibi_state.kI3C_IbiAckNackPending (*C enumerator*), 418
drivers_MCX_MCXW_MCXW727C._i3c_ibi_state.kI3C_IbiDataBuffNeed (*C enumerator*), 418
drivers_MCX_MCXW_MCXW727C._i3c_ibi_state.kI3C_IbiReady (*C enumerator*), 417
drivers_MCX_MCXW_MCXW727C._i3c_ibi_type (*C enum*), 417
drivers_MCX_MCXW_MCXW727C._i3c_ibi_type.kI3C_IbiHotJoin (*C enumerator*), 417
drivers_MCX_MCXW_MCXW727C._i3c_ibi_type.kI3C_IbiMasterRequest (*C enumerator*), 417
drivers_MCX_MCXW_MCXW727C._i3c_ibi_type.kI3C_IbiNormal (*C enumerator*), 417

drivers_MCX_MCXW_MCXW727C._i3c_legacy_i2c_speed_mode (*C enum*), 419
drivers_MCX_MCXW_MCXW727C._i3c_legacy_i2c_speed_mode.kI3C_LegacyI2CFastMode (*C enumerator*), 420
drivers_MCX_MCXW_MCXW727C._i3c_legacy_i2c_speed_mode.kI3C_LegacyI2CFastModePlus (*C enumerator*), 420
drivers_MCX_MCXW_MCXW727C._i3c_legacy_i2c_speed_mode.kI3C_LegacyI2CStandardMode (*C enumerator*), 419
drivers_MCX_MCXW_MCXW727C._i3c_master_baudrate_config (*C struct*), 423
drivers_MCX_MCXW_MCXW727C._i3c_master_baudrate_config.busMode (*C var*), 423
drivers_MCX_MCXW_MCXW727C._i3c_master_baudrate_config.enableODHPP (*C var*), 424
drivers_MCX_MCXW_MCXW727C._i3c_master_baudrate_config.i2cBaud (*C var*), 423
drivers_MCX_MCXW_MCXW727C._i3c_master_baudrate_config.i3cOpenDrainBaud (*C var*), 424
drivers_MCX_MCXW_MCXW727C._i3c_master_baudrate_config.i3cPushPullBaud (*C var*), 423
drivers_MCX_MCXW_MCXW727C._i3c_master_bus_mode (*C enum*), 419
drivers_MCX_MCXW_MCXW727C._i3c_master_bus_mode.kI3C_LimitedBus (*C enumerator*), 419
drivers_MCX_MCXW_MCXW727C._i3c_master_bus_mode.kI3C_MixedFastBus (*C enumerator*), 419
drivers_MCX_MCXW_MCXW727C._i3c_master_bus_mode.kI3C_PureBus (*C enumerator*), 419
drivers_MCX_MCXW_MCXW727C._i3c_master_config (*C struct*), 424
drivers_MCX_MCXW_MCXW727C._i3c_master_config.baudRate_Hz (*C var*), 425
drivers_MCX_MCXW_MCXW727C._i3c_master_config.disableTimeout (*C var*), 425
drivers_MCX_MCXW_MCXW727C._i3c_master_config.enableMaster (*C var*), 425
drivers_MCX_MCXW_MCXW727C._i3c_master_config.enableOpenDrainHigh (*C var*), 425
drivers_MCX_MCXW_MCXW727C._i3c_master_config.enableOpenDrainStop (*C var*), 425
drivers_MCX_MCXW_MCXW727C._i3c_master_config.hKeep (*C var*), 425
drivers_MCX_MCXW_MCXW727C._i3c_master_config.restartSclDelay (*C var*), 425
drivers_MCX_MCXW_MCXW727C._i3c_master_config.startSclDelay (*C var*), 425
drivers_MCX_MCXW_MCXW727C._i3c_master_daa_baudrate (*C struct*), 423
drivers_MCX_MCXW_MCXW727C._i3c_master_daa_baudrate.i3cOpenDrainBaud (*C var*), 423
drivers_MCX_MCXW_MCXW727C._i3c_master_daa_baudrate.i3cPushPullBaud (*C var*), 423
drivers_MCX_MCXW_MCXW727C._i3c_master_daa_baudrate.sourceClock_Hz (*C var*), 423
drivers_MCX_MCXW_MCXW727C._i3c_master_enable (*C enum*), 415
drivers_MCX_MCXW_MCXW727C._i3c_master_enable.kI3C_MasterCapable (*C enumerator*), 416
drivers_MCX_MCXW_MCXW727C._i3c_master_enable.kI3C_MasterOff (*C enumerator*), 415
drivers_MCX_MCXW_MCXW727C._i3c_master_enable.kI3C_MasterOn (*C enumerator*), 416
drivers_MCX_MCXW_MCXW727C._i3c_master_error_flags (*C enum*), 414
drivers_MCX_MCXW_MCXW727C._i3c_master_error_flags.kI3C_MasterAllErrorFlags (*C enumerator*), 415
drivers_MCX_MCXW_MCXW727C._i3c_master_error_flags.kI3C_MasterErrorCrcFlag (*C enumerator*), 414
drivers_MCX_MCXW_MCXW727C._i3c_master_error_flags.kI3C_MasterErrorInvalidReqFlag (*C enumerator*), 415
drivers_MCX_MCXW_MCXW727C._i3c_master_error_flags.kI3C_MasterErrorMsgFlag (*C enumerator*), 415
drivers_MCX_MCXW_MCXW727C._i3c_master_error_flags.kI3C_MasterErrorNackFlag (*C enumerator*), 414
drivers_MCX_MCXW_MCXW727C._i3c_master_error_flags.kI3C_MasterErrorParityFlag (*C enumerator*), 414
drivers_MCX_MCXW_MCXW727C._i3c_master_error_flags.kI3C_MasterErrorReadFlag (*C enumerator*), 414
drivers_MCX_MCXW_MCXW727C._i3c_master_error_flags.kI3C_MasterErrorTimeoutFlag (*C enumerator*), 415
drivers_MCX_MCXW_MCXW727C._i3c_master_error_flags.kI3C_MasterErrorWriteAbortFlag (*C enumerator*), 414
drivers_MCX_MCXW_MCXW727C._i3c_master_error_flags.kI3C_MasterErrorWriteFlag (*C enumerator*), 414
drivers_MCX_MCXW_MCXW727C._i3c_master_flags (*C enum*), 413

drivers_MCX_MCXW_MCXW727C._i3c_master_flags.kI3C_MasterArbitrationWonFlag (*C enumerator*), 414
drivers_MCX_MCXW_MCXW727C._i3c_master_flags.kI3C_MasterBetweenFlag (*C enumerator*), 413
drivers_MCX_MCXW_MCXW727C._i3c_master_flags.kI3C_MasterClearFlags (*C enumerator*), 414
drivers_MCX_MCXW_MCXW727C._i3c_master_flags.kI3C_MasterCompleteFlag (*C enumerator*), 414
drivers_MCX_MCXW_MCXW727C._i3c_master_flags.kI3C_MasterControlDoneFlag (*C enumerator*), 414
drivers_MCX_MCXW_MCXW727C._i3c_master_flags.kI3C_MasterErrorFlag (*C enumerator*), 414
drivers_MCX_MCXW_MCXW727C._i3c_master_flags.kI3C_MasterNackDetectFlag (*C enumerator*), 413
drivers_MCX_MCXW_MCXW727C._i3c_master_flags.kI3C_MasterRxReadyFlag (*C enumerator*), 414
drivers_MCX_MCXW_MCXW727C._i3c_master_flags.kI3C_MasterSlave2MasterFlag (*C enumerator*), 414
drivers_MCX_MCXW_MCXW727C._i3c_master_flags.kI3C_MasterSlaveStartFlag (*C enumerator*), 413
drivers_MCX_MCXW_MCXW727C._i3c_master_flags.kI3C_MasterTxReadyFlag (*C enumerator*), 414
drivers_MCX_MCXW_MCXW727C._i3c_master_handle (*C struct*), 426
drivers_MCX_MCXW_MCXW727C._i3c_master_handle.callback (*C var*), 427
drivers_MCX_MCXW_MCXW727C._i3c_master_handle.ibiAddress (*C var*), 427
drivers_MCX_MCXW_MCXW727C._i3c_master_handle.ibiBuff (*C var*), 427
drivers_MCX_MCXW_MCXW727C._i3c_master_handle.ibiPayloadSize (*C var*), 427
drivers_MCX_MCXW_MCXW727C._i3c_master_handle.ibiType (*C var*), 427
drivers_MCX_MCXW_MCXW727C._i3c_master_handle.remainingBytes (*C var*), 426
drivers_MCX_MCXW_MCXW727C._i3c_master_handle.rxTermOps (*C var*), 426
drivers_MCX_MCXW_MCXW727C._i3c_master_handle.state (*C var*), 426
drivers_MCX_MCXW_MCXW727C._i3c_master_handle.transfer (*C var*), 426
drivers_MCX_MCXW_MCXW727C._i3c_master_handle.userData (*C var*), 427
drivers_MCX_MCXW_MCXW727C._i3c_master_hkeep (*C enum*), 416
drivers_MCX_MCXW_MCXW727C._i3c_master_hkeep.kI3C_MasterHighKeeperNone (*C enumerator*), 416
drivers_MCX_MCXW_MCXW727C._i3c_master_hkeep.kI3C_MasterHighKeeperWiredIn (*C enumerator*), 416
drivers_MCX_MCXW_MCXW727C._i3c_master_hkeep.kI3C_MasterPassiveSDA (*C enumerator*), 416
drivers_MCX_MCXW_MCXW727C._i3c_master_hkeep.kI3C_MasterPassiveSDASCL (*C enumerator*), 416
drivers_MCX_MCXW_MCXW727C._i3c_master_src_clk (*C enum*), 420
drivers_MCX_MCXW_MCXW727C._i3c_master_src_clk.kI3C_MasterSrcClk24M (*C enumerator*), 420
drivers_MCX_MCXW_MCXW727C._i3c_master_src_clk.kI3C_MasterSrcClk25M (*C enumerator*), 420
drivers_MCX_MCXW_MCXW727C._i3c_master_state (*C enum*), 415
drivers_MCX_MCXW_MCXW727C._i3c_master_state.kI3C_MasterStateDaa (*C enumerator*), 415
drivers_MCX_MCXW_MCXW727C._i3c_master_state.kI3C_MasterStateDdr (*C enumerator*), 415
drivers_MCX_MCXW_MCXW727C._i3c_master_state.kI3C_MasterStateIbiAck (*C enumerator*), 415
drivers_MCX_MCXW_MCXW727C._i3c_master_state.kI3C_MasterStateIbiRcv (*C enumerator*), 415
drivers_MCX_MCXW_MCXW727C._i3c_master_state.kI3C_MasterStateIdle (*C enumerator*), 415
drivers_MCX_MCXW_MCXW727C._i3c_master_state.kI3C_MasterStateMsgSdr (*C enumerator*), 415
drivers_MCX_MCXW_MCXW727C._i3c_master_state.kI3C_MasterStateNormAct (*C enumerator*), 415

drivers_MCX_MCXW_MCXW727C._i3c_master_state.kI3C_MasterStateSlvReq (*C enumerator*), 415
drivers_MCX_MCXW_MCXW727C._i3c_master_transfer (*C struct*), 425
drivers_MCX_MCXW_MCXW727C._i3c_master_transfer_callback (*C struct*), 425
drivers_MCX_MCXW_MCXW727C._i3c_master_transfer_callback.ibiCallback (*C var*), 425
drivers_MCX_MCXW_MCXW727C._i3c_master_transfer_callback.slave2Master (*C var*), 425
drivers_MCX_MCXW_MCXW727C._i3c_master_transfer_callback.transferComplete (*C var*), 425
drivers_MCX_MCXW_MCXW727C._i3c_master_transfer_flags (*C enum*), 420
drivers_MCX_MCXW_MCXW727C._i3c_master_transfer_flags.kI3C_TransferDefaultFlag (*C enumerator*), 420
drivers_MCX_MCXW_MCXW727C._i3c_master_transfer_flags.kI3C_TransferDisableRxTermFlag (*C enumerator*), 420
drivers_MCX_MCXW_MCXW727C._i3c_master_transfer_flags.kI3C_TransferNoStartFlag (*C enumerator*), 420
drivers_MCX_MCXW_MCXW727C._i3c_master_transfer_flags.kI3C_TransferNoStopFlag (*C enumerator*), 420
drivers_MCX_MCXW_MCXW727C._i3c_master_transfer_flags.kI3C_TransferRepeatedStartFlag (*C enumerator*), 420
drivers_MCX_MCXW_MCXW727C._i3c_master_transfer_flags.kI3C_TransferRxAutoTermFlag (*C enumerator*), 420
drivers_MCX_MCXW_MCXW727C._i3c_master_transfer_flags.kI3C_TransferStartWithBroadcastAddr (*C enumerator*), 420
drivers_MCX_MCXW_MCXW727C._i3c_master_transfer_flags.kI3C_TransferWordsFlag (*C enumerator*), 420
drivers_MCX_MCXW_MCXW727C._i3c_master_transfer.busType (*C var*), 426
drivers_MCX_MCXW_MCXW727C._i3c_master_transfer.data (*C var*), 426
drivers_MCX_MCXW_MCXW727C._i3c_master_transfer.dataSize (*C var*), 426
drivers_MCX_MCXW_MCXW727C._i3c_master_transfer.direction (*C var*), 426
drivers_MCX_MCXW_MCXW727C._i3c_master_transfer.flags (*C var*), 426
drivers_MCX_MCXW_MCXW727C._i3c_master_transfer.ibiResponse (*C var*), 426
drivers_MCX_MCXW_MCXW727C._i3c_master_transfer.slaveAddress (*C var*), 426
drivers_MCX_MCXW_MCXW727C._i3c_master_transfer.subaddress (*C var*), 426
drivers_MCX_MCXW_MCXW727C._i3c_master_transfer.subaddressSize (*C var*), 426
drivers_MCX_MCXW_MCXW727C._i3c_register_ibi_addr (*C struct*), 422
drivers_MCX_MCXW_MCXW727C._i3c_register_ibi_addr.address (*C var*), 422
drivers_MCX_MCXW_MCXW727C._i3c_register_ibi_addr.i3cFastStart (*C var*), 422
drivers_MCX_MCXW_MCXW727C._i3c_register_ibi_addr.ibiHasPayload (*C var*), 423
drivers_MCX_MCXW_MCXW727C._i3c_rx_term_ops (*C enum*), 419
drivers_MCX_MCXW_MCXW727C._i3c_rx_term_ops.kI3C_RxAutoTerm (*C enumerator*), 419
drivers_MCX_MCXW_MCXW727C._i3c_rx_term_ops.kI3C_RxTermDisable (*C enumerator*), 419
drivers_MCX_MCXW_MCXW727C._i3c_rx_term_ops.kI3C_RxTermLastByte (*C enumerator*), 419
drivers_MCX_MCXW_MCXW727C._i3c_rx_trigger_level (*C enum*), 418
drivers_MCX_MCXW_MCXW727C._i3c_rx_trigger_level.kI3C_RxTriggerOnNotEmpty (*C enumerator*), 418
drivers_MCX_MCXW_MCXW727C._i3c_rx_trigger_level.kI3C_RxTriggerUntilOneHalfOrMore (*C enumerator*), 418
drivers_MCX_MCXW_MCXW727C._i3c_rx_trigger_level.kI3C_RxTriggerUntilOneQuarterOrMore (*C enumerator*), 418
drivers_MCX_MCXW_MCXW727C._i3c_rx_trigger_level.kI3C_RxTriggerUntilThreeQuarterOrMore (*C enumerator*), 418
drivers_MCX_MCXW_MCXW727C._i3c_slave_activity_state (*C enum*), 437
drivers_MCX_MCXW_MCXW727C._i3c_slave_activity_state.kI3C_SlaveLatency1Ms (*C enumerator*), 437
drivers_MCX_MCXW_MCXW727C._i3c_slave_activity_state.kI3C_SlaveLatency10S (*C enumerator*), 437
drivers_MCX_MCXW_MCXW727C._i3c_slave_activity_state.kI3C_SlaveLatency100Ms (*C enumerator*), 437

drivers_MCX_MCXW_MCXW727C._i3c_slave_activity_state.kI3C_SlaveNoLatency (*C enumerator*), 437
drivers_MCX_MCXW_MCXW727C._i3c_slave_config (*C struct*), 439
drivers_MCX_MCXW_MCXW727C._i3c_slave_config.bcr (*C var*), 439
drivers_MCX_MCXW_MCXW727C._i3c_slave_config.dcr (*C var*), 439
drivers_MCX_MCXW_MCXW727C._i3c_slave_config.enableSlave (*C var*), 439
drivers_MCX_MCXW_MCXW727C._i3c_slave_config.hdrMode (*C var*), 439
drivers_MCX_MCXW_MCXW727C._i3c_slave_config.ignoreS0S1Error (*C var*), 439
drivers_MCX_MCXW_MCXW727C._i3c_slave_config.matchSlaveStartStop (*C var*), 440
drivers_MCX_MCXW_MCXW727C._i3c_slave_config.maxReadLength (*C var*), 440
drivers_MCX_MCXW_MCXW727C._i3c_slave_config.maxWriteLength (*C var*), 440
drivers_MCX_MCXW_MCXW727C._i3c_slave_config.nakAllRequest (*C var*), 439
drivers_MCX_MCXW_MCXW727C._i3c_slave_config.offline (*C var*), 440
drivers_MCX_MCXW_MCXW727C._i3c_slave_config.partNumber (*C var*), 439
drivers_MCX_MCXW_MCXW727C._i3c_slave_config.staticAddr (*C var*), 439
drivers_MCX_MCXW_MCXW727C._i3c_slave_config.vendorID (*C var*), 439
drivers_MCX_MCXW_MCXW727C._i3c_slave_error_flags (*C enum*), 435
drivers_MCX_MCXW_MCXW727C._i3c_slave_error_flags.kI3C_SlaveErrorHdrCRCFlag (*C enumerator*), 436
drivers_MCX_MCXW_MCXW727C._i3c_slave_error_flags.kI3C_SlaveErrorHdrParityFlag (*C enumerator*), 436
drivers_MCX_MCXW_MCXW727C._i3c_slave_error_flags.kI3C_SlaveErrorInvalidStartFlag (*C enumerator*), 436
drivers_MCX_MCXW_MCXW727C._i3c_slave_error_flags.kI3C_SlaveErrorOverreadFlag (*C enumerator*), 436
drivers_MCX_MCXW_MCXW727C._i3c_slave_error_flags.kI3C_SlaveErrorOverrunFlag (*C enumerator*), 436
drivers_MCX_MCXW_MCXW727C._i3c_slave_error_flags.kI3C_SlaveErrorOverwriteFlag (*C enumerator*), 436
drivers_MCX_MCXW_MCXW727C._i3c_slave_error_flags.kI3C_SlaveErrorS0S1Flag (*C enumerator*), 436
drivers_MCX_MCXW_MCXW727C._i3c_slave_error_flags.kI3C_SlaveErrorSdrParityFlag (*C enumerator*), 436
drivers_MCX_MCXW_MCXW727C._i3c_slave_error_flags.kI3C_SlaveErrorTermFlag (*C enumerator*), 436
drivers_MCX_MCXW_MCXW727C._i3c_slave_error_flags.kI3C_SlaveErrorUnderrunFlag (*C enumerator*), 436
drivers_MCX_MCXW_MCXW727C._i3c_slave_error_flags.kI3C_SlaveErrorUnderrunNakFlag (*C enumerator*), 436
drivers_MCX_MCXW_MCXW727C._i3c_slave_event (*C enum*), 436
drivers_MCX_MCXW_MCXW727C._i3c_slave_event.kI3C_SlaveEventHotJoinReq (*C enumerator*), 437
drivers_MCX_MCXW_MCXW727C._i3c_slave_event.kI3C_SlaveEventIBI (*C enumerator*), 436
drivers_MCX_MCXW_MCXW727C._i3c_slave_event.kI3C_SlaveEventMasterReq (*C enumerator*), 437
drivers_MCX_MCXW_MCXW727C._i3c_slave_event.kI3C_SlaveEventNormal (*C enumerator*), 436
drivers_MCX_MCXW_MCXW727C._i3c_slave_flags (*C enum*), 434
drivers_MCX_MCXW_MCXW727C._i3c_slave_flags.kI3C_SlaveAllIrqFlags (*C enumerator*), 435
drivers_MCX_MCXW_MCXW727C._i3c_slave_flags.kI3C_SlaveBusDAAFlag (*C enumerator*), 434
drivers_MCX_MCXW_MCXW727C._i3c_slave_flags.kI3C_SlaveBusHDRModeFlag (*C enumerator*), 434
drivers_MCX_MCXW_MCXW727C._i3c_slave_flags.kI3C_SlaveBusStartFlag (*C enumerator*), 434
drivers_MCX_MCXW_MCXW727C._i3c_slave_flags.kI3C_SlaveBusStopFlag (*C enumerator*), 435
drivers_MCX_MCXW_MCXW727C._i3c_slave_flags.kI3C_SlaveCCCHandledFlag (*C enumerator*), 435
drivers_MCX_MCXW_MCXW727C._i3c_slave_flags.kI3C_SlaveClearFlags (*C enumerator*), 435
drivers_MCX_MCXW_MCXW727C._i3c_slave_flags.kI3C_SlaveDynamicAddrChangedFlag (*C enumerator*), 435

drivers_MCX_MCXW_MCXW727C._i3c_slave_flags.kI3C_SlaveErrorFlag (*C enumerator*), 435
drivers_MCX_MCXW_MCXW727C._i3c_slave_flags.kI3C_SlaveEventSentFlag (*C enumerator*), 435
drivers_MCX_MCXW_MCXW727C._i3c_slave_flags.kI3C_SlaveHDRCommandMatchFlag (*C enumerator*), 435
drivers_MCX_MCXW_MCXW727C._i3c_slave_flags.kI3C_SlaveHotJoinDisabledFlag (*C enumerator*), 435
drivers_MCX_MCXW_MCXW727C._i3c_slave_flags.kI3C_SlaveIbiDisableFlag (*C enumerator*), 435
drivers_MCX_MCXW_MCXW727C._i3c_slave_flags.kI3C_SlaveMasterRequestDisabledFlag (*C enumerator*), 435
drivers_MCX_MCXW_MCXW727C._i3c_slave_flags.kI3C_SlaveMatchedFlag (*C enumerator*), 435
drivers_MCX_MCXW_MCXW727C._i3c_slave_flags.kI3C_SlaveMessageFlag (*C enumerator*), 434
drivers_MCX_MCXW_MCXW727C._i3c_slave_flags.kI3C_SlaveNotStopFlag (*C enumerator*), 434
drivers_MCX_MCXW_MCXW727C._i3c_slave_flags.kI3C_SlaveReceivedCCCFlag (*C enumerator*), 435
drivers_MCX_MCXW_MCXW727C._i3c_slave_flags.kI3C_SlaveRequiredReadFlag (*C enumerator*), 434
drivers_MCX_MCXW_MCXW727C._i3c_slave_flags.kI3C_SlaveRequiredWriteFlag (*C enumerator*), 434
drivers_MCX_MCXW_MCXW727C._i3c_slave_flags.kI3C_SlaveRxReadyFlag (*C enumerator*), 435
drivers_MCX_MCXW_MCXW727C._i3c_slave_flags.kI3C_SlaveTxReadyFlag (*C enumerator*), 435
drivers_MCX_MCXW_MCXW727C._i3c_slave_handle (*C struct*), 440
drivers_MCX_MCXW_MCXW727C._i3c_slave_handle.callback (*C var*), 441
drivers_MCX_MCXW_MCXW727C._i3c_slave_handle.eventMask (*C var*), 441
drivers_MCX_MCXW_MCXW727C._i3c_slave_handle.isBusy (*C var*), 441
drivers_MCX_MCXW_MCXW727C._i3c_slave_handle.transfer (*C var*), 441
drivers_MCX_MCXW_MCXW727C._i3c_slave_handle.transferredCount (*C var*), 441
drivers_MCX_MCXW_MCXW727C._i3c_slave_handle.txFifoSize (*C var*), 441
drivers_MCX_MCXW_MCXW727C._i3c_slave_handle.userData (*C var*), 441
drivers_MCX_MCXW_MCXW727C._i3c_slave_handle.wasTransmit (*C var*), 441
drivers_MCX_MCXW_MCXW727C._i3c_slave_transfer (*C struct*), 440
drivers_MCX_MCXW_MCXW727C._i3c_slave_transfer_event (*C enum*), 437
drivers_MCX_MCXW_MCXW727C._i3c_slave_transfer_event.kI3C_SlaveAddressMatchEvent (*C enumerator*), 437
drivers_MCX_MCXW_MCXW727C._i3c_slave_transfer_event.kI3C_SlaveAllEvents (*C enumerator*), 438
drivers_MCX_MCXW_MCXW727C._i3c_slave_transfer_event.kI3C_SlaveCompletionEvent (*C enumerator*), 438
drivers_MCX_MCXW_MCXW727C._i3c_slave_transfer_event.kI3C_SlaveHDRCommandMatchEvent (*C enumerator*), 438
drivers_MCX_MCXW_MCXW727C._i3c_slave_transfer_event.kI3C_SlaveReceivedCCCEvent (*C enumerator*), 438
drivers_MCX_MCXW_MCXW727C._i3c_slave_transfer_event.kI3C_SlaveReceiveEvent (*C enumerator*), 437
drivers_MCX_MCXW_MCXW727C._i3c_slave_transfer_event.kI3C_SlaveRequestSentEvent (*C enumerator*), 438
drivers_MCX_MCXW_MCXW727C._i3c_slave_transfer_event.kI3C_SlaveRequiredTransmitEvent (*C enumerator*), 437
drivers_MCX_MCXW_MCXW727C._i3c_slave_transfer_event.kI3C_SlaveStartEvent (*C enumerator*), 437
drivers_MCX_MCXW_MCXW727C._i3c_slave_transfer_event.kI3C_SlaveTransmitEvent (*C enumerator*), 437
drivers_MCX_MCXW_MCXW727C._i3c_slave_transfer.completionStatus (*C var*), 440
drivers_MCX_MCXW_MCXW727C._i3c_slave_transfer.event (*C var*), 440
drivers_MCX_MCXW_MCXW727C._i3c_slave_transfer.rxData (*C var*), 440
drivers_MCX_MCXW_MCXW727C._i3c_slave_transfer.rxDataSize (*C var*), 440
drivers_MCX_MCXW_MCXW727C._i3c_slave_transfer.transferredCount (*C var*), 440

drivers_MCX_MCXW_MCXW727C._i3c_slave_transfer.txData (*C var*), 440
drivers_MCX_MCXW_MCXW727C._i3c_slave_transfer.txDataSize (*C var*), 440
drivers_MCX_MCXW_MCXW727C._i3c_start_scl_delay (*C enum*), 419
drivers_MCX_MCXW_MCXW727C._i3c_start_scl_delay.kI3C_IncreaseSclHalfPeriod (*C enumerator*), 419
drivers_MCX_MCXW_MCXW727C._i3c_start_scl_delay.kI3C_IncreaseSclOneAndHalfPeriod (*C enumerator*), 419
drivers_MCX_MCXW_MCXW727C._i3c_start_scl_delay.kI3C_IncreaseSclOnePeriod (*C enumerator*), 419
drivers_MCX_MCXW_MCXW727C._i3c_start_scl_delay.kI3C_NoDelay (*C enumerator*), 419
drivers_MCX_MCXW_MCXW727C._i3c_timing (*C struct*), 424
drivers_MCX_MCXW_MCXW727C._i3c_timing.actualI2cBaud (*C var*), 424
drivers_MCX_MCXW_MCXW727C._i3c_timing.actualOdBaud (*C var*), 424
drivers_MCX_MCXW_MCXW727C._i3c_timing.actualPpBaud (*C var*), 424
drivers_MCX_MCXW_MCXW727C._i3c_timing.i2cBaud (*C var*), 424
drivers_MCX_MCXW_MCXW727C._i3c_timing.odBaud (*C var*), 424
drivers_MCX_MCXW_MCXW727C._i3c_timing.odHighPP (*C var*), 424
drivers_MCX_MCXW_MCXW727C._i3c_timing.ppBaud (*C var*), 424
drivers_MCX_MCXW_MCXW727C._i3c_timing.ppLow (*C var*), 424
drivers_MCX_MCXW_MCXW727C._i3c_tx_trigger_level (*C enum*), 418
drivers_MCX_MCXW_MCXW727C._i3c_tx_trigger_level.kI3C_TxTriggerOnEmpty (*C enumerator*), 418
drivers_MCX_MCXW_MCXW727C._i3c_tx_trigger_level.kI3C_TxTriggerUntilOneHalfOrLess (*C enumerator*), 418
drivers_MCX_MCXW_MCXW727C._i3c_tx_trigger_level.kI3C_TxTriggerUntilOneLessThanFull (*C enumerator*), 418
drivers_MCX_MCXW_MCXW727C._i3c_tx_trigger_level.kI3C_TxTriggerUntilOneQuarterOrLess (*C enumerator*), 418
drivers_MCX_MCXW_MCXW727C._imu_interrupts (*C enum*), 446
drivers_MCX_MCXW_MCXW727C._imu_interrupts.kIMU_RxMsgReadyInterrupt (*C enumerator*), 446
drivers_MCX_MCXW_MCXW727C._imu_interrupts.kIMU_TxFifoSpaceAvailableInterrupt (*C enumerator*), 446
drivers_MCX_MCXW_MCXW727C._imu_status_flags (*C enum*), 446
drivers_MCX_MCXW_MCXW727C._imu_status_flags.kIMU_RxFifoAlmostFull (*C enumerator*), 446
drivers_MCX_MCXW_MCXW727C._imu_status_flags.kIMU_RxFifoEmpty (*C enumerator*), 446
drivers_MCX_MCXW_MCXW727C._imu_status_flags.kIMU_RxFifoFull (*C enumerator*), 446
drivers_MCX_MCXW_MCXW727C._imu_status_flags.kIMU_RxFifoLocked (*C enumerator*), 446
drivers_MCX_MCXW_MCXW727C._imu_status_flags.kIMU_TxFifoAlmostFull (*C enumerator*), 446
drivers_MCX_MCXW_MCXW727C._imu_status_flags.kIMU_TxFifoEmpty (*C enumerator*), 446
drivers_MCX_MCXW_MCXW727C._imu_status_flags.kIMU_TxFifoFull (*C enumerator*), 446
drivers_MCX_MCXW_MCXW727C._imu_status_flags.kIMU_TxFifoLocked (*C enumerator*), 446
drivers_MCX_MCXW_MCXW727C._lpadc_calibration_value (*C struct*), 486
drivers_MCX_MCXW_MCXW727C._lpadc_conversion_average_mode (*C enum*), 469
drivers_MCX_MCXW_MCXW727C._lpadc_conversion_average_mode.kLPADC_ConversionAverage1 (*C enumerator*), 469
drivers_MCX_MCXW_MCXW727C._lpadc_conversion_average_mode.kLPADC_ConversionAverage2 (*C enumerator*), 469
drivers_MCX_MCXW_MCXW727C._lpadc_conversion_average_mode.kLPADC_ConversionAverage4 (*C enumerator*), 469
drivers_MCX_MCXW_MCXW727C._lpadc_conversion_average_mode.kLPADC_ConversionAverage8 (*C enumerator*), 469
drivers_MCX_MCXW_MCXW727C._lpadc_conversion_average_mode.kLPADC_ConversionAverage16 (*C enumerator*), 469
drivers_MCX_MCXW_MCXW727C._lpadc_conversion_average_mode.kLPADC_ConversionAverage32 (*C enumerator*), 469

drivers_MCX_MCXW_MCXW727C._lpadc_conversion_average_mode.kLPADC_ConversionAverage64 (*C enumerator*), 469
drivers_MCX_MCXW_MCXW727C._lpadc_conversion_average_mode.kLPADC_ConversionAverage128 (*C enumerator*), 470
drivers_MCX_MCXW_MCXW727C._lpadc_conversion_average_mode.kLPADC_ConversionAverageMax (*C enumerator*), 470
drivers_MCX_MCXW_MCXW727C._lpadc_conversion_resolution_mode (*C enum*), 469
drivers_MCX_MCXW_MCXW727C._lpadc_conversion_resolution_mode.kLPADC_ConversionResolutionHigh (*C enumerator*), 469
drivers_MCX_MCXW_MCXW727C._lpadc_conversion_resolution_mode.kLPADC_ConversionResolutionStandard (*C enumerator*), 469
drivers_MCX_MCXW_MCXW727C._lpadc_hardware_average_mode (*C enum*), 467
drivers_MCX_MCXW_MCXW727C._lpadc_hardware_average_mode.kLPADC_HardwareAverageCount1 (*C enumerator*), 467
drivers_MCX_MCXW_MCXW727C._lpadc_hardware_average_mode.kLPADC_HardwareAverageCount2 (*C enumerator*), 467
drivers_MCX_MCXW_MCXW727C._lpadc_hardware_average_mode.kLPADC_HardwareAverageCount4 (*C enumerator*), 467
drivers_MCX_MCXW_MCXW727C._lpadc_hardware_average_mode.kLPADC_HardwareAverageCount8 (*C enumerator*), 467
drivers_MCX_MCXW_MCXW727C._lpadc_hardware_average_mode.kLPADC_HardwareAverageCount16 (*C enumerator*), 467
drivers_MCX_MCXW_MCXW727C._lpadc_hardware_average_mode.kLPADC_HardwareAverageCount32 (*C enumerator*), 467
drivers_MCX_MCXW_MCXW727C._lpadc_hardware_average_mode.kLPADC_HardwareAverageCount64 (*C enumerator*), 467
drivers_MCX_MCXW_MCXW727C._lpadc_hardware_average_mode.kLPADC_HardwareAverageCount128 (*C enumerator*), 468
drivers_MCX_MCXW_MCXW727C._lpadc_hardware_compare_mode (*C enum*), 468
drivers_MCX_MCXW_MCXW727C._lpadc_hardware_compare_mode.kLPADC_HardwareCompareDisabled (*C enumerator*), 468
drivers_MCX_MCXW_MCXW727C._lpadc_hardware_compare_mode.kLPADC_HardwareCompareRepeatUntilTru (*C enumerator*), 469
drivers_MCX_MCXW_MCXW727C._lpadc_hardware_compare_mode.kLPADC_HardwareCompareStoreOnTrue (*C enumerator*), 468
drivers_MCX_MCXW_MCXW727C._lpadc_interrupt_enable (*C enum*), 465
drivers_MCX_MCXW_MCXW727C._lpadc_interrupt_enable.kLPADC_FIFO0WatermarkInterruptEnable (*C enumerator*), 465
drivers_MCX_MCXW_MCXW727C._lpadc_interrupt_enable.kLPADC_FIFOWatermarkInterruptEnable (*C enumerator*), 465
drivers_MCX_MCXW_MCXW727C._lpadc_interrupt_enable.kLPADC_ResultFIFO0OverflowInterruptEnable (*C enumerator*), 465
drivers_MCX_MCXW_MCXW727C._lpadc_interrupt_enable.kLPADC_ResultFIFOOverflowInterruptEnable (*C enumerator*), 465
drivers_MCX_MCXW_MCXW727C._lpadc_interrupt_enable.kLPADC_Trigger0CompletionInterruptEnable (*C enumerator*), 465
drivers_MCX_MCXW_MCXW727C._lpadc_interrupt_enable.kLPADC_Trigger1CompletionInterruptEnable (*C enumerator*), 466
drivers_MCX_MCXW_MCXW727C._lpadc_interrupt_enable.kLPADC_TriggerExceptionInterruptEnable (*C enumerator*), 465
drivers_MCX_MCXW_MCXW727C._lpadc_offset_calibration_mode (*C enum*), 470
drivers_MCX_MCXW_MCXW727C._lpadc_offset_calibration_mode.kLPADC_OffsetCalibration12bitMode (*C enumerator*), 470
drivers_MCX_MCXW_MCXW727C._lpadc_offset_calibration_mode.kLPADC_OffsetCalibration16bitMode (*C enumerator*), 470
drivers_MCX_MCXW_MCXW727C._lpadc_power_level_mode (*C enum*), 470
drivers_MCX_MCXW_MCXW727C._lpadc_power_level_mode.kLPADC_PowerLevelAlt1 (*C enumerator*), 470

drivers_MCX_MCXW_MCXW727C._lpadc_power_level_mode.kLPADC_PowerLevelAlt2 (*C enumerator*), 470
drivers_MCX_MCXW_MCXW727C._lpadc_power_level_mode.kLPADC_PowerLevelAlt3 (*C enumerator*), 470
drivers_MCX_MCXW_MCXW727C._lpadc_power_level_mode.kLPADC_PowerLevelAlt4 (*C enumerator*), 470
drivers_MCX_MCXW_MCXW727C._lpadc_reference_voltage_mode (*C enum*), 470
drivers_MCX_MCXW_MCXW727C._lpadc_reference_voltage_mode.kLPADC_ReferenceVoltageAlt1 (*C enumerator*), 470
drivers_MCX_MCXW_MCXW727C._lpadc_reference_voltage_mode.kLPADC_ReferenceVoltageAlt2 (*C enumerator*), 470
drivers_MCX_MCXW_MCXW727C._lpadc_reference_voltage_mode.kLPADC_ReferenceVoltageAlt3 (*C enumerator*), 470
drivers_MCX_MCXW_MCXW727C._lpadc_sample_channel_mode (*C enum*), 466
drivers_MCX_MCXW_MCXW727C._lpadc_sample_channel_mode.kLPADC_SampleChannelDiffBothSide (*C enumerator*), 467
drivers_MCX_MCXW_MCXW727C._lpadc_sample_channel_mode.kLPADC_SampleChannelDiffBothSideAB (*C enumerator*), 467
drivers_MCX_MCXW_MCXW727C._lpadc_sample_channel_mode.kLPADC_SampleChannelDiffBothSideBA (*C enumerator*), 467
drivers_MCX_MCXW_MCXW727C._lpadc_sample_channel_mode.kLPADC_SampleChannelDualSingleEndBothSi (*C enumerator*), 467
drivers_MCX_MCXW_MCXW727C._lpadc_sample_channel_mode.kLPADC_SampleChannelSingleEndSideA (*C enumerator*), 466
drivers_MCX_MCXW_MCXW727C._lpadc_sample_channel_mode.kLPADC_SampleChannelSingleEndSideB (*C enumerator*), 467
drivers_MCX_MCXW_MCXW727C._lpadc_sample_scale_mode (*C enum*), 466
drivers_MCX_MCXW_MCXW727C._lpadc_sample_scale_mode.kLPADC_SampleFullScale (*C enumerator*), 466
drivers_MCX_MCXW_MCXW727C._lpadc_sample_scale_mode.kLPADC_SamplePartScale (*C enumerator*), 466
drivers_MCX_MCXW_MCXW727C._lpadc_sample_time_mode (*C enum*), 468
drivers_MCX_MCXW_MCXW727C._lpadc_sample_time_mode.kLPADC_SampleTimeADCK3 (*C enumerator*), 468
drivers_MCX_MCXW_MCXW727C._lpadc_sample_time_mode.kLPADC_SampleTimeADCK5 (*C enumerator*), 468
drivers_MCX_MCXW_MCXW727C._lpadc_sample_time_mode.kLPADC_SampleTimeADCK7 (*C enumerator*), 468
drivers_MCX_MCXW_MCXW727C._lpadc_sample_time_mode.kLPADC_SampleTimeADCK11 (*C enumerator*), 468
drivers_MCX_MCXW_MCXW727C._lpadc_sample_time_mode.kLPADC_SampleTimeADCK19 (*C enumerator*), 468
drivers_MCX_MCXW_MCXW727C._lpadc_sample_time_mode.kLPADC_SampleTimeADCK35 (*C enumerator*), 468
drivers_MCX_MCXW_MCXW727C._lpadc_sample_time_mode.kLPADC_SampleTimeADCK67 (*C enumerator*), 468
drivers_MCX_MCXW_MCXW727C._lpadc_sample_time_mode.kLPADC_SampleTimeADCK131 (*C enumerator*), 468
drivers_MCX_MCXW_MCXW727C._lpadc_status_flags (*C enum*), 464
drivers_MCX_MCXW_MCXW727C._lpadc_status_flags.kLPADC_ActiveFlag (*C enumerator*), 465
drivers_MCX_MCXW_MCXW727C._lpadc_status_flags.kLPADC_CalibrationReadyFlag (*C enumerator*), 465
drivers_MCX_MCXW_MCXW727C._lpadc_status_flags.kLPADC_ResultFIFO0OverflowFlag (*C enumerator*), 464
drivers_MCX_MCXW_MCXW727C._lpadc_status_flags.kLPADC_ResultFIFO0ReadyFlag (*C enumerator*), 465
drivers_MCX_MCXW_MCXW727C._lpadc_status_flags.kLPADC_ResultFIFOOverflowFlag (*C enumerator*), 465

drivers_MCX_MCXW_MCXW727C._lpadc_status_flags.kLPADC_ResultFIFOReadyFlag (*C enumerator*), 465
drivers_MCX_MCXW_MCXW727C._lpadc_status_flags.kLPADC_TriggerCompletionFlag (*C enumerator*), 465
drivers_MCX_MCXW_MCXW727C._lpadc_status_flags.kLPADC_TriggerExceptionFlag (*C enumerator*), 465
drivers_MCX_MCXW_MCXW727C._lpadc_trigger_priority_policy (*C enum*), 471
drivers_MCX_MCXW_MCXW727C._lpadc_trigger_priority_policy.kLPADC_ConvPreemptImmediatelyAutoRestar (*C enumerator*), 471
drivers_MCX_MCXW_MCXW727C._lpadc_trigger_priority_policy.kLPADC_ConvPreemptImmediatelyAutoResum (*C enumerator*), 471
drivers_MCX_MCXW_MCXW727C._lpadc_trigger_priority_policy.kLPADC_ConvPreemptImmediatelyNotAutoRe (*C enumerator*), 471
drivers_MCX_MCXW_MCXW727C._lpadc_trigger_priority_policy.kLPADC_ConvPreemptSoftlyAutoRestarted (*C enumerator*), 471
drivers_MCX_MCXW_MCXW727C._lpadc_trigger_priority_policy.kLPADC_ConvPreemptSoftlyAutoResumed (*C enumerator*), 471
drivers_MCX_MCXW_MCXW727C._lpadc_trigger_priority_policy.kLPADC_ConvPreemptSoftlyNotAutoResumed (*C enumerator*), 471
drivers_MCX_MCXW_MCXW727C._lpadc_trigger_priority_policy.kLPADC_TriggerPriorityExceptionDisabled (*C enumerator*), 472
drivers_MCX_MCXW_MCXW727C._lpadc_trigger_priority_policy.kLPADC_TriggerPriorityPreemptImmediately (*C enumerator*), 471
drivers_MCX_MCXW_MCXW727C._lpadc_trigger_priority_policy.kLPADC_TriggerPriorityPreemptSoftly (*C enumerator*), 472
drivers_MCX_MCXW_MCXW727C._lpadc_trigger_status_flags (*C enum*), 466
drivers_MCX_MCXW_MCXW727C._lpadc_trigger_status_flags.kLPADC_Trigger0CompletedFlag (*C enumerator*), 466
drivers_MCX_MCXW_MCXW727C._lpadc_trigger_status_flags.kLPADC_Trigger0InterruptedFlag (*C enumerator*), 466
drivers_MCX_MCXW_MCXW727C._lpadc_trigger_status_flags.kLPADC_Trigger1CompletedFlag (*C enumerator*), 466
drivers_MCX_MCXW_MCXW727C._lpadc_trigger_status_flags.kLPADC_Trigger1InterruptedFlag (*C enumerator*), 466
drivers_MCX_MCXW_MCXW727C._lpadc_tune_value (*C enum*), 472
drivers_MCX_MCXW_MCXW727C._lpadc_tune_value.kLPADC_TuneValue0 (*C enumerator*), 472
drivers_MCX_MCXW_MCXW727C._lpadc_tune_value.kLPADC_TuneValue1 (*C enumerator*), 472
drivers_MCX_MCXW_MCXW727C._lpadc_tune_value.kLPADC_TuneValue2 (*C enumerator*), 472
drivers_MCX_MCXW_MCXW727C._lpadc_tune_value.kLPADC_TuneValue3 (*C enumerator*), 472
drivers_MCX_MCXW_MCXW727C._lpcmp_close_window_event (*C enum*), 492
drivers_MCX_MCXW_MCXW727C._lpcmp_close_window_event.kLPCMP_CLoseWindowEventBothEdge (*C enumerator*), 493
drivers_MCX_MCXW_MCXW727C._lpcmp_close_window_event.kLPCMP_CloseWindowEventFallingEdge (*C enumerator*), 492
drivers_MCX_MCXW_MCXW727C._lpcmp_close_window_event.kLPCMP_CLoseWindowEventNoSet (*C enumerator*), 492
drivers_MCX_MCXW_MCXW727C._lpcmp_close_window_event.kLPCMP_CloseWindowEventRisingEdge (*C enumerator*), 492
drivers_MCX_MCXW_MCXW727C._lpcmp_config (*C struct*), 496
drivers_MCX_MCXW_MCXW727C._lpcmp_config.enableCmpToDacLink (*C var*), 496
drivers_MCX_MCXW_MCXW727C._lpcmp_config.enableInvertOutput (*C var*), 496
drivers_MCX_MCXW_MCXW727C._lpcmp_config.enableOutputPin (*C var*), 496
drivers_MCX_MCXW_MCXW727C._lpcmp_config.enableStopMode (*C var*), 496
drivers_MCX_MCXW_MCXW727C._lpcmp_config.functionalSourceClock (*C var*), 496
drivers_MCX_MCXW_MCXW727C._lpcmp_config.hysteresisMode (*C var*), 496
drivers_MCX_MCXW_MCXW727C._lpcmp_config.minusInputSrc (*C var*), 496
drivers_MCX_MCXW_MCXW727C._lpcmp_config.plusInputSrc (*C var*), 496
drivers_MCX_MCXW_MCXW727C._lpcmp_config.powerMode (*C var*), 496

drivers_MCX_MCXW_MCXW727C._lpcmp_config.useUnfilteredOutput (*C var*), 496
drivers_MCX_MCXW_MCXW727C._lpcmp_couta_signal (*C enum*), 492
drivers_MCX_MCXW_MCXW727C._lpcmp_couta_signal.kLPCMP_COUTASignalHigh (*C enumerator*), 492
drivers_MCX_MCXW_MCXW727C._lpcmp_couta_signal.kLPCMP_COUTASignalLow (*C enumerator*), 492
drivers_MCX_MCXW_MCXW727C._lpcmp_couta_signal.kLPCMP_COUTASignalNoSet (*C enumerator*), 492
drivers_MCX_MCXW_MCXW727C._lpcmp_dac_config (*C struct*), 495
drivers_MCX_MCXW_MCXW727C._lpcmp_dac_config.DACValue (*C var*), 496
drivers_MCX_MCXW_MCXW727C._lpcmp_dac_config.enableLowPowerMode (*C var*), 496
drivers_MCX_MCXW_MCXW727C._lpcmp_dac_config.referenceVoltageSource (*C var*), 496
drivers_MCX_MCXW_MCXW727C._lpcmp_dac_reference_voltage_source (*C enum*), 491
drivers_MCX_MCXW_MCXW727C._lpcmp_dac_reference_voltage_source.kLPCMP_VrefSourceVin1 (*C enumerator*), 491
drivers_MCX_MCXW_MCXW727C._lpcmp_dac_reference_voltage_source.kLPCMP_VrefSourceVin2 (*C enumerator*), 492
drivers_MCX_MCXW_MCXW727C._lpcmp_filter_config (*C struct*), 495
drivers_MCX_MCXW_MCXW727C._lpcmp_filter_config.enableSample (*C var*), 495
drivers_MCX_MCXW_MCXW727C._lpcmp_filter_config.filterSampleCount (*C var*), 495
drivers_MCX_MCXW_MCXW727C._lpcmp_filter_config.filterSamplePeriod (*C var*), 495
drivers_MCX_MCXW_MCXW727C._lpcmp_functional_source_clock (*C enum*), 492
drivers_MCX_MCXW_MCXW727C._lpcmp_functional_source_clock.kLPCMP_FunctionalClockSource0 (*C enumerator*), 492
drivers_MCX_MCXW_MCXW727C._lpcmp_functional_source_clock.kLPCMP_FunctionalClockSource1 (*C enumerator*), 492
drivers_MCX_MCXW_MCXW727C._lpcmp_functional_source_clock.kLPCMP_FunctionalClockSource2 (*C enumerator*), 492
drivers_MCX_MCXW_MCXW727C._lpcmp_functional_source_clock.kLPCMP_FunctionalClockSource3 (*C enumerator*), 492
drivers_MCX_MCXW_MCXW727C._lpcmp_hysteresis_mode (*C enum*), 491
drivers_MCX_MCXW_MCXW727C._lpcmp_hysteresis_mode.kLPCMP_HysteresisLevel0 (*C enumerator*), 491
drivers_MCX_MCXW_MCXW727C._lpcmp_hysteresis_mode.kLPCMP_HysteresisLevel1 (*C enumerator*), 491
drivers_MCX_MCXW_MCXW727C._lpcmp_hysteresis_mode.kLPCMP_HysteresisLevel2 (*C enumerator*), 491
drivers_MCX_MCXW_MCXW727C._lpcmp_hysteresis_mode.kLPCMP_HysteresisLevel3 (*C enumerator*), 491
drivers_MCX_MCXW_MCXW727C._lpcmp_interrupt_enable (*C enum*), 490
drivers_MCX_MCXW_MCXW727C._lpcmp_interrupt_enable.kLPCMP_OutputFallingInterruptEnable (*C enumerator*), 491
drivers_MCX_MCXW_MCXW727C._lpcmp_interrupt_enable.kLPCMP_OutputRisingInterruptEnable (*C enumerator*), 491
drivers_MCX_MCXW_MCXW727C._lpcmp_interrupt_enable.kLPCMP_RoundRobinInterruptEnable (*C enumerator*), 491
drivers_MCX_MCXW_MCXW727C._lpcmp_minus_input_src (*C enum*), 494
drivers_MCX_MCXW_MCXW727C._lpcmp_minus_input_src.kLPCMP_MinusInputSrcDac (*C enumerator*), 494
drivers_MCX_MCXW_MCXW727C._lpcmp_minus_input_src.kLPCMP_MinusInputSrcMux (*C enumerator*), 494
drivers_MCX_MCXW_MCXW727C._lpcmp_plus_input_src (*C enum*), 493
drivers_MCX_MCXW_MCXW727C._lpcmp_plus_input_src.kLPCMP_PlusInputSrcDac (*C enumerator*), 493
drivers_MCX_MCXW_MCXW727C._lpcmp_plus_input_src.kLPCMP_PlusInputSrcMux (*C enumerator*), 494
drivers_MCX_MCXW_MCXW727C._lpcmp_power_mode (*C enum*), 491
drivers_MCX_MCXW_MCXW727C._lpcmp_power_mode.kLPCMP_HighSpeedPowerMode (*C enu-*

merator), 491
drivers_MCX_MCXW_MCXW727C._lpcmp_power_mode.kLPCMP_LowSpeedPowerMode (*C enumerator*), 491
drivers_MCX_MCXW_MCXW727C._lpcmp_power_mode.kLPCMP_NanoPowerMode (*C enumerator*), 491
drivers_MCX_MCXW_MCXW727C._lpcmp_roundrobin_clock_source (*C enum*), 493
drivers_MCX_MCXW_MCXW727C._lpcmp_roundrobin_clock_source.kLPCMP_RoundRobinClockSource0 (*C enumerator*), 493
drivers_MCX_MCXW_MCXW727C._lpcmp_roundrobin_clock_source.kLPCMP_RoundRobinClockSource1 (*C enumerator*), 493
drivers_MCX_MCXW_MCXW727C._lpcmp_roundrobin_clock_source.kLPCMP_RoundRobinClockSource2 (*C enumerator*), 493
drivers_MCX_MCXW_MCXW727C._lpcmp_roundrobin_clock_source.kLPCMP_RoundRobinClockSource3 (*C enumerator*), 493
drivers_MCX_MCXW_MCXW727C._lpcmp_roundrobin_config (*C struct*), 497
drivers_MCX_MCXW_MCXW727C._lpcmp_roundrobin_config.channelSampleNumbers (*C var*), 497
drivers_MCX_MCXW_MCXW727C._lpcmp_roundrobin_config.checkerChannelMask (*C var*), 498
drivers_MCX_MCXW_MCXW727C._lpcmp_roundrobin_config.fixedChannel (*C var*), 497
drivers_MCX_MCXW_MCXW727C._lpcmp_roundrobin_config.fixedMuxPort (*C var*), 497
drivers_MCX_MCXW_MCXW727C._lpcmp_roundrobin_config.initDelayModules (*C var*), 497
drivers_MCX_MCXW_MCXW727C._lpcmp_roundrobin_config.roundrobinClockSource (*C var*), 497
drivers_MCX_MCXW_MCXW727C._lpcmp_roundrobin_config.roundrobinTriggerSource (*C var*), 497
drivers_MCX_MCXW_MCXW727C._lpcmp_roundrobin_config.sampleClockNumbers (*C var*), 497
drivers_MCX_MCXW_MCXW727C._lpcmp_roundrobin_config.sampleTimeThreshhold (*C var*), 497
drivers_MCX_MCXW_MCXW727C._lpcmp_roundrobin_fixedmuxport (*C enum*), 493
drivers_MCX_MCXW_MCXW727C._lpcmp_roundrobin_fixedmuxport.kLPCMP_FixedMinusMuxPort (*C enumerator*), 493
drivers_MCX_MCXW_MCXW727C._lpcmp_roundrobin_fixedmuxport.kLPCMP_FixedPlusMuxPort (*C enumerator*), 493
drivers_MCX_MCXW_MCXW727C._lpcmp_roundrobin_trigger_source (*C enum*), 493
drivers_MCX_MCXW_MCXW727C._lpcmp_roundrobin_trigger_source.kLPCMP_TriggerSourceExternally (*C enumerator*), 493
drivers_MCX_MCXW_MCXW727C._lpcmp_roundrobin_trigger_source.kLPCMP_TriggerSourceInternally (*C enumerator*), 493
drivers_MCX_MCXW_MCXW727C._lpcmp_status_flags (*C enum*), 490
drivers_MCX_MCXW_MCXW727C._lpcmp_status_flags.kLPCMP_OutputAssertEventFlag (*C enumerator*), 490
drivers_MCX_MCXW_MCXW727C._lpcmp_status_flags.kLPCMP_OutputFallingEventFlag (*C enumerator*), 490
drivers_MCX_MCXW_MCXW727C._lpcmp_status_flags.kLPCMP_OutputRisingEventFlag (*C enumerator*), 490
drivers_MCX_MCXW_MCXW727C._lpcmp_status_flags.kLPCMP_OutputRoundRobinEventFlag (*C enumerator*), 490
drivers_MCX_MCXW_MCXW727C._lpcmp_window_control_config (*C struct*), 497
drivers_MCX_MCXW_MCXW727C._lpcmp_window_control_config.closeWindowEvent (*C var*), 497
drivers_MCX_MCXW_MCXW727C._lpcmp_window_control_config.COUTASignal (*C var*), 497
drivers_MCX_MCXW_MCXW727C._lpcmp_window_control_config.enableInvertWindowSignal (*C var*), 497
drivers_MCX_MCXW_MCXW727C._lpi2c_data_match_config_mode (*C enum*), 510
drivers_MCX_MCXW_MCXW727C._lpi2c_data_match_config_mode.kLPI2C_1stWordAndM1EqualsM0AndM1 (*C enumerator*), 510
drivers_MCX_MCXW_MCXW727C._lpi2c_data_match_config_mode.kLPI2C_1stWordEqualsM0And2ndWordEqua (*C enumerator*), 510
drivers_MCX_MCXW_MCXW727C._lpi2c_data_match_config_mode.kLPI2C_1stWordEqualsM0OrM1 (*C enumerator*), 510

drivers_MCX_MCXW_MCXW727C._lpi2c_data_match_config_mode.kLPI2C_AnyWordAndM1EqualsM0AndM1 (*C enumerator*), 510
drivers_MCX_MCXW_MCXW727C._lpi2c_data_match_config_mode.kLPI2C_AnyWordEqualsM0AndNextWordEq (*C enumerator*), 510
drivers_MCX_MCXW_MCXW727C._lpi2c_data_match_config_mode.kLPI2C_AnyWordEqualsM0OrM1 (*C enumerator*), 510
drivers_MCX_MCXW_MCXW727C._lpi2c_data_match_config_mode.kLPI2C_MatchDisabled (*C enumerator*), 510
drivers_MCX_MCXW_MCXW727C._lpi2c_direction (*C enum*), 509
drivers_MCX_MCXW_MCXW727C._lpi2c_direction.kLPI2C_Read (*C enumerator*), 509
drivers_MCX_MCXW_MCXW727C._lpi2c_direction.kLPI2C_Write (*C enumerator*), 509
drivers_MCX_MCXW_MCXW727C._lpi2c_host_request_polarity (*C enum*), 510
drivers_MCX_MCXW_MCXW727C._lpi2c_host_request_polarity.kLPI2C_HostRequestPinActiveHigh (*C enumerator*), 510
drivers_MCX_MCXW_MCXW727C._lpi2c_host_request_polarity.kLPI2C_HostRequestPinActiveLow (*C enumerator*), 510
drivers_MCX_MCXW_MCXW727C._lpi2c_host_request_source (*C enum*), 510
drivers_MCX_MCXW_MCXW727C._lpi2c_host_request_source.kLPI2C_HostRequestExternalPin (*C enumerator*), 510
drivers_MCX_MCXW_MCXW727C._lpi2c_host_request_source.kLPI2C_HostRequestInputTrigger (*C enumerator*), 510
drivers_MCX_MCXW_MCXW727C._lpi2c_master_config (*C struct*), 512
drivers_MCX_MCXW_MCXW727C._lpi2c_master_config.baudRate_Hz (*C var*), 512
drivers_MCX_MCXW_MCXW727C._lpi2c_master_config.busIdleTimeout_ns (*C var*), 512
drivers_MCX_MCXW_MCXW727C._lpi2c_master_config.debugEnable (*C var*), 512
drivers_MCX_MCXW_MCXW727C._lpi2c_master_config.enableDoze (*C var*), 512
drivers_MCX_MCXW_MCXW727C._lpi2c_master_config.enableMaster (*C var*), 512
drivers_MCX_MCXW_MCXW727C._lpi2c_master_config.hostRequest (*C struct*), 514
drivers_MCX_MCXW_MCXW727C._lpi2c_master_config.hostRequest (*C var*), 513
drivers_MCX_MCXW_MCXW727C._lpi2c_master_config.hostRequest.enable (*C var*), 515
drivers_MCX_MCXW_MCXW727C._lpi2c_master_config.hostRequest.polarity (*C var*), 515
drivers_MCX_MCXW_MCXW727C._lpi2c_master_config.hostRequest.source (*C var*), 515
drivers_MCX_MCXW_MCXW727C._lpi2c_master_config.ignoreAck (*C var*), 512
drivers_MCX_MCXW_MCXW727C._lpi2c_master_config.pinConfig (*C var*), 512
drivers_MCX_MCXW_MCXW727C._lpi2c_master_config.pinLowTimeout_ns (*C var*), 513
drivers_MCX_MCXW_MCXW727C._lpi2c_master_config.sclGlitchFilterWidth_ns (*C var*), 513
drivers_MCX_MCXW_MCXW727C._lpi2c_master_config.sdaGlitchFilterWidth_ns (*C var*), 513
drivers_MCX_MCXW_MCXW727C._lpi2c_master_edma_handle (*C struct*), 517
drivers_MCX_MCXW_MCXW727C._lpi2c_master_edma_handle.base (*C var*), 517
drivers_MCX_MCXW_MCXW727C._lpi2c_master_edma_handle.commandBuffer (*C var*), 517
drivers_MCX_MCXW_MCXW727C._lpi2c_master_edma_handle.completionCallback (*C var*), 517
drivers_MCX_MCXW_MCXW727C._lpi2c_master_edma_handle.isBusy (*C var*), 517
drivers_MCX_MCXW_MCXW727C._lpi2c_master_edma_handle.nbytes (*C var*), 517
drivers_MCX_MCXW_MCXW727C._lpi2c_master_edma_handle.rx (*C var*), 517
drivers_MCX_MCXW_MCXW727C._lpi2c_master_edma_handle.tcds (*C var*), 517
drivers_MCX_MCXW_MCXW727C._lpi2c_master_edma_handle.transfer (*C var*), 517
drivers_MCX_MCXW_MCXW727C._lpi2c_master_edma_handle.tx (*C var*), 517
drivers_MCX_MCXW_MCXW727C._lpi2c_master_edma_handle.userData (*C var*), 517
drivers_MCX_MCXW_MCXW727C._lpi2c_master_flags (*C enum*), 507
drivers_MCX_MCXW_MCXW727C._lpi2c_master_flags.kLPI2C_MasterArbitrationLostFlag (*C enumerator*), 508
drivers_MCX_MCXW_MCXW727C._lpi2c_master_flags.kLPI2C_MasterBusBusyFlag (*C enumerator*), 508
drivers_MCX_MCXW_MCXW727C._lpi2c_master_flags.kLPI2C_MasterBusyFlag (*C enumerator*), 508
drivers_MCX_MCXW_MCXW727C._lpi2c_master_flags.kLPI2C_MasterClearFlags (*C enumerator*), 509

drivers_MCX_MCXW_MCXW727C._lpi2c_master_flags.kLPI2C_MasterDataMatchFlag (*C enumerator*), 508
drivers_MCX_MCXW_MCXW727C._lpi2c_master_flags.kLPI2C_MasterEndOfPacketFlag (*C enumerator*), 508
drivers_MCX_MCXW_MCXW727C._lpi2c_master_flags.kLPI2C_MasterErrorFlags (*C enumerator*), 509
drivers_MCX_MCXW_MCXW727C._lpi2c_master_flags.kLPI2C_MasterFifoErrFlag (*C enumerator*), 508
drivers_MCX_MCXW_MCXW727C._lpi2c_master_flags.kLPI2C_MasterIrqFlags (*C enumerator*), 509
drivers_MCX_MCXW_MCXW727C._lpi2c_master_flags.kLPI2C_MasterNackDetectFlag (*C enumerator*), 508
drivers_MCX_MCXW_MCXW727C._lpi2c_master_flags.kLPI2C_MasterPinLowTimeoutFlag (*C enumerator*), 508
drivers_MCX_MCXW_MCXW727C._lpi2c_master_flags.kLPI2C_MasterRxReadyFlag (*C enumerator*), 508
drivers_MCX_MCXW_MCXW727C._lpi2c_master_flags.kLPI2C_MasterStopDetectFlag (*C enumerator*), 508
drivers_MCX_MCXW_MCXW727C._lpi2c_master_flags.kLPI2C_MasterTxReadyFlag (*C enumerator*), 508
drivers_MCX_MCXW_MCXW727C._lpi2c_master_handle (*C struct*), 514
drivers_MCX_MCXW_MCXW727C._lpi2c_master_handle.buf (*C var*), 514
drivers_MCX_MCXW_MCXW727C._lpi2c_master_handle.chunkSize (*C var*), 514
drivers_MCX_MCXW_MCXW727C._lpi2c_master_handle.commandBuffer (*C var*), 514
drivers_MCX_MCXW_MCXW727C._lpi2c_master_handle.completionCallback (*C var*), 514
drivers_MCX_MCXW_MCXW727C._lpi2c_master_handle.remainingBytes (*C var*), 514
drivers_MCX_MCXW_MCXW727C._lpi2c_master_handle.state (*C var*), 514
drivers_MCX_MCXW_MCXW727C._lpi2c_master_handle.transfer (*C var*), 514
drivers_MCX_MCXW_MCXW727C._lpi2c_master_handle.userData (*C var*), 514
drivers_MCX_MCXW_MCXW727C._lpi2c_master_pin_config (*C enum*), 509
drivers_MCX_MCXW_MCXW727C._lpi2c_master_pin_config.kLPI2C_2PinOpenDrain (*C enumerator*), 509
drivers_MCX_MCXW_MCXW727C._lpi2c_master_pin_config.kLPI2C_2PinOpenDrainWithSeparateSlave (*C enumerator*), 509
drivers_MCX_MCXW_MCXW727C._lpi2c_master_pin_config.kLPI2C_2PinOutputOnly (*C enumerator*), 509
drivers_MCX_MCXW_MCXW727C._lpi2c_master_pin_config.kLPI2C_2PinOutputOnlyWithSeparateSlave (*C enumerator*), 509
drivers_MCX_MCXW_MCXW727C._lpi2c_master_pin_config.kLPI2C_2PinPushPull (*C enumerator*), 509
drivers_MCX_MCXW_MCXW727C._lpi2c_master_pin_config.kLPI2C_2PinPushPullWithSeparateSlave (*C enumerator*), 509
drivers_MCX_MCXW_MCXW727C._lpi2c_master_pin_config.kLPI2C_4PinPushPull (*C enumerator*), 509
drivers_MCX_MCXW_MCXW727C._lpi2c_master_pin_config.kLPI2C_4PinPushPullWithInvertedOutput (*C enumerator*), 509
drivers_MCX_MCXW_MCXW727C._lpi2c_master_transfer (*C struct*), 513
drivers_MCX_MCXW_MCXW727C._lpi2c_master_transfer_flags (*C enum*), 510
drivers_MCX_MCXW_MCXW727C._lpi2c_master_transfer_flags.kLPI2C_TransferDefaultFlag (*C enumerator*), 511
drivers_MCX_MCXW_MCXW727C._lpi2c_master_transfer_flags.kLPI2C_TransferNoStartFlag (*C enumerator*), 511
drivers_MCX_MCXW_MCXW727C._lpi2c_master_transfer_flags.kLPI2C_TransferNoStopFlag (*C enumerator*), 511
drivers_MCX_MCXW_MCXW727C._lpi2c_master_transfer.data (*C var*), 514
drivers_MCX_MCXW_MCXW727C._lpi2c_master_transfer.dataSize (*C var*), 514
drivers_MCX_MCXW_MCXW727C._lpi2c_master_transfer.direction (*C var*), 513
drivers_MCX_MCXW_MCXW727C._lpi2c_master_transfer.flags (*C var*), 513

drivers_MCX_MCXW_MCXW727C._lpi2c_master_transfer.slaveAddress (*C var*), 513
drivers_MCX_MCXW_MCXW727C._lpi2c_master_transfer.subaddress (*C var*), 514
drivers_MCX_MCXW_MCXW727C._lpi2c_master_transfer.subaddressSize (*C var*), 514
drivers_MCX_MCXW_MCXW727C._lpi2c_match_config (*C struct*), 513
drivers_MCX_MCXW_MCXW727C._lpi2c_match_config.match0 (*C var*), 513
drivers_MCX_MCXW_MCXW727C._lpi2c_match_config.match1 (*C var*), 513
drivers_MCX_MCXW_MCXW727C._lpi2c_match_config.matchMode (*C var*), 513
drivers_MCX_MCXW_MCXW727C._lpi2c_match_config.rxDataMatchOnly (*C var*), 513
drivers_MCX_MCXW_MCXW727C._lpi2c_slave_address_match (*C enum*), 524
drivers_MCX_MCXW_MCXW727C._lpi2c_slave_address_match.kLPI2C_MatchAddress0 (*C enumerator*), 524
drivers_MCX_MCXW_MCXW727C._lpi2c_slave_address_match.kLPI2C_MatchAddress0OrAddress1 (*C enumerator*), 524
drivers_MCX_MCXW_MCXW727C._lpi2c_slave_address_match.kLPI2C_MatchAddress0ThroughAddress1 (*C enumerator*), 524
drivers_MCX_MCXW_MCXW727C._lpi2c_slave_config (*C struct*), 526
drivers_MCX_MCXW_MCXW727C._lpi2c_slave_config.address0 (*C var*), 526
drivers_MCX_MCXW_MCXW727C._lpi2c_slave_config.address1 (*C var*), 526
drivers_MCX_MCXW_MCXW727C._lpi2c_slave_config.addressMatchMode (*C var*), 526
drivers_MCX_MCXW_MCXW727C._lpi2c_slave_config.clockHoldTime_ns (*C var*), 527
drivers_MCX_MCXW_MCXW727C._lpi2c_slave_config.dataValidDelay_ns (*C var*), 527
drivers_MCX_MCXW_MCXW727C._lpi2c_slave_config.enableGeneralCall (*C var*), 527
drivers_MCX_MCXW_MCXW727C._lpi2c_slave_config.enableReceivedAddressRead (*C var*), 527
drivers_MCX_MCXW_MCXW727C._lpi2c_slave_config.enableSlave (*C var*), 526
drivers_MCX_MCXW_MCXW727C._lpi2c_slave_config.filterDozeEnable (*C var*), 526
drivers_MCX_MCXW_MCXW727C._lpi2c_slave_config.filterEnable (*C var*), 527
drivers_MCX_MCXW_MCXW727C._lpi2c_slave_config.ignoreAck (*C var*), 527
drivers_MCX_MCXW_MCXW727C._lpi2c_slave_config.sclGlitchFilterWidth_ns (*C var*), 527
drivers_MCX_MCXW_MCXW727C._lpi2c_slave_config.sclStall (*C struct*), 528
drivers_MCX_MCXW_MCXW727C._lpi2c_slave_config.sclStall (*C var*), 527
drivers_MCX_MCXW_MCXW727C._lpi2c_slave_config.sclStall.enableAck (*C var*), 528
drivers_MCX_MCXW_MCXW727C._lpi2c_slave_config.sclStall.enableAddress (*C var*), 529
drivers_MCX_MCXW_MCXW727C._lpi2c_slave_config.sclStall.enableRx (*C var*), 529
drivers_MCX_MCXW_MCXW727C._lpi2c_slave_config.sclStall.enableTx (*C var*), 528
drivers_MCX_MCXW_MCXW727C._lpi2c_slave_config.sdaGlitchFilterWidth_ns (*C var*), 527
drivers_MCX_MCXW_MCXW727C._lpi2c_slave_flags (*C enum*), 523
drivers_MCX_MCXW_MCXW727C._lpi2c_slave_flags.kLPI2C_SlaveAddressMatch0Flag (*C enumerator*), 524
drivers_MCX_MCXW_MCXW727C._lpi2c_slave_flags.kLPI2C_SlaveAddressMatch1Flag (*C enumerator*), 524
drivers_MCX_MCXW_MCXW727C._lpi2c_slave_flags.kLPI2C_SlaveAddressValidFlag (*C enumerator*), 523
drivers_MCX_MCXW_MCXW727C._lpi2c_slave_flags.kLPI2C_SlaveBitErrFlag (*C enumerator*), 524
drivers_MCX_MCXW_MCXW727C._lpi2c_slave_flags.kLPI2C_SlaveBusBusyFlag (*C enumerator*), 524
drivers_MCX_MCXW_MCXW727C._lpi2c_slave_flags.kLPI2C_SlaveBusyFlag (*C enumerator*), 524
drivers_MCX_MCXW_MCXW727C._lpi2c_slave_flags.kLPI2C_SlaveClearFlags (*C enumerator*), 524
drivers_MCX_MCXW_MCXW727C._lpi2c_slave_flags.kLPI2C_SlaveErrorFlags (*C enumerator*), 524
drivers_MCX_MCXW_MCXW727C._lpi2c_slave_flags.kLPI2C_SlaveFifoErrFlag (*C enumerator*), 524
drivers_MCX_MCXW_MCXW727C._lpi2c_slave_flags.kLPI2C_SlaveGeneralCallFlag (*C enumerator*), 524
drivers_MCX_MCXW_MCXW727C._lpi2c_slave_flags.kLPI2C_SlaveIrqFlags (*C enumerator*), 524

drivers_MCX_MCXW_MCXW727C._lpi2c_slave_flags.kLPI2C_SlaveRepeatedStartDetectFlag (*C enumerator*), 524
drivers_MCX_MCXW_MCXW727C._lpi2c_slave_flags.kLPI2C_SlaveRxReadyFlag (*C enumerator*), 523
drivers_MCX_MCXW_MCXW727C._lpi2c_slave_flags.kLPI2C_SlaveStopDetectFlag (*C enumerator*), 524
drivers_MCX_MCXW_MCXW727C._lpi2c_slave_flags.kLPI2C_SlaveTransmitAckFlag (*C enumerator*), 523
drivers_MCX_MCXW_MCXW727C._lpi2c_slave_flags.kLPI2C_SlaveTxReadyFlag (*C enumerator*), 523
drivers_MCX_MCXW_MCXW727C._lpi2c_slave_handle (*C struct*), 528
drivers_MCX_MCXW_MCXW727C._lpi2c_slave_handle.callback (*C var*), 528
drivers_MCX_MCXW_MCXW727C._lpi2c_slave_handle.eventMask (*C var*), 528
drivers_MCX_MCXW_MCXW727C._lpi2c_slave_handle.isBusy (*C var*), 528
drivers_MCX_MCXW_MCXW727C._lpi2c_slave_handle.transfer (*C var*), 528
drivers_MCX_MCXW_MCXW727C._lpi2c_slave_handle.transferredCount (*C var*), 528
drivers_MCX_MCXW_MCXW727C._lpi2c_slave_handle.userData (*C var*), 528
drivers_MCX_MCXW_MCXW727C._lpi2c_slave_handle.wasTransmit (*C var*), 528
drivers_MCX_MCXW_MCXW727C._lpi2c_slave_transfer (*C struct*), 527
drivers_MCX_MCXW_MCXW727C._lpi2c_slave_transfer_event (*C enum*), 525
drivers_MCX_MCXW_MCXW727C._lpi2c_slave_transfer_event.kLPI2C_SlaveAddressMatchEvent (*C enumerator*), 525
drivers_MCX_MCXW_MCXW727C._lpi2c_slave_transfer_event.kLPI2C_SlaveAllEvents (*C enumerator*), 525
drivers_MCX_MCXW_MCXW727C._lpi2c_slave_transfer_event.kLPI2C_SlaveCompletionEvent (*C enumerator*), 525
drivers_MCX_MCXW_MCXW727C._lpi2c_slave_transfer_event.kLPI2C_SlaveReceiveEvent (*C enumerator*), 525
drivers_MCX_MCXW_MCXW727C._lpi2c_slave_transfer_event.kLPI2C_SlaveRepeatedStartEvent (*C enumerator*), 525
drivers_MCX_MCXW_MCXW727C._lpi2c_slave_transfer_event.kLPI2C_SlaveTransmitAckEvent (*C enumerator*), 525
drivers_MCX_MCXW_MCXW727C._lpi2c_slave_transfer_event.kLPI2C_SlaveTransmitEvent (*C enumerator*), 525
drivers_MCX_MCXW_MCXW727C._lpi2c_slave_transfer.completionStatus (*C var*), 527
drivers_MCX_MCXW_MCXW727C._lpi2c_slave_transfer.data (*C var*), 527
drivers_MCX_MCXW_MCXW727C._lpi2c_slave_transfer.dataSize (*C var*), 527
drivers_MCX_MCXW_MCXW727C._lpi2c_slave_transfer.event (*C var*), 527
drivers_MCX_MCXW_MCXW727C._lpi2c_slave_transfer.receivedAddress (*C var*), 527
drivers_MCX_MCXW_MCXW727C._lpi2c_slave_transfer.transferredCount (*C var*), 528
drivers_MCX_MCXW_MCXW727C._lpit_chnl (*C enum*), 532
drivers_MCX_MCXW_MCXW727C._lpit_chnl_params (*C struct*), 535
drivers_MCX_MCXW_MCXW727C._lpit_chnl_params.chainChannel (*C var*), 536
drivers_MCX_MCXW_MCXW727C._lpit_chnl_params.enableReloadOnTrigger (*C var*), 536
drivers_MCX_MCXW_MCXW727C._lpit_chnl_params.enableStartOnTrigger (*C var*), 536
drivers_MCX_MCXW_MCXW727C._lpit_chnl_params.enableStopOnTimeout (*C var*), 536
drivers_MCX_MCXW_MCXW727C._lpit_chnl_params.timerMode (*C var*), 536
drivers_MCX_MCXW_MCXW727C._lpit_chnl_params.triggerSelect (*C var*), 536
drivers_MCX_MCXW_MCXW727C._lpit_chnl_params.triggerSource (*C var*), 536
drivers_MCX_MCXW_MCXW727C._lpit_chnl.kLPIT_Chnl_0 (*C enumerator*), 532
drivers_MCX_MCXW_MCXW727C._lpit_chnl.kLPIT_Chnl_1 (*C enumerator*), 532
drivers_MCX_MCXW_MCXW727C._lpit_chnl.kLPIT_Chnl_2 (*C enumerator*), 532
drivers_MCX_MCXW_MCXW727C._lpit_chnl.kLPIT_Chnl_3 (*C enumerator*), 532
drivers_MCX_MCXW_MCXW727C._lpit_config (*C struct*), 536
drivers_MCX_MCXW_MCXW727C._lpit_config.enableRunInDebug (*C var*), 536
drivers_MCX_MCXW_MCXW727C._lpit_config.enableRunInDoze (*C var*), 536
drivers_MCX_MCXW_MCXW727C._lpit_interrupt_enable (*C enum*), 534

drivers_MCX_MCXW_MCXW727C._lpit_interrupt_enable.kLPIT_Channel0TimerInterruptEnable (*C enumerator*), 534
drivers_MCX_MCXW_MCXW727C._lpit_interrupt_enable.kLPIT_Channel1TimerInterruptEnable (*C enumerator*), 534
drivers_MCX_MCXW_MCXW727C._lpit_interrupt_enable.kLPIT_Channel2TimerInterruptEnable (*C enumerator*), 534
drivers_MCX_MCXW_MCXW727C._lpit_interrupt_enable.kLPIT_Channel3TimerInterruptEnable (*C enumerator*), 534
drivers_MCX_MCXW_MCXW727C._lpit_status_flags (*C enum*), 534
drivers_MCX_MCXW_MCXW727C._lpit_status_flags.kLPIT_Channel0TimerFlag (*C enumerator*), 534
drivers_MCX_MCXW_MCXW727C._lpit_status_flags.kLPIT_Channel1TimerFlag (*C enumerator*), 534
drivers_MCX_MCXW_MCXW727C._lpit_status_flags.kLPIT_Channel2TimerFlag (*C enumerator*), 534
drivers_MCX_MCXW_MCXW727C._lpit_status_flags.kLPIT_Channel3TimerFlag (*C enumerator*), 534
drivers_MCX_MCXW_MCXW727C._lpit_timer_modes (*C enum*), 532
drivers_MCX_MCXW_MCXW727C._lpit_timer_modes.kLPIT_DualPeriodicCounter (*C enumerator*), 532
drivers_MCX_MCXW_MCXW727C._lpit_timer_modes.kLPIT_InputCapture (*C enumerator*), 532
drivers_MCX_MCXW_MCXW727C._lpit_timer_modes.kLPIT_PeriodicCounter (*C enumerator*), 532
drivers_MCX_MCXW_MCXW727C._lpit_timer_modes.kLPIT_TriggerAccumulator (*C enumerator*), 532
drivers_MCX_MCXW_MCXW727C._lpit_trigger_select (*C enum*), 532
drivers_MCX_MCXW_MCXW727C._lpit_trigger_select.kLPIT_Trigger_TimerChn0 (*C enumerator*), 532
drivers_MCX_MCXW_MCXW727C._lpit_trigger_select.kLPIT_Trigger_TimerChn1 (*C enumerator*), 532
drivers_MCX_MCXW_MCXW727C._lpit_trigger_select.kLPIT_Trigger_TimerChn2 (*C enumerator*), 533
drivers_MCX_MCXW_MCXW727C._lpit_trigger_select.kLPIT_Trigger_TimerChn3 (*C enumerator*), 533
drivers_MCX_MCXW_MCXW727C._lpit_trigger_select.kLPIT_Trigger_TimerChn4 (*C enumerator*), 533
drivers_MCX_MCXW_MCXW727C._lpit_trigger_select.kLPIT_Trigger_TimerChn5 (*C enumerator*), 533
drivers_MCX_MCXW_MCXW727C._lpit_trigger_select.kLPIT_Trigger_TimerChn6 (*C enumerator*), 533
drivers_MCX_MCXW_MCXW727C._lpit_trigger_select.kLPIT_Trigger_TimerChn7 (*C enumerator*), 533
drivers_MCX_MCXW_MCXW727C._lpit_trigger_select.kLPIT_Trigger_TimerChn8 (*C enumerator*), 533
drivers_MCX_MCXW_MCXW727C._lpit_trigger_select.kLPIT_Trigger_TimerChn9 (*C enumerator*), 533
drivers_MCX_MCXW_MCXW727C._lpit_trigger_select.kLPIT_Trigger_TimerChn10 (*C enumerator*), 533
drivers_MCX_MCXW_MCXW727C._lpit_trigger_select.kLPIT_Trigger_TimerChn11 (*C enumerator*), 533
drivers_MCX_MCXW_MCXW727C._lpit_trigger_select.kLPIT_Trigger_TimerChn12 (*C enumerator*), 533
drivers_MCX_MCXW_MCXW727C._lpit_trigger_select.kLPIT_Trigger_TimerChn13 (*C enumerator*), 533
drivers_MCX_MCXW_MCXW727C._lpit_trigger_select.kLPIT_Trigger_TimerChn14 (*C enumerator*), 533
drivers_MCX_MCXW_MCXW727C._lpit_trigger_select.kLPIT_Trigger_TimerChn15 (*C enumerator*), 533

drivers_MCX_MCXW_MCXW727C._lpit_trigger_source (*C enum*), 533
drivers_MCX_MCXW_MCXW727C._lpit_trigger_source.kLPIT_TriggerSource_External (*C enumerator*), 533
drivers_MCX_MCXW_MCXW727C._lpit_trigger_source.kLPIT_TriggerSource_Internal (*C enumerator*), 534
drivers_MCX_MCXW_MCXW727C._lpspi_clock_phase (*C enum*), 550
drivers_MCX_MCXW_MCXW727C._lpspi_clock_phase.kLPSPI_ClockPhaseFirstEdge (*C enumerator*), 550
drivers_MCX_MCXW_MCXW727C._lpspi_clock_phase.kLPSPI_ClockPhaseSecondEdge (*C enumerator*), 550
drivers_MCX_MCXW_MCXW727C._lpspi_clock_polarity (*C enum*), 550
drivers_MCX_MCXW_MCXW727C._lpspi_clock_polarity.kLPSPI_ClockPolarityActiveHigh (*C enumerator*), 550
drivers_MCX_MCXW_MCXW727C._lpspi_clock_polarity.kLPSPI_ClockPolarityActiveLow (*C enumerator*), 550
drivers_MCX_MCXW_MCXW727C._lpspi_data_out_config (*C enum*), 552
drivers_MCX_MCXW_MCXW727C._lpspi_data_out_config.kLpspiDataOutRetained (*C enumerator*), 552
drivers_MCX_MCXW_MCXW727C._lpspi_data_out_config.kLpspiDataOutTristate (*C enumerator*), 552
drivers_MCX_MCXW_MCXW727C._lpspi_delay_type (*C enum*), 552
drivers_MCX_MCXW_MCXW727C._lpspi_delay_type.kLPSPI_BetweenTransfer (*C enumerator*), 552
drivers_MCX_MCXW_MCXW727C._lpspi_delay_type.kLPSPI_LastSckToPcs (*C enumerator*), 552
drivers_MCX_MCXW_MCXW727C._lpspi_delay_type.kLPSPI_PcsToSck (*C enumerator*), 552
drivers_MCX_MCXW_MCXW727C._lpspi_dma_enable (*C enum*), 548
drivers_MCX_MCXW_MCXW727C._lpspi_dma_enable.kLPSPI_RxDmaEnable (*C enumerator*), 549
drivers_MCX_MCXW_MCXW727C._lpspi_dma_enable.kLPSPI_TxDmaEnable (*C enumerator*), 549
drivers_MCX_MCXW_MCXW727C._lpspi_flags (*C enum*), 547
drivers_MCX_MCXW_MCXW727C._lpspi_flags.kLPSPI_AllStatusFlag (*C enumerator*), 548
drivers_MCX_MCXW_MCXW727C._lpspi_flags.kLPSPI_DataMatchFlag (*C enumerator*), 548
drivers_MCX_MCXW_MCXW727C._lpspi_flags.kLPSPI_FrameCompleteFlag (*C enumerator*), 547
drivers_MCX_MCXW_MCXW727C._lpspi_flags.kLPSPI_ModuleBusyFlag (*C enumerator*), 548
drivers_MCX_MCXW_MCXW727C._lpspi_flags.kLPSPI_ReceiveErrorFlag (*C enumerator*), 548
drivers_MCX_MCXW_MCXW727C._lpspi_flags.kLPSPI_RxDataReadyFlag (*C enumerator*), 547
drivers_MCX_MCXW_MCXW727C._lpspi_flags.kLPSPI_TransferCompleteFlag (*C enumerator*), 547
drivers_MCX_MCXW_MCXW727C._lpspi_flags.kLPSPI_TransmitErrorFlag (*C enumerator*), 548
drivers_MCX_MCXW_MCXW727C._lpspi_flags.kLPSPI_TxDataRequestFlag (*C enumerator*), 547
drivers_MCX_MCXW_MCXW727C._lpspi_flags.kLPSPI_WordCompleteFlag (*C enumerator*), 547
drivers_MCX_MCXW_MCXW727C._lpspi_host_request_select (*C enum*), 551
drivers_MCX_MCXW_MCXW727C._lpspi_host_request_select.kLPSPI_HostReqExtPin (*C enumerator*), 551
drivers_MCX_MCXW_MCXW727C._lpspi_host_request_select.kLPSPI_HostReqInternalTrigger (*C enumerator*), 551
drivers_MCX_MCXW_MCXW727C._lpspi_interrupt_enable (*C enum*), 548
drivers_MCX_MCXW_MCXW727C._lpspi_interrupt_enable.kLPSPI_AllInterruptEnable (*C enumerator*), 548
drivers_MCX_MCXW_MCXW727C._lpspi_interrupt_enable.kLPSPI_DataMatchInterruptEnable (*C enumerator*), 548
drivers_MCX_MCXW_MCXW727C._lpspi_interrupt_enable.kLPSPI_FrameCompleteInterruptEnable (*C enumerator*), 548
drivers_MCX_MCXW_MCXW727C._lpspi_interrupt_enable.kLPSPI_ReceiveErrorInterruptEnable (*C enumerator*), 548
drivers_MCX_MCXW_MCXW727C._lpspi_interrupt_enable.kLPSPI_RxInterruptEnable (*C enumerator*), 548

drivers_MCX_MCXW_MCXW727C._lpspi_interrupt_enable.kLPSPI_TransferCompleteInterruptEnable (*C enumerator*), 548
drivers_MCX_MCXW_MCXW727C._lpspi_interrupt_enable.kLPSPI_TransmitErrorInterruptEnable (*C enumerator*), 548
drivers_MCX_MCXW_MCXW727C._lpspi_interrupt_enable.kLPSPI_TxInterruptEnable (*C enumerator*), 548
drivers_MCX_MCXW_MCXW727C._lpspi_interrupt_enable.kLPSPI_WordCompleteInterruptEnable (*C enumerator*), 548
drivers_MCX_MCXW_MCXW727C._lpspi_master_config (*C struct*), 556
drivers_MCX_MCXW_MCXW727C._lpspi_master_config.baudRate (*C var*), 556
drivers_MCX_MCXW_MCXW727C._lpspi_master_config.betweenTransferDelayInNanoSec (*C var*), 556
drivers_MCX_MCXW_MCXW727C._lpspi_master_config.bitsPerFrame (*C var*), 556
drivers_MCX_MCXW_MCXW727C._lpspi_master_config.cpha (*C var*), 556
drivers_MCX_MCXW_MCXW727C._lpspi_master_config.cpol (*C var*), 556
drivers_MCX_MCXW_MCXW727C._lpspi_master_config.dataOutConfig (*C var*), 557
drivers_MCX_MCXW_MCXW727C._lpspi_master_config.direction (*C var*), 556
drivers_MCX_MCXW_MCXW727C._lpspi_master_config.enableInputDelay (*C var*), 557
drivers_MCX_MCXW_MCXW727C._lpspi_master_config.lastSckToPcsDelayInNanoSec (*C var*), 556
drivers_MCX_MCXW_MCXW727C._lpspi_master_config.pcsActiveHighOrLow (*C var*), 557
drivers_MCX_MCXW_MCXW727C._lpspi_master_config.pcsToSckDelayInNanoSec (*C var*), 556
drivers_MCX_MCXW_MCXW727C._lpspi_master_config.pinCfg (*C var*), 557
drivers_MCX_MCXW_MCXW727C._lpspi_master_config.whichPcs (*C var*), 556
drivers_MCX_MCXW_MCXW727C._lpspi_master_edma_handle (*C struct*), 565
drivers_MCX_MCXW_MCXW727C._lpspi_master_edma_handle.bytesEachRead (*C var*), 565
drivers_MCX_MCXW_MCXW727C._lpspi_master_edma_handle.bytesEachWrite (*C var*), 565
drivers_MCX_MCXW_MCXW727C._lpspi_master_edma_handle.bytesLastRead (*C var*), 565
drivers_MCX_MCXW_MCXW727C._lpspi_master_edma_handle.callback (*C var*), 566
drivers_MCX_MCXW_MCXW727C._lpspi_master_edma_handle.dataBytesEveryTime (*C var*), 566
drivers_MCX_MCXW_MCXW727C._lpspi_master_edma_handle.dmaTransmitTime (*C var*), 566
drivers_MCX_MCXW_MCXW727C._lpspi_master_edma_handle.edmaRxRegToRxDataHandle (*C var*), 567
drivers_MCX_MCXW_MCXW727C._lpspi_master_edma_handle.edmaTxDataToTxRegHandle (*C var*), 567
drivers_MCX_MCXW_MCXW727C._lpspi_master_edma_handle.fifoSize (*C var*), 565
drivers_MCX_MCXW_MCXW727C._lpspi_master_edma_handle.isByteSwap (*C var*), 565
drivers_MCX_MCXW_MCXW727C._lpspi_master_edma_handle.isMultiDMATransmit (*C var*), 566
drivers_MCX_MCXW_MCXW727C._lpspi_master_edma_handle.isPcsContinuous (*C var*), 565
drivers_MCX_MCXW_MCXW727C._lpspi_master_edma_handle.isThereExtraRxBytes (*C var*), 565
drivers_MCX_MCXW_MCXW727C._lpspi_master_edma_handle.lastTimeDataBytes (*C var*), 566
drivers_MCX_MCXW_MCXW727C._lpspi_master_edma_handle.lastTimeTCD (*C var*), 566
drivers_MCX_MCXW_MCXW727C._lpspi_master_edma_handle.lpspiSoftwareTCD (*C var*), 567
drivers_MCX_MCXW_MCXW727C._lpspi_master_edma_handle.nbytes (*C var*), 566
drivers_MCX_MCXW_MCXW727C._lpspi_master_edma_handle.readRegRemainingTimes (*C var*), 566
drivers_MCX_MCXW_MCXW727C._lpspi_master_edma_handle.rxBuffIfNull (*C var*), 566
drivers_MCX_MCXW_MCXW727C._lpspi_master_edma_handle.rxData (*C var*), 565
drivers_MCX_MCXW_MCXW727C._lpspi_master_edma_handle.rxRemainingByteCount (*C var*), 565
drivers_MCX_MCXW_MCXW727C._lpspi_master_edma_handle.rxWatermark (*C var*), 565
drivers_MCX_MCXW_MCXW727C._lpspi_master_edma_handle.state (*C var*), 566
drivers_MCX_MCXW_MCXW727C._lpspi_master_edma_handle.totalByteCount (*C var*), 566
drivers_MCX_MCXW_MCXW727C._lpspi_master_edma_handle.transferConfigRx (*C var*), 566
drivers_MCX_MCXW_MCXW727C._lpspi_master_edma_handle.transferConfigTx (*C var*), 566
drivers_MCX_MCXW_MCXW727C._lpspi_master_edma_handle.transmitCommand (*C var*), 566
drivers_MCX_MCXW_MCXW727C._lpspi_master_edma_handle.txBuffIfNull (*C var*), 566
drivers_MCX_MCXW_MCXW727C._lpspi_master_edma_handle.txData (*C var*), 565

drivers_MCX_MCXW_MCXW727C._lpspi_master_edma_handle.txRemainingByteCount (*C var*), 565
drivers_MCX_MCXW_MCXW727C._lpspi_master_edma_handle.userData (*C var*), 566
drivers_MCX_MCXW_MCXW727C._lpspi_master_edma_handle.writeRegRemainingTimes (*C var*), 565
drivers_MCX_MCXW_MCXW727C._lpspi_master_handle (*C struct*), 558
drivers_MCX_MCXW_MCXW727C._lpspi_master_handle.bytesEachRead (*C var*), 558
drivers_MCX_MCXW_MCXW727C._lpspi_master_handle.bytesEachWrite (*C var*), 558
drivers_MCX_MCXW_MCXW727C._lpspi_master_handle.bytesPerFrame (*C var*), 558
drivers_MCX_MCXW_MCXW727C._lpspi_master_handle.callback (*C var*), 559
drivers_MCX_MCXW_MCXW727C._lpspi_master_handle.fifoSize (*C var*), 558
drivers_MCX_MCXW_MCXW727C._lpspi_master_handle.frameSize (*C var*), 558
drivers_MCX_MCXW_MCXW727C._lpspi_master_handle.isByteSwap (*C var*), 558
drivers_MCX_MCXW_MCXW727C._lpspi_master_handle.isPcsContinuous (*C var*), 558
drivers_MCX_MCXW_MCXW727C._lpspi_master_handle.isTxMask (*C var*), 558
drivers_MCX_MCXW_MCXW727C._lpspi_master_handle.readRegRemainingTimes (*C var*), 559
drivers_MCX_MCXW_MCXW727C._lpspi_master_handle.rxData (*C var*), 559
drivers_MCX_MCXW_MCXW727C._lpspi_master_handle.rxRemainingByteCount (*C var*), 559
drivers_MCX_MCXW_MCXW727C._lpspi_master_handle.rxWatermark (*C var*), 558
drivers_MCX_MCXW_MCXW727C._lpspi_master_handle.state (*C var*), 559
drivers_MCX_MCXW_MCXW727C._lpspi_master_handle.totalByteCount (*C var*), 559
drivers_MCX_MCXW_MCXW727C._lpspi_master_handle.txBuffIfNull (*C var*), 559
drivers_MCX_MCXW_MCXW727C._lpspi_master_handle.txData (*C var*), 559
drivers_MCX_MCXW_MCXW727C._lpspi_master_handle.txRemainingByteCount (*C var*), 559
drivers_MCX_MCXW_MCXW727C._lpspi_master_handle.userData (*C var*), 559
drivers_MCX_MCXW_MCXW727C._lpspi_master_handle.writeRegRemainingTimes (*C var*), 559
drivers_MCX_MCXW_MCXW727C._lpspi_master_handle.writeTcrInIsr (*C var*), 558
drivers_MCX_MCXW_MCXW727C._lpspi_master_slave_mode (*C enum*), 549
drivers_MCX_MCXW_MCXW727C._lpspi_master_slave_mode.kLPSPI_Master (*C enumerator*), 549
drivers_MCX_MCXW_MCXW727C._lpspi_master_slave_mode.kLPSPI_Slave (*C enumerator*), 549
drivers_MCX_MCXW_MCXW727C._lpspi_match_config (*C enum*), 551
drivers_MCX_MCXW_MCXW727C._lpspi_match_config.kLPSI_1stWordAndM1EqualsM0andM1 (*C enumerator*), 551
drivers_MCX_MCXW_MCXW727C._lpspi_match_config.kLPSI_1stWordEqualsM0and2ndWordEqualsM1 (*C enumerator*), 551
drivers_MCX_MCXW_MCXW727C._lpspi_match_config.kLPSI_1stWordEqualsM0orM1 (*C enumerator*), 551
drivers_MCX_MCXW_MCXW727C._lpspi_match_config.kLPSI_AnyWordAndM1EqualsM0andM1 (*C enumerator*), 551
drivers_MCX_MCXW_MCXW727C._lpspi_match_config.kLPSI_AnyWordEqualsM0andNxtWordEqualsM1 (*C enumerator*), 551
drivers_MCX_MCXW_MCXW727C._lpspi_match_config.kLPSI_AnyWordEqualsM0orM1 (*C enumerator*), 551
drivers_MCX_MCXW_MCXW727C._lpspi_match_config.kLPSI_MatchDisabled (*C enumerator*), 551
drivers_MCX_MCXW_MCXW727C._lpspi_pcs_polarity (*C enum*), 549
drivers_MCX_MCXW_MCXW727C._lpspi_pcs_polarity_config (*C enum*), 549
drivers_MCX_MCXW_MCXW727C._lpspi_pcs_polarity_config.kLPSPI_PcsActiveHigh (*C enumerator*), 549
drivers_MCX_MCXW_MCXW727C._lpspi_pcs_polarity_config.kLPSPI_PcsActiveLow (*C enumerator*), 549
drivers_MCX_MCXW_MCXW727C._lpspi_pcs_polarity.kLPSPI_Pcs0ActiveLow (*C enumerator*), 550
drivers_MCX_MCXW_MCXW727C._lpspi_pcs_polarity.kLPSPI_Pcs1ActiveLow (*C enumerator*), 550
drivers_MCX_MCXW_MCXW727C._lpspi_pcs_polarity.kLPSPI_Pcs2ActiveLow (*C enumerator*),

550
drivers_MCX_MCXW_MCXW727C._lpspi_pcs_polarity.kLPSPI_Pcs3ActiveLow (*C enumerator*), 550
drivers_MCX_MCXW_MCXW727C._lpspi_pcs_polarity.kLPSPI_PcsAllActiveLow (*C enumerator*), 550
drivers_MCX_MCXW_MCXW727C._lpspi_pin_config (*C enum*), 551
drivers_MCX_MCXW_MCXW727C._lpspi_pin_config.kLPSPI_SdiInSdiOut (*C enumerator*), 551
drivers_MCX_MCXW_MCXW727C._lpspi_pin_config.kLPSPI_SdiInSdoOut (*C enumerator*), 551
drivers_MCX_MCXW_MCXW727C._lpspi_pin_config.kLPSPI_SdoInSdiOut (*C enumerator*), 552
drivers_MCX_MCXW_MCXW727C._lpspi_pin_config.kLPSPI_SdoInSdoOut (*C enumerator*), 551
drivers_MCX_MCXW_MCXW727C._lpspi_shift_direction (*C enum*), 550
drivers_MCX_MCXW_MCXW727C._lpspi_shift_direction.kLPSPI_LsbFirst (*C enumerator*), 550
drivers_MCX_MCXW_MCXW727C._lpspi_shift_direction.kLPSPI_MsbFirst (*C enumerator*), 550
drivers_MCX_MCXW_MCXW727C._lpspi_slave_config (*C struct*), 557
drivers_MCX_MCXW_MCXW727C._lpspi_slave_config.bitsPerFrame (*C var*), 557
drivers_MCX_MCXW_MCXW727C._lpspi_slave_config.cpha (*C var*), 557
drivers_MCX_MCXW_MCXW727C._lpspi_slave_config.cpol (*C var*), 557
drivers_MCX_MCXW_MCXW727C._lpspi_slave_config.dataOutConfig (*C var*), 557
drivers_MCX_MCXW_MCXW727C._lpspi_slave_config.direction (*C var*), 557
drivers_MCX_MCXW_MCXW727C._lpspi_slave_config.pcsActiveHighOrLow (*C var*), 557
drivers_MCX_MCXW_MCXW727C._lpspi_slave_config.pinCfg (*C var*), 557
drivers_MCX_MCXW_MCXW727C._lpspi_slave_config.whichPcs (*C var*), 557
drivers_MCX_MCXW_MCXW727C._lpspi_slave_edma_handle (*C struct*), 567
drivers_MCX_MCXW_MCXW727C._lpspi_slave_edma_handle.bytesEachRead (*C var*), 567
drivers_MCX_MCXW_MCXW727C._lpspi_slave_edma_handle.bytesEachWrite (*C var*), 567
drivers_MCX_MCXW_MCXW727C._lpspi_slave_edma_handle.bytesLastRead (*C var*), 567
drivers_MCX_MCXW_MCXW727C._lpspi_slave_edma_handle.callback (*C var*), 568
drivers_MCX_MCXW_MCXW727C._lpspi_slave_edma_handle.edmaRxRegToRxDataHandle (*C var*), 568
drivers_MCX_MCXW_MCXW727C._lpspi_slave_edma_handle.edmaTxDataToTxRegHandle (*C var*), 568
drivers_MCX_MCXW_MCXW727C._lpspi_slave_edma_handle.errorCount (*C var*), 568
drivers_MCX_MCXW_MCXW727C._lpspi_slave_edma_handle.fifoSize (*C var*), 567
drivers_MCX_MCXW_MCXW727C._lpspi_slave_edma_handle.isByteSwap (*C var*), 567
drivers_MCX_MCXW_MCXW727C._lpspi_slave_edma_handle.isThereExtraRxBytes (*C var*), 567
drivers_MCX_MCXW_MCXW727C._lpspi_slave_edma_handle.lpspiSoftwareTCD (*C var*), 568
drivers_MCX_MCXW_MCXW727C._lpspi_slave_edma_handle.nbytes (*C var*), 567
drivers_MCX_MCXW_MCXW727C._lpspi_slave_edma_handle.readRegRemainingTimes (*C var*), 568
drivers_MCX_MCXW_MCXW727C._lpspi_slave_edma_handle.rxBuffIfNull (*C var*), 568
drivers_MCX_MCXW_MCXW727C._lpspi_slave_edma_handle.rxData (*C var*), 567
drivers_MCX_MCXW_MCXW727C._lpspi_slave_edma_handle.rxRemainingByteCount (*C var*), 568
drivers_MCX_MCXW_MCXW727C._lpspi_slave_edma_handle.rxWatermark (*C var*), 567
drivers_MCX_MCXW_MCXW727C._lpspi_slave_edma_handle.state (*C var*), 568
drivers_MCX_MCXW_MCXW727C._lpspi_slave_edma_handle.totalByteCount (*C var*), 568
drivers_MCX_MCXW_MCXW727C._lpspi_slave_edma_handle.txBuffIfNull (*C var*), 568
drivers_MCX_MCXW_MCXW727C._lpspi_slave_edma_handle.txData (*C var*), 567
drivers_MCX_MCXW_MCXW727C._lpspi_slave_edma_handle.txRemainingByteCount (*C var*), 567
drivers_MCX_MCXW_MCXW727C._lpspi_slave_edma_handle.userData (*C var*), 568
drivers_MCX_MCXW_MCXW727C._lpspi_slave_edma_handle.writeRegRemainingTimes (*C var*), 568
drivers_MCX_MCXW_MCXW727C._lpspi_slave_handle (*C struct*), 559
drivers_MCX_MCXW_MCXW727C._lpspi_slave_handle.bytesEachRead (*C var*), 560
drivers_MCX_MCXW_MCXW727C._lpspi_slave_handle.bytesEachWrite (*C var*), 560
drivers_MCX_MCXW_MCXW727C._lpspi_slave_handle.callback (*C var*), 560
drivers_MCX_MCXW_MCXW727C._lpspi_slave_handle.errorCount (*C var*), 560

drivers_MCX_MCXW_MCXW727C._lpspi_slave_handle.fifoSize (*C var*), 559
drivers_MCX_MCXW_MCXW727C._lpspi_slave_handle.isByteSwap (*C var*), 559
drivers_MCX_MCXW_MCXW727C._lpspi_slave_handle.readRegRemainingTimes (*C var*), 560
drivers_MCX_MCXW_MCXW727C._lpspi_slave_handle.rxData (*C var*), 560
drivers_MCX_MCXW_MCXW727C._lpspi_slave_handle.rxRemainingByteCount (*C var*), 560
drivers_MCX_MCXW_MCXW727C._lpspi_slave_handle.rxWatermark (*C var*), 559
drivers_MCX_MCXW_MCXW727C._lpspi_slave_handle.state (*C var*), 560
drivers_MCX_MCXW_MCXW727C._lpspi_slave_handle.totalByteCount (*C var*), 560
drivers_MCX_MCXW_MCXW727C._lpspi_slave_handle.txData (*C var*), 560
drivers_MCX_MCXW_MCXW727C._lpspi_slave_handle.txRemainingByteCount (*C var*), 560
drivers_MCX_MCXW_MCXW727C._lpspi_slave_handle.userData (*C var*), 560
drivers_MCX_MCXW_MCXW727C._lpspi_slave_handle.writeRegRemainingTimes (*C var*), 560
drivers_MCX_MCXW_MCXW727C._lpspi_transfer (*C struct*), 557
drivers_MCX_MCXW_MCXW727C._lpspi_transfer_config_flag_for_master (*C enum*), 552
drivers_MCX_MCXW_MCXW727C._lpspi_transfer_config_flag_for_master.kLPSPI_MasterByteSwap (*C enumerator*), 553
drivers_MCX_MCXW_MCXW727C._lpspi_transfer_config_flag_for_master.kLPSPI_MasterPcs0 (*C enumerator*), 552
drivers_MCX_MCXW_MCXW727C._lpspi_transfer_config_flag_for_master.kLPSPI_MasterPcs1 (*C enumerator*), 552
drivers_MCX_MCXW_MCXW727C._lpspi_transfer_config_flag_for_master.kLPSPI_MasterPcs2 (*C enumerator*), 553
drivers_MCX_MCXW_MCXW727C._lpspi_transfer_config_flag_for_master.kLPSPI_MasterPcs3 (*C enumerator*), 553
drivers_MCX_MCXW_MCXW727C._lpspi_transfer_config_flag_for_master.kLPSPI_MasterPcsContinuous (*C enumerator*), 553
drivers_MCX_MCXW_MCXW727C._lpspi_transfer_config_flag_for_slave (*C enum*), 553
drivers_MCX_MCXW_MCXW727C._lpspi_transfer_config_flag_for_slave.kLPSPI_SlaveByteSwap (*C enumerator*), 553
drivers_MCX_MCXW_MCXW727C._lpspi_transfer_config_flag_for_slave.kLPSPI_SlavePcs0 (*C enumerator*), 553
drivers_MCX_MCXW_MCXW727C._lpspi_transfer_config_flag_for_slave.kLPSPI_SlavePcs1 (*C enumerator*), 553
drivers_MCX_MCXW_MCXW727C._lpspi_transfer_config_flag_for_slave.kLPSPI_SlavePcs2 (*C enumerator*), 553
drivers_MCX_MCXW_MCXW727C._lpspi_transfer_config_flag_for_slave.kLPSPI_SlavePcs3 (*C enumerator*), 553
drivers_MCX_MCXW_MCXW727C._lpspi_transfer_state (*C enum*), 554
drivers_MCX_MCXW_MCXW727C._lpspi_transfer_state.kLPSPI_Busy (*C enumerator*), 554
drivers_MCX_MCXW_MCXW727C._lpspi_transfer_state.kLPSPI_Error (*C enumerator*), 554
drivers_MCX_MCXW_MCXW727C._lpspi_transfer_state.kLPSPI_Idle (*C enumerator*), 554
drivers_MCX_MCXW_MCXW727C._lpspi_transfer_width (*C enum*), 552
drivers_MCX_MCXW_MCXW727C._lpspi_transfer_width.kLPSPI_FourBitXfer (*C enumerator*), 552
drivers_MCX_MCXW_MCXW727C._lpspi_transfer_width.kLPSPI_SingleBitXfer (*C enumerator*), 552
drivers_MCX_MCXW_MCXW727C._lpspi_transfer_width.kLPSPI_TwoBitXfer (*C enumerator*), 552
drivers_MCX_MCXW_MCXW727C._lpspi_transfer.configFlags (*C var*), 558
drivers_MCX_MCXW_MCXW727C._lpspi_transfer.dataSize (*C var*), 558
drivers_MCX_MCXW_MCXW727C._lpspi_transfer.rxData (*C var*), 558
drivers_MCX_MCXW_MCXW727C._lpspi_transfer.txData (*C var*), 558
drivers_MCX_MCXW_MCXW727C._lpspi_which_pcs_config (*C enum*), 549
drivers_MCX_MCXW_MCXW727C._lpspi_which_pcs_config.kLPSPI_Pcs0 (*C enumerator*), 549
drivers_MCX_MCXW_MCXW727C._lpspi_which_pcs_config.kLPSPI_Pcs1 (*C enumerator*), 549
drivers_MCX_MCXW_MCXW727C._lpspi_which_pcs_config.kLPSPI_Pcs2 (*C enumerator*), 549
drivers_MCX_MCXW_MCXW727C._lpspi_which_pcs_config.kLPSPI_Pcs3 (*C enumerator*), 549
drivers_MCX_MCXW_MCXW727C._lptmr_config (*C struct*), 574

drivers_MCX_MCXW_MCXW727C._lptmr_config.bypassPrescaler (*C var*), 574
drivers_MCX_MCXW_MCXW727C._lptmr_config.enableFreeRunning (*C var*), 574
drivers_MCX_MCXW_MCXW727C._lptmr_config.pinPolarity (*C var*), 574
drivers_MCX_MCXW_MCXW727C._lptmr_config.pinSelect (*C var*), 574
drivers_MCX_MCXW_MCXW727C._lptmr_config.prescalerClockSource (*C var*), 574
drivers_MCX_MCXW_MCXW727C._lptmr_config.timerMode (*C var*), 574
drivers_MCX_MCXW_MCXW727C._lptmr_config.value (*C var*), 575
drivers_MCX_MCXW_MCXW727C._lptmr_interrupt_enable (*C enum*), 573
drivers_MCX_MCXW_MCXW727C._lptmr_interrupt_enable.kLPTMR_TimerInterruptEnable (*C enumerator*), 573
drivers_MCX_MCXW_MCXW727C._lptmr_pin_polarity (*C enum*), 571
drivers_MCX_MCXW_MCXW727C._lptmr_pin_polarity.kLPTMR_PinPolarityActiveHigh (*C enumerator*), 571
drivers_MCX_MCXW_MCXW727C._lptmr_pin_polarity.kLPTMR_PinPolarityActiveLow (*C enumerator*), 571
drivers_MCX_MCXW_MCXW727C._lptmr_pin_select (*C enum*), 571
drivers_MCX_MCXW_MCXW727C._lptmr_pin_select.kLPTMR_PinSelectInput_0 (*C enumerator*), 571
drivers_MCX_MCXW_MCXW727C._lptmr_pin_select.kLPTMR_PinSelectInput_1 (*C enumerator*), 571
drivers_MCX_MCXW_MCXW727C._lptmr_pin_select.kLPTMR_PinSelectInput_2 (*C enumerator*), 571
drivers_MCX_MCXW_MCXW727C._lptmr_pin_select.kLPTMR_PinSelectInput_3 (*C enumerator*), 571
drivers_MCX_MCXW_MCXW727C._lptmr_prescaler_clock_select (*C enum*), 572
drivers_MCX_MCXW_MCXW727C._lptmr_prescaler_glitch_value (*C enum*), 571
drivers_MCX_MCXW_MCXW727C._lptmr_prescaler_glitch_value.kLPTMR_Prescale_Glitch_0 (*C enumerator*), 571
drivers_MCX_MCXW_MCXW727C._lptmr_prescaler_glitch_value.kLPTMR_Prescale_Glitch_1 (*C enumerator*), 572
drivers_MCX_MCXW_MCXW727C._lptmr_prescaler_glitch_value.kLPTMR_Prescale_Glitch_2 (*C enumerator*), 572
drivers_MCX_MCXW_MCXW727C._lptmr_prescaler_glitch_value.kLPTMR_Prescale_Glitch_3 (*C enumerator*), 572
drivers_MCX_MCXW_MCXW727C._lptmr_prescaler_glitch_value.kLPTMR_Prescale_Glitch_4 (*C enumerator*), 572
drivers_MCX_MCXW_MCXW727C._lptmr_prescaler_glitch_value.kLPTMR_Prescale_Glitch_5 (*C enumerator*), 572
drivers_MCX_MCXW_MCXW727C._lptmr_prescaler_glitch_value.kLPTMR_Prescale_Glitch_6 (*C enumerator*), 572
drivers_MCX_MCXW_MCXW727C._lptmr_prescaler_glitch_value.kLPTMR_Prescale_Glitch_7 (*C enumerator*), 572
drivers_MCX_MCXW_MCXW727C._lptmr_prescaler_glitch_value.kLPTMR_Prescale_Glitch_8 (*C enumerator*), 572
drivers_MCX_MCXW_MCXW727C._lptmr_prescaler_glitch_value.kLPTMR_Prescale_Glitch_9 (*C enumerator*), 572
drivers_MCX_MCXW_MCXW727C._lptmr_prescaler_glitch_value.kLPTMR_Prescale_Glitch_10 (*C enumerator*), 572
drivers_MCX_MCXW_MCXW727C._lptmr_prescaler_glitch_value.kLPTMR_Prescale_Glitch_11 (*C enumerator*), 572
drivers_MCX_MCXW_MCXW727C._lptmr_prescaler_glitch_value.kLPTMR_Prescale_Glitch_12 (*C enumerator*), 572
drivers_MCX_MCXW_MCXW727C._lptmr_prescaler_glitch_value.kLPTMR_Prescale_Glitch_13 (*C enumerator*), 572
drivers_MCX_MCXW_MCXW727C._lptmr_prescaler_glitch_value.kLPTMR_Prescale_Glitch_14 (*C enumerator*), 572
drivers_MCX_MCXW_MCXW727C._lptmr_prescaler_glitch_value.kLPTMR_Prescale_Glitch_15 (*C enumerator*), 572

drivers_MCX_MCXW_MCXW727C._lptmr_status_flags (*C enum*), 573
drivers_MCX_MCXW_MCXW727C._lptmr_status_flags.kLPTMR_TimerCompareFlag (*C enumerator*), 573
drivers_MCX_MCXW_MCXW727C._lptmr_timer_mode (*C enum*), 571
drivers_MCX_MCXW_MCXW727C._lptmr_timer_mode.kLPTMR_TimerModePulseCounter (*C enumerator*), 571
drivers_MCX_MCXW_MCXW727C._lptmr_timer_mode.kLPTMR_TimerModeTimeCounter (*C enumerator*), 571
drivers_MCX_MCXW_MCXW727C._lpuart_config (*C struct*), 592
drivers_MCX_MCXW_MCXW727C._lpuart_config.baudRate_Bps (*C var*), 593
drivers_MCX_MCXW_MCXW727C._lpuart_config.dataBitsCount (*C var*), 593
drivers_MCX_MCXW_MCXW727C._lpuart_config.enableRx (*C var*), 594
drivers_MCX_MCXW_MCXW727C._lpuart_config.enableRxRTS (*C var*), 593
drivers_MCX_MCXW_MCXW727C._lpuart_config.enableTx (*C var*), 593
drivers_MCX_MCXW_MCXW727C._lpuart_config.enableTxCTS (*C var*), 593
drivers_MCX_MCXW_MCXW727C._lpuart_config.enableTxRTS (*C var*), 593
drivers_MCX_MCXW_MCXW727C._lpuart_config.inverseTxd (*C var*), 594
drivers_MCX_MCXW_MCXW727C._lpuart_config.isMsb (*C var*), 593
drivers_MCX_MCXW_MCXW727C._lpuart_config.parityMode (*C var*), 593
drivers_MCX_MCXW_MCXW727C._lpuart_config.rtsWatermark (*C var*), 593
drivers_MCX_MCXW_MCXW727C._lpuart_config.rxFifoWatermark (*C var*), 593
drivers_MCX_MCXW_MCXW727C._lpuart_config.rxIdleConfig (*C var*), 593
drivers_MCX_MCXW_MCXW727C._lpuart_config.rxIdleType (*C var*), 593
drivers_MCX_MCXW_MCXW727C._lpuart_config.stopBitCount (*C var*), 593
drivers_MCX_MCXW_MCXW727C._lpuart_config.swapTxdRxd (*C var*), 594
drivers_MCX_MCXW_MCXW727C._lpuart_config.txCtsConfig (*C var*), 593
drivers_MCX_MCXW_MCXW727C._lpuart_config.txCtsSource (*C var*), 593
drivers_MCX_MCXW_MCXW727C._lpuart_config.txFifoWatermark (*C var*), 593
drivers_MCX_MCXW_MCXW727C._lpuart_config.txRtsPolarity (*C var*), 593
drivers_MCX_MCXW_MCXW727C._lpuart_data_bits (*C enum*), 587
drivers_MCX_MCXW_MCXW727C._lpuart_data_bits.kLPUART_EightDataBits (*C enumerator*), 587
drivers_MCX_MCXW_MCXW727C._lpuart_data_bits.kLPUART_SevenDataBits (*C enumerator*), 587
drivers_MCX_MCXW_MCXW727C._lpuart_edma_handle (*C struct*), 598
drivers_MCX_MCXW_MCXW727C._lpuart_edma_handle.callback (*C var*), 599
drivers_MCX_MCXW_MCXW727C._lpuart_edma_handle.nbytes (*C var*), 599
drivers_MCX_MCXW_MCXW727C._lpuart_edma_handle.rxDataSizeAll (*C var*), 599
drivers_MCX_MCXW_MCXW727C._lpuart_edma_handle.rxEdmaHandle (*C var*), 599
drivers_MCX_MCXW_MCXW727C._lpuart_edma_handle.rxState (*C var*), 599
drivers_MCX_MCXW_MCXW727C._lpuart_edma_handle.txDataSizeAll (*C var*), 599
drivers_MCX_MCXW_MCXW727C._lpuart_edma_handle.txEdmaHandle (*C var*), 599
drivers_MCX_MCXW_MCXW727C._lpuart_edma_handle.txState (*C var*), 599
drivers_MCX_MCXW_MCXW727C._lpuart_edma_handle.userData (*C var*), 599
drivers_MCX_MCXW_MCXW727C._lpuart_flags (*C enum*), 590
drivers_MCX_MCXW_MCXW727C._lpuart_flags.kLPUART_AllClearFlags (*C enumerator*), 591
drivers_MCX_MCXW_MCXW727C._lpuart_flags.kLPUART_AllFlags (*C enumerator*), 591
drivers_MCX_MCXW_MCXW727C._lpuart_flags.kLPUART_DataMatch1Flag (*C enumerator*), 591
drivers_MCX_MCXW_MCXW727C._lpuart_flags.kLPUART_DataMatch2Flag (*C enumerator*), 591
drivers_MCX_MCXW_MCXW727C._lpuart_flags.kLPUART_FramingErrorFlag (*C enumerator*), 591
drivers_MCX_MCXW_MCXW727C._lpuart_flags.kLPUART_IdleLineFlag (*C enumerator*), 590
drivers_MCX_MCXW_MCXW727C._lpuart_flags.kLPUART_LinBreakFlag (*C enumerator*), 591
drivers_MCX_MCXW_MCXW727C._lpuart_flags.kLPUART_NoiseErrorFlag (*C enumerator*), 591
drivers_MCX_MCXW_MCXW727C._lpuart_flags.kLPUART_ParityErrorFlag (*C enumerator*), 591

drivers_MCX_MCXW_MCXW727C._lpuart_flags.kLPUART_RxActiveEdgeFlag (*C enumerator*), 591
drivers_MCX_MCXW_MCXW727C._lpuart_flags.kLPUART_RxActiveFlag (*C enumerator*), 591
drivers_MCX_MCXW_MCXW727C._lpuart_flags.kLPUART_RxDataRegFullFlag (*C enumerator*), 590
drivers_MCX_MCXW_MCXW727C._lpuart_flags.kLPUART_RxFifoEmptyFlag (*C enumerator*), 591
drivers_MCX_MCXW_MCXW727C._lpuart_flags.kLPUART_RxFifoUnderflowFlag (*C enumerator*), 591
drivers_MCX_MCXW_MCXW727C._lpuart_flags.kLPUART_RxOverrunFlag (*C enumerator*), 591
drivers_MCX_MCXW_MCXW727C._lpuart_flags.kLPUART_TransmissionCompleteFlag (*C enumerator*), 590
drivers_MCX_MCXW_MCXW727C._lpuart_flags.kLPUART_TxDataRegEmptyFlag (*C enumerator*), 590
drivers_MCX_MCXW_MCXW727C._lpuart_flags.kLPUART_TxFifoEmptyFlag (*C enumerator*), 591
drivers_MCX_MCXW_MCXW727C._lpuart_flags.kLPUART_TxFifoOverflowFlag (*C enumerator*), 591
drivers_MCX_MCXW_MCXW727C._lpuart_handle (*C struct*), 594
drivers_MCX_MCXW_MCXW727C._lpuart_handle.__unnamed60__ (*C union*), 595
drivers_MCX_MCXW_MCXW727C._lpuart_handle.__unnamed60__.txData (*C var*), 595
drivers_MCX_MCXW_MCXW727C._lpuart_handle.__unnamed60__.txData16 (*C var*), 595
drivers_MCX_MCXW_MCXW727C._lpuart_handle.__unnamed62__ (*C union*), 595
drivers_MCX_MCXW_MCXW727C._lpuart_handle.__unnamed62__.rxData (*C var*), 596
drivers_MCX_MCXW_MCXW727C._lpuart_handle.__unnamed62__.rxData16 (*C var*), 596
drivers_MCX_MCXW_MCXW727C._lpuart_handle.__unnamed64__ (*C union*), 596
drivers_MCX_MCXW_MCXW727C._lpuart_handle.__unnamed64__.rxRingBuffer (*C var*), 596
drivers_MCX_MCXW_MCXW727C._lpuart_handle.__unnamed64__.rxRingBuffer16 (*C var*), 596
drivers_MCX_MCXW_MCXW727C._lpuart_handle.callback (*C var*), 594
drivers_MCX_MCXW_MCXW727C._lpuart_handle.is16bitData (*C var*), 595
drivers_MCX_MCXW_MCXW727C._lpuart_handle.isSevenDataBits (*C var*), 595
drivers_MCX_MCXW_MCXW727C._lpuart_handle.rxDataSize (*C var*), 594
drivers_MCX_MCXW_MCXW727C._lpuart_handle.rxDataSizeAll (*C var*), 594
drivers_MCX_MCXW_MCXW727C._lpuart_handle.rxRingBufferHead (*C var*), 594
drivers_MCX_MCXW_MCXW727C._lpuart_handle.rxRingBufferSize (*C var*), 594
drivers_MCX_MCXW_MCXW727C._lpuart_handle.rxRingBufferTail (*C var*), 594
drivers_MCX_MCXW_MCXW727C._lpuart_handle.rxState (*C var*), 595
drivers_MCX_MCXW_MCXW727C._lpuart_handle.txDataSize (*C var*), 594
drivers_MCX_MCXW_MCXW727C._lpuart_handle.txDataSizeAll (*C var*), 594
drivers_MCX_MCXW_MCXW727C._lpuart_handle.txState (*C var*), 595
drivers_MCX_MCXW_MCXW727C._lpuart_handle.userData (*C var*), 595
drivers_MCX_MCXW_MCXW727C._lpuart_idle_config (*C enum*), 589
drivers_MCX_MCXW_MCXW727C._lpuart_idle_config.kLPUART_IdleCharacter1 (*C enumerator*), 589
drivers_MCX_MCXW_MCXW727C._lpuart_idle_config.kLPUART_IdleCharacter2 (*C enumerator*), 589
drivers_MCX_MCXW_MCXW727C._lpuart_idle_config.kLPUART_IdleCharacter4 (*C enumerator*), 589
drivers_MCX_MCXW_MCXW727C._lpuart_idle_config.kLPUART_IdleCharacter8 (*C enumerator*), 589
drivers_MCX_MCXW_MCXW727C._lpuart_idle_config.kLPUART_IdleCharacter16 (*C enumerator*), 589
drivers_MCX_MCXW_MCXW727C._lpuart_idle_config.kLPUART_IdleCharacter32 (*C enumerator*), 589
drivers_MCX_MCXW_MCXW727C._lpuart_idle_config.kLPUART_IdleCharacter64 (*C enumerator*), 589
drivers_MCX_MCXW_MCXW727C._lpuart_idle_config.kLPUART_IdleCharacter128 (*C enumerator*), 589

drivers_MCX_MCXW_MCXW727C._lpuart_idle_type_select (*C enum*), 588
drivers_MCX_MCXW_MCXW727C._lpuart_idle_type_select.kLPUART_IdleTypeStartBit (*C enumerator*), 588
drivers_MCX_MCXW_MCXW727C._lpuart_idle_type_select.kLPUART_IdleTypeStopBit (*C enumerator*), 589
drivers_MCX_MCXW_MCXW727C._lpuart_interrupt_enable (*C enum*), 589
drivers_MCX_MCXW_MCXW727C._lpuart_interrupt_enable.kLPUART_AllInterruptEnable (*C enumerator*), 590
drivers_MCX_MCXW_MCXW727C._lpuart_interrupt_enable.kLPUART_FramingErrorInterruptEnable (*C enumerator*), 590
drivers_MCX_MCXW_MCXW727C._lpuart_interrupt_enable.kLPUART_IdleLineInterruptEnable (*C enumerator*), 590
drivers_MCX_MCXW_MCXW727C._lpuart_interrupt_enable.kLPUART_LinBreakInterruptEnable (*C enumerator*), 589
drivers_MCX_MCXW_MCXW727C._lpuart_interrupt_enable.kLPUART_Match1InterruptEnable (*C enumerator*), 590
drivers_MCX_MCXW_MCXW727C._lpuart_interrupt_enable.kLPUART_Match2InterruptEnable (*C enumerator*), 590
drivers_MCX_MCXW_MCXW727C._lpuart_interrupt_enable.kLPUART_NoiseErrorInterruptEnable (*C enumerator*), 590
drivers_MCX_MCXW_MCXW727C._lpuart_interrupt_enable.kLPUART_ParityErrorInterruptEnable (*C enumerator*), 590
drivers_MCX_MCXW_MCXW727C._lpuart_interrupt_enable.kLPUART_RxActiveEdgeInterruptEnable (*C enumerator*), 589
drivers_MCX_MCXW_MCXW727C._lpuart_interrupt_enable.kLPUART_RxDataRegFullInterruptEnable (*C enumerator*), 590
drivers_MCX_MCXW_MCXW727C._lpuart_interrupt_enable.kLPUART_RxFifoUnderflowInterruptEnable (*C enumerator*), 590
drivers_MCX_MCXW_MCXW727C._lpuart_interrupt_enable.kLPUART_RxOverrunInterruptEnable (*C enumerator*), 590
drivers_MCX_MCXW_MCXW727C._lpuart_interrupt_enable.kLPUART_TransmissionCompleteInterruptEnable (*C enumerator*), 589
drivers_MCX_MCXW_MCXW727C._lpuart_interrupt_enable.kLPUART_TxDataRegEmptyInterruptEnable (*C enumerator*), 589
drivers_MCX_MCXW_MCXW727C._lpuart_interrupt_enable.kLPUART_TxFifoOverflowInterruptEnable (*C enumerator*), 590
drivers_MCX_MCXW_MCXW727C._lpuart_parity_mode (*C enum*), 587
drivers_MCX_MCXW_MCXW727C._lpuart_parity_mode.kLPUART_ParityDisabled (*C enumerator*), 587
drivers_MCX_MCXW_MCXW727C._lpuart_parity_mode.kLPUART_ParityEven (*C enumerator*), 587
drivers_MCX_MCXW_MCXW727C._lpuart_parity_mode.kLPUART_ParityOdd (*C enumerator*), 587
drivers_MCX_MCXW_MCXW727C._lpuart_stop_bit_count (*C enum*), 588
drivers_MCX_MCXW_MCXW727C._lpuart_stop_bit_count.kLPUART_OneStopBit (*C enumerator*), 588
drivers_MCX_MCXW_MCXW727C._lpuart_stop_bit_count.kLPUART_TwoStopBit (*C enumerator*), 588
drivers_MCX_MCXW_MCXW727C._lpuart_transfer (*C struct*), 594
drivers_MCX_MCXW_MCXW727C._lpuart_transfer.__unnamed58__ (*C union*), 595
drivers_MCX_MCXW_MCXW727C._lpuart_transfer.__unnamed58__.data (*C var*), 595
drivers_MCX_MCXW_MCXW727C._lpuart_transfer.__unnamed58__.rxData (*C var*), 595
drivers_MCX_MCXW_MCXW727C._lpuart_transfer.__unnamed58__.rxData16 (*C var*), 595
drivers_MCX_MCXW_MCXW727C._lpuart_transfer.__unnamed58__.txData (*C var*), 595
drivers_MCX_MCXW_MCXW727C._lpuart_transfer.__unnamed58__.txData16 (*C var*), 595
drivers_MCX_MCXW_MCXW727C._lpuart_transfer.dataSize (*C var*), 594
drivers_MCX_MCXW_MCXW727C._lpuart_transmit_cts_config (*C enum*), 588
drivers_MCX_MCXW_MCXW727C._lpuart_transmit_cts_config.kLPUART_CtsSampleAtIdle (*C*

enumerator), 588
drivers_MCX_MCXW_MCXW727C._lpuart_transmit_cts_config.kLPUART_CtsSampleAtStart (*C enumerator*), 588
drivers_MCX_MCXW_MCXW727C._lpuart_transmit_cts_source (*C enum*), 588
drivers_MCX_MCXW_MCXW727C._lpuart_transmit_cts_source.kLPUART_CtsSourceMatchResult (*C enumerator*), 588
drivers_MCX_MCXW_MCXW727C._lpuart_transmit_cts_source.kLPUART_CtsSourcePin (*C enumerator*), 588
drivers_MCX_MCXW_MCXW727C._lpuart_transmit_rts_polarity (*C enum*), 588
drivers_MCX_MCXW_MCXW727C._lpuart_transmit_rts_polarity.kLPUART_RtsPolarityHigh (*C enumerator*), 588
drivers_MCX_MCXW_MCXW727C._lpuart_transmit_rts_polarity.kLPUART_RtsPolarityLow (*C enumerator*), 588
drivers_MCX_MCXW_MCXW727C._ltc_aes_key_t (*C enum*), 601
drivers_MCX_MCXW_MCXW727C._ltc_aes_key_t.kLTC_EncryptKey (*C enumerator*), 601
drivers_MCX_MCXW_MCXW727C._ltc_hash_algo_t (*C enum*), 614
drivers_MCX_MCXW_MCXW727C._ltc_hash_algo_t.kLTC_Cmac (*C enumerator*), 614
drivers_MCX_MCXW_MCXW727C._ltc_hash_algo_t.kLTC_Sha1 (*C enumerator*), 614
drivers_MCX_MCXW_MCXW727C._ltc_hash_algo_t.kLTC_Sha224 (*C enumerator*), 615
drivers_MCX_MCXW_MCXW727C._ltc_hash_algo_t.kLTC_Sha256 (*C enumerator*), 615
drivers_MCX_MCXW_MCXW727C._ltc_hash_ctx_t (*C struct*), 616
drivers_MCX_MCXW_MCXW727C._ltc_pkha_ecc_point_t (*C struct*), 625
drivers_MCX_MCXW_MCXW727C._ltc_pkha_ecc_point_t.X (*C var*), 625
drivers_MCX_MCXW_MCXW727C._ltc_pkha_ecc_point_t.Y (*C var*), 625
drivers_MCX_MCXW_MCXW727C._ltc_pkha_f2m_t (*C enum*), 616
drivers_MCX_MCXW_MCXW727C._ltc_pkha_f2m_t.kLTC_PKHA_F2mArith (*C enumerator*), 617
drivers_MCX_MCXW_MCXW727C._ltc_pkha_f2m_t.kLTC_PKHA_IntegerArith (*C enumerator*), 616
drivers_MCX_MCXW_MCXW727C._ltc_pkha_montgomery_form_t (*C enum*), 617
drivers_MCX_MCXW_MCXW727C._ltc_pkha_montgomery_form_t.kLTC_PKHA_MontgomeryFormat (*C enumerator*), 617
drivers_MCX_MCXW_MCXW727C._ltc_pkha_montgomery_form_t.kLTC_PKHA_NormalValue (*C enumerator*), 617
drivers_MCX_MCXW_MCXW727C._ltc_pkha_timing_t (*C enum*), 616
drivers_MCX_MCXW_MCXW727C._ltc_pkha_timing_t.kLTC_PKHA_NoTimingEqualized (*C enumerator*), 616
drivers_MCX_MCXW_MCXW727C._ltc_pkha_timing_t.kLTC_PKHA_TimingEqualized (*C enumerator*), 616
drivers_MCX_MCXW_MCXW727C._mcm_buffer_fault_attribute (*C union*), 629
drivers_MCX_MCXW_MCXW727C._mcm_buffer_fault_attribute.__mcm_buffer_fault_attribut (*C struct*), 629
drivers_MCX_MCXW_MCXW727C._mcm_buffer_fault_attribute.__mcm_buffer_fault_attribut.busErrorAccess (*C var*), 630
drivers_MCX_MCXW_MCXW727C._mcm_buffer_fault_attribute.__mcm_buffer_fault_attribut.busErrorDataAcces (*C var*), 629
drivers_MCX_MCXW_MCXW727C._mcm_buffer_fault_attribute.__mcm_buffer_fault_attribut.busErrorMasterID (*C var*), 630
drivers_MCX_MCXW_MCXW727C._mcm_buffer_fault_attribute.__mcm_buffer_fault_attribut.busErrorOverrun (*C var*), 630
drivers_MCX_MCXW_MCXW727C._mcm_buffer_fault_attribute.__mcm_buffer_fault_attribut.busErrorPrivilegeL (*C var*), 629
drivers_MCX_MCXW_MCXW727C._mcm_buffer_fault_attribute.__mcm_buffer_fault_attribut.busErrorSize (*C var*), 629
drivers_MCX_MCXW_MCXW727C._mcm_buffer_fault_attribute.attribute (*C var*), 629
drivers_MCX_MCXW_MCXW727C._mcm_buffer_fault_attribute.attribute_memory (*C var*), 629
drivers_MCX_MCXW_MCXW727C._mcm_lmem_fault_attribute (*C union*), 630
drivers_MCX_MCXW_MCXW727C._mcm_lmem_fault_attribute.__mcm_lmem_fault_attribut (*C

struct), 630
drivers_MCX_MCXW_MCXW727C._mcm_lmem_fault_attribute._mcm_lmem_fault_attribut.backdoorAccess (*C var*), 630
drivers_MCX_MCXW_MCXW727C._mcm_lmem_fault_attribute._mcm_lmem_fault_attribut.overrun (*C var*), 630
drivers_MCX_MCXW_MCXW727C._mcm_lmem_fault_attribute._mcm_lmem_fault_attribut.parityFaultMasterSi (*C var*), 630
drivers_MCX_MCXW_MCXW727C._mcm_lmem_fault_attribute._mcm_lmem_fault_attribut.parityFaultProtectio (*C var*), 630
drivers_MCX_MCXW_MCXW727C._mcm_lmem_fault_attribute._mcm_lmem_fault_attribut.parityFaultSyndrom (*C var*), 630
drivers_MCX_MCXW_MCXW727C._mcm_lmem_fault_attribute._mcm_lmem_fault_attribut.parityFaultWrite (*C var*), 630
drivers_MCX_MCXW_MCXW727C._mcm_lmem_fault_attribute.attribute (*C var*), 630
drivers_MCX_MCXW_MCXW727C._mcm_lmem_fault_attribute.attribute_memory (*C var*), 630
drivers_MCX_MCXW_MCXW727C._mscm_uid (*C struct*), 631
drivers_MCX_MCXW_MCXW727C._msf1_config (*C struct*), 249
drivers_MCX_MCXW_MCXW727C._mu_general_purpose_interrupt (*C enum*), 643
drivers_MCX_MCXW_MCXW727C._mu_general_purpose_interrupt.kMU_GeneralPurposeInterrupt0 (*C enumerator*), 644
drivers_MCX_MCXW_MCXW727C._mu_general_purpose_interrupt.kMU_GeneralPurposeInterrupt1 (*C enumerator*), 644
drivers_MCX_MCXW_MCXW727C._mu_general_purpose_interrupt.kMU_GeneralPurposeInterrupt2 (*C enumerator*), 644
drivers_MCX_MCXW_MCXW727C._mu_general_purpose_interrupt.kMU_GeneralPurposeInterrupt3 (*C enumerator*), 644
drivers_MCX_MCXW_MCXW727C._mu_interrupt_enable (*C enum*), 642
drivers_MCX_MCXW_MCXW727C._mu_interrupt_enable.kMU_GenInt0InterruptEnable (*C enumerator*), 642
drivers_MCX_MCXW_MCXW727C._mu_interrupt_enable.kMU_GenInt1InterruptEnable (*C enumerator*), 643
drivers_MCX_MCXW_MCXW727C._mu_interrupt_enable.kMU_GenInt2InterruptEnable (*C enumerator*), 643
drivers_MCX_MCXW_MCXW727C._mu_interrupt_enable.kMU_GenInt3InterruptEnable (*C enumerator*), 643
drivers_MCX_MCXW_MCXW727C._mu_interrupt_enable.kMU_MuResetInterruptEnable (*C enumerator*), 643
drivers_MCX_MCXW_MCXW727C._mu_interrupt_enable.kMU_Rx0FullInterruptEnable (*C enumerator*), 642
drivers_MCX_MCXW_MCXW727C._mu_interrupt_enable.kMU_Rx1FullInterruptEnable (*C enumerator*), 642
drivers_MCX_MCXW_MCXW727C._mu_interrupt_enable.kMU_Rx2FullInterruptEnable (*C enumerator*), 642
drivers_MCX_MCXW_MCXW727C._mu_interrupt_enable.kMU_Rx3FullInterruptEnable (*C enumerator*), 642
drivers_MCX_MCXW_MCXW727C._mu_interrupt_enable.kMU_Tx0EmptyInterruptEnable (*C enumerator*), 642
drivers_MCX_MCXW_MCXW727C._mu_interrupt_enable.kMU_Tx1EmptyInterruptEnable (*C enumerator*), 642
drivers_MCX_MCXW_MCXW727C._mu_interrupt_enable.kMU_Tx2EmptyInterruptEnable (*C enumerator*), 642
drivers_MCX_MCXW_MCXW727C._mu_interrupt_enable.kMU_Tx3EmptyInterruptEnable (*C enumerator*), 642
drivers_MCX_MCXW_MCXW727C._mu_interrupt_trigger (*C enum*), 643
drivers_MCX_MCXW_MCXW727C._mu_interrupt_trigger.kMU_GenInt0InterruptTrigger (*C enumerator*), 643
drivers_MCX_MCXW_MCXW727C._mu_interrupt_trigger.kMU_GenInt1InterruptTrigger (*C enumerator*), 643

drivers_MCX_MCXW_MCXW727C._mu_interrupt_trigger.kMU_GenInt2InterruptTrigger (*C enumerator*), 643
drivers_MCX_MCXW_MCXW727C._mu_interrupt_trigger.kMU_GenInt3InterruptTrigger (*C enumerator*), 643
drivers_MCX_MCXW_MCXW727C._mu_msg_reg_index (*C enum*), 643
drivers_MCX_MCXW_MCXW727C._mu_msg_reg_index.kMU_MsgReg0 (*C enumerator*), 643
drivers_MCX_MCXW_MCXW727C._mu_msg_reg_index.kMU_MsgReg1 (*C enumerator*), 643
drivers_MCX_MCXW_MCXW727C._mu_msg_reg_index.kMU_MsgReg2 (*C enumerator*), 643
drivers_MCX_MCXW_MCXW727C._mu_msg_reg_index.kMU_MsgReg3 (*C enumerator*), 643
drivers_MCX_MCXW_MCXW727C._mu_status_flags (*C enum*), 641
drivers_MCX_MCXW_MCXW727C._mu_status_flags.kMU_EventPendingFlag (*C enumerator*), 642
drivers_MCX_MCXW_MCXW727C._mu_status_flags.kMU_FlagsUpdatingFlag (*C enumerator*), 642
drivers_MCX_MCXW_MCXW727C._mu_status_flags.kMU_GenInt0Flag (*C enumerator*), 641
drivers_MCX_MCXW_MCXW727C._mu_status_flags.kMU_GenInt1Flag (*C enumerator*), 641
drivers_MCX_MCXW_MCXW727C._mu_status_flags.kMU_GenInt2Flag (*C enumerator*), 641
drivers_MCX_MCXW_MCXW727C._mu_status_flags.kMU_GenInt3Flag (*C enumerator*), 641
drivers_MCX_MCXW_MCXW727C._mu_status_flags.kMU_GenIntPendingFlag (*C enumerator*), 642
drivers_MCX_MCXW_MCXW727C._mu_status_flags.kMU_MuInResetFlag (*C enumerator*), 642
drivers_MCX_MCXW_MCXW727C._mu_status_flags.kMU_MuResetInterruptFlag (*C enumerator*), 642
drivers_MCX_MCXW_MCXW727C._mu_status_flags.kMU_Rx0FullFlag (*C enumerator*), 641
drivers_MCX_MCXW_MCXW727C._mu_status_flags.kMU_Rx1FullFlag (*C enumerator*), 641
drivers_MCX_MCXW_MCXW727C._mu_status_flags.kMU_Rx2FullFlag (*C enumerator*), 641
drivers_MCX_MCXW_MCXW727C._mu_status_flags.kMU_Rx3FullFlag (*C enumerator*), 641
drivers_MCX_MCXW_MCXW727C._mu_status_flags.kMU_RxFullPendingFlag (*C enumerator*), 641
drivers_MCX_MCXW_MCXW727C._mu_status_flags.kMU_Tx0EmptyFlag (*C enumerator*), 641
drivers_MCX_MCXW_MCXW727C._mu_status_flags.kMU_Tx1EmptyFlag (*C enumerator*), 641
drivers_MCX_MCXW_MCXW727C._mu_status_flags.kMU_Tx2EmptyFlag (*C enumerator*), 641
drivers_MCX_MCXW_MCXW727C._mu_status_flags.kMU_Tx3EmptyFlag (*C enumerator*), 641
drivers_MCX_MCXW_MCXW727C._mu_status_flags.kMU_TxEmptyPendingFlag (*C enumerator*), 642
drivers_MCX_MCXW_MCXW727C._port_digital_filter_clock_source (*C enum*), 652
drivers_MCX_MCXW_MCXW727C._port_digital_filter_clock_source.kPORT_BusClock (*C enumerator*), 652
drivers_MCX_MCXW_MCXW727C._port_digital_filter_clock_source.kPORT_LpoClock (*C enumerator*), 652
drivers_MCX_MCXW_MCXW727C._port_digital_filter_config (*C struct*), 652
drivers_MCX_MCXW_MCXW727C._port_digital_filter_config.clockSource (*C var*), 653
drivers_MCX_MCXW_MCXW727C._port_digital_filter_config.digitalFilterWidth (*C var*), 653
drivers_MCX_MCXW_MCXW727C._port_drive_strength (*C enum*), 649
drivers_MCX_MCXW_MCXW727C._port_drive_strength1 (*C enum*), 649
drivers_MCX_MCXW_MCXW727C._port_drive_strength1.kPORT_DoubleDriveStrength (*C enumerator*), 650
drivers_MCX_MCXW_MCXW727C._port_drive_strength1.kPORT_NormalDriveStrength (*C enumerator*), 650
drivers_MCX_MCXW_MCXW727C._port_drive_strength.kPORT_HighDriveStrength (*C enumerator*), 649
drivers_MCX_MCXW_MCXW727C._port_drive_strength.kPORT_LowDriveStrength (*C enumerator*), 649
drivers_MCX_MCXW_MCXW727C._port_input_buffer (*C enum*), 650
drivers_MCX_MCXW_MCXW727C._port_input_buffer.kPORT_InputBufferDisable (*C enumerator*), 650
drivers_MCX_MCXW_MCXW727C._port_input_buffer.kPORT_InputBufferEnable (*C enumerator*), 650

drivers_MCX_MCXW_MCXW727C._port_invet_input (*C enum*), 650
drivers_MCX_MCXW_MCXW727C._port_invet_input.kPORT_InputInvert (*C enumerator*), 650
drivers_MCX_MCXW_MCXW727C._port_invet_input.kPORT_InputNormal (*C enumerator*), 650
drivers_MCX_MCXW_MCXW727C._port_lock_register (*C enum*), 650
drivers_MCX_MCXW_MCXW727C._port_lock_register.kPORT_LockRegister (*C enumerator*), 650
drivers_MCX_MCXW_MCXW727C._port_lock_register.kPORT_UnlockRegister (*C enumerator*), 650
drivers_MCX_MCXW_MCXW727C._port_mux (*C enum*), 650
drivers_MCX_MCXW_MCXW727C._port_mux.kPORT_MuxAlt0 (*C enumerator*), 651
drivers_MCX_MCXW_MCXW727C._port_mux.kPORT_MuxAlt1 (*C enumerator*), 651
drivers_MCX_MCXW_MCXW727C._port_mux.kPORT_MuxAlt2 (*C enumerator*), 651
drivers_MCX_MCXW_MCXW727C._port_mux.kPORT_MuxAlt3 (*C enumerator*), 651
drivers_MCX_MCXW_MCXW727C._port_mux.kPORT_MuxAlt4 (*C enumerator*), 651
drivers_MCX_MCXW_MCXW727C._port_mux.kPORT_MuxAlt5 (*C enumerator*), 651
drivers_MCX_MCXW_MCXW727C._port_mux.kPORT_MuxAlt6 (*C enumerator*), 651
drivers_MCX_MCXW_MCXW727C._port_mux.kPORT_MuxAlt7 (*C enumerator*), 651
drivers_MCX_MCXW_MCXW727C._port_mux.kPORT_MuxAlt8 (*C enumerator*), 651
drivers_MCX_MCXW_MCXW727C._port_mux.kPORT_MuxAlt9 (*C enumerator*), 651
drivers_MCX_MCXW_MCXW727C._port_mux.kPORT_MuxAlt10 (*C enumerator*), 651
drivers_MCX_MCXW_MCXW727C._port_mux.kPORT_MuxAlt11 (*C enumerator*), 651
drivers_MCX_MCXW_MCXW727C._port_mux.kPORT_MuxAlt12 (*C enumerator*), 651
drivers_MCX_MCXW_MCXW727C._port_mux.kPORT_MuxAlt13 (*C enumerator*), 651
drivers_MCX_MCXW_MCXW727C._port_mux.kPORT_MuxAlt14 (*C enumerator*), 651
drivers_MCX_MCXW_MCXW727C._port_mux.kPORT_MuxAlt15 (*C enumerator*), 651
drivers_MCX_MCXW_MCXW727C._port_mux.kPORT_MuxAsGpio (*C enumerator*), 650
drivers_MCX_MCXW_MCXW727C._port_mux.kPORT_PinDisabledOrAnalog (*C enumerator*), 650
drivers_MCX_MCXW_MCXW727C._port_open_drain_enable (*C enum*), 649
drivers_MCX_MCXW_MCXW727C._port_open_drain_enable.kPORT_OpenDrainDisable (*C enumerator*), 649
drivers_MCX_MCXW_MCXW727C._port_open_drain_enable.kPORT_OpenDrainEnable (*C enumerator*), 649
drivers_MCX_MCXW_MCXW727C._port_passive_filter_enable (*C enum*), 649
drivers_MCX_MCXW_MCXW727C._port_passive_filter_enable.kPORT_PassiveFilterDisable (*C enumerator*), 649
drivers_MCX_MCXW_MCXW727C._port_passive_filter_enable.kPORT_PassiveFilterEnable (*C enumerator*), 649
drivers_MCX_MCXW_MCXW727C._port_pin_config (*C struct*), 653
drivers_MCX_MCXW_MCXW727C._port_pin_config.driveStrength (*C var*), 653
drivers_MCX_MCXW_MCXW727C._port_pin_config.driveStrength1 (*C var*), 653
drivers_MCX_MCXW_MCXW727C._port_pin_config.inputBuffer (*C var*), 653
drivers_MCX_MCXW_MCXW727C._port_pin_config.invertInput (*C var*), 653
drivers_MCX_MCXW_MCXW727C._port_pin_config.lockRegister (*C var*), 653
drivers_MCX_MCXW_MCXW727C._port_pin_config.openDrainEnable (*C var*), 653
drivers_MCX_MCXW_MCXW727C._port_pin_config.passiveFilterEnable (*C var*), 653
drivers_MCX_MCXW_MCXW727C._port_pin_config.pullSelect (*C var*), 653
drivers_MCX_MCXW_MCXW727C._port_pin_config.pullValueSelect (*C var*), 653
drivers_MCX_MCXW_MCXW727C._port_pin_config.slewRate (*C var*), 653
drivers_MCX_MCXW_MCXW727C._port_pull (*C enum*), 648
drivers_MCX_MCXW_MCXW727C._port_pull_value (*C enum*), 648
drivers_MCX_MCXW_MCXW727C._port_pull_value.kPORT_HighPullResistor (*C enumerator*), 649
drivers_MCX_MCXW_MCXW727C._port_pull_value.kPORT_LowPullResistor (*C enumerator*), 648
drivers_MCX_MCXW_MCXW727C._port_pull.kPORT_PullDisable (*C enumerator*), 648
drivers_MCX_MCXW_MCXW727C._port_pull.kPORT_PullDown (*C enumerator*), 648
drivers_MCX_MCXW_MCXW727C._port_pull.kPORT_PullUp (*C enumerator*), 648

drivers_MCX_MCXW_MCXW727C._port_slew_rate (*C enum*), 649
drivers_MCX_MCXW_MCXW727C._port_slew_rate.kPORT_FastSlewRate (*C enumerator*), 649
drivers_MCX_MCXW_MCXW727C._port_slew_rate.kPORT_SlowSlewRate (*C enumerator*), 649
drivers_MCX_MCXW_MCXW727C._port_version_info (*C struct*), 653
drivers_MCX_MCXW_MCXW727C._port_version_info.feature (*C var*), 654
drivers_MCX_MCXW_MCXW727C._port_version_info.major (*C var*), 654
drivers_MCX_MCXW_MCXW727C._port_version_info.minor (*C var*), 654
drivers_MCX_MCXW_MCXW727C._port_voltage_range (*C enum*), 652
drivers_MCX_MCXW_MCXW727C._port_voltage_range.kPORT_VoltageRange1Dot71V_3Dot6V (*C enumerator*), 652
drivers_MCX_MCXW_MCXW727C._port_voltage_range.kPORT_VoltageRange2Dot70V_3Dot6V (*C enumerator*), 652
drivers_MCX_MCXW_MCXW727C._rtc_config (*C struct*), 662
drivers_MCX_MCXW_MCXW727C._rtc_config.clockOutput (*C var*), 662
drivers_MCX_MCXW_MCXW727C._rtc_config.compensationInterval (*C var*), 662
drivers_MCX_MCXW_MCXW727C._rtc_config.compensationTime (*C var*), 662
drivers_MCX_MCXW_MCXW727C._rtc_config.supervisorAccess (*C var*), 662
drivers_MCX_MCXW_MCXW727C._rtc_config.updateMode (*C var*), 662
drivers_MCX_MCXW_MCXW727C._rtc_config.wakeupSelect (*C var*), 662
drivers_MCX_MCXW_MCXW727C._rtc_datetime (*C struct*), 661
drivers_MCX_MCXW_MCXW727C._rtc_datetime.day (*C var*), 661
drivers_MCX_MCXW_MCXW727C._rtc_datetime.hour (*C var*), 661
drivers_MCX_MCXW_MCXW727C._rtc_datetime.minute (*C var*), 661
drivers_MCX_MCXW_MCXW727C._rtc_datetime.month (*C var*), 661
drivers_MCX_MCXW_MCXW727C._rtc_datetime.second (*C var*), 661
drivers_MCX_MCXW_MCXW727C._rtc_datetime.year (*C var*), 661
drivers_MCX_MCXW_MCXW727C._rtc_interrupt_enable (*C enum*), 657
drivers_MCX_MCXW_MCXW727C._rtc_interrupt_enable.kRTC_AlarmInterruptEnable (*C enumerator*), 657
drivers_MCX_MCXW_MCXW727C._rtc_interrupt_enable.kRTC_FlashSecurityInterruptEnable (*C enumerator*), 658
drivers_MCX_MCXW_MCXW727C._rtc_interrupt_enable.kRTC_LossOfClockInterruptEnable (*C enumerator*), 658
drivers_MCX_MCXW_MCXW727C._rtc_interrupt_enable.kRTC_MonotonicOverflowInterruptEnable (*C enumerator*), 657
drivers_MCX_MCXW_MCXW727C._rtc_interrupt_enable.kRTC_SecondsInterruptEnable (*C enumerator*), 657
drivers_MCX_MCXW_MCXW727C._rtc_interrupt_enable.kRTC_SecurityModuleInterruptEnable (*C enumerator*), 658
drivers_MCX_MCXW_MCXW727C._rtc_interrupt_enable.kRTC_TamperPinInterruptEnable (*C enumerator*), 658
drivers_MCX_MCXW_MCXW727C._rtc_interrupt_enable.kRTC_TestModeInterruptEnable (*C enumerator*), 657
drivers_MCX_MCXW_MCXW727C._rtc_interrupt_enable.kRTC_TimeInvalidInterruptEnable (*C enumerator*), 657
drivers_MCX_MCXW_MCXW727C._rtc_interrupt_enable.kRTC_TimeOverflowInterruptEnable (*C enumerator*), 657
drivers_MCX_MCXW_MCXW727C._rtc_osc_cap_load (*C enum*), 658
drivers_MCX_MCXW_MCXW727C._rtc_osc_cap_load.kRTC_Capacitor_2p (*C enumerator*), 658
drivers_MCX_MCXW_MCXW727C._rtc_osc_cap_load.kRTC_Capacitor_4p (*C enumerator*), 658
drivers_MCX_MCXW_MCXW727C._rtc_osc_cap_load.kRTC_Capacitor_8p (*C enumerator*), 659
drivers_MCX_MCXW_MCXW727C._rtc_osc_cap_load.kRTC_Capacitor_16p (*C enumerator*), 659
drivers_MCX_MCXW_MCXW727C._rtc_pin_config (*C struct*), 661
drivers_MCX_MCXW_MCXW727C._rtc_pin_config.filterEnable (*C var*), 661
drivers_MCX_MCXW_MCXW727C._rtc_pin_config.inputLogic (*C var*), 661
drivers_MCX_MCXW_MCXW727C._rtc_pin_config.pinActiveLow (*C var*), 661
drivers_MCX_MCXW_MCXW727C._rtc_pin_config.pullEnable (*C var*), 662

drivers_MCX_MCXW_MCXW727C._rtc_pin_config.pullSelectNegate (*C var*), 661
drivers_MCX_MCXW_MCXW727C._rtc_status_flags (*C enum*), 658
drivers_MCX_MCXW_MCXW727C._rtc_status_flags.kRTC_AlarmFlag (*C enumerator*), 658
drivers_MCX_MCXW_MCXW727C._rtc_status_flags.kRTC_FlashSecurityFlag (*C enumerator*), 658
drivers_MCX_MCXW_MCXW727C._rtc_status_flags.kRTC_LossOfClockTamperFlag (*C enumerator*), 658
drivers_MCX_MCXW_MCXW727C._rtc_status_flags.kRTC_MonotonicOverflowFlag (*C enumerator*), 658
drivers_MCX_MCXW_MCXW727C._rtc_status_flags.kRTC_SecurityTamperFlag (*C enumerator*), 658
drivers_MCX_MCXW_MCXW727C._rtc_status_flags.kRTC_TamperInterruptDetectFlag (*C enumerator*), 658
drivers_MCX_MCXW_MCXW727C._rtc_status_flags.kRTC_TamperPinFlag (*C enumerator*), 658
drivers_MCX_MCXW_MCXW727C._rtc_status_flags.kRTC_TestModeFlag (*C enumerator*), 658
drivers_MCX_MCXW_MCXW727C._rtc_status_flags.kRTC_TimeInvalidFlag (*C enumerator*), 658
drivers_MCX_MCXW_MCXW727C._rtc_status_flags.kRTC_TimeOverflowFlag (*C enumerator*), 658
drivers_MCX_MCXW_MCXW727C._rtc_timer_seconds_interrupt_frequency (*C enum*), 659
drivers_MCX_MCXW_MCXW727C._rtc_timer_seconds_interrupt_frequency.kRTC_TimerSecondsFrequency1Hz (*C enumerator*), 659
drivers_MCX_MCXW_MCXW727C._rtc_timer_seconds_interrupt_frequency.kRTC_TimerSecondsFrequency2Hz (*C enumerator*), 659
drivers_MCX_MCXW_MCXW727C._rtc_timer_seconds_interrupt_frequency.kRTC_TimerSecondsFrequency4Hz (*C enumerator*), 659
drivers_MCX_MCXW_MCXW727C._rtc_timer_seconds_interrupt_frequency.kRTC_TimerSecondsFrequency8Hz (*C enumerator*), 659
drivers_MCX_MCXW_MCXW727C._rtc_timer_seconds_interrupt_frequency.kRTC_TimerSecondsFrequency16Hz (*C enumerator*), 659
drivers_MCX_MCXW_MCXW727C._rtc_timer_seconds_interrupt_frequency.kRTC_TimerSecondsFrequency32Hz (*C enumerator*), 659
drivers_MCX_MCXW_MCXW727C._rtc_timer_seconds_interrupt_frequency.kRTC_TimerSecondsFrequency64Hz (*C enumerator*), 659
drivers_MCX_MCXW_MCXW727C._rtc_timer_seconds_interrupt_frequency.kRTC_TimerSecondsFrequency128Hz (*C enumerator*), 659
drivers_MCX_MCXW_MCXW727C._scg_firc_config_t (*C struct*), 197
drivers_MCX_MCXW_MCXW727C._scg_firc_config_t.enableMode (*C var*), 197
drivers_MCX_MCXW_MCXW727C._scg_firc_config_t.range (*C var*), 197
drivers_MCX_MCXW_MCXW727C._scg_firc_config_t.trimConfig (*C var*), 198
drivers_MCX_MCXW_MCXW727C._scg_firc_range (*C enum*), 190
drivers_MCX_MCXW_MCXW727C._scg_firc_range.kSCG_FircRange48M (*C enumerator*), 190
drivers_MCX_MCXW_MCXW727C._scg_firc_range.kSCG_FircRange64M (*C enumerator*), 190
drivers_MCX_MCXW_MCXW727C._scg_firc_range.kSCG_FircRange96M (*C enumerator*), 190
drivers_MCX_MCXW_MCXW727C._scg_firc_range.kSCG_FircRange192M (*C enumerator*), 190
drivers_MCX_MCXW_MCXW727C._scg_firc_trim_config (*C struct*), 197
drivers_MCX_MCXW_MCXW727C._scg_firc_trim_config.trimCoar (*C var*), 197
drivers_MCX_MCXW_MCXW727C._scg_firc_trim_config.trimDiv (*C var*), 197
drivers_MCX_MCXW_MCXW727C._scg_firc_trim_config.trimFine (*C var*), 197
drivers_MCX_MCXW_MCXW727C._scg_firc_trim_config.trimMode (*C var*), 197
drivers_MCX_MCXW_MCXW727C._scg_firc_trim_config.trimSrc (*C var*), 197
drivers_MCX_MCXW_MCXW727C._scg_firc_trim_mode (*C enum*), 189
drivers_MCX_MCXW_MCXW727C._scg_firc_trim_mode.kSCG_FircTrimNonUpdate (*C enumerator*), 189
drivers_MCX_MCXW_MCXW727C._scg_firc_trim_mode.kSCG_FircTrimUpdate (*C enumerator*), 189
drivers_MCX_MCXW_MCXW727C._scg_firc_trim_src (*C enum*), 189
drivers_MCX_MCXW_MCXW727C._scg_firc_trim_src.kSCG_FircTrimSrcRtcOsc (*C enumerator*), 189

drivers_MCX_MCXW_MCXW727C._scg_firc_trim_src.kSCG_FircTrimSrcSysOsc (*C enumerator*), 189
drivers_MCX_MCXW_MCXW727C._scg_rosc_config (*C struct*), 197
drivers_MCX_MCXW_MCXW727C._scg_rosc_config.monitorMode (*C var*), 197
drivers_MCX_MCXW_MCXW727C._scg_rosc_monitor_mode (*C enum*), 189
drivers_MCX_MCXW_MCXW727C._scg_rosc_monitor_mode.kSCG_RoscMonitorDisable (*C enumerator*), 189
drivers_MCX_MCXW_MCXW727C._scg_rosc_monitor_mode.kSCG_RoscMonitorInt (*C enumerator*), 189
drivers_MCX_MCXW_MCXW727C._scg_rosc_monitor_mode.kSCG_RoscMonitorReset (*C enumerator*), 189
drivers_MCX_MCXW_MCXW727C._scg_sosc_config (*C struct*), 196
drivers_MCX_MCXW_MCXW727C._scg_sosc_config.enableMode (*C var*), 197
drivers_MCX_MCXW_MCXW727C._scg_sosc_config.freq (*C var*), 197
drivers_MCX_MCXW_MCXW727C._scg_sosc_config.monitorMode (*C var*), 197
drivers_MCX_MCXW_MCXW727C._scg_sosc_monitor_mode (*C enum*), 188
drivers_MCX_MCXW_MCXW727C._scg_sosc_monitor_mode.kSCG_SysOscMonitorDisable (*C enumerator*), 188
drivers_MCX_MCXW_MCXW727C._scg_sosc_monitor_mode.kSCG_SysOscMonitorInt (*C enumerator*), 188
drivers_MCX_MCXW_MCXW727C._scg_sosc_monitor_mode.kSCG_SysOscMonitorReset (*C enumerator*), 188
drivers_MCX_MCXW_MCXW727C._scg_status (*C enum*), 186
drivers_MCX_MCXW_MCXW727C._scg_status.kStatus_SCG_Busy (*C enumerator*), 186
drivers_MCX_MCXW_MCXW727C._scg_status.kStatus_SCG_InvalidSrc (*C enumerator*), 186
drivers_MCX_MCXW_MCXW727C._scg_sys_clk (*C enum*), 186
drivers_MCX_MCXW_MCXW727C._scg_sys_clk_config (*C struct*), 196
drivers_MCX_MCXW_MCXW727C._scg_sys_clk_config.__pad0__ (*C var*), 196
drivers_MCX_MCXW_MCXW727C._scg_sys_clk_config.__pad1__ (*C var*), 196
drivers_MCX_MCXW_MCXW727C._scg_sys_clk_config.__pad2__ (*C var*), 196
drivers_MCX_MCXW_MCXW727C._scg_sys_clk_config.divBus (*C var*), 196
drivers_MCX_MCXW_MCXW727C._scg_sys_clk_config.divCore (*C var*), 196
drivers_MCX_MCXW_MCXW727C._scg_sys_clk_config.divSlow (*C var*), 196
drivers_MCX_MCXW_MCXW727C._scg_sys_clk_config.src (*C var*), 196
drivers_MCX_MCXW_MCXW727C._scg_sys_clk_div (*C enum*), 187
drivers_MCX_MCXW_MCXW727C._scg_sys_clk_div.kSCG_SysClkDivBy1 (*C enumerator*), 187
drivers_MCX_MCXW_MCXW727C._scg_sys_clk_div.kSCG_SysClkDivBy2 (*C enumerator*), 187
drivers_MCX_MCXW_MCXW727C._scg_sys_clk_div.kSCG_SysClkDivBy3 (*C enumerator*), 187
drivers_MCX_MCXW_MCXW727C._scg_sys_clk_div.kSCG_SysClkDivBy4 (*C enumerator*), 187
drivers_MCX_MCXW_MCXW727C._scg_sys_clk_div.kSCG_SysClkDivBy5 (*C enumerator*), 187
drivers_MCX_MCXW_MCXW727C._scg_sys_clk_div.kSCG_SysClkDivBy6 (*C enumerator*), 187
drivers_MCX_MCXW_MCXW727C._scg_sys_clk_div.kSCG_SysClkDivBy7 (*C enumerator*), 187
drivers_MCX_MCXW_MCXW727C._scg_sys_clk_div.kSCG_SysClkDivBy8 (*C enumerator*), 187
drivers_MCX_MCXW_MCXW727C._scg_sys_clk_div.kSCG_SysClkDivBy9 (*C enumerator*), 187
drivers_MCX_MCXW_MCXW727C._scg_sys_clk_div.kSCG_SysClkDivBy10 (*C enumerator*), 187
drivers_MCX_MCXW_MCXW727C._scg_sys_clk_div.kSCG_SysClkDivBy11 (*C enumerator*), 187
drivers_MCX_MCXW_MCXW727C._scg_sys_clk_div.kSCG_SysClkDivBy12 (*C enumerator*), 187
drivers_MCX_MCXW_MCXW727C._scg_sys_clk_div.kSCG_SysClkDivBy13 (*C enumerator*), 187
drivers_MCX_MCXW_MCXW727C._scg_sys_clk_div.kSCG_SysClkDivBy14 (*C enumerator*), 187
drivers_MCX_MCXW_MCXW727C._scg_sys_clk_div.kSCG_SysClkDivBy15 (*C enumerator*), 188
drivers_MCX_MCXW_MCXW727C._scg_sys_clk_div.kSCG_SysClkDivBy16 (*C enumerator*), 188
drivers_MCX_MCXW_MCXW727C._scg_sys_clk_src (*C enum*), 186
drivers_MCX_MCXW_MCXW727C._scg_sys_clk_src.kSCG_SysClkSrcFirc (*C enumerator*), 186
drivers_MCX_MCXW_MCXW727C._scg_sys_clk_src.kSCG_SysClkSrcRosc (*C enumerator*), 187
drivers_MCX_MCXW_MCXW727C._scg_sys_clk_src.kSCG_SysClkSrcSirc (*C enumerator*), 186
drivers_MCX_MCXW_MCXW727C._scg_sys_clk_src.kSCG_SysClkSrcSysOsc (*C enumerator*), 186
drivers_MCX_MCXW_MCXW727C._scg_sys_clk.kSCG_SysClkBus (*C enumerator*), 186

drivers_MCX_MCXW_MCXW727C._scg_sys_clk.kSCG_SysClkCore (*C enumerator*), 186
drivers_MCX_MCXW_MCXW727C._scg_sys_clk.kSCG_SysClkPlatform (*C enumerator*), 186
drivers_MCX_MCXW_MCXW727C._scg_sys_clk.kSCG_SysClkSlow (*C enumerator*), 186
drivers_MCX_MCXW_MCXW727C._SDK_AtomicLocalCompareAndSet (*C function*), 464
drivers_MCX_MCXW_MCXW727C._SDK_AtomicTestAndSet (*C function*), 464
drivers_MCX_MCXW_MCXW727C._sema42_gate_status (*C enum*), 663
drivers_MCX_MCXW_MCXW727C._sema42_gate_status.kSEMA42_LockedByProc0 (*C enumerator*), 663
drivers_MCX_MCXW_MCXW727C._sema42_gate_status.kSEMA42_LockedByProc1 (*C enumerator*), 663
drivers_MCX_MCXW_MCXW727C._sema42_gate_status.kSEMA42_LockedByProc2 (*C enumerator*), 663
drivers_MCX_MCXW_MCXW727C._sema42_gate_status.kSEMA42_LockedByProc3 (*C enumerator*), 663
drivers_MCX_MCXW_MCXW727C._sema42_gate_status.kSEMA42_LockedByProc4 (*C enumerator*), 663
drivers_MCX_MCXW_MCXW727C._sema42_gate_status.kSEMA42_LockedByProc5 (*C enumerator*), 663
drivers_MCX_MCXW_MCXW727C._sema42_gate_status.kSEMA42_LockedByProc6 (*C enumerator*), 663
drivers_MCX_MCXW_MCXW727C._sema42_gate_status.kSEMA42_LockedByProc7 (*C enumerator*), 663
drivers_MCX_MCXW_MCXW727C._sema42_gate_status.kSEMA42_LockedByProc8 (*C enumerator*), 663
drivers_MCX_MCXW_MCXW727C._sema42_gate_status.kSEMA42_LockedByProc9 (*C enumerator*), 663
drivers_MCX_MCXW_MCXW727C._sema42_gate_status.kSEMA42_LockedByProc10 (*C enumerator*), 663
drivers_MCX_MCXW_MCXW727C._sema42_gate_status.kSEMA42_LockedByProc11 (*C enumerator*), 663
drivers_MCX_MCXW_MCXW727C._sema42_gate_status.kSEMA42_LockedByProc12 (*C enumerator*), 663
drivers_MCX_MCXW_MCXW727C._sema42_gate_status.kSEMA42_LockedByProc13 (*C enumerator*), 664
drivers_MCX_MCXW_MCXW727C._sema42_gate_status.kSEMA42_LockedByProc14 (*C enumerator*), 664
drivers_MCX_MCXW_MCXW727C._sema42_gate_status.kSEMA42_Unlocked (*C enumerator*), 663
drivers_MCX_MCXW_MCXW727C._sfa_cut_select (*C enum*), 673
drivers_MCX_MCXW_MCXW727C._sfa_cut_select.kSFA_CUTSelect0 (*C enumerator*), 673
drivers_MCX_MCXW_MCXW727C._sfa_cut_select.kSFA_CUTSelect1 (*C enumerator*), 673
drivers_MCX_MCXW_MCXW727C._sfa_cut_select.kSFA_CUTSelect2 (*C enumerator*), 673
drivers_MCX_MCXW_MCXW727C._sfa_cut_select.kSFA_CUTSelect3 (*C enumerator*), 673
drivers_MCX_MCXW_MCXW727C._sfa_cut_select.kSFA_CUTSelect4 (*C enumerator*), 673
drivers_MCX_MCXW_MCXW727C._sfa_cut_select.kSFA_CUTSelect5 (*C enumerator*), 673
drivers_MCX_MCXW_MCXW727C._sfa_cut_select.kSFA_CUTSelect6 (*C enumerator*), 673
drivers_MCX_MCXW_MCXW727C._sfa_cut_select.kSFA_CUTSelect7 (*C enumerator*), 673
drivers_MCX_MCXW_MCXW727C._sfa_cut_select.kSFA_CUTSelect8 (*C enumerator*), 673
drivers_MCX_MCXW_MCXW727C._sfa_cut_select.kSFA_CUTSelect9 (*C enumerator*), 673
drivers_MCX_MCXW_MCXW727C._sfa_cut_select.kSFA_CUTSelect10 (*C enumerator*), 673
drivers_MCX_MCXW_MCXW727C._sfa_cut_select.kSFA_CUTSelect11 (*C enumerator*), 673
drivers_MCX_MCXW_MCXW727C._sfa_cut_select.kSFA_CUTSelect12 (*C enumerator*), 673
drivers_MCX_MCXW_MCXW727C._sfa_cut_select.kSFA_CUTSelect13 (*C enumerator*), 673
drivers_MCX_MCXW_MCXW727C._sfa_cut_select.kSFA_CUTSelect14 (*C enumerator*), 673
drivers_MCX_MCXW_MCXW727C._sfa_cut_select.kSFA_CUTSelect15 (*C enumerator*), 673
drivers_MCX_MCXW_MCXW727C._sfa_init_config (*C struct*), 675
drivers_MCX_MCXW_MCXW727C._sfa_init_config.cutSelect (*C var*), 675
drivers_MCX_MCXW_MCXW727C._sfa_init_config.cutTarget (*C var*), 676

drivers_MCX_MCXW_MCXW727C._sfa_init_config.enableCUTPin (*C var*), 676
drivers_MCX_MCXW_MCXW727C._sfa_init_config.enableTrigMeasurement (*C var*), 676
drivers_MCX_MCXW_MCXW727C._sfa_init_config.endPolarity (*C var*), 676
drivers_MCX_MCXW_MCXW727C._sfa_init_config.mode (*C var*), 675
drivers_MCX_MCXW_MCXW727C._sfa_init_config.prediv (*C var*), 676
drivers_MCX_MCXW_MCXW727C._sfa_init_config.refSelect (*C var*), 675
drivers_MCX_MCXW_MCXW727C._sfa_init_config.refTarget (*C var*), 676
drivers_MCX_MCXW_MCXW727C._sfa_init_config.startPolarity (*C var*), 676
drivers_MCX_MCXW_MCXW727C._sfa_init_config.trigEnd (*C var*), 676
drivers_MCX_MCXW_MCXW727C._sfa_init_config.trigStart (*C var*), 676
drivers_MCX_MCXW_MCXW727C._sfa_interrupts_enable (*C enum*), 672
drivers_MCX_MCXW_MCXW727C._sfa_interrupts_enable.kSFA_FreqGreaterThanMaxInterruptEnable (*C enumerator*), 672
drivers_MCX_MCXW_MCXW727C._sfa_interrupts_enable.kSFA_FreqLessThanMinInterruptEnable (*C enumerator*), 672
drivers_MCX_MCXW_MCXW727C._sfa_interrupts_enable.kSFA_InterruptEnable (*C enumerator*), 672
drivers_MCX_MCXW_MCXW727C._sfa_measurement_mode (*C enum*), 672
drivers_MCX_MCXW_MCXW727C._sfa_measurement_mode.kSFA_CUTPeriodMeasurement (*C enumerator*), 672
drivers_MCX_MCXW_MCXW727C._sfa_measurement_mode.kSFA_FrequencyMeasurement0 (*C enumerator*), 672
drivers_MCX_MCXW_MCXW727C._sfa_measurement_mode.kSFA_FrequencyMeasurement1 (*C enumerator*), 672
drivers_MCX_MCXW_MCXW727C._sfa_measurement_mode.kSFA_TriggerBasedMeasurement (*C enumerator*), 672
drivers_MCX_MCXW_MCXW727C._sfa_ref_select (*C enum*), 673
drivers_MCX_MCXW_MCXW727C._sfa_ref_select.kSFA_REFSelect0 (*C enumerator*), 673
drivers_MCX_MCXW_MCXW727C._sfa_ref_select.kSFA_REFSelect1 (*C enumerator*), 673
drivers_MCX_MCXW_MCXW727C._sfa_ref_select.kSFA_REFSelect2 (*C enumerator*), 674
drivers_MCX_MCXW_MCXW727C._sfa_status_flags (*C enum*), 671
drivers_MCX_MCXW_MCXW727C._sfa_status_flags.kSFA_AllStatusFlags (*C enumerator*), 672
drivers_MCX_MCXW_MCXW727C._sfa_status_flags.kSFA_CutStoppedFlag (*C enumerator*), 671
drivers_MCX_MCXW_MCXW727C._sfa_status_flags.kSFA_FreqGreaterThanMaxInterruptFlag (*C enumerator*), 672
drivers_MCX_MCXW_MCXW727C._sfa_status_flags.kSFA_FreqLessThanMinInterruptFlag (*C enumerator*), 672
drivers_MCX_MCXW_MCXW727C._sfa_status_flags.kSFA_InterruptRequestFlag (*C enumerator*), 672
drivers_MCX_MCXW_MCXW727C._sfa_status_flags.kSFA_MeasurementStartedFlag (*C enumerator*), 672
drivers_MCX_MCXW_MCXW727C._sfa_status_flags.kSFA_ReferenceCounterTimeOutFlag (*C enumerator*), 672
drivers_MCX_MCXW_MCXW727C._sfa_status_flags.kSFA_RefStoppedFlag (*C enumerator*), 671
drivers_MCX_MCXW_MCXW727C._sfa_trigger_end_polarity (*C enum*), 674
drivers_MCX_MCXW_MCXW727C._sfa_trigger_end_polarity.kSFA_TriggerEndPolarityFallEdge (*C enumerator*), 674
drivers_MCX_MCXW_MCXW727C._sfa_trigger_end_polarity.kSFA_TriggerEndPolarityRiseEdge (*C enumerator*), 674
drivers_MCX_MCXW_MCXW727C._sfa_trigger_end_select (*C enum*), 674
drivers_MCX_MCXW_MCXW727C._sfa_trigger_end_select.kSFA_TriggerEndSelect0 (*C enumerator*), 674
drivers_MCX_MCXW_MCXW727C._sfa_trigger_end_select.kSFA_TriggerEndSelect1 (*C enumerator*), 674
drivers_MCX_MCXW_MCXW727C._sfa_trigger_start_polarity (*C enum*), 674
drivers_MCX_MCXW_MCXW727C._sfa_trigger_start_polarity.kSFA_TriggerStartPolarityFallEdge (*C enumerator*), 674
drivers_MCX_MCXW_MCXW727C._sfa_trigger_start_polarity.kSFA_TriggerStartPolarityRiseEdge

(*C enumerator*), 674
drivers_MCX_MCXW_MCXW727C._sfa_trigger_start_select (*C enum*), 674
drivers_MCX_MCXW_MCXW727C._sfa_trigger_start_select.kSFA_TriggerStartSelect0 (*C enumerator*), 674
drivers_MCX_MCXW_MCXW727C._sfa_trigger_start_select.kSFA_TriggerStartSelect1 (*C enumerator*), 674
drivers_MCX_MCXW_MCXW727C._smscm_debug (*C enum*), 676
drivers_MCX_MCXW_MCXW727C._smscm_debug.kSMSCM_AltDebug (*C enumerator*), 677
drivers_MCX_MCXW_MCXW727C._smscm_debug.kSMSCM_AltInvasiveDebug (*C enumerator*), 677
drivers_MCX_MCXW_MCXW727C._smscm_debug.kSMSCM_InvasiveDebug (*C enumerator*), 676
drivers_MCX_MCXW_MCXW727C._smscm_debug.kSMSCM_NonInvasiveDebug (*C enumerator*), 676
drivers_MCX_MCXW_MCXW727C._smscm_debug.kSMSCM_SecureInvasiveDebug (*C enumerator*), 676
drivers_MCX_MCXW_MCXW727C._smscm_debug.kSMSCM_SecureNonInvasiveDebug (*C enumerator*), 676
drivers_MCX_MCXW_MCXW727C._smscm_ecc_ctrl (*C enum*), 677
drivers_MCX_MCXW_MCXW727C._smscm_ecc_ctrl.kSMSCM_EccDisable (*C enumerator*), 677
drivers_MCX_MCXW_MCXW727C._smscm_ecc_ctrl.kSMSCM_EccEnableOnRead (*C enumerator*), 677
drivers_MCX_MCXW_MCXW727C._smscm_ecc_ctrl.kSMSCM_EccEnableOnWrite (*C enumerator*), 677
drivers_MCX_MCXW_MCXW727C._smscm_ecc_ctrl.kSMSCM_EccEnableOnWriteAndRead (*C enumerator*), 677
drivers_MCX_MCXW_MCXW727C._smscm_mem (*C enum*), 677
drivers_MCX_MCXW_MCXW727C._smscm_mem.kSMSCM_Mem0 (*C enumerator*), 677
drivers_MCX_MCXW_MCXW727C._smscm_mem.kSMSCM_Mem2 (*C enumerator*), 677
drivers_MCX_MCXW_MCXW727C._smscm_mem.kSMSCM_Mem3 (*C enumerator*), 677
drivers_MCX_MCXW_MCXW727C._smscm_mem.kSMSCM_Mem5 (*C enumerator*), 677
drivers_MCX_MCXW_MCXW727C._spc_active_mode_core_ldo_option (*C struct*), 718
drivers_MCX_MCXW_MCXW727C._spc_active_mode_core_ldo_option.CoreLDODriveStrength (*C var*), 718
drivers_MCX_MCXW_MCXW727C._spc_active_mode_core_ldo_option.CoreLDOVoltage (*C var*), 718
drivers_MCX_MCXW_MCXW727C._spc_active_mode_dcdc_option (*C struct*), 719
drivers_MCX_MCXW_MCXW727C._spc_active_mode_dcdc_option.DCDCDriveStrength (*C var*), 719
drivers_MCX_MCXW_MCXW727C._spc_active_mode_dcdc_option.DCDCVoltage (*C var*), 719
drivers_MCX_MCXW_MCXW727C._spc_active_mode_regulators_config (*C struct*), 721
drivers_MCX_MCXW_MCXW727C._spc_active_mode_sys_ldo_option (*C struct*), 719
drivers_MCX_MCXW_MCXW727C._spc_active_mode_sys_ldo_option.SysLDODriveStrength (*C var*), 719
drivers_MCX_MCXW_MCXW727C._spc_active_mode_sys_ldo_option.SysLDOVoltage (*C var*), 719
drivers_MCX_MCXW_MCXW727C._spc_bandgap_mode (*C enum*), 684
drivers_MCX_MCXW_MCXW727C._spc_bandgap_mode.kSPC_BandgapDisabled (*C enumerator*), 684
drivers_MCX_MCXW_MCXW727C._spc_bandgap_mode.kSPC_BandgapEnabledBufferDisabled (*C enumerator*), 684
drivers_MCX_MCXW_MCXW727C._spc_bandgap_mode.kSPC_BandgapEnabledBufferEnabled (*C enumerator*), 685
drivers_MCX_MCXW_MCXW727C._spc_bandgap_mode.kSPC_BandgapReserved (*C enumerator*), 685
drivers_MCX_MCXW_MCXW727C._spc_core_ldo_drive_strength (*C enum*), 686
drivers_MCX_MCXW_MCXW727C._spc_core_ldo_drive_strength.kSPC_CoreLDO_LowDriveStrength (*C enumerator*), 686
drivers_MCX_MCXW_MCXW727C._spc_core_ldo_drive_strength.kSPC_CoreLDO_NormalDriveStrength

(*C enumerator*), 686
drivers_MCX_MCXW_MCXW727C._spc_core_ldo_voltage_level (*C enum*), 686
drivers_MCX_MCXW_MCXW727C._spc_core_ldo_voltage_level.kSPC_CoreLDO_MidDriveVoltage (*C enumerator*), 686
drivers_MCX_MCXW_MCXW727C._spc_core_ldo_voltage_level.kSPC_CoreLDO_NormalVoltage (*C enumerator*), 686
drivers_MCX_MCXW_MCXW727C._spc_core_ldo_voltage_level.kSPC_CoreLDO_SafeModeVoltage (*C enumerator*), 686
drivers_MCX_MCXW_MCXW727C._spc_core_ldo_voltage_level.kSPC_CoreLDO_UnderDriveVoltage (*C enumerator*), 686
drivers_MCX_MCXW_MCXW727C._spc_core_voltage_detect_config (*C struct*), 720
drivers_MCX_MCXW_MCXW727C._spc_core_voltage_detect_config.option (*C var*), 721
drivers_MCX_MCXW_MCXW727C._spc_dcdc_burst_config (*C struct*), 720
drivers_MCX_MCXW_MCXW727C._spc_dcdc_burst_config.externalBurstRequest (*C var*), 720
drivers_MCX_MCXW_MCXW727C._spc_dcdc_burst_config.freq (*C var*), 720
drivers_MCX_MCXW_MCXW727C._spc_dcdc_burst_config.sofwareBurstRequest (*C var*), 720
drivers_MCX_MCXW_MCXW727C._spc_dcdc_burst_config.stabilizeBurstFreq (*C var*), 720
drivers_MCX_MCXW_MCXW727C._spc_dcdc_drive_strength (*C enum*), 685
drivers_MCX_MCXW_MCXW727C._spc_dcdc_drive_strength.kSPC_DCDC_LowDriveStrength (*C enumerator*), 685
drivers_MCX_MCXW_MCXW727C._spc_dcdc_drive_strength.kSPC_DCDC_NormalDriveStrength (*C enumerator*), 685
drivers_MCX_MCXW_MCXW727C._spc_dcdc_drive_strength.kSPC_DCDC_PulseRefreshMode (*C enumerator*), 685
drivers_MCX_MCXW_MCXW727C._spc_dcdc_drive_strength.kSPC_DCDC_Reserved (*C enumerator*), 685
drivers_MCX_MCXW_MCXW727C._spc_dcdc_ramp_config (*C struct*), 722
drivers_MCX_MCXW_MCXW727C._spc_dcdc_ramp_config.enableRampControl (*C var*), 722
drivers_MCX_MCXW_MCXW727C._spc_dcdc_ramp_config.rampControl (*C var*), 722
drivers_MCX_MCXW_MCXW727C._spc_dcdc_ramp_config.rampControlLowPower (*C var*), 722
drivers_MCX_MCXW_MCXW727C._spc_dcdc_voltage_level (*C enum*), 685
drivers_MCX_MCXW_MCXW727C._spc_dcdc_voltage_level.kSPC_DCDC_LowUnderVoltage (*C enumerator*), 685
drivers_MCX_MCXW_MCXW727C._spc_dcdc_voltage_level.kSPC_DCDC_MidVoltage (*C enumerator*), 685
drivers_MCX_MCXW_MCXW727C._spc_dcdc_voltage_level.kSPC_DCDC_NormalVoltage (*C enumerator*), 685
drivers_MCX_MCXW_MCXW727C._spc_dcdc_voltage_level.kSPC_DCDC_SafeModeVoltage (*C enumerator*), 685
drivers_MCX_MCXW_MCXW727C._spc_hp_request_override_option (*C enum*), 687
drivers_MCX_MCXW_MCXW727C._spc_hp_request_override_option.kSPC_HpReqestOverride1 (*C enumerator*), 687
drivers_MCX_MCXW_MCXW727C._spc_hp_request_override_option.kSPC_HpRequestOverride0 (*C enumerator*), 687
drivers_MCX_MCXW_MCXW727C._spc_hp_request_override_option.kSPC_HpRequestOverrideDisable (*C enumerator*), 687
drivers_MCX_MCXW_MCXW727C._spc_intergrated_power_switch_config (*C struct*), 718
drivers_MCX_MCXW_MCXW727C._spc_intergrated_power_switch_config.sleep (*C var*), 718
drivers_MCX_MCXW_MCXW727C._spc_intergrated_power_switch_config.wakeup (*C var*), 718
drivers_MCX_MCXW_MCXW727C._spc_io_voltage_detect_config (*C struct*), 721
drivers_MCX_MCXW_MCXW727C._spc_io_voltage_detect_config.level (*C var*), 721
drivers_MCX_MCXW_MCXW727C._spc_io_voltage_detect_config.option (*C var*), 721
drivers_MCX_MCXW_MCXW727C._spc_low_voltage_level_select (*C enum*), 686
drivers_MCX_MCXW_MCXW727C._spc_low_voltage_level_select.kSPC_LowVoltageNormalLevel (*C enumerator*), 686
drivers_MCX_MCXW_MCXW727C._spc_low_voltage_level_select.kSPC_LowVoltageSafeLevel (*C enumerator*), 686
drivers_MCX_MCXW_MCXW727C._spc_lowpower_mode_core_ldo_option (*C struct*), 719

drivers_MCX_MCXW_MCXW727C._spc_lowpower_mode_core_ldo_option.CoreLDODriveStrength (*C var*), 719
drivers_MCX_MCXW_MCXW727C._spc_lowpower_mode_core_ldo_option.CoreLDOVoltage (*C var*), 719
drivers_MCX_MCXW_MCXW727C._spc_lowpower_mode_dcdc_option (*C struct*), 719
drivers_MCX_MCXW_MCXW727C._spc_lowpower_mode_dcdc_option.DCDCDriveStrength (*C var*), 720
drivers_MCX_MCXW_MCXW727C._spc_lowpower_mode_dcdc_option.DCDCVoltage (*C var*), 720
drivers_MCX_MCXW_MCXW727C._spc_lowpower_mode_regulators_config (*C struct*), 721
drivers_MCX_MCXW_MCXW727C._spc_lowpower_mode_sys_ldo_option (*C struct*), 719
drivers_MCX_MCXW_MCXW727C._spc_lowpower_mode_sys_ldo_option.SysLDODriveStrength (*C var*), 719
drivers_MCX_MCXW_MCXW727C._spc_lowpower_request_config (*C struct*), 718
drivers_MCX_MCXW_MCXW727C._spc_lowpower_request_config.enable (*C var*), 718
drivers_MCX_MCXW_MCXW727C._spc_lowpower_request_config.override (*C var*), 718
drivers_MCX_MCXW_MCXW727C._spc_lowpower_request_config.polarity (*C var*), 718
drivers_MCX_MCXW_MCXW727C._spc_lowPower_request_output_override (*C enum*), 684
drivers_MCX_MCXW_MCXW727C._spc_lowPower_request_output_override.kSPC_LowPowerRequestForcedHigh (*C enumerator*), 684
drivers_MCX_MCXW_MCXW727C._spc_lowPower_request_output_override.kSPC_LowPowerRequestForcedLow (*C enumerator*), 684
drivers_MCX_MCXW_MCXW727C._spc_lowPower_request_output_override.kSPC_LowPowerRequestNotForced (*C enumerator*), 684
drivers_MCX_MCXW_MCXW727C._spc_lowPower_request_output_override.kSPC_LowPowerRequestReserved (*C enumerator*), 684
drivers_MCX_MCXW_MCXW727C._spc_lowPower_request_pin_polarity (*C enum*), 684
drivers_MCX_MCXW_MCXW727C._spc_lowPower_request_pin_polarity.kSPC_HighTruePolarity (*C enumerator*), 684
drivers_MCX_MCXW_MCXW727C._spc_lowPower_request_pin_polarity.kSPC_LowTruePolarity (*C enumerator*), 684
drivers_MCX_MCXW_MCXW727C._spc_power_domain_id (*C enum*), 683
drivers_MCX_MCXW_MCXW727C._spc_power_domain_id.kSPC_PowerDomain0 (*C enumerator*), 683
drivers_MCX_MCXW_MCXW727C._spc_power_domain_id.kSPC_PowerDomain1 (*C enumerator*), 683
drivers_MCX_MCXW_MCXW727C._spc_power_domain_id.kSPC_PowerDomain2 (*C enumerator*), 683
drivers_MCX_MCXW_MCXW727C._spc_power_domain_id.kSPC_PowerDomain3 (*C enumerator*), 683
drivers_MCX_MCXW_MCXW727C._spc_power_domain_low_power_mode (*C enum*), 683
drivers_MCX_MCXW_MCXW727C._spc_power_domain_low_power_mode.kSPC_DeepPowerDownWithSysClockO (*C enumerator*), 684
drivers_MCX_MCXW_MCXW727C._spc_power_domain_low_power_mode.kSPC_DeepSleepSysClockOff (*C enumerator*), 684
drivers_MCX_MCXW_MCXW727C._spc_power_domain_low_power_mode.kSPC_PowerDownWithSysClockOff (*C enumerator*), 684
drivers_MCX_MCXW_MCXW727C._spc_power_domain_low_power_mode.kSPC_SleepWithSysClockOff (*C enumerator*), 684
drivers_MCX_MCXW_MCXW727C._spc_power_domain_low_power_mode.kSPC_SleepWithSYSClockRunning (*C enumerator*), 683
drivers_MCX_MCXW_MCXW727C._spc_power_domains (*C enum*), 683
drivers_MCX_MCXW_MCXW727C._spc_power_domains.kSPC_2P4GPowerDoaminRetain (*C enumerator*), 683
drivers_MCX_MCXW_MCXW727C._spc_power_domains.kSPC_INFRAPowerDomainRetain (*C enumerator*), 683
drivers_MCX_MCXW_MCXW727C._spc_power_domains.kSPC_MAINPowerDomainRetain (*C enumerator*), 683
drivers_MCX_MCXW_MCXW727C._spc_power_domains.kSPC_WAKEPowerDomainRetain (*C

enumerator), 683
drivers_MCX_MCXW_MCXW727C._spc_sram_operat_voltage (*C enum*), 686
drivers_MCX_MCXW_MCXW727C._spc_sram_operat_voltage.kSPC_SRAM_OperatVoltage1P0V (*C enumerator*), 687
drivers_MCX_MCXW_MCXW727C._spc_sram_operat_voltage.kSPC_SRAM_OperatVoltage1P1V (*C enumerator*), 687
drivers_MCX_MCXW_MCXW727C._spc_sys_ldo_drive_strength (*C enum*), 686
drivers_MCX_MCXW_MCXW727C._spc_sys_ldo_drive_strength.kSPC_SysLDO_LowDriveStrength (*C enumerator*), 686
drivers_MCX_MCXW_MCXW727C._spc_sys_ldo_drive_strength.kSPC_SysLDO_NormalDriveStrength (*C enumerator*), 686
drivers_MCX_MCXW_MCXW727C._spc_sys_ldo_voltage_level (*C enum*), 685
drivers_MCX_MCXW_MCXW727C._spc_sys_ldo_voltage_level.kSPC_SysLDO_NormalVoltage (*C enumerator*), 685
drivers_MCX_MCXW_MCXW727C._spc_sys_ldo_voltage_level.kSPC_SysLDO_OverDriveVoltage (*C enumerator*), 685
drivers_MCX_MCXW_MCXW727C._spc_system_voltage_detect_config (*C struct*), 721
drivers_MCX_MCXW_MCXW727C._spc_system_voltage_detect_config.level (*C var*), 721
drivers_MCX_MCXW_MCXW727C._spc_system_voltage_detect_config.option (*C var*), 721
drivers_MCX_MCXW_MCXW727C._spc_vdd1p8_voltage_detect_config (*C struct*), 721
drivers_MCX_MCXW_MCXW727C._spc_vdd1p8_voltage_detect_config.LVDInterruptEnable (*C var*), 721
drivers_MCX_MCXW_MCXW727C._spc_vdd_core_glitch_detector_config (*C struct*), 721
drivers_MCX_MCXW_MCXW727C._spc_vdd_core_glitch_detector_config.enableInterrupt (*C var*), 722
drivers_MCX_MCXW_MCXW727C._spc_vdd_core_glitch_detector_config.enableReset (*C var*), 722
drivers_MCX_MCXW_MCXW727C._spc_vdd_core_glitch_detector_config.resetTimeoutValue (*C var*), 722
drivers_MCX_MCXW_MCXW727C._spc_vdd_core_glitch_detector_config.rippleCounterSelect (*C var*), 722
drivers_MCX_MCXW_MCXW727C._spc_vdd_core_glitch_ripple_counter_select (*C enum*), 687
drivers_MCX_MCXW_MCXW727C._spc_vdd_core_glitch_ripple_counter_select.kSPC_selectBit0Of4bitRippleCou (*C enumerator*), 687
drivers_MCX_MCXW_MCXW727C._spc_vdd_core_glitch_ripple_counter_select.kSPC_selectBit1Of4bitRippleCou (*C enumerator*), 687
drivers_MCX_MCXW_MCXW727C._spc_vdd_core_glitch_ripple_counter_select.kSPC_selectBit2Of4bitRippleCou (*C enumerator*), 687
drivers_MCX_MCXW_MCXW727C._spc_vdd_core_glitch_ripple_counter_select.kSPC_selectBit3Of4bitRippleCou (*C enumerator*), 687
drivers_MCX_MCXW_MCXW727C._spc_voltage_detect_flags (*C enum*), 682
drivers_MCX_MCXW_MCXW727C._spc_voltage_detect_flags.kSPC_CoreVDDHighVoltageDetectFlag (*C enumerator*), 682
drivers_MCX_MCXW_MCXW727C._spc_voltage_detect_flags.kSPC_CoreVDDLowVoltageDetectFlag (*C enumerator*), 683
drivers_MCX_MCXW_MCXW727C._spc_voltage_detect_flags.kSPC_IOVDDHighVoltageDetectFlag (*C enumerator*), 682
drivers_MCX_MCXW_MCXW727C._spc_voltage_detect_flags.kSPC_IOVDDLowVoltageDetectFlag (*C enumerator*), 682
drivers_MCX_MCXW_MCXW727C._spc_voltage_detect_flags.kSPC_SystemVDDHighVoltageDetectFlag (*C enumerator*), 682
drivers_MCX_MCXW_MCXW727C._spc_voltage_detect_flags.kSPC_SystemVDDLowVoltageDetectFlag (*C enumerator*), 683
drivers_MCX_MCXW_MCXW727C._spc_voltage_detect_flags.kSPC_VDD1P8LowVoltageDetectFlag (*C enumerator*), 682
drivers_MCX_MCXW_MCXW727C._spc_voltage_detect_option (*C struct*), 720
drivers_MCX_MCXW_MCXW727C._spc_voltage_detect_option.HVDInterruptEnable (*C var*), 720
drivers_MCX_MCXW_MCXW727C._spc_voltage_detect_option.HVDResetEnable (*C var*), 720

drivers_MCX_MCXW_MCXW727C._spc_voltage_detect_option.LVDInterruptEnable (*C var*), 720
drivers_MCX_MCXW_MCXW727C._spc_voltage_detect_option.LVDResetEnable (*C var*), 720
drivers_MCX_MCXW_MCXW727C._status_groups (*C enum*), 452
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_AON_COMMON (*C enumerator*), 460
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_AON_POWER (*C enumerator*), 460
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_ApplicationRangeStart (*C enumerator*), 457
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_ASRC (*C enumerator*), 459
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_AUDMIX (*C enumerator*), 457
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_BMA (*C enumerator*), 460
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_BUTTON (*C enumerator*), 458
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_CAAM (*C enumerator*), 455
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_CLIF (*C enumerator*), 460
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_CODEC (*C enumerator*), 459
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_COMMON_TASK (*C enumerator*), 458
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_CSI (*C enumerator*), 454
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_CSS_PKC (*C enumerator*), 460
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_DCP (*C enumerator*), 455
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_DebugConsole (*C enumerator*), 456
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_DMA (*C enumerator*), 454
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_DMAMGR (*C enumerator*), 455
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_DMIC (*C enumerator*), 455
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_DSPI (*C enumerator*), 452
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_ECSPI (*C enumerator*), 455
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_EDMA (*C enumerator*), 454
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_ELA_CSEC (*C enumerator*), 460
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_ELE (*C enumerator*), 460
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_ELEMU (*C enumerator*), 459
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_ELS_PKC (*C enumerator*), 459
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_ENDAT3 (*C enumerator*), 460
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_ENET (*C enumerator*), 454
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_ENET_QOS (*C enumerator*), 459
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_ESAI (*C enumerator*), 456
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_EVENT_CTRL (*C enumerator*), 461
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_EXTERN_EEPROM (*C enumerator*), 458
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_FLASH (*C enumerator*), 452
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_FLASHIAP (*C enumerator*), 454
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_FLEXCAN (*C enumerator*), 455
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_FLEXCOMM_I2C (*C enumerator*), 454
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_FLEXIO_A_FORMAT (*C enumerator*), 460
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_FLEXIO_CAMERA (*C enumerator*), 455
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_FLEXIO_I2C (*C enumerator*), 452
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_FLEXIO_I2S (*C enumerator*), 453

drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_FLEXIO_MCULCD (*C enumerator*), 453
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_FLEXIO_SPI (*C enumerator*), 452
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_FLEXIO_T_FORMAT (*C enumerator*), 460
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_FLEXIO_UART (*C enumerator*), 452
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_FLEXSPI (*C enumerator*), 456
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_Generic (*C enumerator*), 452
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_GLIKEY (*C enumerator*), 460
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_HAL_ADC_SENSOR (*C enumerator*), 458
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_HAL_FLASH (*C enumerator*), 458
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_HAL_GPIO (*C enumerator*), 457
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_HAL_I2C (*C enumerator*), 457
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_HAL_I2S (*C enumerator*), 458
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_HAL_PWM (*C enumerator*), 458
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_HAL_RNG (*C enumerator*), 458
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_HAL_SPI (*C enumerator*), 457
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_HAL_TIMER (*C enumerator*), 457
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_HAL_UART (*C enumerator*), 457
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_HASHCRYPT (*C enumerator*), 456
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_HIPERFACE (*C enumerator*), 460
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_HOSTIF (*C enumerator*), 460
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_I2C (*C enumerator*), 453
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_I2S (*C enumerator*), 454
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_I3C (*C enumerator*), 456
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_I3CBUS (*C enumerator*), 459
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_IAP (*C enumerator*), 457
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_ICS (*C enumerator*), 456
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_IPED (*C enumerator*), 459
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_IUART (*C enumerator*), 454
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_JPEGDEC (*C enumerator*), 457
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_LED (*C enumerator*), 458
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_LIST (*C enumerator*), 458
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_LMEM (*C enumerator*), 454
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_LOG (*C enumerator*), 459
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_LPC_I2C (*C enumerator*), 455
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_LPC_I2C_1 (*C enumerator*), 456
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_LPC_MINISPI (*C enumerator*), 456
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_LPC_QSPI (*C enumerator*), 460
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_LPC_SPI (*C enumerator*), 455
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_LPC_SPI_SSP (*C enumerator*), 456
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_LPC_USART (*C enumerator*), 455

drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_LPI2C (*C enumerator*), 453
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_LPSCI (*C enumerator*), 453
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_LPSPI (*C enumerator*), 452
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_LPUART (*C enumerator*), 453
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_LTC (*C enumerator*), 455
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_MCAN (*C enumerator*), 455
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_MCG (*C enumerator*), 453
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_MECC (*C enumerator*), 459
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_MEM_MANAGER (*C enumerator*), 458
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_MIPI_DSI (*C enumerator*), 454
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_MMDC (*C enumerator*), 456
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_MSCAN (*C enumerator*), 456
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_MSG (*C enumerator*), 458
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_NETC (*C enumerator*), 460
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_NOTIFIER (*C enumerator*), 456
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_NPX (*C enumerator*), 460
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_OSA (*C enumerator*), 458
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_OTFAD (*C enumerator*), 459
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_OTP (*C enumerator*), 455
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_PDM (*C enumerator*), 456
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_PHY (*C enumerator*), 454
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_PNGDEC (*C enumerator*), 457
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_POWER (*C enumerator*), 454
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_POWER_MANAGER (*C enumerator*), 459
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_PUF (*C enumerator*), 457
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_QSCI (*C enumerator*), 459
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_QSPI (*C enumerator*), 454
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_QUEUEDSPI (*C enumerator*), 459
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_SAI (*C enumerator*), 453
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_SCG (*C enumerator*), 453
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_SDHC (*C enumerator*), 453
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_SDIF (*C enumerator*), 455
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_SDIOSLV (*C enumerator*), 459
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_SDK_FLEXSPINOR (*C enumerator*), 459
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_SDK_OCOTP (*C enumerator*), 459
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_SDMA (*C enumerator*), 456
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_SDMMC (*C enumerator*), 453
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_SDRAMC (*C enumerator*), 454
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_SDSPI (*C enumerator*), 453
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_SEMA42 (*C enumerator*), 453
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_SEMC (*C enumerator*), 456
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_SERIALMANAGER (*C enumerator*), 458
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_SFA (*C enumerator*), 457
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_SHELL (*C enumerator*), 458
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_SMARTCARD (*C enumerator*), 454
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_SPC (*C enumerator*), 457
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_SPDIF (*C enumerator*), 456
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_SPI (*C enumerator*), 453
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_SPIFI (*C enumerator*), 455

drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_TIMERMANAGER (*C enumerator*), 458
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_TOUCH_PANEL (*C enumerator*), 457
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_TRGMUX (*C enumerator*), 454
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_UART (*C enumerator*), 453
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_USDHC (*C enumerator*), 455
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_VBAT (*C enumerator*), 457
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_XRDC (*C enumerator*), 453
drivers_MCX_MCXW_MCXW727C._status_groups.kStatusGroup_XSPI (*C enumerator*), 457
drivers_MCX_MCXW_MCXW727C._syspm_event (*C enum*), 722
drivers_MCX_MCXW_MCXW727C._syspm_event.kSYSPM_Event1 (*C enumerator*), 722
drivers_MCX_MCXW_MCXW727C._syspm_event.kSYSPM_Event2 (*C enumerator*), 723
drivers_MCX_MCXW_MCXW727C._syspm_event.kSYSPM_Event3 (*C enumerator*), 723
drivers_MCX_MCXW_MCXW727C._syspm_mode (*C enum*), 723
drivers_MCX_MCXW_MCXW727C._syspm_mode.kSYSPM_BothMode (*C enumerator*), 723
drivers_MCX_MCXW_MCXW727C._syspm_mode.kSYSPM_PrivilegedMode (*C enumerator*), 723
drivers_MCX_MCXW_MCXW727C._syspm_mode.kSYSPM_UserMode (*C enumerator*), 723
drivers_MCX_MCXW_MCXW727C._syspm_monitor (*C enum*), 722
drivers_MCX_MCXW_MCXW727C._syspm_monitor.kSYSPM_Monitor0 (*C enumerator*), 722
drivers_MCX_MCXW_MCXW727C._syspm_startstop_control (*C enum*), 723
drivers_MCX_MCXW_MCXW727C._syspm_startstop_control.KSYSPM_EnableTraceControl (*C enumerator*), 723
drivers_MCX_MCXW_MCXW727C._syspm_startstop_control.kSYSPM_GlobalStart (*C enumerator*), 723
drivers_MCX_MCXW_MCXW727C._syspm_startstop_control.kSYSPM_GlobalStop (*C enumerator*), 723
drivers_MCX_MCXW_MCXW727C._syspm_startstop_control.kSYSPM_Idle (*C enumerator*), 723
drivers_MCX_MCXW_MCXW727C._syspm_startstop_control.kSYSPM_LocalStart (*C enumerator*), 723
drivers_MCX_MCXW_MCXW727C._syspm_startstop_control.kSYSPM_LocalStop (*C enumerator*), 723
drivers_MCX_MCXW_MCXW727C._tpm2_ip_src (*C enum*), 182
drivers_MCX_MCXW_MCXW727C._tpm2_ip_src.kCLOCK_Tpm2SrcCoreClk (*C enumerator*), 183
drivers_MCX_MCXW_MCXW727C._tpm2_ip_src.kCLOCK_Tpm2SrcSoscClk (*C enumerator*), 183
drivers_MCX_MCXW_MCXW727C._tpm_chnl (*C enum*), 733
drivers_MCX_MCXW_MCXW727C._tpm_chnl_control_bit_mask (*C enum*), 735
drivers_MCX_MCXW_MCXW727C._tpm_chnl_control_bit_mask.kTPM_ChnlELSnAMask (*C enumerator*), 735
drivers_MCX_MCXW_MCXW727C._tpm_chnl_control_bit_mask.kTPM_ChnlELSnBMask (*C enumerator*), 735
drivers_MCX_MCXW_MCXW727C._tpm_chnl_control_bit_mask.kTPM_ChnlMSAMask (*C enumerator*), 735
drivers_MCX_MCXW_MCXW727C._tpm_chnl_control_bit_mask.kTPM_ChnlMSBMask (*C enumerator*), 735
drivers_MCX_MCXW_MCXW727C._tpm_chnl_pwm_signal_param (*C struct*), 742
drivers_MCX_MCXW_MCXW727C._tpm_chnl_pwm_signal_param.chnlNumber (*C var*), 742
drivers_MCX_MCXW_MCXW727C._tpm_chnl_pwm_signal_param.deadTimeValue (*C var*), 742
drivers_MCX_MCXW_MCXW727C._tpm_chnl_pwm_signal_param.dutyCyclePercent (*C var*), 742
drivers_MCX_MCXW_MCXW727C._tpm_chnl_pwm_signal_param.enableComplementary (*C var*), 742
drivers_MCX_MCXW_MCXW727C._tpm_chnl_pwm_signal_param.firstEdgeDelayPercent (*C var*), 742
drivers_MCX_MCXW_MCXW727C._tpm_chnl_pwm_signal_param.level (*C var*), 742
drivers_MCX_MCXW_MCXW727C._tpm_chnl_pwm_signal_param.pauseLevel (*C var*), 742
drivers_MCX_MCXW_MCXW727C._tpm_chnl_pwm_signal_param.secPauseLevel (*C var*), 742

drivers_MCX_MCXW_MCXW727C._tpm_chnl.kTPM_Chnl_0 (*C enumerator*), 733
drivers_MCX_MCXW_MCXW727C._tpm_chnl.kTPM_Chnl_1 (*C enumerator*), 733
drivers_MCX_MCXW_MCXW727C._tpm_chnl.kTPM_Chnl_2 (*C enumerator*), 733
drivers_MCX_MCXW_MCXW727C._tpm_chnl.kTPM_Chnl_3 (*C enumerator*), 733
drivers_MCX_MCXW_MCXW727C._tpm_chnl.kTPM_Chnl_4 (*C enumerator*), 734
drivers_MCX_MCXW_MCXW727C._tpm_chnl.kTPM_Chnl_5 (*C enumerator*), 734
drivers_MCX_MCXW_MCXW727C._tpm_chnl.kTPM_Chnl_6 (*C enumerator*), 734
drivers_MCX_MCXW_MCXW727C._tpm_chnl.kTPM_Chnl_7 (*C enumerator*), 734
drivers_MCX_MCXW_MCXW727C._tpm_clock_prescale (*C enum*), 738
drivers_MCX_MCXW_MCXW727C._tpm_clock_prescale.kTPM_Prescale_Divide_1 (*C enumerator*), 738
drivers_MCX_MCXW_MCXW727C._tpm_clock_prescale.kTPM_Prescale_Divide_2 (*C enumerator*), 738
drivers_MCX_MCXW_MCXW727C._tpm_clock_prescale.kTPM_Prescale_Divide_4 (*C enumerator*), 738
drivers_MCX_MCXW_MCXW727C._tpm_clock_prescale.kTPM_Prescale_Divide_8 (*C enumerator*), 738
drivers_MCX_MCXW_MCXW727C._tpm_clock_prescale.kTPM_Prescale_Divide_16 (*C enumerator*), 738
drivers_MCX_MCXW_MCXW727C._tpm_clock_prescale.kTPM_Prescale_Divide_32 (*C enumerator*), 738
drivers_MCX_MCXW_MCXW727C._tpm_clock_prescale.kTPM_Prescale_Divide_64 (*C enumerator*), 738
drivers_MCX_MCXW_MCXW727C._tpm_clock_prescale.kTPM_Prescale_Divide_128 (*C enumerator*), 738
drivers_MCX_MCXW_MCXW727C._tpm_clock_source (*C enum*), 738
drivers_MCX_MCXW_MCXW727C._tpm_clock_source.kTPM_ExternalClock (*C enumerator*), 738
drivers_MCX_MCXW_MCXW727C._tpm_clock_source.kTPM_ExternalInputTriggerClock (*C enumerator*), 738
drivers_MCX_MCXW_MCXW727C._tpm_clock_source.kTPM_SystemClock (*C enumerator*), 738
drivers_MCX_MCXW_MCXW727C._tpm_config (*C struct*), 743
drivers_MCX_MCXW_MCXW727C._tpm_config.chnlPolarity (*C var*), 744
drivers_MCX_MCXW_MCXW727C._tpm_config.enableDebugMode (*C var*), 744
drivers_MCX_MCXW_MCXW727C._tpm_config.enableDoze (*C var*), 744
drivers_MCX_MCXW_MCXW727C._tpm_config.enablePauseOnTrigger (*C var*), 744
drivers_MCX_MCXW_MCXW727C._tpm_config.enableReloadOnTrigger (*C var*), 744
drivers_MCX_MCXW_MCXW727C._tpm_config.enableStartOnTrigger (*C var*), 744
drivers_MCX_MCXW_MCXW727C._tpm_config.enableStopOnOverflow (*C var*), 744
drivers_MCX_MCXW_MCXW727C._tpm_config.extTriggerPolarity (*C var*), 744
drivers_MCX_MCXW_MCXW727C._tpm_config.prescale (*C var*), 743
drivers_MCX_MCXW_MCXW727C._tpm_config.syncGlobalTimeBase (*C var*), 744
drivers_MCX_MCXW_MCXW727C._tpm_config.triggerSelect (*C var*), 744
drivers_MCX_MCXW_MCXW727C._tpm_config.triggerSource (*C var*), 744
drivers_MCX_MCXW_MCXW727C._tpm_config.useGlobalTimeBase (*C var*), 743
drivers_MCX_MCXW_MCXW727C._tpm_dual_edge_capture_param (*C struct*), 743
drivers_MCX_MCXW_MCXW727C._tpm_dual_edge_capture_param.currChanEdgeMode (*C var*), 743
drivers_MCX_MCXW_MCXW727C._tpm_dual_edge_capture_param.enableSwap (*C var*), 743
drivers_MCX_MCXW_MCXW727C._tpm_dual_edge_capture_param.nextChanEdgeMode (*C var*), 743
drivers_MCX_MCXW_MCXW727C._tpm_ext_trigger_polarity (*C enum*), 736
drivers_MCX_MCXW_MCXW727C._tpm_ext_trigger_polarity.kTPM_ExtTrigger_Active_High (*C enumerator*), 736
drivers_MCX_MCXW_MCXW727C._tpm_ext_trigger_polarity.kTPM_ExtTrigger_Active_Low (*C enumerator*), 736
drivers_MCX_MCXW_MCXW727C._tpm_input_capture_edge (*C enum*), 737

drivers_MCX_MCXW_MCXW727C._tpm_input_capture_edge.kTPM_FallingEdge (*C enumerator*), 737
drivers_MCX_MCXW_MCXW727C._tpm_input_capture_edge.kTPM_RiseAndFallEdge (*C enumerator*), 737
drivers_MCX_MCXW_MCXW727C._tpm_input_capture_edge.kTPM_RisingEdge (*C enumerator*), 737
drivers_MCX_MCXW_MCXW727C._tpm_interrupt_enable (*C enum*), 738
drivers_MCX_MCXW_MCXW727C._tpm_interrupt_enable.kTPM_Chnl0InterruptEnable (*C enumerator*), 738
drivers_MCX_MCXW_MCXW727C._tpm_interrupt_enable.kTPM_Chnl1InterruptEnable (*C enumerator*), 739
drivers_MCX_MCXW_MCXW727C._tpm_interrupt_enable.kTPM_Chnl2InterruptEnable (*C enumerator*), 739
drivers_MCX_MCXW_MCXW727C._tpm_interrupt_enable.kTPM_Chnl3InterruptEnable (*C enumerator*), 739
drivers_MCX_MCXW_MCXW727C._tpm_interrupt_enable.kTPM_Chnl4InterruptEnable (*C enumerator*), 739
drivers_MCX_MCXW_MCXW727C._tpm_interrupt_enable.kTPM_Chnl5InterruptEnable (*C enumerator*), 739
drivers_MCX_MCXW_MCXW727C._tpm_interrupt_enable.kTPM_Chnl6InterruptEnable (*C enumerator*), 739
drivers_MCX_MCXW_MCXW727C._tpm_interrupt_enable.kTPM_Chnl7InterruptEnable (*C enumerator*), 739
drivers_MCX_MCXW_MCXW727C._tpm_interrupt_enable.kTPM_TimeOverflowInterruptEnable (*C enumerator*), 739
drivers_MCX_MCXW_MCXW727C._tpm_output_compare_mode (*C enum*), 736
drivers_MCX_MCXW_MCXW727C._tpm_output_compare_mode.kTPM_ClearOnMatch (*C enumerator*), 737
drivers_MCX_MCXW_MCXW727C._tpm_output_compare_mode.kTPM_HighPulseOutput (*C enumerator*), 737
drivers_MCX_MCXW_MCXW727C._tpm_output_compare_mode.kTPM_LowPulseOutput (*C enumerator*), 737
drivers_MCX_MCXW_MCXW727C._tpm_output_compare_mode.kTPM_NoOutputSignal (*C enumerator*), 736
drivers_MCX_MCXW_MCXW727C._tpm_output_compare_mode.kTPM_SetOnMatch (*C enumerator*), 737
drivers_MCX_MCXW_MCXW727C._tpm_output_compare_mode.kTPM_ToggleOnMatch (*C enumerator*), 736
drivers_MCX_MCXW_MCXW727C._tpm_phase_param (*C struct*), 743
drivers_MCX_MCXW_MCXW727C._tpm_phase_param.phaseFilterVal (*C var*), 743
drivers_MCX_MCXW_MCXW727C._tpm_phase_param.phasePolarity (*C var*), 743
drivers_MCX_MCXW_MCXW727C._tpm_phase_polarity (*C enum*), 737
drivers_MCX_MCXW_MCXW727C._tpm_phase_polarity.kTPM_QuadPhaseInvert (*C enumerator*), 737
drivers_MCX_MCXW_MCXW727C._tpm_phase_polarity.kTPM_QuadPhaseNormal (*C enumerator*), 737
drivers_MCX_MCXW_MCXW727C._tpm_pwm_level_select (*C enum*), 734
drivers_MCX_MCXW_MCXW727C._tpm_pwm_level_select.kTPM_HighTrue (*C enumerator*), 734
drivers_MCX_MCXW_MCXW727C._tpm_pwm_level_select.kTPM_LowTrue (*C enumerator*), 734
drivers_MCX_MCXW_MCXW727C._tpm_pwm_mode (*C enum*), 734
drivers_MCX_MCXW_MCXW727C._tpm_pwm_mode.kTPM_CombinedPwm (*C enumerator*), 734
drivers_MCX_MCXW_MCXW727C._tpm_pwm_mode.kTPM_EdgeAlignedPwm (*C enumerator*), 734
drivers_MCX_MCXW_MCXW727C._tpm_pwm_pause_level_select (*C enum*), 734
drivers_MCX_MCXW_MCXW727C._tpm_pwm_pause_level_select.kTPM_ClearOnPause (*C enumerator*), 734
drivers_MCX_MCXW_MCXW727C._tpm_pwm_pause_level_select.kTPM_SetOnPause (*C enu-*

merator), 734
drivers_MCX_MCXW_MCXW727C._tpm_quad_decode_mode (*C enum*), 737
drivers_MCX_MCXW_MCXW727C._tpm_quad_decode_mode.kTPM_QuadCountAndDir (*C enumerator*), 737
drivers_MCX_MCXW_MCXW727C._tpm_quad_decode_mode.kTPM_QuadPhaseEncode (*C enumerator*), 737
drivers_MCX_MCXW_MCXW727C._tpm_status_flags (*C enum*), 739
drivers_MCX_MCXW_MCXW727C._tpm_status_flags.kTPM_Chnl0Flag (*C enumerator*), 739
drivers_MCX_MCXW_MCXW727C._tpm_status_flags.kTPM_Chnl1Flag (*C enumerator*), 739
drivers_MCX_MCXW_MCXW727C._tpm_status_flags.kTPM_Chnl2Flag (*C enumerator*), 739
drivers_MCX_MCXW_MCXW727C._tpm_status_flags.kTPM_Chnl3Flag (*C enumerator*), 739
drivers_MCX_MCXW_MCXW727C._tpm_status_flags.kTPM_Chnl4Flag (*C enumerator*), 739
drivers_MCX_MCXW_MCXW727C._tpm_status_flags.kTPM_Chnl5Flag (*C enumerator*), 739
drivers_MCX_MCXW_MCXW727C._tpm_status_flags.kTPM_Chnl6Flag (*C enumerator*), 739
drivers_MCX_MCXW_MCXW727C._tpm_status_flags.kTPM_Chnl7Flag (*C enumerator*), 740
drivers_MCX_MCXW_MCXW727C._tpm_status_flags.kTPM_TimeOverflowFlag (*C enumerator*), 740
drivers_MCX_MCXW_MCXW727C._tpm_trigger_select (*C enum*), 735
drivers_MCX_MCXW_MCXW727C._tpm_trigger_select.kTPM_Trigger_Select_0 (*C enumerator*), 735
drivers_MCX_MCXW_MCXW727C._tpm_trigger_select.kTPM_Trigger_Select_1 (*C enumerator*), 735
drivers_MCX_MCXW_MCXW727C._tpm_trigger_select.kTPM_Trigger_Select_2 (*C enumerator*), 735
drivers_MCX_MCXW_MCXW727C._tpm_trigger_select.kTPM_Trigger_Select_3 (*C enumerator*), 735
drivers_MCX_MCXW_MCXW727C._tpm_trigger_select.kTPM_Trigger_Select_4 (*C enumerator*), 735
drivers_MCX_MCXW_MCXW727C._tpm_trigger_select.kTPM_Trigger_Select_5 (*C enumerator*), 735
drivers_MCX_MCXW_MCXW727C._tpm_trigger_select.kTPM_Trigger_Select_6 (*C enumerator*), 735
drivers_MCX_MCXW_MCXW727C._tpm_trigger_select.kTPM_Trigger_Select_7 (*C enumerator*), 735
drivers_MCX_MCXW_MCXW727C._tpm_trigger_select.kTPM_Trigger_Select_8 (*C enumerator*), 735
drivers_MCX_MCXW_MCXW727C._tpm_trigger_select.kTPM_Trigger_Select_9 (*C enumerator*), 735
drivers_MCX_MCXW_MCXW727C._tpm_trigger_select.kTPM_Trigger_Select_10 (*C enumerator*), 736
drivers_MCX_MCXW_MCXW727C._tpm_trigger_select.kTPM_Trigger_Select_11 (*C enumerator*), 736
drivers_MCX_MCXW_MCXW727C._tpm_trigger_select.kTPM_Trigger_Select_12 (*C enumerator*), 736
drivers_MCX_MCXW_MCXW727C._tpm_trigger_select.kTPM_Trigger_Select_13 (*C enumerator*), 736
drivers_MCX_MCXW_MCXW727C._tpm_trigger_select.kTPM_Trigger_Select_14 (*C enumerator*), 736
drivers_MCX_MCXW_MCXW727C._tpm_trigger_select.kTPM_Trigger_Select_15 (*C enumerator*), 736
drivers_MCX_MCXW_MCXW727C._tpm_trigger_source (*C enum*), 736
drivers_MCX_MCXW_MCXW727C._tpm_trigger_source.kTPM_TriggerSource_External (*C enumerator*), 736
drivers_MCX_MCXW_MCXW727C._tpm_trigger_source.kTPM_TriggerSource_Internal (*C enumerator*), 736
drivers_MCX_MCXW_MCXW727C._trdc_controller (*C enum*), 754
drivers_MCX_MCXW_MCXW727C._trdc_controller.kTRDC_MemBlockController0 (*C enumerator*), 754

drivers_MCX_MCXW_MCXW727C._trdc_controller.kTRDC_MemBlockController1 (*C enumerator*), 754
drivers_MCX_MCXW_MCXW727C._trdc_controller.kTRDC_MemBlockController2 (*C enumerator*), 754
drivers_MCX_MCXW_MCXW727C._trdc_controller.kTRDC_MemBlockController3 (*C enumerator*), 754
drivers_MCX_MCXW_MCXW727C._trdc_controller.kTRDC_MemRegionChecker0 (*C enumerator*), 754
drivers_MCX_MCXW_MCXW727C._trdc_controller.kTRDC_MemRegionChecker1 (*C enumerator*), 754
drivers_MCX_MCXW_MCXW727C._trdc_controller.kTRDC_MemRegionChecker2 (*C enumerator*), 754
drivers_MCX_MCXW_MCXW727C._trdc_controller.kTRDC_MemRegionChecker3 (*C enumerator*), 754
drivers_MCX_MCXW_MCXW727C._trdc_controller.kTRDC_MemRegionChecker4 (*C enumerator*), 754
drivers_MCX_MCXW_MCXW727C._trdc_controller.kTRDC_MemRegionChecker5 (*C enumerator*), 754
drivers_MCX_MCXW_MCXW727C._trdc_controller.kTRDC_MemRegionChecker6 (*C enumerator*), 754
drivers_MCX_MCXW_MCXW727C._trdc_did_sel (*C enum*), 752
drivers_MCX_MCXW_MCXW727C._trdc_did_sel.kTRDC_DidInput (*C enumerator*), 753
drivers_MCX_MCXW_MCXW727C._trdc_did_sel.kTRDC_DidMda (*C enumerator*), 753
drivers_MCX_MCXW_MCXW727C._trdc_did_sel.kTRDC_DidMdaAndInput (*C enumerator*), 753
drivers_MCX_MCXW_MCXW727C._trdc_did_sel.kTRDC_DidReserved (*C enumerator*), 753
drivers_MCX_MCXW_MCXW727C._trdc_domain_error (*C struct*), 765
drivers_MCX_MCXW_MCXW727C._trdc_domain_error.address (*C var*), 765
drivers_MCX_MCXW_MCXW727C._trdc_domain_error.controller (*C var*), 765
drivers_MCX_MCXW_MCXW727C._trdc_domain_error.domainId (*C var*), 765
drivers_MCX_MCXW_MCXW727C._trdc_domain_error.errorAttr (*C var*), 765
drivers_MCX_MCXW_MCXW727C._trdc_domain_error.errorPort (*C var*), 765
drivers_MCX_MCXW_MCXW727C._trdc_domain_error.errorState (*C var*), 765
drivers_MCX_MCXW_MCXW727C._trdc_domain_error.errorType (*C var*), 765
drivers_MCX_MCXW_MCXW727C._trdc_domain_error.slaveMemoryIdx (*C var*), 765
drivers_MCX_MCXW_MCXW727C._trdc_error_attr (*C enum*), 755
drivers_MCX_MCXW_MCXW727C._trdc_error_attr.kTRDC_ErrorNonSecurePrivilegeData (*C enumerator*), 755
drivers_MCX_MCXW_MCXW727C._trdc_error_attr.kTRDC_ErrorNonSecurePrivilegeInst (*C enumerator*), 755
drivers_MCX_MCXW_MCXW727C._trdc_error_attr.kTRDC_ErrorNonSecureUserData (*C enumerator*), 755
drivers_MCX_MCXW_MCXW727C._trdc_error_attr.kTRDC_ErrorNonSecureUserInst (*C enumerator*), 755
drivers_MCX_MCXW_MCXW727C._trdc_error_attr.kTRDC_ErrorSecurePrivilegeData (*C enumerator*), 755
drivers_MCX_MCXW_MCXW727C._trdc_error_attr.kTRDC_ErrorSecurePrivilegeInst (*C enumerator*), 755
drivers_MCX_MCXW_MCXW727C._trdc_error_attr.kTRDC_ErrorSecureUserData (*C enumerator*), 755
drivers_MCX_MCXW_MCXW727C._trdc_error_attr.kTRDC_ErrorSecureUserInst (*C enumerator*), 755
drivers_MCX_MCXW_MCXW727C._trdc_error_state (*C enum*), 754
drivers_MCX_MCXW_MCXW727C._trdc_error_state.kTRDC_ErrorStateMulti (*C enumerator*), 755
drivers_MCX_MCXW_MCXW727C._trdc_error_state.kTRDC_ErrorStateNone (*C enumerator*), 754
drivers_MCX_MCXW_MCXW727C._trdc_error_state.kTRDC_ErrorStateNone1 (*C enumerator*), 754

drivers_MCX_MCXW_MCXW727C._trdc_error_state.kTRDC_ErrorStateSingle (*C enumerator*), 755
drivers_MCX_MCXW_MCXW727C._trdc_error_type (*C enum*), 755
drivers_MCX_MCXW_MCXW727C._trdc_error_type.kTRDC_ErrorTypeRead (*C enumerator*), 755
drivers_MCX_MCXW_MCXW727C._trdc_error_type.kTRDC_ErrorTypeWrite (*C enumerator*), 755
drivers_MCX_MCXW_MCXW727C._trdc_flw_config (*C struct*), 764
drivers_MCX_MCXW_MCXW727C._trdc_flw_config.arrayBaseAddr (*C var*), 764
drivers_MCX_MCXW_MCXW727C._trdc_flw_config.blockCount (*C var*), 764
drivers_MCX_MCXW_MCXW727C._trdc_flw_config.enable (*C var*), 765
drivers_MCX_MCXW_MCXW727C._trdc_flw_config.lock (*C var*), 765
drivers_MCX_MCXW_MCXW727C._trdc_hardware_config (*C struct*), 762
drivers_MCX_MCXW_MCXW727C._trdc_hardware_config.domainNumber (*C var*), 762
drivers_MCX_MCXW_MCXW727C._trdc_hardware_config.masterNumber (*C var*), 762
drivers_MCX_MCXW_MCXW727C._trdc_hardware_config.mbcNumber (*C var*), 762
drivers_MCX_MCXW_MCXW727C._trdc_hardware_config.mrcNumber (*C var*), 762
drivers_MCX_MCXW_MCXW727C._trdc_idau_config (*C struct*), 764
drivers_MCX_MCXW_MCXW727C._trdc_idau_config.__pad0__ (*C var*), 764
drivers_MCX_MCXW_MCXW727C._trdc_idau_config.__pad1__ (*C var*), 764
drivers_MCX_MCXW_MCXW727C._trdc_idau_config.lockNonsecureMPU (*C var*), 764
drivers_MCX_MCXW_MCXW727C._trdc_idau_config.lockNonsecureVTOR (*C var*), 764
drivers_MCX_MCXW_MCXW727C._trdc_idau_config.lockSAU (*C var*), 764
drivers_MCX_MCXW_MCXW727C._trdc_idau_config.lockSecureMPU (*C var*), 764
drivers_MCX_MCXW_MCXW727C._trdc_idau_config.lockSecureVTOR (*C var*), 764
drivers_MCX_MCXW_MCXW727C._trdc_MBC_bit (*C enum*), 759
drivers_MCX_MCXW_MCXW727C._trdc_MBC_bit.kTRDC_MbcBit0 (*C enumerator*), 759
drivers_MCX_MCXW_MCXW727C._trdc_MBC_bit.kTRDC_MbcBit1 (*C enumerator*), 759
drivers_MCX_MCXW_MCXW727C._trdc_MBC_bit.kTRDC_MbcBit2 (*C enumerator*), 759
drivers_MCX_MCXW_MCXW727C._trdc_MBC_bit.kTRDC_MbcBit3 (*C enumerator*), 759
drivers_MCX_MCXW_MCXW727C._trdc_MBC_bit.kTRDC_MbcBit4 (*C enumerator*), 759
drivers_MCX_MCXW_MCXW727C._trdc_MBC_bit.kTRDC_MbcBit5 (*C enumerator*), 759
drivers_MCX_MCXW_MCXW727C._trdc_MBC_bit.kTRDC_MbcBit6 (*C enumerator*), 759
drivers_MCX_MCXW_MCXW727C._trdc_MBC_bit.kTRDC_MbcBit7 (*C enumerator*), 759
drivers_MCX_MCXW_MCXW727C._trdc_MBC_bit.kTRDC_MbcBit8 (*C enumerator*), 759
drivers_MCX_MCXW_MCXW727C._trdc_MBC_bit.kTRDC_MbcBit9 (*C enumerator*), 759
drivers_MCX_MCXW_MCXW727C._trdc_MBC_bit.kTRDC_MbcBit10 (*C enumerator*), 759
drivers_MCX_MCXW_MCXW727C._trdc_MBC_bit.kTRDC_MbcBit11 (*C enumerator*), 759
drivers_MCX_MCXW_MCXW727C._trdc_MBC_bit.kTRDC_MbcBit12 (*C enumerator*), 759
drivers_MCX_MCXW_MCXW727C._trdc_MBC_bit.kTRDC_MbcBit13 (*C enumerator*), 759
drivers_MCX_MCXW_MCXW727C._trdc_MBC_bit.kTRDC_MbcBit14 (*C enumerator*), 759
drivers_MCX_MCXW_MCXW727C._trdc_MBC_bit.kTRDC_MbcBit15 (*C enumerator*), 760
drivers_MCX_MCXW_MCXW727C._trdc_MBC_bit.kTRDC_MbcBit16 (*C enumerator*), 760
drivers_MCX_MCXW_MCXW727C._trdc_MBC_bit.kTRDC_MbcBit17 (*C enumerator*), 760
drivers_MCX_MCXW_MCXW727C._trdc_MBC_bit.kTRDC_MbcBit18 (*C enumerator*), 760
drivers_MCX_MCXW_MCXW727C._trdc_MBC_bit.kTRDC_MbcBit19 (*C enumerator*), 760
drivers_MCX_MCXW_MCXW727C._trdc_MBC_bit.kTRDC_MbcBit20 (*C enumerator*), 760
drivers_MCX_MCXW_MCXW727C._trdc_MBC_bit.kTRDC_MbcBit21 (*C enumerator*), 760
drivers_MCX_MCXW_MCXW727C._trdc_MBC_bit.kTRDC_MbcBit22 (*C enumerator*), 760
drivers_MCX_MCXW_MCXW727C._trdc_MBC_bit.kTRDC_MbcBit23 (*C enumerator*), 760
drivers_MCX_MCXW_MCXW727C._trdc_MBC_bit.kTRDC_MbcBit24 (*C enumerator*), 760
drivers_MCX_MCXW_MCXW727C._trdc_MBC_bit.kTRDC_MbcBit25 (*C enumerator*), 760
drivers_MCX_MCXW_MCXW727C._trdc_MBC_bit.kTRDC_MbcBit26 (*C enumerator*), 760
drivers_MCX_MCXW_MCXW727C._trdc_MBC_bit.kTRDC_MbcBit27 (*C enumerator*), 760
drivers_MCX_MCXW_MCXW727C._trdc_MBC_bit.kTRDC_MbcBit28 (*C enumerator*), 760
drivers_MCX_MCXW_MCXW727C._trdc_MBC_bit.kTRDC_MbcBit29 (*C enumerator*), 760
drivers_MCX_MCXW_MCXW727C._trdc_MBC_bit.kTRDC_MbcBit30 (*C enumerator*), 760
drivers_MCX_MCXW_MCXW727C._trdc_MBC_bit.kTRDC_MbcBit31 (*C enumerator*), 761

drivers_MCX_MCXW_MCXW727C._trdc_MBC_domain (*C enum*), 758
drivers_MCX_MCXW_MCXW727C._trdc_MBC_domain.kTRDC_MbcDomain0 (*C enumerator*), 758
drivers_MCX_MCXW_MCXW727C._trdc_MBC_domain.kTRDC_MbcDomain1 (*C enumerator*), 758
drivers_MCX_MCXW_MCXW727C._trdc_MBC_domain.kTRDC_MbcDomain2 (*C enumerator*), 758
drivers_MCX_MCXW_MCXW727C._trdc_MBC_domain.kTRDC_MbcDomain3 (*C enumerator*), 758
drivers_MCX_MCXW_MCXW727C._trdc_MBC_domain.kTRDC_MbcDomain4 (*C enumerator*), 758
drivers_MCX_MCXW_MCXW727C._trdc_MBC_domain.kTRDC_MbcDomain5 (*C enumerator*), 758
drivers_MCX_MCXW_MCXW727C._trdc_MBC_domain.kTRDC_MbcDomain6 (*C enumerator*), 758
drivers_MCX_MCXW_MCXW727C._trdc_MBC_domain.kTRDC_MbcDomain7 (*C enumerator*), 758
drivers_MCX_MCXW_MCXW727C._trdc_MBC_memory (*C enum*), 758
drivers_MCX_MCXW_MCXW727C._trdc_mbc_memory_block_config (*C struct*), 768
drivers_MCX_MCXW_MCXW727C._trdc_mbc_memory_block_config.domainIdx (*C var*), 768
drivers_MCX_MCXW_MCXW727C._trdc_mbc_memory_block_config.mbcIdx (*C var*), 768
drivers_MCX_MCXW_MCXW727C._trdc_mbc_memory_block_config.memoryAccessControlSelect (*C var*), 768
drivers_MCX_MCXW_MCXW727C._trdc_mbc_memory_block_config.memoryBlockIdx (*C var*), 768
drivers_MCX_MCXW_MCXW727C._trdc_mbc_memory_block_config.nseEnable (*C var*), 768
drivers_MCX_MCXW_MCXW727C._trdc_mbc_memory_block_config.slaveMemoryIdx (*C var*), 768
drivers_MCX_MCXW_MCXW727C._trdc_MBC_memory.kTRDC_MbcSlaveMemory0 (*C enumerator*), 758
drivers_MCX_MCXW_MCXW727C._trdc_MBC_memory.kTRDC_MbcSlaveMemory1 (*C enumerator*), 758
drivers_MCX_MCXW_MCXW727C._trdc_MBC_memory.kTRDC_MbcSlaveMemory2 (*C enumerator*), 758
drivers_MCX_MCXW_MCXW727C._trdc_MBC_memory.kTRDC_MbcSlaveMemory3 (*C enumerator*), 758
drivers_MCX_MCXW_MCXW727C._trdc_mbc_nse_update_config (*C struct*), 767
drivers_MCX_MCXW_MCXW727C._trdc_mbc_nse_update_config.__pad0__ (*C var*), 767
drivers_MCX_MCXW_MCXW727C._trdc_mbc_nse_update_config.__pad1__ (*C var*), 767
drivers_MCX_MCXW_MCXW727C._trdc_mbc_nse_update_config.__pad2__ (*C var*), 767
drivers_MCX_MCXW_MCXW727C._trdc_mbc_nse_update_config.__pad3__ (*C var*), 768
drivers_MCX_MCXW_MCXW727C._trdc_mbc_nse_update_config.autoIncrement (*C var*), 768
drivers_MCX_MCXW_MCXW727C._trdc_mbc_nse_update_config.domianSelect (*C var*), 768
drivers_MCX_MCXW_MCXW727C._trdc_mbc_nse_update_config.memorySelect (*C var*), 767
drivers_MCX_MCXW_MCXW727C._trdc_mbc_nse_update_config.wordIdx (*C var*), 767
drivers_MCX_MCXW_MCXW727C._trdc_memory_access_control_config (*C struct*), 765
drivers_MCX_MCXW_MCXW727C._trdc_memory_access_control_config.__pad0__ (*C var*), 766
drivers_MCX_MCXW_MCXW727C._trdc_memory_access_control_config.__pad1__ (*C var*), 766
drivers_MCX_MCXW_MCXW727C._trdc_memory_access_control_config.__pad2__ (*C var*), 766
drivers_MCX_MCXW_MCXW727C._trdc_memory_access_control_config.__pad3__ (*C var*), 766
drivers_MCX_MCXW_MCXW727C._trdc_memory_access_control_config.lock (*C var*), 766
drivers_MCX_MCXW_MCXW727C._trdc_memory_access_control_config.nonsecurePrivR (*C var*), 766
drivers_MCX_MCXW_MCXW727C._trdc_memory_access_control_config.nonsecurePrivW (*C var*), 766
drivers_MCX_MCXW_MCXW727C._trdc_memory_access_control_config.nonsecurePrivX (*C var*), 766
drivers_MCX_MCXW_MCXW727C._trdc_memory_access_control_config.nonsecureUsrR (*C var*),

766
drivers_MCX_MCXW_MCXW727C._trdc_memory_access_control_config.nonsecureUsrW (*C var*), 765
drivers_MCX_MCXW_MCXW727C._trdc_memory_access_control_config.nonsecureUsrX (*C var*), 765
drivers_MCX_MCXW_MCXW727C._trdc_memory_access_control_config.securePrivR (*C var*), 766
drivers_MCX_MCXW_MCXW727C._trdc_memory_access_control_config.securePrivW (*C var*), 766
drivers_MCX_MCXW_MCXW727C._trdc_memory_access_control_config.securePrivX (*C var*), 766
drivers_MCX_MCXW_MCXW727C._trdc_memory_access_control_config.secureUsrR (*C var*), 766
drivers_MCX_MCXW_MCXW727C._trdc_memory_access_control_config.secureUsrW (*C var*), 766
drivers_MCX_MCXW_MCXW727C._trdc_memory_access_control_config.secureUsrX (*C var*), 766
drivers_MCX_MCXW_MCXW727C._trdc_MRC_domain (*C enum*), 757
drivers_MCX_MCXW_MCXW727C._trdc_MRC_domain.kTRDC_MrcDomain0 (*C enumerator*), 757
drivers_MCX_MCXW_MCXW727C._trdc_MRC_domain.kTRDC_MrcDomain1 (*C enumerator*), 757
drivers_MCX_MCXW_MCXW727C._trdc_MRC_domain.kTRDC_MrcDomain2 (*C enumerator*), 757
drivers_MCX_MCXW_MCXW727C._trdc_MRC_domain.kTRDC_MrcDomain3 (*C enumerator*), 757
drivers_MCX_MCXW_MCXW727C._trdc_MRC_domain.kTRDC_MrcDomain4 (*C enumerator*), 757
drivers_MCX_MCXW_MCXW727C._trdc_MRC_domain.kTRDC_MrcDomain5 (*C enumerator*), 757
drivers_MCX_MCXW_MCXW727C._trdc_MRC_domain.kTRDC_MrcDomain6 (*C enumerator*), 757
drivers_MCX_MCXW_MCXW727C._trdc_MRC_domain.kTRDC_MrcDomain7 (*C enumerator*), 757
drivers_MCX_MCXW_MCXW727C._trdc_MRC_domain.kTRDC_MrcDomain8 (*C enumerator*), 757
drivers_MCX_MCXW_MCXW727C._trdc_MRC_domain.kTRDC_MrcDomain9 (*C enumerator*), 757
drivers_MCX_MCXW_MCXW727C._trdc_MRC_domain.kTRDC_MrcDomain10 (*C enumerator*), 757
drivers_MCX_MCXW_MCXW727C._trdc_MRC_domain.kTRDC_MrcDomain11 (*C enumerator*), 757
drivers_MCX_MCXW_MCXW727C._trdc_MRC_domain.kTRDC_MrcDomain12 (*C enumerator*), 757
drivers_MCX_MCXW_MCXW727C._trdc_MRC_domain.kTRDC_MrcDomain13 (*C enumerator*), 757
drivers_MCX_MCXW_MCXW727C._trdc_MRC_domain.kTRDC_MrcDomain14 (*C enumerator*), 757
drivers_MCX_MCXW_MCXW727C._trdc_MRC_domain.kTRDC_MrcDomain15 (*C enumerator*), 758
drivers_MCX_MCXW_MCXW727C._trdc_mrc_region_descriptor_config (*C struct*), 766
drivers_MCX_MCXW_MCXW727C._trdc_mrc_region_descriptor_config.domainIdx (*C var*), 767
drivers_MCX_MCXW_MCXW727C._trdc_mrc_region_descriptor_config.endAddr (*C var*), 767
drivers_MCX_MCXW_MCXW727C._trdc_mrc_region_descriptor_config.memoryAccessControlSelect (*C var*), 767
drivers_MCX_MCXW_MCXW727C._trdc_mrc_region_descriptor_config.mrcIdx (*C var*), 767
drivers_MCX_MCXW_MCXW727C._trdc_mrc_region_descriptor_config.nseEnable (*C var*), 767
drivers_MCX_MCXW_MCXW727C._trdc_mrc_region_descriptor_config.regionIdx (*C var*), 767
drivers_MCX_MCXW_MCXW727C._trdc_mrc_region_descriptor_config.startAddr (*C var*), 767
drivers_MCX_MCXW_MCXW727C._trdc_mrc_region_descriptor_config.valid (*C var*), 767
drivers_MCX_MCXW_MCXW727C._trdc_non_processor_domain_assignment (*C struct*), 763
drivers_MCX_MCXW_MCXW727C._trdc_non_processor_domain_assignment.__pad0__ (*C var*), 763

drivers_MCX_MCXW_MCXW727C._trdc_non_processor_domain_assignment.__pad1__ (*C var*), 764
drivers_MCX_MCXW_MCXW727C._trdc_non_processor_domain_assignment.bypassDomainId (*C var*), 763
drivers_MCX_MCXW_MCXW727C._trdc_non_processor_domain_assignment.domainId (*C var*), 763
drivers_MCX_MCXW_MCXW727C._trdc_non_processor_domain_assignment.lock (*C var*), 764
drivers_MCX_MCXW_MCXW727C._trdc_non_processor_domain_assignment.privilegeAttr (*C var*), 763
drivers_MCX_MCXW_MCXW727C._trdc_non_processor_domain_assignment.secureAttr (*C var*), 763
drivers_MCX_MCXW_MCXW727C._trdc_privilege_attr (*C enum*), 753
drivers_MCX_MCXW_MCXW727C._trdc_privilege_attr.kTRDC_ForcePrivilege (*C enumerator*), 753
drivers_MCX_MCXW_MCXW727C._trdc_privilege_attr.kTRDC_ForceUser (*C enumerator*), 753
drivers_MCX_MCXW_MCXW727C._trdc_privilege_attr.kTRDC_MasterPrivilege (*C enumerator*), 753
drivers_MCX_MCXW_MCXW727C._trdc_privilege_attr.kTRDC_MasterPrivilege1 (*C enumerator*), 753
drivers_MCX_MCXW_MCXW727C._trdc_processor_domain_assignment (*C struct*), 763
drivers_MCX_MCXW_MCXW727C._trdc_processor_domain_assignment.__pad0__ (*C var*), 763
drivers_MCX_MCXW_MCXW727C._trdc_processor_domain_assignment.__pad1__ (*C var*), 763
drivers_MCX_MCXW_MCXW727C._trdc_processor_domain_assignment.__pad2__ (*C var*), 763
drivers_MCX_MCXW_MCXW727C._trdc_processor_domain_assignment.domainId (*C var*), 763
drivers_MCX_MCXW_MCXW727C._trdc_processor_domain_assignment.domainIdSelect (*C var*), 763
drivers_MCX_MCXW_MCXW727C._trdc_processor_domain_assignment.lock (*C var*), 763
drivers_MCX_MCXW_MCXW727C._trdc_processor_domain_assignment.secureAttr (*C var*), 763
drivers_MCX_MCXW_MCXW727C._trdc_region_descriptor (*C enum*), 755
drivers_MCX_MCXW_MCXW727C._trdc_region_descriptor.kTRDC_RegionDescriptor0 (*C enumerator*), 756
drivers_MCX_MCXW_MCXW727C._trdc_region_descriptor.kTRDC_RegionDescriptor1 (*C enumerator*), 756
drivers_MCX_MCXW_MCXW727C._trdc_region_descriptor.kTRDC_RegionDescriptor2 (*C enumerator*), 756
drivers_MCX_MCXW_MCXW727C._trdc_region_descriptor.kTRDC_RegionDescriptor3 (*C enumerator*), 756
drivers_MCX_MCXW_MCXW727C._trdc_region_descriptor.kTRDC_RegionDescriptor4 (*C enumerator*), 756
drivers_MCX_MCXW_MCXW727C._trdc_region_descriptor.kTRDC_RegionDescriptor5 (*C enumerator*), 756
drivers_MCX_MCXW_MCXW727C._trdc_region_descriptor.kTRDC_RegionDescriptor6 (*C enumerator*), 756
drivers_MCX_MCXW_MCXW727C._trdc_region_descriptor.kTRDC_RegionDescriptor7 (*C enumerator*), 756
drivers_MCX_MCXW_MCXW727C._trdc_region_descriptor.kTRDC_RegionDescriptor8 (*C enumerator*), 756
drivers_MCX_MCXW_MCXW727C._trdc_region_descriptor.kTRDC_RegionDescriptor9 (*C enumerator*), 756
drivers_MCX_MCXW_MCXW727C._trdc_region_descriptor.kTRDC_RegionDescriptor10 (*C enumerator*), 756
drivers_MCX_MCXW_MCXW727C._trdc_region_descriptor.kTRDC_RegionDescriptor11 (*C enumerator*), 756
drivers_MCX_MCXW_MCXW727C._trdc_region_descriptor.kTRDC_RegionDescriptor12 (*C enumerator*), 756
drivers_MCX_MCXW_MCXW727C._trdc_region_descriptor.kTRDC_RegionDescriptor13 (*C enumerator*), 756
drivers_MCX_MCXW_MCXW727C._trdc_region_descriptor.kTRDC_RegionDescriptor14 (*C enu-

merator), 756
drivers_MCX_MCXW_MCXW727C._trdc_region_descriptor.kTRDC_RegionDescriptor15 (*C enumerator*), 756
drivers_MCX_MCXW_MCXW727C._trdc_secure_attr (*C enum*), 753
drivers_MCX_MCXW_MCXW727C._trdc_secure_attr.kTRDC_ForceNonSecure (*C enumerator*), 753
drivers_MCX_MCXW_MCXW727C._trdc_secure_attr.kTRDC_ForceSecure (*C enumerator*), 753
drivers_MCX_MCXW_MCXW727C._trdc_secure_attr.kTRDC_MasterSecure (*C enumerator*), 753
drivers_MCX_MCXW_MCXW727C._trdc_secure_attr.kTRDC_MasterSecure1 (*C enumerator*), 753
drivers_MCX_MCXW_MCXW727C._trdc_slave_memory_hardware_config (*C struct*), 762
drivers_MCX_MCXW_MCXW727C._trdc_slave_memory_hardware_config.blockNum (*C var*), 762
drivers_MCX_MCXW_MCXW727C._trdc_slave_memory_hardware_config.blockSize (*C var*), 762
drivers_MCX_MCXW_MCXW727C._trgmux_trigger_input (*C enum*), 769
drivers_MCX_MCXW_MCXW727C._trgmux_trigger_input.kTRGMUX_TriggerInput0 (*C enumerator*), 769
drivers_MCX_MCXW_MCXW727C._trgmux_trigger_input.kTRGMUX_TriggerInput1 (*C enumerator*), 769
drivers_MCX_MCXW_MCXW727C._trgmux_trigger_input.kTRGMUX_TriggerInput2 (*C enumerator*), 769
drivers_MCX_MCXW_MCXW727C._trgmux_trigger_input.kTRGMUX_TriggerInput3 (*C enumerator*), 769
drivers_MCX_MCXW_MCXW727C._vbat_bandgap_refresh_period (*C enum*), 771
drivers_MCX_MCXW_MCXW727C._vbat_bandgap_refresh_period.kVBAT_BandgapRefresh7P8125ms (*C enumerator*), 772
drivers_MCX_MCXW_MCXW727C._vbat_bandgap_refresh_period.kVBAT_BandgapRefresh15P625ms (*C enumerator*), 772
drivers_MCX_MCXW_MCXW727C._vbat_bandgap_refresh_period.kVBAT_BandgapRefresh31P25ms (*C enumerator*), 772
drivers_MCX_MCXW_MCXW727C._vbat_bandgap_refresh_period.kVBAT_BandgapRefresh62P5ms (*C enumerator*), 772
drivers_MCX_MCXW_MCXW727C._vbat_bandgap_timer_id (*C enum*), 771
drivers_MCX_MCXW_MCXW727C._vbat_bandgap_timer_id.kVBAT_BandgapTimer0 (*C enumerator*), 771
drivers_MCX_MCXW_MCXW727C._vbat_bandgap_timer_id.kVBAT_BandgapTimer1 (*C enumerator*), 771
drivers_MCX_MCXW_MCXW727C._vbat_bandgap_timer_timeout_period (*C enum*), 772
drivers_MCX_MCXW_MCXW727C._vbat_bandgap_timer_timeout_period.kVBAT_BangapTimerTimeout1s (*C enumerator*), 772
drivers_MCX_MCXW_MCXW727C._vbat_bandgap_timer_timeout_period.kVBAT_BangapTimerTimeout31P25ms (*C enumerator*), 772
drivers_MCX_MCXW_MCXW727C._vbat_bandgap_timer_timeout_period.kVBAT_BangapTimerTimeout62P5ms (*C enumerator*), 772
drivers_MCX_MCXW_MCXW727C._vbat_bandgap_timer_timeout_period.kVBAT_BangapTimerTimeout125ms (*C enumerator*), 772
drivers_MCX_MCXW_MCXW727C._vbat_bandgap_timer_timeout_period.kVBAT_BangapTimerTimeout250ms (*C enumerator*), 772
drivers_MCX_MCXW_MCXW727C._vbat_bandgap_timer_timeout_period.kVBAT_BangapTimerTimeout500ms (*C enumerator*), 772
drivers_MCX_MCXW_MCXW727C._vbat_fro16k_config (*C struct*), 777
drivers_MCX_MCXW_MCXW727C._vbat_fro16k_config.enableFRO16k (*C var*), 778
drivers_MCX_MCXW_MCXW727C._vbat_fro16k_config.enableFRO16kOutput (*C var*), 778
drivers_MCX_MCXW_MCXW727C._vbat_interrupt_enable (*C enum*), 770
drivers_MCX_MCXW_MCXW727C._vbat_interrupt_enable.kVBAT_AllInterruptsEnable (*C enumerator*), 771
drivers_MCX_MCXW_MCXW727C._vbat_interrupt_enable.kVBAT_InterruptEnableBandgapTimer0 (*C enumerator*), 770
drivers_MCX_MCXW_MCXW727C._vbat_interrupt_enable.kVBAT_InterruptEnableBandgapTimer1

(*C enumerator*), 771
drivers_MCX_MCXW_MCXW727C._vbat_interrupt_enable.kVBAT_InterruptEnableLdoReady (*C enumerator*), 771
drivers_MCX_MCXW_MCXW727C._vbat_interrupt_enable.kVBAT_InterruptEnablePORDetect (*C enumerator*), 770
drivers_MCX_MCXW_MCXW727C._vbat_interrupt_enable.kVBAT_InterruptEnableWakeupPin (*C enumerator*), 770
drivers_MCX_MCXW_MCXW727C._vbat_status_flag (*C enum*), 770
drivers_MCX_MCXW_MCXW727C._vbat_status_flag.kVBAT_StatusFlagBandgapTimer0 (*C enumerator*), 770
drivers_MCX_MCXW_MCXW727C._vbat_status_flag.kVBAT_StatusFlagBandgapTimer1 (*C enumerator*), 770
drivers_MCX_MCXW_MCXW727C._vbat_status_flag.kVBAT_StatusFlagLdoReady (*C enumerator*), 770
drivers_MCX_MCXW_MCXW727C._vbat_status_flag.kVBAT_StatusFlagPORDetect (*C enumerator*), 770
drivers_MCX_MCXW_MCXW727C._vbat_status_flag.kVBAT_StatusFlagWakeupPin (*C enumerator*), 770
drivers_MCX_MCXW_MCXW727C._vbat_wakeup_enable (*C enum*), 771
drivers_MCX_MCXW_MCXW727C._vbat_wakeup_enable.kVBAT_AllWakeupsEnable (*C enumerator*), 771
drivers_MCX_MCXW_MCXW727C._vbat_wakeup_enable.kVBAT_WakeupEnableBandgapTimer0 (*C enumerator*), 771
drivers_MCX_MCXW_MCXW727C._vbat_wakeup_enable.kVBAT_WakeupEnableBandgapTimer1 (*C enumerator*), 771
drivers_MCX_MCXW_MCXW727C._vbat_wakeup_enable.kVBAT_WakeupEnableLdoReady (*C enumerator*), 771
drivers_MCX_MCXW_MCXW727C._vbat_wakeup_enable.kVBAT_WakeupEnablePORDetect (*C enumerator*), 771
drivers_MCX_MCXW_MCXW727C._vbat_wakeup_enable.kVBAT_WakeupEnableWakeupPin (*C enumerator*), 771
drivers_MCX_MCXW_MCXW727C._vref_buffer_mode (*C enum*), 780
drivers_MCX_MCXW_MCXW727C._vref_buffer_mode.kVREF_ModeBandgapOnly (*C enumerator*), 780
drivers_MCX_MCXW_MCXW727C._vref_buffer_mode.kVREF_ModeHighPowerBuffer (*C enumerator*), 780
drivers_MCX_MCXW_MCXW727C._vref_buffer_mode.kVREF_ModeLowPowerBuffer (*C enumerator*), 780
drivers_MCX_MCXW_MCXW727C._vref_config (*C struct*), 780
drivers_MCX_MCXW_MCXW727C._vref_config.bufferMode (*C var*), 780
drivers_MCX_MCXW_MCXW727C._vref_config.enableChopOscillator (*C var*), 780
drivers_MCX_MCXW_MCXW727C._vref_config.enableCurvatureCompensation (*C var*), 781
drivers_MCX_MCXW_MCXW727C._vref_config.enableHCBandgap (*C var*), 780
drivers_MCX_MCXW_MCXW727C._vref_config.enableInternalVoltageRegulator (*C var*), 780
drivers_MCX_MCXW_MCXW727C._vref_config.enableLowPowerBuff (*C var*), 781
drivers_MCX_MCXW_MCXW727C._wdog32_clock_prescaler (*C enum*), 785
drivers_MCX_MCXW_MCXW727C._wdog32_clock_prescaler.kWDOG32_ClockPrescalerDivide1 (*C enumerator*), 785
drivers_MCX_MCXW_MCXW727C._wdog32_clock_prescaler.kWDOG32_ClockPrescalerDivide256 (*C enumerator*), 785
drivers_MCX_MCXW_MCXW727C._wdog32_clock_source (*C enum*), 785
drivers_MCX_MCXW_MCXW727C._wdog32_clock_source.kWDOG32_ClockSource0 (*C enumerator*), 785
drivers_MCX_MCXW_MCXW727C._wdog32_clock_source.kWDOG32_ClockSource1 (*C enumerator*), 785
drivers_MCX_MCXW_MCXW727C._wdog32_clock_source.kWDOG32_ClockSource2 (*C enumerator*), 785
drivers_MCX_MCXW_MCXW727C._wdog32_clock_source.kWDOG32_ClockSource3 (*C enumera-*

tor), 785
drivers_MCX_MCXW_MCXW727C._wdog32_config (*C struct*), 787
drivers_MCX_MCXW_MCXW727C._wdog32_config.clockSource (*C var*), 787
drivers_MCX_MCXW_MCXW727C._wdog32_config.enableInterrupt (*C var*), 787
drivers_MCX_MCXW_MCXW727C._wdog32_config.enableUpdate (*C var*), 787
drivers_MCX_MCXW_MCXW727C._wdog32_config.enableWdog32 (*C var*), 787
drivers_MCX_MCXW_MCXW727C._wdog32_config.enableWindowMode (*C var*), 787
drivers_MCX_MCXW_MCXW727C._wdog32_config.prescaler (*C var*), 787
drivers_MCX_MCXW_MCXW727C._wdog32_config.testMode (*C var*), 787
drivers_MCX_MCXW_MCXW727C._wdog32_config.timeoutValue (*C var*), 788
drivers_MCX_MCXW_MCXW727C._wdog32_config.windowValue (*C var*), 787
drivers_MCX_MCXW_MCXW727C._wdog32_config.workMode (*C var*), 787
drivers_MCX_MCXW_MCXW727C._wdog32_interrupt_enable_t (*C enum*), 786
drivers_MCX_MCXW_MCXW727C._wdog32_interrupt_enable_t.kWDOG32_InterruptEnable (*C enumerator*), 786
drivers_MCX_MCXW_MCXW727C._wdog32_status_flags_t (*C enum*), 786
drivers_MCX_MCXW_MCXW727C._wdog32_status_flags_t.kWDOG32_InterruptFlag (*C enumerator*), 786
drivers_MCX_MCXW_MCXW727C._wdog32_status_flags_t.kWDOG32_RunningFlag (*C enumerator*), 786
drivers_MCX_MCXW_MCXW727C._wdog32_test_mode (*C enum*), 785
drivers_MCX_MCXW_MCXW727C._wdog32_test_mode.kWDOG32_HighByteTest (*C enumerator*), 786
drivers_MCX_MCXW_MCXW727C._wdog32_test_mode.kWDOG32_LowByteTest (*C enumerator*), 786
drivers_MCX_MCXW_MCXW727C._wdog32_test_mode.kWDOG32_TestModeDisabled (*C enumerator*), 785
drivers_MCX_MCXW_MCXW727C._wdog32_test_mode.kWDOG32_UserModeEnabled (*C enumerator*), 785
drivers_MCX_MCXW_MCXW727C._wdog32_work_mode (*C struct*), 787
drivers_MCX_MCXW_MCXW727C._wdog32_work_mode.enableDebug (*C var*), 787
drivers_MCX_MCXW_MCXW727C._wdog32_work_mode.enableStop (*C var*), 787
drivers_MCX_MCXW_MCXW727C._wdog32_work_mode.enableWait (*C var*), 787
drivers_MCX_MCXW_MCXW727C._wuu_external_pin_edge_detection (*C enum*), 790
drivers_MCX_MCXW_MCXW727C._wuu_external_pin_edge_detection.kWUU_ExternalPinAnyEdge (*C enumerator*), 790
drivers_MCX_MCXW_MCXW727C._wuu_external_pin_edge_detection.kWUU_ExternalPinDisable (*C enumerator*), 790
drivers_MCX_MCXW_MCXW727C._wuu_external_pin_edge_detection.kWUU_ExternalPinFallingEdge (*C enumerator*), 790
drivers_MCX_MCXW_MCXW727C._wuu_external_pin_edge_detection.kWUU_ExternalPinRisingEdge (*C enumerator*), 790
drivers_MCX_MCXW_MCXW727C._wuu_external_wakeup_pin_config (*C struct*), 793
drivers_MCX_MCXW_MCXW727C._wuu_external_wakeup_pin_config.edge (*C var*), 793
drivers_MCX_MCXW_MCXW727C._wuu_external_wakeup_pin_config.event (*C var*), 793
drivers_MCX_MCXW_MCXW727C._wuu_external_wakeup_pin_config.mode (*C var*), 793
drivers_MCX_MCXW_MCXW727C._wuu_external_wakeup_pin_event (*C enum*), 790
drivers_MCX_MCXW_MCXW727C._wuu_external_wakeup_pin_event.kWUU_ExternalPinDMARequest (*C enumerator*), 790
drivers_MCX_MCXW_MCXW727C._wuu_external_wakeup_pin_event.kWUU_ExternalPinInterrupt (*C enumerator*), 790
drivers_MCX_MCXW_MCXW727C._wuu_external_wakeup_pin_event.kWUU_ExternalPinTriggerEvent (*C enumerator*), 791
drivers_MCX_MCXW_MCXW727C._wuu_external_wakeup_pin_mode (*C enum*), 791
drivers_MCX_MCXW_MCXW727C._wuu_external_wakeup_pin_mode.kWUU_ExternalPinActiveAlways (*C enumerator*), 791
drivers_MCX_MCXW_MCXW727C._wuu_external_wakeup_pin_mode.kWUU_ExternalPinActiveDSPD (*C enumerator*), 791

drivers_MCX_MCXW_MCXW727C._wuu_external_wakeup_pin_mode.kWUU_ExternalPinActiveLowLeakageMod (*C enumerator*), 791
drivers_MCX_MCXW_MCXW727C._wuu_filter_edge (*C enum*), 791
drivers_MCX_MCXW_MCXW727C._wuu_filter_edge.kWUU_FilterAnyEdge (*C enumerator*), 791
drivers_MCX_MCXW_MCXW727C._wuu_filter_edge.kWUU_FilterDisabled (*C enumerator*), 791
drivers_MCX_MCXW_MCXW727C._wuu_filter_edge.kWUU_FilterNegedgeEnable (*C enumerator*), 791
drivers_MCX_MCXW_MCXW727C._wuu_filter_edge.kWUU_FilterPosedgeEnable (*C enumerator*), 791
drivers_MCX_MCXW_MCXW727C._wuu_filter_event (*C enum*), 791
drivers_MCX_MCXW_MCXW727C._wuu_filter_event.kWUU_FilterDMARequest (*C enumerator*), 792
drivers_MCX_MCXW_MCXW727C._wuu_filter_event.kWUU_FilterInterrupt (*C enumerator*), 792
drivers_MCX_MCXW_MCXW727C._wuu_filter_event.kWUU_FilterTriggerEvent (*C enumerator*), 792
drivers_MCX_MCXW_MCXW727C._wuu_filter_mode (*C enum*), 792
drivers_MCX_MCXW_MCXW727C._wuu_filter_mode.kWUU_FilterActiveAlways (*C enumerator*), 792
drivers_MCX_MCXW_MCXW727C._wuu_filter_mode.kWUU_FilterActiveDSPD (*C enumerator*), 792
drivers_MCX_MCXW_MCXW727C._wuu_filter_mode.kWUU_FilterActiveLowLeakageMode (*C enumerator*), 792
drivers_MCX_MCXW_MCXW727C._wuu_internal_wakeup_module_event (*C enum*), 791
drivers_MCX_MCXW_MCXW727C._wuu_internal_wakeup_module_event.kWUU_InternalModuleDMATrigger (*C enumerator*), 791
drivers_MCX_MCXW_MCXW727C._wuu_internal_wakeup_module_event.kWUU_InternalModuleInterrupt (*C enumerator*), 791
drivers_MCX_MCXW_MCXW727C._wuu_pin_filter_config (*C struct*), 793
drivers_MCX_MCXW_MCXW727C._wuu_pin_filter_config.edge (*C var*), 793
drivers_MCX_MCXW_MCXW727C._wuu_pin_filter_config.event (*C var*), 793
drivers_MCX_MCXW_MCXW727C._wuu_pin_filter_config.mode (*C var*), 793
drivers_MCX_MCXW_MCXW727C._wuu_pin_filter_config.pinIndex (*C var*), 793
drivers_MCX_MCXW_MCXW727C.ADC_OFSTRIM_OFSTRIM_MAX (*C macro*), 474
drivers_MCX_MCXW_MCXW727C.ADC_OFSTRIM_OFSTRIM_SIGN (*C macro*), 474
drivers_MCX_MCXW_MCXW727C.arm_cfft_instance_f32 (*C struct*), 169
drivers_MCX_MCXW_MCXW727C.arm_cfft_instance_f32.bitRevLength (*C var*), 170
drivers_MCX_MCXW_MCXW727C.arm_cfft_instance_f32.fftLen (*C var*), 169
drivers_MCX_MCXW_MCXW727C.arm_cfft_instance_f32.pBitRevTable (*C var*), 170
drivers_MCX_MCXW_MCXW727C.arm_cfft_instance_f32.pTwiddle (*C var*), 170
drivers_MCX_MCXW_MCXW727C.ARRAY_SIZE (*C macro*), 464
drivers_MCX_MCXW_MCXW727C.AT_CACHE_LINE_SECTION (*C macro*), 451
drivers_MCX_MCXW_MCXW727C.AT_CACHE_LINE_SECTION_INIT (*C macro*), 451
drivers_MCX_MCXW_MCXW727C.AT_NONCACHEABLE_SECTION (*C macro*), 451
drivers_MCX_MCXW_MCXW727C.AT_NONCACHEABLE_SECTION_ALIGN (*C macro*), 451
drivers_MCX_MCXW_MCXW727C.AT_NONCACHEABLE_SECTION_ALIGN_INIT (*C macro*), 451
drivers_MCX_MCXW_MCXW727C.AT_NONCACHEABLE_SECTION_INIT (*C macro*), 451
drivers_MCX_MCXW_MCXW727C.AT_QUICKACCESS_SECTION_CODE (*C macro*), 451
drivers_MCX_MCXW_MCXW727C.AT_QUICKACCESS_SECTION_DATA (*C macro*), 452
drivers_MCX_MCXW_MCXW727C.AT_QUICKACCESS_SECTION_DATA_ALIGN (*C macro*), 452
drivers_MCX_MCXW_MCXW727C.ccm32k_clock_monitor_config_t (*C type*), 166
drivers_MCX_MCXW_MCXW727C.ccm32k_clock_monitor_divide_trim_value_t (*C type*), 166
drivers_MCX_MCXW_MCXW727C.ccm32k_clock_monitor_freq_trim_value_t (*C type*), 166
drivers_MCX_MCXW_MCXW727C.ccm32k_clock_output_peripheral_t (*C type*), 166
drivers_MCX_MCXW_MCXW727C.ccm32k_clock_source_select_t (*C type*), 166
drivers_MCX_MCXW_MCXW727C.ccm32k_clock_source_t (*C type*), 165
drivers_MCX_MCXW_MCXW727C.CCM32K_Disable32kFroIFRLoad (*C function*), 157

drivers_MCX_MCXW_MCXW727C.CCM32K_DisableCLKOutToPeripherals (*C function*), 159
drivers_MCX_MCXW_MCXW727C.CCM32K_Enable32kFro (*C function*), 156
drivers_MCX_MCXW_MCXW727C.CCM32K_EnableCLKOutToPeripherals (*C function*), 159
drivers_MCX_MCXW_MCXW727C.CCM32K_EnableClockMonitor (*C function*), 158
drivers_MCX_MCXW_MCXW727C.CCM32K_Get32kFroTrimValue (*C function*), 157
drivers_MCX_MCXW_MCXW727C.CCM32K_GetClockSource (*C function*), 160
drivers_MCX_MCXW_MCXW727C.CCM32K_GetCurrentState (*C function*), 160
drivers_MCX_MCXW_MCXW727C.CCM32K_GetStatusFlag (*C function*), 159
drivers_MCX_MCXW_MCXW727C.CCM32K_Lock32kFroTrimWriteAccess (*C function*), 157
drivers_MCX_MCXW_MCXW727C.CCM32K_Lock32kFroWriteAccess (*C function*), 157
drivers_MCX_MCXW_MCXW727C.CCM32K_Lock32kOscWriteAccess (*C function*), 158
drivers_MCX_MCXW_MCXW727C.CCM32K_LockClockGateWriteAccess (*C function*), 159
drivers_MCX_MCXW_MCXW727C.CCM32K_LockClockMonitorWriteAccess (*C function*), 159
drivers_MCX_MCXW_MCXW727C.CCM32K_OSC32K_CTRL_OSC_MODE (*C macro*), 166
drivers_MCX_MCXW_MCXW727C.CCM32K_OSC32K_CTRL_OSC_MODE_MASK (*C macro*), 166
drivers_MCX_MCXW_MCXW727C.CCM32K_OSC32K_CTRL_OSC_MODE_SHIFT (*C macro*), 166
drivers_MCX_MCXW_MCXW727C.ccm32k_osc_coarse_adjustment_value_t (*C type*), 165
drivers_MCX_MCXW_MCXW727C.ccm32k_osc_config_t (*C type*), 166
drivers_MCX_MCXW_MCXW727C.ccm32k_osc_extal_cap_t (*C type*), 165
drivers_MCX_MCXW_MCXW727C.ccm32k_osc_fine_adjustment_value_t (*C type*), 165
drivers_MCX_MCXW_MCXW727C.ccm32k_osc_mode_t (*C type*), 165
drivers_MCX_MCXW_MCXW727C.ccm32k_osc_xtal_cap_t (*C type*), 165
drivers_MCX_MCXW_MCXW727C.CCM32K_SelectClockSource (*C function*), 159
drivers_MCX_MCXW_MCXW727C.CCM32K_Set32kFroTrimValue (*C function*), 157
drivers_MCX_MCXW_MCXW727C.CCM32K_Set32kOscConfig (*C function*), 157
drivers_MCX_MCXW_MCXW727C.CCM32K_SetClockMonitorConfig (*C function*), 159
drivers_MCX_MCXW_MCXW727C.CCM32K_SetClockMonitorDivideTrimValue (*C function*), 158
drivers_MCX_MCXW_MCXW727C.CCM32K_SetClockMonitorFreqTrimValue (*C function*), 158
drivers_MCX_MCXW_MCXW727C.ccm32k_state_t (*C type*), 165
drivers_MCX_MCXW_MCXW727C.ce_arm_cfft_f32 (*C function*), 169
drivers_MCX_MCXW_MCXW727C.CE_CmdAdd (*C function*), 168
drivers_MCX_MCXW_MCXW727C.CE_CmdCheckStatus (*C function*), 169
drivers_MCX_MCXW_MCXW727C.CE_CmdInitBuffer (*C function*), 168
drivers_MCX_MCXW_MCXW727C.CE_CmdLaunch (*C function*), 168
drivers_MCX_MCXW_MCXW727C.CE_CmdLaunchBlocking (*C function*), 169
drivers_MCX_MCXW_MCXW727C.CE_CmdLaunchNonBlocking (*C function*), 169
drivers_MCX_MCXW_MCXW727C.CE_CmdReset (*C function*), 168
drivers_MCX_MCXW_MCXW727C.CE_COMPUTE_TIMEOUT (*C macro*), 169
drivers_MCX_MCXW_MCXW727C.CE_Copy (*C function*), 167
drivers_MCX_MCXW_MCXW727C.ce_copy_image_t (*C type*), 170
drivers_MCX_MCXW_MCXW727C.CE_ExecCmd (*C function*), 167
drivers_MCX_MCXW_MCXW727C.CE_Init (*C function*), 170
drivers_MCX_MCXW_MCXW727C.CE_InitWithoutFirmware (*C function*), 170
drivers_MCX_MCXW_MCXW727C.CE_InstallFirmware (*C function*), 170
drivers_MCX_MCXW_MCXW727C.CE_MatrixAdd_F32 (*C function*), 171
drivers_MCX_MCXW_MCXW727C.CE_MatrixAdd_Q15 (*C function*), 170
drivers_MCX_MCXW_MCXW727C.CE_MatrixAdd_Q31 (*C function*), 171
drivers_MCX_MCXW_MCXW727C.CE_MatrixChol_CF32 (*C function*), 174
drivers_MCX_MCXW_MCXW727C.CE_MatrixElemMul_F32 (*C function*), 171
drivers_MCX_MCXW_MCXW727C.CE_MatrixEvdHerm_CF32 (*C function*), 173
drivers_MCX_MCXW_MCXW727C.CE_MatrixInvHerm_CF32 (*C function*), 172
drivers_MCX_MCXW_MCXW727C.CE_MatrixMul_CF32 (*C function*), 172
drivers_MCX_MCXW_MCXW727C.CE_MatrixMul_F32 (*C function*), 172
drivers_MCX_MCXW_MCXW727C.CE_NullCmd (*C function*), 167
drivers_MCX_MCXW_MCXW727C.CE_TransformCFFT_F16 (*C function*), 175
drivers_MCX_MCXW_MCXW727C.CE_TransformCFFT_F32 (*C function*), 175

drivers_MCX_MCXW_MCXW727C.CE_TransformIFFT_F16 (*C function*), 176
drivers_MCX_MCXW_MCXW727C.CE_TransformIFFT_F32 (*C function*), 176
drivers_MCX_MCXW_MCXW727C.CLOCK_ClearFircErr (*C function*), 179
drivers_MCX_MCXW_MCXW727C.CLOCK_ClearRoscErr (*C function*), 180
drivers_MCX_MCXW_MCXW727C.CLOCK_ClearSysOscErr (*C function*), 178
drivers_MCX_MCXW_MCXW727C.clock_clkout_src_t (*C type*), 191
drivers_MCX_MCXW_MCXW727C.CLOCK_DeinitFirc (*C function*), 179
drivers_MCX_MCXW_MCXW727C.CLOCK_DeinitRosc (*C function*), 180
drivers_MCX_MCXW_MCXW727C.CLOCK_DeinitSysOsc (*C function*), 177
drivers_MCX_MCXW_MCXW727C.CLOCK_DisableClock (*C function*), 193
drivers_MCX_MCXW_MCXW727C.CLOCK_DisableTPM2 (*C function*), 193
drivers_MCX_MCXW_MCXW727C.CLOCK_EnableClock (*C function*), 192
drivers_MCX_MCXW_MCXW727C.CLOCK_EnableClockLPMode (*C function*), 193
drivers_MCX_MCXW_MCXW727C.CLOCK_EnableTPM2 (*C function*), 193
drivers_MCX_MCXW_MCXW727C.CLOCK_GetBusClkFreq (*C function*), 194
drivers_MCX_MCXW_MCXW727C.CLOCK_GetCoreSysClkFreq (*C function*), 193
drivers_MCX_MCXW_MCXW727C.CLOCK_GetCurSysClkConfig (*C function*), 177
drivers_MCX_MCXW_MCXW727C.CLOCK_GetFircFreq (*C function*), 179
drivers_MCX_MCXW_MCXW727C.CLOCK_GetFlashClkFreq (*C function*), 194
drivers_MCX_MCXW_MCXW727C.CLOCK_GetFreq (*C function*), 193
drivers_MCX_MCXW_MCXW727C.CLOCK_GetIpFreq (*C function*), 194
drivers_MCX_MCXW_MCXW727C.CLOCK_GetPlatClkFreq (*C function*), 194
drivers_MCX_MCXW_MCXW727C.CLOCK_GetRfFro192MFreq (*C function*), 180
drivers_MCX_MCXW_MCXW727C.CLOCK_GetRtcOscFreq (*C function*), 180
drivers_MCX_MCXW_MCXW727C.CLOCK_GetSysClkFreq (*C function*), 177
drivers_MCX_MCXW_MCXW727C.CLOCK_GetSysOscFreq (*C function*), 178
drivers_MCX_MCXW_MCXW727C.CLOCK_InitFirc (*C function*), 178
drivers_MCX_MCXW_MCXW727C.CLOCK_InitRfFro192M (*C function*), 180
drivers_MCX_MCXW_MCXW727C.CLOCK_InitRosc (*C function*), 179
drivers_MCX_MCXW_MCXW727C.CLOCK_InitSysOsc (*C function*), 177
drivers_MCX_MCXW_MCXW727C.clock_ip_control_t (*C type*), 191
drivers_MCX_MCXW_MCXW727C.clock_ip_name_t (*C type*), 191
drivers_MCX_MCXW_MCXW727C.clock_ip_src_t (*C type*), 191
drivers_MCX_MCXW_MCXW727C.CLOCK_IsFIRCAutoTrimLocked (*C function*), 179
drivers_MCX_MCXW_MCXW727C.CLOCK_IsFircErr (*C function*), 179
drivers_MCX_MCXW_MCXW727C.CLOCK_IsFircValid (*C function*), 179
drivers_MCX_MCXW_MCXW727C.CLOCK_IsRoscErr (*C function*), 180
drivers_MCX_MCXW_MCXW727C.CLOCK_IsRoscValid (*C function*), 180
drivers_MCX_MCXW_MCXW727C.CLOCK_IsSysOscErr (*C function*), 178
drivers_MCX_MCXW_MCXW727C.CLOCK_IsSysOscValid (*C function*), 178
drivers_MCX_MCXW_MCXW727C.CLOCK_LockFircControlStatusReg (*C function*), 179
drivers_MCX_MCXW_MCXW727C.CLOCK_LockRoscControlStatusReg (*C function*), 181
drivers_MCX_MCXW_MCXW727C.CLOCK_LockSysOscControlStatusReg (*C function*), 178
drivers_MCX_MCXW_MCXW727C.clock_name_t (*C type*), 191
drivers_MCX_MCXW_MCXW727C.CLOCK_REG (*C macro*), 196
drivers_MCX_MCXW_MCXW727C.CLOCK_SetClkOutSel (*C function*), 177
drivers_MCX_MCXW_MCXW727C.CLOCK_SetIpSrc (*C function*), 193
drivers_MCX_MCXW_MCXW727C.CLOCK_SetIpSrcDiv (*C function*), 193
drivers_MCX_MCXW_MCXW727C.CLOCK_SetRoscMonitorMode (*C function*), 180
drivers_MCX_MCXW_MCXW727C.CLOCK_SetRunModeSysClkConfig (*C function*), 177
drivers_MCX_MCXW_MCXW727C.CLOCK_SetSysOscMonitorMode (*C function*), 178
drivers_MCX_MCXW_MCXW727C.CLOCK_SetTpm2Src (*C function*), 193
drivers_MCX_MCXW_MCXW727C.CLOCK_SetXtal0Freq (*C function*), 181
drivers_MCX_MCXW_MCXW727C.CLOCK_SetXtal32Freq (*C function*), 181
drivers_MCX_MCXW_MCXW727C.CLOCK_UnlockFircControlStatusReg (*C function*), 179
drivers_MCX_MCXW_MCXW727C.CLOCK_UnlockRoscControlStatusReg (*C function*), 181
drivers_MCX_MCXW_MCXW727C.CLOCK_UnlockSysOscControlStatusReg (*C function*), 178
drivers_MCX_MCXW_MCXW727C.CMC_BLR_LOCK_FIELD_WIDTH (*C macro*), 210

drivers_MCX_MCXW_MCXW727C.CMC_BLR_LOCK_IDX (*C macro*), 210
drivers_MCX_MCXW_MCXW727C.CMC_BLR_LOCK_IDX_MASK (*C macro*), 210
drivers_MCX_MCXW_MCXW727C.CMC_BLR_LOCK_IDX_SHIFT (*C macro*), 210
drivers_MCX_MCXW_MCXW727C.CMC_ClearCoreClockGatedStatus (*C function*), 198
drivers_MCX_MCXW_MCXW727C.CMC_ClearISPMODEPinLogic (*C function*), 202
drivers_MCX_MCXW_MCXW727C.CMC_ClearStickySystemResetStatus (*C function*), 199
drivers_MCX_MCXW_MCXW727C.CMC_ClearSystemResetInterruptFlags (*C function*), 202
drivers_MCX_MCXW_MCXW727C.cmc_clock_mode_t (*C type*), 210
drivers_MCX_MCXW_MCXW727C.CMC_ConfigFlashMode (*C function*), 204
drivers_MCX_MCXW_MCXW727C.CMC_ConfigResetPin (*C function*), 201
drivers_MCX_MCXW_MCXW727C.cmc_core_clock_gate_status_t (*C type*), 210
drivers_MCX_MCXW_MCXW727C.CMC_DisableSystemResetInterrupt (*C function*), 202
drivers_MCX_MCXW_MCXW727C.CMC_EnableDebugOperation (*C function*), 204
drivers_MCX_MCXW_MCXW727C.CMC_EnableNonMaskablePinInterrupt (*C function*), 202
drivers_MCX_MCXW_MCXW727C.CMC_EnableSystemResetInterrupt (*C function*), 201
drivers_MCX_MCXW_MCXW727C.CMC_EnterLowPowerMode (*C function*), 205
drivers_MCX_MCXW_MCXW727C.CMC_ForceBootConfiguration (*C function*), 203
drivers_MCX_MCXW_MCXW727C.CMC_GetBootRomStatus (*C function*), 203
drivers_MCX_MCXW_MCXW727C.CMC_GetClockMode (*C function*), 199
drivers_MCX_MCXW_MCXW727C.CMC_GetCoreClockGatedStatus (*C function*), 198
drivers_MCX_MCXW_MCXW727C.CMC_GetISPMODEPinLogic (*C function*), 202
drivers_MCX_MCXW_MCXW727C.CMC_GetMAINPowerMode (*C function*), 201
drivers_MCX_MCXW_MCXW727C.CMC_GetResetCount (*C function*), 199
drivers_MCX_MCXW_MCXW727C.CMC_GetResetInitStatusFlags (*C function*), 200
drivers_MCX_MCXW_MCXW727C.CMC_GetStickySystemResetStatus (*C function*), 199
drivers_MCX_MCXW_MCXW727C.CMC_GetSystemResetInterruptFlags (*C function*), 202
drivers_MCX_MCXW_MCXW727C.CMC_GetSystemResetStatus (*C function*), 199
drivers_MCX_MCXW_MCXW727C.CMC_GetWAKEPowerMode (*C function*), 201
drivers_MCX_MCXW_MCXW727C.CMC_GetWakeupSource (*C function*), 199
drivers_MCX_MCXW_MCXW727C.CMC_GlobalEnterLowPowerMode (*C function*), 205
drivers_MCX_MCXW_MCXW727C.CMC_LockClockModeSetting (*C function*), 198
drivers_MCX_MCXW_MCXW727C.CMC_LockPowerModeProtectionSetting (*C function*), 200
drivers_MCX_MCXW_MCXW727C.cmc_low_power_mode_t (*C type*), 210
drivers_MCX_MCXW_MCXW727C.CMC_PostExitLowPowerMode (*C function*), 204
drivers_MCX_MCXW_MCXW727C.cmc_power_domain_config_t (*C type*), 210
drivers_MCX_MCXW_MCXW727C.CMC_PowerOffSRAMAllMode (*C function*), 203
drivers_MCX_MCXW_MCXW727C.CMC_PowerOffSRAMLowPowerOnly (*C function*), 204
drivers_MCX_MCXW_MCXW727C.CMC_PowerOnSRAMAllMode (*C function*), 203
drivers_MCX_MCXW_MCXW727C.CMC_PowerOnSRAMLowPowerOnly (*C function*), 204
drivers_MCX_MCXW_MCXW727C.CMC_PreEnterLowPowerMode (*C function*), 204
drivers_MCX_MCXW_MCXW727C.cmc_reset_pin_config_t (*C type*), 210
drivers_MCX_MCXW_MCXW727C.CMC_SetClockMode (*C function*), 198
drivers_MCX_MCXW_MCXW727C.CMC_SetGlobalPowerMode (*C function*), 200
drivers_MCX_MCXW_MCXW727C.CMC_SetMAINPowerMode (*C function*), 200
drivers_MCX_MCXW_MCXW727C.CMC_SetPowerModeProtection (*C function*), 200
drivers_MCX_MCXW_MCXW727C.CMC_SetWAKEPowerMode (*C function*), 201
drivers_MCX_MCXW_MCXW727C.CMC_SRAM_BUSY_TIMEOUT (*C macro*), 205
drivers_MCX_MCXW_MCXW727C.CMC_WriteBootRomStatusReg (*C function*), 203
drivers_MCX_MCXW_MCXW727C.COUNT_TO_MSEC (*C macro*), 451
drivers_MCX_MCXW_MCXW727C.COUNT_TO_USEC (*C macro*), 451
drivers_MCX_MCXW_MCXW727C.crc_bits_t (*C type*), 212
drivers_MCX_MCXW_MCXW727C.CRC_CLOCKS (*C macro*), 194
drivers_MCX_MCXW_MCXW727C.crc_config_t (*C type*), 212
drivers_MCX_MCXW_MCXW727C.CRC_Deinit (*C function*), 212
drivers_MCX_MCXW_MCXW727C.CRC_DRIVER_USE_CRC16_CCIT_FALSE_AS_DEFAULT (*C macro*), 213
drivers_MCX_MCXW_MCXW727C.CRC_Get16bitResult (*C function*), 213
drivers_MCX_MCXW_MCXW727C.CRC_Get32bitResult (*C function*), 213

drivers_MCX_MCXW_MCXW727C.CRC_GetDefaultConfig (*C function*), 213
drivers_MCX_MCXW_MCXW727C.CRC_Init (*C function*), 212
drivers_MCX_MCXW_MCXW727C.crc_result_t (*C type*), 212
drivers_MCX_MCXW_MCXW727C.CRC_WriteData (*C function*), 213
drivers_MCX_MCXW_MCXW727C.DCACHE_CleanByRange (*C function*), 156
drivers_MCX_MCXW_MCXW727C.DCACHE_CleanInvalidateByRange (*C function*), 156
drivers_MCX_MCXW_MCXW727C.DCACHE_InvalidateByRange (*C function*), 156
drivers_MCX_MCXW_MCXW727C.DEBUG_CONSOLE_DEVICE_TYPE_FLEXCOMM (*C macro*), 448
drivers_MCX_MCXW_MCXW727C.DEBUG_CONSOLE_DEVICE_TYPE_IUART (*C macro*), 448
drivers_MCX_MCXW_MCXW727C.DEBUG_CONSOLE_DEVICE_TYPE_LPSCI (*C macro*), 448
drivers_MCX_MCXW_MCXW727C.DEBUG_CONSOLE_DEVICE_TYPE_LPUART (*C macro*), 448
drivers_MCX_MCXW_MCXW727C.DEBUG_CONSOLE_DEVICE_TYPE_MINI_USART (*C macro*), 449
drivers_MCX_MCXW_MCXW727C.DEBUG_CONSOLE_DEVICE_TYPE_NONE (*C macro*), 448
drivers_MCX_MCXW_MCXW727C.DEBUG_CONSOLE_DEVICE_TYPE_QSCI (*C macro*), 449
drivers_MCX_MCXW_MCXW727C.DEBUG_CONSOLE_DEVICE_TYPE_SWO (*C macro*), 449
drivers_MCX_MCXW_MCXW727C.DEBUG_CONSOLE_DEVICE_TYPE_UART (*C macro*), 448
drivers_MCX_MCXW_MCXW727C.DEBUG_CONSOLE_DEVICE_TYPE_USBCDC (*C macro*), 448
drivers_MCX_MCXW_MCXW727C.DEBUG_CONSOLE_DEVICE_TYPE_VUSART (*C macro*), 449
drivers_MCX_MCXW_MCXW727C.DisableGlobalIRQ (*C function*), 463
drivers_MCX_MCXW_MCXW727C.DisableIRQ (*C function*), 462
drivers_MCX_MCXW_MCXW727C.DLC_LENGTH_DECODE (*C macro*), 286
drivers_MCX_MCXW_MCXW727C.DMA_MAX_TRANSFER_COUNT (*C macro*), 560
drivers_MCX_MCXW_MCXW727C.E_RX_FIFO (*C macro*), 289
drivers_MCX_MCXW_MCXW727C.EDMA_AbortTransfer (*C function*), 225
drivers_MCX_MCXW_MCXW727C.edma_bandwidth_t (*C type*), 232
drivers_MCX_MCXW_MCXW727C.edma_callback (*C type*), 233
drivers_MCX_MCXW_MCXW727C.edma_channel_link_type_t (*C type*), 232
drivers_MCX_MCXW_MCXW727C.edma_channel_Preemption_config_t (*C type*), 233
drivers_MCX_MCXW_MCXW727C.edma_channel_protection_level_t (*C type*), 232
drivers_MCX_MCXW_MCXW727C.edma_channel_security_level_t (*C type*), 232
drivers_MCX_MCXW_MCXW727C.EDMA_ClearChannelStatusFlags (*C function*), 223
drivers_MCX_MCXW_MCXW727C.EDMA_CLOCKS (*C macro*), 194
drivers_MCX_MCXW_MCXW727C.edma_config_t (*C type*), 232
drivers_MCX_MCXW_MCXW727C.EDMA_CreateHandle (*C function*), 223
drivers_MCX_MCXW_MCXW727C.EDMA_Deinit (*C function*), 214
drivers_MCX_MCXW_MCXW727C.EDMA_DisableChannelInterrupts (*C function*), 219
drivers_MCX_MCXW_MCXW727C.EDMA_DisableChannelRequest (*C function*), 222
drivers_MCX_MCXW_MCXW727C.EDMA_EnableAllChannelLink (*C function*), 215
drivers_MCX_MCXW_MCXW727C.EDMA_EnableAsyncRequest (*C function*), 218
drivers_MCX_MCXW_MCXW727C.EDMA_EnableAutoStopRequest (*C function*), 219
drivers_MCX_MCXW_MCXW727C.EDMA_EnableChannelInterrupts (*C function*), 219
drivers_MCX_MCXW_MCXW727C.EDMA_EnableChannelMasterIDReplication (*C function*), 217
drivers_MCX_MCXW_MCXW727C.EDMA_EnableChannelRequest (*C function*), 222
drivers_MCX_MCXW_MCXW727C.EDMA_GetChannelStatusFlags (*C function*), 223
drivers_MCX_MCXW_MCXW727C.EDMA_GetChannelSystemBusInformation (*C function*), 217
drivers_MCX_MCXW_MCXW727C.EDMA_GetDefaultConfig (*C function*), 215
drivers_MCX_MCXW_MCXW727C.EDMA_GetErrorStatusFlags (*C function*), 223
drivers_MCX_MCXW_MCXW727C.EDMA_GetNextTCDAddress (*C function*), 226
drivers_MCX_MCXW_MCXW727C.EDMA_GetRemainingMajorLoopCount (*C function*), 222
drivers_MCX_MCXW_MCXW727C.EDMA_GetTransferSize (*C function*), 226
drivers_MCX_MCXW_MCXW727C.EDMA_GetUnusedTCDNumber (*C function*), 225
drivers_MCX_MCXW_MCXW727C.edma_handle_t (*C type*), 233

drivers_MCX_MCXW_MCXW727C.EDMA_HandleIRQ (*C function*), 226
drivers_MCX_MCXW_MCXW727C.EDMA_Init (*C function*), 214
drivers_MCX_MCXW_MCXW727C.EDMA_InstallTCD (*C function*), 215
drivers_MCX_MCXW_MCXW727C.EDMA_InstallTCDMemory (*C function*), 223
drivers_MCX_MCXW_MCXW727C.edma_interrupt_enable_t (*C type*), 232
drivers_MCX_MCXW_MCXW727C.edma_memorymap_callback (*C type*), 233
drivers_MCX_MCXW_MCXW727C.edma_minor_offset_config_t (*C type*), 233
drivers_MCX_MCXW_MCXW727C.edma_modulo_t (*C type*), 232
drivers_MCX_MCXW_MCXW727C.EDMA_PrepareTransfer (*C function*), 224
drivers_MCX_MCXW_MCXW727C.EDMA_PrepareTransferConfig (*C function*), 224
drivers_MCX_MCXW_MCXW727C.EDMA_ResetChannel (*C function*), 215
drivers_MCX_MCXW_MCXW727C.EDMA_SetBandWidth (*C function*), 218
drivers_MCX_MCXW_MCXW727C.EDMA_SetCallback (*C function*), 224
drivers_MCX_MCXW_MCXW727C.EDMA_SetChannelArbitrationGroup (*C function*), 216
drivers_MCX_MCXW_MCXW727C.EDMA_SetChannelLink (*C function*), 217
drivers_MCX_MCXW_MCXW727C.EDMA_SetChannelMux (*C function*), 219
drivers_MCX_MCXW_MCXW727C.EDMA_SetChannelPreemptionConfig (*C function*), 217
drivers_MCX_MCXW_MCXW727C.EDMA_SetChannelProtectionLevel (*C function*), 217
drivers_MCX_MCXW_MCXW727C.EDMA_SetChannelSecurityLevel (*C function*), 217
drivers_MCX_MCXW_MCXW727C.EDMA_SetMinorOffsetConfig (*C function*), 216
drivers_MCX_MCXW_MCXW727C.EDMA_SetModulo (*C function*), 218
drivers_MCX_MCXW_MCXW727C.EDMA_SetTransferConfig (*C function*), 216
drivers_MCX_MCXW_MCXW727C.EDMA_StartTransfer (*C function*), 225
drivers_MCX_MCXW_MCXW727C.EDMA_StopTransfer (*C function*), 225
drivers_MCX_MCXW_MCXW727C.EDMA_SubmitTransfer (*C function*), 225
drivers_MCX_MCXW_MCXW727C.edma_tcd_t (*C type*), 233
drivers_MCX_MCXW_MCXW727C.EDMA_TcdDisableInterrupts (*C function*), 222
drivers_MCX_MCXW_MCXW727C.EDMA_TcdEnableAutoStopRequest (*C function*), 221
drivers_MCX_MCXW_MCXW727C.EDMA_TcdEnableInterrupts (*C function*), 221
drivers_MCX_MCXW_MCXW727C.EDMA_TcdReset (*C function*), 219
drivers_MCX_MCXW_MCXW727C.EDMA_TcdSetBandWidth (*C function*), 221
drivers_MCX_MCXW_MCXW727C.EDMA_TcdSetChannelLink (*C function*), 220
drivers_MCX_MCXW_MCXW727C.EDMA_TcdSetMinorOffsetConfig (*C function*), 220
drivers_MCX_MCXW_MCXW727C.EDMA_TcdSetModulo (*C function*), 221
drivers_MCX_MCXW_MCXW727C.EDMA_TcdSetTransferConfig (*C function*), 220
drivers_MCX_MCXW_MCXW727C.edma_transfer_config_t (*C type*), 232
drivers_MCX_MCXW_MCXW727C.edma_transfer_size_t (*C type*), 232
drivers_MCX_MCXW_MCXW727C.edma_transfer_type_t (*C type*), 232
drivers_MCX_MCXW_MCXW727C.EDMA_TriggerChannelStart (*C function*), 222
drivers_MCX_MCXW_MCXW727C.ELEMU_LP_WakeupPathInit (*C function*), 240
drivers_MCX_MCXW_MCXW727C.ELEMU_mu_get_ownership (*C function*), 239
drivers_MCX_MCXW_MCXW727C.ELEMU_mu_get_ownership_status (*C function*), 239
drivers_MCX_MCXW_MCXW727C.ELEMU_mu_get_response (*C function*), 238
drivers_MCX_MCXW_MCXW727C.ELEMU_mu_hal_receive_data (*C function*), 237
drivers_MCX_MCXW_MCXW727C.ELEMU_mu_hal_receive_data_wait (*C function*), 237
drivers_MCX_MCXW_MCXW727C.ELEMU_mu_hal_send_data (*C function*), 237
drivers_MCX_MCXW_MCXW727C.ELEMU_mu_init (*C function*), 240
drivers_MCX_MCXW_MCXW727C.ELEMU_mu_read_data_wait (*C function*), 238
drivers_MCX_MCXW_MCXW727C.ELEMU_mu_read_message (*C function*), 238
drivers_MCX_MCXW_MCXW727C.ELEMU_mu_release_ownership (*C function*), 239
drivers_MCX_MCXW_MCXW727C.ELEMU_mu_release_ownership_force (*C function*), 240
drivers_MCX_MCXW_MCXW727C.ELEMU_mu_send_message (*C function*), 238
drivers_MCX_MCXW_MCXW727C.ELEMU_mu_wait_for_data (*C function*), 239
drivers_MCX_MCXW_MCXW727C.ELEMU_mu_wait_for_ready (*C function*), 239
drivers_MCX_MCXW_MCXW727C.elemu_ownership_status_t (*C enum*), 237
drivers_MCX_MCXW_MCXW727C.elemu_ownership_status_t.kStatus_ELEMU_Free (*C enumerator*), 237

drivers_MCX_MCXW_MCXW727C.elemu_ownership_status_t.kStatus_ELEMU_LockedByMe (*C enumerator*), 237
drivers_MCX_MCXW_MCXW727C.elemu_ownership_status_t.kStatus_ELEMU_LockedByOther (*C enumerator*), 237
drivers_MCX_MCXW_MCXW727C.elemu_ownership_status_t.kStatus_ELEMU_Unknown (*C enumerator*), 237
drivers_MCX_MCXW_MCXW727C.EnableGlobalIRQ (*C function*), 463
drivers_MCX_MCXW_MCXW727C.EnableIRQ (*C function*), 462
drivers_MCX_MCXW_MCXW727C.EnableIRQWithPriority (*C function*), 462
drivers_MCX_MCXW_MCXW727C.ENHANCED_RX_FIFO_FSCH (*C macro*), 288
drivers_MCX_MCXW_MCXW727C.EVENT_COUNT_STABLE_TIMEOUT (*C macro*), 726
drivers_MCX_MCXW_MCXW727C.EWM_CLOCKS (*C macro*), 196
drivers_MCX_MCXW_MCXW727C.ewm_config_t (*C type*), 243
drivers_MCX_MCXW_MCXW727C.EWM_Deinit (*C function*), 241
drivers_MCX_MCXW_MCXW727C.EWM_DisableInterrupts (*C function*), 241
drivers_MCX_MCXW_MCXW727C.EWM_EnableInterrupts (*C function*), 241
drivers_MCX_MCXW_MCXW727C.EWM_GetDefaultConfig (*C function*), 241
drivers_MCX_MCXW_MCXW727C.EWM_GetStatusFlags (*C function*), 241
drivers_MCX_MCXW_MCXW727C.EWM_Init (*C function*), 240
drivers_MCX_MCXW_MCXW727C.ewm_lpo_clock_source_t (*C type*), 243
drivers_MCX_MCXW_MCXW727C.EWM_Refresh (*C function*), 242
drivers_MCX_MCXW_MCXW727C.FLASH_ADDR_MASK (*C macro*), 249
drivers_MCX_MCXW_MCXW727C.flash_config_t (*C type*), 248
drivers_MCX_MCXW_MCXW727C.FLASH_Erase (*C function*), 244
drivers_MCX_MCXW_MCXW727C.FLASH_EraseAll (*C function*), 245
drivers_MCX_MCXW_MCXW727C.FLASH_GetProperty (*C function*), 246
drivers_MCX_MCXW_MCXW727C.flash_ifr_desc_t (*C type*), 248
drivers_MCX_MCXW_MCXW727C.FLASH_Init (*C function*), 244
drivers_MCX_MCXW_MCXW727C.flash_mem_desc_t (*C type*), 248
drivers_MCX_MCXW_MCXW727C.FLASH_Program (*C function*), 245
drivers_MCX_MCXW_MCXW727C.FLASH_ProgramPage (*C function*), 245
drivers_MCX_MCXW_MCXW727C.flash_property_tag_t (*C type*), 248
drivers_MCX_MCXW_MCXW727C.FLASH_Read (*C function*), 246
drivers_MCX_MCXW_MCXW727C.FLASH_VerifyEraseAll (*C function*), 246
drivers_MCX_MCXW_MCXW727C.FLASH_VerifyEraseBlock (*C function*), 246
drivers_MCX_MCXW_MCXW727C.FLASH_VerifyEraseIFRPage (*C function*), 246
drivers_MCX_MCXW_MCXW727C.FLASH_VerifyEraseIFRPhrase (*C function*), 246
drivers_MCX_MCXW_MCXW727C.FLASH_VerifyEraseIFRSector (*C function*), 246
drivers_MCX_MCXW_MCXW727C.FLASH_VerifyErasePage (*C function*), 246
drivers_MCX_MCXW_MCXW727C.FLASH_VerifyErasePhrase (*C function*), 246
drivers_MCX_MCXW_MCXW727C.FLASH_VerifyEraseSector (*C function*), 246
drivers_MCX_MCXW_MCXW727C.FLEXCAN_BusoffErrorHandleIRQ (*C function*), 270
drivers_MCX_MCXW_MCXW727C.flexcan_byte_error_syndrome_t (*C type*), 285
drivers_MCX_MCXW_MCXW727C.FLEXCAN_CalculateImprovedTimingValues (*C function*), 250
drivers_MCX_MCXW_MCXW727C.FLEXCAN_CALLBACK (*C macro*), 290
drivers_MCX_MCXW_MCXW727C.flexcan_capture_point_t (*C type*), 284
drivers_MCX_MCXW_MCXW727C.FLEXCAN_ClearHigh64MbStatusFlags (*C function*), 258
drivers_MCX_MCXW_MCXW727C.FLEXCAN_ClearMbStatusFlags (*C function*), 258
drivers_MCX_MCXW_MCXW727C.FLEXCAN_ClearStatusFlags (*C function*), 257
drivers_MCX_MCXW_MCXW727C.flexcan_clock_source_t (*C type*), 284
drivers_MCX_MCXW_MCXW727C.FLEXCAN_CLOCKS (*C macro*), 195
drivers_MCX_MCXW_MCXW727C.flexcan_config_t (*C type*), 285
drivers_MCX_MCXW_MCXW727C.FLEXCAN_Deinit (*C function*), 252
drivers_MCX_MCXW_MCXW727C.FLEXCAN_DisableHigh64MbInterrupts (*C function*), 260
drivers_MCX_MCXW_MCXW727C.FLEXCAN_DisableInterrupts (*C function*), 259
drivers_MCX_MCXW_MCXW727C.FLEXCAN_DisableMbInterrupts (*C function*), 260
drivers_MCX_MCXW_MCXW727C.flexcan_edma_handle_t (*C type*), 307
drivers_MCX_MCXW_MCXW727C.flexcan_edma_transfer_callback_t (*C type*), 307

drivers_MCX_MCXW_MCXW727C.flexcan_efifo_dma_per_read_length_t (*C type*), 284
drivers_MCX_MCXW_MCXW727C.FLEXCAN_EFIFO_INT_MASK (*C macro*), 289
drivers_MCX_MCXW_MCXW727C.FLEXCAN_EFIFO_INT_UNMASK (*C macro*), 289
drivers_MCX_MCXW_MCXW727C.FLEXCAN_EFIFO_STATUS_MASK (*C macro*), 289
drivers_MCX_MCXW_MCXW727C.FLEXCAN_EFIFO_STATUS_UNMASK (*C macro*), 289
drivers_MCX_MCXW_MCXW727C.FLEXCAN_Enable (*C function*), 260
drivers_MCX_MCXW_MCXW727C.FLEXCAN_EnableHigh64MbInterrupts (*C function*), 259
drivers_MCX_MCXW_MCXW727C.FLEXCAN_EnableInterrupts (*C function*), 259
drivers_MCX_MCXW_MCXW727C.FLEXCAN_EnableMbInterrupts (*C function*), 259
drivers_MCX_MCXW_MCXW727C.FLEXCAN_EnableRxFifoDMA (*C function*), 260
drivers_MCX_MCXW_MCXW727C.flexcan_endianness_t (*C type*), 284
drivers_MCX_MCXW_MCXW727C.flexcan_enhanced_rx_fifo_config_t (*C type*), 286
drivers_MCX_MCXW_MCXW727C.FLEXCAN_ENHANCED_RX_FIFO_EXT_FILTER_WITH_RANGE_HIGH (*C macro*), 289
drivers_MCX_MCXW_MCXW727C.FLEXCAN_ENHANCED_RX_FIFO_EXT_FILTER_WITH_RANGE_LOW (*C macro*), 289
drivers_MCX_MCXW_MCXW727C.flexcan_enhanced_rx_fifo_ext_id_filter_t (*C type*), 286
drivers_MCX_MCXW_MCXW727C.FLEXCAN_ENHANCED_RX_FIFO_EXT_MASK_AND_FILTER_HIGH (*C macro*), 289
drivers_MCX_MCXW_MCXW727C.FLEXCAN_ENHANCED_RX_FIFO_EXT_MASK_AND_FILTER_LOW (*C macro*), 288
drivers_MCX_MCXW_MCXW727C.FLEXCAN_ENHANCED_RX_FIFO_EXT_TWO_FILTERS_HIGH (*C macro*), 289
drivers_MCX_MCXW_MCXW727C.FLEXCAN_ENHANCED_RX_FIFO_EXT_TWO_FILTERS_LOW (*C macro*), 289
drivers_MCX_MCXW_MCXW727C.FLEXCAN_ENHANCED_RX_FIFO_INT_FLAG (*C macro*), 289
drivers_MCX_MCXW_MCXW727C.FLEXCAN_ENHANCED_RX_FIFO_STD_FILTER_WITH_RANGE (*C macro*), 288
drivers_MCX_MCXW_MCXW727C.flexcan_enhanced_rx_fifo_std_id_filter_t (*C type*), 286
drivers_MCX_MCXW_MCXW727C.FLEXCAN_ENHANCED_RX_FIFO_STD_MASK_AND_FILTER (*C macro*), 288
drivers_MCX_MCXW_MCXW727C.FLEXCAN_ENHANCED_RX_FIFO_STD_TWO_FILTERS (*C macro*), 288
drivers_MCX_MCXW_MCXW727C.FLEXCAN_EnhancedRxFifoHandleIRQ (*C function*), 270
drivers_MCX_MCXW_MCXW727C.FLEXCAN_EnterFreezeMode (*C function*), 250
drivers_MCX_MCXW_MCXW727C.FLEXCAN_ERROR_AND_STATUS_INT_FLAG (*C macro*), 289
drivers_MCX_MCXW_MCXW727C.FLEXCAN_ExitFreezeMode (*C function*), 250
drivers_MCX_MCXW_MCXW727C.flexcan_fd_frame_t (*C type*), 285
drivers_MCX_MCXW_MCXW727C.FLEXCAN_FDCalculateImprovedTimingValues (*C function*), 251
drivers_MCX_MCXW_MCXW727C.FLEXCAN_FDInit (*C function*), 252
drivers_MCX_MCXW_MCXW727C.flexcan_fifo_transfer_t (*C type*), 286
drivers_MCX_MCXW_MCXW727C.flexcan_frame_format_t (*C type*), 284
drivers_MCX_MCXW_MCXW727C.flexcan_frame_t (*C type*), 285
drivers_MCX_MCXW_MCXW727C.flexcan_frame_type_t (*C type*), 284
drivers_MCX_MCXW_MCXW727C.FLEXCAN_GetBusErrCount (*C function*), 257
drivers_MCX_MCXW_MCXW727C.FLEXCAN_GetDefaultConfig (*C function*), 252
drivers_MCX_MCXW_MCXW727C.FLEXCAN_GetEnhancedFifoDataCount (*C function*), 259
drivers_MCX_MCXW_MCXW727C.FLEXCAN_GetFDMailboxOffset (*C function*), 250
drivers_MCX_MCXW_MCXW727C.FLEXCAN_GetHigh64MbStatusFlags (*C function*), 258
drivers_MCX_MCXW_MCXW727C.FLEXCAN_GetHighResolutionTimeStamp (*C function*), 269
drivers_MCX_MCXW_MCXW727C.FLEXCAN_GetInstance (*C function*), 250
drivers_MCX_MCXW_MCXW727C.FLEXCAN_GetMbStatusFlags (*C function*), 258
drivers_MCX_MCXW_MCXW727C.FLEXCAN_GetMemoryErrorReportStatus (*C function*), 258
drivers_MCX_MCXW_MCXW727C.FLEXCAN_GetPNMatchCount (*C function*), 259
drivers_MCX_MCXW_MCXW727C.FLEXCAN_GetRxFifoHeadAddr (*C function*), 260

drivers_MCX_MCXW_MCXW727C.FLEXCAN_GetStatusFlags (*C function*), 257
drivers_MCX_MCXW_MCXW727C.FLEXCAN_GetTimeStamp (*C function*), 269
drivers_MCX_MCXW_MCXW727C.flexcan_handle_t (*C type*), 286
drivers_MCX_MCXW_MCXW727C.FLEXCAN_ID_EXT (*C macro*), 287
drivers_MCX_MCXW_MCXW727C.FLEXCAN_ID_STD (*C macro*), 286
drivers_MCX_MCXW_MCXW727C.FLEXCAN_Init (*C function*), 251
drivers_MCX_MCXW_MCXW727C.FLEXCAN_IsInstanceHasFDMode (*C function*), 249
drivers_MCX_MCXW_MCXW727C.FLEXCAN_MB_BUSY_TIMEOUT (*C macro*), 286
drivers_MCX_MCXW_MCXW727C.flexcan_mb_size_t (*C type*), 284
drivers_MCX_MCXW_MCXW727C.flexcan_MB_timestamp_base_t (*C type*), 284
drivers_MCX_MCXW_MCXW727C.flexcan_mb_transfer_t (*C type*), 286
drivers_MCX_MCXW_MCXW727C.FLEXCAN_MbHandleIRQ (*C function*), 270
drivers_MCX_MCXW_MCXW727C.FLEXCAN_MECR_INT_MASK (*C macro*), 289
drivers_MCX_MCXW_MCXW727C.FLEXCAN_MECR_INT_UNMASK (*C macro*), 289
drivers_MCX_MCXW_MCXW727C.FLEXCAN_MECR_STATUS_MASK (*C macro*), 289
drivers_MCX_MCXW_MCXW727C.FLEXCAN_MECR_STATUS_UNMASK (*C macro*), 289
drivers_MCX_MCXW_MCXW727C.flexcan_memory_access_type_t (*C type*), 285
drivers_MCX_MCXW_MCXW727C.FLEXCAN_MEMORY_ERROR_INT_FLAG (*C macro*), 289
drivers_MCX_MCXW_MCXW727C.flexcan_memory_error_report_status_t (*C type*), 285
drivers_MCX_MCXW_MCXW727C.flexcan_memory_error_type_t (*C type*), 285
drivers_MCX_MCXW_MCXW727C.FLEXCAN_MemoryErrorHandleIRQ (*C function*), 271
drivers_MCX_MCXW_MCXW727C.FLEXCAN_MODULE_TIMEOUT (*C macro*), 286
drivers_MCX_MCXW_MCXW727C.flexcan_pn_config_t (*C type*), 286
drivers_MCX_MCXW_MCXW727C.FLEXCAN_PN_EXT_MASK (*C macro*), 289
drivers_MCX_MCXW_MCXW727C.FLEXCAN_PN_INT_MASK (*C macro*), 289
drivers_MCX_MCXW_MCXW727C.FLEXCAN_PN_INT_UNMASK (*C macro*), 289
drivers_MCX_MCXW_MCXW727C.flexcan_pn_match_mode_t (*C type*), 286
drivers_MCX_MCXW_MCXW727C.flexcan_pn_match_source_t (*C type*), 285
drivers_MCX_MCXW_MCXW727C.FLEXCAN_PN_STATUS_MASK (*C macro*), 289
drivers_MCX_MCXW_MCXW727C.FLEXCAN_PN_STATUS_UNMASK (*C macro*), 289
drivers_MCX_MCXW_MCXW727C.FLEXCAN_PN_STD_MASK (*C macro*), 289
drivers_MCX_MCXW_MCXW727C.FLEXCAN_PNWAKE_UP_FLAG (*C macro*), 289
drivers_MCX_MCXW_MCXW727C.FLEXCAN_PNWakeUpHandleIRQ (*C function*), 270
drivers_MCX_MCXW_MCXW727C.FLEXCAN_POLLING_TIMEOUT (*C macro*), 286
drivers_MCX_MCXW_MCXW727C.FLEXCAN_PrepareTransfConfiguration (*C function*), 305
drivers_MCX_MCXW_MCXW727C.FLEXCAN_ReadEnhancedRxFifo (*C function*), 262
drivers_MCX_MCXW_MCXW727C.FLEXCAN_ReadFDRxMb (*C function*), 261
drivers_MCX_MCXW_MCXW727C.FLEXCAN_ReadPNWakeUpMB (*C function*), 262
drivers_MCX_MCXW_MCXW727C.FLEXCAN_ReadRxFifo (*C function*), 262
drivers_MCX_MCXW_MCXW727C.FLEXCAN_ReadRxMb (*C function*), 261
drivers_MCX_MCXW_MCXW727C.flexcan_rx_fifo_config_t (*C type*), 286
drivers_MCX_MCXW_MCXW727C.FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_A (*C macro*), 288
drivers_MCX_MCXW_MCXW727C.FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_B_HIGH (*C macro*), 288
drivers_MCX_MCXW_MCXW727C.FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_B_LOW (*C macro*), 288
drivers_MCX_MCXW_MCXW727C.FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_HIGH (*C macro*), 288
drivers_MCX_MCXW_MCXW727C.FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_LOW (*C macro*), 288
drivers_MCX_MCXW_MCXW727C.FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_MID_HIGH (*C macro*), 288
drivers_MCX_MCXW_MCXW727C.FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_MID_LOW (*C macro*), 288
drivers_MCX_MCXW_MCXW727C.FLEXCAN_RX_FIFO_EXT_MASK_TYPE_A (*C macro*), 287

drivers_MCX_MCXW_MCXW727C.FLEXCAN_RX_FIFO_EXT_MASK_TYPE_B_HIGH (*C macro*), 287
drivers_MCX_MCXW_MCXW727C.FLEXCAN_RX_FIFO_EXT_MASK_TYPE_B_LOW (*C macro*), 287
drivers_MCX_MCXW_MCXW727C.FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_HIGH (*C macro*), 287
drivers_MCX_MCXW_MCXW727C.FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_LOW (*C macro*), 287
drivers_MCX_MCXW_MCXW727C.FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_MID_HIGH (*C macro*), 287
drivers_MCX_MCXW_MCXW727C.FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_MID_LOW (*C macro*), 287
drivers_MCX_MCXW_MCXW727C.flexcan_rx_fifo_filter_type_t (*C type*), 284
drivers_MCX_MCXW_MCXW727C.flexcan_rx_fifo_priority_t (*C type*), 284
drivers_MCX_MCXW_MCXW727C.FLEXCAN_RX_FIFO_STD_FILTER_TYPE_A (*C macro*), 287
drivers_MCX_MCXW_MCXW727C.FLEXCAN_RX_FIFO_STD_FILTER_TYPE_B_HIGH (*C macro*), 287
drivers_MCX_MCXW_MCXW727C.FLEXCAN_RX_FIFO_STD_FILTER_TYPE_B_LOW (*C macro*), 288
drivers_MCX_MCXW_MCXW727C.FLEXCAN_RX_FIFO_STD_FILTER_TYPE_C_HIGH (*C macro*), 288
drivers_MCX_MCXW_MCXW727C.FLEXCAN_RX_FIFO_STD_FILTER_TYPE_C_LOW (*C macro*), 288
drivers_MCX_MCXW_MCXW727C.FLEXCAN_RX_FIFO_STD_FILTER_TYPE_C_MID_HIGH (*C macro*), 288
drivers_MCX_MCXW_MCXW727C.FLEXCAN_RX_FIFO_STD_FILTER_TYPE_C_MID_LOW (*C macro*), 288
drivers_MCX_MCXW_MCXW727C.FLEXCAN_RX_FIFO_STD_MASK_TYPE_A (*C macro*), 287
drivers_MCX_MCXW_MCXW727C.FLEXCAN_RX_FIFO_STD_MASK_TYPE_B_HIGH (*C macro*), 287
drivers_MCX_MCXW_MCXW727C.FLEXCAN_RX_FIFO_STD_MASK_TYPE_B_LOW (*C macro*), 287
drivers_MCX_MCXW_MCXW727C.FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_HIGH (*C macro*), 287
drivers_MCX_MCXW_MCXW727C.FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_LOW (*C macro*), 287
drivers_MCX_MCXW_MCXW727C.FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_MID_HIGH (*C macro*), 287
drivers_MCX_MCXW_MCXW727C.FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_MID_LOW (*C macro*), 287
drivers_MCX_MCXW_MCXW727C.flexcan_rx_mb_config_t (*C type*), 285
drivers_MCX_MCXW_MCXW727C.FLEXCAN_RX_MB_EXT_MASK (*C macro*), 287
drivers_MCX_MCXW_MCXW727C.FLEXCAN_RX_MB_STD_MASK (*C macro*), 287
drivers_MCX_MCXW_MCXW727C.FLEXCAN_SetBitRate (*C function*), 253
drivers_MCX_MCXW_MCXW727C.FLEXCAN_SetEnhancedRxFifoConfig (*C function*), 257
drivers_MCX_MCXW_MCXW727C.FLEXCAN_SetFDBitRate (*C function*), 253
drivers_MCX_MCXW_MCXW727C.FLEXCAN_SetFDMbID (*C function*), 256
drivers_MCX_MCXW_MCXW727C.FLEXCAN_SetFDRxMbConfig (*C function*), 255
drivers_MCX_MCXW_MCXW727C.FLEXCAN_SetFDTimingConfig (*C function*), 253
drivers_MCX_MCXW_MCXW727C.FLEXCAN_SetFDTxMbConfig (*C function*), 255
drivers_MCX_MCXW_MCXW727C.FLEXCAN_SetMbID (*C function*), 255
drivers_MCX_MCXW_MCXW727C.FLEXCAN_SetPNConfig (*C function*), 257
drivers_MCX_MCXW_MCXW727C.FLEXCAN_SetRemoteResponseMbConfig (*C function*), 256
drivers_MCX_MCXW_MCXW727C.FLEXCAN_SetRxFifoConfig (*C function*), 256
drivers_MCX_MCXW_MCXW727C.FLEXCAN_SetRxFifoGlobalMask (*C function*), 254
drivers_MCX_MCXW_MCXW727C.FLEXCAN_SetRxIndividualMask (*C function*), 254

drivers_MCX_MCXW_MCXW727C.FLEXCAN_SetRxMbConfig (*C function*), 255
drivers_MCX_MCXW_MCXW727C.FLEXCAN_SetRxMbGlobalMask (*C function*), 254
drivers_MCX_MCXW_MCXW727C.FLEXCAN_SetTimingConfig (*C function*), 253
drivers_MCX_MCXW_MCXW727C.FLEXCAN_SetTxMbConfig (*C function*), 254
drivers_MCX_MCXW_MCXW727C.FLEXCAN_StartTransferDatafromRxFIFO (*C function*), 305
drivers_MCX_MCXW_MCXW727C.flexcan_timing_config_t (*C type*), 285
drivers_MCX_MCXW_MCXW727C.flexcan_transfer_callback_t (*C type*), 286
drivers_MCX_MCXW_MCXW727C.FLEXCAN_TransferAbortReceive (*C function*), 269
drivers_MCX_MCXW_MCXW727C.FLEXCAN_TransferAbortReceiveEnhancedFifo (*C function*), 270
drivers_MCX_MCXW_MCXW727C.FLEXCAN_TransferAbortReceiveFifo (*C function*), 270
drivers_MCX_MCXW_MCXW727C.FLEXCAN_TransferAbortReceiveFifoEDMA (*C function*), 306
drivers_MCX_MCXW_MCXW727C.FLEXCAN_TransferAbortSend (*C function*), 269
drivers_MCX_MCXW_MCXW727C.FLEXCAN_TransferCreateHandle (*C function*), 266
drivers_MCX_MCXW_MCXW727C.FLEXCAN_TransferCreateHandleEDMA (*C function*), 305
drivers_MCX_MCXW_MCXW727C.FLEXCAN_TransferFDAbortReceive (*C function*), 264
drivers_MCX_MCXW_MCXW727C.FLEXCAN_TransferFDAbortSend (*C function*), 264
drivers_MCX_MCXW_MCXW727C.FLEXCAN_TransferFDReceiveBlocking (*C function*), 263
drivers_MCX_MCXW_MCXW727C.FLEXCAN_TransferFDReceiveNonBlocking (*C function*), 264
drivers_MCX_MCXW_MCXW727C.FLEXCAN_TransferFDSendBlocking (*C function*), 263
drivers_MCX_MCXW_MCXW727C.FLEXCAN_TransferFDSendNonBlocking (*C function*), 263
drivers_MCX_MCXW_MCXW727C.FLEXCAN_TransferGetReceiveEnhancedFifoCount (*C function*), 269
drivers_MCX_MCXW_MCXW727C.FLEXCAN_TransferGetReceiveEnhancedFifoCountEMDA (*C function*), 306
drivers_MCX_MCXW_MCXW727C.FLEXCAN_TransferGetReceiveFifoCount (*C function*), 268
drivers_MCX_MCXW_MCXW727C.FLEXCAN_TransferGetReceiveFifoCountEMDA (*C function*), 306
drivers_MCX_MCXW_MCXW727C.FLEXCAN_TransferHandleIRQ (*C function*), 270
drivers_MCX_MCXW_MCXW727C.FLEXCAN_TransferReceiveBlocking (*C function*), 265
drivers_MCX_MCXW_MCXW727C.FLEXCAN_TransferReceiveEnhancedFifoBlocking (*C function*), 266
drivers_MCX_MCXW_MCXW727C.FLEXCAN_TransferReceiveEnhancedFifoEDMA (*C function*), 306
drivers_MCX_MCXW_MCXW727C.FLEXCAN_TransferReceiveEnhancedFifoNonBlocking (*C function*), 268
drivers_MCX_MCXW_MCXW727C.FLEXCAN_TransferReceiveFifoBlocking (*C function*), 265
drivers_MCX_MCXW_MCXW727C.FLEXCAN_TransferReceiveFifoEDMA (*C function*), 305
drivers_MCX_MCXW_MCXW727C.FLEXCAN_TransferReceiveFifoNonBlocking (*C function*), 268
drivers_MCX_MCXW_MCXW727C.FLEXCAN_TransferReceiveNonBlocking (*C function*), 266
drivers_MCX_MCXW_MCXW727C.FLEXCAN_TransferRemoteRequestNonBlocking (*C function*), 267
drivers_MCX_MCXW_MCXW727C.FLEXCAN_TransferRemoteResponseNonBlocking (*C function*), 267
drivers_MCX_MCXW_MCXW727C.FLEXCAN_TransferSendBlocking (*C function*), 265
drivers_MCX_MCXW_MCXW727C.FLEXCAN_TransferSendNonBlocking (*C function*), 266
drivers_MCX_MCXW_MCXW727C.FLEXCAN_WAIT_TIMEOUT (*C macro*), 286
drivers_MCX_MCXW_MCXW727C.FLEXCAN_WAKE_UP_FLAG (*C macro*), 289
drivers_MCX_MCXW_MCXW727C.flexcan_wake_up_source_t (*C type*), 284
drivers_MCX_MCXW_MCXW727C.FLEXCAN_WriteFDTxMb (*C function*), 261
drivers_MCX_MCXW_MCXW727C.FLEXCAN_WriteTxMb (*C function*), 260
drivers_MCX_MCXW_MCXW727C.FLEXIO_ClearPortOutput (*C function*), 313
drivers_MCX_MCXW_MCXW727C.FLEXIO_ClearPortStatus (*C function*), 315
drivers_MCX_MCXW_MCXW727C.FLEXIO_ClearShifterErrorFlags (*C function*), 312
drivers_MCX_MCXW_MCXW727C.FLEXIO_ClearShifterStatusFlags (*C function*), 312
drivers_MCX_MCXW_MCXW727C.FLEXIO_ClearTimerStatusFlags (*C function*), 312
drivers_MCX_MCXW_MCXW727C.FLEXIO_CLOCKS (*C macro*), 195
drivers_MCX_MCXW_MCXW727C.flexio_config_t (*C type*), 323

drivers_MCX_MCXW_MCXW727C.FLEXIO_ConfigPinOverride (*C function*), 315
drivers_MCX_MCXW_MCXW727C.FLEXIO_Deinit (*C function*), 308
drivers_MCX_MCXW_MCXW727C.FLEXIO_DisableShifterErrorInterrupts (*C function*), 311
drivers_MCX_MCXW_MCXW727C.FLEXIO_DisableShifterStatusInterrupts (*C function*), 310
drivers_MCX_MCXW_MCXW727C.FLEXIO_DisableTimerStatusInterrupts (*C function*), 311
drivers_MCX_MCXW_MCXW727C.FLEXIO_Enable (*C function*), 308
drivers_MCX_MCXW_MCXW727C.FLEXIO_EnablePinOutput (*C function*), 314
drivers_MCX_MCXW_MCXW727C.FLEXIO_EnableShifterErrorInterrupts (*C function*), 311
drivers_MCX_MCXW_MCXW727C.FLEXIO_EnableShifterStatusDMA (*C function*), 313
drivers_MCX_MCXW_MCXW727C.FLEXIO_EnableShifterStatusInterrupts (*C function*), 310
drivers_MCX_MCXW_MCXW727C.FLEXIO_EnableTimerStatusInterrupts (*C function*), 311
drivers_MCX_MCXW_MCXW727C.FLEXIO_GetDefaultConfig (*C function*), 308
drivers_MCX_MCXW_MCXW727C.FLEXIO_GetInstance (*C function*), 308
drivers_MCX_MCXW_MCXW727C.FLEXIO_GetPinOverride (*C function*), 315
drivers_MCX_MCXW_MCXW727C.FLEXIO_GetPinStatus (*C function*), 314
drivers_MCX_MCXW_MCXW727C.FLEXIO_GetShifterBufferAddress (*C function*), 313
drivers_MCX_MCXW_MCXW727C.FLEXIO_GetShifterErrorFlags (*C function*), 312
drivers_MCX_MCXW_MCXW727C.FLEXIO_GetShifterState (*C function*), 309
drivers_MCX_MCXW_MCXW727C.FLEXIO_GetShifterStatusFlags (*C function*), 312
drivers_MCX_MCXW_MCXW727C.FLEXIO_GetTimerStatusFlags (*C function*), 312
drivers_MCX_MCXW_MCXW727C.flexio_gpio_config_t (*C type*), 324
drivers_MCX_MCXW_MCXW727C.flexio_gpio_direction_t (*C type*), 324
drivers_MCX_MCXW_MCXW727C.FLEXIO_I2C_CheckForBusyBus (*C function*), 334
drivers_MCX_MCXW_MCXW727C.flexio_i2c_direction_t (*C type*), 341
drivers_MCX_MCXW_MCXW727C.flexio_i2c_master_config_t (*C type*), 341
drivers_MCX_MCXW_MCXW727C.flexio_i2c_master_handle_t (*C type*), 341
drivers_MCX_MCXW_MCXW727C.flexio_i2c_master_transfer_callback_t (*C type*), 341
drivers_MCX_MCXW_MCXW727C.flexio_i2c_master_transfer_t (*C type*), 341
drivers_MCX_MCXW_MCXW727C.FLEXIO_I2C_MasterAbortStop (*C function*), 336
drivers_MCX_MCXW_MCXW727C.FLEXIO_I2C_MasterClearStatusFlags (*C function*), 335
drivers_MCX_MCXW_MCXW727C.FLEXIO_I2C_MasterDeinit (*C function*), 335
drivers_MCX_MCXW_MCXW727C.FLEXIO_I2C_MasterDisableInterrupts (*C function*), 336
drivers_MCX_MCXW_MCXW727C.FLEXIO_I2C_MasterEnable (*C function*), 335
drivers_MCX_MCXW_MCXW727C.FLEXIO_I2C_MasterEnableAck (*C function*), 336
drivers_MCX_MCXW_MCXW727C.FLEXIO_I2C_MasterEnableInterrupts (*C function*), 335
drivers_MCX_MCXW_MCXW727C.FLEXIO_I2C_MasterGetDefaultConfig (*C function*), 335
drivers_MCX_MCXW_MCXW727C.FLEXIO_I2C_MasterGetStatusFlags (*C function*), 335
drivers_MCX_MCXW_MCXW727C.FLEXIO_I2C_MasterInit (*C function*), 334
drivers_MCX_MCXW_MCXW727C.FLEXIO_I2C_MasterReadBlocking (*C function*), 338
drivers_MCX_MCXW_MCXW727C.FLEXIO_I2C_MasterReadByte (*C function*), 337
drivers_MCX_MCXW_MCXW727C.FLEXIO_I2C_MasterRepeatedStart (*C function*), 336
drivers_MCX_MCXW_MCXW727C.FLEXIO_I2C_MasterSetBaudRate (*C function*), 336
drivers_MCX_MCXW_MCXW727C.FLEXIO_I2C_MasterSetTransferCount (*C function*), 337
drivers_MCX_MCXW_MCXW727C.FLEXIO_I2C_MasterStart (*C function*), 336
drivers_MCX_MCXW_MCXW727C.FLEXIO_I2C_MasterStop (*C function*), 336
drivers_MCX_MCXW_MCXW727C.FLEXIO_I2C_MasterTransferAbort (*C function*), 339
drivers_MCX_MCXW_MCXW727C.FLEXIO_I2C_MasterTransferBlocking (*C function*), 338
drivers_MCX_MCXW_MCXW727C.FLEXIO_I2C_MasterTransferCreateHandle (*C function*), 338
drivers_MCX_MCXW_MCXW727C.FLEXIO_I2C_MasterTransferGetCount (*C function*), 339
drivers_MCX_MCXW_MCXW727C.FLEXIO_I2C_MasterTransferHandleIRQ (*C function*), 340
drivers_MCX_MCXW_MCXW727C.FLEXIO_I2C_MasterTransferNonBlocking (*C function*), 339
drivers_MCX_MCXW_MCXW727C.FLEXIO_I2C_MasterWriteBlocking (*C function*), 337
drivers_MCX_MCXW_MCXW727C.FLEXIO_I2C_MasterWriteByte (*C function*), 337
drivers_MCX_MCXW_MCXW727C.FLEXIO_I2C_Type (*C type*), 341
drivers_MCX_MCXW_MCXW727C.flexio_i2s_callback_t (*C type*), 352
drivers_MCX_MCXW_MCXW727C.flexio_i2s_config_t (*C type*), 352
drivers_MCX_MCXW_MCXW727C.FLEXIO_I2S_Deinit (*C function*), 344
drivers_MCX_MCXW_MCXW727C.FLEXIO_I2S_DisableInterrupts (*C function*), 345

drivers_MCX_MCXW_MCXW727C.FLEXIO_I2S_Enable (*C function*), 344
drivers_MCX_MCXW_MCXW727C.FLEXIO_I2S_EnableInterrupts (*C function*), 344
drivers_MCX_MCXW_MCXW727C.flexio_i2s_format_t (*C type*), 352
drivers_MCX_MCXW_MCXW727C.FLEXIO_I2S_GetDefaultConfig (*C function*), 344
drivers_MCX_MCXW_MCXW727C.FLEXIO_I2S_GetStatusFlags (*C function*), 344
drivers_MCX_MCXW_MCXW727C.flexio_i2s_handle_t (*C type*), 352
drivers_MCX_MCXW_MCXW727C.FLEXIO_I2S_Init (*C function*), 343
drivers_MCX_MCXW_MCXW727C.flexio_i2s_master_slave_t (*C type*), 352
drivers_MCX_MCXW_MCXW727C.FLEXIO_I2S_MasterSetFormat (*C function*), 345
drivers_MCX_MCXW_MCXW727C.FLEXIO_I2S_ReadBlocking (*C function*), 346
drivers_MCX_MCXW_MCXW727C.FLEXIO_I2S_ReadData (*C function*), 347
drivers_MCX_MCXW_MCXW727C.FLEXIO_I2S_RxEnableDMA (*C function*), 345
drivers_MCX_MCXW_MCXW727C.FLEXIO_I2S_RxGetDataRegisterAddress (*C function*), 345
drivers_MCX_MCXW_MCXW727C.flexio_i2s_sample_rate_t (*C type*), 352
drivers_MCX_MCXW_MCXW727C.FLEXIO_I2S_SlaveSetFormat (*C function*), 345
drivers_MCX_MCXW_MCXW727C.flexio_i2s_transfer_t (*C type*), 352
drivers_MCX_MCXW_MCXW727C.FLEXIO_I2S_TransferAbortReceive (*C function*), 349
drivers_MCX_MCXW_MCXW727C.FLEXIO_I2S_TransferAbortSend (*C function*), 348
drivers_MCX_MCXW_MCXW727C.FLEXIO_I2S_TransferGetReceiveCount (*C function*), 349
drivers_MCX_MCXW_MCXW727C.FLEXIO_I2S_TransferGetSendCount (*C function*), 349
drivers_MCX_MCXW_MCXW727C.FLEXIO_I2S_TransferReceiveNonBlocking (*C function*), 348
drivers_MCX_MCXW_MCXW727C.FLEXIO_I2S_TransferRxCreateHandle (*C function*), 347
drivers_MCX_MCXW_MCXW727C.FLEXIO_I2S_TransferRxHandleIRQ (*C function*), 350
drivers_MCX_MCXW_MCXW727C.FLEXIO_I2S_TransferSendNonBlocking (*C function*), 347
drivers_MCX_MCXW_MCXW727C.FLEXIO_I2S_TransferSetFormat (*C function*), 347
drivers_MCX_MCXW_MCXW727C.FLEXIO_I2S_TransferTxCreateHandle (*C function*), 347
drivers_MCX_MCXW_MCXW727C.FLEXIO_I2S_TransferTxHandleIRQ (*C function*), 349
drivers_MCX_MCXW_MCXW727C.FLEXIO_I2S_TxEnableDMA (*C function*), 345
drivers_MCX_MCXW_MCXW727C.FLEXIO_I2S_TxGetDataRegisterAddress (*C function*), 345
drivers_MCX_MCXW_MCXW727C.FLEXIO_I2S_Type (*C type*), 352
drivers_MCX_MCXW_MCXW727C.flexio_i2s_word_width_t (*C type*), 352
drivers_MCX_MCXW_MCXW727C.FLEXIO_I2S_WriteBlocking (*C function*), 346
drivers_MCX_MCXW_MCXW727C.FLEXIO_I2S_WriteData (*C function*), 346
drivers_MCX_MCXW_MCXW727C.FLEXIO_I2S_XFER_QUEUE_SIZE (*C macro*), 352
drivers_MCX_MCXW_MCXW727C.FLEXIO_Init (*C function*), 308
drivers_MCX_MCXW_MCXW727C.flexio_isr_t (*C type*), 324
drivers_MCX_MCXW_MCXW727C.flexio_pin_config_t (*C type*), 322
drivers_MCX_MCXW_MCXW727C.flexio_pin_input_config_t (*C type*), 324
drivers_MCX_MCXW_MCXW727C.flexio_pin_polarity_t (*C type*), 322
drivers_MCX_MCXW_MCXW727C.FLEXIO_PinRead (*C function*), 314
drivers_MCX_MCXW_MCXW727C.FLEXIO_PinWrite (*C function*), 314
drivers_MCX_MCXW_MCXW727C.FLEXIO_ReadPinInput (*C function*), 309
drivers_MCX_MCXW_MCXW727C.FLEXIO_RegisterHandleIRQ (*C function*), 313
drivers_MCX_MCXW_MCXW727C.FLEXIO_Reset (*C function*), 308
drivers_MCX_MCXW_MCXW727C.FLEXIO_SetClockMode (*C function*), 310
drivers_MCX_MCXW_MCXW727C.FLEXIO_SetPinConfig (*C function*), 324
drivers_MCX_MCXW_MCXW727C.FLEXIO_SetPinLevel (*C function*), 315
drivers_MCX_MCXW_MCXW727C.FLEXIO_SetPortOutput (*C function*), 314
drivers_MCX_MCXW_MCXW727C.FLEXIO_SetShifterConfig (*C function*), 309
drivers_MCX_MCXW_MCXW727C.FLEXIO_SetTimerConfig (*C function*), 309
drivers_MCX_MCXW_MCXW727C.flexio_shifter_buffer_type_t (*C type*), 323
drivers_MCX_MCXW_MCXW727C.flexio_shifter_config_t (*C type*), 324
drivers_MCX_MCXW_MCXW727C.flexio_shifter_input_source_t (*C type*), 323
drivers_MCX_MCXW_MCXW727C.flexio_shifter_mode_t (*C type*), 323
drivers_MCX_MCXW_MCXW727C.flexio_shifter_start_bit_t (*C type*), 323
drivers_MCX_MCXW_MCXW727C.flexio_shifter_stop_bit_t (*C type*), 323
drivers_MCX_MCXW_MCXW727C.flexio_shifter_timer_polarity_t (*C type*), 323
drivers_MCX_MCXW_MCXW727C.FLEXIO_SPI_ClearStatusFlags (*C function*), 357

drivers_MCX_MCXW_MCXW727C.flexio_spi_clock_phase_t (*C type*), 366
drivers_MCX_MCXW_MCXW727C.flexio_spi_data_bitcount_mode_t (*C type*), 366
drivers_MCX_MCXW_MCXW727C.FLEXIO_SPI_DisableInterrupts (*C function*), 358
drivers_MCX_MCXW_MCXW727C.FLEXIO_SPI_DUMMYDATA (*C macro*), 366
drivers_MCX_MCXW_MCXW727C.FLEXIO_SPI_Enable (*C function*), 359
drivers_MCX_MCXW_MCXW727C.FLEXIO_SPI_EnableDMA (*C function*), 358
drivers_MCX_MCXW_MCXW727C.FLEXIO_SPI_EnableInterrupts (*C function*), 358
drivers_MCX_MCXW_MCXW727C.FLEXIO_SPI_FlushShifters (*C function*), 361
drivers_MCX_MCXW_MCXW727C.FLEXIO_SPI_GetRxDataRegisterAddress (*C function*), 359
drivers_MCX_MCXW_MCXW727C.FLEXIO_SPI_GetStatusFlags (*C function*), 357
drivers_MCX_MCXW_MCXW727C.FLEXIO_SPI_GetTxDataRegisterAddress (*C function*), 358
drivers_MCX_MCXW_MCXW727C.flexio_spi_master_config_t (*C type*), 366
drivers_MCX_MCXW_MCXW727C.flexio_spi_master_edma_handle_t (*C type*), 330
drivers_MCX_MCXW_MCXW727C.flexio_spi_master_edma_transfer_callback_t (*C type*), 330
drivers_MCX_MCXW_MCXW727C.flexio_spi_master_handle_t (*C type*), 366
drivers_MCX_MCXW_MCXW727C.flexio_spi_master_transfer_callback_t (*C type*), 366
drivers_MCX_MCXW_MCXW727C.FLEXIO_SPI_MasterDeinit (*C function*), 356
drivers_MCX_MCXW_MCXW727C.FLEXIO_SPI_MasterGetDefaultConfig (*C function*), 356
drivers_MCX_MCXW_MCXW727C.FLEXIO_SPI_MasterInit (*C function*), 355
drivers_MCX_MCXW_MCXW727C.FLEXIO_SPI_MasterSetBaudRate (*C function*), 359
drivers_MCX_MCXW_MCXW727C.FLEXIO_SPI_MasterTransferAbort (*C function*), 361
drivers_MCX_MCXW_MCXW727C.FLEXIO_SPI_MasterTransferAbortEDMA (*C function*), 328
drivers_MCX_MCXW_MCXW727C.FLEXIO_SPI_MasterTransferBlocking (*C function*), 360
drivers_MCX_MCXW_MCXW727C.FLEXIO_SPI_MasterTransferCreateHandle (*C function*), 361
drivers_MCX_MCXW_MCXW727C.FLEXIO_SPI_MasterTransferCreateHandleEDMA (*C function*), 327
drivers_MCX_MCXW_MCXW727C.FLEXIO_SPI_MasterTransferEDMA (*C function*), 328
drivers_MCX_MCXW_MCXW727C.FLEXIO_SPI_MasterTransferGetCount (*C function*), 362
drivers_MCX_MCXW_MCXW727C.FLEXIO_SPI_MasterTransferGetCountEDMA (*C function*), 328
drivers_MCX_MCXW_MCXW727C.FLEXIO_SPI_MasterTransferHandleIRQ (*C function*), 362
drivers_MCX_MCXW_MCXW727C.FLEXIO_SPI_MasterTransferNonBlocking (*C function*), 361
drivers_MCX_MCXW_MCXW727C.FLEXIO_SPI_ReadBlocking (*C function*), 360
drivers_MCX_MCXW_MCXW727C.FLEXIO_SPI_ReadData (*C function*), 359
drivers_MCX_MCXW_MCXW727C.flexio_spi_shift_direction_t (*C type*), 366
drivers_MCX_MCXW_MCXW727C.flexio_spi_slave_config_t (*C type*), 366
drivers_MCX_MCXW_MCXW727C.flexio_spi_slave_edma_handle_t (*C type*), 330
drivers_MCX_MCXW_MCXW727C.flexio_spi_slave_edma_transfer_callback_t (*C type*), 330
drivers_MCX_MCXW_MCXW727C.flexio_spi_slave_handle_t (*C type*), 366
drivers_MCX_MCXW_MCXW727C.flexio_spi_slave_transfer_callback_t (*C type*), 366
drivers_MCX_MCXW_MCXW727C.FLEXIO_SPI_SlaveDeinit (*C function*), 357
drivers_MCX_MCXW_MCXW727C.FLEXIO_SPI_SlaveGetDefaultConfig (*C function*), 357
drivers_MCX_MCXW_MCXW727C.FLEXIO_SPI_SlaveInit (*C function*), 356
drivers_MCX_MCXW_MCXW727C.FLEXIO_SPI_SlaveTransferAbort (*C function*), 363
drivers_MCX_MCXW_MCXW727C.FLEXIO_SPI_SlaveTransferAbortEDMA (*C function*), 329
drivers_MCX_MCXW_MCXW727C.FLEXIO_SPI_SlaveTransferCreateHandle (*C function*), 362
drivers_MCX_MCXW_MCXW727C.FLEXIO_SPI_SlaveTransferCreateHandleEDMA (*C function*), 329
drivers_MCX_MCXW_MCXW727C.FLEXIO_SPI_SlaveTransferEDMA (*C function*), 329
drivers_MCX_MCXW_MCXW727C.FLEXIO_SPI_SlaveTransferGetCount (*C function*), 363
drivers_MCX_MCXW_MCXW727C.FLEXIO_SPI_SlaveTransferGetCountEDMA (*C function*), 330
drivers_MCX_MCXW_MCXW727C.FLEXIO_SPI_SlaveTransferHandleIRQ (*C function*), 363
drivers_MCX_MCXW_MCXW727C.FLEXIO_SPI_SlaveTransferNonBlocking (*C function*), 362
drivers_MCX_MCXW_MCXW727C.flexio_spi_transfer_t (*C type*), 366
drivers_MCX_MCXW_MCXW727C.FLEXIO_SPI_Type (*C type*), 366
drivers_MCX_MCXW_MCXW727C.FLEXIO_SPI_WriteBlocking (*C function*), 360
drivers_MCX_MCXW_MCXW727C.FLEXIO_SPI_WriteData (*C function*), 359
drivers_MCX_MCXW_MCXW727C.FLEXIO_SPI_XFER_DATA_FORMAT (*C macro*), 366

drivers_MCX_MCXW_MCXW727C.flexio_timer_config_t (*C type*), 324
drivers_MCX_MCXW_MCXW727C.flexio_timer_decrement_source_t (*C type*), 323
drivers_MCX_MCXW_MCXW727C.flexio_timer_disable_condition_t (*C type*), 323
drivers_MCX_MCXW_MCXW727C.flexio_timer_enable_condition_t (*C type*), 323
drivers_MCX_MCXW_MCXW727C.flexio_timer_mode_t (*C type*), 323
drivers_MCX_MCXW_MCXW727C.flexio_timer_output_state_t (*C type*), 323
drivers_MCX_MCXW_MCXW727C.flexio_timer_output_t (*C type*), 323
drivers_MCX_MCXW_MCXW727C.flexio_timer_reset_condition_t (*C type*), 323
drivers_MCX_MCXW_MCXW727C.flexio_timer_start_bit_condition_t (*C type*), 323
drivers_MCX_MCXW_MCXW727C.flexio_timer_stop_bit_condition_t (*C type*), 323
drivers_MCX_MCXW_MCXW727C.flexio_timer_trigger_polarity_t (*C type*), 322
drivers_MCX_MCXW_MCXW727C.FLEXIO_TIMER_TRIGGER_SEL_PININPUT (*C macro*), 325
drivers_MCX_MCXW_MCXW727C.FLEXIO_TIMER_TRIGGER_SEL_SHIFTnSTAT (*C macro*), 325
drivers_MCX_MCXW_MCXW727C.FLEXIO_TIMER_TRIGGER_SEL_TIMn (*C macro*), 325
drivers_MCX_MCXW_MCXW727C.flexio_timer_trigger_source_t (*C type*), 322
drivers_MCX_MCXW_MCXW727C.FLEXIO_TogglePortOutput (*C function*), 314
drivers_MCX_MCXW_MCXW727C.flexio_uart_bit_count_per_char_t (*C type*), 378
drivers_MCX_MCXW_MCXW727C.FLEXIO_UART_ClearStatusFlags (*C function*), 370
drivers_MCX_MCXW_MCXW727C.flexio_uart_config_t (*C type*), 378
drivers_MCX_MCXW_MCXW727C.FLEXIO_UART_Deinit (*C function*), 370
drivers_MCX_MCXW_MCXW727C.FLEXIO_UART_DisableInterrupts (*C function*), 371
drivers_MCX_MCXW_MCXW727C.flexio_uart_edma_handle_t (*C type*), 333
drivers_MCX_MCXW_MCXW727C.flexio_uart_edma_transfer_callback_t (*C type*), 333
drivers_MCX_MCXW_MCXW727C.FLEXIO_UART_Enable (*C function*), 372
drivers_MCX_MCXW_MCXW727C.FLEXIO_UART_EnableInterrupts (*C function*), 371
drivers_MCX_MCXW_MCXW727C.FLEXIO_UART_EnableRxDMA (*C function*), 372
drivers_MCX_MCXW_MCXW727C.FLEXIO_UART_EnableTxDMA (*C function*), 371
drivers_MCX_MCXW_MCXW727C.FLEXIO_UART_FlushShifters (*C function*), 376
drivers_MCX_MCXW_MCXW727C.FLEXIO_UART_GetDefaultConfig (*C function*), 370
drivers_MCX_MCXW_MCXW727C.FLEXIO_UART_GetRxDataRegisterAddress (*C function*), 371
drivers_MCX_MCXW_MCXW727C.FLEXIO_UART_GetStatusFlags (*C function*), 370
drivers_MCX_MCXW_MCXW727C.FLEXIO_UART_GetTxDataRegisterAddress (*C function*), 371
drivers_MCX_MCXW_MCXW727C.flexio_uart_handle_t (*C type*), 378
drivers_MCX_MCXW_MCXW727C.FLEXIO_UART_Init (*C function*), 369
drivers_MCX_MCXW_MCXW727C.FLEXIO_UART_ReadBlocking (*C function*), 373
drivers_MCX_MCXW_MCXW727C.FLEXIO_UART_ReadByte (*C function*), 372
drivers_MCX_MCXW_MCXW727C.flexio_uart_transfer_callback_t (*C type*), 378
drivers_MCX_MCXW_MCXW727C.flexio_uart_transfer_t (*C type*), 378
drivers_MCX_MCXW_MCXW727C.FLEXIO_UART_TransferAbortReceive (*C function*), 376
drivers_MCX_MCXW_MCXW727C.FLEXIO_UART_TransferAbortReceiveEDMA (*C function*), 332
drivers_MCX_MCXW_MCXW727C.FLEXIO_UART_TransferAbortSend (*C function*), 374
drivers_MCX_MCXW_MCXW727C.FLEXIO_UART_TransferAbortSendEDMA (*C function*), 332
drivers_MCX_MCXW_MCXW727C.FLEXIO_UART_TransferCreateHandle (*C function*), 373
drivers_MCX_MCXW_MCXW727C.FLEXIO_UART_TransferCreateHandleEDMA (*C function*), 331
drivers_MCX_MCXW_MCXW727C.FLEXIO_UART_TransferGetReceiveCount (*C function*), 376
drivers_MCX_MCXW_MCXW727C.FLEXIO_UART_TransferGetReceiveCountEDMA (*C function*), 333
drivers_MCX_MCXW_MCXW727C.FLEXIO_UART_TransferGetSendCount (*C function*), 375
drivers_MCX_MCXW_MCXW727C.FLEXIO_UART_TransferGetSendCountEDMA (*C function*), 332
drivers_MCX_MCXW_MCXW727C.FLEXIO_UART_TransferHandleIRQ (*C function*), 376
drivers_MCX_MCXW_MCXW727C.FLEXIO_UART_TransferReceiveEDMA (*C function*), 332
drivers_MCX_MCXW_MCXW727C.FLEXIO_UART_TransferReceiveNonBlocking (*C function*), 375
drivers_MCX_MCXW_MCXW727C.FLEXIO_UART_TransferSendEDMA (*C function*), 331

drivers_MCX_MCXW_MCXW727C.FLEXIO_UART_TransferSendNonBlocking (*C function*), 374
drivers_MCX_MCXW_MCXW727C.FLEXIO_UART_TransferStartRingBuffer (*C function*), 373
drivers_MCX_MCXW_MCXW727C.FLEXIO_UART_TransferStopRingBuffer (*C function*), 374
drivers_MCX_MCXW_MCXW727C.FLEXIO_UART_Type (*C type*), 378
drivers_MCX_MCXW_MCXW727C.FLEXIO_UART_WriteBlocking (*C function*), 372
drivers_MCX_MCXW_MCXW727C.FLEXIO_UART_WriteByte (*C function*), 372
drivers_MCX_MCXW_MCXW727C.FLEXIO_UnregisterHandleIRQ (*C function*), 313
drivers_MCX_MCXW_MCXW727C.fro192m_rf_clk_config_t (*C type*), 192
drivers_MCX_MCXW_MCXW727C.fro192m_rf_clk_div_t (*C type*), 192
drivers_MCX_MCXW_MCXW727C.fro192m_rf_range_t (*C type*), 192
drivers_MCX_MCXW_MCXW727C.FSL_CACHE_DRIVER_VERSION (*C macro*), 156
drivers_MCX_MCXW_MCXW727C.FSL_CCM32K_DRIVER_VERSION (*C macro*), 160
drivers_MCX_MCXW_MCXW727C.FSL_CE_DRIVER_VERSION (*C macro*), 170
drivers_MCX_MCXW_MCXW727C.FSL_CLOCK_DRIVER_VERSION (*C macro*), 181
drivers_MCX_MCXW_MCXW727C.FSL_CMC_DRIVER_VERSION (*C macro*), 205
drivers_MCX_MCXW_MCXW727C.FSL_COMMON_DRIVER_VERSION (*C macro*), 448
drivers_MCX_MCXW_MCXW727C.FSL_COMPONENT_ID (*C macro*), 629, 631, 652
drivers_MCX_MCXW_MCXW727C.FSL_CRC_DRIVER_VERSION (*C macro*), 211
drivers_MCX_MCXW_MCXW727C.FSL_DRIVER_TRANSFER_DOUBLE_WEAK_IRQ (*C macro*), 464
drivers_MCX_MCXW_MCXW727C.FSL_EDMA_DRIVER_VERSION (*C macro*), 226
drivers_MCX_MCXW_MCXW727C.FSL_EWM_DRIVER_VERSION (*C macro*), 242
drivers_MCX_MCXW_MCXW727C.FSL_FLASH_DRIVER_VERSION (*C macro*), 247
drivers_MCX_MCXW_MCXW727C.FSL_FLEXCAN_DRIVER_VERSION (*C macro*), 271
drivers_MCX_MCXW_MCXW727C.FSL_FLEXCAN_EDMA_DRIVER_VERSION (*C macro*), 307
drivers_MCX_MCXW_MCXW727C.FSL_FLEXIO_DRIVER_VERSION (*C macro*), 315
drivers_MCX_MCXW_MCXW727C.FSL_FLEXIO_I2C_MASTER_DRIVER_VERSION (*C macro*), 340
drivers_MCX_MCXW_MCXW727C.FSL_FLEXIO_I2S_DRIVER_VERSION (*C macro*), 350
drivers_MCX_MCXW_MCXW727C.FSL_FLEXIO_SPI_DRIVER_VERSION (*C macro*), 363
drivers_MCX_MCXW_MCXW727C.FSL_FLEXIO_SPI_EDMA_DRIVER_VERSION (*C macro*), 330
drivers_MCX_MCXW_MCXW727C.FSL_FLEXIO_UART_DRIVER_VERSION (*C macro*), 376
drivers_MCX_MCXW_MCXW727C.FSL_FLEXIO_UART_EDMA_DRIVER_VERSION (*C macro*), 333
drivers_MCX_MCXW_MCXW727C.FSL_GPIO_DRIVER_VERSION (*C macro*), 381
drivers_MCX_MCXW_MCXW727C.FSL_I3C_DRIVER_VERSION (*C macro*), 390
drivers_MCX_MCXW_MCXW727C.FSL_IMU_DRIVER_VERSION (*C macro*), 445
drivers_MCX_MCXW_MCXW727C.FSL_LPADC_DRIVER_VERSION (*C macro*), 483
drivers_MCX_MCXW_MCXW727C.FSL_LPCMP_DRIVER_VERSION (*C macro*), 490
drivers_MCX_MCXW_MCXW727C.FSL_LPI2C_DRIVER_VERSION (*C macro*), 498
drivers_MCX_MCXW_MCXW727C.FSL_LPIT_DRIVER_VERSION (*C macro*), 531
drivers_MCX_MCXW_MCXW727C.FSL_LPSPI_DRIVER_VERSION (*C macro*), 547
drivers_MCX_MCXW_MCXW727C.FSL_LPSPI_EDMA_DRIVER_VERSION (*C macro*), 560
drivers_MCX_MCXW_MCXW727C.FSL_LPTMR_DRIVER_VERSION (*C macro*), 571
drivers_MCX_MCXW_MCXW727C.FSL_LPUART_DRIVER_VERSION (*C macro*), 586
drivers_MCX_MCXW_MCXW727C.FSL_LPUART_EDMA_DRIVER_VERSION (*C macro*), 598
drivers_MCX_MCXW_MCXW727C.FSL_LTC_DRIVER_VERSION (*C macro*), 599
drivers_MCX_MCXW_MCXW727C.FSL_MCM_DRIVER_VERSION (*C macro*), 625
drivers_MCX_MCXW_MCXW727C.FSL_MSCM_DRIVER_VERSION (*C macro*), 631, 676
drivers_MCX_MCXW_MCXW727C.FSL_MU_DRIVER_VERSION (*C macro*), 641
drivers_MCX_MCXW_MCXW727C.FSL_PORT_DRIVER_VERSION (*C macro*), 648
drivers_MCX_MCXW_MCXW727C.FSL_RTC_DRIVER_VERSION (*C macro*), 657
drivers_MCX_MCXW_MCXW727C.FSL_SDK_DISABLE_DRIVER_CLOCK_CONTROL (*C macro*), 194
drivers_MCX_MCXW_MCXW727C.FSL_SEMA42_DRIVER_VERSION (*C macro*), 662
drivers_MCX_MCXW_MCXW727C.FSL_SFA_DRIVER_VERSION (*C macro*), 670
drivers_MCX_MCXW_MCXW727C.FSL_SPC_DRIVER_VERSION (*C macro*), 717

drivers_MCX_MCXW_MCXW727C.FSL_SYSPM_DRIVER_VERSION (*C macro*), 726
drivers_MCX_MCXW_MCXW727C.FSL_TPM_DRIVER_VERSION (*C macro*), 733
drivers_MCX_MCXW_MCXW727C.FSL_TRDC_DRIVER_VERSION (*C macro*), 762
drivers_MCX_MCXW_MCXW727C.FSL_TRGMUX_DRIVER_VERSION (*C macro*), 769
drivers_MCX_MCXW_MCXW727C.FSL_VBAT_DRIVER_VERSION (*C macro*), 777
drivers_MCX_MCXW_MCXW727C.FSL_VREF_DRIVER_VERSION (*C macro*), 780
drivers_MCX_MCXW_MCXW727C.FSL_WDOG32_DRIVER_VERSION (*C macro*), 785
drivers_MCX_MCXW_MCXW727C.FSL_WUU_DRIVER_VERSION (*C macro*), 790
drivers_MCX_MCXW_MCXW727C.g_lpspiDummyData (*C var*), 555
drivers_MCX_MCXW_MCXW727C.g_xtal0Freq (*C var*), 192
drivers_MCX_MCXW_MCXW727C.g_xtal32Freq (*C var*), 192
drivers_MCX_MCXW_MCXW727C.GPIO_CheckAttributeBytes (*C function*), 390
drivers_MCX_MCXW_MCXW727C.gpio_checker_attribute_t (*C type*), 383
drivers_MCX_MCXW_MCXW727C.GPIO_CLOCKS (*C macro*), 195
drivers_MCX_MCXW_MCXW727C.GPIO_DisableInterruptControlNonPrivilege (*C function*), 386
drivers_MCX_MCXW_MCXW727C.GPIO_DisableInterruptControlNonSecure (*C function*), 386
drivers_MCX_MCXW_MCXW727C.GPIO_DisablePinControlNonPrivilege (*C function*), 386
drivers_MCX_MCXW_MCXW727C.GPIO_DisablePinControlNonSecure (*C function*), 385
drivers_MCX_MCXW_MCXW727C.GPIO_EnableInterruptControlNonPrivilege (*C function*), 386
drivers_MCX_MCXW_MCXW727C.GPIO_EnableInterruptControlNonSecure (*C function*), 386
drivers_MCX_MCXW_MCXW727C.GPIO_EnablePinControlNonPrivilege (*C function*), 386
drivers_MCX_MCXW_MCXW727C.GPIO_EnablePinControlNonSecure (*C function*), 385
drivers_MCX_MCXW_MCXW727C.GPIO_FIT_REG (*C macro*), 384
drivers_MCX_MCXW_MCXW727C.GPIO_GetVersionInfo (*C function*), 385
drivers_MCX_MCXW_MCXW727C.GPIO_GpioClearInterruptChannelFlags (*C function*), 389
drivers_MCX_MCXW_MCXW727C.GPIO_GpioClearInterruptFlags (*C function*), 389
drivers_MCX_MCXW_MCXW727C.GPIO_GpioGetInterruptChannelFlags (*C function*), 388
drivers_MCX_MCXW_MCXW727C.GPIO_GpioGetInterruptFlags (*C function*), 388
drivers_MCX_MCXW_MCXW727C.gpio_interrupt_config_t (*C type*), 383
drivers_MCX_MCXW_MCXW727C.gpio_interrupt_selection_t (*C type*), 383
drivers_MCX_MCXW_MCXW727C.gpio_pin_config_t (*C type*), 383
drivers_MCX_MCXW_MCXW727C.gpio_pin_direction_t (*C type*), 383
drivers_MCX_MCXW_MCXW727C.gpio_pin_interrupt_control_t (*C enum*), 383
drivers_MCX_MCXW_MCXW727C.gpio_pin_interrupt_control_t.kGPIO_InterruptControlNonPrivilege (*C enumerator*), 383
drivers_MCX_MCXW_MCXW727C.gpio_pin_interrupt_control_t.kGPIO_InterruptControlNonSecure (*C enumerator*), 383
drivers_MCX_MCXW_MCXW727C.gpio_pin_interrupt_control_t.kGPIO_PinControlNonPrivilege (*C enumerator*), 383
drivers_MCX_MCXW_MCXW727C.gpio_pin_interrupt_control_t.kGPIO_PinControlNonSecure (*C enumerator*), 383
drivers_MCX_MCXW_MCXW727C.GPIO_PinClearInterruptFlag (*C function*), 389
drivers_MCX_MCXW_MCXW727C.GPIO_PinGetInterruptFlag (*C function*), 389
drivers_MCX_MCXW_MCXW727C.GPIO_PinInit (*C function*), 385
drivers_MCX_MCXW_MCXW727C.GPIO_PinRead (*C function*), 387
drivers_MCX_MCXW_MCXW727C.GPIO_PinWrite (*C function*), 387
drivers_MCX_MCXW_MCXW727C.GPIO_PortClear (*C function*), 387
drivers_MCX_MCXW_MCXW727C.GPIO_PortClearInterruptFlags (*C macro*), 390
drivers_MCX_MCXW_MCXW727C.GPIO_PortDenit (*C function*), 384
drivers_MCX_MCXW_MCXW727C.GPIO_PortGetInterruptFlags (*C macro*), 390
drivers_MCX_MCXW_MCXW727C.GPIO_PortInit (*C function*), 384
drivers_MCX_MCXW_MCXW727C.GPIO_PortInputDisable (*C function*), 386
drivers_MCX_MCXW_MCXW727C.GPIO_PortInputEnable (*C function*), 386
drivers_MCX_MCXW_MCXW727C.GPIO_PortSet (*C function*), 387
drivers_MCX_MCXW_MCXW727C.GPIO_PortToggle (*C function*), 387
drivers_MCX_MCXW_MCXW727C.GPIO_SecurePrivilegeLock (*C function*), 385
drivers_MCX_MCXW_MCXW727C.GPIO_SetMultipleInterruptPinsConfig (*C function*), 389
drivers_MCX_MCXW_MCXW727C.GPIO_SetPinInterruptChannel (*C function*), 388

drivers_MCX_MCXW_MCXW727C.GPIO_SetPinInterruptConfig (*C function*), 387
drivers_MCX_MCXW_MCXW727C.gpio_version_info_t (*C type*), 384
drivers_MCX_MCXW_MCXW727C.I2C_RETRY_TIMES (*C macro*), 341, 499
drivers_MCX_MCXW_MCXW727C.I2S_RETRY_TIMES (*C macro*), 352
drivers_MCX_MCXW_MCXW727C.i3c_baudrate_hz_t (*C type*), 421
drivers_MCX_MCXW_MCXW727C.i3c_bus_request_t (*C type*), 421
drivers_MCX_MCXW_MCXW727C.i3c_bus_type_t (*C type*), 421
drivers_MCX_MCXW_MCXW727C.I3C_CheckForBusyBus (*C function*), 397
drivers_MCX_MCXW_MCXW727C.I3C_CLOCKS (*C macro*), 195
drivers_MCX_MCXW_MCXW727C.i3c_config_t (*C type*), 393
drivers_MCX_MCXW_MCXW727C.i3c_device_info_t (*C type*), 392
drivers_MCX_MCXW_MCXW727C.i3c_direction_t (*C type*), 421
drivers_MCX_MCXW_MCXW727C.I3C_DriverIRQHandler (*C function*), 413
drivers_MCX_MCXW_MCXW727C.I3C_GetDefaultConfig (*C function*), 393
drivers_MCX_MCXW_MCXW727C.I3C_GetIBIAddress (*C function*), 409
drivers_MCX_MCXW_MCXW727C.I3C_GetIBIType (*C function*), 409
drivers_MCX_MCXW_MCXW727C.I3C_GetInstance (*C function*), 393
drivers_MCX_MCXW_MCXW727C.i3c_hdr_mode_t (*C type*), 392
drivers_MCX_MCXW_MCXW727C.I3C_IBI_BUFF_SIZE (*C macro*), 392
drivers_MCX_MCXW_MCXW727C.i3c_ibi_response_t (*C type*), 421
drivers_MCX_MCXW_MCXW727C.i3c_ibi_state_t (*C type*), 421
drivers_MCX_MCXW_MCXW727C.i3c_ibi_type_t (*C type*), 421
drivers_MCX_MCXW_MCXW727C.I3C_Init (*C function*), 394
drivers_MCX_MCXW_MCXW727C.i3c_legacy_i2c_speed_mode_t (*C type*), 422
drivers_MCX_MCXW_MCXW727C.i3c_master_baudrate_config_t (*C type*), 422
drivers_MCX_MCXW_MCXW727C.i3c_master_bus_mode_t (*C type*), 422
drivers_MCX_MCXW_MCXW727C.i3c_master_config_t (*C type*), 422
drivers_MCX_MCXW_MCXW727C.i3c_master_daa_baudrate_t (*C type*), 421
drivers_MCX_MCXW_MCXW727C.i3c_master_enable_t (*C type*), 421
drivers_MCX_MCXW_MCXW727C.i3c_master_handle_t (*C type*), 422
drivers_MCX_MCXW_MCXW727C.i3c_master_hkeep_t (*C type*), 421
drivers_MCX_MCXW_MCXW727C.i3c_master_isr_t (*C type*), 422
drivers_MCX_MCXW_MCXW727C.i3c_master_src_clk_t (*C type*), 422
drivers_MCX_MCXW_MCXW727C.i3c_master_state_t (*C type*), 421
drivers_MCX_MCXW_MCXW727C.i3c_master_transfer_callback_t (*C type*), 422
drivers_MCX_MCXW_MCXW727C.i3c_master_transfer_t (*C type*), 422
drivers_MCX_MCXW_MCXW727C.I3C_MasterCalcTiming (*C function*), 405
drivers_MCX_MCXW_MCXW727C.I3C_MasterCheckAndClearError (*C function*), 396
drivers_MCX_MCXW_MCXW727C.I3C_MasterClearDeviceCount (*C function*), 410
drivers_MCX_MCXW_MCXW727C.I3C_MasterClearErrorStatusFlags (*C function*), 398
drivers_MCX_MCXW_MCXW727C.I3C_MasterClearStatusFlags (*C function*), 398
drivers_MCX_MCXW_MCXW727C.I3C_MasterDeinit (*C function*), 396
drivers_MCX_MCXW_MCXW727C.I3C_MasterDisableInterrupts (*C function*), 401
drivers_MCX_MCXW_MCXW727C.I3C_MasterEmitIBIResponse (*C function*), 409
drivers_MCX_MCXW_MCXW727C.I3C_MasterEmitRequest (*C function*), 409
drivers_MCX_MCXW_MCXW727C.I3C_MasterEnable (*C function*), 396
drivers_MCX_MCXW_MCXW727C.I3C_MasterEnableDMA (*C function*), 402
drivers_MCX_MCXW_MCXW727C.I3C_MasterEnableInterrupts (*C function*), 400
drivers_MCX_MCXW_MCXW727C.I3C_MasterGetBusIdleState (*C function*), 405
drivers_MCX_MCXW_MCXW727C.I3C_MasterGetDefaultConfig (*C function*), 396
drivers_MCX_MCXW_MCXW727C.I3C_MasterGetDeviceListAfterDAA (*C function*), 410
drivers_MCX_MCXW_MCXW727C.I3C_MasterGetEnabledInterrupts (*C function*), 401
drivers_MCX_MCXW_MCXW727C.I3C_MasterGetErrorStatusFlags (*C function*), 398
drivers_MCX_MCXW_MCXW727C.I3C_MasterGetFifoCounts (*C function*), 404
drivers_MCX_MCXW_MCXW727C.I3C_MasterGetI2CPresetTiming (*C function*), 405
drivers_MCX_MCXW_MCXW727C.I3C_MasterGetIBIRules (*C function*), 409
drivers_MCX_MCXW_MCXW727C.I3C_MasterGetPendingInterrupts (*C function*), 401
drivers_MCX_MCXW_MCXW727C.I3C_MasterGetRxFifoAddress (*C function*), 403

drivers_MCX_MCXW_MCXW727C.I3C_MasterGetState (*C function*), 399
drivers_MCX_MCXW_MCXW727C.I3C_MasterGetStatusFlags (*C function*), 397
drivers_MCX_MCXW_MCXW727C.I3C_MasterGetTxFifoAddress (*C function*), 403
drivers_MCX_MCXW_MCXW727C.I3C_MasterInit (*C function*), 396
drivers_MCX_MCXW_MCXW727C.I3C_MasterProcessDAA (*C function*), 410
drivers_MCX_MCXW_MCXW727C.I3C_MasterProcessDAASpecifiedBaudrate (*C function*), 409
drivers_MCX_MCXW_MCXW727C.I3C_MasterReceive (*C function*), 408
drivers_MCX_MCXW_MCXW727C.I3C_MasterRegisterIBI (*C function*), 409
drivers_MCX_MCXW_MCXW727C.I3C_MasterRepeatedStart (*C function*), 407
drivers_MCX_MCXW_MCXW727C.I3C_MasterRepeatedStartWithRxSize (*C function*), 406
drivers_MCX_MCXW_MCXW727C.I3C_MasterSend (*C function*), 407
drivers_MCX_MCXW_MCXW727C.I3C_MasterSetBaudRate (*C function*), 404
drivers_MCX_MCXW_MCXW727C.I3C_MasterSetTiming (*C function*), 405
drivers_MCX_MCXW_MCXW727C.I3C_MasterSetWatermarks (*C function*), 403
drivers_MCX_MCXW_MCXW727C.I3C_MasterStart (*C function*), 406
drivers_MCX_MCXW_MCXW727C.I3C_MasterStartWithRxSize (*C function*), 406
drivers_MCX_MCXW_MCXW727C.I3C_MasterStop (*C function*), 408
drivers_MCX_MCXW_MCXW727C.I3C_MasterTransferAbort (*C function*), 412
drivers_MCX_MCXW_MCXW727C.I3C_MasterTransferBlocking (*C function*), 410
drivers_MCX_MCXW_MCXW727C.I3C_MasterTransferCreateHandle (*C function*), 411
drivers_MCX_MCXW_MCXW727C.I3C_MasterTransferGetCount (*C function*), 412
drivers_MCX_MCXW_MCXW727C.I3C_MasterTransferHandleIRQ (*C function*), 413
drivers_MCX_MCXW_MCXW727C.I3C_MasterTransferNonBlocking (*C function*), 412
drivers_MCX_MCXW_MCXW727C.I3C_MasterWaitForCtrlDone (*C function*), 397
drivers_MCX_MCXW_MCXW727C.I3C_MAX_DEVCNT (*C macro*), 392
drivers_MCX_MCXW_MCXW727C.i3c_register_ibi_addr_t (*C type*), 421
drivers_MCX_MCXW_MCXW727C.I3C_RETRY_TIMES (*C macro*), 392
drivers_MCX_MCXW_MCXW727C.i3c_rx_term_ops_t (*C type*), 421
drivers_MCX_MCXW_MCXW727C.i3c_rx_trigger_level_t (*C type*), 421
drivers_MCX_MCXW_MCXW727C.i3c_slave_activity_state_t (*C type*), 438
drivers_MCX_MCXW_MCXW727C.i3c_slave_config_t (*C type*), 438
drivers_MCX_MCXW_MCXW727C.i3c_slave_event_t (*C type*), 438
drivers_MCX_MCXW_MCXW727C.i3c_slave_handle_t (*C type*), 438
drivers_MCX_MCXW_MCXW727C.i3c_slave_isr_t (*C type*), 439
drivers_MCX_MCXW_MCXW727C.i3c_slave_transfer_callback_t (*C type*), 438
drivers_MCX_MCXW_MCXW727C.i3c_slave_transfer_event_t (*C type*), 438
drivers_MCX_MCXW_MCXW727C.i3c_slave_transfer_t (*C type*), 438
drivers_MCX_MCXW_MCXW727C.I3C_SlaveCheckAndClearError (*C function*), 400
drivers_MCX_MCXW_MCXW727C.I3C_SlaveClearErrorStatusFlags (*C function*), 400, 429
drivers_MCX_MCXW_MCXW727C.I3C_SlaveClearStatusFlags (*C function*), 399, 428
drivers_MCX_MCXW_MCXW727C.I3C_SlaveDeinit (*C function*), 397, 427
drivers_MCX_MCXW_MCXW727C.I3C_SlaveDisableInterrupts (*C function*), 402, 430
drivers_MCX_MCXW_MCXW727C.I3C_SlaveEnable (*C function*), 397, 428
drivers_MCX_MCXW_MCXW727C.I3C_SlaveEnableDMA (*C function*), 403, 430
drivers_MCX_MCXW_MCXW727C.I3C_SlaveEnableInterrupts (*C function*), 401, 429
drivers_MCX_MCXW_MCXW727C.I3C_SlaveGetActivityState (*C function*), 400, 429
drivers_MCX_MCXW_MCXW727C.I3C_SlaveGetDefaultConfig (*C function*), 397, 427
drivers_MCX_MCXW_MCXW727C.I3C_SlaveGetEnabledInterrupts (*C function*), 402, 430
drivers_MCX_MCXW_MCXW727C.I3C_SlaveGetErrorStatusFlags (*C function*), 400, 428
drivers_MCX_MCXW_MCXW727C.I3C_SlaveGetFifoCounts (*C function*), 404, 431
drivers_MCX_MCXW_MCXW727C.I3C_SlaveGetPendingInterrupts (*C function*), 402, 430
drivers_MCX_MCXW_MCXW727C.I3C_SlaveGetRxFifoAddress (*C function*), 403, 431
drivers_MCX_MCXW_MCXW727C.I3C_SlaveGetStatusFlags (*C function*), 399, 428
drivers_MCX_MCXW_MCXW727C.I3C_SlaveGetTxFifoAddress (*C function*), 403, 431
drivers_MCX_MCXW_MCXW727C.I3C_SlaveInit (*C function*), 397, 427
drivers_MCX_MCXW_MCXW727C.I3C_SlaveReceive (*C function*), 411, 432
drivers_MCX_MCXW_MCXW727C.I3C_SlaveSend (*C function*), 411, 432
drivers_MCX_MCXW_MCXW727C.I3C_SlaveSetWatermarks (*C function*), 404, 431

drivers_MCX_MCXW_MCXW727C.I3C_SlaveTransferAbort (*C function*), 433
drivers_MCX_MCXW_MCXW727C.I3C_SlaveTransferCreateHandle (*C function*), 432
drivers_MCX_MCXW_MCXW727C.I3C_SlaveTransferGetCount (*C function*), 433
drivers_MCX_MCXW_MCXW727C.I3C_SlaveTransferHandleIRQ (*C function*), 433
drivers_MCX_MCXW_MCXW727C.I3C_SlaveTransferNonBlocking (*C function*), 432
drivers_MCX_MCXW_MCXW727C.i3c_start_scl_delay_t (*C type*), 421
drivers_MCX_MCXW_MCXW727C.i3c_timing_t (*C type*), 422
drivers_MCX_MCXW_MCXW727C.i3c_tx_trigger_level_t (*C type*), 421
drivers_MCX_MCXW_MCXW727C.ICACHE_InvalidateByRange (*C function*), 156
drivers_MCX_MCXW_MCXW727C.ID_EXT (*C macro*), 288
drivers_MCX_MCXW_MCXW727C.ID_STD_HIGH (*C macro*), 288
drivers_MCX_MCXW_MCXW727C.ID_STD_LOW (*C macro*), 288
drivers_MCX_MCXW_MCXW727C.IMU_BUSY_POLL_COUNT (*C macro*), 448
drivers_MCX_MCXW_MCXW727C.IMU_ClearPendingInterrupts (*C function*), 445
drivers_MCX_MCXW_MCXW727C.IMU_Deinit (*C function*), 442
drivers_MCX_MCXW_MCXW727C.IMU_ERR_TIMEOUT (*C macro*), 447
drivers_MCX_MCXW_MCXW727C.IMU_ERR_TX_FIFO_LOCKED (*C macro*), 447
drivers_MCX_MCXW_MCXW727C.IMU_FIFO_SW_WRAPAROUND (*C macro*), 448
drivers_MCX_MCXW_MCXW727C.IMU_FlushSendFifo (*C function*), 445
drivers_MCX_MCXW_MCXW727C.IMU_GetReceivedMsgCount (*C function*), 445
drivers_MCX_MCXW_MCXW727C.IMU_GetSendFifoEmptySpace (*C function*), 445
drivers_MCX_MCXW_MCXW727C.IMU_GetStatusFlags (*C function*), 445
drivers_MCX_MCXW_MCXW727C.IMU_Init (*C function*), 441
drivers_MCX_MCXW_MCXW727C.IMU_LOCK_TX_FIFO (*C macro*), 447
drivers_MCX_MCXW_MCXW727C.IMU_LockSendFifo (*C function*), 444
drivers_MCX_MCXW_MCXW727C.IMU_MAX_MSG_FIFO_WATER_MARK (*C macro*), 448
drivers_MCX_MCXW_MCXW727C.IMU_MSG_FIFO_CNTL_FIFO_FLUSH_MASK (*C macro*), 447
drivers_MCX_MCXW_MCXW727C.IMU_MSG_FIFO_CNTL_FIFO_FULL_WATERMARK (*C macro*), 447
drivers_MCX_MCXW_MCXW727C.IMU_MSG_FIFO_CNTL_FIFO_FULL_WATERMARK_MASK (*C macro*), 447
drivers_MCX_MCXW_MCXW727C.IMU_MSG_FIFO_CNTL_FIFO_FULL_WATERMARK_SHIFT (*C macro*), 447
drivers_MCX_MCXW_MCXW727C.IMU_MSG_FIFO_CNTL_MSG_RDY_INT_CLR_MASK (*C macro*), 447
drivers_MCX_MCXW_MCXW727C.IMU_MSG_FIFO_CNTL_SP_AV_INT_CLR_MASK (*C macro*), 447
drivers_MCX_MCXW_MCXW727C.IMU_MSG_FIFO_CNTL_WAIT_FOR_ACK_MASK (*C macro*), 447
drivers_MCX_MCXW_MCXW727C.IMU_MSG_FIFO_MAX_COUNT (*C macro*), 447
drivers_MCX_MCXW_MCXW727C.IMU_MSG_FIFO_STATUS_MSG_COUNT_MASK (*C macro*), 446
drivers_MCX_MCXW_MCXW727C.IMU_MSG_FIFO_STATUS_MSG_COUNT_SHIFT (*C macro*), 446
drivers_MCX_MCXW_MCXW727C.IMU_MSG_FIFO_STATUS_MSG_FIFO_ALMOST_FULL_MASK (*C macro*), 446
drivers_MCX_MCXW_MCXW727C.IMU_MSG_FIFO_STATUS_MSG_FIFO_EMPTY_MASK (*C macro*), 446
drivers_MCX_MCXW_MCXW727C.IMU_MSG_FIFO_STATUS_MSG_FIFO_FULL_MASK (*C macro*), 446
drivers_MCX_MCXW_MCXW727C.IMU_MSG_FIFO_STATUS_MSG_FIFO_LOCKED_MASK (*C macro*), 446
drivers_MCX_MCXW_MCXW727C.IMU_MSG_FIFO_STATUS_RD_PTR_MASK (*C macro*), 447
drivers_MCX_MCXW_MCXW727C.IMU_MSG_FIFO_STATUS_RD_PTR_SHIFT (*C macro*), 447
drivers_MCX_MCXW_MCXW727C.IMU_MSG_FIFO_STATUS_WR_PTR_MASK (*C macro*),

446
drivers_MCX_MCXW_MCXW727C.IMU_MSG_FIFO_STATUS_WR_PTR_SHIFT (*C macro*), 446
drivers_MCX_MCXW_MCXW727C.IMU_RD_MSG (*C macro*), 447
drivers_MCX_MCXW_MCXW727C.IMU_RD_PTR (*C macro*), 448
drivers_MCX_MCXW_MCXW727C.IMU_ReadMsg (*C function*), 442
drivers_MCX_MCXW_MCXW727C.IMU_ReceiveMsgsBlocking (*C function*), 443
drivers_MCX_MCXW_MCXW727C.IMU_RX_FIFO_EMPTY (*C macro*), 447
drivers_MCX_MCXW_MCXW727C.IMU_RX_FIFO_LOCKED (*C macro*), 447
drivers_MCX_MCXW_MCXW727C.IMU_RX_FIFO_LOCKED_FROM_STATUS (*C macro*), 447
drivers_MCX_MCXW_MCXW727C.IMU_RX_FIFO_MSG_COUNT (*C macro*), 447
drivers_MCX_MCXW_MCXW727C.IMU_RX_FIFO_MSG_COUNT_FROM_STATUS (*C macro*), 447
drivers_MCX_MCXW_MCXW727C.IMU_RX_FIFO_STATUS (*C macro*), 447
drivers_MCX_MCXW_MCXW727C.IMU_SendMsgPtrBlocking (*C function*), 444
drivers_MCX_MCXW_MCXW727C.IMU_SendMsgsBlocking (*C function*), 442
drivers_MCX_MCXW_MCXW727C.IMU_SetSendFifoWaterMark (*C function*), 445
drivers_MCX_MCXW_MCXW727C.IMU_TryReceiveMsgs (*C function*), 443
drivers_MCX_MCXW_MCXW727C.IMU_TrySendMsgs (*C function*), 442
drivers_MCX_MCXW_MCXW727C.IMU_TX_FIFO_ALMOST_FULL (*C macro*), 447
drivers_MCX_MCXW_MCXW727C.IMU_TX_FIFO_CNTL (*C macro*), 447
drivers_MCX_MCXW_MCXW727C.IMU_TX_FIFO_LOCKED (*C macro*), 447
drivers_MCX_MCXW_MCXW727C.IMU_TX_FIFO_MSG_COUNT (*C macro*), 447
drivers_MCX_MCXW_MCXW727C.IMU_TX_FIFO_STATUS (*C macro*), 447
drivers_MCX_MCXW_MCXW727C.IMU_Type (*C struct*), 448
drivers_MCX_MCXW_MCXW727C.IMU_UNLOCK_TX_FIFO (*C macro*), 447
drivers_MCX_MCXW_MCXW727C.IMU_WR_MSG (*C macro*), 447
drivers_MCX_MCXW_MCXW727C.IMU_WR_PTR (*C macro*), 448
drivers_MCX_MCXW_MCXW727C.IMU_WriteMsg (*C function*), 442
drivers_MCX_MCXW_MCXW727C.INSTRUCTION_COUNT_STABLE_TIMEOUT (*C macro*), 726
drivers_MCX_MCXW_MCXW727C.IRQ_ClearPendingIRQ (*C function*), 463
drivers_MCX_MCXW_MCXW727C.IRQ_SetPriority (*C function*), 463
drivers_MCX_MCXW_MCXW727C.kLpi2cMasterIrqs (*C var*), 499
drivers_MCX_MCXW_MCXW727C.kLpi2cSlaveIrqs (*C var*), 499
drivers_MCX_MCXW_MCXW727C.L1CACHE_CleanDCacheByRange (*C function*), 155
drivers_MCX_MCXW_MCXW727C.L1CACHE_CleanInvalidateDCacheByRange (*C function*), 155
drivers_MCX_MCXW_MCXW727C.L1CACHE_DisableCodeCache (*C function*), 155
drivers_MCX_MCXW_MCXW727C.L1CACHE_EnableCodeCache (*C function*), 155
drivers_MCX_MCXW_MCXW727C.L1CACHE_InvalidateCodeCache (*C function*), 155
drivers_MCX_MCXW_MCXW727C.L1CACHE_InvalidateDCacheByRange (*C function*), 155
drivers_MCX_MCXW_MCXW727C.L1CACHE_InvalidateICacheByRange (*C function*), 155
drivers_MCX_MCXW_MCXW727C.LPADC_CALIBRATION_READY_TIMEOUT (*C macro*), 474
drivers_MCX_MCXW_MCXW727C.lpadc_calibration_value_t (*C type*), 474
drivers_MCX_MCXW_MCXW727C.LPADC_ClearStatusFlags (*C function*), 475
drivers_MCX_MCXW_MCXW727C.LPADC_ClearTriggerStatusFlags (*C function*), 476
drivers_MCX_MCXW_MCXW727C.LPADC_CLOCKS (*C macro*), 195
drivers_MCX_MCXW_MCXW727C.lpadc_config_t (*C struct*), 483
drivers_MCX_MCXW_MCXW727C.lpadc_config_t.conversionAverageMode (*C var*), 483
drivers_MCX_MCXW_MCXW727C.lpadc_config_t.convPauseDelay (*C var*), 484
drivers_MCX_MCXW_MCXW727C.lpadc_config_t.enableAnalogPreliminary (*C var*), 483
drivers_MCX_MCXW_MCXW727C.lpadc_config_t.enableConvPause (*C var*), 484
drivers_MCX_MCXW_MCXW727C.lpadc_config_t.enableInDozeMode (*C var*), 483
drivers_MCX_MCXW_MCXW727C.lpadc_config_t.enableInternalClock (*C var*), 483
drivers_MCX_MCXW_MCXW727C.lpadc_config_t.enableVref1LowVoltage (*C var*), 483
drivers_MCX_MCXW_MCXW727C.lpadc_config_t.FIFOWatermark (*C var*), 484
drivers_MCX_MCXW_MCXW727C.lpadc_config_t.powerLevelMode (*C var*), 484
drivers_MCX_MCXW_MCXW727C.lpadc_config_t.powerUpDelay (*C var*), 483

drivers_MCX_MCXW_MCXW727C.lpadc_config_t.referenceVoltageSource (*C var*), 484
drivers_MCX_MCXW_MCXW727C.lpadc_config_t.triggerPriorityPolicy (*C var*), 484
drivers_MCX_MCXW_MCXW727C.lpadc_conv_command_config_t (*C struct*), 484
drivers_MCX_MCXW_MCXW727C.lpadc_conv_command_config_t.chainedNextCommandNumber (*C var*), 485
drivers_MCX_MCXW_MCXW727C.lpadc_conv_command_config_t.channelBNumber (*C var*), 484
drivers_MCX_MCXW_MCXW727C.lpadc_conv_command_config_t.channelBScaleMode (*C var*), 484
drivers_MCX_MCXW_MCXW727C.lpadc_conv_command_config_t.channelNumber (*C var*), 484
drivers_MCX_MCXW_MCXW727C.lpadc_conv_command_config_t.conversionResolutionMode (*C var*), 485
drivers_MCX_MCXW_MCXW727C.lpadc_conv_command_config_t.enableAutoChannelIncrement (*C var*), 485
drivers_MCX_MCXW_MCXW727C.lpadc_conv_command_config_t.enableWaitTrigger (*C var*), 485
drivers_MCX_MCXW_MCXW727C.lpadc_conv_command_config_t.hardwareAverageMode (*C var*), 485
drivers_MCX_MCXW_MCXW727C.lpadc_conv_command_config_t.hardwareCompareMode (*C var*), 485
drivers_MCX_MCXW_MCXW727C.lpadc_conv_command_config_t.hardwareCompareValueHigh (*C var*), 485
drivers_MCX_MCXW_MCXW727C.lpadc_conv_command_config_t.hardwareCompareValueLow (*C var*), 485
drivers_MCX_MCXW_MCXW727C.lpadc_conv_command_config_t.loopCount (*C var*), 485
drivers_MCX_MCXW_MCXW727C.lpadc_conv_command_config_t.sampleChannelMode (*C var*), 484
drivers_MCX_MCXW_MCXW727C.lpadc_conv_command_config_t.sampleScaleMode (*C var*), 484
drivers_MCX_MCXW_MCXW727C.lpadc_conv_command_config_t.sampleTimeMode (*C var*), 485
drivers_MCX_MCXW_MCXW727C.lpadc_conv_result_t (*C struct*), 486
drivers_MCX_MCXW_MCXW727C.lpadc_conv_result_t.commandIdSource (*C var*), 486
drivers_MCX_MCXW_MCXW727C.lpadc_conv_result_t.convValue (*C var*), 486
drivers_MCX_MCXW_MCXW727C.lpadc_conv_result_t.loopCountIndex (*C var*), 486
drivers_MCX_MCXW_MCXW727C.lpadc_conv_result_t.triggerIdSource (*C var*), 486
drivers_MCX_MCXW_MCXW727C.lpadc_conv_trigger_config_t (*C struct*), 485
drivers_MCX_MCXW_MCXW727C.lpadc_conv_trigger_config_t.delayPower (*C var*), 485
drivers_MCX_MCXW_MCXW727C.lpadc_conv_trigger_config_t.enableHardwareTrigger (*C var*), 486
drivers_MCX_MCXW_MCXW727C.lpadc_conv_trigger_config_t.priority (*C var*), 486
drivers_MCX_MCXW_MCXW727C.lpadc_conv_trigger_config_t.targetCommandId (*C var*), 485
drivers_MCX_MCXW_MCXW727C.lpadc_conversion_average_mode_t (*C type*), 473
drivers_MCX_MCXW_MCXW727C.LPADC_CONVERSION_COMPLETE_TIMEOUT (*C macro*), 474
drivers_MCX_MCXW_MCXW727C.lpadc_conversion_resolution_mode_t (*C type*), 473
drivers_MCX_MCXW_MCXW727C.LPADC_Deinit (*C function*), 475
drivers_MCX_MCXW_MCXW727C.LPADC_DisableInterrupts (*C function*), 476
drivers_MCX_MCXW_MCXW727C.LPADC_DoAutoCalibration (*C function*), 479
drivers_MCX_MCXW_MCXW727C.LPADC_DoOffsetCalibration (*C function*), 480
drivers_MCX_MCXW_MCXW727C.LPADC_DoResetConfig (*C function*), 475
drivers_MCX_MCXW_MCXW727C.LPADC_DoResetFIFO (*C function*), 475
drivers_MCX_MCXW_MCXW727C.LPADC_DoSoftwareTrigger (*C function*), 478
drivers_MCX_MCXW_MCXW727C.LPADC_Enable (*C function*), 475
drivers_MCX_MCXW_MCXW727C.LPADC_EnableCalibration (*C function*), 479
drivers_MCX_MCXW_MCXW727C.LPADC_EnableExtraCycle (*C function*), 482
drivers_MCX_MCXW_MCXW727C.LPADC_EnableFIFOWatermarkDMA (*C function*), 476
drivers_MCX_MCXW_MCXW727C.LPADC_EnableHardwareTriggerCommandSelection (*C function*), 478
drivers_MCX_MCXW_MCXW727C.LPADC_EnableHighSpeedConversionMode (*C function*), 482
drivers_MCX_MCXW_MCXW727C.LPADC_EnableInterrupts (*C function*), 476
drivers_MCX_MCXW_MCXW727C.LPADC_EnableJustifiedLeft (*C function*), 483

drivers_MCX_MCXW_MCXW727C.LPADC_EnableOffsetCalibration (*C function*), 480
drivers_MCX_MCXW_MCXW727C.LPADC_FinishAutoCalibration (*C function*), 480
drivers_MCX_MCXW_MCXW727C.LPADC_GAIN_CAL_READY_TIMEOUT (*C macro*), 474
drivers_MCX_MCXW_MCXW727C.LPADC_GET_ACTIVE_COMMAND_STATUS (*C macro*), 474
drivers_MCX_MCXW_MCXW727C.LPADC_GET_ACTIVE_TRIGGER_STATUE (*C macro*), 474
drivers_MCX_MCXW_MCXW727C.LPADC_GetCalibrationValue (*C function*), 481
drivers_MCX_MCXW_MCXW727C.LPADC_GetConvResult (*C function*), 477
drivers_MCX_MCXW_MCXW727C.LPADC_GetConvResultBlocking (*C function*), 477
drivers_MCX_MCXW_MCXW727C.LPADC_GetConvResultCount (*C function*), 477
drivers_MCX_MCXW_MCXW727C.LPADC_GetDefaultConfig (*C function*), 474
drivers_MCX_MCXW_MCXW727C.LPADC_GetDefaultConvCommandConfig (*C function*), 478
drivers_MCX_MCXW_MCXW727C.LPADC_GetDefaultConvTriggerConfig (*C function*), 477
drivers_MCX_MCXW_MCXW727C.LPADC_GetHighSpeedTrimValue (*C function*), 481
drivers_MCX_MCXW_MCXW727C.LPADC_GetOffsetValue (*C function*), 480
drivers_MCX_MCXW_MCXW727C.LPADC_GetStatusFlags (*C function*), 475
drivers_MCX_MCXW_MCXW727C.LPADC_GetTriggerStatusFlags (*C function*), 476
drivers_MCX_MCXW_MCXW727C.LPADC_GetTuneValue (*C function*), 482
drivers_MCX_MCXW_MCXW727C.lpadc_hardware_average_mode_t (*C type*), 472
drivers_MCX_MCXW_MCXW727C.lpadc_hardware_compare_mode_t (*C type*), 473
drivers_MCX_MCXW_MCXW727C.LPADC_Init (*C function*), 474
drivers_MCX_MCXW_MCXW727C.lpadc_offset_calibration_mode_t (*C type*), 473
drivers_MCX_MCXW_MCXW727C.lpadc_power_level_mode_t (*C type*), 473
drivers_MCX_MCXW_MCXW727C.LPADC_PrepareAutoCalibration (*C function*), 480
drivers_MCX_MCXW_MCXW727C.lpadc_reference_voltage_source_t (*C type*), 473
drivers_MCX_MCXW_MCXW727C.LPADC_RequestHighSpeedModeTrim (*C function*), 481
drivers_MCX_MCXW_MCXW727C.lpadc_sample_channel_mode_t (*C type*), 472
drivers_MCX_MCXW_MCXW727C.lpadc_sample_scale_mode_t (*C type*), 472
drivers_MCX_MCXW_MCXW727C.lpadc_sample_time_mode_t (*C type*), 472
drivers_MCX_MCXW_MCXW727C.LPADC_SetCalibrationValue (*C function*), 481
drivers_MCX_MCXW_MCXW727C.LPADC_SetConvCommandConfig (*C function*), 478
drivers_MCX_MCXW_MCXW727C.LPADC_SetConvTriggerConfig (*C function*), 477
drivers_MCX_MCXW_MCXW727C.LPADC_SetHighSpeedTrimValue (*C function*), 482
drivers_MCX_MCXW_MCXW727C.LPADC_SetOffsetCalibrationMode (*C function*), 480
drivers_MCX_MCXW_MCXW727C.LPADC_SetOffsetValue (*C function*), 479
drivers_MCX_MCXW_MCXW727C.LPADC_SetTuneValue (*C function*), 482
drivers_MCX_MCXW_MCXW727C.lpadc_trigger_priority_policy_t (*C type*), 473
drivers_MCX_MCXW_MCXW727C.lpadc_tune_value_t (*C type*), 473
drivers_MCX_MCXW_MCXW727C.LPADC_USE_FIXED_POINT_GAIN_CALCULATION (*C macro*), 474
drivers_MCX_MCXW_MCXW727C.LPCMP_CCR1_COUTA_CFG (*C macro*), 495
drivers_MCX_MCXW_MCXW727C.LPCMP_CCR1_COUTA_CFG_MASK (*C macro*), 495
drivers_MCX_MCXW_MCXW727C.LPCMP_CCR1_COUTA_CFG_SHIFT (*C macro*), 495
drivers_MCX_MCXW_MCXW727C.LPCMP_CCR1_EVT_SEL_CFG (*C macro*), 495
drivers_MCX_MCXW_MCXW727C.LPCMP_CCR1_EVT_SEL_CFG_MASK (*C macro*), 495
drivers_MCX_MCXW_MCXW727C.LPCMP_CCR1_EVT_SEL_CFG_SHIFT (*C macro*), 495
drivers_MCX_MCXW_MCXW727C.LPCMP_ClearInputChangedFlags (*C function*), 490
drivers_MCX_MCXW_MCXW727C.LPCMP_ClearStatusFlags (*C function*), 488
drivers_MCX_MCXW_MCXW727C.LPCMP_CLOCKS (*C macro*), 195
drivers_MCX_MCXW_MCXW727C.lpcmp_close_window_event_t (*C type*), 494
drivers_MCX_MCXW_MCXW727C.lpcmp_config_t (*C type*), 495
drivers_MCX_MCXW_MCXW727C.lpcmp_couta_signal_t (*C type*), 494
drivers_MCX_MCXW_MCXW727C.lpcmp_dac_config_t (*C type*), 495
drivers_MCX_MCXW_MCXW727C.lpcmp_dac_reference_voltage_source_t (*C type*), 494
drivers_MCX_MCXW_MCXW727C.LPCMP_Deinit (*C function*), 487
drivers_MCX_MCXW_MCXW727C.LPCMP_DisableInterrupts (*C function*), 488
drivers_MCX_MCXW_MCXW727C.LPCMP_Enable (*C function*), 487
drivers_MCX_MCXW_MCXW727C.LPCMP_EnableDMA (*C function*), 487

drivers_MCX_MCXW_MCXW727C.LPCMP_EnableInterrupts (*C function*), 488
drivers_MCX_MCXW_MCXW727C.LPCMP_EnableRoundRobinInternalTimer (*C function*), 489
drivers_MCX_MCXW_MCXW727C.LPCMP_EnableRoundRobinMode (*C function*), 489
drivers_MCX_MCXW_MCXW727C.LPCMP_EnableWindowMode (*C function*), 488
drivers_MCX_MCXW_MCXW727C.lpcmp_filter_config_t (*C type*), 494
drivers_MCX_MCXW_MCXW727C.lpcmp_functional_source_clock_t (*C type*), 494
drivers_MCX_MCXW_MCXW727C.LPCMP_GetComparisonResult (*C function*), 490
drivers_MCX_MCXW_MCXW727C.LPCMP_GetDefaultConfig (*C function*), 487
drivers_MCX_MCXW_MCXW727C.LPCMP_GetInputChangedFlags (*C function*), 490
drivers_MCX_MCXW_MCXW727C.LPCMP_GetStatusFlags (*C function*), 488
drivers_MCX_MCXW_MCXW727C.lpcmp_hysteresis_mode_t (*C type*), 494
drivers_MCX_MCXW_MCXW727C.LPCMP_Init (*C function*), 486
drivers_MCX_MCXW_MCXW727C.lpcmp_minus_input_src_t (*C type*), 495
drivers_MCX_MCXW_MCXW727C.lpcmp_plus_input_src_t (*C type*), 495
drivers_MCX_MCXW_MCXW727C.lpcmp_power_mode_t (*C type*), 494
drivers_MCX_MCXW_MCXW727C.lpcmp_roundrobin_clock_source_t (*C type*), 494
drivers_MCX_MCXW_MCXW727C.lpcmp_roundrobin_config_t (*C type*), 495
drivers_MCX_MCXW_MCXW727C.lpcmp_roundrobin_fixedmuxport_t (*C type*), 494
drivers_MCX_MCXW_MCXW727C.lpcmp_roundrobin_trigger_source_t (*C type*), 494
drivers_MCX_MCXW_MCXW727C.LPCMP_SetDACConfig (*C function*), 488
drivers_MCX_MCXW_MCXW727C.LPCMP_SetFilterConfig (*C function*), 488
drivers_MCX_MCXW_MCXW727C.LPCMP_SetInputChannels (*C function*), 487
drivers_MCX_MCXW_MCXW727C.LPCMP_SetPreSetValue (*C function*), 489
drivers_MCX_MCXW_MCXW727C.LPCMP_SetRoundRobinConfig (*C function*), 489
drivers_MCX_MCXW_MCXW727C.LPCMP_SetRoundRobinInternalTimer (*C function*), 489
drivers_MCX_MCXW_MCXW727C.LPCMP_SetWindowControl (*C function*), 489
drivers_MCX_MCXW_MCXW727C.lpcmp_window_control_config_t (*C type*), 495
drivers_MCX_MCXW_MCXW727C.LPI2C_CheckForBusyBus (*C function*), 500
drivers_MCX_MCXW_MCXW727C.LPI2C_CLOCKS (*C macro*), 195
drivers_MCX_MCXW_MCXW727C.lpi2c_data_match_config_mode_t (*C type*), 511
drivers_MCX_MCXW_MCXW727C.lpi2c_data_match_config_t (*C type*), 511
drivers_MCX_MCXW_MCXW727C.lpi2c_direction_t (*C type*), 511
drivers_MCX_MCXW_MCXW727C.LPI2C_DriverIRQHandler (*C function*), 498
drivers_MCX_MCXW_MCXW727C.LPI2C_GetInstance (*C function*), 499
drivers_MCX_MCXW_MCXW727C.lpi2c_host_request_polarity_t (*C type*), 511
drivers_MCX_MCXW_MCXW727C.lpi2c_host_request_source_t (*C type*), 511
drivers_MCX_MCXW_MCXW727C.lpi2c_master_config_t (*C type*), 511
drivers_MCX_MCXW_MCXW727C.lpi2c_master_edma_handle_t (*C type*), 516
drivers_MCX_MCXW_MCXW727C.lpi2c_master_edma_transfer_callback_t (*C type*), 516
drivers_MCX_MCXW_MCXW727C.lpi2c_master_handle_t (*C type*), 511
drivers_MCX_MCXW_MCXW727C.lpi2c_master_isr_t (*C type*), 512
drivers_MCX_MCXW_MCXW727C.lpi2c_master_pin_config_t (*C type*), 511
drivers_MCX_MCXW_MCXW727C.lpi2c_master_transfer_callback_t (*C type*), 511
drivers_MCX_MCXW_MCXW727C.lpi2c_master_transfer_t (*C type*), 511
drivers_MCX_MCXW_MCXW727C.LPI2C_MasterCheckAndClearError (*C function*), 500
drivers_MCX_MCXW_MCXW727C.LPI2C_MasterClearStatusFlags (*C function*), 501
drivers_MCX_MCXW_MCXW727C.LPI2C_MasterConfigureDataMatch (*C function*), 500
drivers_MCX_MCXW_MCXW727C.LPI2C_MasterCreateEDMAHandle (*C function*), 515
drivers_MCX_MCXW_MCXW727C.LPI2C_MasterDeinit (*C function*), 500
drivers_MCX_MCXW_MCXW727C.LPI2C_MasterDisableInterrupts (*C function*), 502
drivers_MCX_MCXW_MCXW727C.LPI2C_MasterEnable (*C function*), 501
drivers_MCX_MCXW_MCXW727C.LPI2C_MasterEnableDMA (*C function*), 502
drivers_MCX_MCXW_MCXW727C.LPI2C_MasterEnableInterrupts (*C function*), 502
drivers_MCX_MCXW_MCXW727C.LPI2C_MasterGetBusIdleState (*C function*), 504
drivers_MCX_MCXW_MCXW727C.LPI2C_MasterGetDefaultConfig (*C function*), 499
drivers_MCX_MCXW_MCXW727C.LPI2C_MasterGetEnabledInterrupts (*C function*), 502
drivers_MCX_MCXW_MCXW727C.LPI2C_MasterGetFifoCounts (*C function*), 503
drivers_MCX_MCXW_MCXW727C.LPI2C_MasterGetRxFifoAddress (*C function*), 503

drivers_MCX_MCXW_MCXW727C.LPI2C_MasterGetStatusFlags (*C function*), 501
drivers_MCX_MCXW_MCXW727C.LPI2C_MasterGetTxFifoAddress (*C function*), 502
drivers_MCX_MCXW_MCXW727C.LPI2C_MasterInit (*C function*), 500
drivers_MCX_MCXW_MCXW727C.LPI2C_MasterReceive (*C function*), 505
drivers_MCX_MCXW_MCXW727C.LPI2C_MasterRepeatedStart (*C function*), 504
drivers_MCX_MCXW_MCXW727C.LPI2C_MasterReset (*C function*), 501
drivers_MCX_MCXW_MCXW727C.LPI2C_MasterSend (*C function*), 505
drivers_MCX_MCXW_MCXW727C.LPI2C_MasterSetBaudRate (*C function*), 503
drivers_MCX_MCXW_MCXW727C.LPI2C_MasterSetWatermarks (*C function*), 503
drivers_MCX_MCXW_MCXW727C.LPI2C_MasterStart (*C function*), 504
drivers_MCX_MCXW_MCXW727C.LPI2C_MasterStop (*C function*), 505
drivers_MCX_MCXW_MCXW727C.LPI2C_MasterTransferAbort (*C function*), 507
drivers_MCX_MCXW_MCXW727C.LPI2C_MasterTransferAbortEDMA (*C function*), 516
drivers_MCX_MCXW_MCXW727C.LPI2C_MasterTransferBlocking (*C function*), 506
drivers_MCX_MCXW_MCXW727C.LPI2C_MasterTransferCreateHandle (*C function*), 506
drivers_MCX_MCXW_MCXW727C.LPI2C_MasterTransferEDMA (*C function*), 515
drivers_MCX_MCXW_MCXW727C.LPI2C_MasterTransferGetCount (*C function*), 507
drivers_MCX_MCXW_MCXW727C.LPI2C_MasterTransferGetCountEDMA (*C function*), 516
drivers_MCX_MCXW_MCXW727C.LPI2C_MasterTransferHandleIRQ (*C function*), 507
drivers_MCX_MCXW_MCXW727C.LPI2C_MasterTransferNonBlocking (*C function*), 506
drivers_MCX_MCXW_MCXW727C.lpi2c_slave_address_match_t (*C type*), 525
drivers_MCX_MCXW_MCXW727C.lpi2c_slave_config_t (*C type*), 525
drivers_MCX_MCXW_MCXW727C.lpi2c_slave_handle_t (*C type*), 526
drivers_MCX_MCXW_MCXW727C.lpi2c_slave_transfer_callback_t (*C type*), 526
drivers_MCX_MCXW_MCXW727C.lpi2c_slave_transfer_event_t (*C type*), 525
drivers_MCX_MCXW_MCXW727C.lpi2c_slave_transfer_t (*C type*), 526
drivers_MCX_MCXW_MCXW727C.LPI2C_SlaveClearStatusFlags (*C function*), 519
drivers_MCX_MCXW_MCXW727C.LPI2C_SlaveDeinit (*C function*), 518
drivers_MCX_MCXW_MCXW727C.LPI2C_SlaveDisableInterrupts (*C function*), 520
drivers_MCX_MCXW_MCXW727C.LPI2C_SlaveEnable (*C function*), 518
drivers_MCX_MCXW_MCXW727C.LPI2C_SlaveEnableAckStall (*C function*), 521
drivers_MCX_MCXW_MCXW727C.LPI2C_SlaveEnableDMA (*C function*), 520
drivers_MCX_MCXW_MCXW727C.LPI2C_SlaveEnableInterrupts (*C function*), 519
drivers_MCX_MCXW_MCXW727C.LPI2C_SlaveGetBusIdleState (*C function*), 520
drivers_MCX_MCXW_MCXW727C.LPI2C_SlaveGetDefaultConfig (*C function*), 517
drivers_MCX_MCXW_MCXW727C.LPI2C_SlaveGetEnabledInterrupts (*C function*), 520
drivers_MCX_MCXW_MCXW727C.LPI2C_SlaveGetReceivedAddress (*C function*), 521
drivers_MCX_MCXW_MCXW727C.LPI2C_SlaveGetStatusFlags (*C function*), 519
drivers_MCX_MCXW_MCXW727C.LPI2C_SlaveInit (*C function*), 518
drivers_MCX_MCXW_MCXW727C.LPI2C_SlaveReceive (*C function*), 521
drivers_MCX_MCXW_MCXW727C.LPI2C_SlaveReset (*C function*), 518
drivers_MCX_MCXW_MCXW727C.LPI2C_SlaveSend (*C function*), 521
drivers_MCX_MCXW_MCXW727C.LPI2C_SlaveTransferAbort (*C function*), 523
drivers_MCX_MCXW_MCXW727C.LPI2C_SlaveTransferCreateHandle (*C function*), 521
drivers_MCX_MCXW_MCXW727C.LPI2C_SlaveTransferGetCount (*C function*), 522
drivers_MCX_MCXW_MCXW727C.LPI2C_SlaveTransferHandleIRQ (*C function*), 523
drivers_MCX_MCXW_MCXW727C.LPI2C_SlaveTransferNonBlocking (*C function*), 522
drivers_MCX_MCXW_MCXW727C.LPI2C_SlaveTransmitAck (*C function*), 520
drivers_MCX_MCXW_MCXW727C.lpit_chnl_params_t (*C type*), 535
drivers_MCX_MCXW_MCXW727C.lpit_chnl_t (*C type*), 534
drivers_MCX_MCXW_MCXW727C.LPIT_ClearStatusFlags (*C function*), 530
drivers_MCX_MCXW_MCXW727C.LPIT_CLOCKS (*C macro*), 195
drivers_MCX_MCXW_MCXW727C.lpit_config_t (*C type*), 535
drivers_MCX_MCXW_MCXW727C.LPIT_Deinit (*C function*), 529
drivers_MCX_MCXW_MCXW727C.LPIT_DisableInterrupts (*C function*), 530
drivers_MCX_MCXW_MCXW727C.LPIT_EnableInterrupts (*C function*), 530
drivers_MCX_MCXW_MCXW727C.LPIT_GetCurrentTimerCount (*C function*), 531
drivers_MCX_MCXW_MCXW727C.LPIT_GetDefaultConfig (*C function*), 529

drivers_MCX_MCXW_MCXW727C.LPIT_GetEnabledInterrupts (*C function*), 530
drivers_MCX_MCXW_MCXW727C.LPIT_GetStatusFlags (*C function*), 530
drivers_MCX_MCXW_MCXW727C.LPIT_Init (*C function*), 529
drivers_MCX_MCXW_MCXW727C.lpit_interrupt_enable_t (*C type*), 535
drivers_MCX_MCXW_MCXW727C.LPIT_Reset (*C function*), 535
drivers_MCX_MCXW_MCXW727C.LPIT_RESET_STATE_DELAY (*C macro*), 535
drivers_MCX_MCXW_MCXW727C.LPIT_ResetStateDelay (*C function*), 535
drivers_MCX_MCXW_MCXW727C.LPIT_SetTimerPeriod (*C function*), 530
drivers_MCX_MCXW_MCXW727C.LPIT_SetTimerValue (*C function*), 531
drivers_MCX_MCXW_MCXW727C.LPIT_SetupChannel (*C function*), 529
drivers_MCX_MCXW_MCXW727C.LPIT_StartTimer (*C function*), 531
drivers_MCX_MCXW_MCXW727C.lpit_status_flags_t (*C type*), 535
drivers_MCX_MCXW_MCXW727C.LPIT_StopTimer (*C function*), 531
drivers_MCX_MCXW_MCXW727C.lpit_timer_modes_t (*C type*), 534
drivers_MCX_MCXW_MCXW727C.lpit_trigger_select_t (*C type*), 535
drivers_MCX_MCXW_MCXW727C.lpit_trigger_source_t (*C type*), 535
drivers_MCX_MCXW_MCXW727C.LPSPI_CheckTransferArgument (*C function*), 540
drivers_MCX_MCXW_MCXW727C.LPSPI_ClearStatusFlags (*C function*), 538
drivers_MCX_MCXW_MCXW727C.lpspi_clock_phase_t (*C type*), 554
drivers_MCX_MCXW_MCXW727C.lpspi_clock_polarity_t (*C type*), 554
drivers_MCX_MCXW_MCXW727C.LPSPI_CLOCKS (*C macro*), 195
drivers_MCX_MCXW_MCXW727C.lpspi_data_out_config_t (*C type*), 554
drivers_MCX_MCXW_MCXW727C.LPSPI_Deinit (*C function*), 537
drivers_MCX_MCXW_MCXW727C.lpspi_delay_type_t (*C type*), 554
drivers_MCX_MCXW_MCXW727C.LPSPI_DisableDMA (*C function*), 539
drivers_MCX_MCXW_MCXW727C.LPSPI_DisableInterrupts (*C function*), 539
drivers_MCX_MCXW_MCXW727C.LPSPI_DriverIRQHandler (*C function*), 547
drivers_MCX_MCXW_MCXW727C.LPSPI_DUMMY_DATA (*C macro*), 555
drivers_MCX_MCXW_MCXW727C.LPSPI_Enable (*C function*), 537
drivers_MCX_MCXW_MCXW727C.LPSPI_EnableDMA (*C function*), 539
drivers_MCX_MCXW_MCXW727C.LPSPI_EnableInterrupts (*C function*), 539
drivers_MCX_MCXW_MCXW727C.LPSPI_FlushFifo (*C function*), 541
drivers_MCX_MCXW_MCXW727C.LPSPI_GetInstance (*C function*), 537
drivers_MCX_MCXW_MCXW727C.LPSPI_GetRxFifoCount (*C function*), 538
drivers_MCX_MCXW_MCXW727C.LPSPI_GetRxFifoSize (*C function*), 538
drivers_MCX_MCXW_MCXW727C.LPSPI_GetRxRegisterAddress (*C function*), 540
drivers_MCX_MCXW_MCXW727C.LPSPI_GetStatusFlags (*C function*), 538
drivers_MCX_MCXW_MCXW727C.LPSPI_GetTcr (*C function*), 539
drivers_MCX_MCXW_MCXW727C.LPSPI_GetTxFifoCount (*C function*), 538
drivers_MCX_MCXW_MCXW727C.LPSPI_GetTxFifoSize (*C function*), 538
drivers_MCX_MCXW_MCXW727C.LPSPI_GetTxRegisterAddress (*C function*), 539
drivers_MCX_MCXW_MCXW727C.lpspi_host_request_select_t (*C type*), 554
drivers_MCX_MCXW_MCXW727C.LPSPI_IsMaster (*C function*), 541
drivers_MCX_MCXW_MCXW727C.lpspi_master_config_t (*C type*), 555
drivers_MCX_MCXW_MCXW727C.lpspi_master_edma_handle_t (*C type*), 561
drivers_MCX_MCXW_MCXW727C.lpspi_master_edma_transfer_callback_t (*C type*), 561
drivers_MCX_MCXW_MCXW727C.lpspi_master_handle_t (*C type*), 555
drivers_MCX_MCXW_MCXW727C.LPSPI_MASTER_PCS_MASK (*C macro*), 556
drivers_MCX_MCXW_MCXW727C.LPSPI_MASTER_PCS_SHIFT (*C macro*), 556
drivers_MCX_MCXW_MCXW727C.lpspi_master_slave_mode_t (*C type*), 554
drivers_MCX_MCXW_MCXW727C.lpspi_master_transfer_callback_t (*C type*), 555
drivers_MCX_MCXW_MCXW727C.LPSPI_MasterGetDefaultConfig (*C function*), 537
drivers_MCX_MCXW_MCXW727C.LPSPI_MasterInit (*C function*), 536
drivers_MCX_MCXW_MCXW727C.LPSPI_MasterSetBaudRate (*C function*), 542
drivers_MCX_MCXW_MCXW727C.LPSPI_MasterSetDelayScaler (*C function*), 542
drivers_MCX_MCXW_MCXW727C.LPSPI_MasterSetDelayTimes (*C function*), 543
drivers_MCX_MCXW_MCXW727C.LPSPI_MasterTransferAbort (*C function*), 545
drivers_MCX_MCXW_MCXW727C.LPSPI_MasterTransferAbortEDMA (*C function*), 563

drivers_MCX_MCXW_MCXW727C.LPSPI_MasterTransferBlocking (*C function*), 544
drivers_MCX_MCXW_MCXW727C.LPSPI_MasterTransferCreateHandle (*C function*), 544
drivers_MCX_MCXW_MCXW727C.LPSPI_MasterTransferCreateHandleEDMA (*C function*), 561
drivers_MCX_MCXW_MCXW727C.LPSPI_MasterTransferEDMA (*C function*), 562
drivers_MCX_MCXW_MCXW727C.LPSPI_MasterTransferEDMALite (*C function*), 562
drivers_MCX_MCXW_MCXW727C.LPSPI_MasterTransferGetCount (*C function*), 545
drivers_MCX_MCXW_MCXW727C.LPSPI_MasterTransferGetCountEDMA (*C function*), 563
drivers_MCX_MCXW_MCXW727C.LPSPI_MasterTransferHandleIRQ (*C function*), 545
drivers_MCX_MCXW_MCXW727C.LPSPI_MasterTransferNonBlocking (*C function*), 544
drivers_MCX_MCXW_MCXW727C.LPSPI_MasterTransferPrepareEDMALite (*C function*), 562
drivers_MCX_MCXW_MCXW727C.lpspi_match_config_t (*C type*), 554
drivers_MCX_MCXW_MCXW727C.lpspi_pcs_polarity_config_t (*C type*), 554
drivers_MCX_MCXW_MCXW727C.lpspi_pin_config_t (*C type*), 554
drivers_MCX_MCXW_MCXW727C.LPSPI_ReadData (*C function*), 543
drivers_MCX_MCXW_MCXW727C.LPSPI_Reset (*C function*), 537
drivers_MCX_MCXW_MCXW727C.LPSPI_SelectTransferPCS (*C function*), 540
drivers_MCX_MCXW_MCXW727C.LPSPI_SetAllPcsPolarity (*C function*), 541
drivers_MCX_MCXW_MCXW727C.LPSPI_SetDummyData (*C function*), 544
drivers_MCX_MCXW_MCXW727C.LPSPI_SetFifoWatermarks (*C function*), 541
drivers_MCX_MCXW_MCXW727C.LPSPI_SetFrameSize (*C function*), 542
drivers_MCX_MCXW_MCXW727C.LPSPI_SetMasterSlaveMode (*C function*), 540
drivers_MCX_MCXW_MCXW727C.LPSPI_SetPCSContinous (*C function*), 540
drivers_MCX_MCXW_MCXW727C.lpspi_shift_direction_t (*C type*), 554
drivers_MCX_MCXW_MCXW727C.lpspi_slave_config_t (*C type*), 555
drivers_MCX_MCXW_MCXW727C.lpspi_slave_edma_handle_t (*C type*), 561
drivers_MCX_MCXW_MCXW727C.lpspi_slave_edma_transfer_callback_t (*C type*), 561
drivers_MCX_MCXW_MCXW727C.lpspi_slave_handle_t (*C type*), 555
drivers_MCX_MCXW_MCXW727C.LPSPI_SLAVE_PCS_MASK (*C macro*), 556
drivers_MCX_MCXW_MCXW727C.LPSPI_SLAVE_PCS_SHIFT (*C macro*), 556
drivers_MCX_MCXW_MCXW727C.lpspi_slave_transfer_callback_t (*C type*), 555
drivers_MCX_MCXW_MCXW727C.LPSPI_SlaveGetDefaultConfig (*C function*), 537
drivers_MCX_MCXW_MCXW727C.LPSPI_SlaveInit (*C function*), 537
drivers_MCX_MCXW_MCXW727C.LPSPI_SlaveTransferAbort (*C function*), 546
drivers_MCX_MCXW_MCXW727C.LPSPI_SlaveTransferAbortEDMA (*C function*), 564
drivers_MCX_MCXW_MCXW727C.LPSPI_SlaveTransferCreateHandle (*C function*), 545
drivers_MCX_MCXW_MCXW727C.LPSPI_SlaveTransferCreateHandleEDMA (*C function*), 563
drivers_MCX_MCXW_MCXW727C.LPSPI_SlaveTransferEDMA (*C function*), 564
drivers_MCX_MCXW_MCXW727C.LPSPI_SlaveTransferGetCount (*C function*), 546
drivers_MCX_MCXW_MCXW727C.LPSPI_SlaveTransferGetCountEDMA (*C function*), 564
drivers_MCX_MCXW_MCXW727C.LPSPI_SlaveTransferHandleIRQ (*C function*), 546
drivers_MCX_MCXW_MCXW727C.LPSPI_SlaveTransferNonBlocking (*C function*), 545
drivers_MCX_MCXW_MCXW727C.lpspi_transfer_t (*C type*), 555
drivers_MCX_MCXW_MCXW727C.lpspi_transfer_width_t (*C type*), 554
drivers_MCX_MCXW_MCXW727C.LPSPI_WaitTxFifoEmpty (*C function*), 546
drivers_MCX_MCXW_MCXW727C.lpspi_which_pcs_t (*C type*), 554
drivers_MCX_MCXW_MCXW727C.LPSPI_WriteData (*C function*), 543
drivers_MCX_MCXW_MCXW727C.LPTMR_ClearStatusFlags (*C function*), 570
drivers_MCX_MCXW_MCXW727C.lptmr_config_t (*C type*), 573
drivers_MCX_MCXW_MCXW727C.LPTMR_Deinit (*C function*), 569
drivers_MCX_MCXW_MCXW727C.LPTMR_DisableInterrupts (*C function*), 569
drivers_MCX_MCXW_MCXW727C.LPTMR_EnableInterrupts (*C function*), 569
drivers_MCX_MCXW_MCXW727C.LPTMR_EnableTimerDMA (*C function*), 574
drivers_MCX_MCXW_MCXW727C.LPTMR_GetCurrentTimerCount (*C function*), 570
drivers_MCX_MCXW_MCXW727C.LPTMR_GetDefaultConfig (*C function*), 569
drivers_MCX_MCXW_MCXW727C.LPTMR_GetEnabledInterrupts (*C function*), 569
drivers_MCX_MCXW_MCXW727C.LPTMR_GetStatusFlags (*C function*), 569
drivers_MCX_MCXW_MCXW727C.LPTMR_Init (*C function*), 568
drivers_MCX_MCXW_MCXW727C.lptmr_interrupt_enable_t (*C type*), 573

drivers_MCX_MCXW_MCXW727C.lptmr_pin_polarity_t (*C type*), 573
drivers_MCX_MCXW_MCXW727C.lptmr_pin_select_t (*C type*), 573
drivers_MCX_MCXW_MCXW727C.lptmr_prescaler_clock_select_t (*C type*), 573
drivers_MCX_MCXW_MCXW727C.lptmr_prescaler_glitch_value_t (*C type*), 573
drivers_MCX_MCXW_MCXW727C.LPTMR_SetTimerPeriod (*C function*), 570
drivers_MCX_MCXW_MCXW727C.LPTMR_StartTimer (*C function*), 570
drivers_MCX_MCXW_MCXW727C.lptmr_status_flags_t (*C type*), 573
drivers_MCX_MCXW_MCXW727C.LPTMR_StopTimer (*C function*), 570
drivers_MCX_MCXW_MCXW727C.lptmr_timer_mode_t (*C type*), 573
drivers_MCX_MCXW_MCXW727C.LPUART_ClearStatusFlags (*C function*), 578
drivers_MCX_MCXW_MCXW727C.LPUART_CLOCKS (*C macro*), 195
drivers_MCX_MCXW_MCXW727C.lpuart_config_t (*C type*), 592
drivers_MCX_MCXW_MCXW727C.lpuart_data_bits_t (*C type*), 592
drivers_MCX_MCXW_MCXW727C.LPUART_Deinit (*C function*), 575
drivers_MCX_MCXW_MCXW727C.LPUART_DisableInterrupts (*C function*), 578
drivers_MCX_MCXW_MCXW727C.LPUART_DriverIRQHandler (*C function*), 586
drivers_MCX_MCXW_MCXW727C.lpuart_edma_handle_t (*C type*), 598
drivers_MCX_MCXW_MCXW727C.lpuart_edma_transfer_callback_t (*C type*), 598
drivers_MCX_MCXW_MCXW727C.LPUART_Enable9bitMode (*C function*), 576
drivers_MCX_MCXW_MCXW727C.LPUART_EnableInterrupts (*C function*), 578
drivers_MCX_MCXW_MCXW727C.LPUART_EnableMatchAddress (*C function*), 577
drivers_MCX_MCXW_MCXW727C.LPUART_EnableRx (*C function*), 580
drivers_MCX_MCXW_MCXW727C.LPUART_EnableRxDMA (*C function*), 579
drivers_MCX_MCXW_MCXW727C.LPUART_EnableTx (*C function*), 580
drivers_MCX_MCXW_MCXW727C.LPUART_EnableTxDMA (*C function*), 579
drivers_MCX_MCXW_MCXW727C.LPUART_GetDataRegisterAddress (*C function*), 579
drivers_MCX_MCXW_MCXW727C.LPUART_GetDefaultConfig (*C function*), 576
drivers_MCX_MCXW_MCXW727C.LPUART_GetEnabledInterrupts (*C function*), 579
drivers_MCX_MCXW_MCXW727C.LPUART_GetInstance (*C function*), 580
drivers_MCX_MCXW_MCXW727C.LPUART_GetRxFifoCount (*C function*), 580
drivers_MCX_MCXW_MCXW727C.LPUART_GetStatusFlags (*C function*), 577
drivers_MCX_MCXW_MCXW727C.LPUART_GetTxFifoCount (*C function*), 581
drivers_MCX_MCXW_MCXW727C.lpuart_handle_t (*C type*), 592
drivers_MCX_MCXW_MCXW727C.lpuart_idle_config_t (*C type*), 592
drivers_MCX_MCXW_MCXW727C.lpuart_idle_type_select_t (*C type*), 592
drivers_MCX_MCXW_MCXW727C.LPUART_Init (*C function*), 575
drivers_MCX_MCXW_MCXW727C.lpuart_isr_t (*C type*), 592
drivers_MCX_MCXW_MCXW727C.lpuart_parity_mode_t (*C type*), 591
drivers_MCX_MCXW_MCXW727C.LPUART_ReadBlocking (*C function*), 582
drivers_MCX_MCXW_MCXW727C.LPUART_ReadBlocking16bit (*C function*), 582
drivers_MCX_MCXW_MCXW727C.LPUART_ReadByte (*C function*), 580
drivers_MCX_MCXW_MCXW727C.LPUART_ReceiveEDMA (*C function*), 597
drivers_MCX_MCXW_MCXW727C.LPUART_SendAddress (*C function*), 581
drivers_MCX_MCXW_MCXW727C.LPUART_SendEDMA (*C function*), 596
drivers_MCX_MCXW_MCXW727C.LPUART_SetBaudRate (*C function*), 576
drivers_MCX_MCXW_MCXW727C.LPUART_SetMatchAddress (*C function*), 576
drivers_MCX_MCXW_MCXW727C.LPUART_SetRxFifoWatermark (*C function*), 577
drivers_MCX_MCXW_MCXW727C.LPUART_SetTxFifoWatermark (*C function*), 577
drivers_MCX_MCXW_MCXW727C.LPUART_SoftwareReset (*C function*), 575
drivers_MCX_MCXW_MCXW727C.lpuart_stop_bit_count_t (*C type*), 592
drivers_MCX_MCXW_MCXW727C.lpuart_transfer_callback_t (*C type*), 592
drivers_MCX_MCXW_MCXW727C.lpuart_transfer_t (*C type*), 592
drivers_MCX_MCXW_MCXW727C.LPUART_TransferAbortReceive (*C function*), 585
drivers_MCX_MCXW_MCXW727C.LPUART_TransferAbortReceiveEDMA (*C function*), 597
drivers_MCX_MCXW_MCXW727C.LPUART_TransferAbortSend (*C function*), 584
drivers_MCX_MCXW_MCXW727C.LPUART_TransferAbortSendEDMA (*C function*), 597
drivers_MCX_MCXW_MCXW727C.LPUART_TransferCreateHandle (*C function*), 582
drivers_MCX_MCXW_MCXW727C.LPUART_TransferCreateHandleEDMA (*C function*), 596

drivers_MCX_MCXW_MCXW727C.LPUART_TransferEdmaHandleIRQ (*C function*), 598
drivers_MCX_MCXW_MCXW727C.LPUART_TransferEnable16Bit (*C function*), 577
drivers_MCX_MCXW_MCXW727C.LPUART_TransferGetReceiveCount (*C function*), 585
drivers_MCX_MCXW_MCXW727C.LPUART_TransferGetReceiveCountEDMA (*C function*), 598
drivers_MCX_MCXW_MCXW727C.LPUART_TransferGetRxRingBufferLength (*C function*), 584
drivers_MCX_MCXW_MCXW727C.LPUART_TransferGetSendCount (*C function*), 584
drivers_MCX_MCXW_MCXW727C.LPUART_TransferGetSendCountEDMA (*C function*), 597
drivers_MCX_MCXW_MCXW727C.LPUART_TransferHandleErrorIRQ (*C function*), 586
drivers_MCX_MCXW_MCXW727C.LPUART_TransferHandleIRQ (*C function*), 585
drivers_MCX_MCXW_MCXW727C.LPUART_TransferReceiveNonBlocking (*C function*), 584
drivers_MCX_MCXW_MCXW727C.LPUART_TransferSendNonBlocking (*C function*), 583
drivers_MCX_MCXW_MCXW727C.LPUART_TransferStartRingBuffer (*C function*), 583
drivers_MCX_MCXW_MCXW727C.LPUART_TransferStopRingBuffer (*C function*), 584
drivers_MCX_MCXW_MCXW727C.lpuart_transmit_cts_config_t (*C type*), 592
drivers_MCX_MCXW_MCXW727C.lpuart_transmit_cts_source_t (*C type*), 592
drivers_MCX_MCXW_MCXW727C.lpuart_transmit_rts_polarity_t (*C type*), 592
drivers_MCX_MCXW_MCXW727C.LPUART_WriteBlocking (*C function*), 581
drivers_MCX_MCXW_MCXW727C.LPUART_WriteBlocking16bit (*C function*), 581
drivers_MCX_MCXW_MCXW727C.LPUART_WriteByte (*C function*), 580
drivers_MCX_MCXW_MCXW727C.LTC_AES_BLOCK_SIZE (*C macro*), 605
drivers_MCX_MCXW_MCXW727C.LTC_AES_CryptCtr (*C function*), 603
drivers_MCX_MCXW_MCXW727C.LTC_AES_DecryptCbc (*C function*), 602
drivers_MCX_MCXW_MCXW727C.LTC_AES_DecryptCtr (*C macro*), 605
drivers_MCX_MCXW_MCXW727C.LTC_AES_DecryptEcb (*C function*), 601
drivers_MCX_MCXW_MCXW727C.LTC_AES_DecryptTagCcm (*C function*), 605
drivers_MCX_MCXW_MCXW727C.LTC_AES_DecryptTagGcm (*C function*), 604
drivers_MCX_MCXW_MCXW727C.LTC_AES_EncryptCbc (*C function*), 602
drivers_MCX_MCXW_MCXW727C.LTC_AES_EncryptCtr (*C macro*), 605
drivers_MCX_MCXW_MCXW727C.LTC_AES_EncryptEcb (*C function*), 601
drivers_MCX_MCXW_MCXW727C.LTC_AES_EncryptTagCcm (*C function*), 604
drivers_MCX_MCXW_MCXW727C.LTC_AES_EncryptTagGcm (*C function*), 603
drivers_MCX_MCXW_MCXW727C.LTC_AES_IV_SIZE (*C macro*), 605
drivers_MCX_MCXW_MCXW727C.ltc_aes_key_t (*C type*), 601
drivers_MCX_MCXW_MCXW727C.LTC_Deinit (*C function*), 600
drivers_MCX_MCXW_MCXW727C.LTC_DES2_DecryptCbc (*C function*), 609
drivers_MCX_MCXW_MCXW727C.LTC_DES2_DecryptCfb (*C function*), 610
drivers_MCX_MCXW_MCXW727C.LTC_DES2_DecryptEcb (*C function*), 608
drivers_MCX_MCXW_MCXW727C.LTC_DES2_DecryptOfb (*C function*), 611
drivers_MCX_MCXW_MCXW727C.LTC_DES2_EncryptCbc (*C function*), 609
drivers_MCX_MCXW_MCXW727C.LTC_DES2_EncryptCfb (*C function*), 610
drivers_MCX_MCXW_MCXW727C.LTC_DES2_EncryptEcb (*C function*), 608
drivers_MCX_MCXW_MCXW727C.LTC_DES2_EncryptOfb (*C function*), 610
drivers_MCX_MCXW_MCXW727C.LTC_DES3_DecryptCbc (*C function*), 612
drivers_MCX_MCXW_MCXW727C.LTC_DES3_DecryptCfb (*C function*), 613
drivers_MCX_MCXW_MCXW727C.LTC_DES3_DecryptEcb (*C function*), 611
drivers_MCX_MCXW_MCXW727C.LTC_DES3_DecryptOfb (*C function*), 614
drivers_MCX_MCXW_MCXW727C.LTC_DES3_EncryptCbc (*C function*), 612
drivers_MCX_MCXW_MCXW727C.LTC_DES3_EncryptCfb (*C function*), 613
drivers_MCX_MCXW_MCXW727C.LTC_DES3_EncryptEcb (*C function*), 611
drivers_MCX_MCXW_MCXW727C.LTC_DES3_EncryptOfb (*C function*), 613
drivers_MCX_MCXW_MCXW727C.LTC_DES_DecryptCbc (*C function*), 606
drivers_MCX_MCXW_MCXW727C.LTC_DES_DecryptCfb (*C function*), 607
drivers_MCX_MCXW_MCXW727C.LTC_DES_DecryptEcb (*C function*), 606
drivers_MCX_MCXW_MCXW727C.LTC_DES_DecryptOfb (*C function*), 608
drivers_MCX_MCXW_MCXW727C.LTC_DES_EncryptCbc (*C function*), 606
drivers_MCX_MCXW_MCXW727C.LTC_DES_EncryptCfb (*C function*), 607
drivers_MCX_MCXW_MCXW727C.LTC_DES_EncryptEcb (*C function*), 606
drivers_MCX_MCXW_MCXW727C.LTC_DES_EncryptOfb (*C function*), 607

drivers_MCX_MCXW_MCXW727C.LTC_DES_IV_SIZE (*C macro*), 614
drivers_MCX_MCXW_MCXW727C.LTC_DES_KEY_SIZE (*C macro*), 614
drivers_MCX_MCXW_MCXW727C.LTC_HASH (*C function*), 616
drivers_MCX_MCXW_MCXW727C.ltc_hash_algo_t (*C type*), 615
drivers_MCX_MCXW_MCXW727C.LTC_HASH_CTX_SIZE (*C macro*), 616
drivers_MCX_MCXW_MCXW727C.ltc_hash_ctx_t (*C type*), 615
drivers_MCX_MCXW_MCXW727C.LTC_HASH_Finish (*C function*), 615
drivers_MCX_MCXW_MCXW727C.LTC_HASH_Init (*C function*), 615
drivers_MCX_MCXW_MCXW727C.LTC_HASH_Update (*C function*), 615
drivers_MCX_MCXW_MCXW727C.LTC_Init (*C function*), 600
drivers_MCX_MCXW_MCXW727C.LTC_KEY_REGISTER_READABLE (*C macro*), 605
drivers_MCX_MCXW_MCXW727C.LTC_PKHA_CompareBigNum (*C function*), 617
drivers_MCX_MCXW_MCXW727C.ltc_pkha_ecc_point_t (*C type*), 617
drivers_MCX_MCXW_MCXW727C.LTC_PKHA_ECC_PointAdd (*C function*), 623
drivers_MCX_MCXW_MCXW727C.LTC_PKHA_ECC_PointDouble (*C function*), 623
drivers_MCX_MCXW_MCXW727C.LTC_PKHA_ECC_PointMul (*C function*), 624
drivers_MCX_MCXW_MCXW727C.ltc_pkha_f2m_t (*C type*), 617
drivers_MCX_MCXW_MCXW727C.LTC_PKHA_GCD (*C function*), 622
drivers_MCX_MCXW_MCXW727C.LTC_PKHA_ModAdd (*C function*), 619
drivers_MCX_MCXW_MCXW727C.LTC_PKHA_ModExp (*C function*), 621
drivers_MCX_MCXW_MCXW727C.LTC_PKHA_ModInv (*C function*), 621
drivers_MCX_MCXW_MCXW727C.LTC_PKHA_ModMul (*C function*), 620
drivers_MCX_MCXW_MCXW727C.LTC_PKHA_ModR2 (*C function*), 622
drivers_MCX_MCXW_MCXW727C.LTC_PKHA_ModRed (*C function*), 621
drivers_MCX_MCXW_MCXW727C.LTC_PKHA_ModSub1 (*C function*), 619
drivers_MCX_MCXW_MCXW727C.LTC_PKHA_ModSub2 (*C function*), 619
drivers_MCX_MCXW_MCXW727C.ltc_pkha_montgomery_form_t (*C type*), 617
drivers_MCX_MCXW_MCXW727C.LTC_PKHA_MontgomeryToNormal (*C function*), 618
drivers_MCX_MCXW_MCXW727C.LTC_PKHA_NormalToMontgomery (*C function*), 617
drivers_MCX_MCXW_MCXW727C.LTC_PKHA_PrimalityTest (*C function*), 623
drivers_MCX_MCXW_MCXW727C.ltc_pkha_timing_t (*C type*), 617
drivers_MCX_MCXW_MCXW727C.LTC_SetDpaMaskSeed (*C function*), 600
drivers_MCX_MCXW_MCXW727C.MAKE_MRCC_REGADDR (*C macro*), 196
drivers_MCX_MCXW_MCXW727C.MAKE_STATUS (*C macro*), 464
drivers_MCX_MCXW_MCXW727C.MAKE_VERSION (*C macro*), 464
drivers_MCX_MCXW_MCXW727C.MAX (*C macro*), 449
drivers_MCX_MCXW_MCXW727C.mcm_buffer_fault_attribute_t (*C type*), 626
drivers_MCX_MCXW_MCXW727C.MCM_BypassFixedCodeCacheMap (*C function*), 627
drivers_MCX_MCXW_MCXW727C.MCM_ClearCacheWriteBufferErroStatus (*C function*), 626
drivers_MCX_MCXW_MCXW727C.MCM_ClearCodeBusCache (*C function*), 628
drivers_MCX_MCXW_MCXW727C.MCM_DisableInterruptStatus (*C function*), 626
drivers_MCX_MCXW_MCXW727C.MCM_EnableCacheParityReporting (*C function*), 628
drivers_MCX_MCXW_MCXW727C.MCM_EnableCodeBusCache (*C function*), 627
drivers_MCX_MCXW_MCXW727C.MCM_EnableCodeCacheWriteBuffer (*C function*), 627
drivers_MCX_MCXW_MCXW727C.MCM_EnableCrossbarRoundRobin (*C function*), 626
drivers_MCX_MCXW_MCXW727C.MCM_EnableInterruptStatus (*C function*), 626
drivers_MCX_MCXW_MCXW727C.MCM_EnablePcParity (*C function*), 628
drivers_MCX_MCXW_MCXW727C.MCM_EnablePcParityFaultReport (*C function*), 628
drivers_MCX_MCXW_MCXW727C.MCM_ForceCodeCacheToNoAllocation (*C function*), 627
drivers_MCX_MCXW_MCXW727C.MCM_GetBufferFaultAddress (*C function*), 626
drivers_MCX_MCXW_MCXW727C.MCM_GetBufferFaultAttribute (*C function*), 626
drivers_MCX_MCXW_MCXW727C.MCM_GetBufferFaultData (*C function*), 627
drivers_MCX_MCXW_MCXW727C.MCM_GetInterruptStatus (*C function*), 626
drivers_MCX_MCXW_MCXW727C.MCM_GetLmemFaultAddress (*C function*), 628
drivers_MCX_MCXW_MCXW727C.MCM_GetLmemFaultAttribute (*C function*), 628
drivers_MCX_MCXW_MCXW727C.MCM_GetLmemFaultData (*C function*), 629
drivers_MCX_MCXW_MCXW727C.MCM_ISCR_STAT_MASK (*C macro*), 629

drivers_MCX_MCXW_MCXW727C.MCM_LimitCodeCachePeripheralWriteBuffering (*C function*), 627
drivers_MCX_MCXW_MCXW727C.mcm_lmem_fault_attribute_t (*C type*), 626
drivers_MCX_MCXW_MCXW727C.MCM_LMFATR_BUFF_MASK (*C macro*), 629
drivers_MCX_MCXW_MCXW727C.MCM_LMFATR_CACH_MASK (*C macro*), 629
drivers_MCX_MCXW_MCXW727C.MCM_LMFATR_MODE_MASK (*C macro*), 629
drivers_MCX_MCXW_MCXW727C.MCM_LMFATR_TYPE_MASK (*C macro*), 629
drivers_MCX_MCXW_MCXW727C.MCM_LockConfigState (*C function*), 628
drivers_MCX_MCXW_MCXW727C.MCUX_DEPRECATED (*C macro*), 452
drivers_MCX_MCXW_MCXW727C.MCUX_DEPRECATED_MACRO (*C macro*), 452
drivers_MCX_MCXW_MCXW727C.MCUX_EXPERIMENTAL (*C macro*), 452
drivers_MCX_MCXW_MCXW727C.MCUX_EXPERIMENTAL_MACRO (*C macro*), 452
drivers_MCX_MCXW_MCXW727C.MCUX_MASK_INVERT_8 (*C macro*), 449
drivers_MCX_MCXW_MCXW727C.MCUX_MASK_INVERT_16 (*C macro*), 449
drivers_MCX_MCXW_MCXW727C.MCUX_MASK_INVERT_32 (*C macro*), 449
drivers_MCX_MCXW_MCXW727C.MCUX_RAMFUNC (*C macro*), 452
drivers_MCX_MCXW_MCXW727C.MCUX_REG_BIT_CLEAR8 (*C macro*), 450
drivers_MCX_MCXW_MCXW727C.MCUX_REG_BIT_CLEAR16 (*C macro*), 450
drivers_MCX_MCXW_MCXW727C.MCUX_REG_BIT_CLEAR32 (*C macro*), 450
drivers_MCX_MCXW_MCXW727C.MCUX_REG_BIT_GET8 (*C macro*), 450
drivers_MCX_MCXW_MCXW727C.MCUX_REG_BIT_GET16 (*C macro*), 450
drivers_MCX_MCXW_MCXW727C.MCUX_REG_BIT_GET32 (*C macro*), 450
drivers_MCX_MCXW_MCXW727C.MCUX_REG_BIT_SET8 (*C macro*), 450
drivers_MCX_MCXW_MCXW727C.MCUX_REG_BIT_SET16 (*C macro*), 450
drivers_MCX_MCXW_MCXW727C.MCUX_REG_BIT_SET32 (*C macro*), 450
drivers_MCX_MCXW_MCXW727C.MCUX_REG_MODIFY8 (*C macro*), 450
drivers_MCX_MCXW_MCXW727C.MCUX_REG_MODIFY16 (*C macro*), 450
drivers_MCX_MCXW_MCXW727C.MCUX_REG_MODIFY32 (*C macro*), 450
drivers_MCX_MCXW_MCXW727C.MCUX_REG_READ8 (*C macro*), 449
drivers_MCX_MCXW_MCXW727C.MCUX_REG_READ16 (*C macro*), 449
drivers_MCX_MCXW_MCXW727C.MCUX_REG_READ32 (*C macro*), 450
drivers_MCX_MCXW_MCXW727C.MCUX_REG_WRITE8 (*C macro*), 449
drivers_MCX_MCXW_MCXW727C.MCUX_REG_WRITE16 (*C macro*), 449
drivers_MCX_MCXW_MCXW727C.MCUX_REG_WRITE32 (*C macro*), 449
drivers_MCX_MCXW_MCXW727C.MIN (*C macro*), 449
drivers_MCX_MCXW_MCXW727C.MSCM_GetSecureIrq (*C function*), 631
drivers_MCX_MCXW_MCXW727C.MSCM_GetUID (*C function*), 631
drivers_MCX_MCXW_MCXW727C.MSCM_SetSecureIrqParameter (*C function*), 631
drivers_MCX_MCXW_MCXW727C.mscm_uid_t (*C type*), 631
drivers_MCX_MCXW_MCXW727C.MSDK_HAS_DWT_CYCCNT (*C macro*), 464
drivers_MCX_MCXW_MCXW727C.MSDK_INVALID_IRQ_HANDLER (*C macro*), 464
drivers_MCX_MCXW_MCXW727C.MSDK_REG_NONSECURE_ADDR (*C macro*), 464
drivers_MCX_MCXW_MCXW727C.MSDK_REG_SECURE_ADDR (*C macro*), 464
drivers_MCX_MCXW_MCXW727C.MSEC_TO_COUNT (*C macro*), 451
drivers_MCX_MCXW_MCXW727C.msf1_config_t (*C type*), 248
drivers_MCX_MCXW_MCXW727C.MU1_BUSY_POLL_COUNT (*C macro*), 645
drivers_MCX_MCXW_MCXW727C.MU_BootOtherCore (*C function*), 639
drivers_MCX_MCXW_MCXW727C.MU_ClearGeneralPurposeStatusFlags (*C function*), 638
drivers_MCX_MCXW_MCXW727C.MU_ClearStatusFlags (*C function*), 635
drivers_MCX_MCXW_MCXW727C.MU_CLOCKS (*C macro*), 196
drivers_MCX_MCXW_MCXW727C.MU_CORE_FLAG (*C macro*), 644
drivers_MCX_MCXW_MCXW727C.MU_CORE_INTR (*C macro*), 644
drivers_MCX_MCXW_MCXW727C.MU_Deinit (*C function*), 631
drivers_MCX_MCXW_MCXW727C.MU_DisableGeneralPurposeInterrupts (*C function*), 637
drivers_MCX_MCXW_MCXW727C.MU_DisableInterrupts (*C function*), 636
drivers_MCX_MCXW_MCXW727C.MU_EnableGeneralPurposeInterrupts (*C function*), 637
drivers_MCX_MCXW_MCXW727C.MU_EnableInterrupts (*C function*), 636
drivers_MCX_MCXW_MCXW727C.mu_general_purpose_interrupt_t (*C type*), 644

drivers_MCX_MCXW_MCXW727C.MU_GET_CORE_FLAG (*C macro*), 644
drivers_MCX_MCXW_MCXW727C.MU_GET_CORE_INTR (*C macro*), 644
drivers_MCX_MCXW_MCXW727C.MU_GET_GI_FLAG (*C macro*), 644
drivers_MCX_MCXW_MCXW727C.MU_GET_GI_INTR (*C macro*), 644
drivers_MCX_MCXW_MCXW727C.MU_GET_RX_FLAG (*C macro*), 644
drivers_MCX_MCXW_MCXW727C.MU_GET_RX_INTR (*C macro*), 644
drivers_MCX_MCXW_MCXW727C.MU_GET_STAT_FLAG (*C macro*), 644
drivers_MCX_MCXW_MCXW727C.MU_GET_TX_FLAG (*C macro*), 644
drivers_MCX_MCXW_MCXW727C.MU_GET_TX_INTR (*C macro*), 644
drivers_MCX_MCXW_MCXW727C.MU_GetFlags (*C function*), 634
drivers_MCX_MCXW_MCXW727C.MU_GetGeneralPurposeStatusFlags (*C function*), 637
drivers_MCX_MCXW_MCXW727C.MU_GetInterruptsPending (*C function*), 635
drivers_MCX_MCXW_MCXW727C.MU_GetRxStatusFlags (*C function*), 638
drivers_MCX_MCXW_MCXW727C.MU_GetStatusFlags (*C function*), 634
drivers_MCX_MCXW_MCXW727C.MU_GI_FLAG (*C macro*), 644
drivers_MCX_MCXW_MCXW727C.MU_GI_INTR (*C macro*), 644
drivers_MCX_MCXW_MCXW727C.MU_HardwareResetOtherCore (*C function*), 640
drivers_MCX_MCXW_MCXW727C.mu_hdr (*C struct*), 240
drivers_MCX_MCXW_MCXW727C.mu_hdr_t (*C type*), 237
drivers_MCX_MCXW_MCXW727C.MU_HoldOtherCoreReset (*C function*), 639
drivers_MCX_MCXW_MCXW727C.MU_Init (*C function*), 631
drivers_MCX_MCXW_MCXW727C.MU_MISC_INTR (*C macro*), 644
drivers_MCX_MCXW_MCXW727C.mu_msg_reg_index_t (*C type*), 644
drivers_MCX_MCXW_MCXW727C.MU_ReceiveMsg (*C function*), 633
drivers_MCX_MCXW_MCXW727C.MU_ReceiveMsgNonBlocking (*C function*), 632
drivers_MCX_MCXW_MCXW727C.MU_ReceiveMsgTimeout (*C function*), 632
drivers_MCX_MCXW_MCXW727C.MU_ResetBothSides (*C function*), 639
drivers_MCX_MCXW_MCXW727C.MU_RX_FLAG (*C macro*), 644
drivers_MCX_MCXW_MCXW727C.MU_RX_INTR (*C macro*), 644
drivers_MCX_MCXW_MCXW727C.MU_SendMsg (*C function*), 632
drivers_MCX_MCXW_MCXW727C.MU_SendMsgNonBlocking (*C function*), 631
drivers_MCX_MCXW_MCXW727C.MU_SetFlags (*C function*), 634
drivers_MCX_MCXW_MCXW727C.MU_SetFlagsNonBlocking (*C function*), 633
drivers_MCX_MCXW_MCXW727C.MU_STAT_FLAG (*C macro*), 644
drivers_MCX_MCXW_MCXW727C.MU_TriggerGeneralPurposeInterrupts (*C function*), 638
drivers_MCX_MCXW_MCXW727C.MU_TriggerInterrupts (*C function*), 636
drivers_MCX_MCXW_MCXW727C.MU_TX_FLAG (*C macro*), 644
drivers_MCX_MCXW_MCXW727C.MU_TX_INTR (*C macro*), 644
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kCCM32K_32kFroActiveStatusFlag (*C enumerator*), 163
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kCCM32K_32kOscActiveStatusFlag (*C enumerator*), 163
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kCCM32K_32kOscReadyStatusFlag (*C enumerator*), 163
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kCCM32K_ClockDetectStatusFlag (*C enumerator*), 163
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kEDMA_AttributeOutput (*C enumerator*), 231
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kEDMA_DestinationAddressErrorFlag (*C enumerator*), 230
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kEDMA_DestinationBusErrorFlag (*C enumerator*), 230
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kEDMA_DestinationOffsetErrorFlag (*C enumerator*), 230
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kEDMA_DoneFlag (*C enumerator*), 230
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kEDMA_ErrorChannelFlag (*C enumerator*), 231

drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kEDMA_ErrorFlag (*C enumerator*), 230
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kEDMA_InterruptFlag (*C enumerator*), 230
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kEDMA_MasterId (*C enumerator*), 231
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kEDMA_NbytesErrorFlag (*C enumerator*), 230
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kEDMA_PrivilegedAccessLevel (*C enumerator*), 231
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kEDMA_ScatterGatherErrorFlag (*C enumerator*), 230
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kEDMA_SourceAddressErrorFlag (*C enumerator*), 231
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kEDMA_SourceBusErrorFlag (*C enumerator*), 230
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kEDMA_SourceOffsetErrorFlag (*C enumerator*), 231
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kEDMA_TransferCanceledFlag (*C enumerator*), 231
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kEDMA_ValidFlag (*C enumerator*), 231
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kFLEXCAN_RxFifoFrameAvlFlag (*C enumerator*), 281
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kFLEXCAN_RxFifoOverflowFlag (*C enumerator*), 281
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kFLEXCAN_RxFifoWarningFlag (*C enumerator*), 281
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kFLEXIO_I2S_RxDataRegFullFlag (*C enumerator*), 351
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kFLEXIO_I2S_RxDataRegFullInterruptEnable (*C enumerator*), 351
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kFLEXIO_I2S_TxDataRegEmptyFlag (*C enumerator*), 351
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kFLEXIO_I2S_TxDataRegEmptyInterruptEnable (*C enumerator*), 350
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kMCM_CacheWriteBuffer (*C enumerator*), 625
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kMCM_FPUDivideByZero (*C enumerator*), 625
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kMCM_FPUInexact (*C enumerator*), 625
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kMCM_FPUInputDenormalInterrupt (*C enumerator*), 626
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kMCM_FPUInvalidOperation (*C enumerator*), 625
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kMCM_FPUOverflow (*C enumerator*), 625
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kMCM_FPUUnderflow (*C enumerator*), 625
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kMCM_ParityError (*C enumerator*), 625
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kSCG_FircDisable (*C enumerator*), 189
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kSCG_FircEnable (*C enumerator*), 189
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kSCG_FircEnableInSleep (*C enumerator*), 189
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kSCG_SoscDisable (*C enumerator*),

188
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kSCG_SoscEnable (*C enumerator*), 188
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kSCG_SoscEnableInSleep (*C enumerator*), 188
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_Busy (*C enumerator*), 461
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_EDMA_Busy (*C enumerator*), 232
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_EDMA_QueueFull (*C enumerator*), 232
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_ELEMU_AgumentOutOfRange (*C enumerator*), 236
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_ELEMU_Busy (*C enumerator*), 237
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_ELEMU_InvalidArgument (*C enumerator*), 236
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_ELEMU_RequestTimeout (*C enumerator*), 236
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_Fail (*C enumerator*), 461
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_FLEXCAN_ErrorStatus (*C enumerator*), 271
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_FLEXCAN_MemoryError (*C enumerator*), 272
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_FLEXCAN_RxBusy (*C enumerator*), 271
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_FLEXCAN_RxFifoBusy (*C enumerator*), 271
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_FLEXCAN_RxFifoDisabled (*C enumerator*), 271
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_FLEXCAN_RxFifoIdle (*C enumerator*), 271
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_FLEXCAN_RxFifoOverflow (*C enumerator*), 271
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_FLEXCAN_RxFifoUnderflow (*C enumerator*), 272
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_FLEXCAN_RxFifoWarning (*C enumerator*), 271
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_FLEXCAN_RxIdle (*C enumerator*), 271
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_FLEXCAN_RxOverflow (*C enumerator*), 271
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_FLEXCAN_RxRemote (*C enumerator*), 272
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_FLEXCAN_TxBusy (*C enumerator*), 271
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_FLEXCAN_TxIdle (*C enumerator*), 271
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_FLEXCAN_TxSwitchToRx (*C enumerator*), 271
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_FLEXCAN_UnHandled (*C enumerator*), 272
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_FLEXCAN_WakeUp (*C enumerator*), 272
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_FLEXIO_I2C_Busy (*C enumerator*), 340
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_FLEXIO_I2C_Idle (*C enumerator*), 340
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_FLEXIO_I2C_Nak (*C enu-*

merator), 340
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_FLEXIO_I2C_Timeout (*C enumerator*), 340
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_FLEXIO_I2S_Error (*C enumerator*), 350
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_FLEXIO_I2S_Idle (*C enumerator*), 350
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_FLEXIO_I2S_QueueFull (*C enumerator*), 350
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_FLEXIO_I2S_RxBusy (*C enumerator*), 350
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_FLEXIO_I2S_Timeout (*C enumerator*), 350
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_FLEXIO_I2S_TxBusy (*C enumerator*), 350
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_FLEXIO_SPI_Busy (*C enumerator*), 363
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_FLEXIO_SPI_Error (*C enumerator*), 363
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_FLEXIO_SPI_Idle (*C enumerator*), 363
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_FLEXIO_SPI_Timeout (*C enumerator*), 364
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_FLEXIO_UART_BaudrateNotSupport (*C enumerator*), 377
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_FLEXIO_UART_ERROR (*C enumerator*), 377
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_FLEXIO_UART_RxBusy (*C enumerator*), 377
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_FLEXIO_UART_RxHardwareOverrun (*C enumerator*), 377
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_FLEXIO_UART_RxIdle (*C enumerator*), 377
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_FLEXIO_UART_RxRingBufferOverrun (*C enumerator*), 377
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_FLEXIO_UART_Timeout (*C enumerator*), 377
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_FLEXIO_UART_TxBusy (*C enumerator*), 376
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_FLEXIO_UART_TxIdle (*C enumerator*), 377
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_I3C_Busy (*C enumerator*), 390
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_I3C_CrcError (*C enumerator*), 391
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_I3C_HdrParityError (*C enumerator*), 391
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_I3C_IBIWon (*C enumerator*), 391
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_I3C_Idle (*C enumerator*), 390
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_I3C_InvalidReq (*C enumerator*), 391
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_I3C_InvalidStart (*C enumerator*), 391
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_I3C_MsgError (*C enumerator*), 391
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_I3C_Nak (*C enumerator*), 390
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_I3C_OverrunError (*C enu-*

merator), 391
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_I3C_ReadFifoError (*C enumerator*), 391
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_I3C_S0S1Error (*C enumerator*), 391
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_I3C_SdrParityError (*C enumerator*), 391
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_I3C_SlaveCountExceed (*C enumerator*), 391
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_I3C_Term (*C enumerator*), 391
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_I3C_Timeout (*C enumerator*), 391
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_I3C_UnderrunError (*C enumerator*), 391
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_I3C_UnderrunNak (*C enumerator*), 391
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_I3C_WriteAbort (*C enumerator*), 390
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_I3C_WriteFifoError (*C enumerator*), 391
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_InvalidArgument (*C enumerator*), 461
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_LPI2C_ArbitrationLost (*C enumerator*), 498
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_LPI2C_BitError (*C enumerator*), 498
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_LPI2C_Busy (*C enumerator*), 498
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_LPI2C_DmaRequestFail (*C enumerator*), 498
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_LPI2C_FifoError (*C enumerator*), 498
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_LPI2C_Idle (*C enumerator*), 498
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_LPI2C_Nak (*C enumerator*), 498
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_LPI2C_NoTransferInProgress (*C enumerator*), 498
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_LPI2C_PinLowTimeout (*C enumerator*), 498
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_LPI2C_Timeout (*C enumerator*), 499
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_LPSPI_Busy (*C enumerator*), 547
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_LPSPI_Error (*C enumerator*), 547
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_LPSPI_Idle (*C enumerator*), 547
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_LPSPI_OutOfRange (*C enumerator*), 547
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_LPSPI_Timeout (*C enumerator*), 547
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_LPUART_BaudrateNotSupport (*C enumerator*), 587
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_LPUART_Error (*C enumerator*), 586
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_LPUART_FlagCannotClearManually

(*C enumerator*), 586
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_LPUART_FramingError (*C enumerator*), 587
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_LPUART_IdleLineDetected (*C enumerator*), 587
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_LPUART_NoiseError (*C enumerator*), 587
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_LPUART_ParityError (*C enumerator*), 587
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_LPUART_RxBusy (*C enumerator*), 586
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_LPUART_RxHardwareOverrun (*C enumerator*), 587
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_LPUART_RxIdle (*C enumerator*), 586
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_LPUART_RxRingBufferOverrun (*C enumerator*), 587
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_LPUART_RxWatermarkTooLarge (*C enumerator*), 586
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_LPUART_Timeout (*C enumerator*), 587
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_LPUART_TxBusy (*C enumerator*), 586
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_LPUART_TxIdle (*C enumerator*), 586
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_LPUART_TxWatermarkTooLarge (*C enumerator*), 586
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_NoData (*C enumerator*), 461
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_NoTransferInProgress (*C enumerator*), 461
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_OutOfRange (*C enumerator*), 461
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_ReadOnly (*C enumerator*), 461
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_SEMA42_Busy (*C enumerator*), 662
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_SEMA42_Reseting (*C enumerator*), 663
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_SFA_CUTClockFreqGreaterThanMaxLimit (*C enumerator*), 671
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_SFA_CUTClockFreqLessThanMinLimit (*C enumerator*), 671
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_SFA_CUTCounterTimeout (*C enumerator*), 671
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_SFA_MeasurementCompleted (*C enumerator*), 671
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_SFA_ReferenceCounterTimeout (*C enumerator*), 671
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_SPC_BandgapModeWrong (*C enumerator*), 682
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_SPC_Busy (*C enumerator*), 681
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_SPC_CORELDOLowDriveStrengthIgnore (*C enumerator*), 682
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_SPC_CORELDOVoltageSetFail (*C enumerator*), 682
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_SPC_CORELDOVoltageWrong (*C enumerator*), 682

drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_SPC_DCDCLowDriveStrengthIgnore (*C enumerator*), 681
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_SPC_DCDCPulseRefreshModeIgnore (*C enumerator*), 681
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_SPC_SYSLDOLowDriveStrengthIgnore (*C enumerator*), 682
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_SPC_SYSLDOOverDriveVoltageFail (*C enumerator*), 682
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_Success (*C enumerator*), 461
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_Timeout (*C enumerator*), 461
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_TRGMUX_Locked (*C enumerator*), 769
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_VBAT_BandgapNotEnabled (*C enumerator*), 770
drivers_MCX_MCXW_MCXW727C.PhonyNameDueToError.kStatus_VBAT_Fro16kNotEnabled (*C enumerator*), 770
drivers_MCX_MCXW_MCXW727C.PORT_ClearAllHighEFTDetectors (*C function*), 648
drivers_MCX_MCXW_MCXW727C.PORT_ClearAllLowEFTDetectors (*C function*), 648
drivers_MCX_MCXW_MCXW727C.PORT_CLOCKS (*C macro*), 195
drivers_MCX_MCXW_MCXW727C.port_digital_filter_clock_source_t (*C type*), 652
drivers_MCX_MCXW_MCXW727C.port_digital_filter_config_t (*C type*), 652
drivers_MCX_MCXW_MCXW727C.PORT_DisableEFTDetectInterrupts (*C function*), 648
drivers_MCX_MCXW_MCXW727C.PORT_EnableEFTDetectInterrupts (*C function*), 648
drivers_MCX_MCXW_MCXW727C.PORT_EnablePinDoubleDriveStrength (*C function*), 647
drivers_MCX_MCXW_MCXW727C.PORT_EnablePinsDigitalFilter (*C function*), 646
drivers_MCX_MCXW_MCXW727C.PORT_GetEFTDetectFlags (*C function*), 647
drivers_MCX_MCXW_MCXW727C.PORT_GetVersionInfo (*C function*), 645
drivers_MCX_MCXW_MCXW727C.port_mux_t (*C type*), 652
drivers_MCX_MCXW_MCXW727C.port_pin_config_t (*C type*), 652
drivers_MCX_MCXW_MCXW727C.PORT_SecletPortVoltageRange (*C function*), 645
drivers_MCX_MCXW_MCXW727C.PORT_SetDigitalFilterConfig (*C function*), 647
drivers_MCX_MCXW_MCXW727C.PORT_SetMultiplePinsConfig (*C function*), 646
drivers_MCX_MCXW_MCXW727C.PORT_SetPinConfig (*C function*), 645
drivers_MCX_MCXW_MCXW727C.PORT_SetPinDriveStrength (*C function*), 647
drivers_MCX_MCXW_MCXW727C.PORT_SetPinMux (*C function*), 646
drivers_MCX_MCXW_MCXW727C.PORT_SetPinPullValue (*C function*), 647
drivers_MCX_MCXW_MCXW727C.port_version_info_t (*C type*), 652
drivers_MCX_MCXW_MCXW727C.port_voltage_range_t (*C type*), 652
drivers_MCX_MCXW_MCXW727C.RAMFUNCTION_SECTION_CODE (*C macro*), 452
drivers_MCX_MCXW_MCXW727C.Read_IFR_Into_MISR (*C function*), 246
drivers_MCX_MCXW_MCXW727C.Read_Into_MISR (*C function*), 246
drivers_MCX_MCXW_MCXW727C.RF_CLOCKS (*C macro*), 195
drivers_MCX_MCXW_MCXW727C.RTC_ClearStatusFlags (*C function*), 656
drivers_MCX_MCXW_MCXW727C.rtc_config_t (*C type*), 660
drivers_MCX_MCXW_MCXW727C.rtc_datetime_t (*C type*), 659
drivers_MCX_MCXW_MCXW727C.RTC_Deinit (*C function*), 654
drivers_MCX_MCXW_MCXW727C.RTC_DisableInterrupts (*C function*), 655
drivers_MCX_MCXW_MCXW727C.RTC_EnableClockOutput (*C function*), 660
drivers_MCX_MCXW_MCXW727C.RTC_EnableInterrupts (*C function*), 655
drivers_MCX_MCXW_MCXW727C.RTC_EnableLPOClock (*C function*), 656
drivers_MCX_MCXW_MCXW727C.RTC_EnableWakeUpPin (*C function*), 660
drivers_MCX_MCXW_MCXW727C.RTC_GetAlarm (*C function*), 655
drivers_MCX_MCXW_MCXW727C.RTC_GetDatetime (*C function*), 655
drivers_MCX_MCXW_MCXW727C.RTC_GetDefaultConfig (*C function*), 654
drivers_MCX_MCXW_MCXW727C.RTC_GetEnabledInterrupts (*C function*), 655
drivers_MCX_MCXW_MCXW727C.RTC_GetMonotonicCounter (*C function*), 656
drivers_MCX_MCXW_MCXW727C.RTC_GetStatusFlags (*C function*), 656
drivers_MCX_MCXW_MCXW727C.RTC_GetTamperTimeSeconds (*C function*), 660

drivers_MCX_MCXW_MCXW727C.RTC_IncrementMonotonicCounter (*C function*), 657
drivers_MCX_MCXW_MCXW727C.RTC_Init (*C function*), 654
drivers_MCX_MCXW_MCXW727C.rtc_interrupt_enable_t (*C type*), 659
drivers_MCX_MCXW_MCXW727C.rtc_osc_cap_load_t (*C type*), 659
drivers_MCX_MCXW_MCXW727C.rtc_pin_config_t (*C type*), 660
drivers_MCX_MCXW_MCXW727C.RTC_Reset (*C function*), 660
drivers_MCX_MCXW_MCXW727C.RTC_SetAlarm (*C function*), 655
drivers_MCX_MCXW_MCXW727C.RTC_SetDatetime (*C function*), 654
drivers_MCX_MCXW_MCXW727C.RTC_SetMonotonicCounter (*C function*), 657
drivers_MCX_MCXW_MCXW727C.RTC_SetOscCapLoad (*C function*), 660
drivers_MCX_MCXW_MCXW727C.RTC_SetTimerSecondsInterruptFrequency (*C function*), 660
drivers_MCX_MCXW_MCXW727C.RTC_StartTimer (*C function*), 656
drivers_MCX_MCXW_MCXW727C.rtc_status_flags_t (*C type*), 659
drivers_MCX_MCXW_MCXW727C.RTC_StopTimer (*C function*), 656
drivers_MCX_MCXW_MCXW727C.rtc_timer_seconds_interrupt_frequency_t (*C type*), 659
drivers_MCX_MCXW_MCXW727C.RTR_EXT (*C macro*), 288
drivers_MCX_MCXW_MCXW727C.RTR_STD_HIGH (*C macro*), 288
drivers_MCX_MCXW_MCXW727C.RTR_STD_LOW (*C macro*), 288
drivers_MCX_MCXW_MCXW727C.s_flexioBases (*C var*), 324
drivers_MCX_MCXW_MCXW727C.s_flexioClocks (*C var*), 324
drivers_MCX_MCXW_MCXW727C.s_lpi2cMasterHandle (*C var*), 499
drivers_MCX_MCXW_MCXW727C.s_lpi2cMasterIsr (*C var*), 499
drivers_MCX_MCXW_MCXW727C.s_lpuartHandle (*C var*), 592
drivers_MCX_MCXW_MCXW727C.s_lpuartIsr (*C var*), 592
drivers_MCX_MCXW_MCXW727C.s_lpuartTxIRQ (*C var*), 592
drivers_MCX_MCXW_MCXW727C.scg_firc_config_t (*C type*), 192
drivers_MCX_MCXW_MCXW727C.scg_firc_range_t (*C type*), 192
drivers_MCX_MCXW_MCXW727C.scg_firc_trim_config_t (*C type*), 192
drivers_MCX_MCXW_MCXW727C.scg_firc_trim_mode_t (*C type*), 192
drivers_MCX_MCXW_MCXW727C.scg_firc_trim_src_t (*C type*), 192
drivers_MCX_MCXW_MCXW727C.scg_rosc_config_t (*C type*), 191
drivers_MCX_MCXW_MCXW727C.scg_rosc_monitor_mode_t (*C type*), 191
drivers_MCX_MCXW_MCXW727C.scg_sosc_config_t (*C type*), 191
drivers_MCX_MCXW_MCXW727C.scg_sosc_monitor_mode_t (*C type*), 191
drivers_MCX_MCXW_MCXW727C.scg_sys_clk_config_t (*C type*), 191
drivers_MCX_MCXW_MCXW727C.scg_sys_clk_div_t (*C type*), 191
drivers_MCX_MCXW_MCXW727C.scg_sys_clk_src_t (*C type*), 191
drivers_MCX_MCXW_MCXW727C.scg_sys_clk_t (*C type*), 191
drivers_MCX_MCXW_MCXW727C.SDK_ALIGN (*C macro*), 451
drivers_MCX_MCXW_MCXW727C.SDK_ATOMIC_LOCAL_ADD (*C macro*), 450
drivers_MCX_MCXW_MCXW727C.SDK_ATOMIC_LOCAL_CLEAR (*C macro*), 450
drivers_MCX_MCXW_MCXW727C.SDK_ATOMIC_LOCAL_CLEAR_AND_SET (*C macro*), 450
drivers_MCX_MCXW_MCXW727C.SDK_ATOMIC_LOCAL_COMPARE_AND_SET (*C macro*), 450
drivers_MCX_MCXW_MCXW727C.SDK_ATOMIC_LOCAL_SET (*C macro*), 450
drivers_MCX_MCXW_MCXW727C.SDK_ATOMIC_LOCAL_SUB (*C macro*), 450
drivers_MCX_MCXW_MCXW727C.SDK_ATOMIC_LOCAL_TEST_AND_SET (*C macro*), 451
drivers_MCX_MCXW_MCXW727C.SDK_ATOMIC_LOCAL_TOGGLE (*C macro*), 450
drivers_MCX_MCXW_MCXW727C.SDK_DelayAtLeastUs (*C function*), 462
drivers_MCX_MCXW_MCXW727C.SDK_DEVICE_MAXIMUM_CPU_CLOCK_FREQUENCY (*C macro*), 194
drivers_MCX_MCXW_MCXW727C.SDK_Free (*C function*), 461
drivers_MCX_MCXW_MCXW727C.SDK_IS_ALIGNED (*C macro*), 451
drivers_MCX_MCXW_MCXW727C.SDK_ISR_EXIT_BARRIER (*C macro*), 451
drivers_MCX_MCXW_MCXW727C.SDK_Malloc (*C function*), 461
drivers_MCX_MCXW_MCXW727C.SDK_SIZEALIGN (*C macro*), 451
drivers_MCX_MCXW_MCXW727C.SDK_SIZEALIGN_DOWN (*C macro*), 451
drivers_MCX_MCXW_MCXW727C.SDK_SIZEALIGN_UP (*C macro*), 451

drivers_MCX_MCXW_MCXW727C.SEMA42_BUSY_POLL_COUNT (*C macro*), 666
drivers_MCX_MCXW_MCXW727C.SEMA42_CLOCKS (*C macro*), 196
drivers_MCX_MCXW_MCXW727C.SEMA42_Deinit (*C function*), 664
drivers_MCX_MCXW_MCXW727C.SEMA42_GATE_NUM_RESET_ALL (*C macro*), 666
drivers_MCX_MCXW_MCXW727C.sema42_gate_status_t (*C type*), 664
drivers_MCX_MCXW_MCXW727C.SEMA42_GATEn (*C macro*), 666
drivers_MCX_MCXW_MCXW727C.SEMA42_GetGateStatus (*C function*), 665
drivers_MCX_MCXW_MCXW727C.SEMA42_Init (*C function*), 664
drivers_MCX_MCXW_MCXW727C.SEMA42_Lock (*C function*), 664
drivers_MCX_MCXW_MCXW727C.SEMA42_ResetAllGates (*C function*), 665
drivers_MCX_MCXW_MCXW727C.SEMA42_ResetGate (*C function*), 665
drivers_MCX_MCXW_MCXW727C.SEMA42_TryLock (*C function*), 664
drivers_MCX_MCXW_MCXW727C.SEMA42_Unlock (*C function*), 665
drivers_MCX_MCXW_MCXW727C.SFA_AbortMeasureSequence (*C function*), 668
drivers_MCX_MCXW_MCXW727C.SFA_CalculateFrequencyOrPeriod (*C function*), 668
drivers_MCX_MCXW_MCXW727C.sfa_callback_t (*C type*), 674
drivers_MCX_MCXW_MCXW727C.SFA_ClearStatusFlag (*C function*), 667
drivers_MCX_MCXW_MCXW727C.SFA_CLOCKS (*C macro*), 194
drivers_MCX_MCXW_MCXW727C.sfa_config_t (*C type*), 675
drivers_MCX_MCXW_MCXW727C.SFA_CUT_CLK_Enable (*C macro*), 675
drivers_MCX_MCXW_MCXW727C.SFA_CUT_COUNTER_STOP_TIMEOUT (*C macro*), 670
drivers_MCX_MCXW_MCXW727C.sfa_cut_select_t (*C type*), 675
drivers_MCX_MCXW_MCXW727C.SFA_Deinit (*C function*), 666
drivers_MCX_MCXW_MCXW727C.SFA_DisableInterrupts (*C function*), 667
drivers_MCX_MCXW_MCXW727C.SFA_EnableCUTPin (*C function*), 667
drivers_MCX_MCXW_MCXW727C.SFA_EnableInterrupts (*C function*), 667
drivers_MCX_MCXW_MCXW727C.SFA_GetCUTCounter (*C function*), 669
drivers_MCX_MCXW_MCXW727C.SFA_GetCUTHighLimitClockCount (*C function*), 669
drivers_MCX_MCXW_MCXW727C.SFA_GetCUTLowLimitClockCount (*C function*), 669
drivers_MCX_MCXW_MCXW727C.SFA_GetCUTPredivide (*C function*), 667
drivers_MCX_MCXW_MCXW727C.SFA_GetCUTTargetCount (*C function*), 669
drivers_MCX_MCXW_MCXW727C.SFA_GetDefaultConfig (*C function*), 666
drivers_MCX_MCXW_MCXW727C.SFA_GetMode (*C function*), 667
drivers_MCX_MCXW_MCXW727C.SFA_GetREFCounter (*C function*), 669
drivers_MCX_MCXW_MCXW727C.SFA_GetREFEndCount (*C function*), 670
drivers_MCX_MCXW_MCXW727C.SFA_GetREFHighLimitClockCount (*C function*), 670
drivers_MCX_MCXW_MCXW727C.SFA_GetREFLowLimitClockCount (*C function*), 670
drivers_MCX_MCXW_MCXW727C.SFA_GetREFStartCount (*C function*), 670
drivers_MCX_MCXW_MCXW727C.SFA_GetREFTargetCount (*C function*), 670
drivers_MCX_MCXW_MCXW727C.SFA_GetStatusFlags (*C function*), 667
drivers_MCX_MCXW_MCXW727C.SFA_Init (*C function*), 666
drivers_MCX_MCXW_MCXW727C.SFA_InstallCallback (*C function*), 668
drivers_MCX_MCXW_MCXW727C.SFA_MeasureBlocking (*C function*), 668
drivers_MCX_MCXW_MCXW727C.sfa_measurement_mode_t (*C type*), 675
drivers_MCX_MCXW_MCXW727C.SFA_MEASUREMENT_START_TIMEOUT (*C macro*), 670
drivers_MCX_MCXW_MCXW727C.SFA_MeasureNonBlocking (*C function*), 668
drivers_MCX_MCXW_MCXW727C.SFA_REF_COUNTER_STOP_TIMEOUT (*C macro*), 671
drivers_MCX_MCXW_MCXW727C.sfa_ref_select_t (*C type*), 675
drivers_MCX_MCXW_MCXW727C.SFA_SetCUTHighLimitClockCount (*C function*), 669
drivers_MCX_MCXW_MCXW727C.SFA_SetCUTLowLimitClockCount (*C function*), 669
drivers_MCX_MCXW_MCXW727C.SFA_SetCUTTargetCount (*C function*), 669
drivers_MCX_MCXW_MCXW727C.SFA_SetMeasureConfig (*C function*), 668
drivers_MCX_MCXW_MCXW727C.SFA_SetREFHighLimitClockCount (*C function*), 670
drivers_MCX_MCXW_MCXW727C.SFA_SetREFLowLimitClockCount (*C function*), 670
drivers_MCX_MCXW_MCXW727C.SFA_SetREFTargetCount (*C function*), 669
drivers_MCX_MCXW_MCXW727C.sfa_trigger_end_polarity_t (*C type*), 675
drivers_MCX_MCXW_MCXW727C.sfa_trigger_end_select_t (*C type*), 675
drivers_MCX_MCXW_MCXW727C.sfa_trigger_start_polarity_t (*C type*), 675

drivers_MCX_MCXW_MCXW727C.sfa_trigger_start_select_t (*C type*), 675
drivers_MCX_MCXW_MCXW727C.SMSCM_ClearEccError (*C function*), 680
drivers_MCX_MCXW_MCXW727C.SMSCM_ClearFlashCache (*C function*), 679
drivers_MCX_MCXW_MCXW727C.SMSCM_DebugLock (*C function*), 678
drivers_MCX_MCXW_MCXW727C.SMSCM_DecreaseSecurityCount (*C function*), 678
drivers_MCX_MCXW_MCXW727C.SMSCM_DecreaseSecurityCountBy1 (*C function*), 678
drivers_MCX_MCXW_MCXW727C.SMSCM_DisableDebug (*C function*), 677
drivers_MCX_MCXW_MCXW727C.smscm_ecc_ctrl_t (*C type*), 677
drivers_MCX_MCXW_MCXW727C.smscm_ecc_fault_attr (*C struct*), 681
drivers_MCX_MCXW_MCXW727C.smscm_ecc_fault_attr_t (*C type*), 677
drivers_MCX_MCXW_MCXW727C.SMSCM_EnableDataPrefetch (*C function*), 679
drivers_MCX_MCXW_MCXW727C.SMSCM_EnableDebug (*C function*), 677
drivers_MCX_MCXW_MCXW727C.SMSCM_EnableEccReport (*C function*), 680
drivers_MCX_MCXW_MCXW727C.SMSCM_EnableFlashCache (*C function*), 679
drivers_MCX_MCXW_MCXW727C.SMSCM_EnableFlashDataCache (*C function*), 679
drivers_MCX_MCXW_MCXW727C.SMSCM_EnableFlashDataNonCorrectableBusError (*C function*), 679
drivers_MCX_MCXW_MCXW727C.SMSCM_EnableFlashInstructionCache (*C function*), 679
drivers_MCX_MCXW_MCXW727C.SMSCM_EnableFlashInstructionNonCorrectableBusError (*C function*), 680
drivers_MCX_MCXW_MCXW727C.SMSCM_EnableFlashSpeculate (*C function*), 679
drivers_MCX_MCXW_MCXW727C.SMSCM_GetEccAddress (*C function*), 680
drivers_MCX_MCXW_MCXW727C.SMSCM_GetEccAttribute (*C function*), 681
drivers_MCX_MCXW_MCXW727C.SMSCM_GetEccFaultDataHigh (*C function*), 681
drivers_MCX_MCXW_MCXW727C.SMSCM_GetEccFaultDataLow (*C function*), 681
drivers_MCX_MCXW_MCXW727C.SMSCM_GetEccLocation (*C function*), 680
drivers_MCX_MCXW_MCXW727C.SMSCM_GetEccValid (*C function*), 680
drivers_MCX_MCXW_MCXW727C.SMSCM_GetSecurityCount (*C function*), 678
drivers_MCX_MCXW_MCXW727C.SMSCM_IncreaseSecurityCount (*C function*), 678
drivers_MCX_MCXW_MCXW727C.SMSCM_IncreaseSecurityCountBy1 (*C function*), 678
drivers_MCX_MCXW_MCXW727C.SMSCM_LockFlashIFR1 (*C function*), 679
drivers_MCX_MCXW_MCXW727C.SMSCM_LockMemControlReg (*C function*), 679
drivers_MCX_MCXW_MCXW727C.smscm_mem_t (*C type*), 677
drivers_MCX_MCXW_MCXW727C.SMSCM_SetMemEccControl (*C function*), 680
drivers_MCX_MCXW_MCXW727C.SMSCM_SetSecurityCount (*C function*), 678
drivers_MCX_MCXW_MCXW727C.spc_active_mode_core_ldo_option_t (*C type*), 688
drivers_MCX_MCXW_MCXW727C.spc_active_mode_dcdc_option_t (*C type*), 688
drivers_MCX_MCXW_MCXW727C.spc_active_mode_regulators_config_t (*C type*), 689
drivers_MCX_MCXW_MCXW727C.spc_active_mode_sys_ldo_option_t (*C type*), 688
drivers_MCX_MCXW_MCXW727C.spc_bandgap_mode_t (*C type*), 688
drivers_MCX_MCXW_MCXW727C.SPC_BUSY_TIMEOUT (*C macro*), 690
drivers_MCX_MCXW_MCXW727C.SPC_CheckGlitchRippleCounterOutput (*C function*), 717
drivers_MCX_MCXW_MCXW727C.SPC_CheckHPCfgSelected (*C function*), 690
drivers_MCX_MCXW_MCXW727C.SPC_CheckLowPowerReqest (*C function*), 697
drivers_MCX_MCXW_MCXW727C.SPC_CheckPowerDomainLowPowerRequest (*C function*), 698
drivers_MCX_MCXW_MCXW727C.SPC_CheckPowerSwitchState (*C function*), 697
drivers_MCX_MCXW_MCXW727C.SPC_CheckVddCoreVoltageGlitchResetControlState (*C function*), 717
drivers_MCX_MCXW_MCXW727C.SPC_ClearGlitchRippleCounterOutput (*C function*), 717
drivers_MCX_MCXW_MCXW727C.SPC_ClearLowPowerRequest (*C function*), 697
drivers_MCX_MCXW_MCXW727C.SPC_ClearPeriphIOIsolationFlag (*C function*), 696
drivers_MCX_MCXW_MCXW727C.SPC_ClearPowerDomainLowPowerRequestFlag (*C function*), 698
drivers_MCX_MCXW_MCXW727C.SPC_ClearVoltageDetectStatusFlag (*C function*), 704
drivers_MCX_MCXW_MCXW727C.SPC_ConfigVddCoreGlitchDetector (*C function*), 716
drivers_MCX_MCXW_MCXW727C.spc_core_ldo_drive_strength_t (*C type*), 688
drivers_MCX_MCXW_MCXW727C.spc_core_ldo_voltage_level_t (*C type*), 688
drivers_MCX_MCXW_MCXW727C.spc_core_voltage_detect_config_t (*C type*), 689

drivers_MCX_MCXW_MCXW727C.SPC_DCDC_ACK_TIMEOUT (*C macro*), 690
drivers_MCX_MCXW_MCXW727C.spc_dcdc_burst_config_t (*C type*), 689
drivers_MCX_MCXW_MCXW727C.spc_dcdc_drive_strength_t (*C type*), 688
drivers_MCX_MCXW_MCXW727C.spc_dcdc_ramp_config_t (*C type*), 690
drivers_MCX_MCXW_MCXW727C.spc_dcdc_voltage_level_t (*C type*), 688
drivers_MCX_MCXW_MCXW727C.SPC_DisableActiveModeVddCoreGlitchDetect (*C function*), 701
drivers_MCX_MCXW_MCXW727C.SPC_DisableHighPowerModeVddCoreGlitchDetect (*C function*), 692
drivers_MCX_MCXW_MCXW727C.SPC_DisableLowPowerModeVddCoreGlitchDetect (*C function*), 704
drivers_MCX_MCXW_MCXW727C.SPC_EnableActiveModeCMPBandgapBuffer (*C function*), 700
drivers_MCX_MCXW_MCXW727C.SPC_EnableActiveModeCoreHighVoltageDetect (*C function*), 705
drivers_MCX_MCXW_MCXW727C.SPC_EnableActiveModeCoreLowVoltageDetect (*C function*), 705
drivers_MCX_MCXW_MCXW727C.SPC_EnableActiveModeDCDC2P5VOutput (*C function*), 693
drivers_MCX_MCXW_MCXW727C.SPC_EnableActiveModeIOHighVoltageDetect (*C function*), 710
drivers_MCX_MCXW_MCXW727C.SPC_EnableActiveModeIOLowVoltageDetect (*C function*), 710
drivers_MCX_MCXW_MCXW727C.SPC_EnableActiveModeSystemHighVoltageDetect (*C function*), 708
drivers_MCX_MCXW_MCXW727C.SPC_EnableActiveModeSystemLowVoltageDetect (*C function*), 709
drivers_MCX_MCXW_MCXW727C.SPC_EnableActiveModeVDD1P8LowVoltageDetect (*C function*), 707
drivers_MCX_MCXW_MCXW727C.SPC_EnableCoreLDORegulator (*C function*), 712
drivers_MCX_MCXW_MCXW727C.SPC_EnableDCDCRegulator (*C function*), 715
drivers_MCX_MCXW_MCXW727C.SPC_EnableHighPowerModeCMPBandgapBuffer (*C function*), 692
drivers_MCX_MCXW_MCXW727C.SPC_EnableHighPowerModeCoreHighVoltageDetect (*C function*), 695
drivers_MCX_MCXW_MCXW727C.SPC_EnableHighPowerModeCoreLowVoltageDetect (*C function*), 695
drivers_MCX_MCXW_MCXW727C.SPC_EnableHighPowerModeDCDC2P5VOutput (*C function*), 693
drivers_MCX_MCXW_MCXW727C.SPC_EnableHighPowerModeIOHighVoltageDetect (*C function*), 696
drivers_MCX_MCXW_MCXW727C.SPC_EnableHighPowerModeIOLowVoltageDetect (*C function*), 696
drivers_MCX_MCXW_MCXW727C.SPC_EnableHighPowerModeSystemHighVoltageDetect (*C function*), 695
drivers_MCX_MCXW_MCXW727C.SPC_EnableHighPowerModeSystemLowVoltageDetect (*C function*), 695
drivers_MCX_MCXW_MCXW727C.SPC_EnableHighPowerModeVDD1P8LowVoltageDetect (*C function*), 696
drivers_MCX_MCXW_MCXW727C.SPC_EnableHighPowerRequest (*C function*), 690
drivers_MCX_MCXW_MCXW727C.SPC_EnableLowPowerModeCMPBandgapBufferMode (*C function*), 703
drivers_MCX_MCXW_MCXW727C.SPC_EnableLowPowerModeCoreHighVoltageDetect (*C function*), 706
drivers_MCX_MCXW_MCXW727C.SPC_EnableLowPowerModeCoreLowVoltageDetect (*C function*), 706
drivers_MCX_MCXW_MCXW727C.SPC_EnableLowPowerModeCoreVDDInternalVoltageScaling (*C function*), 703
drivers_MCX_MCXW_MCXW727C.SPC_EnableLowPowerModeIOHighVoltageDetect (*C function*), 711
drivers_MCX_MCXW_MCXW727C.SPC_EnableLowPowerModeIOLowVoltageDetect (*C function*), 711
drivers_MCX_MCXW_MCXW727C.SPC_EnableLowPowerModeLowPowerIREF (*C function*), 702

drivers_MCX_MCXW_MCXW727C.SPC_EnableLowPowerModeSystemHighVoltageDetect (*C function*), 709
drivers_MCX_MCXW_MCXW727C.SPC_EnableLowPowerModeSystemLowVoltageDetect (*C function*), 709
drivers_MCX_MCXW_MCXW727C.SPC_EnableLowPowerModeVDD1P8LowVoltageDetect (*C function*), 707
drivers_MCX_MCXW_MCXW727C.SPC_EnableLowPowerModeVDDCWellBias (*C function*), 702
drivers_MCX_MCXW_MCXW727C.SPC_EnableSystemLDORegulator (*C function*), 713
drivers_MCX_MCXW_MCXW727C.SPC_EnableSystemLDOSinkFeature (*C function*), 714
drivers_MCX_MCXW_MCXW727C.SPC_EVD_CFG_REG_EVDISO (*C macro*), 718
drivers_MCX_MCXW_MCXW727C.SPC_EVD_CFG_REG_EVDISO_SHIFT (*C macro*), 717
drivers_MCX_MCXW_MCXW727C.SPC_EVD_CFG_REG_EVDLPISO (*C macro*), 718
drivers_MCX_MCXW_MCXW727C.SPC_EVD_CFG_REG_EVDLPISO_SHIFT (*C macro*), 718
drivers_MCX_MCXW_MCXW727C.SPC_EVD_CFG_REG_EVDSTAT (*C macro*), 718
drivers_MCX_MCXW_MCXW727C.SPC_EVD_CFG_REG_EVDSTAT_SHIFT (*C macro*), 718
drivers_MCX_MCXW_MCXW727C.SPC_GetActiveModeBandgapMode (*C function*), 699
drivers_MCX_MCXW_MCXW727C.SPC_GetActiveModeCoreLDOVDDVoltageLevel (*C function*), 699
drivers_MCX_MCXW_MCXW727C.SPC_GetActiveModeVoltageDetectStatus (*C function*), 700
drivers_MCX_MCXW_MCXW727C.SPC_GetBusyStatusFlag (*C function*), 697
drivers_MCX_MCXW_MCXW727C.SPC_GetExternalDomainsStatus (*C function*), 712
drivers_MCX_MCXW_MCXW727C.SPC_GetHighPowerModeBandgapMode (*C function*), 691
drivers_MCX_MCXW_MCXW727C.SPC_GetHighPowerModeCoreLDOVDDVoltageLevel (*C function*), 691
drivers_MCX_MCXW_MCXW727C.SPC_GetHighPowerModeVoltageDetectStatus (*C function*), 691
drivers_MCX_MCXW_MCXW727C.SPC_GetLowPowerCoreLDOVDDDriveStrength (*C function*), 702
drivers_MCX_MCXW_MCXW727C.SPC_GetLowPowerCoreLDOVDDVoltageLevel (*C function*), 702
drivers_MCX_MCXW_MCXW727C.SPC_GetLowPowerModeBandgapMode (*C function*), 702
drivers_MCX_MCXW_MCXW727C.SPC_GetLowPowerModeVoltageDetectStatus (*C function*), 702
drivers_MCX_MCXW_MCXW727C.SPC_GetPeriphIOIsolationStatus (*C function*), 696
drivers_MCX_MCXW_MCXW727C.SPC_GetPowerDomainLowPowerMode (*C function*), 698
drivers_MCX_MCXW_MCXW727C.SPC_GetVoltageDetectStatusFlag (*C function*), 704
drivers_MCX_MCXW_MCXW727C.SPC_GetWakeUpValue (*C function*), 699
drivers_MCX_MCXW_MCXW727C.SPC_HP_CNFG_CTRL_OVERRIDE_OPT (*C macro*), 690
drivers_MCX_MCXW_MCXW727C.SPC_HP_CNFG_CTRL_OVERRIDE_OPT_MASK (*C macro*), 690
drivers_MCX_MCXW_MCXW727C.SPC_HP_CNFG_CTRL_OVERRIDE_OPT_SHIFT (*C macro*), 690
drivers_MCX_MCXW_MCXW727C.spc_hp_mode_core_ldo_option_t (*C type*), 689
drivers_MCX_MCXW_MCXW727C.spc_hp_mode_dcdc_option_t (*C type*), 689
drivers_MCX_MCXW_MCXW727C.spc_hp_mode_regulators_config_t (*C type*), 689
drivers_MCX_MCXW_MCXW727C.spc_hp_mode_sys_ldo_option_t (*C type*), 689
drivers_MCX_MCXW_MCXW727C.spc_hp_override_request_option_t (*C type*), 689
drivers_MCX_MCXW_MCXW727C.spc_intergrated_power_switch_config_t (*C type*), 688
drivers_MCX_MCXW_MCXW727C.spc_io_voltage_detect_config_t (*C type*), 689
drivers_MCX_MCXW_MCXW727C.SPC_LockCoreVoltageDetectResetSetting (*C function*), 705
drivers_MCX_MCXW_MCXW727C.SPC_LockIOVoltageDetectResetSetting (*C function*), 710
drivers_MCX_MCXW_MCXW727C.SPC_LockSystemVoltageDetectResetSetting (*C function*), 708
drivers_MCX_MCXW_MCXW727C.SPC_LockVddCoreVoltageGlitchDetectResetControl (*C function*), 717
drivers_MCX_MCXW_MCXW727C.spc_low_voltage_level_select_t (*C type*), 688
drivers_MCX_MCXW_MCXW727C.spc_lowpower_mode_core_ldo_option_t (*C type*), 688
drivers_MCX_MCXW_MCXW727C.spc_lowpower_mode_dcdc_option_t (*C type*), 689
drivers_MCX_MCXW_MCXW727C.spc_lowpower_mode_regulators_config_t (*C type*), 689
drivers_MCX_MCXW_MCXW727C.spc_lowpower_mode_sys_ldo_option_t (*C type*), 689
drivers_MCX_MCXW_MCXW727C.spc_lowpower_request_config_t (*C type*), 688

drivers_MCX_MCXW_MCXW727C.spc_lowpower_request_output_override_t (*C type*), 687
drivers_MCX_MCXW_MCXW727C.spc_lowpower_request_pin_polarity_t (*C type*), 687
drivers_MCX_MCXW_MCXW727C.SPC_OverrideHighPowerRequest (*C function*), 691
drivers_MCX_MCXW_MCXW727C.spc_power_domain_id_t (*C type*), 687
drivers_MCX_MCXW_MCXW727C.spc_power_domain_low_power_mode_t (*C type*), 687
drivers_MCX_MCXW_MCXW727C.SPC_PowerModeControlPowerSwitch (*C function*), 699
drivers_MCX_MCXW_MCXW727C.SPC_PullDownCoreLDORegulator (*C function*), 712
drivers_MCX_MCXW_MCXW727C.SPC_SetActiveModeBandgapModeConfig (*C function*), 700
drivers_MCX_MCXW_MCXW727C.SPC_SetActiveModeCoreLDORegulatorConfig (*C function*), 712
drivers_MCX_MCXW_MCXW727C.SPC_SetActiveModeDCDCRegulatorConfig (*C function*), 715
drivers_MCX_MCXW_MCXW727C.SPC_SetActiveModeIntegratedPowerSwitchConfig (*C function*), 700
drivers_MCX_MCXW_MCXW727C.SPC_SetActiveModeRegulatorsConfig (*C function*), 701
drivers_MCX_MCXW_MCXW727C.SPC_SetActiveModeSystemLDORegulatorConfig (*C function*), 714
drivers_MCX_MCXW_MCXW727C.SPC_SetActiveModeVoltageTrimDelay (*C function*), 700
drivers_MCX_MCXW_MCXW727C.SPC_SetCoreVoltageDetectConfig (*C function*), 705
drivers_MCX_MCXW_MCXW727C.SPC_SetDCDCBurstConfig (*C function*), 715
drivers_MCX_MCXW_MCXW727C.SPC_SetDCDCRampControlConfig (*C function*), 694
drivers_MCX_MCXW_MCXW727C.SPC_SetDCDCRefreshCount (*C function*), 715
drivers_MCX_MCXW_MCXW727C.SPC_SetExternalVoltageDomainsConfig (*C function*), 711
drivers_MCX_MCXW_MCXW727C.SPC_SetHighPowerModeBandgapModeConfig (*C function*), 691
drivers_MCX_MCXW_MCXW727C.SPC_SetHighPowerModeCoreLDORegulatorConfig (*C function*), 692
drivers_MCX_MCXW_MCXW727C.SPC_SetHighPowerModeDCDCRegulatorConfig (*C function*), 693
drivers_MCX_MCXW_MCXW727C.SPC_SetHighPowerModeRegulatorsConfig (*C function*), 694
drivers_MCX_MCXW_MCXW727C.SPC_SetHighPowerModeSystemLDORegulatorConfig (*C function*), 692
drivers_MCX_MCXW_MCXW727C.SPC_SetIOVDDLowVoltageLevel (*C function*), 709
drivers_MCX_MCXW_MCXW727C.SPC_SetIOVoltageDetectConfig (*C function*), 710
drivers_MCX_MCXW_MCXW727C.SPC_SetLowPowerModeBandgapmodeConfig (*C function*), 702
drivers_MCX_MCXW_MCXW727C.SPC_SetLowPowerModeCoreLDORegulatorConfig (*C function*), 713
drivers_MCX_MCXW_MCXW727C.SPC_SetLowPowerModeDCDCRegulatorConfig (*C function*), 716
drivers_MCX_MCXW_MCXW727C.SPC_SetLowPowerModeIntegratedPowerSwitchConfig (*C function*), 701
drivers_MCX_MCXW_MCXW727C.SPC_SetLowPowerModeRegulatorsConfig (*C function*), 703
drivers_MCX_MCXW_MCXW727C.SPC_SetLowPowerModeSystemLDORegulatorConfig (*C function*), 714
drivers_MCX_MCXW_MCXW727C.SPC_SetLowPowerRequestConfig (*C function*), 698
drivers_MCX_MCXW_MCXW727C.SPC_SetLowPowerWakeUpDelay (*C function*), 703
drivers_MCX_MCXW_MCXW727C.SPC_SetSRAMOperateVoltage (*C function*), 716
drivers_MCX_MCXW_MCXW727C.SPC_SetSystemVDDLowVoltageLevel (*C function*), 707
drivers_MCX_MCXW_MCXW727C.SPC_SetSystemVoltageDetectConfig (*C function*), 708
drivers_MCX_MCXW_MCXW727C.SPC_SetVDD1P8VoltageDetectConfig (*C function*), 707
drivers_MCX_MCXW_MCXW727C.SPC_SetWakeUpValue (*C function*), 699
drivers_MCX_MCXW_MCXW727C.SPC_SoftwareGatePowerSwitch (*C function*), 698
drivers_MCX_MCXW_MCXW727C.SPC_SRAM_ACK_TIMEOUT (*C macro*), 690
drivers_MCX_MCXW_MCXW727C.spc_sram_operat_voltage_t (*C type*), 688
drivers_MCX_MCXW_MCXW727C.spc_sys_ldo_drive_strength_t (*C type*), 688
drivers_MCX_MCXW_MCXW727C.spc_sys_ldo_voltage_level_t (*C type*), 688
drivers_MCX_MCXW_MCXW727C.spc_system_voltage_detect_config_t (*C type*), 689
drivers_MCX_MCXW_MCXW727C.SPC_UnlockCoreVoltageDetectResetSetting (*C function*), 705
drivers_MCX_MCXW_MCXW727C.SPC_UnlockIOVoltageDetectResetSetting (*C function*), 710
drivers_MCX_MCXW_MCXW727C.SPC_UnlockSystemVoltageDetectResetSetting (*C function*), 708

drivers_MCX_MCXW_MCXW727C.SPC_UnlockVddCoreVoltageGlitchDetectResetControl (*C function*), 717
drivers_MCX_MCXW_MCXW727C.SPC_VD_STAT_VALID_FLAGS_BASE_MASK (*C macro*), 690
drivers_MCX_MCXW_MCXW727C.SPC_VD_STAT_VALID_FLAGS_MASK (*C macro*), 690
drivers_MCX_MCXW_MCXW727C.SPC_VD_STAT_VALID_FLAGS_SYS_MASK (*C macro*), 690
drivers_MCX_MCXW_MCXW727C.spc_vdd1p8_voltage_detect_config_t (*C type*), 689
drivers_MCX_MCXW_MCXW727C.spc_vdd_core_glitch_detector_config_t (*C type*), 689
drivers_MCX_MCXW_MCXW727C.spc_vdd_core_glitch_ripple_counter_select_t (*C type*), 689
drivers_MCX_MCXW_MCXW727C.spc_voltage_detect_option_t (*C type*), 689
drivers_MCX_MCXW_MCXW727C.SPI_RETRY_TIMES (*C macro*), 366, 555
drivers_MCX_MCXW_MCXW727C.status_t (*C type*), 461
drivers_MCX_MCXW_MCXW727C.SUPPRESS_FALL_THROUGH_WARNING (*C macro*), 464
drivers_MCX_MCXW_MCXW727C.SYSPM_CLOCKS (*C macro*), 194
drivers_MCX_MCXW_MCXW727C.SYSPM_COUNT_STABLE_TIMEOUT_RETURN_VALUE (*C macro*), 726
drivers_MCX_MCXW_MCXW727C.SYSPM_Deinit (*C function*), 724
drivers_MCX_MCXW_MCXW727C.SYSPM_DisableCounter (*C function*), 725
drivers_MCX_MCXW_MCXW727C.syspm_event_t (*C type*), 723
drivers_MCX_MCXW_MCXW727C.SYSPM_GetEventCounter (*C function*), 725
drivers_MCX_MCXW_MCXW727C.SYSPM_GetInstructionCounter (*C function*), 725
drivers_MCX_MCXW_MCXW727C.SYSPM_Init (*C function*), 724
drivers_MCX_MCXW_MCXW727C.syspm_mode_t (*C type*), 724
drivers_MCX_MCXW_MCXW727C.syspm_monitor_t (*C type*), 723
drivers_MCX_MCXW_MCXW727C.SYSPM_ResetEvent (*C function*), 724
drivers_MCX_MCXW_MCXW727C.SYSPM_ResetInstructionEvent (*C function*), 724
drivers_MCX_MCXW_MCXW727C.SYSPM_SelectEvent (*C function*), 724
drivers_MCX_MCXW_MCXW727C.SYSPM_SetCountMode (*C function*), 724
drivers_MCX_MCXW_MCXW727C.SYSPM_SetStartStopControl (*C function*), 725
drivers_MCX_MCXW_MCXW727C.syspm_startstop_control_t (*C type*), 724
drivers_MCX_MCXW_MCXW727C.tpm2_src_t (*C type*), 191
drivers_MCX_MCXW_MCXW727C.TPM_CalculateCounterClkDiv (*C function*), 727
drivers_MCX_MCXW_MCXW727C.tpm_callback_t (*C type*), 741
drivers_MCX_MCXW_MCXW727C.tpm_chnl_control_bit_mask_t (*C type*), 740
drivers_MCX_MCXW_MCXW727C.tpm_chnl_pwm_signal_param_t (*C type*), 740
drivers_MCX_MCXW_MCXW727C.tpm_chnl_t (*C type*), 740
drivers_MCX_MCXW_MCXW727C.TPM_ClearStatusFlags (*C function*), 732
drivers_MCX_MCXW_MCXW727C.tpm_clock_prescale_t (*C type*), 741
drivers_MCX_MCXW_MCXW727C.tpm_clock_source_t (*C type*), 741
drivers_MCX_MCXW_MCXW727C.TPM_CLOCKS (*C macro*), 195
drivers_MCX_MCXW_MCXW727C.tpm_config_t (*C type*), 741
drivers_MCX_MCXW_MCXW727C.TPM_Deinit (*C function*), 726
drivers_MCX_MCXW_MCXW727C.TPM_DisableChannel (*C function*), 729
drivers_MCX_MCXW_MCXW727C.TPM_DisableInterrupts (*C function*), 731
drivers_MCX_MCXW_MCXW727C.TPM_DriverIRQHandler (*C function*), 731
drivers_MCX_MCXW_MCXW727C.tpm_dual_edge_capture_param_t (*C type*), 741
drivers_MCX_MCXW_MCXW727C.TPM_EnableChannel (*C function*), 729
drivers_MCX_MCXW_MCXW727C.TPM_EnableChannelExtTrigger (*C function*), 730
drivers_MCX_MCXW_MCXW727C.TPM_EnableInterrupts (*C function*), 731
drivers_MCX_MCXW_MCXW727C.tpm_ext_trigger_polarity_t (*C type*), 740
drivers_MCX_MCXW_MCXW727C.TPM_GetChannelContorlBits (*C function*), 728
drivers_MCX_MCXW_MCXW727C.TPM_GetChannelControlBits (*C function*), 729
drivers_MCX_MCXW_MCXW727C.TPM_GetChannelValue (*C function*), 732
drivers_MCX_MCXW_MCXW727C.TPM_GetCurrentTimerCount (*C function*), 733
drivers_MCX_MCXW_MCXW727C.TPM_GetDefaultConfig (*C function*), 726
drivers_MCX_MCXW_MCXW727C.TPM_GetEnabledInterrupts (*C function*), 731
drivers_MCX_MCXW_MCXW727C.TPM_GetInstance (*C function*), 726

drivers_MCX_MCXW_MCXW727C.TPM_GetStatusFlags (*C function*), 732
drivers_MCX_MCXW_MCXW727C.TPM_Init (*C function*), 726
drivers_MCX_MCXW_MCXW727C.tpm_input_capture_edge_t (*C type*), 741
drivers_MCX_MCXW_MCXW727C.tpm_interrupt_enable_t (*C type*), 741
drivers_MCX_MCXW_MCXW727C.TPM_MAX_COUNTER_VALUE (*C macro*), 742
drivers_MCX_MCXW_MCXW727C.tpm_output_compare_mode_t (*C type*), 741
drivers_MCX_MCXW_MCXW727C.tpm_phase_params_t (*C type*), 741
drivers_MCX_MCXW_MCXW727C.tpm_phase_polarity_t (*C type*), 741
drivers_MCX_MCXW_MCXW727C.tpm_pwm_level_select_t (*C type*), 740
drivers_MCX_MCXW_MCXW727C.tpm_pwm_mode_t (*C type*), 740
drivers_MCX_MCXW_MCXW727C.tpm_pwm_pause_level_select_t (*C type*), 740
drivers_MCX_MCXW_MCXW727C.tpm_quad_decode_mode_t (*C type*), 741
drivers_MCX_MCXW_MCXW727C.TPM_RegisterCallBack (*C function*), 731
drivers_MCX_MCXW_MCXW727C.TPM_Reset (*C function*), 727
drivers_MCX_MCXW_MCXW727C.TPM_SetChannelPolarity (*C function*), 730
drivers_MCX_MCXW_MCXW727C.TPM_SetTimerPeriod (*C function*), 732
drivers_MCX_MCXW_MCXW727C.TPM_SetupDualEdgeCapture (*C function*), 730
drivers_MCX_MCXW_MCXW727C.TPM_SetupInputCapture (*C function*), 729
drivers_MCX_MCXW_MCXW727C.TPM_SetupOutputCompare (*C function*), 729
drivers_MCX_MCXW_MCXW727C.TPM_SetupPwm (*C function*), 727
drivers_MCX_MCXW_MCXW727C.TPM_SetupQuadDecode (*C function*), 730
drivers_MCX_MCXW_MCXW727C.TPM_StartTimer (*C function*), 733
drivers_MCX_MCXW_MCXW727C.tpm_status_flags_t (*C type*), 741
drivers_MCX_MCXW_MCXW727C.TPM_StopTimer (*C function*), 733
drivers_MCX_MCXW_MCXW727C.TPM_TIMEOUT (*C macro*), 742
drivers_MCX_MCXW_MCXW727C.tpm_trigger_select_t (*C type*), 740
drivers_MCX_MCXW_MCXW727C.tpm_trigger_source_t (*C type*), 740
drivers_MCX_MCXW_MCXW727C.TPM_UpdateChnlEdgeLevelSelect (*C function*), 728
drivers_MCX_MCXW_MCXW727C.TPM_UpdatePwmDutycycle (*C function*), 728
drivers_MCX_MCXW_MCXW727C.trdc_controller_t (*C type*), 761
drivers_MCX_MCXW_MCXW727C.TRDC_Deinit (*C function*), 745
drivers_MCX_MCXW_MCXW727C.trdc_did_sel_t (*C type*), 761
drivers_MCX_MCXW_MCXW727C.trdc_domain_error_t (*C type*), 762
drivers_MCX_MCXW_MCXW727C.TRDC_EnableFlashLogicalWindow (*C function*), 748
drivers_MCX_MCXW_MCXW727C.TRDC_EnableProcessorDomainAssignment (*C function*), 746
drivers_MCX_MCXW_MCXW727C.trdc_error_attr_t (*C type*), 761
drivers_MCX_MCXW_MCXW727C.trdc_error_state_t (*C type*), 761
drivers_MCX_MCXW_MCXW727C.trdc_error_type_t (*C type*), 761
drivers_MCX_MCXW_MCXW727C.trdc_flw_config_t (*C type*), 761
drivers_MCX_MCXW_MCXW727C.TRDC_GetAndClearFirstDomainError (*C function*), 749
drivers_MCX_MCXW_MCXW727C.TRDC_GetAndClearFirstSpecificDomainError (*C function*), 749
drivers_MCX_MCXW_MCXW727C.TRDC_GetCurrentMasterDomainId (*C function*), 745
drivers_MCX_MCXW_MCXW727C.TRDC_GetDefaultFlashLogicalWindowConfig (*C function*), 748
drivers_MCX_MCXW_MCXW727C.TRDC_GetDefaultIDAUConfig (*C function*), 747
drivers_MCX_MCXW_MCXW727C.TRDC_GetDefaultNonProcessorDomainAssignment (*C function*), 746
drivers_MCX_MCXW_MCXW727C.TRDC_GetDefaultProcessorDomainAssignment (*C function*), 745
drivers_MCX_MCXW_MCXW727C.TRDC_GetFlashLogicalWindowPbase (*C function*), 748
drivers_MCX_MCXW_MCXW727C.TRDC_GetHardwareConfig (*C function*), 745
drivers_MCX_MCXW_MCXW727C.TRDC_GetMbcHardwareConfig (*C function*), 751
drivers_MCX_MCXW_MCXW727C.TRDC_GetMrcRegionNumber (*C function*), 749
drivers_MCX_MCXW_MCXW727C.TRDC_GetSetFlashLogicalWindowSize (*C function*), 748
drivers_MCX_MCXW_MCXW727C.trdc_hardware_config_t (*C type*), 761
drivers_MCX_MCXW_MCXW727C.trdc_idau_config_t (*C type*), 761
drivers_MCX_MCXW_MCXW727C.TRDC_Init (*C function*), 744
drivers_MCX_MCXW_MCXW727C.TRDC_LockFlashLogicalWindow (*C function*), 748
drivers_MCX_MCXW_MCXW727C.TRDC_LockMasterDomainAssignment (*C function*), 745

drivers_MCX_MCXW_MCXW727C.trdc_mbc_memory_block_config_t (*C type*), 762
drivers_MCX_MCXW_MCXW727C.trdc_mbc_nse_update_config_t (*C type*), 762
drivers_MCX_MCXW_MCXW727C.TRDC_MbcNseClearAll (*C function*), 752
drivers_MCX_MCXW_MCXW727C.TRDC_MbcSetMemoryAccessConfig (*C function*), 752
drivers_MCX_MCXW_MCXW727C.TRDC_MbcSetMemoryBlockConfig (*C function*), 752
drivers_MCX_MCXW_MCXW727C.TRDC_MbcSetNseUpdateConfig (*C function*), 751
drivers_MCX_MCXW_MCXW727C.TRDC_MbcWordNseClear (*C function*), 751
drivers_MCX_MCXW_MCXW727C.TRDC_MbcWordNseSet (*C function*), 751
drivers_MCX_MCXW_MCXW727C.trdc_memory_access_control_config_t (*C type*), 762
drivers_MCX_MCXW_MCXW727C.trdc_mrc_region_descriptor_config_t (*C type*), 762
drivers_MCX_MCXW_MCXW727C.TRDC_MrcDomainNseClear (*C function*), 750
drivers_MCX_MCXW_MCXW727C.TRDC_MrcEnableDomainNseUpdate (*C function*), 750
drivers_MCX_MCXW_MCXW727C.TRDC_MrcRegionNseClear (*C function*), 750
drivers_MCX_MCXW_MCXW727C.TRDC_MrcRegionNseSet (*C function*), 750
drivers_MCX_MCXW_MCXW727C.TRDC_MrcSetMemoryAccessConfig (*C function*), 749
drivers_MCX_MCXW_MCXW727C.TRDC_MrcSetRegionDescriptorConfig (*C function*), 751
drivers_MCX_MCXW_MCXW727C.trdc_non_processor_domain_assignment_t (*C type*), 761
drivers_MCX_MCXW_MCXW727C.trdc_privilege_attr_t (*C type*), 761
drivers_MCX_MCXW_MCXW727C.trdc_processor_domain_assignment_t (*C type*), 761
drivers_MCX_MCXW_MCXW727C.trdc_secure_attr_t (*C type*), 761
drivers_MCX_MCXW_MCXW727C.TRDC_SetDacGlobalValid (*C function*), 745
drivers_MCX_MCXW_MCXW727C.TRDC_SetFlashLogicalWindow (*C function*), 748
drivers_MCX_MCXW_MCXW727C.TRDC_SetIDAU (*C function*), 747
drivers_MCX_MCXW_MCXW727C.TRDC_SetMasterDomainAssignmentValid (*C function*), 745
drivers_MCX_MCXW_MCXW727C.TRDC_SetMbcGlobalValid (*C function*), 751
drivers_MCX_MCXW_MCXW727C.TRDC_SetMrcGlobalValid (*C function*), 749
drivers_MCX_MCXW_MCXW727C.TRDC_SetNonProcessorDomainAssignment (*C function*), 747
drivers_MCX_MCXW_MCXW727C.TRDC_SetProcessorDomainAssignment (*C function*), 746
drivers_MCX_MCXW_MCXW727C.trdc_slave_memory_hardware_config_t (*C type*), 761
drivers_MCX_MCXW_MCXW727C.TRGMUX_LockRegister (*C function*), 768
drivers_MCX_MCXW_MCXW727C.TRGMUX_SetTriggerSource (*C function*), 769
drivers_MCX_MCXW_MCXW727C.trgmux_trigger_input_t (*C type*), 769
drivers_MCX_MCXW_MCXW727C.TSTMR_CLOCKS (*C macro*), 195
drivers_MCX_MCXW_MCXW727C.UART_RETRY_TIMES (*C macro*), 378, 592
drivers_MCX_MCXW_MCXW727C.UINT16_MAX (*C macro*), 449
drivers_MCX_MCXW_MCXW727C.UINT32_MAX (*C macro*), 449
drivers_MCX_MCXW_MCXW727C.UINT64_H (*C macro*), 449
drivers_MCX_MCXW_MCXW727C.UINT64_L (*C macro*), 449
drivers_MCX_MCXW_MCXW727C.UINTPTR_SIZE (*C macro*), 449
drivers_MCX_MCXW_MCXW727C.USEC_TO_COUNT (*C macro*), 451
drivers_MCX_MCXW_MCXW727C.vbat_bandgap_refresh_period_t (*C type*), 772
drivers_MCX_MCXW_MCXW727C.vbat_bandgap_timer_timeout_period_t (*C type*), 772
drivers_MCX_MCXW_MCXW727C.VBAT_CheckBandgapEnabled (*C function*), 774
drivers_MCX_MCXW_MCXW727C.VBAT_CheckFRO16kEnabled (*C function*), 773
drivers_MCX_MCXW_MCXW727C.VBAT_ClearStatusFlags (*C function*), 776
drivers_MCX_MCXW_MCXW727C.VBAT_ConfigFRO16k (*C function*), 773
drivers_MCX_MCXW_MCXW727C.VBAT_DisableInterrupts (*C function*), 777
drivers_MCX_MCXW_MCXW727C.VBAT_DisableWakeup (*C function*), 777
drivers_MCX_MCXW_MCXW727C.VBAT_EnableBackupSRAMRegulator (*C function*), 774
drivers_MCX_MCXW_MCXW727C.VBAT_EnableBandgap (*C function*), 773
drivers_MCX_MCXW_MCXW727C.VBAT_EnableBandgapRefreshMode (*C function*), 774
drivers_MCX_MCXW_MCXW727C.VBAT_EnableBandgapTimer (*C function*), 776
drivers_MCX_MCXW_MCXW727C.VBAT_EnableFRO16k (*C function*), 773
drivers_MCX_MCXW_MCXW727C.VBAT_EnableInterrupts (*C function*), 777
drivers_MCX_MCXW_MCXW727C.VBAT_EnableSRAMArrayRetained (*C function*), 775
drivers_MCX_MCXW_MCXW727C.VBAT_EnableSRAMIsolation (*C function*), 775
drivers_MCX_MCXW_MCXW727C.VBAT_EnableWakeup (*C function*), 777
drivers_MCX_MCXW_MCXW727C.vbat_fro16k_config_t (*C type*), 772

drivers_MCX_MCXW_MCXW727C.VBAT_GetStatusFlags (*C function*), 776
drivers_MCX_MCXW_MCXW727C.VBAT_LDO_READY_TIMEOUT (*C macro*), 772
drivers_MCX_MCXW_MCXW727C.VBAT_LockFRO16kSettings (*C function*), 773
drivers_MCX_MCXW_MCXW727C.VBAT_LockInterruptWakeupSettings (*C function*), 777
drivers_MCX_MCXW_MCXW727C.VBAT_LockLdoRamSettings (*C function*), 775
drivers_MCX_MCXW_MCXW727C.VBAT_SetBandgapTimerTimeoutValue (*C function*), 776
drivers_MCX_MCXW_MCXW727C.VBAT_SwitchSRAMPowerBySocSupply (*C function*), 775
drivers_MCX_MCXW_MCXW727C.VBAT_SwitchSRAMPowerByVBAT (*C function*), 775
drivers_MCX_MCXW_MCXW727C.VBAT_UngateFRO16k (*C function*), 773
drivers_MCX_MCXW_MCXW727C.vref_buffer_mode_t (*C type*), 780
drivers_MCX_MCXW_MCXW727C.VREF_CLOCKS (*C macro*), 195
drivers_MCX_MCXW_MCXW727C.vref_config_t (*C type*), 780
drivers_MCX_MCXW_MCXW727C.VREF_Deinit (*C function*), 778
drivers_MCX_MCXW_MCXW727C.VREF_GetDefaultConfig (*C function*), 778
drivers_MCX_MCXW_MCXW727C.VREF_GetTrim21Val (*C function*), 779
drivers_MCX_MCXW_MCXW727C.VREF_GetVrefTrimVal (*C function*), 779
drivers_MCX_MCXW_MCXW727C.VREF_Init (*C function*), 778
drivers_MCX_MCXW_MCXW727C.VREF_SetAdcNegChannelRef (*C function*), 779
drivers_MCX_MCXW_MCXW727C.VREF_SetAdcPosChannelRef (*C function*), 780
drivers_MCX_MCXW_MCXW727C.VREF_SetTrim21Val (*C function*), 779
drivers_MCX_MCXW_MCXW727C.VREF_SetVrefTrimVal (*C function*), 779
drivers_MCX_MCXW_MCXW727C.WDOG32_ClearStatusFlags (*C function*), 783
drivers_MCX_MCXW_MCXW727C.wdog32_clock_prescaler_t (*C type*), 786
drivers_MCX_MCXW_MCXW727C.wdog32_clock_source_t (*C type*), 786
drivers_MCX_MCXW_MCXW727C.wdog32_config_t (*C type*), 786
drivers_MCX_MCXW_MCXW727C.WDOG32_Deinit (*C function*), 782
drivers_MCX_MCXW_MCXW727C.WDOG32_Disable (*C function*), 782
drivers_MCX_MCXW_MCXW727C.WDOG32_DisableInterrupts (*C function*), 783
drivers_MCX_MCXW_MCXW727C.WDOG32_Enable (*C function*), 782
drivers_MCX_MCXW_MCXW727C.WDOG32_EnableInterrupts (*C function*), 783
drivers_MCX_MCXW_MCXW727C.WDOG32_GetCounterValue (*C function*), 784
drivers_MCX_MCXW_MCXW727C.WDOG32_GetDefaultConfig (*C function*), 781
drivers_MCX_MCXW_MCXW727C.WDOG32_GetStatusFlags (*C function*), 783
drivers_MCX_MCXW_MCXW727C.WDOG32_Init (*C function*), 781
drivers_MCX_MCXW_MCXW727C.WDOG32_Refresh (*C function*), 784
drivers_MCX_MCXW_MCXW727C.WDOG32_SetTimeoutValue (*C function*), 784
drivers_MCX_MCXW_MCXW727C.WDOG32_SetWindowValue (*C function*), 784
drivers_MCX_MCXW_MCXW727C.wdog32_test_mode_t (*C type*), 786
drivers_MCX_MCXW_MCXW727C.WDOG32_Unlock (*C function*), 782
drivers_MCX_MCXW_MCXW727C.wdog32_work_mode_t (*C type*), 786
drivers_MCX_MCXW_MCXW727C.WDOG_CLOCKS (*C macro*), 195
drivers_MCX_MCXW_MCXW727C.WDOG_FIRST_WORD_OF_REFRESH (*C macro*), 784
drivers_MCX_MCXW_MCXW727C.WDOG_FIRST_WORD_OF_UNLOCK (*C macro*), 784
drivers_MCX_MCXW_MCXW727C.WDOG_SECOND_WORD_OF_REFRESH (*C macro*), 785
drivers_MCX_MCXW_MCXW727C.WDOG_SECOND_WORD_OF_UNLOCK (*C macro*), 784
drivers_MCX_MCXW_MCXW727C.WUU_ClearExternalWakeupPinFlag (*C function*), 790
drivers_MCX_MCXW_MCXW727C.WUU_ClearExternalWakeupPinsConfig (*C function*), 788
drivers_MCX_MCXW_MCXW727C.WUU_ClearExternalWakeUpPinsFlag (*C function*), 788
drivers_MCX_MCXW_MCXW727C.WUU_ClearInternalWakeUpModulesConfig (*C function*), 789
drivers_MCX_MCXW_MCXW727C.WUU_ClearPinFilterFlag (*C function*), 790
drivers_MCX_MCXW_MCXW727C.wuu_external_pin_edge_detection_t (*C type*), 792
drivers_MCX_MCXW_MCXW727C.wuu_external_wakeup_pin_config_t (*C type*), 792
drivers_MCX_MCXW_MCXW727C.wuu_external_wakeup_pin_event_t (*C type*), 792
drivers_MCX_MCXW_MCXW727C.wuu_external_wakeup_pin_mode_t (*C type*), 792
drivers_MCX_MCXW_MCXW727C.wuu_filter_edge_t (*C type*), 792
drivers_MCX_MCXW_MCXW727C.wuu_filter_event_t (*C type*), 792
drivers_MCX_MCXW_MCXW727C.wuu_filter_mode_t (*C type*), 792
drivers_MCX_MCXW_MCXW727C.WUU_GetExternalWakeupPinFlag (*C function*), 790

drivers_MCX_MCXW_MCXW727C.WUU_GetExternalWakeUpPinsFlag (*C function*), 788
drivers_MCX_MCXW_MCXW727C.WUU_GetInternalWakeupModuleFlag (*C function*), 789
drivers_MCX_MCXW_MCXW727C.WUU_GetModuleInterruptFlag (*C function*), 789
drivers_MCX_MCXW_MCXW727C.WUU_GetPinFilterFlag (*C function*), 789
drivers_MCX_MCXW_MCXW727C.wuu_internal_wakeup_module_event_t (*C type*), 792
drivers_MCX_MCXW_MCXW727C.wuu_pin_filter_config_t (*C type*), 793
drivers_MCX_MCXW_MCXW727C.WUU_SetExternalWakeUpPinsConfig (*C function*), 788
drivers_MCX_MCXW_MCXW727C.WUU_SetInternalWakeUpModulesConfig (*C function*), 788
drivers_MCX_MCXW_MCXW727C.WUU_SetPinFilterConfig (*C function*), 789

G

gAdvDataChange_c (*C macro*), 1506
gaGattDynamicAttrBlob (*C var*), 1449
gaGattDynamicValBlob (*C var*), 1449
Gap_AcceptPairingRequest (*C function*), 1310
Gap_AddDeviceToFilterAcceptList (*C function*), 1324
Gap_AddDeviceToMonAdvList (*C function*), 1318
Gap_AddSecurityModesAndLevels (*C macro*), 1360
Gap_Authorize (*C function*), 1305
Gap_BleAdvIndexChange (*C function*), 1337
Gap_CancelInitiatingConnection (*C macro*), 1360
Gap_CheckIfBonded (*C function*), 1323
Gap_CheckIfConnected (*C function*), 1323
Gap_CheckIndicationStatus (*C function*), 1307
Gap_CheckNotificationStatus (*C function*), 1307
Gap_CheckNvmIndex (*C function*), 1324
Gap_ClearFilterAcceptList (*C function*), 1324
Gap_ClearMonAdvList (*C function*), 1318
Gap_Connect (*C function*), 1320
Gap_ConnectFromPawr (*C function*), 1320
Gap_ConnectionSubrateRequest (*C function*), 1333
Gap_ControllerEnhancedNotification (*C function*), 1336
Gap_ControllerTest (*C function*), 1334
Gap_CreateRandomDeviceAddress (*C function*), 1325
Gap_DecryptAdvertisingData (*C function*), 1359
Gap_DenyLongTermKey (*C function*), 1314
Gap_Disconnect (*C function*), 1321
Gap_EattConnectionAccept (*C function*), 1354
Gap_EattConnectionRequest (*C function*), 1354
Gap_EattDisconnect (*C function*), 1355
Gap_EattReconfigureRequest (*C function*), 1355
Gap_EattSendCredits (*C function*), 1355
Gap_EnableConnectionCteRequest (*C function*), 1351
Gap_EnableConnectionCteResponse (*C function*), 1351
Gap_EnableConnectionlessCteTransmit (*C function*), 1350
Gap_EnableConnectionlessIqSampling (*C function*), 1350
Gap_EnableControllerPrivacy (*C function*), 1333
Gap_EnableHostPrivacy (*C function*), 1333
Gap_EnableMonAdv (*C function*), 1318
Gap_EnablePathLossReporting (*C function*), 1353
Gap_EnableTransmitPowerReporting (*C function*), 1353
Gap_EnableUpdateConnectionParameters (*C function*), 1332
Gap_EncryptAdvertisingData (*C function*), 1358
Gap_EncryptLink (*C function*), 1309
Gap_EncryptLinkTak (*C function*), 1310
Gap_EnhancedReadTransmitPowerLevel (*C function*), 1352
Gap_EnterPasskey (*C function*), 1311
Gap_GenerateDhKeyV2 (*C function*), 1353

Gap_GetBondedDeviceName (*C function*), 1327
Gap_GetBondedDevicesCount (*C function*), 1326
Gap_GetBondedDevicesIdentityInformation (*C function*), 1308
Gap_GetConnectionHandleFromDeviceId (*C function*), 1356
Gap_GetConnParams (*C macro*), 1361
Gap_GetConnParamsMonitoring (*C function*), 1356
Gap_GetDeviceIdFromConnHandle (*C function*), 1356
Gap_GetHostVersion (*C function*), 1356
Gap_InitMonitoringAdvertisers (*C function*), 1317
Gap_LeChannelOverride (*C function*), 1357
Gap_LeChasConfig (*C function*), 1302
Gap_LeModulationTest (*C function*), 1303
Gap_LePeriodicAdvUpdateSync (*C function*), 1358
Gap_LeReadPhy (*C function*), 1335
Gap_LeScGetLocalOobData (*C function*), 1313
Gap_LeScRegeneratePublicKey (*C function*), 1312
Gap_LeScSendKeypressNotification (*C function*), 1313
Gap_LeScSetPeerOobData (*C function*), 1313
Gap_LeScValidateNumericValue (*C function*), 1312
Gap_LeSetHostFeature (*C function*), 1358
Gap_LeSetLocalPeripheralLatencyEnable (*C function*), 1358
Gap_LeSetPhy (*C function*), 1336
Gap_LeSetSchedulerPriority (*C function*), 1358
Gap_LoadCustomBondedDeviceInformation (*C function*), 1322
Gap_LoadCustomPeerInformation (*C function*), 1321
Gap_LoadEncryptionInformation (*C function*), 1315
Gap_LoadKeys (*C function*), 1337
Gap_ModifySleepClockAccuracy (*C function*), 1335
Gap_Pair (*C function*), 1308
Gap_PeriodicAdvCreateSync (*C function*), 1347
Gap_PeriodicAdvReceiveDisable (*C function*), 1348
Gap_PeriodicAdvReceiveEnable (*C function*), 1347
Gap_PeriodicAdvSetInfoTransfer (*C function*), 1348
Gap_PeriodicAdvSyncTransfer (*C function*), 1348
Gap_PeriodicAdvTerminateSync (*C function*), 1347
Gap_ProvideLongTermKey (*C function*), 1314
Gap_ProvideLongTermKeyTak (*C function*), 1314
Gap_ProvideOob (*C function*), 1311
Gap_ReadAdvertisingTxPowerLevel (*C macro*), 1360
Gap_ReadAntennaInformation (*C function*), 1351
Gap_ReadChannelMap (*C function*), 1339
Gap_ReadControllerLocalRPA (*C function*), 1330
Gap_ReadFilterAcceptListSize (*C function*), 1324
Gap_ReadMonAdvListSize (*C function*), 1319
Gap_ReadPublicDeviceAddress (*C function*), 1325
Gap_ReadRadioPowerLevel (*C function*), 1329
Gap_ReadRemoteTransmitPowerLevel (*C function*), 1352
Gap_ReadRemoteVersionInformation (*C function*), 1356
Gap_ReadRssi (*C macro*), 1360
Gap_ReadTxPowerLevelInConnection (*C macro*), 1361
Gap_RegisterDeviceSecurityRequirements (*C function*), 1303
Gap_RejectKeyExchangeRequest (*C function*), 1312
Gap_RejectPairing (*C function*), 1310
Gap_RejectPasskeyRequest (*C function*), 1311
Gap_RemoveAdvSet (*C function*), 1343
Gap_RemoveAllBonds (*C function*), 1328
Gap_RemoveBond (*C function*), 1328
Gap_RemoveDeviceFromFilterAcceptList (*C function*), 1325

Gap_RemoveDeviceFromMonAdvList (*C function*), 1318
Gap_ResumeLeScStateMachine (*C function*), 1349
Gap_SaveCccd (*C function*), 1306
Gap_SaveCustomPeerInformation (*C function*), 1321
Gap_SaveDeviceName (*C function*), 1326
Gap_SaveKeys (*C function*), 1338
Gap_SendPeripheralSecurityRequest (*C function*), 1309
Gap_SendSmpKeys (*C function*), 1312
Gap_SetAdvertisingData (*C function*), 1304
Gap_SetAdvertisingParameters (*C function*), 1303
Gap_SetBondedDeviceName (*C function*), 1327
Gap_SetChannelMap (*C function*), 1339
Gap_SetConnectionCallback (*C function*), 1359
Gap_SetConnectionCteReceiveParameters (*C function*), 1350
Gap_SetConnectionCteTransmitParameters (*C function*), 1351
Gap_SetConnectionlessCteTransmitParameters (*C function*), 1349
Gap_SetDataRelatedAddressChanges (*C function*), 1302
Gap_SetDecisionInstructions (*C function*), 1317
Gap_SetDefaultConnectionSubrateParameters (*C function*), 1332
Gap_SetDefaultPairingParameters (*C function*), 1331
Gap_SetDefaultPeriodicAdvSyncTransferParams (*C function*), 1349
Gap_SetExtAdvertisingData (*C function*), 1340
Gap_SetExtAdvertisingDecisionData (*C function*), 1341
Gap_SetExtAdvertisingParameters (*C function*), 1340
Gap_SetLocalPasskey (*C function*), 1315
Gap_SetPathLossReportingParameters (*C function*), 1352
Gap_SetPeriodicAdvertisingData (*C function*), 1345
Gap_SetPeriodicAdvParameters (*C function*), 1343
Gap_SetPeriodicAdvResponseData (*C function*), 1344
Gap_SetPeriodicAdvSubeventData (*C function*), 1343
Gap_SetPeriodicAdvSyncTransferParams (*C function*), 1349
Gap_SetPeriodicSyncSubevent (*C function*), 1344
Gap_SetPrivacyMode (*C function*), 1334
Gap_SetRandomAddress (*C function*), 1330
Gap_SetScanMode (*C function*), 1316
Gap_SetScanningCallback (*C function*), 1359
Gap_SetTxPowerLevel (*C function*), 1329
Gap_StartAdvertising (*C function*), 1304
Gap_StartExtAdvertising (*C function*), 1341
Gap_StartPeriodicAdvertising (*C function*), 1345
Gap_StartScanning (*C function*), 1319
Gap_StopAdvertising (*C function*), 1305
Gap_StopExtAdvertising (*C function*), 1342
Gap_StopPeriodicAdvertising (*C function*), 1346
Gap_StopScanning (*C function*), 1319
Gap_UpdateConnectionParameters (*C function*), 1331
Gap_UpdateLeDataLength (*C function*), 1332
Gap_UpdatePeriodicAdvList (*C function*), 1346
Gap_VerifyPrivateResolvableAddress (*C function*), 1329
gapAddrReadyEvent_t (*C type*), 1500
gapAddrReadyEvent_t_tag (*C struct*), 1522, 1556
gapAddrReadyEvent_t_tag.aAddress (*C var*), 1522, 1557
gapAddrReadyEvent_t_tag.advHandle (*C var*), 1522, 1557
gapAdStructure_t (*C struct*), 1382, 1582
gapAdStructure_t.aData (*C var*), 1383, 1582
gapAdStructure_t.adType (*C var*), 1383, 1582
gapAdStructure_t.length (*C var*), 1382, 1582
gapAdType_t (*C enum*), 1286

gapAdType_t.gAd3dInformationData_c (*C enumerator*), 1288
gapAdType_t.gAdAdvertisingInterval_c (*C enumerator*), 1288
gapAdType_t.gAdAdvertisingIntervalLong_c (*C enumerator*), 1288
gapAdType_t.gAdAppearance_c (*C enumerator*), 1288
gapAdType_t.gAdChannelMapUpdateIndication_c (*C enumerator*), 1288
gapAdType_t.gAdClassOfDevice_c (*C enumerator*), 1287
gapAdType_t.gAdComplete16bitServiceList_c (*C enumerator*), 1286
gapAdType_t.gAdComplete32bitServiceList_c (*C enumerator*), 1286
gapAdType_t.gAdComplete128bitServiceList_c (*C enumerator*), 1286
gapAdType_t.gAdCompleteLocalName_c (*C enumerator*), 1287
gapAdType_t.gAdEncryptedAdvertisingData_c (*C enumerator*), 1288
gapAdType_t.gAdFlags_c (*C enumerator*), 1286
gapAdType_t.gAdIncomplete16bitServiceList_c (*C enumerator*), 1286
gapAdType_t.gAdIncomplete32bitServiceList_c (*C enumerator*), 1286
gapAdType_t.gAdIncomplete128bitServiceList_c (*C enumerator*), 1286
gapAdType_t.gAdLeDeviceAddress_c (*C enumerator*), 1288
gapAdType_t.gAdLeRole_c (*C enumerator*), 1288
gapAdType_t.gAdLeSupportedFeatures_c (*C enumerator*), 1288
gapAdType_t.gAdManufacturerSpecificData_c (*C enumerator*), 1288
gapAdType_t.gAdPeripheralConnectionIntervalRange_c (*C enumerator*), 1287
gapAdType_t.gAdPublicTargetAddress_c (*C enumerator*), 1287
gapAdType_t.gAdRandomTargetAddress_c (*C enumerator*), 1288
gapAdType_t.gAdSecurityManagerOobFlags_c (*C enumerator*), 1287
gapAdType_t.gAdSecurityManagerTkValue_c (*C enumerator*), 1287
gapAdType_t.gAdServiceData16bit_c (*C enumerator*), 1287
gapAdType_t.gAdServiceData32bit_c (*C enumerator*), 1287
gapAdType_t.gAdServiceData128bit_c (*C enumerator*), 1287
gapAdType_t.gAdServiceSolicitationList16bit_c (*C enumerator*), 1287
gapAdType_t.gAdServiceSolicitationList32bit_c (*C enumerator*), 1287
gapAdType_t.gAdServiceSolicitationList128bit_c (*C enumerator*), 1287
gapAdType_t.gAdShortenedLocalName_c (*C enumerator*), 1287
gapAdType_t.gAdSimplePairingHashC192_c (*C enumerator*), 1287
gapAdType_t.gAdSimplePairingHashC256_c (*C enumerator*), 1288
gapAdType_t.gAdSimplePairingRandomizerR192_c (*C enumerator*), 1287
gapAdType_t.gAdSimplePairingRandomizerR256_c (*C enumerator*), 1288
gapAdType_t.gAdTxPowerLevel_c (*C enumerator*), 1287
gapAdType_t.gAdUniformResourceIdentifier_c (*C enumerator*), 1288
gapAdTypeFlags_t (*C type*), 1301
gapAdvertisingCallback_t (*C type*), 1302
gapAdvertisingChannelMapFlags_t (*C enum*), 1284
gapAdvertisingChannelMapFlags_t.gAdvChanMapFlag37_c (*C enumerator*), 1284
gapAdvertisingChannelMapFlags_t.gAdvChanMapFlag38_c (*C enumerator*), 1284
gapAdvertisingChannelMapFlags_t.gAdvChanMapFlag39_c (*C enumerator*), 1285
gapAdvertisingData_t (*C struct*), 1383, 1583
gapAdvertisingData_t.aAdStructures (*C var*), 1383, 1583
gapAdvertisingData_t.cNumAdStructures (*C var*), 1383, 1583
gapAdvertisingDecisionData_t (*C struct*), 1383, 1583
gapAdvertisingDecisionData_t.dataLength (*C var*), 1383, 1583
gapAdvertisingDecisionData_t.pDecisionData (*C var*), 1383, 1583
gapAdvertisingDecisionData_t.pKey (*C var*), 1383, 1583
gapAdvertisingDecisionData_t.pPrand (*C var*), 1383, 1583
gapAdvertisingDecisionData_t.resolvableTagPresent (*C var*), 1383, 1583
gapAdvertisingEvent_t (*C struct*), 1386, 1586
gapAdvertisingEvent_t.eventData (*C union*), 1400, 1598
gapAdvertisingEvent_t.eventData (*C var*), 1386, 1586
gapAdvertisingEvent_t.eventData.advHandle (*C var*), 1401, 1598
gapAdvertisingEvent_t.eventData.advSetTerminated (*C var*), 1401, 1598
gapAdvertisingEvent_t.eventData.failReason (*C var*), 1400, 1598

gapAdvertisingEvent_t.eventData.perAdvResponse (*C var*), 1401, 1598
gapAdvertisingEvent_t.eventData.scanNotification (*C var*), 1401, 1598
gapAdvertisingEvent_t.eventData.subeventDataRequest (*C var*), 1401, 1598
gapAdvertisingEvent_t.eventType (*C var*), 1386, 1586
gapAdvertisingEventType_t (*C enum*), 1292
gapAdvertisingEventType_t.gAdvertisingCommandFailed_c (*C enumerator*), 1292
gapAdvertisingEventType_t.gAdvertisingSetTerminated_c (*C enumerator*), 1292
gapAdvertisingEventType_t.gAdvertisingStateChanged_c (*C enumerator*), 1292
gapAdvertisingEventType_t.gExtAdvertisingStateChanged_c (*C enumerator*), 1292
gapAdvertisingEventType_t.gExtScanNotification_c (*C enumerator*), 1292
gapAdvertisingEventType_t.gPerAdvResponse_c (*C enumerator*), 1292
gapAdvertisingEventType_t.gPerAdvSubeventDataRequest_c (*C enumerator*), 1292
gapAdvertisingFilterPolicy_t (*C enum*), 1285
gapAdvertisingFilterPolicy_t.gProcessAll_c (*C enumerator*), 1285
gapAdvertisingFilterPolicy_t.gProcessConnAllScanWL_c (*C enumerator*), 1285
gapAdvertisingFilterPolicy_t.gProcessFilterAcceptListOnly_c (*C enumerator*), 1285
gapAdvertisingFilterPolicy_t.gProcessScanAllConnWL_c (*C enumerator*), 1285
gapAdvertisingParameters_t (*C struct*), 1371, 1574
gapAdvertisingParameters_t.advertisingType (*C var*), 1371, 1575
gapAdvertisingParameters_t.channelMap (*C var*), 1371, 1575
gapAdvertisingParameters_t.filterPolicy (*C var*), 1371, 1575
gapAdvertisingParameters_t.maxInterval (*C var*), 1371, 1575
gapAdvertisingParameters_t.minInterval (*C var*), 1371, 1575
gapAdvertisingParameters_t.ownAddressType (*C var*), 1371, 1575
gapAdvertisingParameters_t.peerAddress (*C var*), 1371, 1575
gapAdvertisingParameters_t.peerAddressType (*C var*), 1371, 1575
gapAdvertisingSetTerminated_t (*C struct*), 1385, 1585
gapAdvertisingSetTerminated_t.deviceId (*C var*), 1386, 1585
gapAdvertisingSetTerminated_t.handle (*C var*), 1386, 1585
gapAdvertisingSetTerminated_t.numCompletedExtAdvEvents (*C var*), 1386, 1585
gapAdvertisingSetTerminated_t.status (*C var*), 1385, 1585
gapAppearance_t (*C type*), 1302
gapAppearance_tag (*C enum*), 1298
gapAppearance_tag.gBarcodeScanner_c (*C enumerator*), 1299
gapAppearance_tag.gBloodPressureArm_c (*C enumerator*), 1299
gapAppearance_tag.gBloodPressureWrist_c (*C enumerator*), 1299
gapAppearance_tag.gCardReader_c (*C enumerator*), 1299
gapAppearance_tag.gCyclingCadenceSensor_c (*C enumerator*), 1300
gapAppearance_tag.gCyclingComputer_c (*C enumerator*), 1300
gapAppearance_tag.gCyclingPowerSensor_c (*C enumerator*), 1300
gapAppearance_tag.gCyclingSpeedandCadenceSensor_c (*C enumerator*), 1300
gapAppearance_tag.gCyclingSpeedSensor_c (*C enumerator*), 1300
gapAppearance_tag.gDigitalPen_c (*C enumerator*), 1299
gapAppearance_tag.gDigitizerTablet_c (*C enumerator*), 1299
gapAppearance_tag.gFingertip_c (*C enumerator*), 1300
gapAppearance_tag.gGamepad_c (*C enumerator*), 1299
gapAppearance_tag.gGenericBarcodeScanner_c (*C enumerator*), 1299
gapAppearance_tag.gGenericBloodPressure_c (*C enumerator*), 1299
gapAppearance_tag.gGenericClock_c (*C enumerator*), 1298
gapAppearance_tag.gGenericComputer_c (*C enumerator*), 1298
gapAppearance_tag.gGenericCycling_c (*C enumerator*), 1300
gapAppearance_tag.gGenericDisplay_c (*C enumerator*), 1298
gapAppearance_tag.gGenericEyeglasses_c (*C enumerator*), 1299
gapAppearance_tag.gGenericGlucoseMeter_c (*C enumerator*), 1299
gapAppearance_tag.gGenericHeartrateSensor_c (*C enumerator*), 1299
gapAppearance_tag.gGenericKeyring_c (*C enumerator*), 1299
gapAppearance_tag.gGenericMediaPlayer_c (*C enumerator*), 1299
gapAppearance_tag.gGenericOutdoorSportsActivity_c (*C enumerator*), 1300

gapAppearance_tag.gGenericPhone_c (*C enumerator*), 1298
gapAppearance_tag.gGenericPulseOximeter_c (*C enumerator*), 1300
gapAppearance_tag.gGenericRemoteControl_c (*C enumerator*), 1299
gapAppearance_tag.gGenericRunningWalkingSensor_c (*C enumerator*), 1300
gapAppearance_tag.gGenericTag_c (*C enumerator*), 1299
gapAppearance_tag.gGenericThermometer_c (*C enumerator*), 1299
gapAppearance_tag.gGenericWatch_c (*C enumerator*), 1298
gapAppearance_tag.gGenericWeightScale_c (*C enumerator*), 1300
gapAppearance_tag.gHeartRateSensorHeartRateBelt_c (*C enumerator*), 1299
gapAppearance_tag.gHumanInterfaceDevice_c (*C enumerator*), 1299
gapAppearance_tag.gJoystick_c (*C enumerator*), 1299
gapAppearance_tag.gKeyboard_c (*C enumerator*), 1299
gapAppearance_tag.gLocationandNavigationDisplayDevice_c (*C enumerator*), 1300
gapAppearance_tag.gLocationAndNavigationPod_c (*C enumerator*), 1300
gapAppearance_tag.gLocationDisplayDevice_c (*C enumerator*), 1300
gapAppearance_tag.gLocationPod_c (*C enumerator*), 1300
gapAppearance_tag.gMouse_c (*C enumerator*), 1299
gapAppearance_tag.gRunningWalkingSensorInShoe_c (*C enumerator*), 1300
gapAppearance_tag.gRunningWalkingSensorOnHip_c (*C enumerator*), 1300
gapAppearance_tag.gRunningWalkingSensorOnShoe_c (*C enumerator*), 1300
gapAppearance_tag.gSportsWatch_c (*C enumerator*), 1298
gapAppearance_tag.gThermometerEar_c (*C enumerator*), 1299
gapAppearance_tag.gUnknown_c (*C enumerator*), 1298
gapAppearance_tag.gWristWorn_c (*C enumerator*), 1300
gapAuthenticationRejectedEvent_t (*C struct*), 1393, 1591
gapAuthenticationRejectedEvent_t.rejectReason (*C var*), 1393, 1591
gapAuthenticationRejectReason_t (*C type*), 1301
gapAutoConnectParams_t (*C struct*), 1399, 1596
gapAutoConnectParams_t.aAutoConnectData (*C var*), 1400, 1596
gapAutoConnectParams_t.cNumAddresses (*C var*), 1400, 1596
gapAutoConnectParams_t.writeInFilterAcceptList (*C var*), 1400, 1596
gapCarSupport_t (*C enum*), 1298
gapCarSupport_t.CAR_Supported (*C enumerator*), 1298
gapCarSupport_t.CAR_Unavailable (*C enumerator*), 1298
gapCarSupport_t.CAR_Unknown (*C enumerator*), 1298
gapCarSupport_t.CAR_Unsupported (*C enumerator*), 1298
gapConnCteRequestFailed_t (*C struct*), 1396, 1593
gapConnCteRequestFailed_t.status (*C var*), 1397, 1593
gapConnectedEvent_t (*C struct*), 1392, 1590
gapConnectedEvent_t.advHandle (*C var*), 1393, 1590
gapConnectedEvent_t.connectionRole (*C var*), 1393, 1590
gapConnectedEvent_t.connParameters (*C var*), 1392, 1590
gapConnectedEvent_t.localRpa (*C var*), 1393, 1590
gapConnectedEvent_t.localRpaUsed (*C var*), 1392, 1590
gapConnectedEvent_t.peerAddress (*C var*), 1392, 1590
gapConnectedEvent_t.peerAddressType (*C var*), 1392, 1590
gapConnectedEvent_t.peerRpa (*C var*), 1392, 1590
gapConnectedEvent_t.peerRpaResolved (*C var*), 1392, 1590
gapConnectedEvent_t.syncHandle (*C var*), 1393, 1590
gapConnectionCallback_t (*C type*), 1302
gapConnectionCteReceiveParams_t (*C struct*), 1381, 1581
gapConnectionCteReceiveParams_t.aAntennaIds (*C var*), 1381, 1582
gapConnectionCteReceiveParams_t.iqSamplingEnable (*C var*), 1381, 1582
gapConnectionCteReceiveParams_t.slotDurations (*C var*), 1381, 1582
gapConnectionCteReceiveParams_t.switchingPatternLength (*C var*), 1381, 1582
gapConnectionCteReqEnableParams_t (*C struct*), 1381, 1582
gapConnectionCteReqEnableParams_t.cteReqEnable (*C var*), 1382, 1582
gapConnectionCteReqEnableParams_t.cteReqInterval (*C var*), 1382, 1582

gapConnectionCteReqEnableParams_t.requestedCteLength (*C var*), 1382, 1582
gapConnectionCteReqEnableParams_t.requestedCteType (*C var*), 1382, 1582
gapConnectionCteTransmitParams_t (*C struct*), 1381, 1581
gapConnectionCteTransmitParams_t.aAntennaIds (*C var*), 1381, 1581
gapConnectionCteTransmitParams_t.cteTypes (*C var*), 1381, 1581
gapConnectionCteTransmitParams_t.switchingPatternLength (*C var*), 1381, 1581
gapConnectionEvent_t (*C struct*), 1399, 1596
gapConnectionEvent_t.eventData (*C union*), 1402, 1599
gapConnectionEvent_t.eventData (*C var*), 1399, 1596
gapConnectionEvent_t.eventData.authenticationRejectedEvent (*C var*), 1402, 1599
gapConnectionEvent_t.eventData.channelMap (*C var*), 1403, 1600
gapConnectionEvent_t.eventData.connectedEvent (*C var*), 1402, 1599
gapConnectionEvent_t.eventData.connectionUpdateComplete (*C var*), 1403, 1600
gapConnectionEvent_t.eventData.connectionUpdateRequest (*C var*), 1403, 1600
gapConnectionEvent_t.eventData.connIqReport (*C var*), 1403, 1600
gapConnectionEvent_t.eventData.cteRequestFailedEvent (*C var*), 1403, 1600
gapConnectionEvent_t.eventData.disconnectedEvent (*C var*), 1403, 1599
gapConnectionEvent_t.eventData.eattBearerStatusNotification (*C var*), 1404, 1600
gapConnectionEvent_t.eventData.eattConnectionComplete (*C var*), 1404, 1600
gapConnectionEvent_t.eventData.eattConnectionRequest (*C var*), 1404, 1600
gapConnectionEvent_t.eventData.eattReconfigureResponse (*C var*), 1404, 1600
gapConnectionEvent_t.eventData.encryptionChangedEvent (*C var*), 1402, 1599
gapConnectionEvent_t.eventData.failReason (*C var*), 1403, 1600
gapConnectionEvent_t.eventData.gapSubrateChangeEvent (*C var*), 1404, 1601
gapConnectionEvent_t.eventData.handoverConnectedEvent (*C var*), 1404, 1600
gapConnectionEvent_t.eventData.handoverDisconnectedEvent (*C var*), 1404, 1600
gapConnectionEvent_t.eventData.incomingKeypressNotification (*C var*), 1403, 1600
gapConnectionEvent_t.eventData.keyExchangeRequestEvent (*C var*), 1402, 1599
gapConnectionEvent_t.eventData.keysReceivedEvent (*C var*), 1402, 1599
gapConnectionEvent_t.eventData.leDataLengthChanged (*C var*), 1403, 1600
gapConnectionEvent_t.eventData.longTermKeyRequestEvent (*C var*), 1402, 1599
gapConnectionEvent_t.eventData.numericValueForDisplay (*C var*), 1403, 1600
gapConnectionEvent_t.eventData.pairingCompleteEvent (*C var*), 1402, 1599
gapConnectionEvent_t.eventData.pairingEvent (*C var*), 1402, 1599
gapConnectionEvent_t.eventData.passkeyForDisplay (*C var*), 1403, 1600
gapConnectionEvent_t.eventData.pathLossThreshold (*C var*), 1403, 1600
gapConnectionEvent_t.eventData.perAdvSyncTransferStatus (*C var*), 1403, 1600
gapConnectionEvent_t.eventData.peripheralSecurityRequestEvent (*C var*), 1402, 1599
gapConnectionEvent_t.eventData.remoteFeatures (*C var*), 1404, 1601
gapConnectionEvent_t.eventData.rssi_dBm (*C var*), 1403, 1600
gapConnectionEvent_t.eventData.smError (*C var*), 1404, 1600
gapConnectionEvent_t.eventData.transmitPowerInfo (*C var*), 1403, 1600
gapConnectionEvent_t.eventData.transmitPowerReporting (*C var*), 1403, 1600
gapConnectionEvent_t.eventData.txPowerLevel_dBm (*C var*), 1403, 1600
gapConnectionEvent_t.eventType (*C var*), 1399, 1596
gapConnectionEventType_t (*C enum*), 1293
gapConnectionEventType_t.gConnEvtAuthenticationRejected_c (*C enumerator*), 1294
gapConnectionEventType_t.gConnEvtChannelMapRead_c (*C enumerator*), 1295
gapConnectionEventType_t.gConnEvtChannelMapReadFailure_c (*C enumerator*), 1295
gapConnectionEventType_t.gConnEvtChanSelectionAlgorithm2_c (*C enumerator*), 1296
gapConnectionEventType_t.gConnEvtConnected_c (*C enumerator*), 1294
gapConnectionEventType_t.gConnEvtCteReceiveParamsSetupComplete_c (*C enumerator*), 1296
gapConnectionEventType_t.gConnEvtCteReqStateChanged_c (*C enumerator*), 1296
gapConnectionEventType_t.gConnEvtCteRequestFailed_c (*C enumerator*), 1296
gapConnectionEventType_t.gConnEvtCteRspStateChanged_c (*C enumerator*), 1296
gapConnectionEventType_t.gConnEvtCteTransmitParamsSetupComplete_c (*C enumerator*), 1296
gapConnectionEventType_t.gConnEvtDisconnected_c (*C enumerator*), 1295
gapConnectionEventType_t.gConnEvtEattBearerStatusNotification_c (*C enumerator*), 1297

gapConnectionEventType_t.gConnEvtEattChannelReconfigureResponse_c (*C enumerator*), 1297
gapConnectionEventType_t.gConnEvtEattConnectionComplete_c (*C enumerator*), 1297
gapConnectionEventType_t.gConnEvtEattConnectionRequest_c (*C enumerator*), 1297
gapConnectionEventType_t.gConnEvtEncryptionChanged_c (*C enumerator*), 1294
gapConnectionEventType_t.gConnEvtEnhancedReadTransmitPowerLevel_c (*C enumerator*), 1296
gapConnectionEventType_t.gConnEvtHandoverConnected_c (*C enumerator*), 1297
gapConnectionEventType_t.gConnEvtIqReportReceived_c (*C enumerator*), 1296
gapConnectionEventType_t.gConnEvtKeyExchangeRequest_c (*C enumerator*), 1294
gapConnectionEventType_t.gConnEvtKeysReceived_c (*C enumerator*), 1294
gapConnectionEventType_t.gConnEvtLeDataLengthChanged_c (*C enumerator*), 1295
gapConnectionEventType_t.gConnEvtLeScDisplayNumericValue_c (*C enumerator*), 1295
gapConnectionEventType_t.gConnEvtLeScKeypressNotification_c (*C enumerator*), 1295
gapConnectionEventType_t.gConnEvtLeScOobDataRequest_c (*C enumerator*), 1295
gapConnectionEventType_t.gConnEvtLeSetDataLengthFailure_c (*C enumerator*), 1297
gapConnectionEventType_t.gConnEvtLeSetLocalPeripheralLatencyEnableComplete_c (*C enumerator*), 1297
gapConnectionEventType_t.gConnEvtLeSubrateChange_c (*C enumerator*), 1297
gapConnectionEventType_t.gConnEvtLongTermKeyRequest_c (*C enumerator*), 1294
gapConnectionEventType_t.gConnEvtOobRequest_c (*C enumerator*), 1294
gapConnectionEventType_t.gConnEvtPairingAlreadyStarted_c (*C enumerator*), 1296
gapConnectionEventType_t.gConnEvtPairingComplete_c (*C enumerator*), 1295
gapConnectionEventType_t.gConnEvtPairingNoLtk_c (*C enumerator*), 1296
gapConnectionEventType_t.gConnEvtPairingRequest_c (*C enumerator*), 1294
gapConnectionEventType_t.gConnEvtPairingResponse_c (*C enumerator*), 1294
gapConnectionEventType_t.gConnEvtParameterUpdateComplete_c (*C enumerator*), 1295
gapConnectionEventType_t.gConnEvtParameterUpdateRequest_c (*C enumerator*), 1295
gapConnectionEventType_t.gConnEvtPasskeyDisplay_c (*C enumerator*), 1294
gapConnectionEventType_t.gConnEvtPasskeyRequest_c (*C enumerator*), 1294
gapConnectionEventType_t.gConnEvtPathLossReportingParamsSetupComplete_c (*C enumerator*), 1296
gapConnectionEventType_t.gConnEvtPathLossReportingStateChanged_c (*C enumerator*), 1296
gapConnectionEventType_t.gConnEvtPathLossThreshold_c (*C enumerator*), 1296
gapConnectionEventType_t.gConnEvtPeripheralSecurityRequest_c (*C enumerator*), 1294
gapConnectionEventType_t.gConnEvtPowerReadFailure_c (*C enumerator*), 1295
gapConnectionEventType_t.gConnEvtRemoteFeaturesRead_c (*C enumerator*), 1297
gapConnectionEventType_t.gConnEvtRssiRead_c (*C enumerator*), 1295
gapConnectionEventType_t.gConnEvtSmError_c (*C enumerator*), 1297
gapConnectionEventType_t.gConnEvtTransmitPowerReporting_c (*C enumerator*), 1296
gapConnectionEventType_t.gConnEvtTransmitPowerReportingStateChanged_c (*C enumerator*), 1296
gapConnectionEventType_t.gConnEvtTxPowerLevelRead_c (*C enumerator*), 1295
gapConnectionEventType_t.gHandoverDisconnected_c (*C enumerator*), 1297
gapConnectionFromPawrParameters_t (*C struct*), 1377, 1579
gapConnectionFromPawrParameters_t.advHandle (*C var*), 1379, 1580
gapConnectionFromPawrParameters_t.connEventLengthMax (*C var*), 1378, 1579
gapConnectionFromPawrParameters_t.connEventLengthMin (*C var*), 1378, 1579
gapConnectionFromPawrParameters_t.connIntervalMax (*C var*), 1378, 1579
gapConnectionFromPawrParameters_t.connIntervalMin (*C var*), 1378, 1579
gapConnectionFromPawrParameters_t.connLatency (*C var*), 1378, 1579
gapConnectionFromPawrParameters_t.filterPolicy (*C var*), 1378, 1579
gapConnectionFromPawrParameters_t.initiatingPHYs (*C var*), 1378, 1579
gapConnectionFromPawrParameters_t.ownAddressType (*C var*), 1378, 1579
gapConnectionFromPawrParameters_t.peerAddress (*C var*), 1378, 1579
gapConnectionFromPawrParameters_t.peerAddressType (*C var*), 1378, 1579
gapConnectionFromPawrParameters_t.scanInterval (*C var*), 1377, 1579
gapConnectionFromPawrParameters_t.scanWindow (*C var*), 1378, 1579
gapConnectionFromPawrParameters_t.subevent (*C var*), 1379, 1580
gapConnectionFromPawrParameters_t.supervisionTimeout (*C var*), 1378, 1579

gapConnectionFromPawrParameters_t.usePeerIdentityAddress (*C var*), 1378, 1579
gapConnectionlessCteTransmitParams_t (*C struct*), 1380, 1580
gapConnectionlessCteTransmitParams_t.aAntennaIds (*C var*), 1380, 1581
gapConnectionlessCteTransmitParams_t.cteCount (*C var*), 1380, 1581
gapConnectionlessCteTransmitParams_t.cteLength (*C var*), 1380, 1581
gapConnectionlessCteTransmitParams_t.cteType (*C var*), 1380, 1581
gapConnectionlessCteTransmitParams_t.handle (*C var*), 1380, 1581
gapConnectionlessCteTransmitParams_t.switchingPatternLength (*C var*), 1380, 1581
gapConnectionlessIqReport_t (*C struct*), 1391, 1589
gapConnectionlessIqReport_t.aI_samples (*C var*), 1391, 1589
gapConnectionlessIqReport_t.aQ_samples (*C var*), 1391, 1589
gapConnectionlessIqReport_t.channelIndex (*C var*), 1391, 1589
gapConnectionlessIqReport_t.cteType (*C var*), 1391, 1589
gapConnectionlessIqReport_t.packetStatus (*C var*), 1391, 1589
gapConnectionlessIqReport_t.periodicEventCounter (*C var*), 1391, 1589
gapConnectionlessIqReport_t.rssi (*C var*), 1391, 1589
gapConnectionlessIqReport_t.rssiAntennaId (*C var*), 1391, 1589
gapConnectionlessIqReport_t.sampleCount (*C var*), 1391, 1589
gapConnectionlessIqReport_t.slotDurations (*C var*), 1391, 1589
gapConnectionlessIqReport_t.syncHandle (*C var*), 1391, 1589
gapConnectionlessIqSamplingParams_t (*C struct*), 1380, 1581
gapConnectionlessIqSamplingParams_t.aAntennaIds (*C var*), 1381, 1581
gapConnectionlessIqSamplingParams_t.iqSamplingEnable (*C var*), 1380, 1581
gapConnectionlessIqSamplingParams_t.maxSampledCtes (*C var*), 1381, 1581
gapConnectionlessIqSamplingParams_t.slotDurations (*C var*), 1381, 1581
gapConnectionlessIqSamplingParams_t.switchingPatternLength (*C var*), 1381, 1581
gapConnectionParameters_t (*C struct*), 1379, 1580
gapConnectionParameters_t.centralClockAccuracy (*C var*), 1379, 1580
gapConnectionParameters_t.connInterval (*C var*), 1379, 1580
gapConnectionParameters_t.connLatency (*C var*), 1379, 1580
gapConnectionParameters_t.supervisionTimeout (*C var*), 1379, 1580
gapConnectionRequestParameters_t (*C struct*), 1376, 1578
gapConnectionRequestParameters_t.connEventLengthMax (*C var*), 1377, 1579
gapConnectionRequestParameters_t.connEventLengthMin (*C var*), 1377, 1579
gapConnectionRequestParameters_t.connIntervalMax (*C var*), 1377, 1578
gapConnectionRequestParameters_t.connIntervalMin (*C var*), 1377, 1578
gapConnectionRequestParameters_t.connLatency (*C var*), 1377, 1579
gapConnectionRequestParameters_t.filterPolicy (*C var*), 1376, 1578
gapConnectionRequestParameters_t.initiatingPHYs (*C var*), 1377, 1579
gapConnectionRequestParameters_t.ownAddressType (*C var*), 1376, 1578
gapConnectionRequestParameters_t.peerAddress (*C var*), 1377, 1578
gapConnectionRequestParameters_t.peerAddressType (*C var*), 1377, 1578
gapConnectionRequestParameters_t.scanInterval (*C var*), 1376, 1578
gapConnectionRequestParameters_t.scanWindow (*C var*), 1376, 1578
gapConnectionRequestParameters_t.supervisionTimeout (*C var*), 1377, 1579
gapConnectionRequestParameters_t.usePeerIdentityAddress (*C var*), 1377, 1579
gapConnectionSecurityInformation_t (*C struct*), 1369, 1574
gapConnectionSecurityInformation_t.authenticated (*C var*), 1370, 1574
gapConnectionSecurityInformation_t.authorizedToRead (*C var*), 1370, 1574
gapConnectionSecurityInformation_t.authorizedToWrite (*C var*), 1370, 1574
gapConnectionSubrateParameters_t (*C struct*), 1379, 1580
gapConnectionSubrateParameters_t.continuationNumber (*C var*), 1379, 1580
gapConnectionSubrateParameters_t.latencyMax (*C var*), 1379, 1580
gapConnectionSubrateParameters_t.subrateMax (*C var*), 1379, 1580
gapConnectionSubrateParameters_t.subrateMin (*C var*), 1379, 1580
gapConnectionSubrateParameters_t.supervisionTimeout (*C var*), 1379, 1580
gapConnIqReport_t (*C struct*), 1396, 1592
gapConnIqReport_t.aI_samples (*C var*), 1396, 1593

gapConnIqReport_t.aQ_samples (*C var*), 1396, 1593
gapConnIqReport_t.connEventCounter (*C var*), 1396, 1593
gapConnIqReport_t.cteType (*C var*), 1396, 1593
gapConnIqReport_t.dataChannelIndex (*C var*), 1396, 1593
gapConnIqReport_t.packetStatus (*C var*), 1396, 1593
gapConnIqReport_t.rssi (*C var*), 1396, 1593
gapConnIqReport_t.rssiAntennaId (*C var*), 1396, 1593
gapConnIqReport_t.rxPhy (*C var*), 1396, 1593
gapConnIqReport_t.sampleCount (*C var*), 1396, 1593
gapConnIqReport_t.slotDurations (*C var*), 1396, 1593
gapConnLeDataLengthChanged_t (*C struct*), 1395, 1592
gapConnLeDataLengthChanged_t.maxRxOctets (*C var*), 1395, 1592
gapConnLeDataLengthChanged_t.maxRxTime (*C var*), 1395, 1592
gapConnLeDataLengthChanged_t.maxTxOctets (*C var*), 1395, 1592
gapConnLeDataLengthChanged_t.maxTxTime (*C var*), 1395, 1592
gapConnParamsUpdateComplete_t (*C struct*), 1395, 1592
gapConnParamsUpdateComplete_t.connInterval (*C var*), 1395, 1592
gapConnParamsUpdateComplete_t.connLatency (*C var*), 1395, 1592
gapConnParamsUpdateComplete_t.status (*C var*), 1592
gapConnParamsUpdateComplete_t.supervisionTimeout (*C var*), 1395, 1592
gapConnParamsUpdateReq_t (*C struct*), 1394, 1592
gapConnParamsUpdateReq_t.intervalMax (*C var*), 1395, 1592
gapConnParamsUpdateReq_t.intervalMin (*C var*), 1395, 1592
gapConnParamsUpdateReq_t.peripheralLatency (*C var*), 1395, 1592
gapConnParamsUpdateReq_t.timeoutMultiplier (*C var*), 1395, 1592
gapControllerTestCmd_t (*C enum*), 1291
gapControllerTestCmd_t.gControllerTestCmdEnd_c (*C enumerator*), 1291
gapControllerTestCmd_t.gControllerTestCmdStartRx_c (*C enumerator*), 1291
gapControllerTestCmd_t.gControllerTestCmdStartTx_c (*C enumerator*), 1291
gapControllerTestEvent_t (*C struct*), 1521, 1555
gapControllerTestEvent_t.receivedPackets (*C var*), 1555
gapControllerTestEvent_t.testEventType (*C var*), 1555
gapControllerTestEventType_t (*C enum*), 1492
gapControllerTestEventType_t.gControllerReceiverTestStarted_c (*C enumerator*), 1492
gapControllerTestEventType_t.gControllerTestEnded_c (*C enumerator*), 1492
gapControllerTestEventType_t.gControllerTransmitterTestStarted_c (*C enumerator*), 1492
gapControllerTestTxType_t (*C type*), 1302
gapControllerTestTxType_tag (*C enum*), 1291
gapControllerTestTxType_tag.gControllerTestTx00_c (*C enumerator*), 1291
gapControllerTestTxType_tag.gControllerTestTx0F_c (*C enumerator*), 1292
gapControllerTestTxType_tag.gControllerTestTx55_c (*C enumerator*), 1292
gapControllerTestTxType_tag.gControllerTestTxAA_c (*C enumerator*), 1291
gapControllerTestTxType_tag.gControllerTestTxF0_c (*C enumerator*), 1291
gapControllerTestTxType_tag.gControllerTestTxFF_c (*C enumerator*), 1291
gapControllerTestTxType_tag.gControllerTestTxPrbs9_c (*C enumerator*), 1291
gapControllerTestTxType_tag.gControllerTestTxPrbs15_c (*C enumerator*), 1291
gapCreateSyncReqFilterPolicy_t (*C type*), 1301
gapCreateSyncReqFilterPolicy_tag (*C enum*), 1286
gapCreateSyncReqFilterPolicy_tag.gUseCommandParameters_c (*C enumerator*), 1286
gapCreateSyncReqFilterPolicy_tag.gUsePeriodicAdvList_c (*C enumerator*), 1286
gapCreateSyncReqOptions_t (*C type*), 1301
gapCreateSyncReqOptions_tag (*C struct*), 1375, 1577
gapCreateSyncReqOptions_tag.duplicateFilteringEnabled (*C var*), 1578
gapCreateSyncReqOptions_tag.filterPolicy (*C var*), 1578
gapCreateSyncReqOptions_tag.reportingEnabled (*C var*), 1578
gapDecisionInstructionsAdvAChecks_t (*C enum*), 1290
gapDecisionInstructionsAdvAChecks_t.gDIAAC_AdvAinFilterAcceptList_c (*C enumerator*), 1290
gapDecisionInstructionsAdvAChecks_t.gDIAAC_AdvAmatchAddress1_c (*C enumerator*), 1290

gapDecisionInstructionsAdvAChecks_t.gDIAAC_AdvAmatchAddress1orAddress2_c (*C enumerator*), 1290
gapDecisionInstructionsAdvMode_t (*C type*), 1301
gapDecisionInstructionsData_t (*C type*), 1301
gapDecisionInstructionsData_tag (*C struct*), 1383, 1583
gapDecisionInstructionsData_tag.passCriteria (*C var*), 1583
gapDecisionInstructionsData_tag.relevantField (*C var*), 1583
gapDecisionInstructionsData_tag.testGroup (*C var*), 1583
gapDecisionInstructionsData_tag.testParameters (*C union*), 1400, 1597
gapDecisionInstructionsData_tag.testParameters (*C var*), 1583
gapDecisionInstructionsData_tag.testParameters.advA (*C struct*), 1400, 1598
gapDecisionInstructionsData_tag.testParameters.advA (*C var*), 1400, 1597
gapDecisionInstructionsData_tag.testParameters.advA.address1 (*C var*), 1598
gapDecisionInstructionsData_tag.testParameters.advA.address1Type (*C var*), 1598
gapDecisionInstructionsData_tag.testParameters.advA.address2 (*C var*), 1598
gapDecisionInstructionsData_tag.testParameters.advA.address2Type (*C var*), 1598
gapDecisionInstructionsData_tag.testParameters.advA.check (*C var*), 1598
gapDecisionInstructionsData_tag.testParameters.advMode (*C var*), 1400, 1597
gapDecisionInstructionsData_tag.testParameters.arbitraryData (*C struct*), 1400, 1597
gapDecisionInstructionsData_tag.testParameters.arbitraryData (*C var*), 1400, 1597
gapDecisionInstructionsData_tag.testParameters.arbitraryData.mask (*C var*), 1597
gapDecisionInstructionsData_tag.testParameters.arbitraryData.target (*C var*), 1597
gapDecisionInstructionsData_tag.testParameters.pathLoss (*C struct*), 1400, 1597
gapDecisionInstructionsData_tag.testParameters.pathLoss (*C var*), 1400, 1597
gapDecisionInstructionsData_tag.testParameters.pathLoss.max (*C var*), 1598
gapDecisionInstructionsData_tag.testParameters.pathLoss.min (*C var*), 1598
gapDecisionInstructionsData_tag.testParameters.resolvableTagKey (*C var*), 1400, 1597
gapDecisionInstructionsData_tag.testParameters.rssi (*C struct*), 1400, 1597
gapDecisionInstructionsData_tag.testParameters.rssi (*C var*), 1400, 1597
gapDecisionInstructionsData_tag.testParameters.rssi.max (*C var*), 1597
gapDecisionInstructionsData_tag.testParameters.rssi.min (*C var*), 1597
gapDecisionInstructionsRelevantField_t (*C enum*), 1288
gapDecisionInstructionsRelevantField_t.gDIRF_AdvAddress_c (*C enumerator*), 1289
gapDecisionInstructionsRelevantField_t.gDIRF_AdvMode_c (*C enumerator*), 1289
gapDecisionInstructionsRelevantField_t.gDIRF_ArbitraryDataOfAtLeast_1Byte_c (*C enumerator*), 1289
gapDecisionInstructionsRelevantField_t.gDIRF_ArbitraryDataOfAtLeast_2Bytes_c (*C enumerator*), 1289
gapDecisionInstructionsRelevantField_t.gDIRF_ArbitraryDataOfAtLeast_3Bytes_c (*C enumerator*), 1289
gapDecisionInstructionsRelevantField_t.gDIRF_ArbitraryDataOfAtLeast_4Bytes_c (*C enumerator*), 1289
gapDecisionInstructionsRelevantField_t.gDIRF_ArbitraryDataOfAtLeast_5Bytes_c (*C enumerator*), 1289
gapDecisionInstructionsRelevantField_t.gDIRF_ArbitraryDataOfAtLeast_6Bytes_c (*C enumerator*), 1289
gapDecisionInstructionsRelevantField_t.gDIRF_ArbitraryDataOfAtLeast_7Bytes_c (*C enumerator*), 1289
gapDecisionInstructionsRelevantField_t.gDIRF_ArbitraryDataOfAtLeast_8Bytes_c (*C enumerator*), 1289
gapDecisionInstructionsRelevantField_t.gDIRF_ArbitraryDataOfAtMost_1Byte_c (*C enumerator*), 1289
gapDecisionInstructionsRelevantField_t.gDIRF_ArbitraryDataOfAtMost_2Bytes_c (*C enumerator*), 1289
gapDecisionInstructionsRelevantField_t.gDIRF_ArbitraryDataOfAtMost_3Bytes_c (*C enumerator*), 1289
gapDecisionInstructionsRelevantField_t.gDIRF_ArbitraryDataOfAtMost_4Bytes_c (*C enumerator*), 1290

gapDecisionInstructionsRelevantField_t.gDIRF_ArbitraryDataOfAtMost_5Bytes_c (*C enumerator*), 1290
gapDecisionInstructionsRelevantField_t.gDIRF_ArbitraryDataOfAtMost_6Bytes_c (*C enumerator*), 1290
gapDecisionInstructionsRelevantField_t.gDIRF_ArbitraryDataOfAtMost_7Bytes_c (*C enumerator*), 1290
gapDecisionInstructionsRelevantField_t.gDIRF_ArbitraryDataOfAtMost_8Bytes_c (*C enumerator*), 1290
gapDecisionInstructionsRelevantField_t.gDIRF_ArbitraryDataOfExactly_1Byte_c (*C enumerator*), 1289
gapDecisionInstructionsRelevantField_t.gDIRF_ArbitraryDataOfExactly_2Bytes_c (*C enumerator*), 1289
gapDecisionInstructionsRelevantField_t.gDIRF_ArbitraryDataOfExactly_3Bytes_c (*C enumerator*), 1289
gapDecisionInstructionsRelevantField_t.gDIRF_ArbitraryDataOfExactly_4Bytes_c (*C enumerator*), 1289
gapDecisionInstructionsRelevantField_t.gDIRF_ArbitraryDataOfExactly_5Bytes_c (*C enumerator*), 1289
gapDecisionInstructionsRelevantField_t.gDIRF_ArbitraryDataOfExactly_6Bytes_c (*C enumerator*), 1289
gapDecisionInstructionsRelevantField_t.gDIRF_ArbitraryDataOfExactly_7Bytes_c (*C enumerator*), 1289
gapDecisionInstructionsRelevantField_t.gDIRF_ArbitraryDataOfExactly_8Bytes_c (*C enumerator*), 1289
gapDecisionInstructionsRelevantField_t.gDIRF_PathLoss_c (*C enumerator*), 1289
gapDecisionInstructionsRelevantField_t.gDIRF_ResolvableTag_c (*C enumerator*), 1288
gapDecisionInstructionsRelevantField_t.gDIRF_RSSI_c (*C enumerator*), 1289
gapDecisionInstructionsTestGroup_t (*C enum*), 1290
gapDecisionInstructionsTestGroup_t.gDITG_NewTestGroup_c (*C enumerator*), 1290
gapDecisionInstructionsTestGroup_t.gDITG_SameTestGroup_c (*C enumerator*), 1290
gapDecisionInstructionsTestPassCriteria_t (*C enum*), 1290
gapDecisionInstructionsTestPassCriteria_t.gDITPC_Always_c (*C enumerator*), 1290
gapDecisionInstructionsTestPassCriteria_t.gDITPC_CheckFails_c (*C enumerator*), 1290
gapDecisionInstructionsTestPassCriteria_t.gDITPC_CheckFailsOrRelevantFieldNotPresent_c (*C enumerator*), 1290
gapDecisionInstructionsTestPassCriteria_t.gDITPC_CheckPasses_c (*C enumerator*), 1290
gapDecisionInstructionsTestPassCriteria_t.gDITPC_CheckPassesOrRelevantFieldNotPresent_c (*C enumerator*), 1290
gapDecisionInstructionsTestPassCriteria_t.gDITPC_Never_c (*C enumerator*), 1290
gapDecisionInstructionsTestPassCriteria_t.gDITPC_RelevantFieldNotPresent_c (*C enumerator*), 1290
gapDecisionInstructionsTestPassCriteria_t.gDITPC_RelevantFieldPresent_c (*C enumerator*), 1290
gapDeviceSecurityRequirements_t (*C struct*), 1369, 1573
gapDeviceSecurityRequirements_t.aServiceSecurityRequirements (*C var*), 1369, 1573
gapDeviceSecurityRequirements_t.cNumServices (*C var*), 1369, 1573
gapDeviceSecurityRequirements_t.pSecurityRequirements (*C var*), 1369, 1573
gapDisconnectedEvent_t (*C struct*), 1394, 1591
gapDisconnectedEvent_t.reason (*C var*), 1394, 1592
gapDisconnectionReason_t (*C type*), 1302
gapEattBearerStatus_t (*C enum*), 1297
gapEattBearerStatus_t.gEnhancedBearerActive_c (*C enumerator*), 1297
gapEattBearerStatus_t.gEnhancedBearerDisconnected_c (*C enumerator*), 1297
gapEattBearerStatus_t.gEnhancedBearerNoPeerCredits_c (*C enumerator*), 1297
gapEattBearerStatus_t.gEnhancedBearerStatusEnd_c (*C enumerator*), 1298
gapEattBearerStatus_t.gEnhancedBearerSuspendedNoLocalCredits_c (*C enumerator*), 1297
gapEattBearerStatusNotification_t (*C struct*), 1398, 1595
gapEattBearerStatusNotification_t.bearerId (*C var*), 1595
gapEattBearerStatusNotification_t.status (*C var*), 1595

gapEattConnectionComplete_t (*C struct*), 1398, 1595
gapEattConnectionComplete_t.aBearerIds (*C var*), 1595
gapEattConnectionComplete_t.cBearers (*C var*), 1595
gapEattConnectionComplete_t.mtu (*C var*), 1595
gapEattConnectionComplete_t.status (*C var*), 1595
gapEattConnectionRequest_t (*C struct*), 1398, 1594
gapEattConnectionRequest_t.cBearers (*C var*), 1594
gapEattConnectionRequest_t.initialCredits (*C var*), 1594
gapEattConnectionRequest_t.mtu (*C var*), 1594
gapEattReconfigureResponse_t (*C struct*), 1398, 1595
gapEattReconfigureResponse_t.aBearerIds (*C var*), 1595
gapEattReconfigureResponse_t.cBearers (*C var*), 1595
gapEattReconfigureResponse_t.localMtu (*C var*), 1595
gapEattReconfigureResponse_t.status (*C var*), 1595
gapEncryptionChangedEvent_t (*C struct*), 1394, 1591
gapEncryptionChangedEvent_t.newEncryptionState (*C var*), 1394, 1591
gapExtAdvertisingParameters_t (*C type*), 1301
gapExtAdvertisingParameters_tag (*C struct*), 1371, 1575
gapExtAdvertisingParameters_tag.channelMap (*C var*), 1372, 1575
gapExtAdvertisingParameters_tag.enableScanReqNotification (*C var*), 1373, 1576
gapExtAdvertisingParameters_tag.extAdvProperties (*C var*), 1372, 1575
gapExtAdvertisingParameters_tag.filterPolicy (*C var*), 1372, 1575
gapExtAdvertisingParameters_tag.handle (*C var*), 1372, 1575
gapExtAdvertisingParameters_tag.maxInterval (*C var*), 1372, 1575
gapExtAdvertisingParameters_tag.minInterval (*C var*), 1372, 1575
gapExtAdvertisingParameters_tag.ownAddressType (*C var*), 1372, 1575
gapExtAdvertisingParameters_tag.ownRandomAddr (*C var*), 1372, 1575
gapExtAdvertisingParameters_tag.peerAddress (*C var*), 1372, 1575
gapExtAdvertisingParameters_tag.peerAddressType (*C var*), 1372, 1575
gapExtAdvertisingParameters_tag.primaryAdvPhyOptions (*C var*), 1373, 1576
gapExtAdvertisingParameters_tag.primaryPHY (*C var*), 1372, 1576
gapExtAdvertisingParameters_tag.secondaryAdvMaxSkip (*C var*), 1373, 1576
gapExtAdvertisingParameters_tag.secondaryAdvPhyOptions (*C var*), 1373, 1576
gapExtAdvertisingParameters_tag.secondaryPHY (*C var*), 1373, 1576
gapExtAdvertisingParameters_tag.SID (*C var*), 1372, 1575
gapExtAdvertisingParameters_tag.txPower (*C var*), 1372, 1575
gapExtScannedDevice_t (*C struct*), 1387, 1586
gapExtScannedDevice_t.aAddress (*C var*), 1387, 1587
gapExtScannedDevice_t.addressType (*C var*), 1387, 1587
gapExtScannedDevice_t.advertisingAddressResolved (*C var*), 1387, 1587
gapExtScannedDevice_t.advEventProperties (*C var*), 1388, 1587
gapExtScannedDevice_t.dataLength (*C var*), 1388, 1587
gapExtScannedDevice_t.directRpa (*C var*), 1388, 1587
gapExtScannedDevice_t.directRpaType (*C var*), 1388, 1587
gapExtScannedDevice_t.directRpaUsed (*C var*), 1388, 1587
gapExtScannedDevice_t.pData (*C var*), 1388, 1587
gapExtScannedDevice_t.periodicAdvInterval (*C var*), 1388, 1587
gapExtScannedDevice_t.primaryPHY (*C var*), 1388, 1587
gapExtScannedDevice_t.rssi (*C var*), 1388, 1587
gapExtScannedDevice_t.secondaryPHY (*C var*), 1388, 1587
gapExtScannedDevice_t.SID (*C var*), 1387, 1587
gapExtScannedDevice_t.txPower (*C var*), 1388, 1587
gapExtScanNotification_t (*C struct*), 1385, 1584
gapExtScanNotification_t.aScannerAddr (*C var*), 1385, 1585
gapExtScanNotification_t.handle (*C var*), 1385, 1585
gapExtScanNotification_t.scannerAddrResolved (*C var*), 1385, 1585
gapExtScanNotification_t.scannerAddrType (*C var*), 1385, 1585
gapFilterDuplicates_t (*C enum*), 1285

gapFilterDuplicates_t.gGapDuplicateFilteringDisable_c (*C enumerator*), 1285
gapFilterDuplicates_t.gGapDuplicateFilteringEnable_c (*C enumerator*), 1285
gapFilterDuplicates_t.gGapDuplicateFilteringPeriodicEnable_c (*C enumerator*), 1286
gapGenerateDHKeyV2Params_t (*C struct*), 1380, 1580
gapGenerateDHKeyV2Params_t.keyType (*C var*), 1380, 1580
gapGenerateDHKeyV2Params_t.remoteP256PublicKey (*C var*), 1380, 1580
gapGenericCallback_t (*C type*), 1501
gapGenericEvent_t (*C struct*), 1534, 1566
gapGenericEvent_t.eventData (*C union*), 1536, 1569
gapGenericEvent_t.eventData (*C var*), 1534, 1567
gapGenericEvent_t.eventData.aAddress (*C var*), 1536, 1569
gapGenericEvent_t.eventData.aControllerLocalRPA (*C var*), 1537, 1570
gapGenericEvent_t.eventData.addrReady (*C var*), 1536, 1570
gapGenericEvent_t.eventData.advHandle (*C var*), 1536, 1570
gapGenericEvent_t.eventData.advTxPowerLevel_dBm (*C var*), 1536, 1570
gapGenericEvent_t.eventData.antennaInformation (*C var*), 1537, 1570
gapGenericEvent_t.eventData.bondCreatedEvent (*C var*), 1537, 1570
gapGenericEvent_t.eventData.deviceId (*C var*), 1537, 1570
gapGenericEvent_t.eventData.filterAcceptListSize (*C var*), 1536, 1569
gapGenericEvent_t.eventData.gapRemoteVersionInfoRead (*C var*), 1539, 1571
gapGenericEvent_t.eventData.gapSkdReport (*C var*), 1539, 1571
gapGenericEvent_t.eventData.getConnParams (*C var*), 1537, 1570
gapGenericEvent_t.eventData.handoverAnchorMonitor (*C var*), 1538, 1571
gapGenericEvent_t.eventData.handoverAnchorMonitorPacket (*C var*), 1539, 1571
gapGenericEvent_t.eventData.handoverAnchorMonitorPacketContinue (*C var*), 1539, 1571
gapGenericEvent_t.eventData.handoverAnchorNotificationStateChanged (*C var*), 1539, 1571
gapGenericEvent_t.eventData.handoverAnchorSearchStart (*C var*), 1538, 1571
gapGenericEvent_t.eventData.handoverAnchorSearchStop (*C var*), 1538, 1571
gapGenericEvent_t.eventData.handoverApplyConnectionUpdateProcedure (*C var*), 1539, 1572
gapGenericEvent_t.eventData.handoverConnect (*C var*), 1538, 1571
gapGenericEvent_t.eventData.handoverConnectionUpdateProcedure (*C var*), 1539, 1572
gapGenericEvent_t.eventData.handoverConnParamUpdate (*C var*), 1539, 1571
gapGenericEvent_t.eventData.handoverGetCsLlContext (*C var*), 1538, 1571
gapGenericEvent_t.eventData.handoverGetData (*C var*), 1538, 1571
gapGenericEvent_t.eventData.handoverGetTime (*C var*), 1538, 1571
gapGenericEvent_t.eventData.handoverLlPendingDataIndication (*C var*), 1539, 1572
gapGenericEvent_t.eventData.handoverResumeTransmitComplete (*C var*), 1539, 1571
gapGenericEvent_t.eventData.handoverSetData (*C var*), 1538, 1571
gapGenericEvent_t.eventData.handoverSuspendTransmitComplete (*C var*), 1538, 1571
gapGenericEvent_t.eventData.handoverTimeSync (*C var*), 1538, 1571
gapGenericEvent_t.eventData.handoverUpdateConnParams (*C var*), 1539, 1571
gapGenericEvent_t.eventData.initCompleteData (*C var*), 1537, 1570
gapGenericEvent_t.eventData.internalError (*C var*), 1536, 1569
gapGenericEvent_t.eventData.leGenerateDhKeyCompleteEvent (*C var*), 1538, 1571
gapGenericEvent_t.eventData.localOobData (*C var*), 1536, 1570
gapGenericEvent_t.eventData.monAdvListSize (*C var*), 1539, 1572
gapGenericEvent_t.eventData.newControllerPrivacyState (*C var*), 1537, 1570
gapGenericEvent_t.eventData.newHostPrivacyState (*C var*), 1536, 1570
gapGenericEvent_t.eventData.notifEvent (*C var*), 1537, 1570
gapGenericEvent_t.eventData.pawrAdvHandle (*C var*), 1538, 1571
gapGenericEvent_t.eventData.pawrSyncHandle (*C var*), 1538, 1571
gapGenericEvent_t.eventData.perAdvSetDefaultPerAdvSyncTransferParams (*C var*), 1538, 1571
gapGenericEvent_t.eventData.perAdvSetInfoTransfer (*C var*), 1537, 1570
gapGenericEvent_t.eventData.perAdvSetSyncTransferParams (*C var*), 1538, 1570
gapGenericEvent_t.eventData.perAdvSyncTransfer (*C var*), 1537, 1570
gapGenericEvent_t.eventData.perAdvSyncTransferEnable (*C var*), 1537, 1570
gapGenericEvent_t.eventData.perAdvSyncTransferReceived (*C var*), 1538, 1571
gapGenericEvent_t.eventData.phyEvent (*C var*), 1537, 1570

gapGenericEvent_t.eventData.setupFailError (*C var*), 1536, 1570
gapGenericEvent_t.eventData.syncHandle (*C var*), 1537, 1570
gapGenericEvent_t.eventData.testEvent (*C var*), 1537, 1570
gapGenericEvent_t.eventData.txPowerLevelSetStatus (*C var*), 1537, 1570
gapGenericEvent_t.eventData.unitaryTestData (*C var*), 1539, 1572
gapGenericEvent_t.eventData.vendorDebug (*C var*), 1539, 1572
gapGenericEvent_t.eventData.verified (*C var*), 1536, 1570
gapGenericEvent_t.eventType (*C var*), 1534, 1567
gapGenericEventType_t (*C enum*), 1478
gapGenericEventType_t.gAdvertisingDataSetupComplete_c (*C enumerator*), 1479
gapGenericEventType_t.gAdvertisingParametersSetupComplete_c (*C enumerator*), 1479
gapGenericEventType_t.gAdvertisingSetupFailed_c (*C enumerator*), 1479
gapGenericEventType_t.gAdvTxPowerLevelRead_c (*C enumerator*), 1479
gapGenericEventType_t.gAntennaInformationRead_c (*C enumerator*), 1481
gapGenericEventType_t.gBondCreatedEvent_c (*C enumerator*), 1480
gapGenericEventType_t.gChannelMapSet_c (*C enumerator*), 1480
gapGenericEventType_t.gConnectionlessCteTransmitParamsSetupComplete_c (*C enumerator*), 1481
gapGenericEventType_t.gConnectionlessCteTransmitStateChanged_c (*C enumerator*), 1481
gapGenericEventType_t.gConnectionlessIqSamplingStateChanged_c (*C enumerator*), 1481
gapGenericEventType_t.gConnEvtLeGenerateDhKeyComplete_c (*C enumerator*), 1482
gapGenericEventType_t.gControllerLocalRPARead_c (*C enumerator*), 1481
gapGenericEventType_t.gControllerNotificationEvent_c (*C enumerator*), 1480
gapGenericEventType_t.gControllerPrivacyStateChanged_c (*C enumerator*), 1480
gapGenericEventType_t.gControllerTestEvent_c (*C enumerator*), 1480
gapGenericEventType_t.gCreateConnectionCanceled_c (*C enumerator*), 1479
gapGenericEventType_t.gDecisionInstructionsSetupComplete_c (*C enumerator*), 1483
gapGenericEventType_t.gDeInitializationComplete_c (*C enumerator*), 1483
gapGenericEventType_t.gDeviceAddedToFilterAcceptList_c (*C enumerator*), 1479
gapGenericEventType_t.gDeviceAddedToMonAdvList_c (*C enumerator*), 1484
gapGenericEventType_t.gDeviceRemovedFromFilterAcceptList_c (*C enumerator*), 1479
gapGenericEventType_t.gDeviceRemovedFromMonAdvList_c (*C enumerator*), 1484
gapGenericEventType_t.gExtAdvertisingDataSetupComplete_c (*C enumerator*), 1480
gapGenericEventType_t.gExtAdvertisingDecisionDataSetupComplete_c (*C enumerator*), 1483
gapGenericEventType_t.gExtAdvertisingParametersSetupComplete_c (*C enumerator*), 1480
gapGenericEventType_t.gExtAdvertisingSetRemoveComplete_c (*C enumerator*), 1480
gapGenericEventType_t.gFilterAcceptListCleared_c (*C enumerator*), 1479
gapGenericEventType_t.gFilterAcceptListSizeRead_c (*C enumerator*), 1479
gapGenericEventType_t.gGetConnParamsComplete_c (*C enumerator*), 1480
gapGenericEventType_t.gHandoverAnchorMonitorEvent_c (*C enumerator*), 1483
gapGenericEventType_t.gHandoverAnchorMonitorPacketContinueEvent_c (*C enumerator*), 1483
gapGenericEventType_t.gHandoverAnchorMonitorPacketEvent_c (*C enumerator*), 1483
gapGenericEventType_t.gHandoverAnchorNotificationStateChanged_c (*C enumerator*), 1482
gapGenericEventType_t.gHandoverAnchorSearchStarted_c (*C enumerator*), 1482
gapGenericEventType_t.gHandoverAnchorSearchStopped_c (*C enumerator*), 1482
gapGenericEventType_t.gHandoverApplyConnectionUpdateProcedureComplete_c (*C enumerator*), 1484
gapGenericEventType_t.gHandoverConnectionUpdateProcedureEvent_c (*C enumerator*), 1484
gapGenericEventType_t.gHandoverConnParamUpdateEvent_c (*C enumerator*), 1483
gapGenericEventType_t.gHandoverFreeComplete_c (*C enumerator*), 1483
gapGenericEventType_t.gHandoverGetComplete_c (*C enumerator*), 1482
gapGenericEventType_t.gHandoverGetCsLlContextComplete_c (*C enumerator*), 1482
gapGenericEventType_t.gHandoverGetTime_c (*C enumerator*), 1482
gapGenericEventType_t.gHandoverLlPendingData_c (*C enumerator*), 1483
gapGenericEventType_t.gHandoverResumeTransmitComplete_c (*C enumerator*), 1482
gapGenericEventType_t.gHandoverSetComplete_c (*C enumerator*), 1482
gapGenericEventType_t.gHandoverSetCsLlContextComplete_c (*C enumerator*), 1482
gapGenericEventType_t.gHandoverSuspendTransmitComplete_c (*C enumerator*), 1482
gapGenericEventType_t.gHandoverTimeSyncEvent_c (*C enumerator*), 1483

gapGenericEventType_t.gHandoverTimeSyncReceiveComplete_c (*C enumerator*), 1483
gapGenericEventType_t.gHandoverTimeSyncTransmitStateChanged_c (*C enumerator*), 1482
gapGenericEventType_t.gHandoverUpdateConnParamsComplete_c (*C enumerator*), 1483
gapGenericEventType_t.gHostPrivacyStateChanged_c (*C enumerator*), 1480
gapGenericEventType_t.gInitializationComplete_c (*C enumerator*), 1479
gapGenericEventType_t.gInternalError_c (*C enumerator*), 1479
gapGenericEventType_t.gLeChannelOverrideComplete_c (*C enumerator*), 1484
gapGenericEventType_t.gLeChasConfigComplete_c (*C enumerator*), 1485
gapGenericEventType_t.gLeModulationTestComplete_c (*C enumerator*), 1485
gapGenericEventType_t.gLePeriodicAdvUpdateSyncComplete_c (*C enumerator*), 1484
gapGenericEventType_t.gLePhyEvent_c (*C enumerator*), 1480
gapGenericEventType_t.gLeScLocalOobData_c (*C enumerator*), 1480
gapGenericEventType_t.gLeScPublicKeyRegenerated_c (*C enumerator*), 1480
gapGenericEventType_t.gLeSetDataRelatedAddressChangesComplete_c (*C enumerator*), 1484
gapGenericEventType_t.gLeSetDefaultConnectionSubrateParametersSetupComplete_c (*C enumerator*), 1484
gapGenericEventType_t.gLeSetSchedulerPriorityComplete_c (*C enumerator*), 1483
gapGenericEventType_t.gLlSkdReportEvent_c (*C enumerator*), 1483
gapGenericEventType_t.gModifiedSleepClockAccuracy_c (*C enumerator*), 1481
gapGenericEventType_t.gMonAdvEnabled_c (*C enumerator*), 1484
gapGenericEventType_t.gMonAdvListCleared_c (*C enumerator*), 1484
gapGenericEventType_t.gMonAdvListSizeRead_c (*C enumerator*), 1484
gapGenericEventType_t.gPeriodicAdvCreateSyncCancelled_c (*C enumerator*), 1481
gapGenericEventType_t.gPeriodicAdvDataSetupComplete_c (*C enumerator*), 1481
gapGenericEventType_t.gPeriodicAdvertisingStateChanged_c (*C enumerator*), 1481
gapGenericEventType_t.gPeriodicAdvListUpdateComplete_c (*C enumerator*), 1481
gapGenericEventType_t.gPeriodicAdvParamSetupComplete_c (*C enumerator*), 1480
gapGenericEventType_t.gPeriodicAdvRecvEnableComplete_c (*C enumerator*), 1481
gapGenericEventType_t.gPeriodicAdvSetInfoTransferComplete_c (*C enumerator*), 1481
gapGenericEventType_t.gPeriodicAdvSetResponseDataComplete_c (*C enumerator*), 1484
gapGenericEventType_t.gPeriodicAdvSetSubeventDataComplete_c (*C enumerator*), 1484
gapGenericEventType_t.gPeriodicAdvSyncTransferComplete_c (*C enumerator*), 1481
gapGenericEventType_t.gPeriodicAdvSyncTransferFailed_c (*C enumerator*), 1482
gapGenericEventType_t.gPeriodicAdvSyncTransferSucceeded_c (*C enumerator*), 1482
gapGenericEventType_t.gPeriodicSyncSubeventComplete_c (*C enumerator*), 1484
gapGenericEventType_t.gPrivateResolvableAddressVerified_c (*C enumerator*), 1479
gapGenericEventType_t.gPublicAddressRead_c (*C enumerator*), 1479
gapGenericEventType_t.gRandomAddressReady_c (*C enumerator*), 1479
gapGenericEventType_t.gRandomAddressSet_c (*C enumerator*), 1480
gapGenericEventType_t.gRemoteVersionInformationRead_c (*C enumerator*), 1483
gapGenericEventType_t.gSetDefaultPeriodicAdvSyncTransferParamsComplete_c (*C enumerator*), 1482
gapGenericEventType_t.gSetPeriodicAdvSyncTransferParamsComplete_c (*C enumerator*), 1481
gapGenericEventType_t.gTxEntryAvailable_c (*C enumerator*), 1481
gapGenericEventType_t.gTxPowerLevelSetComplete_c (*C enumerator*), 1480
gapGenericEventType_t.gVendorDebugEvent_c (*C enumerator*), 1484
gapGenericEventType_t.gVendorUnitaryTestComplete_c (*C enumerator*), 1484
gapHandleList_t (*C struct*), 1369, 1573
gapHandleList_t.aHandles (*C var*), 1369, 1573
gapHandleList_t.cNumHandles (*C var*), 1369, 1573
gapHandoverConnectedEvent_t (*C struct*), 1398, 1595
gapHandoverConnectedEvent_t.connectionRole (*C var*), 1399, 1595
gapHandoverConnectedEvent_t.peerAddress (*C var*), 1399, 1595
gapHandoverConnectedEvent_t.peerAddressType (*C var*), 1399, 1595
gapHandoverDisconnectedEvent_t (*C struct*), 1399, 1596
gapHandoverDisconnectedEvent_t.status (*C var*), 1399, 1596
gapHostVersion_t (*C type*), 1302
gapHostVersion_tag (*C struct*), 1400, 1596

gapHostVersion_tag.bleHostVerMajor (*C var*), 1597
gapHostVersion_tag.bleHostVerMinor (*C var*), 1597
gapHostVersion_tag.bleHostVerPatch (*C var*), 1597
gapIdentityInformation_t (*C struct*), 1399, 1596
gapIdentityInformation_t.identityAddress (*C var*), 1399, 1596
gapIdentityInformation_t.irk (*C var*), 1399, 1596
gapIdentityInformation_t.privacyMode (*C var*), 1399, 1596
gapInitComplete_t (*C struct*), 1522, 1556
gapInitComplete_t.leExtendedFeatures (*C var*), 1556
gapInitComplete_t.maxAdvDataSize (*C var*), 1556
gapInitComplete_t.numOfSupportedAdvSets (*C var*), 1556
gapInitComplete_t.periodicAdvListSize (*C var*), 1556
gapInitComplete_t.supportedFeatures (*C var*), 1556
gapInternalError_t (*C struct*), 1521, 1555
gapInternalError_t.errorCode (*C var*), 1521, 1555
gapInternalError_t.errorSource (*C var*), 1521, 1555
gapInternalError_t.hciCommandOpcode (*C var*), 1521, 1555
gapInternalErrorSource_t (*C enum*), 1485
gapInternalErrorSource_t.gAddDeviceToFilterAcceptList_c (*C enumerator*), 1485
gapInternalErrorSource_t.gAddDeviceToMonAdvList_c (*C enumerator*), 1491
gapInternalErrorSource_t.gAddNewConnection_c (*C enumerator*), 1485
gapInternalErrorSource_t.gCancelCreateConnection_c (*C enumerator*), 1485
gapInternalErrorSource_t.gCheckPrivateResolvableAddress_c (*C enumerator*), 1485
gapInternalErrorSource_t.gClearFilterAcceptList_c (*C enumerator*), 1485
gapInternalErrorSource_t.gClearMonAdvList_c (*C enumerator*), 1492
gapInternalErrorSource_t.gConnect_c (*C enumerator*), 1486
gapInternalErrorSource_t.gConnectionCteReqEnable_c (*C enumerator*), 1488
gapInternalErrorSource_t.gConnectionCteRspEnable_c (*C enumerator*), 1488
gapInternalErrorSource_t.gConnectionSubrateRequest_c (*C enumerator*), 1492
gapInternalErrorSource_t.gCreateRandomAddress_c (*C enumerator*), 1485
gapInternalErrorSource_t.gDefaultPairingProcedure_c (*C enumerator*), 1486
gapInternalErrorSource_t.gDenyLongTermKey_c (*C enumerator*), 1486
gapInternalErrorSource_t.gDisconnect_c (*C enumerator*), 1486
gapInternalErrorSource_t.gEattConnectionAccept_c (*C enumerator*), 1488
gapInternalErrorSource_t.gEattConnectionRequest_c (*C enumerator*), 1488
gapInternalErrorSource_t.gEattDisconnectRequest_c (*C enumerator*), 1488
gapInternalErrorSource_t.gEattL2caCancelConnection_c (*C enumerator*), 1489
gapInternalErrorSource_t.gEattL2caDisconnectLePsm_c (*C enumerator*), 1489
gapInternalErrorSource_t.gEattL2caEnhancedCancelConnection_c (*C enumerator*), 1489
gapInternalErrorSource_t.gEattL2caEnhancedReconfigureReq_c (*C enumerator*), 1489
gapInternalErrorSource_t.gEattL2caHandleRecvLeCbData_c (*C enumerator*), 1489
gapInternalErrorSource_t.gEattL2caHandleSendLeCbData_c (*C enumerator*), 1489
gapInternalErrorSource_t.gEattL2caSendLeFlowControlCredit_c (*C enumerator*), 1489
gapInternalErrorSource_t.gEattReconfigureRequest_c (*C enumerator*), 1488
gapInternalErrorSource_t.gEattSendCreditsRequest_c (*C enumerator*), 1488
gapInternalErrorSource_t.gEnableControllerPrivacy_c (*C enumerator*), 1486
gapInternalErrorSource_t.gEnableHostPrivacy_c (*C enumerator*), 1486
gapInternalErrorSource_t.gEnableLdmTimer_c (*C enumerator*), 1487
gapInternalErrorSource_t.gEnableMonAdv_c (*C enumerator*), 1492
gapInternalErrorSource_t.gEncryptLink_c (*C enumerator*), 1485
gapInternalErrorSource_t.gEncryptLinkTak_c (*C enumerator*), 1492
gapInternalErrorSource_t.gEnhancedReadTransmitPowerLevel_c (*C enumerator*), 1488
gapInternalErrorSource_t.gEnterPasskey_c (*C enumerator*), 1486
gapInternalErrorSource_t.gExtAdvReportProcess_c (*C enumerator*), 1487
gapInternalErrorSource_t.gGenerateDHKeyV2_c (*C enumerator*), 1488
gapInternalErrorSource_t.gGetConnParams_c (*C enumerator*), 1489
gapInternalErrorSource_t.gHandover_c (*C enumerator*), 1489
gapInternalErrorSource_t.gHandoverAnchorNotifHciTx_c (*C enumerator*), 1490

gapInternalErrorSource_t.gHandoverAnchorNotifLl_c (*C enumerator*), 1491
gapInternalErrorSource_t.gHandoverAnchorSearchStartHciTx_c (*C enumerator*), 1490
gapInternalErrorSource_t.gHandoverAnchorSearchStartLl_c (*C enumerator*), 1491
gapInternalErrorSource_t.gHandoverAnchorSearchStopHciTx_c (*C enumerator*), 1490
gapInternalErrorSource_t.gHandoverAnchorSearchStopLl_c (*C enumerator*), 1491
gapInternalErrorSource_t.gHandoverApplyConnectionUpdateProcedure_c (*C enumerator*), 1491
gapInternalErrorSource_t.gHandoverConnect_c (*C enumerator*), 1489
gapInternalErrorSource_t.gHandoverDisconnect_c (*C enumerator*), 1489
gapInternalErrorSource_t.gHandoverGetCsLlContext_c (*C enumerator*), 1489
gapInternalErrorSource_t.gHandoverGetLlContext_c (*C enumerator*), 1489
gapInternalErrorSource_t.gHandoverGetTimeLl_c (*C enumerator*), 1490
gapInternalErrorSource_t.gHandoverResumeTransmitHciTx_c (*C enumerator*), 1490
gapInternalErrorSource_t.gHandoverResumeTransmitLl_c (*C enumerator*), 1490
gapInternalErrorSource_t.gHandoverSetCsLlContext_c (*C enumerator*), 1490
gapInternalErrorSource_t.gHandoverSetLlPendingData_c (*C enumerator*), 1491
gapInternalErrorSource_t.gHandoverSetSkd_c (*C enumerator*), 1489
gapInternalErrorSource_t.gHandoverSuspendTransmitHciTx_c (*C enumerator*), 1490
gapInternalErrorSource_t.gHandoverSuspendTransmitHostTxInProgress_c (*C enumerator*), 1489
gapInternalErrorSource_t.gHandoverSuspendTransmitLl_c (*C enumerator*), 1490
gapInternalErrorSource_t.gHandoverSuspendTransmitLlProcInProgress_c (*C enumerator*), 1489
gapInternalErrorSource_t.gHandoverTimeSyncRxHciTx_c (*C enumerator*), 1490
gapInternalErrorSource_t.gHandoverTimeSyncRxLl_c (*C enumerator*), 1491
gapInternalErrorSource_t.gHandoverTimeSyncTxHciTx_c (*C enumerator*), 1490
gapInternalErrorSource_t.gHandoverTimeSyncTxLl_c (*C enumerator*), 1491
gapInternalErrorSource_t.gHandoverUpdateConnParams_c (*C enumerator*), 1490
gapInternalErrorSource_t.gHciCommandStatus_c (*C enumerator*), 1485
gapInternalErrorSource_t.gHciDataDiscardedAlloc_c (*C enumerator*), 1489
gapInternalErrorSource_t.gHciDataDiscardedInvalidStateParam_c (*C enumerator*), 1489
gapInternalErrorSource_t.gHciEventReceiveHandler_c (*C enumerator*), 1487
gapInternalErrorSource_t.gHciRecvFragmentOfPacket_c (*C enumerator*), 1489
gapInternalErrorSource_t.gL2capRxPacket_c (*C enumerator*), 1487
gapInternalErrorSource_t.gLeAddDeviceToPeriodicAdvList_c (*C enumerator*), 1487
gapInternalErrorSource_t.gLeChannelOverride_c (*C enumerator*), 1491
gapInternalErrorSource_t.gLeChasConfig_c (*C enumerator*), 1492
gapInternalErrorSource_t.gLeClearPeriodicAdvList_c (*C enumerator*), 1487
gapInternalErrorSource_t.gLeControllerTest_c (*C enumerator*), 1486
gapInternalErrorSource_t.gLeModulationTest_c (*C enumerator*), 1492
gapInternalErrorSource_t.gLePeriodicAdvResponseReport_c (*C enumerator*), 1491
gapInternalErrorSource_t.gLePeriodicAdvSyncEstb_c (*C enumerator*), 1487
gapInternalErrorSource_t.gLePeriodicAdvSyncLost_c (*C enumerator*), 1487
gapInternalErrorSource_t.gLePeriodicAdvUpdateSync_c (*C enumerator*), 1491
gapInternalErrorSource_t.gLeReadMaxAdvDataLen_c (*C enumerator*), 1487
gapInternalErrorSource_t.gLeReadNumOfSupportedAdvSets_c (*C enumerator*), 1487
gapInternalErrorSource_t.gLeReadPeriodicAdvListSize_c (*C enumerator*), 1487
gapInternalErrorSource_t.gLeReadPhy_c (*C enumerator*), 1486
gapInternalErrorSource_t.gLeRemoveDeviceFromPeriodicAdvList_c (*C enumerator*), 1487
gapInternalErrorSource_t.gLeScGetLocalOobData_c (*C enumerator*), 1486
gapInternalErrorSource_t.gLeScRegeneratePublicKey_c (*C enumerator*), 1486
gapInternalErrorSource_t.gLeScSendKeypressNotification_c (*C enumerator*), 1486
gapInternalErrorSource_t.gLeScSetPeerOobData_c (*C enumerator*), 1486
gapInternalErrorSource_t.gLeScValidateNumericValue_c (*C enumerator*), 1486
gapInternalErrorSource_t.gLeSetHostFeature_c (*C enumerator*), 1490
gapInternalErrorSource_t.gLeSetLocalPeripheralLatencyEnable_c (*C enumerator*), 1492
gapInternalErrorSource_t.gLeSetPeriodicAdvParamsV2_c (*C enumerator*), 1491
gapInternalErrorSource_t.gLeSetPeriodicAdvResponseData_c (*C enumerator*), 1491
gapInternalErrorSource_t.gLeSetPeriodicAdvSubeventData_c (*C enumerator*), 1491
gapInternalErrorSource_t.gLeSetPeriodicSyncSubevent_c (*C enumerator*), 1491
gapInternalErrorSource_t.gLeSetPhy_c (*C enumerator*), 1487

gapInternalErrorSource_t.gLeSetResolvablePrivateAddressTimeout_c (*C enumerator*), 1486
gapInternalErrorSource_t.gLeSetSchedulerPriority_c (*C enumerator*), 1490
gapInternalErrorSource_t.gModifySleepClockAccuracy_c (*C enumerator*), 1488
gapInternalErrorSource_t.gPeriodicAdvCancelSync (*C enumerator*), 1487
gapInternalErrorSource_t.gPeriodicAdvCreateSync (*C enumerator*), 1487
gapInternalErrorSource_t.gPeriodicAdvRcvEnable_c (*C enumerator*), 1488
gapInternalErrorSource_t.gPeriodicAdvSetInfoTransfer_c (*C enumerator*), 1488
gapInternalErrorSource_t.gPeriodicAdvSyncTransfer_c (*C enumerator*), 1488
gapInternalErrorSource_t.gPeriodicAdvTerminateSync (*C enumerator*), 1487
gapInternalErrorSource_t.gProvideLongTermKey_c (*C enumerator*), 1486
gapInternalErrorSource_t.gProvideOob_c (*C enumerator*), 1486
gapInternalErrorSource_t.gReadAllLocalSupportedFeatures_c (*C enumerator*), 1491
gapInternalErrorSource_t.gReadAntennaInformation_c (*C enumerator*), 1488
gapInternalErrorSource_t.gReadControllerLocalRPA_c (*C enumerator*), 1487
gapInternalErrorSource_t.gReadDeviceAddress_c (*C enumerator*), 1485
gapInternalErrorSource_t.gReadFilterAcceptListSize_c (*C enumerator*), 1485
gapInternalErrorSource_t.gReadLeBufferSize_c (*C enumerator*), 1485
gapInternalErrorSource_t.gReadLocalSupportedCommands_c (*C enumerator*), 1487
gapInternalErrorSource_t.gReadLocalSupportedFeatures_c (*C enumerator*), 1485
gapInternalErrorSource_t.gReadMonAdvListSize_c (*C enumerator*), 1492
gapInternalErrorSource_t.gReadRadioPower_c (*C enumerator*), 1485
gapInternalErrorSource_t.gReadRemoteTransmitPowerLevel_c (*C enumerator*), 1488
gapInternalErrorSource_t.gReadRemoteVersionInfo_c (*C enumerator*), 1490
gapInternalErrorSource_t.gReadSuggestedDefaultDataLength_c (*C enumerator*), 1486
gapInternalErrorSource_t.gRemoveAdvertisingSet_c (*C enumerator*), 1487
gapInternalErrorSource_t.gRemoveDeviceFromFilterAcceptList_c (*C enumerator*), 1485
gapInternalErrorSource_t.gRemoveDeviceFromMonAdvList_c (*C enumerator*), 1492
gapInternalErrorSource_t.gResetController_c (*C enumerator*), 1485
gapInternalErrorSource_t.gSaveKeys_c (*C enumerator*), 1487
gapInternalErrorSource_t.gSendPeripheralSecurityRequest_c (*C enumerator*), 1486
gapInternalErrorSource_t.gSendSmpKeys_c (*C enumerator*), 1486
gapInternalErrorSource_t.gSetChannelMap_c (*C enumerator*), 1487
gapInternalErrorSource_t.gSetConnectionCteReceiveParams_c (*C enumerator*), 1488
gapInternalErrorSource_t.gSetConnectionCteTransmitParams_c (*C enumerator*), 1488
gapInternalErrorSource_t.gSetConnectionlessCteTransmitEnable_c (*C enumerator*), 1487
gapInternalErrorSource_t.gSetConnectionlessCteTransmitParams_c (*C enumerator*), 1487
gapInternalErrorSource_t.gSetConnectionlessIqSamplingEnable_c (*C enumerator*), 1488
gapInternalErrorSource_t.gSetDataRelatedAddressChanges_c (*C enumerator*), 1491
gapInternalErrorSource_t.gSetDecisionInstructions_c (*C enumerator*), 1490
gapInternalErrorSource_t.gSetDefaultConnectionSubrateParameters_c (*C enumerator*), 1492
gapInternalErrorSource_t.gSetDefaultPeriodicAdvSyncTransferParams_c (*C enumerator*), 1488
gapInternalErrorSource_t.gSetEventMask_c (*C enumerator*), 1485
gapInternalErrorSource_t.gSetExpmSupportedFeatures_c (*C enumerator*), 1490
gapInternalErrorSource_t.gSetExtAdvDecisionData_c (*C enumerator*), 1490
gapInternalErrorSource_t.gSetLeEventMask_c (*C enumerator*), 1485
gapInternalErrorSource_t.gSetPathLossReportingEnable_c (*C enumerator*), 1488
gapInternalErrorSource_t.gSetPathLossReportingParams_c (*C enumerator*), 1488
gapInternalErrorSource_t.gSetPeriodicAdvSyncTransferParams_c (*C enumerator*), 1488
gapInternalErrorSource_t.gSetRandomAddress_c (*C enumerator*), 1485
gapInternalErrorSource_t.gSetTransmitPowerReportingEnable_c (*C enumerator*), 1488
gapInternalErrorSource_t.gTerminatePairing_c (*C enumerator*), 1486
gapInternalErrorSource_t.gUpdateLeDataLength_c (*C enumerator*), 1486
gapInternalErrorSource_t.gVendorEnableExpmFeatures_c (*C enumerator*), 1492
gapInternalErrorSource_t.gVendorEnableLePowerControl_c (*C enumerator*), 1492
gapInternalErrorSource_t.gVendorUnitaryTest_c (*C enumerator*), 1491
gapInternalErrorSource_t.gWriteSuggestedDefaultDataLength_c (*C enumerator*), 1486
gapIoCapabilities_t (*C type*), 1300
gapKeyExchangeRequestEvent_t (*C struct*), 1393, 1590

gapKeyExchangeRequestEvent_t.requestedKeys (*C var*), 1393, 1591
gapKeyExchangeRequestEvent_t.requestedLtkSize (*C var*), 1393, 1591
gapKeypressNotification_t (*C type*), 1301
gapKeypressNotification_tag (*C enum*), 1284
gapKeypressNotification_tag (*C type*), 1301
gapKeypressNotification_tag.gKnPasskeyCleared_c (*C enumerator*), 1284
gapKeypressNotification_tag.gKnPasskeyDigitErased_c (*C enumerator*), 1284
gapKeypressNotification_tag.gKnPasskeyDigitStarted_c (*C enumerator*), 1284
gapKeypressNotification_tag.gKnPasskeyEntryCompleted_c (*C enumerator*), 1284
gapKeypressNotification_tag.gKnPasskeyEntryStarted_c (*C enumerator*), 1284
gapKeysReceivedEvent_t (*C struct*), 1393, 1591
gapKeysReceivedEvent_t.pKeys (*C var*), 1393, 1591
gapLeAllPhyFlags_t (*C enum*), 1492
gapLeAllPhyFlags_t.gLeRxPhyNoPreference_c (*C enumerator*), 1493
gapLeAllPhyFlags_t.gLeTxPhyNoPreference_c (*C enumerator*), 1493
gapLEGenerateDhKeyCompleteEvent_t (*C struct*), 1533, 1565
gapLEGenerateDhKeyCompleteEvent_t.aDHKey (*C var*), 1565
gapLeModulationTestMode_t (*C enum*), 1472
gapLeModulationTestMode_t.gLeModulationTestDisable_c (*C enumerator*), 1472
gapLeModulationTestMode_t.gLeModulationTestModulatedOnes_c (*C enumerator*), 1472
gapLeModulationTestMode_t.gLeModulationTestModulatedPRBS_c (*C enumerator*), 1472
gapLeModulationTestMode_t.gLeModulationTestModulatedZeros_c (*C enumerator*), 1472
gapLeModulationTestMode_t.gLeModulationTestRssiMeasurement_c (*C enumerator*), 1472
gapLeModulationTestMode_t.gLeModulationTestUnmodulated_c (*C enumerator*), 1472
gapLePhyFlags_t (*C type*), 1500
gapLePhyMode_t (*C type*), 1500
gapLePhyMode_tag (*C enum*), 1493
gapLePhyMode_tag.gLePhy1M_c (*C enumerator*), 1493
gapLePhyMode_tag.gLePhy2M_c (*C enumerator*), 1493
gapLePhyMode_tag.gLePhyCoded_c (*C enumerator*), 1493
gapLePhyOptionsFlags_t (*C enum*), 1493
gapLePhyOptionsFlags_t.gLeCodingNoPreference_c (*C enumerator*), 1493
gapLePhyOptionsFlags_t.gLeCodingS2_c (*C enumerator*), 1493
gapLePhyOptionsFlags_t.gLeCodingS2Req_c (*C enumerator*), 1493
gapLePhyOptionsFlags_t.gLeCodingS8_c (*C enumerator*), 1493
gapLePhyOptionsFlags_t.gLeCodingS8Req_c (*C enumerator*), 1493
gapLeScOobData_t (*C struct*), 1520, 1554
gapLeScOobData_t.confirmValue (*C var*), 1521, 1555
gapLeScOobData_t.randomValue (*C var*), 1520, 1555
gapLlSkdReport_t (*C type*), 1501
gapLlSkdReport_tag (*C struct*), 1533, 1566
gapLlSkdReport_tag.aSKD (*C var*), 1534, 1566
gapLlSkdReport_tag.deviceId (*C var*), 1534, 1566
gapLongTermKeyRequestEvent_t (*C struct*), 1394, 1591
gapLongTermKeyRequestEvent_t.aRand (*C var*), 1394, 1591
gapLongTermKeyRequestEvent_t.ediv (*C var*), 1394, 1591
gapLongTermKeyRequestEvent_t.randSize (*C var*), 1394, 1591
gapMonAdvReportReport_t (*C struct*), 1391, 1589
gapMonAdvReportReport_t.condition (*C var*), 1392, 1590
gapMonAdvReportReport_t.peerAddress (*C var*), 1392, 1590
gapMonAdvReportReport_t.peerAddressType (*C var*), 1392, 1590
gApp2Host_TaskQueue (*C var*), 1502
gapPairingCompleteEvent_t (*C struct*), 1393, 1591
gapPairingCompleteEvent_t.pairingCompleteData (*C union*), 1402, 1599
gapPairingCompleteEvent_t.pairingCompleteData (*C var*), 1394, 1591
gapPairingCompleteEvent_t.pairingCompleteData.failReason (*C var*), 1402, 1599
gapPairingCompleteEvent_t.pairingCompleteData.withBonding (*C var*), 1402, 1599
gapPairingCompleteEvent_t.pairingSuccessful (*C var*), 1394, 1591

gapPairingParameters_t (*C struct*), 1370, 1574
gapPairingParameters_t.centralKeys (*C var*), 1370, 1574
gapPairingParameters_t.leSecureConnectionSupported (*C var*), 1370, 1574
gapPairingParameters_t.localIoCapabilities (*C var*), 1370, 1574
gapPairingParameters_t.maxEncryptionKeySize (*C var*), 1370, 1574
gapPairingParameters_t.oobAvailable (*C var*), 1370, 1574
gapPairingParameters_t.peripheralKeys (*C var*), 1370, 1574
gapPairingParameters_t.securityModeAndLevel (*C var*), 1370, 1574
gapPairingParameters_t.useKeypressNotifications (*C var*), 1370, 1574
gapPairingParameters_t.withBonding (*C var*), 1370, 1574
gapPathLossReportingParams_t (*C struct*), 1382, 1582
gapPathLossReportingParams_t.highHysteresis (*C var*), 1382, 1582
gapPathLossReportingParams_t.highThreshold (*C var*), 1382, 1582
gapPathLossReportingParams_t.lowHysteresis (*C var*), 1382, 1582
gapPathLossReportingParams_t.lowThreshold (*C var*), 1382, 1582
gapPathLossReportingParams_t.minTimeSpent (*C var*), 1382, 1582
gapPathLossThresholdEvent_t (*C struct*), 1397, 1593
gapPathLossThresholdEvent_t.currentPathLoss (*C var*), 1397, 1593
gapPathLossThresholdEvent_t.zoneEntered (*C var*), 1397, 1593
gapPerAdvResponse_t (*C struct*), 1386, 1585
gapPerAdvResponse_t.advHandle (*C var*), 1585
gapPerAdvResponse_t.cteType (*C var*), 1585
gapPerAdvResponse_t.dataLength (*C var*), 1586
gapPerAdvResponse_t.pData (*C var*), 1586
gapPerAdvResponse_t.responseSlot (*C var*), 1586
gapPerAdvResponse_t.rssi (*C var*), 1585
gapPerAdvResponse_t.subevent (*C var*), 1585
gapPerAdvResponse_t.txPower (*C var*), 1585
gapPerAdvSubeventDataRequest_t (*C struct*), 1386, 1585
gapPerAdvSubeventDataRequest_t.handle (*C var*), 1386, 1585
gapPerAdvSubeventDataRequest_t.subeventDataCount (*C var*), 1386, 1585
gapPerAdvSubeventDataRequest_t.subeventStart (*C var*), 1386, 1585
gapPeriodicAdvertisingResponseData_t (*C struct*), 1384, 1584
gapPeriodicAdvertisingResponseData_t.pResponseData (*C var*), 1385, 1584
gapPeriodicAdvertisingResponseData_t.requestEvent (*C var*), 1384, 1584
gapPeriodicAdvertisingResponseData_t.requestSubevent (*C var*), 1384, 1584
gapPeriodicAdvertisingResponseData_t.responseSlot (*C var*), 1385, 1584
gapPeriodicAdvertisingResponseData_t.responseSubevent (*C var*), 1384, 1584
gapPeriodicAdvertisingSubeventData_t (*C struct*), 1384, 1584
gapPeriodicAdvertisingSubeventData_t.aSubeventDataStructures (*C var*), 1384, 1584
gapPeriodicAdvertisingSubeventData_t.cNumSubevents (*C var*), 1384, 1584
gapPeriodicAdvListOperation_t (*C enum*), 1298
gapPeriodicAdvListOperation_t.gAddDevice_c (*C enumerator*), 1298
gapPeriodicAdvListOperation_t.gRemoveAllDevices_c (*C enumerator*), 1298
gapPeriodicAdvListOperation_t.gRemoveDevice_c (*C enumerator*), 1298
gapPeriodicAdvParameters_t (*C type*), 1301
gapPeriodicAdvParameters_tag (*C struct*), 1373, 1576
gapPeriodicAdvParameters_tag.addTxPowerInAdv (*C var*), 1373, 1576
gapPeriodicAdvParameters_tag.handle (*C var*), 1373, 1576
gapPeriodicAdvParameters_tag.maxInterval (*C var*), 1373, 1576
gapPeriodicAdvParameters_tag.minInterval (*C var*), 1373, 1576
gapPeriodicAdvParameters_tag.numResponseSlots (*C var*), 1374, 1576
gapPeriodicAdvParameters_tag.numSubevents (*C var*), 1373, 1576
gapPeriodicAdvParameters_tag.responseSlotDelay (*C var*), 1373, 1576
gapPeriodicAdvParameters_tag.responseSlotSpacing (*C var*), 1374, 1576
gapPeriodicAdvParameters_tag.subeventInterval (*C var*), 1373, 1576
gapPeriodicAdvSetInfoTransfer_t (*C type*), 1301
gapPeriodicAdvSetInfoTransfer_tag (*C struct*), 1374, 1577

gapPeriodicAdvSetInfoTransfer_tag.advHandle (*C var*), 1374, 1577
gapPeriodicAdvSetInfoTransfer_tag.deviceId (*C var*), 1374, 1577
gapPeriodicAdvSetInfoTransfer_tag.serviceData (*C var*), 1374, 1577
gapPeriodicAdvSyncMode_t (*C type*), 1301
gapPeriodicAdvSyncMode_tag (*C enum*), 1285
gapPeriodicAdvSyncMode_tag.gapPeriodicSyncNoReports_c (*C enumerator*), 1285
gapPeriodicAdvSyncMode_tag.gapPeriodicSyncNoSyncMode_c (*C enumerator*), 1285
gapPeriodicAdvSyncMode_tag.gapPeriodicSyncReportsEnabled_c (*C enumerator*), 1285
gapPeriodicAdvSyncMode_tag.gapPeriodicSyncReportsEnabledWithDF_c (*C enumerator*), 1285
gapPeriodicAdvSyncReq_t (*C type*), 1301
gapPeriodicAdvSyncReq_tag (*C struct*), 1375, 1578
gapPeriodicAdvSyncReq_tag.cteType (*C var*), 1376, 1578
gapPeriodicAdvSyncReq_tag.options (*C var*), 1376, 1578
gapPeriodicAdvSyncReq_tag.peerAddress (*C var*), 1376, 1578
gapPeriodicAdvSyncReq_tag.peerAddressType (*C var*), 1376, 1578
gapPeriodicAdvSyncReq_tag.SID (*C var*), 1376, 1578
gapPeriodicAdvSyncReq_tag.skipCount (*C var*), 1376, 1578
gapPeriodicAdvSyncReq_tag.timeout (*C var*), 1376, 1578
gapPeriodicAdvSyncTransfer_t (*C type*), 1301
gapPeriodicAdvSyncTransfer_tag (*C struct*), 1374, 1576
gapPeriodicAdvSyncTransfer_tag.deviceId (*C var*), 1374, 1576
gapPeriodicAdvSyncTransfer_tag.serviceData (*C var*), 1374, 1576
gapPeriodicAdvSyncTransfer_tag.syncHandle (*C var*), 1374, 1576
gapPeriodicScannedDevice_t (*C struct*), 1388, 1587
gapPeriodicScannedDevice_t.cteType (*C var*), 1389, 1587
gapPeriodicScannedDevice_t.dataLength (*C var*), 1389, 1587
gapPeriodicScannedDevice_t.pData (*C var*), 1389, 1588
gapPeriodicScannedDevice_t.rssi (*C var*), 1389, 1587
gapPeriodicScannedDevice_t.syncHandle (*C var*), 1388, 1587
gapPeriodicScannedDevice_t.txPower (*C var*), 1389, 1587
gapPeriodicScannedDeviceV2_t (*C struct*), 1389, 1588
gapPeriodicScannedDeviceV2_t.cteType (*C var*), 1389, 1588
gapPeriodicScannedDeviceV2_t.dataLength (*C var*), 1389, 1588
gapPeriodicScannedDeviceV2_t.pData (*C var*), 1389, 1588
gapPeriodicScannedDeviceV2_t.periodicEventCounter (*C var*), 1389, 1588
gapPeriodicScannedDeviceV2_t.rssi (*C var*), 1389, 1588
gapPeriodicScannedDeviceV2_t.subevent (*C var*), 1389, 1588
gapPeriodicScannedDeviceV2_t.syncHandle (*C var*), 1389, 1588
gapPeriodicScannedDeviceV2_t.txPower (*C var*), 1389, 1588
gapPeriodicSyncSubeventParameters_t (*C struct*), 1385, 1584
gapPeriodicSyncSubeventParameters_t.aSubevents (*C var*), 1385, 1584
gapPeriodicSyncSubeventParameters_t.numSubevents (*C var*), 1385, 1584
gapPeriodicSyncSubeventParameters_t.perAdvProperties (*C var*), 1385, 1584
gapPeripheralSecurityRequestParameters_t (*C struct*), 1370, 1574
gapPeripheralSecurityRequestParameters_t.authenticationRequired (*C var*), 1371, 1574
gapPeripheralSecurityRequestParameters_t.bondAfterPairing (*C var*), 1371, 1574
gapPhyEvent_t (*C struct*), 1521, 1555
gapPhyEvent_t.deviceId (*C var*), 1555
gapPhyEvent_t.phyEventType (*C var*), 1555
gapPhyEvent_t.rxPhy (*C var*), 1555
gapPhyEvent_t.txPhy (*C var*), 1555
gapPhyEventType_t (*C enum*), 1493
gapPhyEventType_t.gPhyRead_c (*C enumerator*), 1493
gapPhyEventType_t.gPhySetDefaultComplete_c (*C enumerator*), 1493
gapPhyEventType_t.gPhyUpdateComplete_c (*C enumerator*), 1494
gapPrivateKeyType_t (*C enum*), 1286
gapPrivateKeyType_t.gUseDebugKey_c (*C enumerator*), 1286
gapPrivateKeyType_t.gUseGeneratedKey_c (*C enumerator*), 1286

gapRadioPowerLevelReadType_t (*C enum*), 1290
gapRadioPowerLevelReadType_t.gRssi_c (*C enumerator*), 1291
gapRadioPowerLevelReadType_t.gTxPowerCurrentLevelInConnection_c (*C enumerator*), 1291
gapRadioPowerLevelReadType_t.gTxPowerLevelForAdvertising_c (*C enumerator*), 1291
gapRadioPowerLevelReadType_t.gTxPowerMaximumLevelInConnection_c (*C enumerator*), 1291
gapRemoteVersionInfoRead_t (*C type*), 1501
gapRemoteVersionInfoRead_tag (*C struct*), 1533, 1565
gapRemoteVersionInfoRead_tag.deviceId (*C var*), 1566
gapRemoteVersionInfoRead_tag.manufacturerName (*C var*), 1566
gapRemoteVersionInfoRead_tag.status (*C var*), 1566
gapRemoteVersionInfoRead_tag.subversion (*C var*), 1566
gapRemoteVersionInfoRead_tag.version (*C var*), 1566
gapRole_t (*C enum*), 1283
gapRole_t.gGapBroadcaster_c (*C enumerator*), 1284
gapRole_t.gGapCentral_c (*C enumerator*), 1283
gapRole_t.gGapObserver_c (*C enumerator*), 1283
gapRole_t.gGapPeripheral_c (*C enumerator*), 1283
gapScanMode_t (*C enum*), 1284
gapScanMode_t.gAutoConnect_c (*C enumerator*), 1284
gapScanMode_t.gDefaultScan_c (*C enumerator*), 1284
gapScanMode_t.gGeneralDiscovery_c (*C enumerator*), 1284
gapScanMode_t.gLimitedDiscovery_c (*C enumerator*), 1284
gapScannedDevice_t (*C struct*), 1386, 1586
gapScannedDevice_t.aAddress (*C var*), 1387, 1586
gapScannedDevice_t.addressType (*C var*), 1387, 1586
gapScannedDevice_t.advertisingAddressResolved (*C var*), 1387, 1586
gapScannedDevice_t.advEventType (*C var*), 1387, 1586
gapScannedDevice_t.data (*C var*), 1387, 1586
gapScannedDevice_t.dataLength (*C var*), 1387, 1586
gapScannedDevice_t.directRpa (*C var*), 1387, 1586
gapScannedDevice_t.directRpaUsed (*C var*), 1387, 1586
gapScannedDevice_t.rssi (*C var*), 1387, 1586
gapScanningCallback_t (*C type*), 1302
gapScanningEvent_t (*C struct*), 1392, 1590
gapScanningEvent_t.eventData (*C union*), 1401, 1598
gapScanningEvent_t.eventData (*C var*), 1392, 1590
gapScanningEvent_t.eventData.extScannedDevice (*C var*), 1401, 1598
gapScanningEvent_t.eventData.failReason (*C var*), 1401, 1598
gapScanningEvent_t.eventData.iqReport (*C var*), 1401, 1599
gapScanningEvent_t.eventData.monAdvReport (*C var*), 1402, 1599
gapScanningEvent_t.eventData.periodicScannedDevice (*C var*), 1401, 1599
gapScanningEvent_t.eventData.periodicScannedDeviceV2 (*C var*), 1401, 1599
gapScanningEvent_t.eventData.scannedDevice (*C var*), 1401, 1598
gapScanningEvent_t.eventData.syncEstb (*C var*), 1401, 1599
gapScanningEvent_t.eventData.syncLost (*C var*), 1401, 1599
gapScanningEvent_t.eventType (*C var*), 1392, 1590
gapScanningEventType_t (*C enum*), 1292
gapScanningEventType_t.gConnectionlessIqReportReceived_c (*C enumerator*), 1293
gapScanningEventType_t.gDeviceScanned_c (*C enumerator*), 1293
gapScanningEventType_t.gExtDeviceScanned_c (*C enumerator*), 1293
gapScanningEventType_t.gMonAdvReportEventReceived_c (*C enumerator*), 1293
gapScanningEventType_t.gPeriodicAdvSyncEstablished_c (*C enumerator*), 1293
gapScanningEventType_t.gPeriodicAdvSyncLost_c (*C enumerator*), 1293
gapScanningEventType_t.gPeriodicAdvSyncTerminated_c (*C enumerator*), 1293
gapScanningEventType_t.gPeriodicDeviceScanned_c (*C enumerator*), 1293
gapScanningEventType_t.gPeriodicDeviceScannedV2_c (*C enumerator*), 1293
gapScanningEventType_t.gScanCommandFailed_c (*C enumerator*), 1293
gapScanningEventType_t.gScanStateChanged_c (*C enumerator*), 1292

gapScanningParameters_t (*C struct*), 1375, 1577
gapScanningParameters_t.filterPolicy (*C var*), 1375, 1577
gapScanningParameters_t.interval (*C var*), 1375, 1577
gapScanningParameters_t.ownAddressType (*C var*), 1375, 1577
gapScanningParameters_t.scanningPHYs (*C var*), 1375, 1577
gapScanningParameters_t.type (*C var*), 1375, 1577
gapScanningParameters_t.window (*C var*), 1375, 1577
gapScanResponseData_t (*C type*), 1302
gapSecurityLevel_t (*C type*), 1301
gapSecurityMode_t (*C type*), 1300
gapSecurityModeAndLevel_t (*C type*), 1301
gapSecurityRequirements_t (*C struct*), 1368, 1573
gapSecurityRequirements_t.authorization (*C var*), 1368, 1573
gapSecurityRequirements_t.minimumEncryptionKeySize (*C var*), 1368, 1573
gapSecurityRequirements_t.securityModeLevel (*C var*), 1368, 1573
gapServiceSecurityRequirements_t (*C struct*), 1369, 1573
gapServiceSecurityRequirements_t.requirements (*C var*), 1369, 1573
gapServiceSecurityRequirements_t.serviceHandle (*C var*), 1369, 1573
gapSetPeriodicAdvSyncTransferParams_t (*C type*), 1301
gapSetPeriodicAdvSyncTransferParams_tag (*C struct*), 1374, 1577
gapSetPeriodicAdvSyncTransferParams_tag.CTEType (*C var*), 1375, 1577
gapSetPeriodicAdvSyncTransferParams_tag.deviceId (*C var*), 1374, 1577
gapSetPeriodicAdvSyncTransferParams_tag.mode (*C var*), 1374, 1577
gapSetPeriodicAdvSyncTransferParams_tag.skip (*C var*), 1375, 1577
gapSetPeriodicAdvSyncTransferParams_tag.syncTimeout (*C var*), 1375, 1577
gapSleepClockAccuracy_t (*C enum*), 1292
gapSleepClockAccuracy_t.gSwitchToLessAccurateClock_c (*C enumerator*), 1292
gapSleepClockAccuracy_t.gSwitchToMoreAccurateClock_c (*C enumerator*), 1292
gapSmpKeyFlags_t (*C type*), 1300
gapSmpKeys_t (*C struct*), 1368, 1572
gapSmpKeys_t.aAddress (*C var*), 1368, 1573
gapSmpKeys_t.aCsrk (*C var*), 1368, 1572
gapSmpKeys_t.addressType (*C var*), 1368, 1573
gapSmpKeys_t.aIrk (*C var*), 1368, 1572
gapSmpKeys_t.aLtk (*C var*), 1368, 1572
gapSmpKeys_t.aRand (*C var*), 1368, 1572
gapSmpKeys_t.cLtkSize (*C var*), 1368, 1572
gapSmpKeys_t.cRandSize (*C var*), 1368, 1572
gapSmpKeys_t.ediv (*C var*), 1368, 1573
gapSubeventDataStructure_t (*C struct*), 1384, 1583
gapSubeventDataStructure_t.pAdvertisingData (*C var*), 1384, 1584
gapSubeventDataStructure_t.responseSlotCount (*C var*), 1384, 1584
gapSubeventDataStructure_t.responseSlotStart (*C var*), 1384, 1583
gapSubeventDataStructure_t.subevent (*C var*), 1384, 1583
gapSubrateChangeEvent_t (*C struct*), 1397, 1594
gapSubrateChangeEvent_t.continuationNumber (*C var*), 1398, 1594
gapSubrateChangeEvent_t.peripheralLatency (*C var*), 1398, 1594
gapSubrateChangeEvent_t.status (*C var*), 1594
gapSubrateChangeEvent_t.subrateFactor (*C var*), 1398, 1594
gapSubrateChangeEvent_t.supervisionTimeout (*C var*), 1398, 1594
gapSyncEstbEventData_t (*C struct*), 1390, 1588
gapSyncEstbEventData_t.advertiserClockAccuracy (*C var*), 1390, 1588
gapSyncEstbEventData_t.numSubevents (*C var*), 1390, 1588
gapSyncEstbEventData_t.peerAddress (*C var*), 1390, 1588
gapSyncEstbEventData_t.peerAddressType (*C var*), 1390, 1588
gapSyncEstbEventData_t.periodicAdvInterval (*C var*), 1390, 1588
gapSyncEstbEventData_t.PHY (*C var*), 1390, 1588
gapSyncEstbEventData_t.responseSlotDelay (*C var*), 1390, 1589

gapSyncEstbEventData_t.responseSlotSpacing (*C var*), 1390, 1589
gapSyncEstbEventData_t.SID (*C var*), 1390, 1588
gapSyncEstbEventData_t.status (*C var*), 1390, 1588
gapSyncEstbEventData_t.subeventInterval (*C var*), 1390, 1589
gapSyncEstbEventData_t.syncHandle (*C var*), 1390, 1588
gapSyncLostEventData_t (*C struct*), 1390, 1589
gapSyncLostEventData_t.syncHandle (*C var*), 1391, 1589
gapSyncTransferReceivedEventData_t (*C type*), 1500
gapSyncTransferReceivedEventData_tag (*C struct*), 1523, 1558
gapSyncTransferReceivedEventData_tag.advAddress (*C var*), 1523, 1558
gapSyncTransferReceivedEventData_tag.advAddressType (*C var*), 1523, 1558
gapSyncTransferReceivedEventData_tag.advClockAccuracy (*C var*), 1524, 1558
gapSyncTransferReceivedEventData_tag.advPhy (*C var*), 1523, 1558
gapSyncTransferReceivedEventData_tag.advSID (*C var*), 1523, 1558
gapSyncTransferReceivedEventData_tag.deviceId (*C var*), 1523, 1558
gapSyncTransferReceivedEventData_tag.numSubevents (*C var*), 1524, 1558
gapSyncTransferReceivedEventData_tag.periodicAdvInt (*C var*), 1524, 1558
gapSyncTransferReceivedEventData_tag.responseSlotDelay (*C var*), 1524, 1558
gapSyncTransferReceivedEventData_tag.responseSlotSpacing (*C var*), 1524, 1558
gapSyncTransferReceivedEventData_tag.serviceData (*C var*), 1523, 1558
gapSyncTransferReceivedEventData_tag.status (*C var*), 1558
gapSyncTransferReceivedEventData_tag.subeventInterval (*C var*), 1524, 1558
gapSyncTransferReceivedEventData_tag.syncHandle (*C var*), 1523, 1558
gapTransmitPowerInfo_t (*C struct*), 1398, 1594
gapTransmitPowerInfo_t.currTxPowerLevel (*C var*), 1398, 1594
gapTransmitPowerInfo_t.maxTxPowerLevel (*C var*), 1398, 1594
gapTransmitPowerInfo_t.phy (*C var*), 1398, 1594
gapTransmitPowerReporting_t (*C struct*), 1397, 1593
gapTransmitPowerReporting_t.delta (*C var*), 1397, 1594
gapTransmitPowerReporting_t.flags (*C var*), 1397, 1594
gapTransmitPowerReporting_t.phy (*C var*), 1397, 1594
gapTransmitPowerReporting_t.reason (*C var*), 1397, 1594
gapTransmitPowerReporting_t.txPowerLevel (*C var*), 1397, 1594
Gatt_GetMtu (*C function*), 1408
Gatt_Init (*C function*), 1408
gattAttribute_t (*C struct*), 1409, 1601
gattAttribute_t.handle (*C var*), 1409, 1601
gattAttribute_t.maxValueLength (*C var*), 1409, 1601
gattAttribute_t.paValue (*C var*), 1409, 1601
gattAttribute_t.uuid (*C var*), 1409, 1601
gattAttribute_t.uuidType (*C var*), 1409, 1601
gattAttribute_t.valueLength (*C var*), 1409, 1601
gattAttributePermissionsBitFields_t (*C type*), 1449
gattCachingClientState_c (*C enum*), 1406
gattCachingClientState_c.gGattClientChangeAware_c (*C enumerator*), 1406
gattCachingClientState_c.gGattClientChangeUnaware_c (*C enumerator*), 1406
gattCachingClientState_c.gGattClientStateChangePending_c (*C enumerator*), 1406
gattCccdFlags_t (*C type*), 1408
gattCharacteristic_t (*C struct*), 1409, 1601
gattCharacteristic_t.aDescriptors (*C var*), 1409, 1601
gattCharacteristic_t.cNumDescriptors (*C var*), 1409, 1601
gattCharacteristic_t.properties (*C var*), 1409, 1601
gattCharacteristic_t.value (*C var*), 1409, 1601
gattCharacteristicPropertiesBitFields_t (*C type*), 1449
gattCharacteristicPropertiesBitFields_tag (*C enum*), 1448
gattCharacteristicPropertiesBitFields_tag.gGattCharPropAuthSignedWrites_c (*C enumerator*), 1448
gattCharacteristicPropertiesBitFields_tag.gGattCharPropBroadcast_c (*C enumerator*), 1448

gattCharacteristicPropertiesBitFields_tag.gGattCharPropExtendedProperties_c (*C enumerator*), 1449
gattCharacteristicPropertiesBitFields_tag.gGattCharPropIndicate_c (*C enumerator*), 1448
gattCharacteristicPropertiesBitFields_tag.gGattCharPropNone_c (*C enumerator*), 1448
gattCharacteristicPropertiesBitFields_tag.gGattCharPropNotify_c (*C enumerator*), 1448
gattCharacteristicPropertiesBitFields_tag.gGattCharPropRead_c (*C enumerator*), 1448
gattCharacteristicPropertiesBitFields_tag.gGattCharPropWrite_c (*C enumerator*), 1448
gattCharacteristicPropertiesBitFields_tag.gGattCharPropWriteWithoutRsp_c (*C enumerator*), 1448
GattClient_CharacteristicWriteWithoutResponse (*C macro*), 1434
GattClient_DiscoverAllCharacteristicDescriptors (*C function*), 1419
GattClient_DiscoverAllCharacteristicsOfService (*C function*), 1418
GattClient_DiscoverAllPrimaryServices (*C function*), 1417
GattClient_DiscoverCharacteristicOfServiceByUuid (*C function*), 1419
GattClient_DiscoverPrimaryServicesByUuid (*C function*), 1417
GattClient_EnhancedDiscoverAllCharacteristicDescriptors (*C function*), 1429
GattClient_EnhancedDiscoverAllCharacteristicsOfService (*C function*), 1427
GattClient_EnhancedDiscoverAllPrimaryServices (*C function*), 1426
GattClient_EnhancedDiscoverCharacteristicOfServiceByUuid (*C function*), 1428
GattClient_EnhancedDiscoverPrimaryServicesByUuid (*C function*), 1426
GattClient_EnhancedFindIncludedServices (*C function*), 1427
GattClient_EnhancedReadCharacteristicDescriptor (*C function*), 1432
GattClient_EnhancedReadCharacteristicValue (*C function*), 1429
GattClient_EnhancedReadMultipleCharacteristicValues (*C function*), 1431
GattClient_EnhancedReadMultipleVariableCharacteristicValues (*C function*), 1433
GattClient_EnhancedReadUsingCharacteristicUuid (*C function*), 1430
GattClient_EnhancedWriteCharacteristicDescriptor (*C function*), 1433
GattClient_EnhancedWriteCharacteristicValue (*C function*), 1431
GattClient_ExchangeMtu (*C function*), 1416
GattClient_FindIncludedServices (*C function*), 1418
GattClient_Init (*C function*), 1414
GattClient_ReadCharacteristicDescriptor (*C function*), 1423
GattClient_ReadCharacteristicValue (*C function*), 1420
GattClient_ReadMultipleCharacteristicValues (*C function*), 1421
GattClient_ReadMultipleVariableCharacteristicValues (*C function*), 1424
GattClient_ReadUsingCharacteristicUuid (*C function*), 1421
GattClient_RegisterEnhancedIndicationCallback (*C function*), 1425
GattClient_RegisterEnhancedMultipleValueNotificationCallback (*C function*), 1425
GattClient_RegisterEnhancedNotificationCallback (*C function*), 1424
GattClient_RegisterEnhancedProcedureCallback (*C function*), 1424
GattClient_RegisterIndicationCallback (*C function*), 1415
GattClient_RegisterMultipleValueNotificationCallback (*C function*), 1416
GattClient_RegisterNotificationCallback (*C function*), 1415
GattClient_RegisterProcedureCallback (*C function*), 1415
GattClient_ResetProcedure (*C function*), 1414
GattClient_SimpleCharacteristicWrite (*C macro*), 1434
GattClient_WriteCharacteristicDescriptor (*C function*), 1423
GattClient_WriteCharacteristicValue (*C function*), 1422
gattClientEnhancedIndicationCallback_t (*C type*), 1414
gattClientEnhancedMultipleValueNotificationCallback_t (*C type*), 1414
gattClientEnhancedNotificationCallback_t (*C type*), 1414
gattClientEnhancedProcedureCallback_t (*C type*), 1414
gattClientHashUpdateType_t (*C enum*), 1406
gattClientHashUpdateType_t.gattClientActiveConnectionUpdate (*C enumerator*), 1406
gattClientHashUpdateType_t.gattClientFirstConnection_c (*C enumerator*), 1406
gattClientHashUpdateType_t.gattClientNoChange (*C enumerator*), 1406
gattClientHashUpdateType_t.gattClientReconnectBondedPeer_c (*C enumerator*), 1406
gattClientIndicationCallback_t (*C type*), 1414
gattClientMultipleValueNotificationCallback_t (*C type*), 1414

gattClientNotificationCallback_t (*C type*), 1414
gattClientProcedureCallback_t (*C type*), 1413
gattDatabase (*C var*), 1449
GattDb_ComputeDatabaseHash (*C function*), 1450
GattDb_Deinit (*C function*), 1451
GattDb_FindCccdHandleForCharValueHandle (*C function*), 1453
GattDb_FindCharValueHandleInService (*C function*), 1453
GattDb_FindDescriptorHandleForCharValueHandle (*C function*), 1454
GattDb_FindServiceHandle (*C function*), 1452
GattDb_FindServiceRange (*C function*), 1450
GattDb_GetAttributeValueSize (*C function*), 1450
GattDb_GetIndexOfHandle (*C function*), 1449
GattDb_Init (*C function*), 1450
GattDb_ReadAttribute (*C function*), 1452
GattDb_ServiceStartHandle (*C function*), 1449
GattDb_WriteAttribute (*C function*), 1451
gattDbAccessType_t (*C enum*), 1449
gattDbAccessType_t.gAccessNotify_c (*C enumerator*), 1449
gattDbAccessType_t.gAccessRead_c (*C enumerator*), 1449
gattDbAccessType_t.gAccessWrite_c (*C enumerator*), 1449
gattDbAttribute_t (*C struct*), 1455, 1606
gattDbAttribute_t.handle (*C var*), 1455, 1607
gattDbAttribute_t.maxVariableValueLength (*C var*), 1456, 1607
gattDbAttribute_t.permissions (*C var*), 1455, 1607
gattDbAttribute_t.pValue (*C var*), 1456, 1607
gattDbAttribute_t.uuid (*C var*), 1455, 1607
gattDbAttribute_t.uuidType (*C var*), 1456, 1607
gattDbAttribute_t.valueLength (*C var*), 1456, 1607
gattDbCharPresFormat_t (*C struct*), 1410, 1602
gattDbCharPresFormat_t.description (*C var*), 1411, 1602
gattDbCharPresFormat_t.exponent (*C var*), 1410, 1602
gattDbCharPresFormat_t.format (*C var*), 1410, 1602
gattDbCharPresFormat_t.ns (*C var*), 1410, 1602
gattDbCharPresFormat_t.unitUuid16 (*C var*), 1410, 1602
gAttDefaultMtu_c (*C macro*), 1541
gattHandleRange_t (*C struct*), 1411, 1602
gattHandleRange_t.endHandle (*C var*), 1411, 1602
gattHandleRange_t.startHandle (*C var*), 1411, 1602
gAttMaxMtu_c (*C macro*), 1541
gAttMaxValueLength_c (*C macro*), 1541
gattProcedurePhase_t (*C enum*), 1407
gattProcedurePhase_t.gattProcPhaseInitiated (*C enumerator*), 1408
gattProcedurePhase_t.gattProcPhaseRunning (*C enumerator*), 1408
gattProcedureResult_t (*C enum*), 1413
gattProcedureResult_t.gGattProcError_c (*C enumerator*), 1413
gattProcedureResult_t.gGattProcSuccess_c (*C enumerator*), 1413
gattProcedureType_t (*C enum*), 1406
gattProcedureType_t.gGattProcDiscoverAllCharacteristicDescriptors_c (*C enumerator*), 1407
gattProcedureType_t.gGattProcDiscoverAllCharacteristics_c (*C enumerator*), 1407
gattProcedureType_t.gGattProcDiscoverAllPrimaryServices_c (*C enumerator*), 1407
gattProcedureType_t.gGattProcDiscoverCharacteristicByUuid_c (*C enumerator*), 1407
gattProcedureType_t.gGattProcDiscoverPrimaryServicesByUuid_c (*C enumerator*), 1407
gattProcedureType_t.gGattProcExchangeMtu_c (*C enumerator*), 1406
gattProcedureType_t.gGattProcFindIncludedServices_c (*C enumerator*), 1407
gattProcedureType_t.gGattProcReadCharacteristicDescriptor_c (*C enumerator*), 1407
gattProcedureType_t.gGattProcReadCharacteristicValue_c (*C enumerator*), 1407
gattProcedureType_t.gGattProcReadMultipleCharacteristicValues_c (*C enumerator*), 1407
gattProcedureType_t.gGattProcReadMultipleVarLengthCharValues_c (*C enumerator*), 1407

gattProcedureType_t.gGattProcReadUsingCharacteristicUuid_c (*C enumerator*), 1407
gattProcedureType_t.gGattProcSignalServiceDiscoveryComplete_c (*C enumerator*), 1407
gattProcedureType_t.gGattProcUpdateDatabaseCopy_c (*C enumerator*), 1407
gattProcedureType_t.gGattProcWriteCharacteristicDescriptor_c (*C enumerator*), 1407
gattProcedureType_t.gGattProcWriteCharacteristicValue_c (*C enumerator*), 1407
GattServer_EnhancedSendAttributeReadStatus (*C function*), 1443
GattServer_EnhancedSendAttributeWrittenStatus (*C function*), 1442
GattServer_EnhancedSendIndication (*C function*), 1444
GattServer_EnhancedSendInstantValueIndication (*C function*), 1445
GattServer_EnhancedSendInstantValueNotification (*C function*), 1444
GattServer_EnhancedSendMultipleHandleValueNotification (*C function*), 1445
GattServer_EnhancedSendNotification (*C function*), 1443
GattServer_Init (*C function*), 1436
GattServer_RegisterCallback (*C function*), 1436
GattServer_RegisterEnhancedCallback (*C function*), 1442
GattServer_RegisterHandlesForReadNotifications (*C function*), 1438
GattServer_RegisterHandlesForWriteNotifications (*C function*), 1436
GattServer_RegisterUniqueHandlesForNotifications (*C function*), 1441
GattServer_SendAttributeReadStatus (*C function*), 1439
GattServer_SendAttributeWrittenStatus (*C function*), 1438
GattServer_SendIndication (*C function*), 1440
GattServer_SendInstantValueIndication (*C function*), 1441
GattServer_SendInstantValueNotification (*C function*), 1440
GattServer_SendMultipleHandleValueNotification (*C function*), 1441
GattServer_SendNotification (*C function*), 1440
GattServer_UnregisterHandlesForReadNotifications (*C function*), 1439
GattServer_UnregisterHandlesForWriteNotifications (*C function*), 1437
gattServerAttributeReadEvent_t (*C struct*), 1446, 1605
gattServerAttributeReadEvent_t.handle (*C var*), 1447, 1605
gattServerAttributeWrittenEvent_t (*C struct*), 1446, 1605
gattServerAttributeWrittenEvent_t.aValue (*C var*), 1446, 1605
gattServerAttributeWrittenEvent_t.bearerId (*C var*), 1446, 1605
gattServerAttributeWrittenEvent_t.cValueLength (*C var*), 1446, 1605
gattServerAttributeWrittenEvent_t.handle (*C var*), 1446, 1605
gattServerCallback_t (*C type*), 1436
gattServerCccdWrittenEvent_t (*C struct*), 1446, 1605
gattServerCccdWrittenEvent_t.handle (*C var*), 1446, 1605
gattServerCccdWrittenEvent_t.newCccd (*C var*), 1446, 1605
gattServerEnhancedCallback_t (*C type*), 1436
gattServerEvent_t (*C struct*), 1447, 1606
gattServerEvent_t.eventData (*C union*), 1447, 1606
gattServerEvent_t.eventData (*C var*), 1447, 1606
gattServerEvent_t.eventData.attributeOpCode (*C var*), 1448, 1606
gattServerEvent_t.eventData.attributeReadEvent (*C var*), 1448, 1606
gattServerEvent_t.eventData.attributeWrittenEvent (*C var*), 1448, 1606
gattServerEvent_t.eventData.charCccdWrittenEvent (*C var*), 1448, 1606
gattServerEvent_t.eventData.longCharWrittenEvent (*C var*), 1448, 1606
gattServerEvent_t.eventData.mtuChangedEvent (*C var*), 1448, 1606
gattServerEvent_t.eventData.procedureError (*C var*), 1448, 1606
gattServerEvent_t.eventType (*C var*), 1447, 1606
gattServerEventType_t (*C enum*), 1435
gattServerEventType_t.gEvtAttributeRead_c (*C enumerator*), 1435
gattServerEventType_t.gEvtAttributeWritten_c (*C enumerator*), 1435
gattServerEventType_t.gEvtAttributeWrittenWithoutResponse_c (*C enumerator*), 1435
gattServerEventType_t.gEvtCharacteristicCccdWritten_c (*C enumerator*), 1435
gattServerEventType_t.gEvtError_c (*C enumerator*), 1435
gattServerEventType_t.gEvtHandleValueConfirmation_c (*C enumerator*), 1435
gattServerEventType_t.gEvtInvalidPduReceived_c (*C enumerator*), 1435

gattServerEventType_t.gEvtLongCharacteristicWritten_c (*C enumerator*), 1435
gattServerEventType_t.gEvtMtuChanged_c (*C enumerator*), 1435
gattServerInvalidPdu_t (*C struct*), 1447, 1606
gattServerInvalidPdu_t.attOpCode (*C var*), 1447, 1606
gattServerLongCharacteristicWrittenEvent_t (*C struct*), 1446, 1605
gattServerLongCharacteristicWrittenEvent_t.aValue (*C var*), 1446, 1605
gattServerLongCharacteristicWrittenEvent_t.cValueLength (*C var*), 1446, 1605
gattServerLongCharacteristicWrittenEvent_t.handle (*C var*), 1446, 1605
gattServerMtuChangedEvent_t (*C struct*), 1445, 1604
gattServerMtuChangedEvent_t.newMtu (*C var*), 1446, 1605
gattServerProcedureError_t (*C struct*), 1447, 1605
gattServerProcedureError_t.error (*C var*), 1447, 1606
gattServerProcedureError_t.procedureType (*C var*), 1447, 1606
gattServerProcedureType_t (*C enum*), 1435
gattServerProcedureType_t.gSendAttributeReadStatus_c (*C enumerator*), 1435
gattServerProcedureType_t.gSendAttributeWrittenStatus_c (*C enumerator*), 1435
gattServerProcedureType_t.gSendIndication_c (*C enumerator*), 1436
gattServerProcedureType_t.gSendMultipleValNotification_c (*C enumerator*), 1436
gattServerProcedureType_t.gSendNotification_c (*C enumerator*), 1435
gattService_t (*C type*), 1408
gattService_tag (*C struct*), 1410, 1601
gattService_tag.aCharacteristics (*C var*), 1410, 1602
gattService_tag.aIncludedServices (*C var*), 1410, 1602
gattService_tag.cNumCharacteristics (*C var*), 1410, 1602
gattService_tag.cNumIncludedServices (*C var*), 1410, 1602
gattService_tag.endHandle (*C var*), 1410, 1602
gattService_tag.startHandle (*C var*), 1410, 1602
gattService_tag.uuid (*C var*), 1410, 1602
gattService_tag.uuidType (*C var*), 1410, 1602
gBasicFilteredScan_c (*C macro*), 1507
gBasicUnfilteredScan_c (*C macro*), 1507
gBleAddrTypePublic_c (*C macro*), 1505
gBleAddrTypeRandom_c (*C macro*), 1505
gBleAdvTxPowerNoPreference_c (*C macro*), 1542
gBleBondDataDescriptorSize_c (*C macro*), 1540
gBleBondDataDeviceInfoSize_c (*C macro*), 1540
gBleBondDataDynamicSize_c (*C macro*), 1540
gBleBondDataLegacySize_c (*C macro*), 1540
gBleBondDataSize_c (*C macro*), 1540
gBleBondDataStaticSize_c (*C macro*), 1540
gBleCteMaxTxOctets_c (*C macro*), 1542
gBleCteMaxTxTime_c (*C macro*), 1542
gBleCteMinTxOctets_c (*C macro*), 1542
gBleCteMinTxTime_c (*C macro*), 1542
gBleEattMaxConnectionChannels (*C var*), 1502
gBleEattPsmMtu (*C var*), 1502
gBleExtAdvDefaultSetHandle_c (*C macro*), 1542
gBleExtAdvDefaultSetId_c (*C macro*), 1542
gBleExtAdvLegacySetHandle_c (*C macro*), 1542
gBleExtAdvLegacySetId_c (*C macro*), 1542
gBleExtAdvMaxAuxOffsetUsec_c (*C macro*), 1543
gBleExtAdvMaxSetId_c (*C macro*), 1542
gBleExtAdvNoDuration_c (*C macro*), 1542
gBleExtAdvNoMaxEvents_c (*C macro*), 1542
gBleHighDutyDirectedAdvDuration (*C macro*), 1542
gBleMaxActiveConnections (*C var*), 1502
gBleMaxADStructureLength_c (*C macro*), 1543
gBleMaxAdvSets (*C var*), 1502

gBleMaxExtAdvDataLength_c (*C macro*), 1543
gBleMaxTxOctets_c (*C macro*), 1542
gBleMaxTxTime_c (*C macro*), 1542
gBleMaxTxTimeCodedPhy_c (*C macro*), 1542
gBleMinTxOctets_c (*C macro*), 1541
gBleMinTxTime_c (*C macro*), 1541
gBleMonAdvRssiThresholdMax_c (*C macro*), 1544
gBleMonAdvRssiThresholdMin_c (*C macro*), 1544
gBleP256KeyLength_c (*C macro*), 1543
gBlePeriodicAdvDefaultHandle_c (*C macro*), 1542
gBlePeriodicAdvMaxSyncHandle_c (*C macro*), 1542
gBlePeriodicAdvOngoingSyncCancelHandle (*C macro*), 1365
gBlePeriodicAdvSkipMax_c (*C macro*), 1543
gBlePeriodicAdvSyncTimeoutMax_c (*C macro*), 1543
gBlePeriodicAdvSyncTimeoutMin_c (*C macro*), 1543
gBleSig_AlertLevel_d (*C macro*), 1512
gBleSig_AlertNotifControlPoint_d (*C macro*), 1514
gBleSig_AlertNotificationService_d (*C macro*), 1510
gBleSig_AlertStatus_d (*C macro*), 1514
gBleSig_BatteryLevel_d (*C macro*), 1512
gBleSig_BatteryService_d (*C macro*), 1510
gBleSig_BloodPressureFeature_d (*C macro*), 1514
gBleSig_BloodPressureService_d (*C macro*), 1510
gBleSig_BodySensorLocation_d (*C macro*), 1514
gBleSig_BootKeyboardInputReport_d (*C macro*), 1513
gBleSig_BootKeyboardOutputReport_d (*C macro*), 1513
gBleSig_BootMouseInputReport_d (*C macro*), 1513
gBleSig_BpEnhancedMeasurement_d (*C macro*), 1513
gBleSig_BpMeasurement_d (*C macro*), 1513
gBleSig_BpRecord_d (*C macro*), 1513
gBleSig_BtpService_d (*C macro*), 1511
gBleSig_CAR_NotSupported_d (*C macro*), 1517
gBleSig_CAR_Supported_d (*C macro*), 1517
gBleSig_CCCD_d (*C macro*), 1509
gBleSig_CentralAddressResolution_d (*C macro*), 1516
gBleSig_Characteristic_d (*C macro*), 1509
gBleSig_CharAggregateFormat_d (*C macro*), 1509
gBleSig_CharExtendedProperties_d (*C macro*), 1509
gBleSig_CharPresFormatDescriptor_d (*C macro*), 1509
gBleSig_CharUserDescription_d (*C macro*), 1509
gBleSig_CpControlPoint_d (*C macro*), 1516
gBleSig_CpFeature_d (*C macro*), 1515
gBleSig_CpMeasurement_d (*C macro*), 1515
gBleSig_CpVector_d (*C macro*), 1515
gBleSig_CscFeature_d (*C macro*), 1515
gBleSig_CscMeasurement_d (*C macro*), 1515
gBleSig_CurrentTime_d (*C macro*), 1513
gBleSig_CurrentTimeService_d (*C macro*), 1510
gBleSig_CyclingPowerService_d (*C macro*), 1511
gBleSig_CyclingSpeedAndCadenceService_d (*C macro*), 1511
gBleSig_DeviceInformationService_d (*C macro*), 1510
gBleSig_EncryptedDataKeyMaterial_d (*C macro*), 1517
gBleSig_FirmwareRevisionString_d (*C macro*), 1513
gBleSig_GapAppearance_d (*C macro*), 1511
gBleSig_GapDeviceName_d (*C macro*), 1511
gBleSig_GapPpcp_d (*C macro*), 1511
gBleSig_GattClientSupportedFeatures_d (*C macro*), 1511
gBleSig_GattDatabaseHash_d (*C macro*), 1512

gBleSig_GattSecurityLevels_d (*C macro*), 1517
gBleSig_GattServerSupportedFeatures_d (*C macro*), 1511
gBleSig_GattServiceChanged_d (*C macro*), 1511
gBleSig_GenericAccessProfile_d (*C macro*), 1510
gBleSig_GenericAttributeProfile_d (*C macro*), 1510
gBleSig_GlucoseFeature_d (*C macro*), 1515
gBleSig_GlucoseMeasurement_d (*C macro*), 1512
gBleSig_GlucoseMeasurementContext_d (*C macro*), 1513
gBleSig_GlucoseService_d (*C macro*), 1510
gBleSig_HardwareRevisionString_d (*C macro*), 1513
gBleSig_HealthThermometerService_d (*C macro*), 1510
gBleSig_HeartRateService_d (*C macro*), 1510
gBleSig_HidBootMouseInputReport_d (*C macro*), 1514
gBleSig_HidCtrlPoint_d (*C macro*), 1514
gBleSig_HidInformation_d (*C macro*), 1514
gBleSig_HidService_d (*C macro*), 1511
gBleSig_HrControlPoint_d (*C macro*), 1514
gBleSig_HrMeasurement_d (*C macro*), 1513
gBleSig_HTTP_ControlPoint_d (*C macro*), 1516
gBleSig_HTTP_EntityBody_d (*C macro*), 1516
gBleSig_HTTP_Headers_d (*C macro*), 1516
gBleSig_HTTP_StatusCode_d (*C macro*), 1516
gBleSig_HTTPProxyService_d (*C macro*), 1511
gBleSig_HTTPS_Security_d (*C macro*), 1516
gBleSig_IeeeRcdl_d (*C macro*), 1513
gBleSig_ImmediateAlertService_d (*C macro*), 1510
gBleSig_Include_d (*C macro*), 1509
gBleSig_IntermediateCuffPressure_d (*C macro*), 1513
gBleSig_IntermediateTemperature_d (*C macro*), 1512
gBleSig_IpsService_d (*C macro*), 1511
gBleSig_LinkLossService_d (*C macro*), 1510
gBleSig_LnControlPoint_d (*C macro*), 1516
gBleSig_LnFeature_d (*C macro*), 1516
gBleSig_LocalTimeInformation_d (*C macro*), 1512
gBleSig_LocationAndNavigationService_d (*C macro*), 1511
gBleSig_LocationAndSpeed_d (*C macro*), 1516
gBleSig_ManufacturerNameString_d (*C macro*), 1513
gBleSig_MeasurementInterval_d (*C macro*), 1512
gBleSig_MeshProvDataIn_d (*C macro*), 1517
gBleSig_MeshProvDataOut_d (*C macro*), 1517
gBleSig_MeshProvisioningService_d (*C macro*), 1516
gBleSig_MeshProxyDataIn_d (*C macro*), 1517
gBleSig_MeshProxyDataOut_d (*C macro*), 1517
gBleSig_MeshProxyService_d (*C macro*), 1517
gBleSig_ModelNumberString_d (*C macro*), 1512
gBleSig_Navigation_d (*C macro*), 1516
gBleSig_NewAlert_d (*C macro*), 1514
gBleSig_NextDSTChangeService_d (*C macro*), 1510
gBleSig_PhoneAlertStatusService_d (*C macro*), 1510
gBleSig_PlxContMeasurement_d (*C macro*), 1515
gBleSig_PlxSCMeasurement_d (*C macro*), 1515
gBleSig_PnpId_d (*C macro*), 1515
gBleSig_PositionQuality_d (*C macro*), 1516
gBleSig_PrimaryService_d (*C macro*), 1509
gBleSig_ProtocolMode_d (*C macro*), 1514
gBleSig_PulseOximeterFeature_d (*C macro*), 1515
gBleSig_PulseOximeterService_d (*C macro*), 1511
gBleSig_RaCtrlPoint_d (*C macro*), 1515

gBleSig_RangingService_d (*C macro*), 1517
gBleSig_RasControlPoint_d (*C macro*), 1517
gBleSig_RasFeature_d (*C macro*), 1517
gBleSig_RasOnDemandProcData_d (*C macro*), 1517
gBleSig_RasprocDataOverwritten_d (*C macro*), 1517
gBleSig_RasProcDataReady_d (*C macro*), 1517
gBleSig_RasRealTimeProcData_d (*C macro*), 1517
gBleSig_ReferenceTimeInformation_d (*C macro*), 1512
gBleSig_ReferenceTimeUpdateService_d (*C macro*), 1510
gBleSig_Report_d (*C macro*), 1514
gBleSig_ResolvablePrivateAddressOnly_d (*C macro*), 1516
gBleSig_RingerControlPoint_d (*C macro*), 1514
gBleSig_RingerSetting_d (*C macro*), 1514
gBleSig_RPAO_Used_d (*C macro*), 1517
gBleSig_RscFeature_d (*C macro*), 1515
gBleSig_RscMeasurement_d (*C macro*), 1515
gBleSig_RunningSpeedAndCadenceService_d (*C macro*), 1511
gBleSig_ScanIntervalWindow_d (*C macro*), 1515
gBleSig_SCCD_d (*C macro*), 1509
gBleSig_ScControlPoint_d (*C macro*), 1515
gBleSig_SecondaryService_d (*C macro*), 1509
gBleSig_SensorLocation_d (*C macro*), 1515
gBleSig_SerialNumberString_d (*C macro*), 1513
gBleSig_SoftwareRevisionString_d (*C macro*), 1513
gBleSig_SupportedNewAlertCategory_d (*C macro*), 1514
gBleSig_SupportedUnreadAlertCategory_d (*C macro*), 1514
gBleSig_SystemId_d (*C macro*), 1512
gBleSig_Temperature_d (*C macro*), 1516
gBleSig_TemperatureMeasurement_d (*C macro*), 1512
gBleSig_TemperatureType_d (*C macro*), 1512
gBleSig_TimeUpdateControlPoint_d (*C macro*), 1512
gBleSig_TimeUpdateState_d (*C macro*), 1512
gBleSig_TimeWithDST_d (*C macro*), 1512
gBleSig_TireAcceleration_d (*C macro*), 1518
gBleSig_TirePressure_d (*C macro*), 1518
gBleSig_TirePressureMonitoringService_d (*C macro*), 1518
gBleSig_TireTemperature_d (*C macro*), 1518
gBleSig_TpmsMonitoringModeDutyCycle_d (*C macro*), 1518
gBleSig_TpmsPosition_d (*C macro*), 1518
gBleSig_TpmsProperties_d (*C macro*), 1518
gBleSig_TpmsSigningKey_d (*C macro*), 1518
gBleSig_TxPower_d (*C macro*), 1512
gBleSig_TxPowerService_d (*C macro*), 1510
gBleSig_UnreadAlertStatus_d (*C macro*), 1514
gBleSig_URI_d (*C macro*), 1516
gBleSig_ValidRangeDescriptor_d (*C macro*), 1509
gBleSig_WPTService_d (*C macro*), 1511
gBleUuidType16_c (*C macro*), 1506
gBleUuidType32_c (*C macro*), 1506
gBleUuidType128_c (*C macro*), 1506
gbmpDIAM_ConnectableUndirected_c (*C macro*), 1367
gbmpDIAM_NonConnectableNonScannableUndirected_c (*C macro*), 1367
gbmpDIAM_ScannableUndirected_c (*C macro*), 1367
gBrEdrNotSupported_c (*C macro*), 1365
gCancelOngoingInitiatingConnection_d (*C macro*), 1361
gcBleChannelMapSize_c (*C macro*), 1541
gcBleDeviceAddressSize_c (*C macro*), 1540
gcBleLongUuidSize_c (*C macro*), 1540

gCccdEmpty_c (*C macro*), 1408
gCccdIndication_c (*C macro*), 1409
gCccdNotification_c (*C macro*), 1409
gcConnectionEventMaxDefault_c (*C var*), 1502
gcConnectionEventMinDefault_c (*C var*), 1502
gcConnectionIntervalMax_c (*C macro*), 1504
gcConnectionIntervalMaxDefault_c (*C macro*), 1505
gcConnectionIntervalMin_c (*C macro*), 1504
gcConnectionIntervalMinDefault_c (*C macro*), 1505
gcConnectionPeripheralLatencyDefault_c (*C macro*), 1505
gcConnectionPeripheralLatencyMax_c (*C macro*), 1505
gcConnectionSubrateContinuationNumberMax_c (*C macro*), 1505
gcConnectionSubrateFactorMax_c (*C macro*), 1505
gcConnectionSubrateFactorMin_c (*C macro*), 1505
gcConnectionSupervisionTimeoutDefault_c (*C macro*), 1505
gcConnectionSupervisionTimeoutMax_c (*C macro*), 1505
gcConnectionSupervisionTimeoutMin_c (*C macro*), 1505
gcDecisionDataKeySize_c (*C macro*), 1543
gcDecisionDataMaxSize_c (*C macro*), 1543
gcDecisionDataPrandSize_c (*C macro*), 1543
gcDecisionDataResolvableTagSize_c (*C macro*), 1543
gcDecisionInstructionsArbitraryDataMaskSize_c (*C macro*), 1543
gcDecisionInstructionsArbitraryDataTargetSize_c (*C macro*), 1543
gcDecisionInstructionsParamSize_c (*C macro*), 1543
gcEadAadValue_c (*C macro*), 1544
gcEadIvSize_c (*C macro*), 1544
gcEadKeySize_c (*C macro*), 1544
gcEadMicSize_c (*C macro*), 1544
gcEadRandomizerSize_c (*C macro*), 1543
gcGapMaxAdvertisingDataLength_c (*C macro*), 1541
gcGapMaxAuthorizationHandles_c (*C macro*), 1540
gcGapMaxDeviceNameSize_c (*C macro*), 1541
gcGapMaximumSavedCccds_c (*C macro*), 1540
gcMaxNumTestsInDecisionInstruction_c (*C macro*), 1543
gcReservedFlashSizeForCustomInformation_c (*C macro*), 1541
gCsAccessAdressSize_c (*C macro*), 1508
gcSecureModeSavedLocalKeysNo_c (*C macro*), 1540
gCSMaxSubeventLen_c (*C macro*), 1508
gcSmpCsrkSize_c (*C macro*), 1540
gcSmpIrkSize_c (*C macro*), 1540
gcSmpMaxBlobSize_c (*C macro*), 1540
gcSmpMaxIrkBlobSize_c (*C macro*), 1540
gcSmpMaxLtkSize_c (*C macro*), 1540
gcSmpMaxRandSize_c (*C macro*), 1541
gcSmpOobSize_c (*C macro*), 1541
gCSNumPhysMax_c (*C macro*), 1508
gCSNumToneAntennaIds_c (*C macro*), 1508
gCSReflectorTableSize_c (*C macro*), 1508
gCsrk_c (*C macro*), 1365
gCSSyncRandomSize_c (*C macro*), 1508
gCSTestMaxChannelLength_c (*C macro*), 1508
gDefaultEncryptionKeySize_d (*C macro*), 1362
gDevicePrivacy_c (*C macro*), 1507
gDHKeySize_c (*C macro*), 1507
gDRBGKeySize_c (*C macro*), 1508
gEattMaxMtu_c (*C macro*), 1541
gEattMinMtu_c (*C macro*), 1541
gEnableCoredumpPackets (*C var*), 1502

gEnhancedL2capMinimumMps_c (*C macro*), 1552
getConnParams_t (*C type*), 1500
getConnParams_tag (*C struct*), 1524, 1559
getConnParams_tag.aChMapBm (*C var*), 1524, 1559
getConnParams_tag.aCrcInitVal (*C var*), 1524, 1559
getConnParams_tag.aucRemoteMasRxPHY (*C var*), 1525, 1559
getConnParams_tag.connectionHandle (*C var*), 1524, 1559
getConnParams_tag.seqNum (*C var*), 1525, 1559
getConnParams_tag.ucCentralSCA (*C var*), 1525, 1559
getConnParams_tag.ucChannelSelection (*C var*), 1524, 1559
getConnParams_tag.ucHop (*C var*), 1525, 1559
getConnParams_tag.ucRole (*C var*), 1525, 1559
getConnParams_tag.ucUnMapChIdx (*C var*), 1525, 1559
getConnParams_tag.uiAnchorDelay (*C var*), 1525, 1559
getConnParams_tag.uiConnEvent (*C var*), 1525, 1559
getConnParams_tag.uiConnInterval (*C var*), 1524, 1559
getConnParams_tag.uiConnLatency (*C var*), 1524, 1559
getConnParams_tag.uiSuperTO (*C var*), 1524, 1559
getConnParams_tag.ulAnchorClk (*C var*), 1525, 1559
getConnParams_tag.ulRxInstant (*C var*), 1525, 1559
getConnParams_tag.ulTxAccCode (*C var*), 1524, 1559
getLeExtendedFeatureBit (*C macro*), 1509
getLeExtendedFeatureByte (*C macro*), 1509
getSecurityLevel (*C macro*), 1362
getSecurityMode (*C macro*), 1362
gExpandAsEnum_m (*C macro*), 1552
gExpandAsTable_m (*C macro*), 1552
gExtendedFilteredScan_c (*C macro*), 1507
gExtendedUnfilteredScan_c (*C macro*), 1507
gGapAdvertisingChannelMapDefault_c (*C macro*), 1363
gGapAdvertisingIntervalDefault_c (*C macro*), 1362
gGapAdvertisingIntervalRangeMaximum_c (*C macro*), 1362
gGapAdvertisingIntervalRangeMinimum_c (*C macro*), 1362
gGapChSelAlgorithmNo2 (*C macro*), 1364
gGapConnEventLengthMax_d (*C macro*), 1364
gGapConnEventLengthMin_d (*C macro*), 1364
gGapConnIntervalMax_d (*C macro*), 1364
gGapConnIntervalMin_d (*C macro*), 1364
gGapConnLatencyMax_d (*C macro*), 1364
gGapConnLatencyMin_d (*C macro*), 1364
gGapConnSuperTimeoutMax_d (*C macro*), 1364
gGapConnSuperTimeoutMin_d (*C macro*), 1364
gGapCteMaxCount_c (*C macro*), 1367
gGapCteMaxLength_c (*C macro*), 1367
gGapCteMinCount_c (*C macro*), 1367
gGapCteMinLength_c (*C macro*), 1367
gGapDefaultAdvertisingParameters_d (*C macro*), 1363
gGapDefaultConnectionLatency_d (*C macro*), 1364
gGapDefaultConnectionRequestParameters_d (*C macro*), 1364
gGapDefaultDeviceSecurity_d (*C macro*), 1362
gGapDefaultExtAdvertisingParameters_d (*C macro*), 1363
gGapDefaultMaxConnectionInterval_d (*C macro*), 1364
gGapDefaultMinConnectionInterval_d (*C macro*), 1364
gGapDefaultPeriodicAdvParameters_d (*C macro*), 1363
gGapDefaultScanningParameters_d (*C macro*), 1364
gGapDefaultSecurityRequirements_d (*C macro*), 1362
gGapDefaultSupervisionTimeout_d (*C macro*), 1364
gGapEattMaxBearers (*C macro*), 1367

gGapExtAdvertisingIntervalDefault_c (*C macro*), 1362
gGapExtAdvertisingIntervalRangeMaximum_c (*C macro*), 1362
gGapExtAdvertisingIntervalRangeMinimum_c (*C macro*), 1362
gGapInvalidSyncHandle (*C macro*), 1365
gGapMaxSwitchingPatternLength_c (*C macro*), 1367
gGapMinSwitchingPatternLength_c (*C macro*), 1367
gGapPeriodicAdvIntervalDefault_c (*C macro*), 1363
gGapPeriodicAdvIntervalRangeMaximum_c (*C macro*), 1363
gGapPeriodicAdvIntervalRangeMinimum_c (*C macro*), 1362
gGapRssiMax_d (*C macro*), 1363
gGapRssiMin_d (*C macro*), 1363
gGapRssiNotAvailable_d (*C macro*), 1363
gGapScanContinuously_d (*C macro*), 1363
gGapScanIntervalDefault_d (*C macro*), 1363
gGapScanIntervalMax_d (*C macro*), 1363
gGapScanIntervalMin_d (*C macro*), 1363
gGapScanPeriodicDisabled_d (*C macro*), 1364
gGapScanWindowDefault_d (*C macro*), 1363
gGapScanWindowMax_d (*C macro*), 1363
gGapScanWindowMin_d (*C macro*), 1363
gGattDatabaseHashSize_c (*C macro*), 1455
gGattDbAttributeCount_c (*C var*), 1449
gGattDbInvalidHandle_d (*C macro*), 1454
gGattDbInvalidHandleIndex_d (*C macro*), 1454
gGattDynamicAttrSize (*C var*), 1449
gGattDynamicValSize (*C var*), 1449
gHci2Host_TaskQueue (*C var*), 1502
gHCICSChannelMapSize (*C macro*), 1508
gHciTransportUartChannel_c (*C macro*), 1541
gHostInitEnableExpmFeatures (*C var*), 1502
gHostInitEnableLePowerControl (*C var*), 1502
gIncompatibleIoCapabilities_c (*C macro*), 1366
gInvalidDeviceId_c (*C macro*), 1504
gInvalidNvmIndex_c (*C macro*), 1504
gIoDisplayOnly_c (*C macro*), 1367
gIoDisplayYesNo_c (*C macro*), 1367
gIoKeyboardDisplay_c (*C macro*), 1367
gIoKeyboardOnly_c (*C macro*), 1367
gIoNone_c (*C macro*), 1367
gIrk_c (*C macro*), 1365
gL2caLePsmDynamicFirst_c (*C macro*), 1552
gL2caLePsmDynamicLast_c (*C macro*), 1552
gL2caLePsmSigAssignedEatt_c (*C macro*), 1551
gL2caLePsmSigAssignedFirst_c (*C macro*), 1551
gL2caLePsmSigAssignedLast_c (*C macro*), 1551
gL2capCidAtt_c (*C macro*), 1551
gL2capCidLePsmDynamicFirst_c (*C macro*), 1551
gL2capCidLePsmDynamicLast_c (*C macro*), 1551
gL2capCidNotApplicable_c (*C macro*), 1551
gL2capCidNull_c (*C macro*), 1551
gL2capCidSigAssignedFirst_c (*C macro*), 1551
gL2capCidSigAssignedLast_c (*C macro*), 1551
gL2capCidSignaling_c (*C macro*), 1551
gL2capCidSmp_c (*C macro*), 1551
gL2capDefaultMps_c (*C macro*), 1552
gL2capDefaultMtu_c (*C macro*), 1552
gL2capEnhancedMaxChannels_c (*C macro*), 1552
gL2capHeaderLength_c (*C macro*), 1552

gL2capMaximumMps_c (*C macro*), 1552
gLeExtendedFeaturesSize_c (*C macro*), 1508
gLeGeneralDiscoverableMode_c (*C macro*), 1365
gLeLimitedDiscoverableMode_c (*C macro*), 1365
gLeModulationTestMaxBleRfChannel_c (*C macro*), 1544
gLeModulationTestMaxGenericChannel_c (*C macro*), 1544
gLeModulationTestMaxPowerIndex_c (*C macro*), 1544
gLeModulationTestOptionGenericChannel_c (*C macro*), 1509
gLeModulationTestOptionPhy2M_c (*C macro*), 1509
gLeModulationTestOptionPowerIndex_c (*C macro*), 1509
gLePhy1MFlag_c (*C macro*), 1506
gLePhy2MFlag_c (*C macro*), 1506
gLePhyCodedFlag_c (*C macro*), 1506
gLePsmSigAssignedNumbersTable_m (*C macro*), 1552
gLinkEncryptionFailed_c (*C macro*), 1367
gLowEncryptionKeySize_c (*C macro*), 1366
gLtk_c (*C macro*), 1365
gMaxAdvReportQueueSize (*C var*), 1502
gMaxEncryptionKeySize_d (*C macro*), 1362
gMode_2_Mask_d (*C macro*), 1361
gNetworkPrivacy_c (*C macro*), 1507
gNoKeys_c (*C macro*), 1365
gNone_c (*C macro*), 1365
gOobNotAvailable_c (*C macro*), 1366
gPairingNotSupported_c (*C macro*), 1366
gPermissionFlagReadable_c (*C macro*), 1455
gPermissionFlagReadWithAuthentication_c (*C macro*), 1455
gPermissionFlagReadWithAuthorization_c (*C macro*), 1455
gPermissionFlagReadWithEncryption_c (*C macro*), 1455
gPermissionFlagWritable_c (*C macro*), 1455
gPermissionFlagWriteWithAuthentication_c (*C macro*), 1455
gPermissionFlagWriteWithAuthorization_c (*C macro*), 1455
gPermissionFlagWriteWithEncryption_c (*C macro*), 1455
gPermissionNone_c (*C macro*), 1455
gRepeatedAttempts_c (*C macro*), 1366
gScanAll_c (*C macro*), 1507
gScanAllPDUs_c (*C macro*), 1507
gScanOnlyDecisionPDUs_c (*C macro*), 1507
gScanOnlyNonDecisionPDUs_c (*C macro*), 1507
gScanRspDataChange_c (*C macro*), 1506
gScanWithFilterAcceptList_c (*C macro*), 1507
gSecurityLevel_LeSecureConnections_c (*C macro*), 1366
gSecurityLevel_NoMitmProtection_c (*C macro*), 1366
gSecurityLevel_NoSecurity_c (*C macro*), 1365
gSecurityLevel_WithMitmProtection_c (*C macro*), 1366
gSecurityMode_1_c (*C macro*), 1365
gSecurityMode_1_Level_1_c (*C macro*), 1366
gSecurityMode_1_Level_2_c (*C macro*), 1366
gSecurityMode_1_Level_3_c (*C macro*), 1366
gSecurityMode_1_Level_4_c (*C macro*), 1366
gSecurityMode_2_c (*C macro*), 1365
gSecurityMode_2_Level_1_c (*C macro*), 1366
gSecurityMode_2_Level_2_c (*C macro*), 1366
gSimultaneousLeBrEdrCapableController_c (*C macro*), 1365
gSimultaneousLeBrEdrCapableHost_c (*C macro*), 1365
gSkdSize_c (*C macro*), 1507
gSmpLeScRandomConfirmValueSize_c (*C macro*), 1541
gSmpLeScRandomValueSize_c (*C macro*), 1541

gUnenhancedBearerId_c (*C macro*), 1507
gUnspecifiedReason_c (*C macro*), 1366
gUseDecisionPDU_UseFilterAcceptListForOtherPDUs_c (*C macro*), 1507
gUseDeviceAddress_c (*C macro*), 1506
gUseDeviceAddressNoDecisionPDUs_c (*C macro*), 1506
gUseFilterAcceptList_c (*C macro*), 1506
gUseFilterAcceptListAllPDUs_c (*C macro*), 1506
gUseFilterAcceptListNoDecisionPDUs_c (*C macro*), 1506
gUseHciCommandFlowControl (*C var*), 1502
gUseOnlyDecisionPDUs_c (*C macro*), 1506
gVendorDebugEventSize_c (*C macro*), 1508
gVendorHandoverMaxCsLlContextSize_c (*C macro*), 1508
gVendorUnitaryTestSize_c (*C macro*), 1508

H

handoverAnchorMonitorEvent_t (*C type*), 1500
handoverAnchorMonitorEvent_tag (*C struct*), 1527, 1561
handoverAnchorMonitorEvent_tag.anchorClock625Us (*C var*), 1528, 1561
handoverAnchorMonitorEvent_tag.anchorDelay (*C var*), 1528, 1561
handoverAnchorMonitorEvent_tag.chIdx (*C var*), 1528, 1561
handoverAnchorMonitorEvent_tag.connectionHandle (*C var*), 1527, 1561
handoverAnchorMonitorEvent_tag.connEvent (*C var*), 1527, 1561
handoverAnchorMonitorEvent_tag.lqiActive (*C var*), 1528, 1561
handoverAnchorMonitorEvent_tag.lqiRemote (*C var*), 1527, 1561
handoverAnchorMonitorEvent_tag.rssiActive (*C var*), 1527, 1561
handoverAnchorMonitorEvent_tag.rssiRemote (*C var*), 1527, 1561
handoverAnchorMonitorEvent_tag.statusActive (*C var*), 1528, 1561
handoverAnchorMonitorEvent_tag.statusRemote (*C var*), 1527, 1561
handoverAnchorMonitorEvent_tag.ucNbReports (*C var*), 1528, 1561
handoverAnchorMonitorPacketContinueEvent_t (*C type*), 1501
handoverAnchorMonitorPacketContinueEvent_tag (*C struct*), 1529, 1562
handoverAnchorMonitorPacketContinueEvent_tag.connectionHandle (*C var*), 1529, 1562
handoverAnchorMonitorPacketContinueEvent_tag.packetCounter (*C var*), 1529, 1562
handoverAnchorMonitorPacketContinueEvent_tag.pduSize (*C var*), 1529, 1562
handoverAnchorMonitorPacketContinueEvent_tag.pPdu (*C var*), 1529, 1562
handoverAnchorMonitorPacketEvent_t (*C type*), 1500
handoverAnchorMonitorPacketEvent_tag (*C struct*), 1528, 1562
handoverAnchorMonitorPacketEvent_tag.anchorClock625Us (*C var*), 1529, 1562
handoverAnchorMonitorPacketEvent_tag.anchorDelay (*C var*), 1529, 1562
handoverAnchorMonitorPacketEvent_tag.chIdx (*C var*), 1528, 1562
handoverAnchorMonitorPacketEvent_tag.connectionHandle (*C var*), 1528, 1562
handoverAnchorMonitorPacketEvent_tag.connEvent (*C var*), 1528, 1562
handoverAnchorMonitorPacketEvent_tag.lqiPacket (*C var*), 1528, 1562
handoverAnchorMonitorPacketEvent_tag.packetCounter (*C var*), 1528, 1562
handoverAnchorMonitorPacketEvent_tag.pduSize (*C var*), 1529, 1562
handoverAnchorMonitorPacketEvent_tag.phy (*C var*), 1528, 1562
handoverAnchorMonitorPacketEvent_tag.pPdu (*C var*), 1529, 1562
handoverAnchorMonitorPacketEvent_tag.rssiPacket (*C var*), 1528, 1562
handoverAnchorMonitorPacketEvent_tag.statusPacket (*C var*), 1528, 1562
handoverAnchorMonitorPacketEvent_tag.ucNbConnIntervals (*C var*), 1529, 1562
handoverAnchorNotificationStateChanged_t (*C type*), 1501
handoverAnchorNotificationStateChanged_tag (*C struct*), 1533, 1565
handoverAnchorNotificationStateChanged_tag.connectionHandle (*C var*), 1533, 1565
handoverAnchorSearchStart_t (*C type*), 1500
handoverAnchorSearchStart_tag (*C struct*), 1525, 1560
handoverAnchorSearchStart_tag.connectionHandle (*C var*), 1526, 1560
handoverAnchorSearchStart_tag.status (*C var*), 1526, 1560
handoverAnchorSearchStop_t (*C type*), 1500

handoverAnchorSearchStop_tag (*C struct*), 1526, 1560
handoverAnchorSearchStop_tag.connectionHandle (*C var*), 1526, 1560
handoverAnchorSearchStop_tag.status (*C var*), 1526, 1560
handoverApplyConnectionUpdateProcedure_t (*C type*), 1501
handoverApplyConnectionUpdateProcedure_tag (*C struct*), 1533, 1565
handoverApplyConnectionUpdateProcedure_tag.connectionHandle (*C var*), 1533, 1565
handoverConnect_t (*C type*), 1500
handoverConnect_tag (*C struct*), 1526, 1560
handoverConnect_tag.connectionHandle (*C var*), 1526, 1560
handoverConnect_tag.status (*C var*), 1526, 1560
handoverConnectionUpdateProcedureEvent_t (*C type*), 1501
handoverConnectionUpdateProcedureEvent_tag (*C struct*), 1529, 1562
handoverConnectionUpdateProcedureEvent_tag.connectionHandle (*C var*), 1529, 1563
handoverConnectionUpdateProcedureEvent_tag.currentEventCounter (*C var*), 1530, 1563
handoverConnectionUpdateProcedureEvent_tag.instant (*C var*), 1530, 1563
handoverConnectionUpdateProcedureEvent_tag.interval (*C var*), 1530, 1563
handoverConnectionUpdateProcedureEvent_tag.latency (*C var*), 1530, 1563
handoverConnectionUpdateProcedureEvent_tag.timeout (*C var*), 1530, 1563
handoverConnectionUpdateProcedureEvent_tag.winOffset (*C var*), 1530, 1563
handoverConnectionUpdateProcedureEvent_tag.winSize (*C var*), 1529, 1563
handoverConnParamUpdateEvent_t (*C type*), 1501
handoverConnParamUpdateEvent_tag (*C struct*), 1530, 1563
handoverConnParamUpdateEvent_tag.aChMapBm (*C var*), 1531, 1564
handoverConnParamUpdateEvent_tag.aCrcInitVal (*C var*), 1531, 1563
handoverConnParamUpdateEvent_tag.aucRemoteMasRxPHY (*C var*), 1531, 1564
handoverConnParamUpdateEvent_tag.connectionHandle (*C var*), 1531, 1563
handoverConnParamUpdateEvent_tag.seqNum (*C var*), 1531, 1564
handoverConnParamUpdateEvent_tag.status (*C var*), 1531, 1563
handoverConnParamUpdateEvent_tag.ucCentralSCA (*C var*), 1531, 1564
handoverConnParamUpdateEvent_tag.ucChannelSelection (*C var*), 1531, 1564
handoverConnParamUpdateEvent_tag.ucHop (*C var*), 1531, 1564
handoverConnParamUpdateEvent_tag.ucRole (*C var*), 1531, 1564
handoverConnParamUpdateEvent_tag.ucUnMapChIdx (*C var*), 1531, 1564
handoverConnParamUpdateEvent_tag.uiAnchorDelay (*C var*), 1532, 1564
handoverConnParamUpdateEvent_tag.uiConnEvent (*C var*), 1531, 1564
handoverConnParamUpdateEvent_tag.uiConnInterval (*C var*), 1531, 1563
handoverConnParamUpdateEvent_tag.uiConnLatency (*C var*), 1531, 1564
handoverConnParamUpdateEvent_tag.uiSuperTO (*C var*), 1531, 1564
handoverConnParamUpdateEvent_tag.ulAnchorClk (*C var*), 1532, 1564
handoverConnParamUpdateEvent_tag.ulRxInstant (*C var*), 1532, 1564
handoverConnParamUpdateEvent_tag.ulTxAccCode (*C var*), 1531, 1563
handoverGetCsLlContext_t (*C type*), 1500
handoverGetCsLlContext_tag (*C struct*), 1527, 1561
handoverGetCsLlContext_tag.llContext (*C var*), 1527, 1561
handoverGetCsLlContext_tag.llContextLength (*C var*), 1527, 1561
handoverGetCsLlContext_tag.responseMask (*C var*), 1527, 1561
handoverGetCsLlContext_tag.status (*C var*), 1527, 1561
handoverGetData_t (*C type*), 1500
handoverGetData_tag (*C struct*), 1526, 1560
handoverGetData_tag.pData (*C var*), 1526, 1560
handoverGetData_tag.status (*C var*), 1526, 1560
handoverGetTime_t (*C type*), 1500
handoverGetTime_tag (*C struct*), 1525, 1559
handoverGetTime_tag.slot (*C var*), 1525, 1560
handoverGetTime_tag.status (*C var*), 1525, 1560
handoverGetTime_tag.us_offset (*C var*), 1525, 1560
handoverLlPendingDataIndication_t (*C type*), 1501
handoverLlPendingDataIndication_tag (*C struct*), 1533, 1565

handoverLlPendingDataIndication_tag.dataSize (*C var*), 1533, 1565
handoverLlPendingDataIndication_tag.pData (*C var*), 1533, 1565
handoverResumeTransmitCompleteEvent_t (*C type*), 1501
handoverResumeTransmitCompleteEvent_tag (*C struct*), 1532, 1564
handoverResumeTransmitCompleteEvent_tag.connectionHandle (*C var*), 1532, 1565
handoverSetData_t (*C type*), 1500
handoverSetData_tag (*C struct*), 1526, 1560
handoverSetData_tag.pData (*C var*), 1527, 1561
handoverSetData_tag.status (*C var*), 1527, 1561
handoverSuspendTransmitCompleteEvent_t (*C type*), 1501
handoverSuspendTransmitCompleteEvent_tag (*C struct*), 1532, 1564
handoverSuspendTransmitCompleteEvent_tag.connectionHandle (*C var*), 1532, 1564
handoverSuspendTransmitCompleteEvent_tag.noOfPendingAclPackets (*C var*), 1532, 1564
handoverSuspendTransmitCompleteEvent_tag.sizeOfDataNAckInOldestPacket (*C var*), 1532, 1564
handoverSuspendTransmitCompleteEvent_tag.sizeOfDataTxInOldestPacket (*C var*), 1532, 1564
handoverSuspendTransmitCompleteEvent_tag.sizeOfPendingAclPackets (*C var*), 1532, 1564
handoverTimeSyncEvent_t (*C type*), 1501
handoverTimeSyncEvent_tag (*C struct*), 1530, 1563
handoverTimeSyncEvent_tag.rssi (*C var*), 1530, 1563
handoverTimeSyncEvent_tag.rxClkSlot (*C var*), 1530, 1563
handoverTimeSyncEvent_tag.rxUs (*C var*), 1530, 1563
handoverTimeSyncEvent_tag.txClkSlot (*C var*), 1530, 1563
handoverTimeSyncEvent_tag.txUs (*C var*), 1530, 1563
handoverUpdateConnParams_t (*C type*), 1501
handoverUpdateConnParams_tag (*C struct*), 1532, 1565
handoverUpdateConnParams_tag.connectionHandle (*C var*), 1533, 1565
handoverUpdateConnParams_tag.status (*C var*), 1533, 1565
hciHostToControllerInterface_t (*C type*), 1501
hciPacketType_t (*C enum*), 1473
hciPacketType_t.gHciCommandPacket_c (*C enumerator*), 1473
hciPacketType_t.gHciDataPacket_c (*C enumerator*), 1473
hciPacketType_t.gHciEventPacket_c (*C enumerator*), 1474
hciPacketType_t.gHciIsoDataPacket_c (*C enumerator*), 1474
hciPacketType_t.gHciSynchronousDataPacket_c (*C enumerator*), 1473
Host_TaskHandler (*C function*), 1504

I

IncrementU8Saturating (*C function*), 1504
IncrementU16Saturating (*C function*), 1504
isMode_1 (*C macro*), 1362
isMode_2 (*C macro*), 1362
isSameMode (*C macro*), 1362
isSupportedLeExtendedFeature (*C macro*), 1509

L

L2ca_CancelConnection (*C function*), 1549
L2ca_ConnectLePsm (*C function*), 1548
L2ca_DeregisterLePsm (*C function*), 1548
L2ca_DisconnectLeCbChannel (*C function*), 1548
L2ca_EnhancedCancelConnection (*C function*), 1551
L2ca_EnhancedChannelReconfigure (*C function*), 1550
L2ca_EnhancedConnectLePsm (*C function*), 1550
L2ca_RegisterLeCbCallbacks (*C function*), 1547
L2ca_RegisterLePsm (*C function*), 1547
L2ca_SendLeCbData (*C function*), 1549
L2ca_SendLeCredit (*C function*), 1549
l2caChannelStatus_t (*C enum*), 1546
l2caChannelStatus_t.gL2ca_ChannelStatusChannelBusy_c (*C enumerator*), 1546

l2caChannelStatus_t.gL2ca_ChannelStatusChannelIdle_c (*C enumerator*), 1546
l2caControlCallback_t (*C type*), 1547
l2caEattCallback_t (*C type*), 1547
l2caEnhancedConnectionComplete_t (*C type*), 1547
l2caEnhancedConnectionComplete_tag (*C struct*), 1553, 1610
l2caEnhancedConnectionComplete_tag.aCids (*C var*), 1610
l2caEnhancedConnectionComplete_tag.deviceId (*C var*), 1610
l2caEnhancedConnectionComplete_tag.initialCredits (*C var*), 1610
l2caEnhancedConnectionComplete_tag.noOfChannels (*C var*), 1610
l2caEnhancedConnectionComplete_tag.peerMps (*C var*), 1610
l2caEnhancedConnectionComplete_tag.peerMtu (*C var*), 1610
l2caEnhancedConnectionComplete_tag.result (*C var*), 1610
l2caEnhancedConnectionRequest_t (*C type*), 1547
l2caEnhancedConnectionRequest_tag (*C struct*), 1553, 1609
l2caEnhancedConnectionRequest_tag.aCids (*C var*), 1610
l2caEnhancedConnectionRequest_tag.deviceId (*C var*), 1609
l2caEnhancedConnectionRequest_tag.initialCredits (*C var*), 1610
l2caEnhancedConnectionRequest_tag.lePsm (*C var*), 1609
l2caEnhancedConnectionRequest_tag.noOfChannels (*C var*), 1610
l2caEnhancedConnectionRequest_tag.peerMps (*C var*), 1609
l2caEnhancedConnectionRequest_tag.peerMtu (*C var*), 1609
l2caEnhancedReconfigureRequest_t (*C type*), 1547
l2caEnhancedReconfigureRequest_tag (*C struct*), 1553, 1610
l2caEnhancedReconfigureRequest_tag.aCids (*C var*), 1610
l2caEnhancedReconfigureRequest_tag.deviceId (*C var*), 1610
l2caEnhancedReconfigureRequest_tag.newMps (*C var*), 1610
l2caEnhancedReconfigureRequest_tag.newMtu (*C var*), 1610
l2caEnhancedReconfigureRequest_tag.noOfChannels (*C var*), 1610
l2caEnhancedReconfigureRequest_tag.result (*C var*), 1610
l2caEnhancedReconfigureResponse_t (*C type*), 1547
l2caEnhancedReconfigureResponse_tag (*C struct*), 1553, 1610
l2caEnhancedReconfigureResponse_tag.deviceId (*C var*), 1611
l2caEnhancedReconfigureResponse_tag.result (*C var*), 1611
l2caErrorSource_t (*C enum*), 1545
l2caErrorSource_t.gL2ca_CancelConnection_c (*C enumerator*), 1545
l2caErrorSource_t.gL2ca_DisconnectLePsm_c (*C enumerator*), 1545
l2caErrorSource_t.gL2ca_EnhancedCancelConnection_c (*C enumerator*), 1545
l2caErrorSource_t.gL2ca_EnhancedReconfigureReq_c (*C enumerator*), 1545
l2caErrorSource_t.gL2ca_HandleLeFlowControlCredit_c (*C enumerator*), 1545
l2caErrorSource_t.gL2ca_HandleRecvLeCbData_c (*C enumerator*), 1545
l2caErrorSource_t.gL2ca_HandleSendLeCbData_c (*C enumerator*), 1545
l2caErrorSource_t.gL2ca_SendLeFlowControlCredit_c (*C enumerator*), 1545
l2caGenericCallback_t (*C type*), 1547
l2caHandoverConnectionComplete_t (*C type*), 1546
l2caHandoverConnectionComplete_tag (*C struct*), 1552, 1608
l2caHandoverConnectionComplete_tag.cId (*C var*), 1608
l2caHandoverConnectionComplete_tag.credits (*C var*), 1608
l2caHandoverConnectionComplete_tag.deviceId (*C var*), 1608
l2caHandoverConnectionComplete_tag.peerMps (*C var*), 1608
l2caHandoverConnectionComplete_tag.peerMtu (*C var*), 1608
l2caLeCbChannelStatusNotification_t (*C type*), 1547
l2caLeCbChannelStatusNotification_tag (*C struct*), 1552, 1609
l2caLeCbChannelStatusNotification_tag.cId (*C var*), 1609
l2caLeCbChannelStatusNotification_tag.deviceId (*C var*), 1609
l2caLeCbChannelStatusNotification_tag.status (*C var*), 1609
l2caLeCbConnectionComplete_t (*C type*), 1546
l2caLeCbConnectionComplete_tag (*C struct*), 1552, 1607
l2caLeCbConnectionComplete_tag.cId (*C var*), 1607

l2caLeCbConnectionComplete_tag.deviceId (*C var*), 1607
l2caLeCbConnectionComplete_tag.initialCredits (*C var*), 1608
l2caLeCbConnectionComplete_tag.peerMps (*C var*), 1607
l2caLeCbConnectionComplete_tag.peerMtu (*C var*), 1607
l2caLeCbConnectionComplete_tag.result (*C var*), 1608
l2caLeCbConnectionRequest_t (*C type*), 1546
l2caLeCbConnectionRequest_tag (*C struct*), 1552, 1607
l2caLeCbConnectionRequest_tag.deviceId (*C var*), 1607
l2caLeCbConnectionRequest_tag.initialCredits (*C var*), 1607
l2caLeCbConnectionRequest_tag.lePsm (*C var*), 1607
l2caLeCbConnectionRequest_tag.peerMps (*C var*), 1607
l2caLeCbConnectionRequest_tag.peerMtu (*C var*), 1607
l2caLeCbConnectionRequestResult_t (*C enum*), 1544
l2caLeCbConnectionRequestResult_t.gCommandRejected_c (*C enumerator*), 1545
l2caLeCbConnectionRequestResult_t.gInsufficientAuthentication_c (*C enumerator*), 1544
l2caLeCbConnectionRequestResult_t.gInsufficientAuthorization_c (*C enumerator*), 1544
l2caLeCbConnectionRequestResult_t.gInsufficientEncryption_c (*C enumerator*), 1544
l2caLeCbConnectionRequestResult_t.gInsufficientEncryptionKeySize_c (*C enumerator*), 1544
l2caLeCbConnectionRequestResult_t.gInvalidParameters_c (*C enumerator*), 1545
l2caLeCbConnectionRequestResult_t.gInvalidSourceCid_c (*C enumerator*), 1545
l2caLeCbConnectionRequestResult_t.gLePsmNotSupported_c (*C enumerator*), 1544
l2caLeCbConnectionRequestResult_t.gNoResourcesAvailable_c (*C enumerator*), 1544
l2caLeCbConnectionRequestResult_t.gResponseTimeout_c (*C enumerator*), 1545
l2caLeCbConnectionRequestResult_t.gSourceCidAreadyAllocated_c (*C enumerator*), 1545
l2caLeCbConnectionRequestResult_t.gSuccessful_c (*C enumerator*), 1544
l2caLeCbConnectionRequestResult_t.gUnacceptableParameters_c (*C enumerator*), 1545
l2caLeCbControlCallback_t (*C type*), 1547
l2caLeCbDataCallback_t (*C type*), 1547
l2caLeCbDisconnection_t (*C type*), 1546
l2caLeCbDisconnection_tag (*C struct*), 1552, 1608
l2caLeCbDisconnection_tag.cId (*C var*), 1608
l2caLeCbDisconnection_tag.deviceId (*C var*), 1608
l2caLeCbError_t (*C type*), 1547
l2caLeCbError_tag (*C struct*), 1552, 1609
l2caLeCbError_tag.deviceId (*C var*), 1609
l2caLeCbError_tag.errorSource (*C var*), 1609
l2caLeCbError_tag.result (*C var*), 1609
l2caLeCbLocalCreditsNotification_t (*C type*), 1547
l2caLeCbLocalCreditsNotification_tag (*C struct*), 1552, 1609
l2caLeCbLocalCreditsNotification_tag.cId (*C var*), 1609
l2caLeCbLocalCreditsNotification_tag.deviceId (*C var*), 1609
l2caLeCbLocalCreditsNotification_tag.localCredits (*C var*), 1609
l2caLeCbLowPeerCredits_t (*C type*), 1547
l2caLeCbLowPeerCredits_tag (*C struct*), 1552, 1608
l2caLeCbLowPeerCredits_tag.cId (*C var*), 1608
l2caLeCbLowPeerCredits_tag.deviceId (*C var*), 1608
l2caLeCbNoPeerCredits_t (*C type*), 1547
l2caLeCbNoPeerCredits_tag (*C struct*), 1552, 1608
l2caLeCbNoPeerCredits_tag.cId (*C var*), 1608
l2caLeCbNoPeerCredits_tag.deviceId (*C var*), 1608
l2capControlMessage_t (*C type*), 1547
l2capControlMessage_tag (*C struct*), 1553, 1611
l2capControlMessage_tag.messageData (*C union*), 1553, 1611
l2capControlMessage_tag.messageData (*C var*), 1611
l2capControlMessage_tag.messageData.channelStatusNotification (*C var*), 1553, 1611
l2capControlMessage_tag.messageData.connectionComplete (*C var*), 1553, 1611
l2capControlMessage_tag.messageData.connectionRequest (*C var*), 1553, 1611
l2capControlMessage_tag.messageData.disconnection (*C var*), 1553, 1611

l2capControlMessage_tag.messageData.enhancedConnComplete (*C var*), 1553, 1611
l2capControlMessage_tag.messageData.enhancedConnRequest (*C var*), 1553, 1611
l2capControlMessage_tag.messageData.error (*C var*), 1553, 1611
l2capControlMessage_tag.messageData.handoverConnectionComplete (*C var*), 1553, 1611
l2capControlMessage_tag.messageData.localCreditsNotification (*C var*), 1553, 1611
l2capControlMessage_tag.messageData.lowPeerCredits (*C var*), 1553, 1611
l2capControlMessage_tag.messageData.noPeerCredits (*C var*), 1553, 1611
l2capControlMessage_tag.messageData.reconfigureRequest (*C var*), 1553, 1611
l2capControlMessage_tag.messageData.reconfigureResponse (*C var*), 1553, 1611
l2capControlMessage_tag.messageType (*C var*), 1611
l2capControlMessageType_t (*C enum*), 1546
l2capControlMessageType_t.gL2ca_ChannelStatusNotification_c (*C enumerator*), 1546
l2capControlMessageType_t.gL2ca_EnhancedReconfigureRequest_c (*C enumerator*), 1546
l2capControlMessageType_t.gL2ca_EnhancedReconfigureResponse_c (*C enumerator*), 1546
l2capControlMessageType_t.gL2ca_Error_c (*C enumerator*), 1546
l2capControlMessageType_t.gL2ca_HandoverConnectionComplete_c (*C enumerator*), 1546
l2capControlMessageType_t.gL2ca_LePsmConnectionComplete_c (*C enumerator*), 1546
l2capControlMessageType_t.gL2ca_LePsmConnectRequest_c (*C enumerator*), 1546
l2capControlMessageType_t.gL2ca_LePsmDisconnectNotification_c (*C enumerator*), 1546
l2capControlMessageType_t.gL2ca_LePsmEnhancedConnectionComplete_c (*C enumerator*), 1546
l2capControlMessageType_t.gL2ca_LePsmEnhancedConnectRequest_c (*C enumerator*), 1546
l2capControlMessageType_t.gL2ca_LocalCreditsNotification_c (*C enumerator*), 1546
l2capControlMessageType_t.gL2ca_LowPeerCredits_c (*C enumerator*), 1546
l2capControlMessageType_t.gL2ca_NoPeerCredits_c (*C enumerator*), 1546
l2capReconfigureResponse_t (*C enum*), 1545
l2capReconfigureResponse_t.gDestinationCidInvalid_c (*C enumerator*), 1545
l2capReconfigureResponse_t.gMtuReductionNotAllowed_c (*C enumerator*), 1545
l2capReconfigureResponse_t.gMultipleChannelMpsReductionNotAllowed_c (*C enumerator*), 1545
l2capReconfigureResponse_t.gReconfigurationSuccessful_c (*C enumerator*), 1545
l2capReconfigureResponse_t.gReconfigurationTimeout_c (*C enumerator*), 1545
l2capReconfigureResponse_t.gReconfReserved_c (*C enumerator*), 1546
l2capReconfigureResponse_t.gUnacceptableReconfParameters_c (*C enumerator*), 1545
leExtendedSupportedFeatures_tag (*C enum*), 1478
leExtendedSupportedFeatures_tag.gCsEnhancements1_c (*C enumerator*), 1478
leExtendedSupportedFeatures_tag.gLeMonitoringAdvertisers_c (*C enumerator*), 1478
leSupportedFeatures_t (*C type*), 1499
leSupportedFeatures_tag (*C enum*), 1477
leSupportedFeatures_tag.gAdvertisingCodingSelection_c (*C enumerator*), 1478
leSupportedFeatures_tag.gAdvertisingCodingSelectionHostSupport_c (*C enumerator*), 1478
leSupportedFeatures_tag.gLe2MbPhy_c (*C enumerator*), 1477
leSupportedFeatures_tag.gLeChannelSelAlg2_c (*C enumerator*), 1478
leSupportedFeatures_tag.gLeChannelSounding_c (*C enumerator*), 1478
leSupportedFeatures_tag.gLeChannelSoundingHostSupport_c (*C enumerator*), 1478
leSupportedFeatures_tag.gLeCodedPhy_c (*C enumerator*), 1477
leSupportedFeatures_tag.gLeConnectionParametersRequestProcedure_c (*C enumerator*), 1477
leSupportedFeatures_tag.gLeConnectionSubrating_c (*C enumerator*), 1478
leSupportedFeatures_tag.gLeConnectionSubratingHostSupport_c (*C enumerator*), 1478
leSupportedFeatures_tag.gLeDataPacketLengthExtension_c (*C enumerator*), 1477
leSupportedFeatures_tag.gLeDecisionBasedAdvertisingFiltering_c (*C enumerator*), 1478
leSupportedFeatures_tag.gLeEncryption_c (*C enumerator*), 1477
leSupportedFeatures_tag.gLeExtendedAdv_c (*C enumerator*), 1477
leSupportedFeatures_tag.gLeExtendedFeatures_c (*C enumerator*), 1478
leSupportedFeatures_tag.gLeExtendedRejectIndication_c (*C enumerator*), 1477
leSupportedFeatures_tag.gLeExtendedScannerFilterPolicies_c (*C enumerator*), 1477
leSupportedFeatures_tag.gLeLlPrivacy_c (*C enumerator*), 1477
leSupportedFeatures_tag.gLeMinNumOfUsedChanProcedure_c (*C enumerator*), 1478
leSupportedFeatures_tag.gLePathLossMonitoring_c (*C enumerator*), 1478
leSupportedFeatures_tag.gLePawrAdvertiser_c (*C enumerator*), 1478

leSupportedFeatures_tag.gLePawrScanner_c (*C enumerator*), 1478
leSupportedFeatures_tag.gLePeriodicAdv_c (*C enumerator*), 1477
leSupportedFeatures_tag.gLePeriodicAdvSyncTransferReceiver_c (*C enumerator*), 1478
leSupportedFeatures_tag.gLePeriodicAdvSyncTransferSender_c (*C enumerator*), 1478
leSupportedFeatures_tag.gLePeripheralInitiatedFeaturesExchange_c (*C enumerator*), 1477
leSupportedFeatures_tag.gLePing_c (*C enumerator*), 1477
leSupportedFeatures_tag.gLePowerClass1_c (*C enumerator*), 1478
leSupportedFeatures_tag.gLePowerControlRequest1_c (*C enumerator*), 1478
leSupportedFeatures_tag.gLePowerControlRequest2_c (*C enumerator*), 1478
leSupportedFeatures_tag.gLeStableModulationIdxRx_c (*C enumerator*), 1477
leSupportedFeatures_tag.gLeStableModulationIdxTx_c (*C enumerator*), 1477
LL_PHY_1M (*C macro*), 1508
LL_PHY_2M (*C macro*), 1508
LL_PHY_S2 (*C macro*), 1508
LL_PHY_S8 (*C macro*), 1508

M

mServerServiceChangedCCCDHandle (*C var*), 1449
mServerServiceChangedCharHandle (*C var*), 1449

P

PACKED_STRUCT (*C macro*), 1518
periodicAdvSetInfoTransferEvent_t (*C type*), 1500
periodicAdvSetInfoTransferEvent_tag (*C struct*), 1523, 1557
periodicAdvSetInfoTransferEvent_tag.deviceId (*C var*), 1558
periodicAdvSetInfoTransferEvent_tag.status (*C var*), 1558
periodicAdvSetSyncTransferParamsEvent_t (*C type*), 1500
periodicAdvSetSyncTransferParamsEvent_tag (*C struct*), 1523, 1558
periodicAdvSetSyncTransferParamsEvent_tag.deviceId (*C var*), 1558
periodicAdvSetSyncTransferParamsEvent_tag.status (*C var*), 1558
periodicAdvSyncTransferEvent_t (*C type*), 1500
periodicAdvSyncTransferEvent_tag (*C struct*), 1523, 1557
periodicAdvSyncTransferEvent_tag.deviceId (*C var*), 1557
periodicAdvSyncTransferEvent_tag.status (*C var*), 1557
pProcedureData (*C var*), 1414
procDataStruct_t (*C struct*), 1411, 1603
procDataStruct_t.array (*C union*), 1412, 1603
procDataStruct_t.array (*C var*), 1412, 1603
procDataStruct_t.array.aBytes (*C var*), 1412, 1604
procDataStruct_t.array.aChars (*C var*), 1412, 1604
procDataStruct_t.array.aDescriptors (*C var*), 1412, 1604
procDataStruct_t.array.aHandles (*C var*), 1412, 1604
procDataStruct_t.array.aServices (*C var*), 1412, 1604
procDataStruct_t.bAllocatedArray (*C var*), 1412, 1603
procDataStruct_t.charUuid (*C var*), 1411, 1603
procDataStruct_t.charUuidType (*C var*), 1411, 1603
procDataStruct_t.index (*C var*), 1411, 1603
procDataStruct_t.max (*C var*), 1411, 1603
procDataStruct_t.pOutActualCount (*C union*), 1412, 1603
procDataStruct_t.pOutActualCount (*C var*), 1412, 1603
procDataStruct_t.pOutActualCount.pCount8b (*C var*), 1412, 1603
procDataStruct_t.pOutActualCount.pCount16b (*C var*), 1412, 1603
procDataStruct_t.reliableLongWrite (*C var*), 1412, 1603
procDataStruct_t.reqParams (*C union*), 1412, 1604
procDataStruct_t.reqParams (*C var*), 1412, 1603
procDataStruct_t.reqParams.ewParams (*C var*), 1413, 1604
procDataStruct_t.reqParams.fbtvParams (*C var*), 1413, 1604
procDataStruct_t.reqParams.fiParams (*C var*), 1413, 1604

procDataStruct_t.reqParams.pwParams (*C var*), 1413, 1604
procDataStruct_t.reqParams.rbgtParams (*C var*), 1413, 1604
procDataStruct_t.reqParams.rbParams (*C var*), 1413, 1604
procDataStruct_t.reqParams.rbtParams (*C var*), 1413, 1604
procDataStruct_t.reqParams.rmParams (*C var*), 1413, 1604
procDataStruct_t.reqParams.rParams (*C var*), 1413, 1604
procDataStruct_t.reqParams.wParams (*C var*), 1413, 1604
procStatus_t (*C struct*), 1411, 1602
procStatus_t.isOngoing (*C var*), 1411, 1603
procStatus_t.ongoingProcedurePhase (*C var*), 1411, 1603
procStatus_t.ongoingProcedureType (*C var*), 1411, 1603

S

STATIC (*C macro*), 1505

U

Uint16toUint8 (*C macro*), 1519
Uint32toUint8 (*C macro*), 1519
Uint32toUint16 (*C macro*), 1519
Utils_BeExtractFourByteValue (*C macro*), 1519
Utils_BeExtractThreeByteValue (*C macro*), 1519
Utils_BeExtractTwoByteValue (*C macro*), 1519
Utils_BePackFourByteValue (*C macro*), 1519
Utils_BePackThreeByteValue (*C macro*), 1519
Utils_BePackTwoByteValue (*C macro*), 1519
Utils_Copy8 (*C macro*), 1519
Utils_Copy16 (*C macro*), 1519
Utils_Copy32 (*C macro*), 1519
Utils_Copy64 (*C macro*), 1519
Utils_ExtractEightByteValue (*C macro*), 1519
Utils_ExtractFourByteValue (*C macro*), 1519
Utils_ExtractThreeByteValue (*C macro*), 1518
Utils_ExtractTwoByteValue (*C macro*), 1518
Utils_PackEightByteValue (*C macro*), 1519
Utils_PackFourByteValue (*C macro*), 1519
Utils_PackThreeByteValue (*C macro*), 1519
Utils_PackTwoByteValue (*C macro*), 1519
Utils_RevertByteArray (*C macro*), 1519
Uuid16 (*C macro*), 1518
Uuid32 (*C macro*), 1518
UuidArray (*C macro*), 1518

V

vendorSpecificDbgEvent_t (*C type*), 1501
vendorSpecificDbgEvent_tag (*C struct*), 1534, 1566
vendorSpecificDbgEvent_tag.dataSize (*C var*), 1566
vendorSpecificDbgEvent_tag.pData (*C var*), 1566
vendorUnitaryTestEvent_t (*C type*), 1501
vendorUnitaryTestEvent_tag (*C struct*), 1534, 1566
vendorUnitaryTestEvent_tag.aParam (*C var*), 1566
vendorUnitaryTestEvent_tag.paramLength (*C var*), 1566
vendorUnitaryTestEvent_tag.status (*C var*), 1566