MCUXpresso SDK Documentation

Release 26.09.00

NXP
Sep 29, 2026

Table of contents

1 MIMXRT1024-EVK **3**
- 1.1 Overview . . . 3
- 1.2 SDK Introduction . . . 3
  - 1.2.1 MCUXpresso Software Development Kit (SDK) . . . 3
- 1.3 Getting Started with MCUXpresso SDK Package . . . 7
  - 1.3.1 Getting Started with Package . . . 7
- 1.4 Getting Started with MCUXpresso SDK GitHub . . . 23
  - 1.4.1 Getting Started with MCUXpresso SDK Repository . . . 23
- 1.5 Release Notes . . . 50
  - 1.5.1 MCUXpresso SDK Release Notes . . . 50
- 1.6 ChangeLog . . . 55
  - 1.6.1 MCUXpresso SDK Changelog . . . 55
- 1.7 Driver API Reference Manual . . . 171
- 1.8 Middleware Documentation . . . 171
  - 1.8.1 FreeMASTER . . . 171
  - 1.8.2 NXP Wi-Fi . . . 171
  - 1.8.3 FreeRTOS . . . 172
  - 1.8.4 lwIP . . . 172
  - 1.8.5 File systemFatfs . . . 172

2 MIMXRT1024 **173**
- 2.1 ADC: 12-bit Analog to Digital Converter Driver . . . 173
- 2.2 ADC_ETC: ADC External Trigger Control . . . 182
- 2.3 AIPSTZ: AHB to IP Bridge . . . 186
- 2.4 AOI: Crossbar AND/OR/INVERT Driver . . . 190
- 2.5 BEE: Bus Encryption Engine . . . 193
- 2.6 CACHE: ARMV7-M7 CACHE Memory Controller . . . 199
- 2.7 Clock Driver . . . 201
- 2.8 CMP: Analog Comparator Driver . . . 245
- 2.9 DCDC: DCDC Converter . . . 250
- 2.10 DCP: Data Co-Processor . . . 261
- 2.11 DCP AES blocking driver . . . 266
- 2.12 DCP HASH driver . . . 268
- 2.13 DCP AES non-blocking driver . . . 270
- 2.14 DMAMUX: Direct Memory Access Multiplexer Driver . . . 272
- 2.15 eDMA: Enhanced Direct Memory Access (eDMA) Controller Driver . . . 273
- 2.16 ENC: Quadrature Encoder/Decoder . . . 295
- 2.17 ENET: Ethernet MAC Driver . . . 307
- 2.18 EWM: External Watchdog Monitor Driver . . . 342
- 2.19 FlexCAN: Flex Controller Area Network Driver . . . 345
- 2.20 FlexCAN Driver . . . 345
- 2.21 FlexIO: FlexIO Driver . . . 394
- 2.22 FlexIO Driver . . . 394
- 2.23 FlexIO eDMA I2S Driver . . . 414
- 2.24 FlexIO eDMA SPI Driver . . . 417
- 2.25 FlexIO eDMA UART Driver . . . 421

2.26 FlexIO I2C Master Driver . . . 424
2.27 FlexIO I2S Driver . . . 433
2.28 FlexIO SPI Driver . . . 445
2.29 FlexIO UART Driver . . . 459
2.30 FlexPWM: Enhanced Flex Pulse Width Modulator . . . 471
2.31 FLEXRAM: on-chip RAM manager . . . 530
2.32 FLEXSPI: Flexible Serial Peripheral Interface Driver . . . 539
2.33 FLEXSPI eDMA Driver . . . 559
2.34 GPC: General Power Controller Driver . . . 561
2.35 GPIO: General-Purpose Input/Output Driver . . . 563
2.36 GPT: General Purpose Timer . . . 568
2.37 IOMUXC: IOMUX Controller . . . 576
2.38 KPP: KeyPad Port Driver . . . 606
2.39 Common Driver . . . 608
2.40 LPI2C: Low Power Inter-Integrated Circuit Driver . . . 625
2.41 LPI2C Master Driver . . . 626
2.42 LPI2C Master DMA Driver . . . 642
2.43 LPI2C Slave Driver . . . 644
2.44 LPSPI: Low Power Serial Peripheral Interface . . . 655
2.45 LPSPI Peripheral driver . . . 656
2.46 LPSPI eDMA Driver . . . 680
2.47 LPUART: Low Power Universal Asynchronous Receiver/Transmitter Driver . . . 688
2.48 LPUART Driver . . . 688
2.49 LPUART eDMA Driver . . . 709
2.50 OCOTP: On Chip One-Time Programmable controller. . . . 712
2.51 PIT: Periodic Interrupt Timer . . . 716
2.52 PMU: Power Management Unit . . . 720
2.53 PWM: Pulse Width Modulator . . . 727
2.54 QTMR: Quad Timer Driver . . . 756
2.55 RTWDOG: 32-bit Watchdog Timer . . . 768
2.56 SAI: Serial Audio Interface . . . 775
2.57 SAI Driver . . . 775
2.58 SAI EDMA Driver . . . 804
2.59 SEMC: Smart External DRAM Controller Driver . . . 812
2.60 SNVS: Secure Non-Volatile Storage . . . 842
2.61 Secure Non-Volatile Storage High-Power . . . 842
2.62 Secure Non-Volatile Storage Low-Power . . . 851
2.63 SPDIF: Sony/Philips Digital Interface . . . 862
2.64 SPDIF eDMA Driver . . . 877
2.65 SRC: System Reset Controller Driver . . . 881
2.66 TEMPMON: Temperature Monitor Module . . . 887
2.67 TRNG: True Random Number Generator . . . 889
2.68 USDHC: Ultra Secured Digital Host Controller Driver . . . 894
2.69 WDOG: Watchdog Timer Driver . . . 929
2.70 XBARA: Inter-Peripheral Crossbar Switch . . . 933
2.71 XBARB: Inter-Peripheral Crossbar Switch . . . 937

3 Middleware 939
3.1 Connectivity . . . 939
3.1.1 lwIP . . . 939
3.2 File System . . . 953
3.2.1 FatFs . . . 953
3.3 Motor Control . . . 955
3.3.1 FreeMASTER . . . 955
3.4 Multicore SDK . . . 993
3.4.1 Quick links . . . 993
3.4.2 Components . . . 994
3.5 Wireless . . . 995

3.5.1 NXP Wireless Framework and Stacks . . . 995

4 RTOS **1057**
4.1 FreeRTOS . . . 1057
4.1.1 FreeRTOS kernel . . . 1057
4.1.2 FreeRTOS drivers . . . 1061
4.1.3 backoffalgorithm . . . 1061
4.1.4 corehttp . . . 1061
4.1.5 corejson . . . 1061
4.1.6 coremqtt . . . 1061
4.1.7 corepkcs11 . . . 1062
4.1.8 freertos-plus-tcp . . . 1062

5 Downloads **1063**
5.1 boards . . . 1063
5.2 develop . . . 1063

Index **1065**

This documentation contains information specific to the evkmimxrt1024 board.

Chapter 1

MIMXRT1024-EVK

1.1 Overview

The NXP MIMXRT1024-EVK is a development board for the i.MX MIMXRT1024 500 MHz 32-bit ARM Cortex-M7 MCUs. The i.MX RT1024 EVK is a 2-layer low-cost through-hole USB-powered PCB. At its heart lies the i.MX RT1024 crossover MCU in LQFP144 package, featuring NXPs advanced implementation of the Arm Cortex-M7 core. This core operates at speeds up to 500 MHz to provide high CPU performance and excellent real-time response.

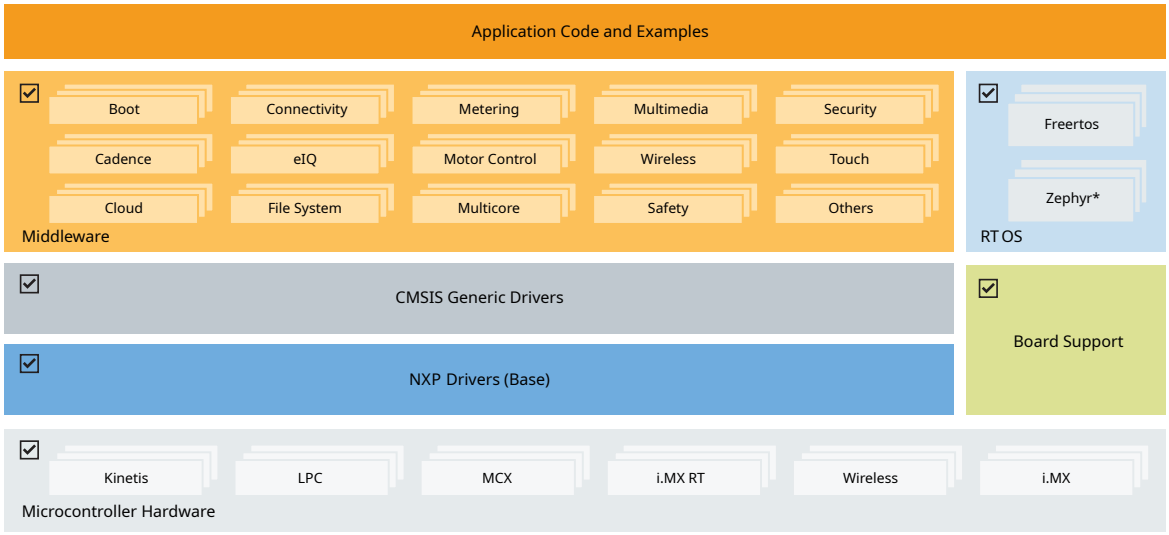
MCU device and part on board is shown below:

- Device: MIMXRT1024
- PartNumber: MIMXRT1024DAG5A

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 MCUXpresso SDK Package

1.3.1 Getting Started with Package

MCUXpresso SDK board support package folders

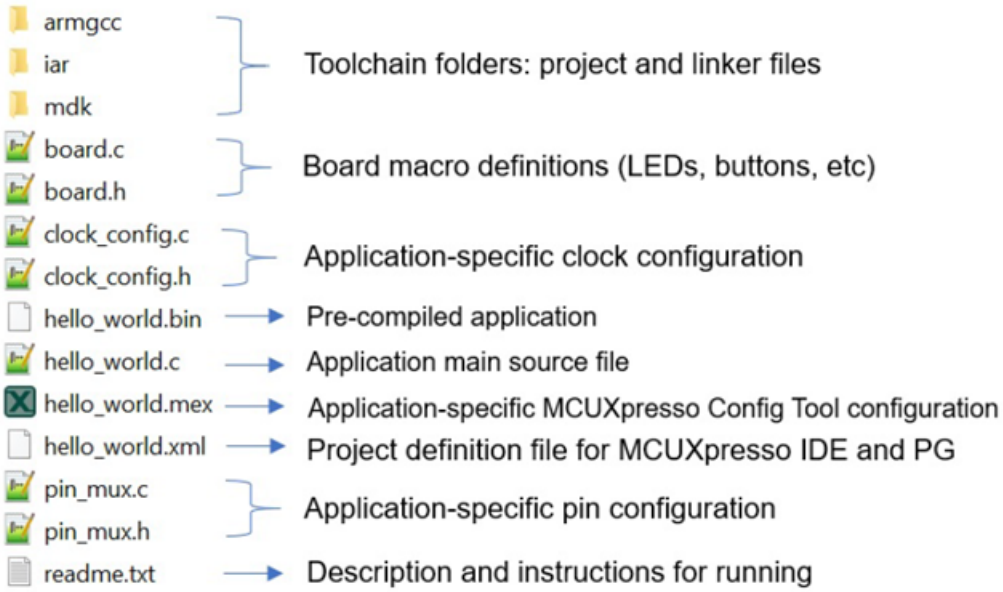
MCUXpresso SDK board support package provides example applications for NXP development and evaluation boards for Arm Cortex-M cores including Freedom, Tower System, and LPCXpresso boards. Board support packages are found inside the top-level boards folder and each supported board has its own folder (an MCUXpresso SDK package can support multiple boards). Within each <board_name> folder, there are various subfolders to classify the type of examples it contains. These include (but are not limited to):

- demo_apps: Full-featured applications that highlight key functionality and use cases of the target MCU. These applications typically use multiple MCU peripherals and may leverage stacks and middleware.
- driver_examples: Simple applications that show how to use the MCUXpresso SDK's peripheral drivers for a single use case. These applications typically only use a single peripheral but there are cases where multiple peripherals are used (for example, SPI conversion using DMA).
- rtos_examples: Basic FreeRTOSM OS examples that show the use of various RTOS objects (semaphores, queues, and so on) and interfaces with the MCUXpresso SDK's RTOS drivers
- wireless_examples: Applications that use the Zigbee and OpenThread stacks.

Example application structure This section describes how the various types of example applications interact with the other components in the MCUXpresso SDK. To get a comprehensive understanding of all MCUXpresso SDK components and folder structure, see *MCUXpresso SDK API Reference Manual*.

Each <board_name> folder in the boards directory contains a comprehensive set of examples that are relevant to that specific piece of hardware. Although we use the hello_world example (part of the demo_apps folder), the same general rules apply to any type of example in the <board_name> folder.

In the hello_world application folder you see the following contents:



All files in the application folder are specific to that example, so it is easy to copy and paste an existing example to start developing a custom application based on a project provided in the MCUXpresso SDK.

Locating example application source files When opening an example application in any of the supported IDEs, various source files are referenced. The MCUXpresso SDK devices folder is the central component to all example applications. It means that the examples reference the same source files and, if one of these files is modified, it could potentially impact the behavior of other examples.

The main areas of the MCUXpresso SDK tree used in all example applications are:

- devices/<device_name>: The device's CMSIS header file, MCUXpresso SDK feature file, and a few other files
- devices/<device_name>/drivers: All of the peripheral drivers for your specific MCU
- devices/<device_name>/<tool_name>: Toolchain-specific startup code, including vector table definitions
- devices/<device_name>/utilities: Items such as the debug console that are used by many of the example applications

For examples containing an RTOS, there are references to the appropriate source code. RTOSes are in the rtos folder. The core files of each of these are shared, so modifying one could have potential impacts on other projects that depend on that file.

Parent topic:*MCUXpresso SDK board support package folders*

Run a demo application using IAR

Build an example application Do the following steps to build the hello_world example application.

1. Open the desired demo application workspace. Most example application workspace files can be located using the following path:

   ```
   <install_dir>/boards/<board_name>/<example_type>/<application_name>/iar
   ```

   Using the EVK-MIMXRT1024 hardware platform as an example, the hello_world workspace is located in:

   ```
   <install_dir>/boards/evkmimxrt1024/demo_apps/hello_world/iar/hello_world.eww
   ```

   Other example applications may have additional folders in their path.

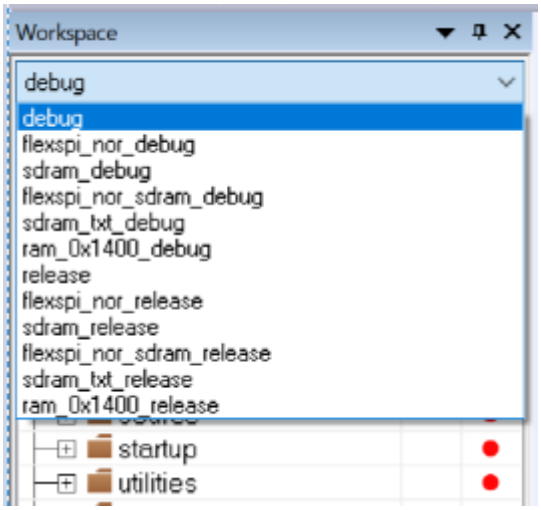
2. Select the desired build target from the drop-down menu.

   There are twelve project configurations (build targets) supported for most MCUXpresso SDK projects:

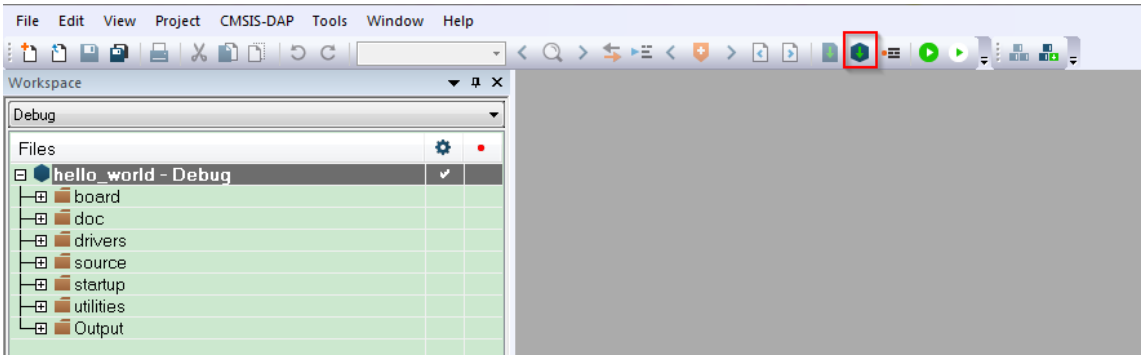
   - Debug – Compiler optimization is set to low, and debug information is generated for the executable. The linker file is RAM linker, where text and data section is put in internal TCM.
   - Release – Compiler optimization is set to high, and debug information is not generated. The linker file is RAM linker, where text and data section is put in internal TCM.
   - ram_0x1400_debug – Project configuration is same as the debug target. The linker file is RAM_0x1400 linker, where text is put in ITCM with offset 0x1400 and data put in DTCM.

- ram_0x1400_release – Project configuration is same as the release target. The linker file is RAM_0x1400 linker, where text is put in ITCM with offset 0x1400 and data put in DTCM.
- sdram_debug – Project configuration is same as the debug target. The linker file is SDRAM linker, where text is put in internal TCM and data put in SDRAM.
- sdram_release – Project configuration is same as the release target. The linker file is SDRAM linker, where text is put in internal TCM and data put in SDRAM.
- sdram_txt_debug – Project configuration is same as the debug target. The linker file is SDRAM_txt linker, where text is put in SDRAM and data put in OCRAM.
- sdram_txt_release – Project configuration is same as the release target. The linker file is SDRAM_txt linker, where text is put in SDRAM and data put in OCRAM.
- flexspi_nor_debug – Project configuration is same as the debug target. The linker file is flexspi_nor linker, where text is put in flash and data put in TCM.
- flexspi_nor_release – Project configuration is same as the release target. The linker file is flexspi_nor linker, where text is put in flash and data put in TCM.
- flexspi_nor_sdram_release - Project configuration is same as the release target. The linker file is flexspi_nor_sdram linker, where text is put in flash and data put in SDRAM.
- flexspi_nor_sdram_debug – Project configuration is same as the debug target. The linker file is flexspi_nor_sdram linker, where text is put in flash and data put in SDRAM. For some examples need large data memory, only sdram_debug and sdram_release targets are supported.

For this example, select **hello_world** – **debug**.



3. To build the demo application, click **Make**, highlighted in red in *Figure 2*.

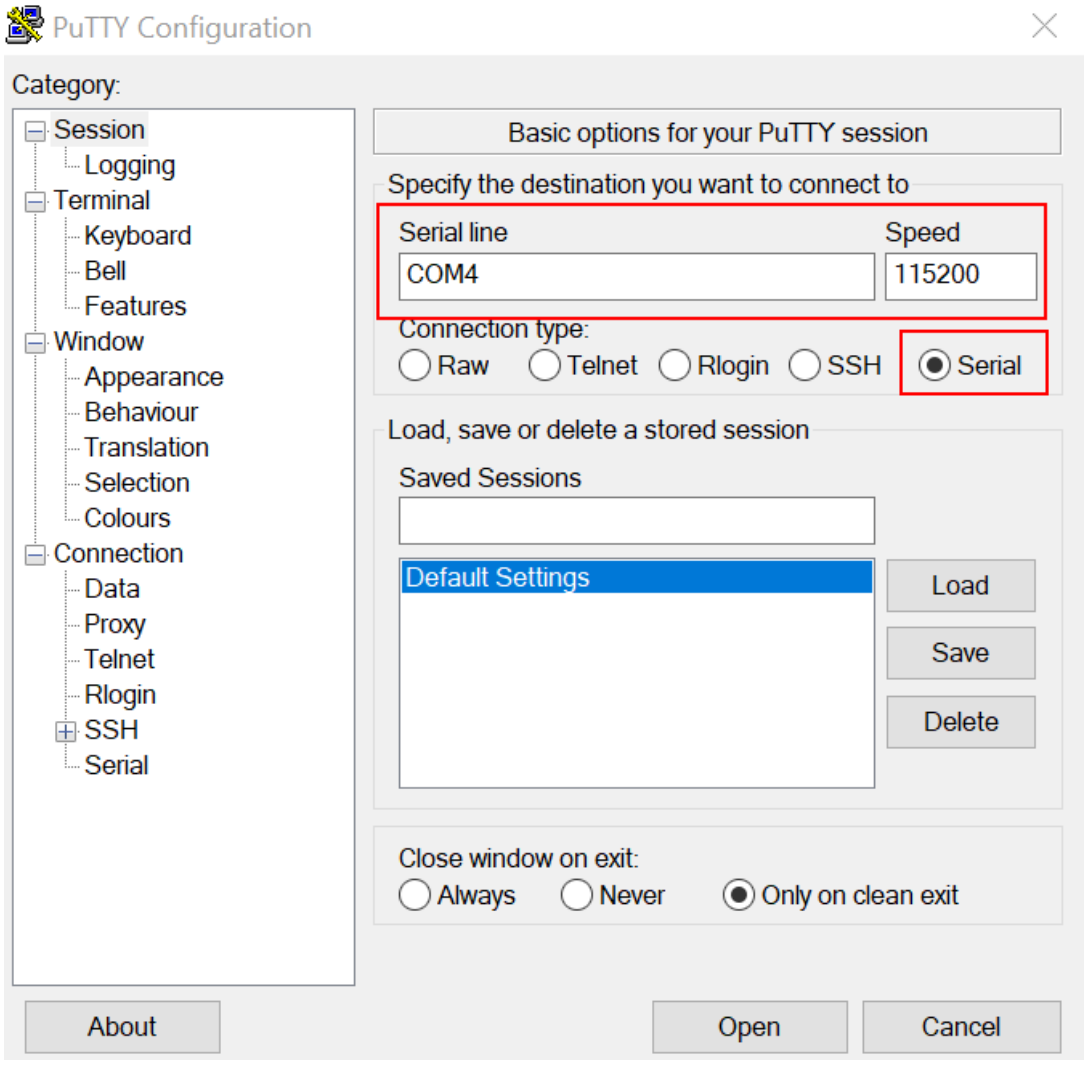


4. The build completes without errors.

Parent topic:*Run a demo application using IAR*

Run an example application To download and run the application, perform these steps:

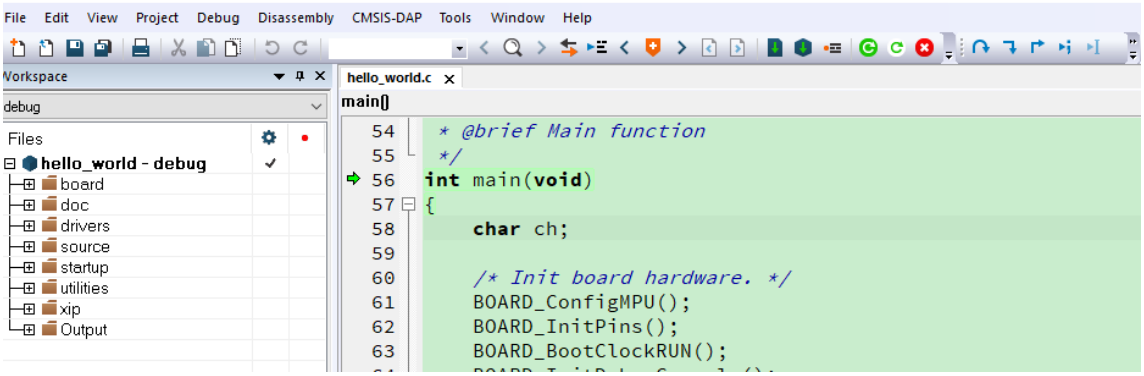
1. This board supports the CMSIS-DAP/mbed/DAPLink debug probe by default. Visit os.mbed.com/handbook/Windows-serial-configuration and follow the instructions to install the Windows operating system serial driver. If running on Linux OS, this step is not required.
2. Connect the development platform to your PC via USB cable. Connect the USB cable to J41 and make sure SW7[1:4] is **0010b**.
3. Open the terminal application on the PC, such as PuTTY or TeraTerm, and connect to the debug COM port (to determine the COM port number, see *How to determine COM port*). Configure the terminal with these settings:
    1. 115200 or 9600 baud rate, depending on your board (reference BOARD_DEBUG_UART_BAUDRATE variable in the board.h file)
    2. No parity
    3. 8 data bits
    4. 1 stop bit



4. In IAR, click the **Download and Debug** button to download the application to the target.

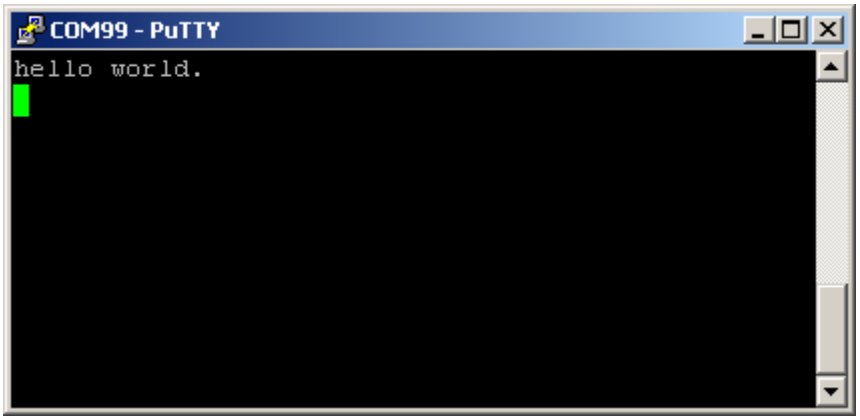
When using jlink to debug debug/release targets on IAR, an extra option is specified in debugger settings. Check **Debugger** > **Extra Options** > **Command-line** options and fill the following contents: --jlink_script_file=$PROJ_DIR$/../evkmimxrt1024_sdram_init.jlinkscript.

5. The application is then downloaded to the target and automatically runs to the main() function.



6. Run the code by clicking the **Go** button to start the application.

7. The hello_world application is now running and a banner is displayed on the terminal. If this is not true, check your terminal settings and connections.



Parent topic:*Run a demo application using IAR*

Run a demo using Keil MDK/µVision

This section describes the steps required to build, run, and debug example applications provided in the MCUXpresso SDK.

Install CMSIS device pack After the MDK tools are installed, Cortex Microcontroller Software Interface Standard (CMSIS) device packs must be installed to fully support the device from a debug perspective. These packs include things such as memory map information, register definitions, and flash programming algorithms. Follow these steps to install the MIMXRT102x CMSIS pack.

1. Download the MIMXRT102x packs .
2. After downloading the DFP, double click to install it.

Parent topic:*Run a demo using Keil MDK/µVision*

Build an example application

1. Open the desired example application workspace in:

   ```
   <install_dir>/boards/<board_name>/*<example\_type\>*/<application_name>/mdk
   ```

   The workspace file is named as <demo_name>.uvmpw. For this specific example, the actual path is:

   ```
   <install_dir>/boards/evkmimxrt1024/demo_apps/hello_world/mdk/hello_world.uvmpw
   ```

2. To build the demo project, select **Rebuild**, highlighted in red.

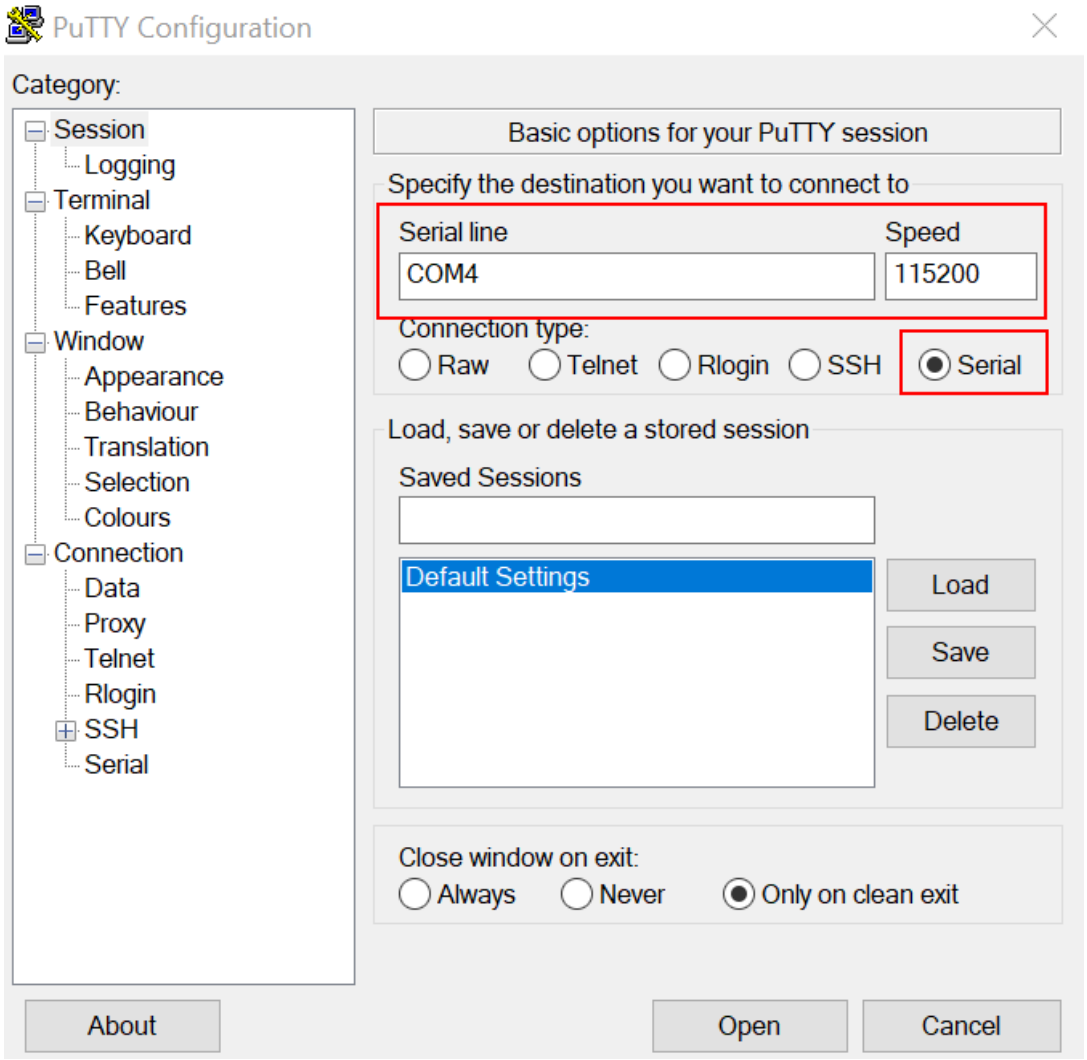
3. The build completes without errors.

Parent topic:*Run a demo using Keil MDK/µVision*

Run an example application

To download and run the application, perform these steps:

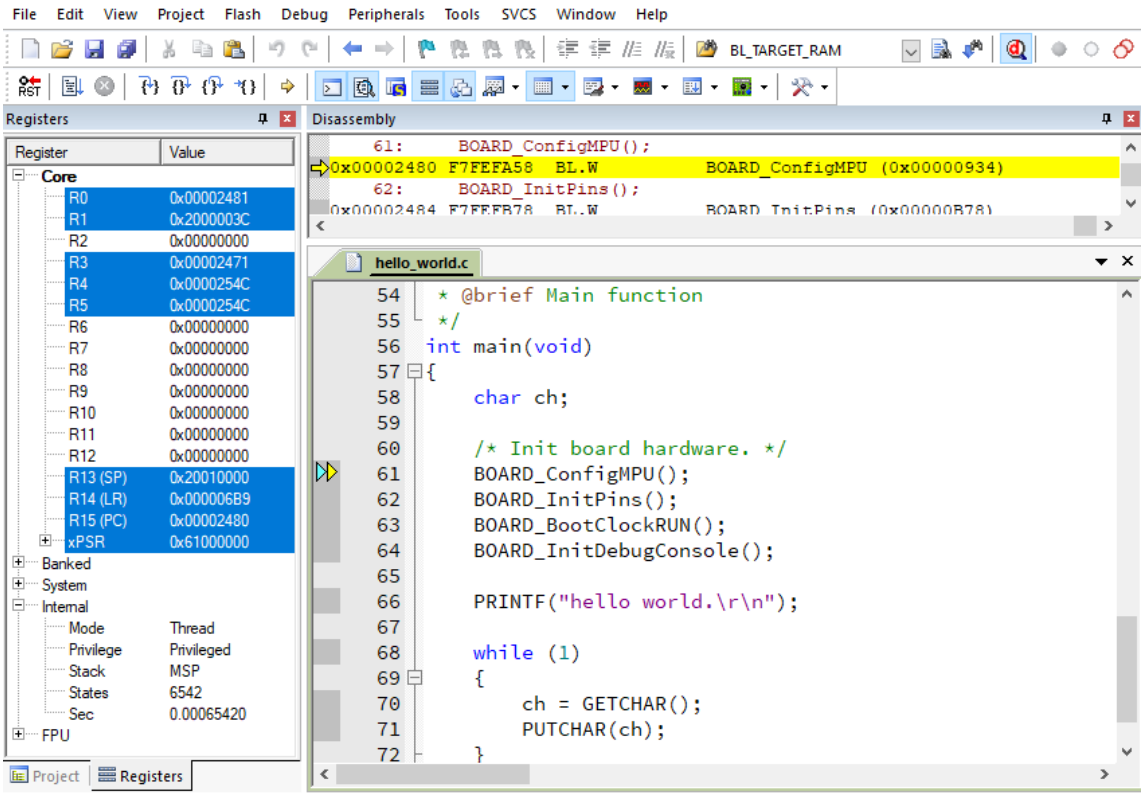
1. This board supports the CMSIS-DAP/mbed/DAPLink debug probe by default. Visit os.mbed.com/handbook/Windows-serial-configuration and follow the instructions to install the Windows operating system serial driver. If running on Linux OS, this step is not required.
2. Connect the development platform to your PC via USB cable.
3. Open the terminal application on the PC, such as PuTTY or TeraTerm, and connect to the debug serial port number (to determine the COM port number, see *How to determine COM port*). Configure the terminal with these settings:
   1. 115200 or 9600 baud rate, depending on your board (reference BOARD_DEBUG_UART_BAUDRATE variable in the board.h file)
   2. No parity
   3. 8 data bits



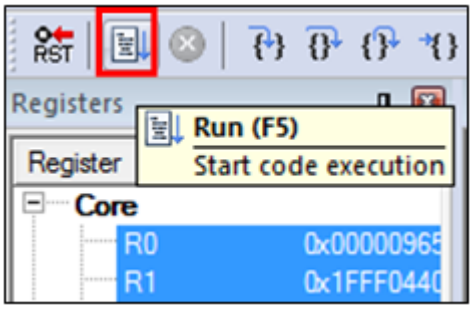
4. 1 stop bit

4. To debug the application, click **load** (or press the F8 key). Then, click the **Start/Stop Debug Session** button, highlighted in red in Figure 2. If using **J-Link** as the debugger, click **Project option** >**Debug** >**Settings** >**Debug** >**Port**, and select **SW**.

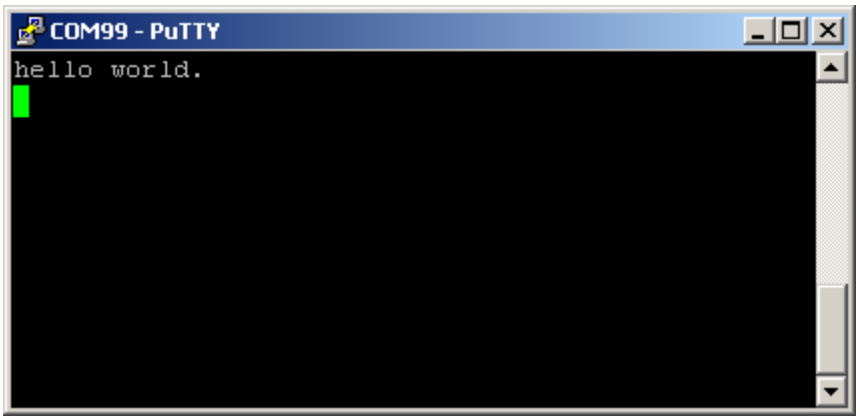
   Note: When debugging with jlink, it expects one jlinkscript file named JLinkSettings.JLinkScript in the folder where the uVision project files are located. For details, see SEGGER Wiki. For the contents in this JlinkSettings.JLinkScript, use contents in evkmimxrt1020_sdram_init.jlinkscript.



5. Run the code by clicking **Run** to start the application, as shown in Figure 3.



The hello_world application is now running and a banner is displayed on the terminal, as shown in Figure 4. If this is not true, check your terminal settings and connections.



Parent topic:*Run a demo using Keil MDK/µVision*

Run a demo using ARMGCC / VSCODE

This section describes the steps to run an example application from the SDK archive using the ARMGCC / VSCODE toolchain.

Refer to the *running a demo using MCUXpresso VSC* section for detailed instructions on setting up and configuring your project in Visual Studio Code.

Refer to the *CLI* section for detailed instructions on building and running your project from the command line.

Run a demo using MCUXpresso IDE

Note:

Ensure that the MCUXpresso IDE toolchain is included when generating the MCUXpresso SDK Package.

MCUXpresso IDE is not supported in this release.

This section describes the steps required to configure MCUXpresso IDE to build, run, and debug example applications. The hello_world demo application targeted for the MIMXRT1024-EVK platform is used as an example, though these steps can be applied to any example application in the MCUXpresso SDK.

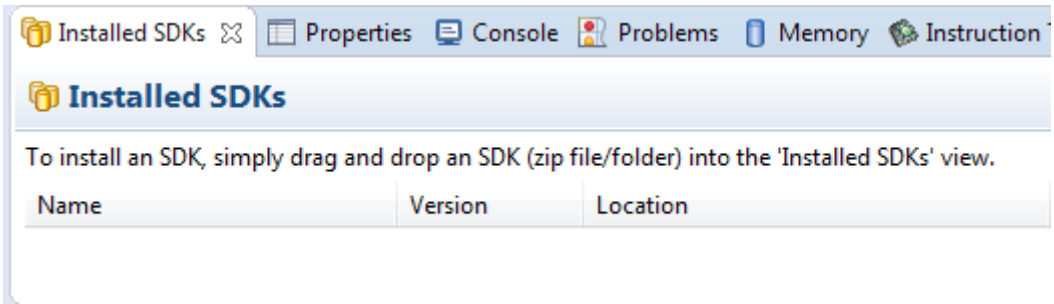
Note: By default, three macros, XIP_EXTERNAL_FLASH=1, XIP_BOOT_HEADER_ENABLE=1, and XIP_BOOT_HEADER_DCD_ENABLE=1, are set in the project. If you do not use Board_Flash in the project, these macros should be removed or set value to **0** in project settings.

Select the workspace location Every time MCUXpresso IDE launches, it prompts the user to select a workspace location. MCUXpresso IDE is built on top of Eclipse which uses workspace to store information about its current configuration, and in some use cases, source files for the projects are in the workspace. The location of the workspace can be anywhere, but it is recommended that the workspace be located outside the MCUXpresso SDK tree.

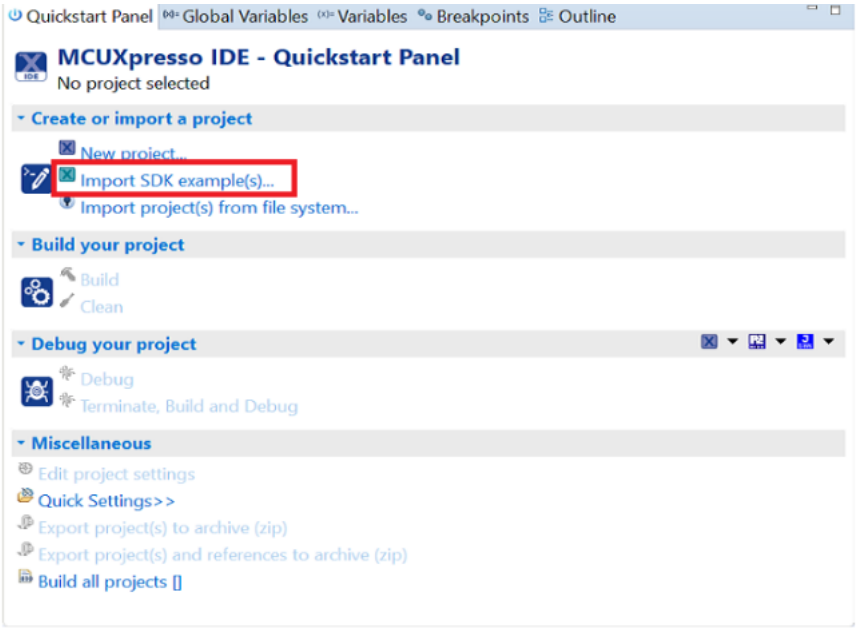
Parent topic:*Run a demo using MCUXpresso IDE*

Build an example application To build an example application, follow these steps.

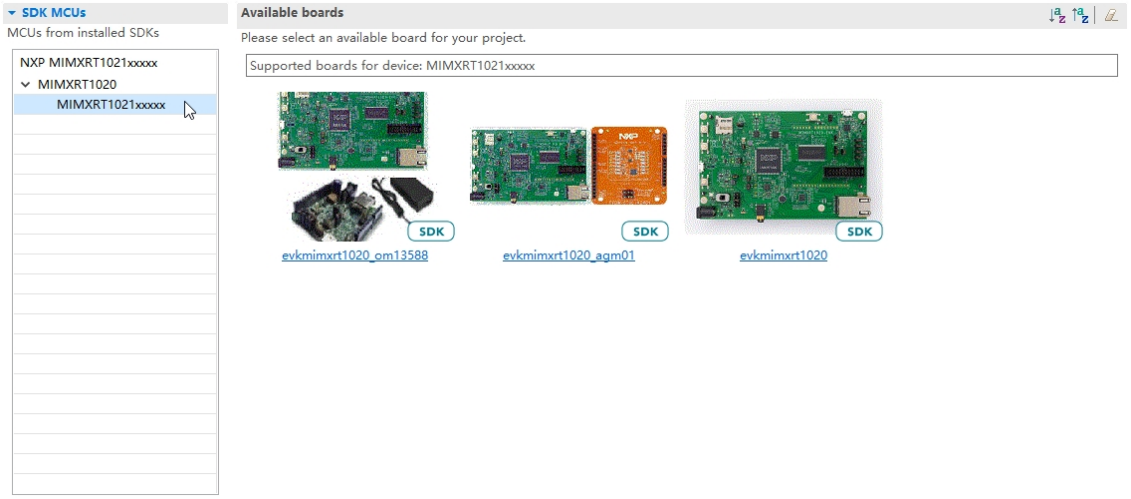
1. Drag and drop the SDK zip file into the **Installed SDKs** view to install the MCUXpresso SDK. In the window that appears, click **OK** and wait until the import has finished.



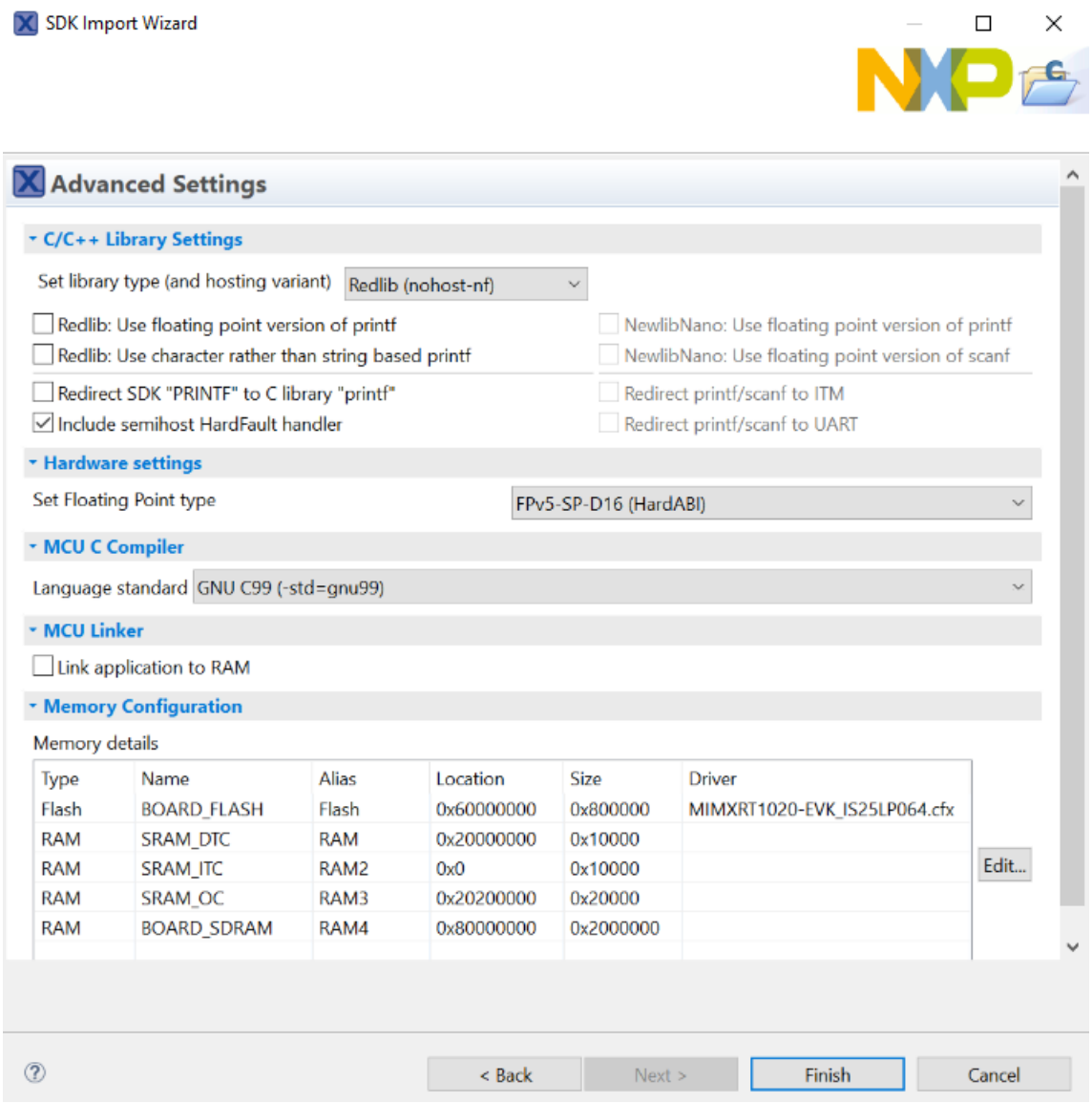
2. On the **Quickstart Panel**, click **Import SDK example(s)...**, as shown in *Figure 2*.



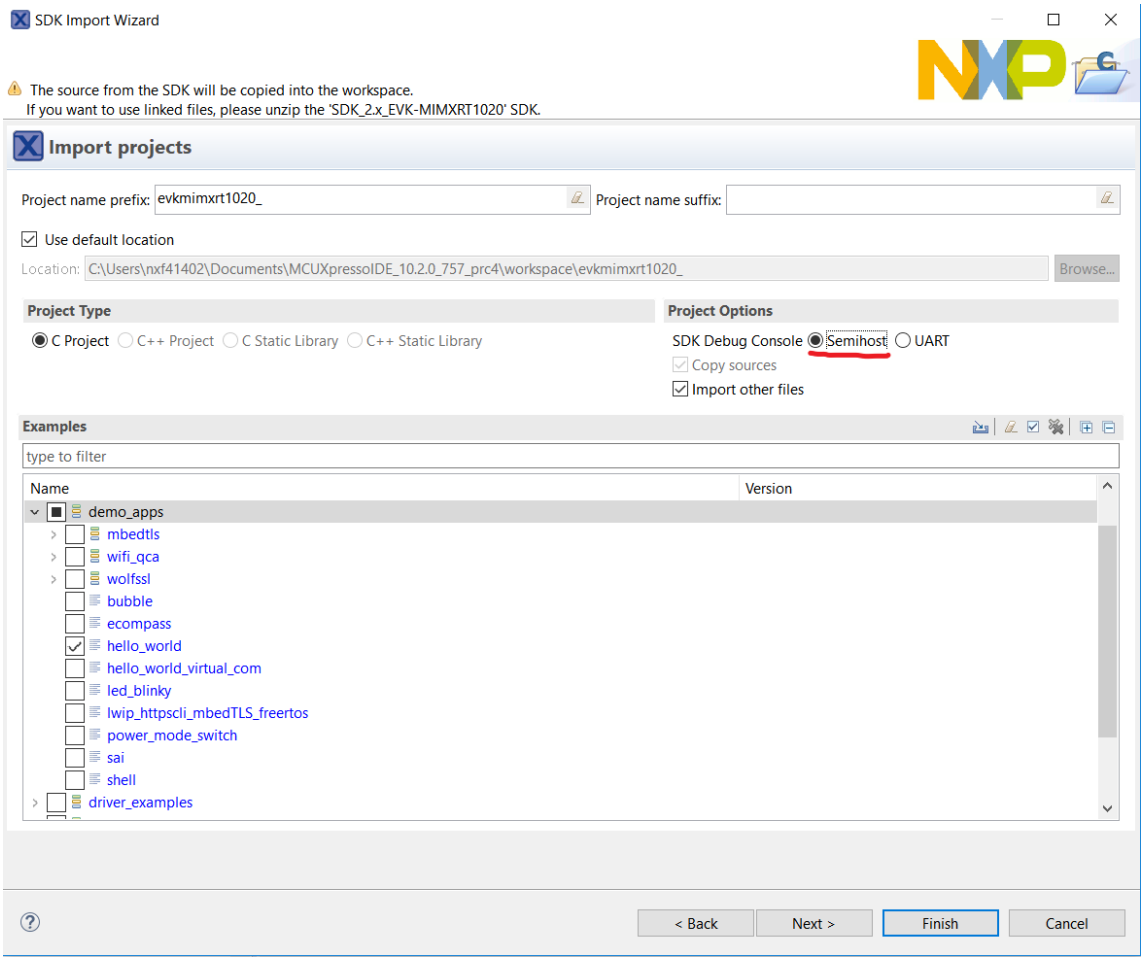
3. In the window that appears, expand the **MIMXRT1024**folder and select MIMXRT1024xxxxx. Then, select evkmimxrt1024 and click **Next**, as shown in *Figure 3*.



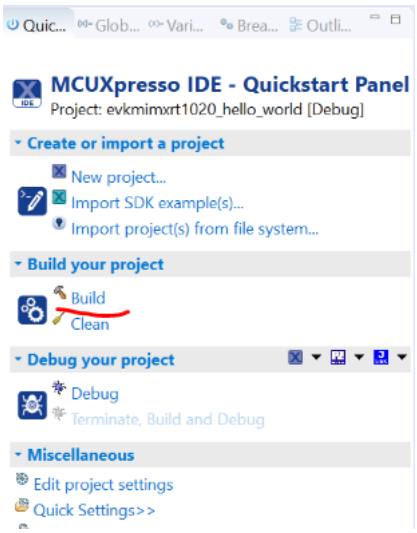
4. Expand the demo_apps folder, select hello_world, and then click **Next**.
5. Ensure the option **Redlib: Use floating-point version of printf** is selected if the cases print floating-point numbers on the terminal (for demo applications such as dac32_adc12, dac_adc, dac_cadc, ecompass, sai, coremark, mbedtls_benchmark, wolfssl_benchmark, and for mmcau_examples such as mmcau_api). Otherwise, there is no need to select it. Click **Finish**.



Note: If you want to use semihost to print log, first select the **Semihost** button when importing projects, as shown in *Figure 5*.



6. On the **Quickstart** panel, click **build** evkmimxrt1024_demo_apps_hello_world \[Debug\], as shown in *Figure 6*.



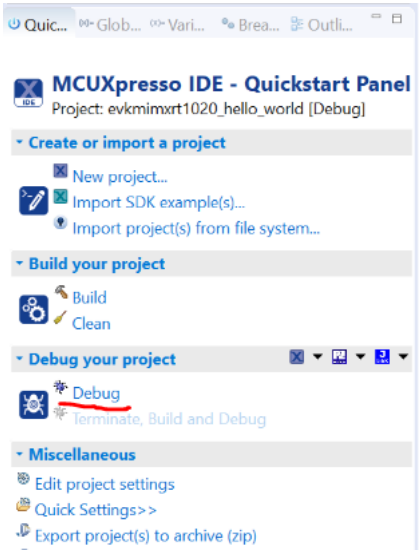
Parent topic:*Run a demo using MCUXpresso IDE*

Run an example application For more information on debug probe support in the MCUXpresso IDE, visit community.nxp.com.

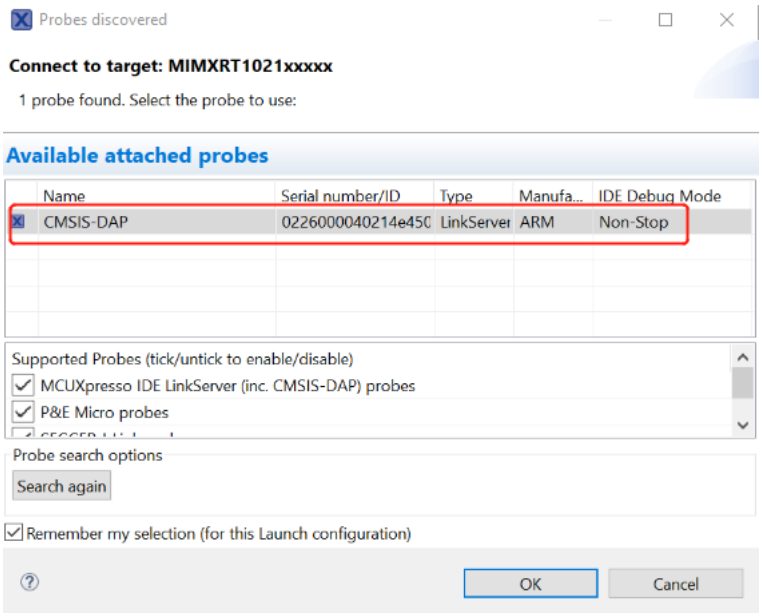
To download and run the application, perform these steps:

Note: Make sure that the board is on the QSPI_Flash mode before download (set SW8: 0010).

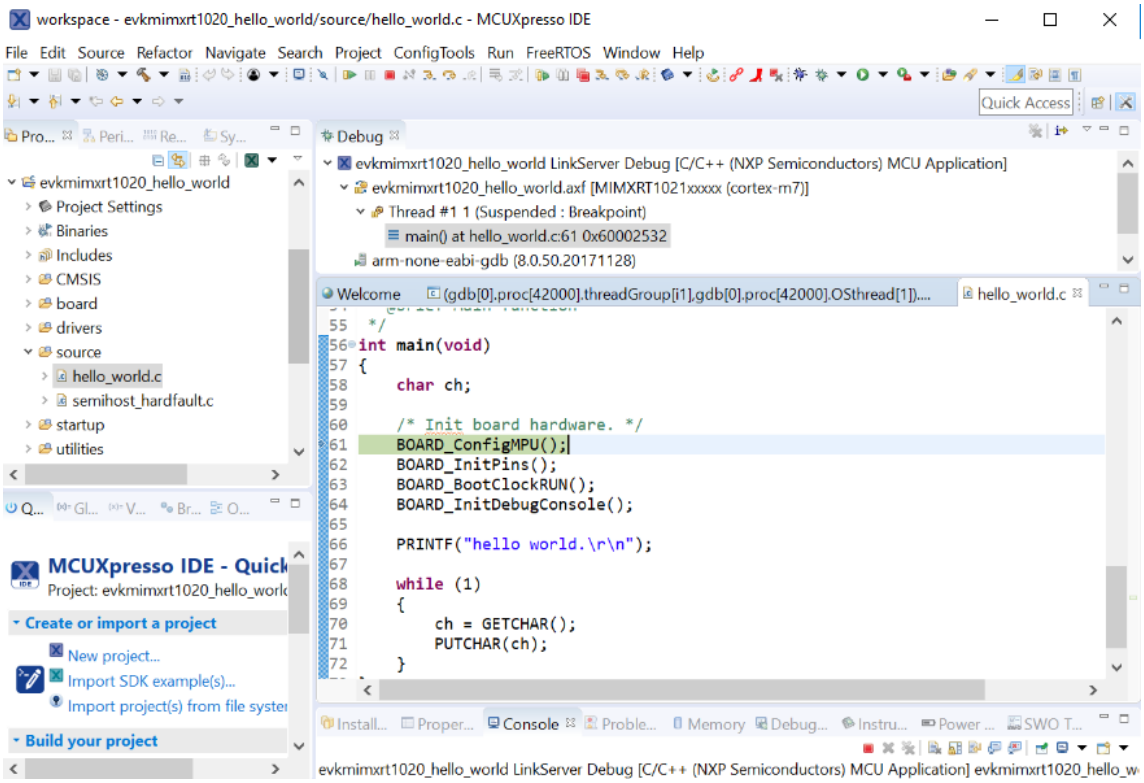
1. On the **Quickstart Panel**, click **Debug** evkmimxrt1024_demo_apps_hello_world [Debug].



2. The first time you debug a project, the **Debug Emulator Selection Dialog** is displayed, showing all supported probes that are attached to your computer. Select the probe through which you want to debug and click **OK**. (For any future debug sessions, the stored probe selection is automatically used, unless the probe cannot be found.)

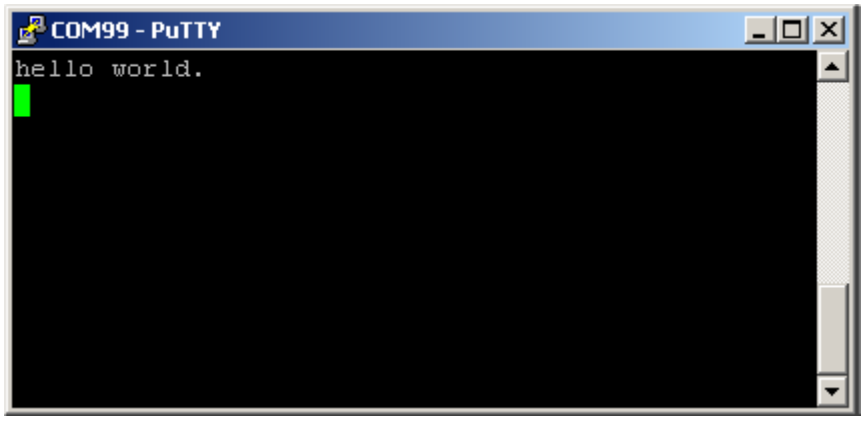


3. The application is downloaded to the target and automatically runs to main().



4. Start the application by clicking the **Resume** button.

The hello_world application is now running and a banner is displayed on the MCUXpresso IDE console window. If not, check your terminal settings and connections.

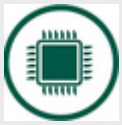


Parent topic:*Run a demo using MCUXpresso IDE*

MCUXpresso Config Tools

MCUXpresso Config Tools can help configure the processor and generate initialization code for the on chip peripherals. The tools are able to modify any existing example project, or create a new configuration for the selected board or processor. The generated code is designed to be used with MCUXpresso SDK version 2.x.

Table 1 describes the tools included in the MCUXpresso Config Tools.

Config Tool	Description	Image
Pins tool	For configuration of pin routing and pin electrical properties.	

|**Clock tool**|For system clock configuration|

|**Peripherals tools**|For configuration of other peripherals|

|**TEE tool**|Configures access policies for memory area and peripherals helping to protect and isolate sensitive parts of the application.|

|**Device Configuration tool**|Configures Device Configuration Data (DCD) contained in the program image that the Boot ROM code interprets to setup various on-chip peripherals prior to the program launch.|

MCUXpresso Config Tools can be accessed in the following products:

- **Integrated** in the MCUXpresso IDE. Config tools are integrated with both compiler and debugger which makes it the easiest way to begin the development.
- **Standalone version** available for download from www.nxp.com/mcuxpresso. Recommended for customers using IAR Embedded Workbench, Keil MDK µVision, or Arm GCC.
- **Online version** available on mcuxpresso.nxp.com. Recommended doing a quick evaluation of the processor or use the tool without installation.

Each version of the product contains a specific *Quick Start Guide* document MCUXpresso IDE Config Tools installation folder that can help start your work.

MCUXpresso IDE New Project Wizard

MCUXpresso IDE features a new project wizard. The wizard provides functionality for the user to create new projects from the installed SDKs (and from pre-installed part support). It offers user the flexibility to select and change multiple builds. The wizard also includes a library and provides source code options. The source code is organized as software components, categorized as drivers, utilities, and middleware.

To use the wizard, start the MCUXpresso IDE. This is located in the **QuickStart Panel** at the bottom left of the MCUXpresso IDE window. Select **New project**, as shown in *Figure 1*.

For more details and usage of new project wizard, see the *MCUXpresso_IDE_User_Guide.pdf* in the MCUXpresso IDE installation folder.

How to determine COM port

This section describes the steps necessary to determine the debug COM port number of your NXP hardware development platform.

1. **Linux**: The serial port can be determined by running the following command after the USB Serial is connected to the host:

   ```
   $ dmesg | grep "ttyUSB"
                   [503175.307873] usb 3-12: cp210x converter now attached to ttyUSB0
                   [503175.309372] usb 3-12: cp210x converter now attached to ttyUSB1
   ```

   There are two ports, one is Cortex-A core debug console and the other is for Cortex M4.

2. **Windows**: To determine the COM port open Device Manager in the Windows operating system. Click the **Start** menu and type **Device Manager** in the search bar.
3. In the Device Manager, expand the **Ports (COM & LPT)** section to view the available ports. The COM port names will be different for all the NXP boards.

How to define IRQ handler in CPP files

With MCUXpresso SDK, users could define their own IRQ handler in application level to override the default IRQ handler. For example, to override the default PIT_IRQHandler define in startup_DEVICE.s, application code like app.c can be implement like:

```
c
void PIT_IRQHandler(void)
{
    // Your code
}
```

When application file is CPP file, like app.cpp, then extern "C" should be used to ensure the function prototype alignment.

```
cpp
extern "C" {
    void PIT_IRQHandler(void);
}
void PIT_IRQHandler(void)
{
    // Your code
}
```

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:

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
```

(continues on next page)

(continued from previous page)

```
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`

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.

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
device	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
  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.

Development Workflows

Get started with building and running projects:

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

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.

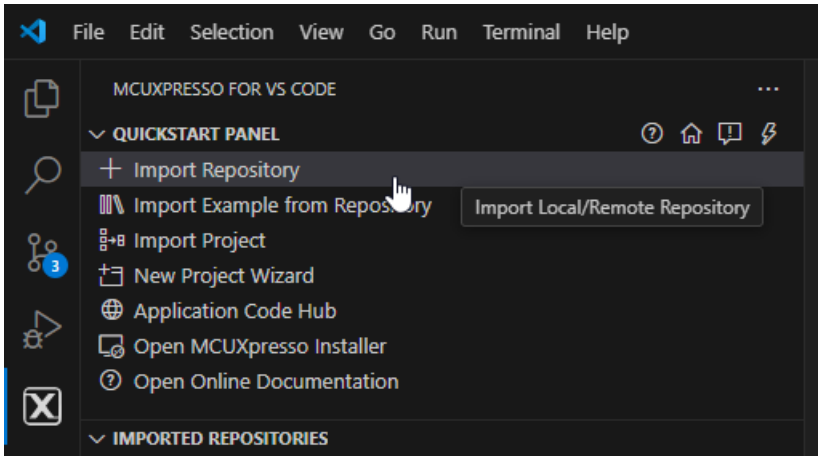
SDK Import and Setup

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.

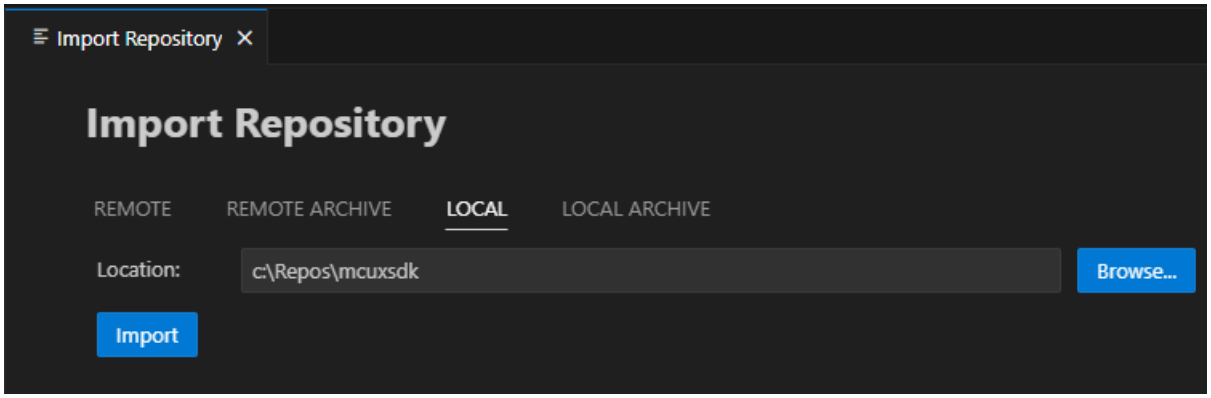
Import GitHub Repository SDK

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

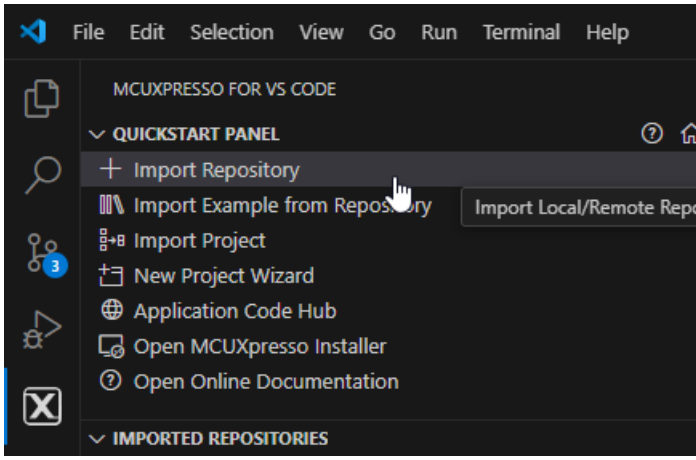


Note: You can import the SDK in several ways. Refer to the MCUXpresso for VS Code documentation for details.

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

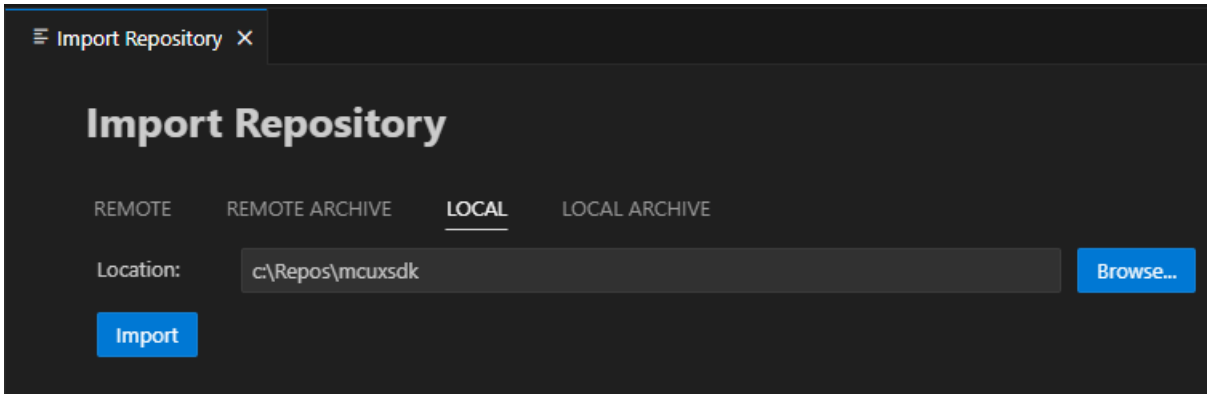


Import Repository-Layout SDK Package

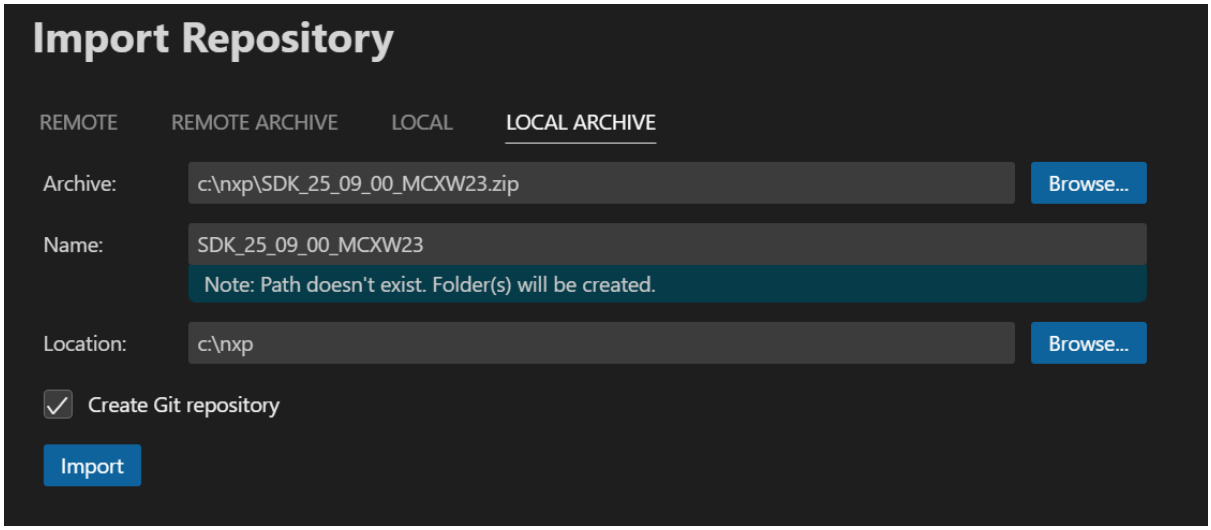


Click **Import Repository** from the **QUICKSTART PANEL**

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



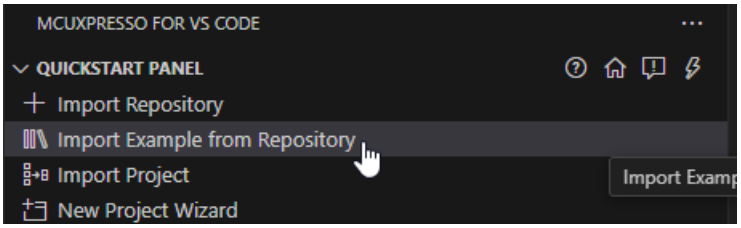
Else if the SDK is ZIP archive, select **Local Archive**, browse to the downloaded SDK ZIP file, fill the link of expect location, then click **Import**.



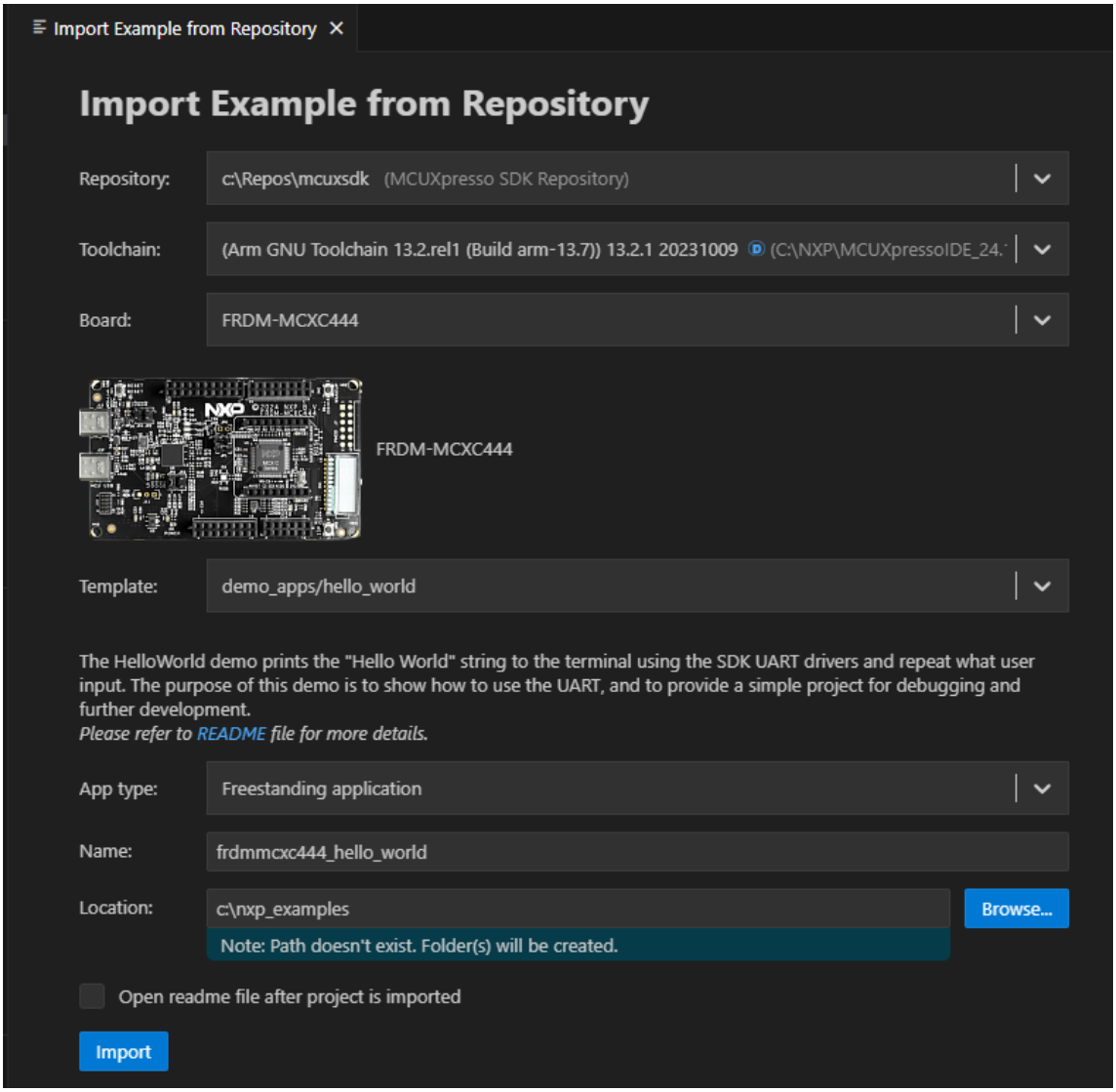
Building Example Applications

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

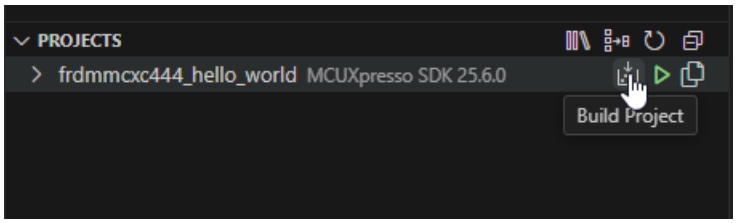
VS Code will prompt you to confirm if the imported files are trusted. Click **Yes** to proceed.

Building Projects

Build Process

1. Navigate to **PROJECTS** view
2. Find your project

3. Click the **Build Project** icon

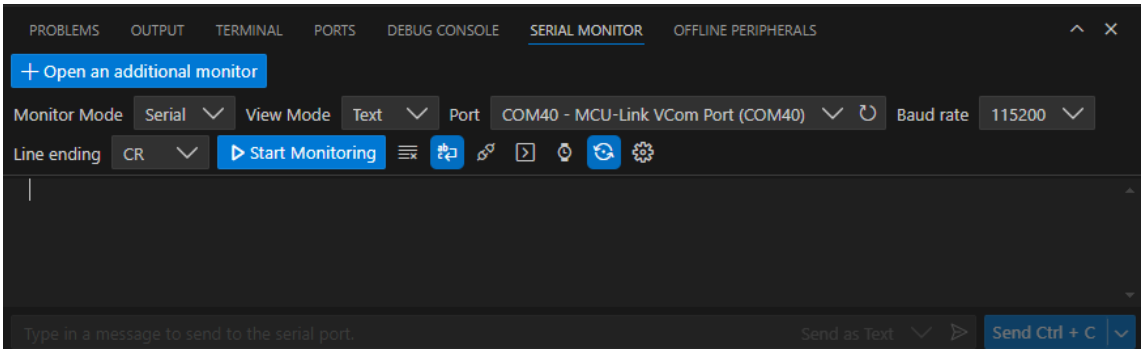


The integrated terminal will display build output at the bottom of the VS Code window.

Running and Debugging

Serial Monitor Setup

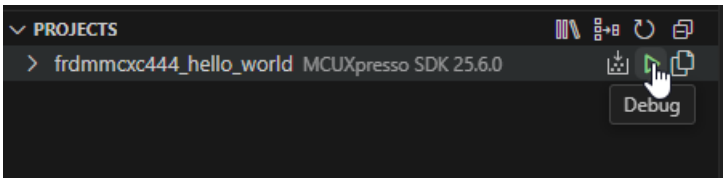
1. Open **Serial Monitor** from VS Code's integrated terminal



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



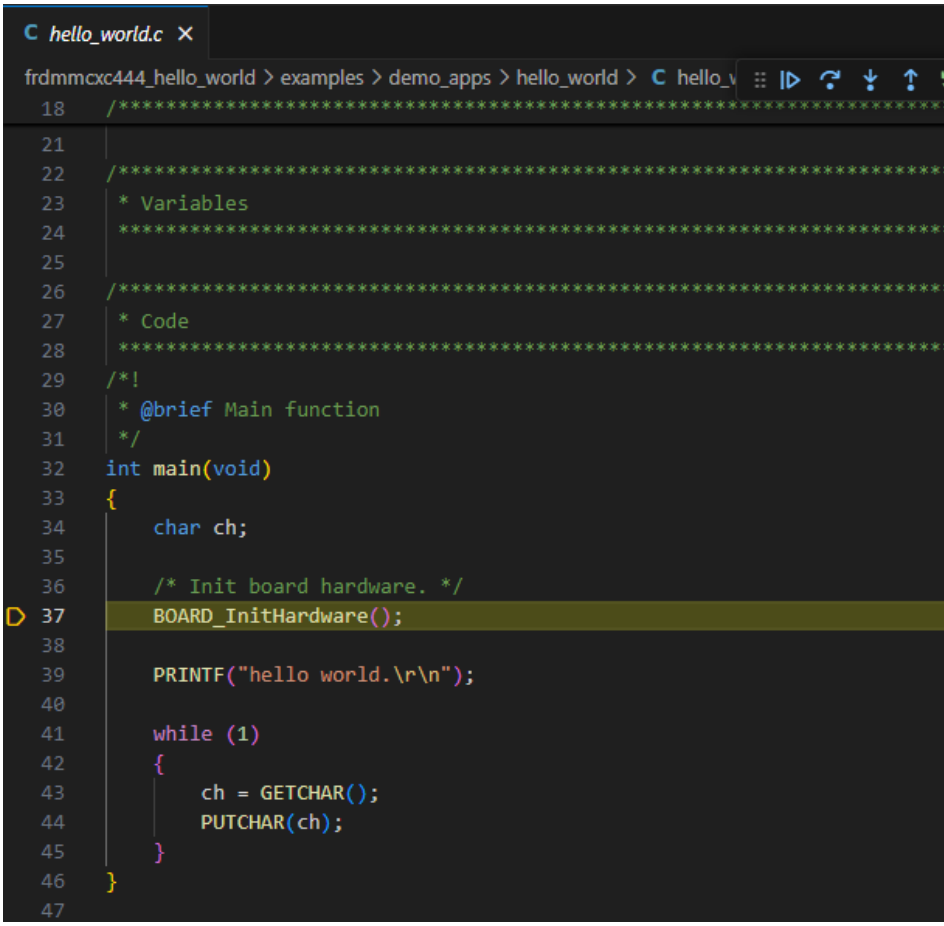
The debug session will begin with debug controls initially at the top of the interface.

Debug Controls

Use the debug controls to manage execution:

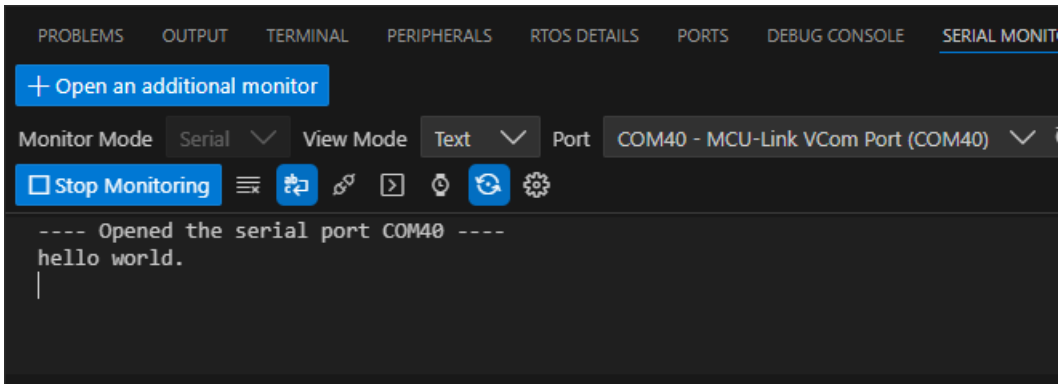
- **Continue**: Resume code execution
- **Step controls**: Navigate through code

- **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

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.

- MCUXpresso IDE, Rev. 25.06.xx
- IAR Embedded Workbench for Arm, version is 9.70.4
- Keil MDK, version is 5.43a
- 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		
MIMXRT1024-EVK	MIMXRT1024CAG4A, MIMXRT1024DAG5B	MIMXRT1024CAG4B,	**MIMXRT1024DAG5A**,

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

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

MCU Boot MCU Boot (formerly KBOOT) NXP/Freescale proprietary loader

corePKCS11 Reference implementation of the PKCS#11 standard for secure credential storage and cryptographic operations.

coreMQTT Lightweight MQTT client library for IoT devices from the AWS IoT Device SDK.

coreJSON Strictly conforming RFC 8259 JSON parser library from the AWS IoT Device SDK.

coreHTTP coreHTTP

backoffAlgorithm Library for calculating exponential backoff with jitter for network retry attempts.

NXP Wi-Fi The MCUXpresso SDK provides driver for NXP Wi-Fi external modules. The Wi-Fi driver is integrated with LWIP TCPIP stack and demonstrated with several network applications (iperf and AWS IoT).

For more information, see Getting Started with NXP based Wireless Modules and i.MX RT Platform Running on RTOS (document: UM11441).

USB Type-C PD Stack See the *MCUXpresso SDK USB Type-C PD Stack User's Guide* (document MCUXSDKUSBPDUG) for more information

USB Host, Device, OTG Stack See the MCUXpresso SDK USB Stack User's Guide (document MCUXSDKUSBSUG) for more information.

Peripheral devices tested with USB Host stack

Table 1 provides a list of USB devices tested with the USB Host stack.

Device type	Device
USB HUB	BELKIN F5U233BELKIN F5U304BELKIN F5U307BELKIN F4U040UNITEK Y-2151Z-TEK ZK032AHYUNDAI HY-HB608
USB flash drive	ADATA C008 32 GBADATA S102 8 GADATA S102 16 GVerbatim STORE N GO USB Device 8 GKingston DataTraveler DT101 G2SanDisk Cruzer Blade 8 GBUnisplendour 1 GImation 2 GBV-mux 2 GBSanmina-SCI 128 MCorporate Express 1 GTOSHIBA THUHYBS-008G 8 GTranscend JF700 8 GNetac U903 16 GSSK SFD205 8 GBRex 4 GBSAMSUNG USB3.0 16GB
USB card reader/	SSK TF adapterKawau Multi Card ReaderKawau TF adapterKawau SDHC cardUSB Mouse DELL MS111-PDELL M066U0ADELL MUAVDEL8TARGUS AMU76APDELL MD56U0DELL MS111-TRAPOO M110
USB Keyboard	DELL SK8135DELL SK8115

TinyCBOR Concise Binary Object Representation (CBOR) Library

Simple Open EtherCAT Master Simple Open EtherCAT Master (SOEM) is an open source EtherCAT master stack that is used to write custom EtherCAT Master applications. For more information on how to use SOEM, see the Getting Started with MCUXpresso SDK for SOEM document.

SDMMC stack The SDMMC software is integrated with MCUXpresso SDK to support SD/MMC/SDIO standard specification. This also includes a host adapter layer for bare-metal/RTOS applications.

PKCS#11 The PKCS#11 standard specifies an application programming interface (API), called "Cryptoki," for devices that hold cryptographic information and perform cryptographic functions. Cryptoki follows a simple object based approach, addressing the goals of technology independence (any kind of device) and resource sharing (multiple applications accessing multiple devices), presenting to applications a common, logical view of the device called a "cryptographic token".

PSA Test Suite Arm Platform Security Architecture Test Suite

mbedTLS mbedtls SSL/TLS library v3.x

mbedTLS mbedtls SSL/TLS library v2.x

LittleFS LittleFS filesystem stack

lwIP The lwIP TCP/IP stack is pre-integrated with MCUXpresso SDK and runs on top of the MCUXpresso SDK Ethernet driver with Ethernet-capable devices/boards.

For details, see the *lwIP TCPIP Stack and MCUXpresso SDK Integration User's Guide* (document MCUXSDKLWIPUG).

lwIP is a small independent implementation of the TCP/IP protocol suite.

LVGL LVGL Open Source Graphics Library

llhttp HTTP parser llhttp

JPEG library JPEG library

FreeMASTER FreeMASTER communication driver for 32-bit platforms.

File systemFatfs The FatFs file system is integrated with the MCUXpresso SDK and can be used to access either the SD card or the USB memory stick when the SD card driver or the USB Mass Storage Device class implementation is used.

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

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

Known issues

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

New Project Wizard compile failure

The following components request the user to manually select other components that they depend upon in order to compile.

These components depend on several other components and the New Project Wizard (NPW) is not able to decide which one is needed by the user.

Note: xxx means core variants, such as, cm0plus, cm33, cm4, cm33_nodsp.

Components:issdk_mag3110, issdk_host, systick, gpio_kinetis, gpio_lpc, issdk_mpl3115, sensor_fusion_agm01, sensor_fusion_agm01_lpc, issdk_mma845x, issdk_mma8491q, issdk_mma865x, issdk_mma9553, and CMSIS_RTOS2.CMSIS_RTOS2, and components which include cache driver, such as enet_qos.

Also for low-level adapter components, currently the different types of the same adapter cannot be selected at the same time.

For example, if there are two types of timer adapters, gpt_adapter and pit_adapter, only one can be selected as timer adapter

in one project at a time. Duplicate implementation of the function results in an error.

Note: Most of middleware components have complex dependencies and are not fully supported in new project wizard. Adding a middleware component may result in compile failure.

CMSIS PACK new project compile failure

The generated configuration cannot be applied globally. The components, serial_manager_usb_cdc_virtual and serial_manager_usb_cdc_virtual_xxx (xxx means core variants like cm0plus, cm33, cm4, and cm33_nodsp) are unsupported for new project wizard of CMSIS pack and will lead to compile failure if selected while creating new project(s).

Non XIP target debug issue on toolchain MDK

When debugging non XIP targets in flash boot mode, if application changes any settings which have impacts on flexspi, the build output window might show “Debug access failed” when start debugging next time. It is recommended to keep the board in serial downloader mode when debugging non XIP targets.

RAM targets build issue in CMSIS bsp pack

CMSIS pack does not support different macro definitions for different targets, all RAM targets for projects inside CMSIS BSP PACKs for RT10XX boards will get the same macro definitions with Flash targets, resulting in build failure. To pass build for RAM targets, manually update the XIP_EXTERNAL_FLASH and XIP_BOOT_HEADER_ENABLE value to 0 in RTE_Components.h.

The bee example does not complete successfully on MCUXpresso IDE

The bee example fails when built and run via MCUXpresso IDE due to misconfigured default memory configuration.

Examples: bee

Affected toolchains: mcux

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

ADC

[2.0.4]

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

[2.0.3]

- Bug Fixes
  - Fixed the violations of MISRA C-2012 rules:
    - Rule 10.1 10.4 10.7 17.7.

[2.0.2]

- Improvements
  - Used conversion control feature macro instead of that in IO map.

[2.0.1]

- New Features
  - Added a control macro to enable/disable CLOCK code in current driver.

[2.0.0]

- Initial version.

ADC_ETC

[2.3.2]

- Improvements
  - Corrected that FSL_FEATURE_ADC_ETC_HAS_NO_TSC1_TRIG should be used instead of FSL_FEATURE_ADC_ETC_HAS_NO_TSC0_TRIG in some places.
  - For ADC_ETC without TSC trigger source, CTRL [bit 30] shall be cleared explicitly.

[2.3.1]

- Improvements
  - Change ADC_ETC default DMA Mode to kADC_ETC_TrigDMAWithPulsedSignal. Generally speaking, DMA transfer requests should only be cleared by DMA ACK, and the CPU should not clear the request source. If some users use option kADC_ETC_TrigDMAWithLatchedSignal, changing the mode to kADC_ETC_TrigDMAWithPulsedSignal also meet their requirements.

[2.3.0]

- Improvements
  - Added blocking way to implement SW trigger.

[2.2.1]

- Improvements
  - Moditied macro “ADC_ETC_DONE2_ERR_IRQ_TRIG0_DONE2_MASK” to “ADC_ETC_DONE2_3_ERR_IRQ_TRIG0_DONE2_MASK” based on the updates of header file.

[2.2.0]

- Improvements
  - Defined two macros to support some devices that do not equipped with TSC trigger.

[2.1.1]

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

[2.1.0]

- New Features
  - Supported independent IRQ enable bit in ADC-ETC chain configuration registers.
  - Supported trigger n DONE3 interrupt operations.
- Bug Fixes
  - Fixed the violation of MISRA-2012 rules:
    * Rule 10.1 10.3 10.7 15.5 16.1 16.3 16.4 17.7

[2.0.1]

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

[2.0.0]

- Initial version.

AIPSTZ

[2.0.1]

- Bug Fixes
  - MISRA C-2012 issue fixed: rule 10.3, 10.4, and 14.4.

[2.0.0]

- Initial version.

AOI

[2.0.2]

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

[2.0.1]

- Bug Fixes
  - MISRA C-2012 issue fixed: rule 10.8, 2.2.

[2.0.0]

- Initial version.

BEE

[2.0.2]

- Bug Fixes
  - Fix MISRA issue.

[2.0.1]

- Bug Fixes
  - Fixed bug in key user key loading sequence. BEE must be enabled during loading of user key.
  - Fixed typos in comments.
- New Features
  - Added configuration setting for endian swap, access permission and region security level.
- Improvements
  - Setting of AES nonce was moved from BEE_SetRegionKey() into separate BEE_SetRegionNonce() function.
  - Changed handling of region settings. Both regions are configured simultaneously by BEE_SetConfig() function. Configuration of FAC start and end address using IOMUXC_GPRs was moved to application.
  - Default value for region address offset was changed to 0.

[2.0.0]

- Initial version.

CACHE ARMv7-M7

[2.0.6]

- Bug Fixes
  - Fix cache invalidate range function issue to prevent address overflow.

[2.0.5]

- Bug Fixes
  - Fixed cache operations to handle zero size and overflow in invalidate/clean functions

[2.0.4]

- Bug Fixes
  - Fixed doxygen issue.

[2.0.3]

- Improvements
  - Deleted redundancy code about calculating cache clean/invalidate size and address aligns.

[2.0.2]

- Bug Fixes
  - Fixed violation of MISRA C-2012 Rule 10.1, 10.3 and 10.4.

[2.0.1]

- Bug Fixes
  - Fixed cache size issue in L2CACHE_GetDefaultConfig API.

[2.0.0]

- Initial version.

CLOCK

[2.5.4]

- Bug Fixes
  - Fixed CERT-C INT30-C unsigned-wrap and INT31-C narrowing-cast MSG violations in CLOCK_GetClockRootFreq (CCM tuple enum casts via inline suppression), CLOCK_GetPllFreq (Sys/USB1/Audio PLL freq computations promoted to uint64), CLOCK_InitSysPfd / CLOCK_InitUsb1Pfd (pfdIndex bounded), CLOCK_GetSysPfdFreq / CLOCK_GetUsb1PfdFreq (PFD freq narrowing cast clamped), and CLOCK_SetMux, CLOCK_GetMux, CLOCK_SetDiv, CLOCK_GetDiv, CLOCK_ControlGate (CCM tuple enum casts via inline suppression).

[2.5.3]

- Bug Fixes
  - Fixed the violations of MISRA C-2012 rule 10.7.

[2.5.2]

- Bug Fixes
  - Fixed issues in CLOCK_GetSysPfdFreq() and CLOCK_GetUsb1PfdFreq() which produce incorrect result.

[2.5.1]

- Improvements
  - Added enumeration clock_div_value_t.

[2.5.0]

- New features
  - Added CLOCK_IsUsb1PfdEnabled and CLOCK_IsSysPfdEnabled to get the clock source status.

[2.4.1]

- Improvements
  - Placed function internal constants into initialized data segment.

[2.4.0]

- New Features
  - Added clock output related APIs and data structures.
  - Added one function CLOCK_GetClockRootFreq to get the frequency of each clock root.

[2.3.2]

- Bug Fixes
  - Fixed MISRA C-2012 rule 20.7 rule 8.6 rule 10.3 and rule 10.1.
  - Fixed IAR warning Pa082 for the clock driver.

[2.3.1]

- Improvements
  - Added new macro in case that the CCM has CAN clock affect issue.

[2.3.0]

- New feature:
  - Moved SDK_DelayAtLeastUs function from clock driver to common driver.

[2.2.0]

- New feature
  - Adding new API CLOCK_DelayAtLeastUs() implemented by DWT to allow users set delay in unit of microsecond.

[2.1.6]

- Bug Fix:
  - Fix build issue with GCC compiler when include header from C++ file.

[2.1.5]

- Bug Fix:
  - Add initialization of the fractional mode and spread spectrum mode in CLOCK_InitSysPll().

[2.1.4]

- Optimization:
  - Add PerClk in clock_name_t and CLOCK_GetFreq.
  - Add APIs to get the frequency of AHB clock and SEMC, IPG clock and PER clock.

[2.1.3]

- Use double instead of uint64_t to achieve better performance with double precision FPU.

[2.1.2]

- some minor fixes.

[2.0.0]

- initial version.

CMP

[2.0.3]

- Improvements
  - Updated to clear CMP settings in DeInit function.

[2.0.2]

- Bug Fixes
  - Fixed the violations of MISRA 2012 rules:
    - Rule 10.3

[2.0.1]

- Bug Fixes
  - Fixed MISRA-2012 rules.
    - Rule 14.4, rule 10.3, rule 10.1, rule 10.4 and rule 17.7.

[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.

DCDC

[2.3.0]

- Improvements
  - REG3[MISC_DELAY_TIMING], REG2[LOOPCTRL_DC_R], and REG2[LOOPCTRL_DC_C] are reserved in the latest RM, deleted corresponding functions.

[2.2.1]

- Improvements
  - Fixed the doxygen warning.

[2.2.0]

- New Features
  - Added supports for i.MXRT1170 series.
- Bug Fixes
  - Fixed the warning that the DCDC_ConvertByteArrayToWord function defined but not used.
- Improvements
  - Updated rcscale to reduce the ripple when booting into DCM.

[2.1.0]

- Improvements
  - Divided the DCDC_AdjustTargetVoltage() into two APIs for two different modes.
- Bug Fixes
  - Fixed the violations of MISRA C-2012 rules:
    - Rule 10.1, 10.4, 16.4, 17.7.

[2.0.0]

- Initial version.

DCP

[2.1.8]

- Bug Fix
  - Fix missing OTP flag in DCP Control0 field when using OTP UNIQUE keys.

[2.1.7]

- Bug Fix
  - Reduce optimization level for critical functions working with SRF.

[2.1.6]

- Bug Fix
  - MISRA C-2012 issue fix.

[2.1.5]

- Improvements
  - Added support when DCACHE enabled. Input and output buffers should be in non-cached memory or handled properly (DCACHE Clean and Invalidate).

[2.1.4]

- Bug Fix
  - Fix CRC-32 computation issue on the code's block boundary size.

[2.1.3]

- Bug Fix
  - MISRA C-2012 issue fixed: rule 10.1, 10.3, 10.4, 11.9, 14.4, 16.4 and 17.7.

[2.1.2]

- Bug Fix
  - Fix sign-compare warning in dcp_reverse_and_copy.

[2.1.1]

- Improvements
  - Added DCP status clearing when channel operation is complete.

[2.1.0]

- New Features
  - Added byte/word swap feature for key, input, and output data.

[2.0.0]

- Initial version.

DMAMUX

[2.1.4]

- Bug Fixes
  - Fixed CERT INT31-C violation in DMAMUX_SetSource by adding non-negative assertion and explicit cast.

[2.1.3]

- Improvements
  - Wrap DMAMUX_GetInstance into FSL_SDK_DISABLE_DRIVER_CLOCK_CONTROL to avoid build issues.

[2.1.2]

- Bug Fixes
  - Add macro FSL_DMAMUX_CHANNEL_NUM to calculat correct DMAMUX channel number when input EDAM channel number.

[2.1.1]

- Improvements
  - Add macro FSL_FEATURE_DMAMUX_CHANNEL_NEEDS_ENDIAN_CONVERT and DMAMUX_CHANNEL_ENDIAN_CONVERTn do channel endian convert.

[2.1.0]

- Improvements
  - Modify the type of parameter source from uint32_t to int32_t in the DMAMUX_SetSource.

[2.0.5]

- Improvements
  - Added feature FSL_FEATURE_DMAMUX_CHCFG_REGISTER_WIDTH for the difference of CHCFG register width.

[2.0.4]

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

[2.0.3]

- Bug Fixes
  - Fixed the issue for MISRA-2012 check.
    * Fixed rule 10.4 and rule 10.3.

[2.0.2]

- New Features
  - Added an always-on enable feature to a DMA channel for ULP1 DMAMUX support.

[2.0.1]

- Bug Fixes
  - Fixed the build warning issue by changing the type of parameter source from uint8_t to uint32_t when setting DMA request source in DMAMUX_SetSourceChange.

[2.0.0]

- Initial version.

EDMA

[2.4.9]

- Bug Fixes
  - Fixed CERT INT31-C and INT30-C violations in edma driver.

[2.4.8]

- New Features
  - Added common IRQ handler entry EDMA_DriverIRQHandler.

[2.4.7]

- Bug Fixes
  - Fixed coverity MSG issues with CERT INT31-C compliance.

[2.4.6]

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

[2.4.5]

- Bug Fixes
  - Fixed memory convert would convert NULL as zero address issue.

[2.4.4]

- Bug Fixes
  - Fixed comments by replacing STCD with TCD
  - Fixed the TCD overwrite issue when submit transfer request in the callback if there is a active TCD in hardware.
  - Fixed violations of MISRA C-2012 rule 10.8,5.6.

[2.4.3]

- Improvements
  - Added FSL_FEATURE_MEMORY_HAS_ADDRESS_OFFSET to convert the address between system mapped address and dma quick access address.
- Bug Fixes
  - Fixed the wrong tcd done count calculated in first TCD interrupt for the non scatter gather case.

[2.4.2]

- Bug Fixes
  - Fixed the wrong tcd done count calculated in first TCD interrupt by correct the initial value of the header.
  - Fixed violations of MISRA C-2012 rule 10.3, 10.4.

[2.4.1]

- Bug Fixes
  - Added clear CITER and BITER registers in EDMA_AbortTransfer to make sure the TCD registers in a correct state for next calling of EDMA_SubmitTransfer.
  - Removed the clear DONE status for ESG not enabled case to aovid DONE bit cleared unexpectedly.

[2.4.0]

- Improvements
  - Added api EDMA_EnableContinuousChannelLinkMode to support continuous link mode.
  - Added apis EDMA_SetMajorOffsetConfig/EDMA_TcdSetMajorOffsetConfig to support major loop address offset feature.
  - Added api EDMA_EnableChannelMinorLoopMapping for minor loop offset feature.
  - Removed the reduntant IRQ Handler in edma driver.

[2.3.2]

- Improvements
  - Fixed HIS ccm issue in function EDMA_PrepareTransferConfig.
  - Fixed violations of MISRA C-2012 rule 11.6, 10.7, 10.3, 18.1.
- Bug Fixes

  - Added ACTIVE & BITER & CITER bitfields to determine the channel status to fixed the issue of the transfer request cannot submit by function EDMA_SubmitTransfer when channel is idle.

[2.3.1]

- Improvements
  - Added source/destination address alignment check.
  - Added driver IRQ handler support for multi DMA instance in one SOC.

[2.3.0]

- Improvements
  - Added new api EDMA_PrepareTransferConfig to allow different configurations of width and offset.
- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 10.4, 10.1.
  - Fixed the Coverity issue regarding out-of-bounds write.

[2.2.0]

- Improvements
  - Added peripheral-to-peripheral support in EDMA driver.

[2.1.9]

- Bug Fixes
  - Fixed MISRA issue: Rule 10.7 and 10.8 in function EDMA_DisableChannelInterrupts and EDMA_SubmitTransfer.
  - Fixed MISRA issue: Rule 10.7 in function EDMA_EnableAsyncRequest.

[2.1.8]

- Bug Fixes
  - Fixed incorrect channel preemption base address used in EDMA_SetChannelPreemptionConfig API which causes incorrect configuration of the channel preemption register.

[2.1.7]

- Bug Fixes
  - Fixed incorrect transfer size setting.
    - Added 8 bytes transfer configuration and feature for RT series;
    - Added feature to support 16 bytes transfer for Kinetis.
  - Fixed the issue that EDMA_HandleIRQ would go to incorrect branch when TCD was not used and callback function not registered.

[2.1.6]

- Bug Fixes
  - Fixed KW3X MISRA Issue.
    - Rule 14.4, 10.8, 10.4, 10.7, 10.1, 10.3, 13.5, and 13.2.
- Improvements
  - Cleared the IRQ handler unavailable for specific platform with macro FSL_FEATURE_EDMA_MODULE_CHANNEL_IRQ_ENTRY_SHARED_OFFSET.

[2.1.5]

- Improvements
  - Improved EDMA IRQ handler to support half interrupt feature.

[2.1.4]

- Bug Fixes
  - Cleared enabled request, status during EDMA_Init for the case that EDMA is halted before reinitialization.

[2.1.3]

- Bug Fixes
  - Added clear DONE bit in IRQ handler to avoid overwrite TCD issue.
  - Optimized above solution for the case that transfer request occurs in callback.

[2.1.2]

- Improvements
  - Added interface to get next TCD address.
  - Added interface to get the unused TCD number.

[2.1.1]

- Improvements
  - Added documentation for eDMA data flow when scatter/gather is implemented for the EDMA_HandleIRQ API.
  - Updated and corrected some related comments in the EDMA_HandleIRQ API and edma_handle_t struct.

[2.1.0]

- Improvements
  - Changed the EDMA_GetRemainingBytes API into EDMA_GetRemainingMajorLoopCount due to eDMA IP limitation (see API comments/note for further details).

[2.0.5]

- Improvements
  - Added pubweak DriverIRQHandler for K32H844P (16 channels shared).

[2.0.4]

- Improvements
  - Added support for SoCs with multiple eDMA instances.
  - Added pubweak DriverIRQHandler for KL28T DMA1 and MCIMX7U5_M4.

[2.0.3]

- Bug Fixes
  - Fixed the incorrect pubweak IRQHandler name issue, which caused re-definition build errors when client set his/her own IRQHandler, by changing the 32-channel IRQHandler name to DriverIRQHandler.

[2.0.2]

- Bug Fixes
  - Fixed incorrect minorLoopBytes type definition in _edma_transfer_config struct, and defined minorLoopBytes as uint32_t instead of uint16_t.

[2.0.1]

- Bug Fixes
  - Fixed the eDMA callback issue (which did not check valid status) in EDMA_HandleIRQ API.

[2.0.0]

- Initial version.

ENC

[2.2.2]

- Bug Fixes
  - Fixed CERT INT31-C violations.

[2.2.1]

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

[2.2.0]

- New Features
  - Supported input filter prescaler.

[2.1.0]

- Improvements
  - Supported period measurement function.

[2.0.2]

- Improvements
  - Added feature macro for CTRL2[SABIE] and CTRL2[SABIRQ] bits.

[2.0.1]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 10.1, 10.3, 10.4, 10.6, 17.7.

[2.0.0]

- Initial version.

ENET

[2.11.4]

- New Features
  - Added the parameterized common IRQ handlers ENET_CommonFrame0DriverIRQHandler and ENET_Ptp1588DriverIRQHandler, and bound the COMM__ENET vectors to them on i.MX RT2660.

[2.11.3]

- Bug Fixes
  - Fixed a condition in the receive path where software could read receive buffer descriptor fields directly while hardware/DMA could still be updating them.

[2.11.2]

- Improvements
  - Removed conditional compiling of kENET_RgmiiMode/kENET_MiiSpeed1000M definition.

[2.11.1]

- Bug Fixes
  - Fixed TX timestamp second value being incorrect when second rollover occurs between frame transmission and descriptor reclaim. The driver now reconstructs the correct second value by comparing TX nanoseconds with current nanoseconds, following the same pattern used for RX timestamps in SDK examples.

[2.11.0]

- New Features
  - Added function ENET_Ptp1588JumpTimer which adjusts the ENET PTP 1588 timer by jumping a relative time difference. Compared to ENET_Ptp1588SetTimer, this function yields more accurate results when the relative time difference between the PTP clock and the target clock is known.

[2.10.1]

- Bug Fixes
  - Fixed WAKEUP interrupt not being handled.

[2.10.0]

- New Features
  - Added function ENET_Ptp1588GetChannelCaptureValue to read last captured time from PTP 1588 timer.

[2.9.3]

- Bug Fixes
  - Fixed ENET_Ptp1588GetTimer incorrect timestamps when timer wraps occur after nanosecond capture:
    - Now only increments second field when nanosecond value is less than half a second

[2.9.2]

- Bug Fixes
  - RGMII mode is (temporarily) disabled before selecting between 10/100-Mbit/s and 1000-Mbit/s modes of operation. The bit RGMII_EN of RCR register must not be set while changing ECR register's speed bit, otherwise there is a possibility of ENET IP ending in an incorrect state.

[2.9.1]

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

[2.9.0]

- Bug Fixes
  - Enabled collection of transfer statistics, so the function ENET_GetStatistics does not always return zeroes.
- New Features
  - Added new function ENET_EnableStatistics to enable/disable collection of transfer statistics.
  - Added new function ENET_ResetStatistics to reset transfer statistics.
- Improvements

  - Renamed the function ENET_ResetHareware to ENET_ResetHardware.

[2.8.0]

- New Features
  - Added the function to reset hardware on certain devices.

[2.7.1]

- Bug Fixes
  - Fixed the issue that free wrong buffer address when one frame stores in multiple buffers and memory pool is not enough to allocate these buffers to receive one complete frame.

[2.7.0]

- Improvements
  - Deleted deprecated zero copy Tx/Rx functions and set callback function which can be configured in ENET_Init.
  - Moved the Rx zero copy buffer allocation to Rx BD initialization function to reduce unnecessary looping code.
- Bug Fixes
  - Fixed the issue that predefined Rx buffers which should not be used when enabling Rx zero copy are still be handled by cache operation, it causes hardfault on some platforms.
  - Fixed the issue that zero-copy Rx function doesn't check Rx length of 0 in the BD with EMPTY bit is 0, it may occur in the corner case reported by customer. Not sure how it turns out, consider it as an ENET IP issue and drop this abnormal BD.

[2.6.3]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 11.6.

[2.6.2]

- Improvements
  - Changed ENET1_MAC0_Rx_Tx_Done0_DriverIRQHandler/ENET1_MAC0_Rx_Tx_Done1_DriverIRQHan
    to ENET1_MAC0_Rx_Tx_Done1_DriverIRQHandler/ENET1_MAC0_Rx_Tx_Done2_DriverIRQHandler
    which represent ring 1 and ring 2.

[2.6.1]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 10.3, 10.4, 10.7, 11.6, 11.8.

[2.6.0]

- Improvements
  - Added MDIO access wrapper APIs for ease of use.
  - Fixed the build warning introduced by 64-bit compatibility patch.

[2.5.4]

- Improvements
  - Made the driver compatible with 64-bit platforms.

[2.5.3]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 11.6.

[2.5.2]

- Improvements
  - Updated the TXIC/RXIC register handling code according to the new header file.

[2.5.1]

- Bug Fixes
  - Fixed document typo.

[2.5.0]

- Bug Fixes
  - Fixed the SendFrame/SendFrameZeroCopy functions issue with scattered buffers.
  - Updated the formula of MDC calculation.
  - Used a feature macro to distinguish the old IP design from the new design, because old IP design always reads a value zero from ATCR->CAPTURE bit. For old IP, driver caculates and wait the necessary delay cycles after setting ATCR->CAPTURE then gets the timestamp value.
- New Features
  - Added new zero copy Tx/Rx function.
  - New zero copy Tx function combines scattered and contiguous Tx buffer in one API, it also supports more Tx featrues which buffer descriptor supports but previous Tx function doesn't support.
  - New zero copy Rx function use dynamic buffer mechanism and simpler interface.
- Improvements
  - Corrected the interrupt handler for PTP timestamp IRQ and PTP1588 event IRQ since platform difference.
  - Added missing IRQ handlers for PTP1588 events on some platforms.

  - Corrected the max Tx frame length verification, it will not depend on a fixed macro. The ENET_FRAME_MAX_FRAMELEN is only an default value for driver, application can configure it. Driver caculates the limitation with the max frame length in register which may takes extended 4 or 8 bytes VLAN tag if VLAN/SVLAN enables.
  - Deleted deprecated Clause 45 read/write legacy APIs.

[2.4.3]

- Improvements
  - Aligned the IRQ handler name with header file.

[2.4.2]

- Bug Fixes
  - Fixed the MISRA issue of speculative out-of-bounds access.

[2.4.1]

- Bug Fixes
  - Fixed the PTP time capture issue.

[2.4.0]

- Improvements
  - Exposed API ENET_ReclaimTxDescriptor for user application to relaim tx descriptors in their application.
  - Added counter to record multicast hash conflict in struct _enet_handle, improved the situation that one multicast group could be left by other conflict multicast address left operation.
  - Improved concurrent usage of relaim and send frame operation.

[2.3.4]

- Bug Fixes
  - Fixed the issue that interrupt handler only checks the interrupt event flag but not checks interrupt mask flag.

[2.3.3]

- Bug Fixes
  - Fixed the issue that some compilers may choose the memcpy with 4-bit aligned address limitation due to the type of address pointer is 'unsigned int *', the data address doesn't have to be 4-bit aligned.

[2.3.2]

- New Features
  - Added the feature that ENET driver can be used in the platform which integrates both 10/100M and 1G ENET IP.
  - Deleted duplicated code about ARM errata 838869 in first/second level IRQ handler.

[2.3.1]

- Improvements
  - Added function pointer checking in IRQ handler to make sure code can be used even it runs into the interrupt when the second level interupt handler is NULL.

[2.3.0]

- Bug Fixes
  - Fixed the issue that clause 45 MDIO read/write API doesn't check the transmission over status between two transmissions.
  - Fixed violations of the MISRA C-2012 rules 2.2,10.3,10.4,10.7,11.6,11.8,13.5,14.4,15.7,17.7.
- New Features
  - Added APIs to support send/receive frame with Zero-Copy.
- Improvements
  - Separated the clock configuration from module configuration when init and deinit.
  - Added functions to set second level interrupt handler.
  - Provided new function to get 1588 timer count without disabling interrupt.
  - Improved timestamp controlling, deleted all old timestamp management APIs and data structures.
  - Merged the single/multiple ring(s) APIs, now these APIs can handle both.
  - Used base and index to control buffer descriptor, aligned with qos and lpc enet driver.

[2.2.6]

- Bug Fixes
  - Updated MII speed formula referring to the manual.

[2.2.5]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 10.1, 10.3, 10.4, 10.6, 10.7, 11.6, 11.9, 13.5, 14.4, 16.4, 17.7, 21.15, 3.1, 8.4.
  - Changed to use ARRAY_SIZE(s_enetBases) as the array size for s_ENETHandle, fixed the hardfault issue for using some ENET instance when ARRAY_SIZE(s_enetBases) is not same as FSL_FEATURE_SOC_ENET_COUNT.

[2.2.4]

- Improvements
  - Added call to Data Synchronization Barrier instruction before activating Tx/Rx buffer descriptor to ensure previous data update is completed.
  - Improved ENET_TransmitIRQHandler to store timestamps for multiple transmit buffer descriptors.
  - Bug Fixes
  - Fixed the issue that ENET_Ptp1588GetTimer did not handle the timer wrap situation.

[2.2.3]

- Improvements
  - Improved data buffer cache maintenance in the ENET driver.

[2.2.2]

- New Features
  - Added APIs for extended multi-ring support.
  - Added the AVB configure API for extended AVB feature support.

[2.2.1]

- Improvements
  - Changed the input data pointer attribute to const in ENET_SendFrame().

[2.1.1]

- New Features
  - Added the extended MDIO IEEE802.3 Clause 45 MDIO format SMI command APIs.
  - Added the extended interrupt coalescing feature.
- Improvements
  - Combined all storage operations in the ENET_Init to ENET_SetHandler API.

[2.0.1]

- Bug Fixes
  - Used direct transmit busy check when doing data transmit.
- Miscellaneous Changes
  - Updated IRQ handler work flow.
  - Changed the TX/RX interrupt macro from kENET_RxByteInterrupt to kENET_RxBufferInterrupt, from kENET_TxByteInterrupt to kENET_TxBufferInterrupt.
  - Deleted unnecessary parameters in ENET handler.

[2.0.0]

- Initial version.

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.

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.

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_I2S_EDMA

[2.1.9]

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

[2.1.8]

- Improvements
  - Applied EDMA ERRATA 51327.

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

FlexPWM

[2.0.0]

- Initial version of FlexPWM driver which is refactored from PWM driver (fsl_pwm.c/h).
- Improvements
  - Decomposed monolithic initialization into 11 dedicated configuration functions following single responsibility principle.
    - FLEXPWM_ConfigSubmodule for counter and reload configuration.
    - FLEXPWM_ConfigPWM for PWM output configuration.
    - FLEXPWM_ConfigPWMChannelX for PWM_X channel configuration.
    - FLEXPWM_ConfigFaultProtection for fault protection settings.
    - FLEXPWM_ConfigInputCapture for input capture configuration.
    - FLEXPWM_ConfigOutputTrigger for output trigger configuration.
    - FLEXPWM_ConfigDMAControl for DMA control settings.
    - And other specialized configuration functions.
  - Added inline register access APIs for zero-overhead direct control.
    - FLEXPWM_SetVAL0 through FLEXPWM_SetVAL5 compile to single-instruction register writes.
  - Reorganized data structures into hierarchical configuration for better clarity.
    - flexpwm_submodule_config_t for submodule-level settings.
    - flexpwm_pwm_config_t for PWM signal parameters.
    - flexpwm_fault_protection_config_t for fault protection.
  - Added atomic multi-submodule control using bitmask-based operations.
    - FLEXPWM_EnableSubmoduleCounter supports simultaneous start of multiple submodules.
    - Eliminates race conditions in multi-channel scenarios.
  - Improved fault protection configuration with dedicated structures and APIs.
  - Updated naming convention from PWM_* to FLEXPWM_* to align with peripheral name.
  - Removed implicit side effects from initialization functions for more predictable behavior.

FLEXRAM

[2.3.0]

- New Features
  - Supported platforms which have ECC but no ECC error injection.

[2.2.0]

- New Features
  - Supported flexram ECC error injection function.

[2.1.0]

- New Features
  - Supported flexram ECC function.

[2.0.7]

- Bug Fixes
  - Fixed doxygen issue.

[2.0.6]

- New Features
  - Updated bank configuration and TCM size with GPR16/GPR17/GPR18 into SOC level for different SOC.

[2.0.5]

- New Features
  - Added the magic address feature for OCRAM, DTCM and ITCM.

[2.0.4]

- Bug Fixes
  - Fixed FlexRAM driver's missing extern C around functions in header file.
  - Removed magic address feature from driver.

[2.0.3]

- Bug Fixes
  - Fixed the issue that TCM size configuration was wrong when TCM bank number was not a value power of 2.

[2.0.2]

- Bug Fixes
  - Updated driver due to Reference Manual update.

[2.0.1]

- Bug Fixes
  - Fixed MISRA issue.

[2.0.0]

- Initial version.

FLEXSPI

[2.9.3]

- Bug Fixes
  - Fixed CERT INT30-C violations in FLEXSPI_TransferBlocking: added assert guards before unsigned arithmetic on flashAddress to prevent wrap-around.
  - Fixed CERT INT31-C violations in FLEXSPI_Init and FLEXSPI_UpdateAhbBuffersSettings: added assert to verify FSL_FEATURE_FLEXSPI_AHB_RX_BUFFER_SIZEn returns a non-negative value before casting to uint32_t.

[2.9.2]

- New Features
  - Added common IRQ handler entry FLEXSPI_CommonDriverIRQHandler.

[2.9.1]

- Bug Fixes
  - Fixed CERT INT31-C violations: added masks before uint8_t casts and replaced bool fields with ternary expressions.
- Improvements
  - Modified FLEXSPI_WriteBlocking() to trigger IP command after TX FIFO is filled or when all data has been written, preventing premature command execution
  - Updated $\mathrm{FLEXSPI_TransferBlocking()}$ to distinguish between Write and Read operation flows according to FlexSPI IP command specification.
    - For Write/Program operations: Fill IP transmit FIFO with data before triggering the IP command.
    - For Read operations: Trigger IP command first, then read data from IP receive FIFO.
    - Added detailed flow comments for both Write and Read operations.

[2.9.0]

- New Features
  - Added FlexSPI root clock divider configuration.
- Bug Fixes
  - Add explicit boolean to integer conversions (? 1U : 0U) for all enable flags
  - Add assertions to prevent unsigned integer overflow in buffer size calculations
  - Add assertions to prevent unsigned integer underflow in watermark calculations
  - Add safe type conversion for uint32_t to uint8_t operations
  - Add bounds checking assertion for LUT index operations
  - Follow INT30-C and INT31-C CERT coding standards

[2.8.1]

- Improvements
  - Updated the LUT configuration parameter checking with flexible way to adapt different Socs.

[2.8.0]

- Bug Fixes
  - Introduced the **disableAhbReadResume** field in the flexspi_config_t structure to provide control over the AHBCR[RESUMEDISABLE] register bit.
  - Implemented a workaround for hardware erratum ERR052733 by setting the default value of **disableAhbReadResume** to **true**.
  - Fixed issue in FLEXSPI_TransferHandleIRQ where the transfer completion was incorrectly signaled despite pending read/write operations.
- New Features
  - Introduced a new function(FLEXSPI_UpdateAhbBuffersSettings) that allows users to update the AHB buffer configuration after the FLEXSPI module has been initialized

[2.7.0]

- New Features
  - Added new API to support address remapping.

[2.6.4]

- Improvements
  - Added new macro "FSL_SDK_ENABLE_FLEXSPI_RESET_CONTROL" to support driver level reset control.

[2.6.3]

- Bug Fixes
  - Fixed an issue which cause IPCR1[IPAREN] cleared by mistake.

[2.6.2]

- Bug Fixes
  - Wait Bus IDLE before operation of FLEXSPI_SoftwareReset(), FLEXSPI_TransferBlocking() and FLEXSPI_TransferNonBlocking().

[2.6.1]

- Bug Fixes
  - Updated code of reset peripheral.
  - Updated FLEXSPI_UpdateLUT() to check if input lut address is not in Flexspi AMBA region.
  - Updated FLEXSPI_Init() to check if input AHB buffer size exceeded maximum AHB size.

[2.6.0]

- New Features
  - Added new API to set AHB memory-mapped flash base address.
  - Added support of DLLxCR[REFPHASEGAP] bit field, it is recommended to set it as 0x2 if DLL calibration is enabled.

[2.5.1]

- Bugfixes
  - Fixed handling of W1C bits in the INTR register
  - Removed FIFO resets from FLEXSPI_CheckAndClearError
  - FLEXSPI_TransferBlocking is observing IPCMDDONE and then fetches the final status of the transfer
  - Fixed issue that FLEXSPI2_DriverIRQHandler not defined.

[2.5.0]

- Improvements
  - Supported word un-aligned access for write/read blocking/non-blocking API functions.
  - Fixed dead loop issue in DLL update function when using FRO clock source.
  - Fixed violations of the MISRA C-2012 Rule 10.3.

[2.4.0]

- Improvements
  - Isolated IP command parallel mode and AHB command parallel mode using feature MACRO.
  - Supported new column address shift feature for external memory.

[2.3.5]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 Rule 14.2.

[2.3.4]

- Bug Fixes
  - Updated flexspi_config_t structure and FlexSPI_Init to support new feature FSL_FEATURE_FLEXSPI_HAS_NO_MCR0_CONBINATION.

[2.3.3]

- Bug Fixes
  - Removed feature FSL_FEATURE_FLEXSPI_DQS_DELAY_PS for DLL delay setting. Changed to use feature FSL_FEATURE_FLEXSPI_DQS_DELAY_MIN to set slave delay target as 0 for DLL enable and clock frequency higher than 100MHz.

[2.3.2]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 Rule 8.4, 8.5, 10.1, 10.3, 10.4, 11.6 and 14.4.

[2.3.1]

- Bug Fixes
  - Wait for bus to be idle before using it as access to external flash with new setting in FLEXSPI_SetFlashConfig() API.
  - Fixed the potential buffer overread and Tx FIFO overwrite issue in FLEXSPI_WriteBlocking.

[2.3.0]

- New Features
  - Added new API FLEXSPI_UpdateDllValue for users to update DLL value after updating flexspi root clock.
  - Corrected grammatical issues for comments.
  - Added support for new feature FSL_FEATURE_FLEXSPI_DQS_DELAY_PS in DLL configuration.

[2.2.2]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 Rule 10.1, 10.3 and 10.4.
  - Updated _flexspi_command from named enumerator into anonymous enumerator.

[2.2.1]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 Rule 10.1, 10.3, 10.4, 10.8, 11.9, 14.4, 15.7, 16.4, 17.7, 7.3.
  - Fixed IAR build warning Pe167.
  - Fixed the potential buffer overwrite and Rx FIFO overread issue in FLEXSPI_ReadBlocking.

[2.2.0]

- Bug Fixes
  - Fixed flag name typos: kFLEXSPI_IpTxFifoWatermarkEmpltyFlag to kFLEXSPI_IpTxFifoWatermarkEmptyFlag; kFLEXSPI_IpCommandExcutionDoneFlag to kFLEXSPI_IpCommandExecutionDoneFlag.
  - Fixed comments typos such as sequencen->sequence, levle->level.
  - Fixed FLSHCR2[ARDSEQID] field clean issue.
  - Updated flexspi_config_t structure and FlexSPI_Init to support new feature FSL_FEATURE_FLEXSPI_HAS_NO_MCR0_ATDFEN and FSL_FEATURE_FLEXSPI_HAS_NO_MCR0_ARDFEN.

  - Updated flexspi_flags_t structure to support new feature FSL_FEATURE_FLEXSPI_HAS_INTEN_AHBBUSERROREN.

[2.1.1]

- Improvements
  - Defaulted enable prefetch for AHB RX buffer configuration in FLEXSPI_GetDefaultConfig, which is align with the reset value in AHBRXBUFxCR0.
  - Added software workaround for ERR011377 in FLEXSPI_SetFlashConfig; added some delay after DLL lock status set to ensure correct data read/write.

[2.1.0]

- New Features
  - Added new API FLEXSPI_UpdateRxSampleClock for users to update read sample clock source after initialization.
  - Added reset peripheral operation in FLEXSPI_Init if required.

[2.0.5]

- Bug Fixes
  - Fixed FLEXSPI_UpdateLUT cannot do partial update issue.

[2.0.4]

- Bug Fixes
  - Reset flash size to zero for all ports in FLEXSPI_Init; fixed the possible out-of-range flash access with no error reported.

[2.0.3]

- Bug Fixes
  - Fixed AHB receive buffer size configuration issue. The FLEXSPI_AHBRXBUFCR0_BUFSZ field should configure 64 bits size, and currently the AHB receive buffer size is in bytes which means 8-bit, so the correct configuration should be config->ahbConfig.buffer[i].bufferSize / 8.

[2.0.2]

- New Features
  - Supported DQS write mask enable/disable feature during set FLEXSPI configuration.
  - Provided new API FLEXSPI_TransferUpdateSizeEDMA for users to update eDMA transfer size(SSIZE/DSIZE) per DMA transfer.
- Bug Fixes
  - Fixed invalid operation of FLEXSPI_Init to enable AHB bus Read Access to IP RX FIFO.
  - Fixed incorrect operation of FLEXSPI_Init to configure IP TX FIFO watermark.

[2.0.1]

- Bug Fixes
  - Fixed the flag clear issue and AHB read Command index configuration issue in FLEXSPI_SetFlashConfig.
  - Updated FLEXSPI_UpdateLUT function to update LUT table from any index instead of previous command index.
  - Added bus idle wait in FLEXSPI_SetFlashConfig and FLEXSPI_UpdateLUT to ensure bus is idle before any change to FlexSPI controller.
  - Updated interrupt API FLEXSPI_TransferNonBlocking and interrupt handle flow FLEXSPI_TransferHandleIRQ.
  - Updated eDMA API FLEXSPI_TransferEDMA.

[2.0.0]

- Initial version.

FLEXSPI EDMA Driver

[2.3.4]

- Bug Fixes
  - Fixed CERT INT31-C violations in FLEXSPI_TransferEDMA: added explicit uint32_t casts and masks before narrowing to uint8_t for handle->nbytes and FLEXSPI_CalculatePower arguments.

[2.3.3]

- Bug Fixes
  - Fixed FLEXSPI_TransferEDMA bug that, the DMA channel not configured correctly when using kFLEXSPI_Read.

[2.3.2]

- Bug Fixes
  - Fixed the bug that internal variable s_edmaPrivateHandle overflows when using FlexSPI2.

[2.0.2]

- New Features
  - Provided new API FLEXSPI_TransferUpdateSizeEDMA for users to update eDMA transfer size(SSIZE/DSIZE) per DMA transfer.

[2.0.0]

- Initial version.

GPC

[2.1.1]

- Bug Fixes
  - Moved the assert sentence that irq register number has to be greater than 0 to platforms which irq 0-31 is not available.
  - Fixed the violations of MISRA C-2012 rules:
    * Rule 10.7 12.2.

[2.1.0]

- Improvements
  - Updated driver for IMXRT.

[2.0.0]

- Initial version.

GPIO

[2.0.7]

- Bug Fixes
  - Fixed coverity MSG issues with CERT INT30-C compliance.

[2.0.6]

- Bug Fixes
  - Fixed compile warning: 'GPIO_GetInstance' defined but not used when macro FSL_SDK_DISABLE_DRIVER_CLOCK_CONTROL is defined.

[2.0.5]

- Bug Fixes
  - Fixed MISRA C-2012 issue: rule-17.7.

[2.0.4]

- Improvements
  - Updated the GPIO_PinWrite to use atomic operation if possible.
- Bug Fixes
  - Fixed GPIO_PortToggle bug with platforms don't have register DR_TOGGLE.

[2.0.3]

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

[2.0.2]

- Bug Fixes
  - Fixed the bug of enabling wrong GPIO clock gate in initial API. Since some GPIO instances may not have a clock gate enabled, it checks the clock gate number and makes sure the clock gate is valid.

[2.0.1]

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

[2.0.0]

- Initial version.

GPT

[2.0.6]

- Bug Fixes
  - Fix CERT INT30-C issues.

[2.0.5]

- Improvements
  - Support workaround for ERR003777. This workaround helps switching the clock sources.

[2.0.4]

- Bug Fixes
  - Fixed compiler warning when built with FSL_SDK_DISABLE_DRIVER_CLOCK_CONTROL flag enabled.

[2.0.3]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 5.3 by customizing function parameter.

[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 10.1, 10.3, 10.4, 10.6, 10.8, 17.7.

[2.0.0]

- Initial version.

IOMUXC

[2.0.1]

- Doxygen improvement.

[2.0.0]

- initial version.

KPP

[2.1.1]

- Bug Fixes
  - Fixed coverity MSG issues with CERT INT30-C, CERT ARR30-C compliance.

[2.1.0]

- Improvements
  - Optimize rowData debounce method to adapt to multi-key detection
  - Modify the KPP_keyPressScanning type to status_t.

[2.0.1]

- Bug Fixes
  - Fixed the violations of MISRA 2012 rules:
    - Rule 10.3 10.4 10.6 14.4 17.7

[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

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.

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

OCOTP

[2.1.5]

- Improvements
  - Renamed API functions to clarify that they operate on fuse words, not shadow registers:
    - OCOTP_ReadFuseShadowRegister -> OCOTP_ReadFuse
    - OCOTP_ReadFuseShadowRegisterExt -> OCOTP_ReadFuseExt
    - OCOTP_WriteFuseShadowRegister -> OCOTP_WriteFuse
    - OCOTP_WriteFuseShadowRegisterWithLock -> OCOTP_WriteFuseWithLock
  - Added backward compatibility macros for old API names, which will be removed in later releases.

[2.1.4]

- Bug fixes
  - Fixed the bug that OCOTP_ReadFuseShadowRegisterExt can't read more than one word.

[2.1.3]

- Bug fixes
  - Fixed MISRA 2012 issue: 8.4, 10.3, 10.4, 14.3.
  - Fixed doxygen warning.

[2.1.2]

- Improvements
  - Updated for new MIMXRT117X header file.

[2.1.1]

- Improvements
  - Updated OCOTP_ReloadShadowRegister to return error status.
  - Added functions OCOTP_ReadFuseShadowRegisterExt and OCOTP_WriteFuseShadowRegisterWithLock.
- Bug fixes
  - Fixed MISRA 2012 rule 10.3 issue.

[2.0.1]

- Bug Fixes
  - Fixed doxygen issues.

[2.0.0]

- Initial version.

PIT

[2.2.0]

- Bug Fixes
  - According to ERR050763, PIT_LDVAL_STAT register is not reliable in dynamic load mode, so remove the status check in PIT_SetRtiTimerPeriod which added since 2.1.1.
  - Removed not used bit PIT_RTI_TCTRL_CHN_MASK.
- Improvements
  - Added more guide about get RTI load status in PIT_SetRtiTimerPeriod's API comment.
  - Change PIT_RTI_Deinit to inline API.
  - Ensure PIT peripheral clock enabled in PIT_RTI_Init.
- New Features
  - Added PIT_ClearRtiSyncStatus API to clear the RTI_LDVAL_STAT register.

[2.1.1]

- Bug Fixes
  - Enable PIT when using RTI to ensure RTI can work properly in debug mode.
- Improvements
  - Added status check in PIT_SetRtiTimerPeriod to ensure the load value is synchronized into the RTI clock domain.
  - Added note for PIT_RTI_Init to remind users wait RTI sync.

[2.1.0]

- New Features
  - Support RTI (Real Time Interrupt) timer.

[2.0.5]

- Improvements
  - Support workaround for ERR007914. This workaround guarantee the write to MCR register is not ignored.

[2.0.4]

- Bug Fixes
  - Fixed PIT_SetTimerPeriod implementation, the load value trigger should be PIT clock cycles minus 1.

[2.0.3]

- Bug Fixes
  - Clear all status bits for all channels to make sure the status of all TCTRL registers is clean.

[2.0.2]

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

[2.0.1]

- Bug Fixes
  - Cleared timer enable bit for all channels in function PIT_Init() to make sure all channels stay in disable status before setting other configurations.
  - Fixed MISRA-2012 rules.
    - Rule 14.4, rule 10.4.

[2.0.0]

- Initial version.

PMU

[2.1.1]

- Bug Fixes
  - Fixed the violations of MISRA 2012 rules: Rule 10.1 10.4

[2.1.0]

- Improvements
  - Added feature macros for low power control APIs to support conditional compile.
  - Renamed “PMU_2P1EnablePullDown” to “PMU_2P5EnablePullDown”.

[2.0.0]

- Initial version.

PWM

[2.10.1]

- Bug Fixes
  - Fixed MISRA C-2012 Rule 17.7, 20.6, 16.3, 10.8

[2.10.0]

- Improvements
  - Added API $\mathrm{PWM_EnableLocalForce}$ to perform a software-controlled local force initialization, causing a FORCE_OUT event that latches all pending FORCE-buffered shadow registers (DTSRCSEL, MCTRL[IPOL], SWCOUT) into their active registers.

  - Added API PWM_UpdateCurrentPolarity to selecting which VAL comparator pair drives the complementary output pair. The change takes effect only after a subsequent FORCE_OUT event (e.g. via PWM_EnableLocalForce).

[2.9.4]

- Bug Fixes
  - Fixed CERT INT31-C issues.

[2.9.3]

- Improvements
  - Cherry-pick patch from hal_nxp repository. Enhanced input validation in PWM_SetupPwm by returning kStatus_InvalidArgument when the calculated pulse count exceeds UINT16_MAX, to prevent silent failures when assertions are disabled in production builds.

[2.9.2]

- Improvements
  - Add new API PWM_GetInputCaptureValue to read the value captured from the submodule counter.

[2.9.1]

- Improvements
  - Add new API PWM_SetupFaultsExt and PWM_SetupFaultInputFilterExt to support FlexPWM which has more than one fault input channels.
  - Support fault 4-7 interrupt and its flag.
- Bug Fixes
  - Fixed violations of the CERT INT31-C.

[2.9.0]

- Improvements
  - Support PWMX channel output for edge aligned PWM.
  - Forbid submodule 0 counter initialize with master sync and master reload mode.
  - Clarify kPWM_BusClock meaning.
  - Verify pulseCnt within 65535 when update period register.

[2.8.4]

- Improvements
  - Support workaround for ERR051989. This function helps realize no phase delay between submodule 0 and other submodule.

[2.8.3]

- Bug Fixes
  - Fixed MISRA C-2012 Rule 15.7

[2.8.2]

- Bug Fixes
  - Fixed warning conversion from 'int' to 'uint16_t' on API PWM_Init.
  - Fixed warning unused variable 'reg' on API PWM_SetPwmForceOutputToZero.

[2.8.1]

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

[2.8.0]

- Improvements
  - Added API PWM_UpdatePwmPeriodAndDutycycle to update the PWM signal's period and dutycycle for a PWM submodule.
  - Added API PWM_SetPeriodRegister and PWM_SetDutycycleRegister to merge duplicate code in API PWM_SetupPwm, PWM_UpdatePwmDutycycleHighAccuracy and PWM_UpdatePwmPeriodAndDutycycle

[2.7.1]

- Improvements
  - Supported UPDATE_MASK bit in MASK register.

[2.7.0]

- Improvements
  - Supported platforms which don't have Capture feature with channel A and B.
  - Supported platforms which don't have Submodule 3.
  - Added assert function in API PWM_SetPhaseDelay to prevent wrong argument.

[2.6.1]

- Bug Fixes
  - Fixed violations of MISRA C-2012 rules: 10.3.

[2.6.0]

- Improvements
  - Added API PWM_SetPhaseDelay to set the phase delay from the master sync signal of submodule 0.
  - Added API PWM_SetFilterSampleCountthe to set number of consecutive samples that must agree prior to the input filter.

  - Added API PWM_SetFilterSamplePeriod to set set the sampling period of the fault pin input filter.

[2.5.1]

- Bug Fixes
  - Fixed MISRA C-2012 rules: 10.1, 10.3, 10.4 , 10.6 and 10.8.
  - Fixed the issue that PWM_UpdatePwmDutycycle() can't update duty cycle status value correct.

[2.5.0]

- Improvements
  - Added API PWM_SetOouputToIdle to set pwm channel output to idle.
  - Added API PWM_GetPwmChannelState to get the pwm channel output duty cycle value.
  - Added API PWM_SetPwmForceOutputToZero to set the pwm channel output to zero logic.
  - Added API PWM_SetChannelOutput to set the pwm channel output state.
  - Added API PWM_SetClockMode to set the value of the clock prescaler.
  - Added API PWM_SetupPwmPhaseShift to set PWM which a special phase shift and 50% duty cycle.
  - Added API PWM_SetVALxValue/PWM_GetVALxValue to set/get PWM VALs registers values directly.

[2.4.0]

- Improvements
  - Supported the PWM which can't work in wait mode.

[2.3.0]

- Improvements
  - Add PWM output enable&disbale API for SDK.
- Bug Fixes
  - Fixed changing channel B configuration when parameter is kPWM_PWMX and PWMX configuration is not supported yet.

[2.2.1]

- Bug Fixes
  - Fixed violations of MISRA C-2012 rules: 10.3, 10.4.
- Bug Fixes
  - Fixed the issue that PWM drivers computed VAL1 improperly.
- Improvements
  - Updated calculation accuracy of reloadValue in dutyCycleToReloadValue function.

[2.2.0]

- Improvements
  - Added new enumeration and two APIs to support enabling and disabling one or more PWM output triggers.
  - Added a new function to make the most of 16-bit resolution PWM.
  - Added one API to support updating fault status of PWM output.
  - Added one API to support PWM DMA write request.
  - Added three APIs to support PWM DMA capture read request.
  - Added one API to support get default fault config of PWM.
  - Added one API to support setting PWM fault disable mapping.

[2.1.0]

- Improvements
  - Moved the configuration of fault input filter into a new API to avoid be initialized multiple times.
- Bug Fixes
  - MISRA C-2012 issue fixed.
    - Fix rules, containing: rule-10.2, rule-10.3, rule-10.4, rule-10.7, rule-10.8, rule-14.4, rule-16.4.

[2.0.1]

- Bug Fixes
  - Fixed the issue that PWM submodule may be initialized twice in function PWM_SetupPwm().

[2.0.0]

- Initial version.

QTMR

[2.3.3]

- Improvements
  - Added support for platforms without Timer Channel DMA Enable Register using FSL_FEATURE_TMR_HAS_NO_DMA_REGISTER.

[2.3.2]

- Bug Fixes
  - Fixed race condition in QTMR_ClearStatusFlags() where a status flag raised between the SCTRL/CSCTRL register read and write-back could be accidentally cleared even if not included in the mask parameter.

[2.3.1]

- Bug Fixes
  - Fixed CERT INT31-C and INT30-C violations.

[2.3.0]

- Improvements
  - Support for platforms which QTMR registers are 32-bit.

[2.2.2]

- Bug Fixes
  - Fixed violations of MISRA C-2012 rules: 10.1, 10.8.

[2.2.1]

- Bug Fixes
  - Fixed violations of MISRA C-2012 rules: 10.1, 10.8.

[2.2.0]

- Improvements
  - Added API QTMR_SetPwmOutputToIdle to set the generated pwm signal to the configured idle value.
  - Added API QTMR_GetPwmOutputStatus to return the output status of the generated pwm signal.
  - Added API QTMR_GetPwmChannelStatus to return the channel dutycycle value.
  - Added API QTMR_SetPwmClockMode to set clock mode change peripheral clock frequency.
- Bug Fixes
  - Fixed the issue that pwm duty cycle could not be 0 and 100.

[2.1.0]

- Bug Fixes
  - Fixed the issue QTMR_SetTimerPeriod needs to decrement down count by 1, and added new APIs to configure the LOAD register, COMP register.

[2.0.2]

- Bug Fixes
  - Fixed the issue introduced by previous code correction for improving the output signal accuracy.

[2.0.1]

- Bug Fixes
  - Fixed violations of MISRA C-2012 rules: 10.1, 10.3, 11.5, 11.9.
- Improvements
  - Improved the output signal accuracy.

[2.0.0]

- Initial version.

RTWDOG

[2.1.4]

- Bug Fixes
  - Fixed CERT INT30-C, INT31-C issue.
  - Make API RTWDOG_CountToMesec return 0 if result overflow.

[2.1.3]

- Improvements
  - Waited the over status after CS register operation in case next CS operation causes problem.

[2.1.2]

- Bug Fixes
  - Fixed doxygen issue.

[2.1.1]

- Bug Fixes
  - MISRA C-2012 issue fixed.
    - Fixed rules, containing: rule-10.3, rule-10.8, rule-11.9, rule-14.4, rule-15.5.

[2.1.0]

- Improvements
  - Added an API to enable or disable the window mode.
  - Added an API to convert a raw count value to millisecond.
  - Used AT_QUICKACCESS_SECTION_CODE macro to decorate RTWDOG_Init, and copied this function from flash to QUICKACCESS section.

[2.0.1]

- Bug Fixes
  - Fixed bug in the RTWDOG_Init; added check for register's unlock status when configuring the RTWDOG in RTWDOG_init.

[2.0.0]

- Initial version.

SAI

[2.4.11]

- Bug Fixes
  - Fixed MSG findings for platforms with multiple FIFO size support.

[2.4.10]

- Improvements
  - Allow enabling/disabling implicit channel configuration.
  - Allow NULL FIFO watermark.
- Bug Fixes
  - Fix compilation warnings when asserts are disabled

[2.4.9]

- Added Errata ERR051421 workaround.

[2.4.8]

- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 10.1, 10.3, 10.4, 10.5, 10.6, 10.7, 10.8, 12.4.

[2.4.7]

- Added conditional support for bit clock swap feature
- Added common IRQ handler entry SAI_DriverIRQHandler.

[2.4.6]

- Bug Fixes
  - Fixed the IAR build warning.

[2.4.5]

- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 10.1, 10.3, 10.4, 10.5, 10.6, 10.7, 10.8, 12.4.

[2.4.4]

- Bug Fixes
  - Fixed enumeration sai_fifo_combine_t - add RX configuration.

[2.4.3]

- Bug Fixes
  - Fixed enumeration sai_fifo_combine_t value configuration issue.

[2.4.2]

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

[2.4.1]

- Bug Fixes
  - Fixed bitWidth incorrectly assigned issue.

[2.4.0]

- Improvements
  - Removed deprecated APIs.

[2.3.8]

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

[2.3.7]

- Improvements
  - Change feature “FSL_FEATURE_SAI_FIFO_COUNT” to “FSL_FEATURE_SAI_HAS_FIFO”.
  - Added feature “FSL_FEATURE_SAI_FIFO_COUNTn(x)” to align SAI fifo count function with IP in function

[2.3.6]

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

[2.3.5]

- Improvements
  - Make driver to be aarch64 compatible.

[2.3.4]

- Bug Fixes
  - Corrected the fifo combine feature macro used in driver.

[2.3.3]

- Bug Fixes
  - Added bit clock polarity configuration when sai act as slave.
  - Fixed out of bound access coverity issue.
  - Fixed violations of MISRA C-2012 rule 10.3, 10.4.

[2.3.2]

- Bug Fixes
  - Corrected the frame sync configuration when sai act as slave.

[2.3.1]

- Bug Fixes
  - Corrected the peripheral name in function SAI0_DriverIRQHandler.
  - Fixed violations of MISRA C-2012 rule 17.7.

[2.3.0]

- Bug Fixes
  - Fixed the build error caused by the SOC has no fifo feature.

[2.2.3]

- Bug Fixes
  - Corrected the peripheral name in function SAI0_DriverIRQHandler.

[2.2.2]

- Bug Fixes
  - Fixed the issue of MISRA 2004 rule 9.3.
  - Fixed sign-compare warning.
  - Fixed the PA082 build warning.
  - Fixed sign-compare warning.
  - Fixed violations of MISRA C-2012 rule 10.3,17.7,10.4,8.4,10.7,10.8,14.4,17.7,11.6,10.1,10.6,8.4,14.3,16.4,18
  - Allow to reset Rx or Tx FIFO pointers only when Rx or Tx is disabled.
- Improvements
  - Added 24bit raw audio data width support in sai sdma driver.
  - Disabled the interrupt/DMA request in the SAI_Init to avoid generates unexpected sai FIFO requests.

[2.2.1]

- Improvements
  - Added mclk post divider support in function SAI_SetMasterClockDivider.
  - Removed useless configuration code in SAI_RxSetSerialDataConfig.
- Bug Fixes
  - Fixed the SAI SDMA driver build issue caused by the wrong structure member name used in the function SAI_TransferRxSetConfigSDMA/SAI_TransferTxSetConfigSDMA.
  - Fixed BAD BIT SHIFT OPERATION issue caused by the FSL_FEATURE_SAI_CHANNEL_COUNTn.
  - Applied ERR05144: not set FCONT = 1 when TMR > 0, otherwise the TX may not work.

[2.2.0]

- Improvements
  - Added new APIs for parameters collection and simplified user interfaces:
    - SAI_Init
    - SAI_SetMasterClockConfig
    - SAI_TxSetBitClockRate
    - SAI_TxSetSerialDataConfig
    - SAI_TxSetFrameSyncConfig
    - SAI_TxSetFifoConfig
    - SAI_TxSetBitclockConfig
    - SAI_TxSetConfig
    - SAI_TxSetTransferConfig
    - SAI_RxSetBitClockRate
    - SAI_RxSetSerialDataConfig
    - SAI_RxSetFrameSyncConfig
    - SAI_RxSetFifoConfig
    - SAI_RxSetBitclockConfig
    - SAI_RXSetConfig
    - SAI_RxSetTransferConfig
    - SAI_GetClassicI2SConfig
    - SAI_GetLeftJustifiedConfig
    - SAI_GetRightJustifiedConfig
    - SAI_GetTDMConfig

[2.1.9]

- Improvements
  - Improved SAI driver comment for clock polarity.
  - Added enumeration for SAI for sample inputs on different edges.

  - Changed FSL_FEATURE_SAI_CHANNEL_COUNT to FSL_FEATURE_SAI_CHANNEL_COUNTn(base) for the difference between the different SAI instances.
- Added new APIs:
  - SAI_TxSetBitClockDirection
  - SAI_RxSetBitClockDirection
  - SAI_RxSetFrameSyncDirection
  - SAI_TxSetFrameSyncDirection

[2.1.8]

- Improvements
  - Added feature macro test for the sync mode2 and mode 3.
  - Added feature macro test for masterClockHz in sai_transfer_format_t.

[2.1.7]

- Improvements
  - Added feature macro test for the mclkSource member in sai_config_t.
  - Changed "FSL_FEATURE_SAI5_SAI6_SHARE_IRQ" to "FSL_FEATURE_SAI_SAI5_SAI6_SHARE_IRQ".
  - Added #ifndef #endif check for SAI_XFER_QUEUE_SIZE to allow redefinition.
- Bug Fixes
  - Fixed build error caused by feature macro test for mclkSource.

[2.1.6]

- Improvements
  - Added feature macro test for mclkSourceClockHz check.
  - Added bit clock source name for general devices.
- Bug Fixes
  - Fixed incorrect channel numbers setting while calling RX/TX set format together.

[2.1.5]

- Bug Fixes
  - Corrected SAI3 driver IRQ handler name.
  - Added I2S4/5/6 IRQ handler.
  - Added base in handler structure to support different instances sharing one IRQ number.
- New Features
  - Updated SAI driver for MCR bit MICS.
  - Added 192 KHZ/384 KHZ in the sample rate enumeration.
  - Added multi FIFO interrupt/SDMA transfer support for TX/RX.
  - Added an API to read/write multi FIFO data in a blocking method.
  - Added bclk bypass support when bclk is same with mclk.

[2.1.4]

- New Features
  - Added an API to enable/disable auto FIFO error recovery in platforms that support this feature.
  - Added an API to set data packing feature in platforms which support this feature.

[2.1.3]

- New Features
  - Added feature to make I2S frame sync length configurable according to bitWidth.

[2.1.2]

- Bug Fixes
  - Added 24-bit support for SAI eDMA transfer. All data shall be 32 bits for send/receive, as eDMA cannot directly handle 3-Byte transfer.

[2.1.1]

- Improvements
  - Reduced code size while not using transactional API.

[2.1.0]

- Improvements
  - API name changes:
    - SAI_GetSendRemainingBytes -> SAI_GetSentCount.
    - SAI_GetReceiveRemainingBytes -> SAI_GetReceivedCount.
    - All names of transactional APIs were added with “Transfer” prefix.
    - All transactional APIs use base and handle as input parameter.
    - Unified the parameter names.
- Bug Fixes
  - Fixed WLC bug while reading TCSR/RCSR registers.
  - Fixed MOE enable flow issue. Moved MOE enable after MICS settings in SAI_TxInit/SAI_RxInit.

[2.0.0]

- Initial version.

SAI_EDMA

[2.7.4]

- Bug Fixes
  - Fixed MSG finding for platforms with multiple FIFO size support.

[2.7.3]

- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 10.1, 10.3, 10.4, 10.5, 10.6, 10.7, 10.8, 12.4.

[2.7.2]

- Improvements
  - Add macros MCUX_SDK_SAI_EDMA_TX_ENABLE_INTERNAL and MCUX_SDK_SAI_EDMA_RX_ENABLE_INTERNAL to let the user decide whether to enable SAI when calling SAI_TransferSendEDMA/SAI_TransferReceiveEDMA.

[2.7.1]

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

[2.7.0]

- Improvements
  - Updated api SAI_TransferReceiveEDMA to support voice channel block interleave transfer.
  - Updated api SAI_TransferSendEDMA to support voice channel block interleave transfer.
  - Added new api SAI_TransferSetInterleaveType to support channel interleave type configurations.

[2.6.0]

- Improvements
  - Removed deprecated APIs.

[2.5.1]

- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 20.7.

[2.5.0]

- Improvements
  - Added new api SAI_TransferSendLoopEDMA/SAI_TransferReceiveLoopEDMA to support loop transfer.
  - Added multi sai channel transfer support.

[2.4.0]

- Improvements
  - Added new api SAI_TransferGetValidTransferSlotsEDMA which can be used to get valid transfer slot count in the sai edma transfer queue.
  - Deprecated the api SAI_TransferRxSetFormatEDMA and SAI_TransferTxSetFormatEDMA.
- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 10.3,10.4.

[2.3.2]

- Refer SAI driver change log 2.1.0 to 2.3.2

SEMC

[2.8.1]

- Bug Fixes
  - Fixed CERT-C INT30-C violation in SEMC_ConvertTiming: replaced (time_ps + tClk_ps - 1) ceiling-division formula (which could overflow uint64_t) with an overflow-safe equivalent using quotient plus remainder check.

[2.8.0]

- Bug Fixes
  - Fixed IOCR pinmux field masking so SEMC_ConfigureSRAMWithChipSelection() can reliably configure SRAM CE on chip selects other than CS0.
  - Extended SRAM timing parameters in semc_sram_config_t to 32-bit and updated timing conversion to safely handle values greater than 255 ns.
  - Guard readback in SEMC_SendIPCommand() and return kStatus_InvalidArgument when a read command is issued with a null output pointer

[2.7.2]

- Bug Fixes
  - Add assertions to prevent unsigned integer overflow in increment operations
  - Add assertions to prevent unsigned integer overflow in multiplication calculations
  - Add safe type conversion with temporary variables for uint32_t to uint16_t and uint8_t operations
  - Add explicit boolean to integer conversions (? 1U : 0U) for enable flags
  - Add bounds checking for narrowing conversions in prescale and dataSize calculations
  - Follow INT30-C and INT31-C CERT coding standards for safe integer operations

[2.7.1]

- Bug Fixes
  - Fixed the wrong write operation to INTR register. The INTR register is a W1C register, so the right write operation is write directly to it to clear.

[2.7.0]

- Improvements
  - Add new autofreshTimes parameter in semc_sdram_config_t.
- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 10.4.

[2.6.0]

- Bug Fixes
  - Fixed the SEMC SRAM function bug that some configuration options can't be set.
  - Correct legacy SEMC SRAM function feature macros.
- Improvements
  - Add new SEMC SRAM function feature macros.

[2.5.1]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 Rule 14.3.
  - Fixed SEMC_ConfigureDBI bug that RDX not set correctly.

[2.5.0]

- Bug Fixes
  - Fixed definitions of bitfields of BMCR0 and BMCR1 - wrong field order and incorrect semantical naming
  - The fix alters the driver API regarding configuration of AXI bus queue reordering

[2.4.3]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 Rule 5.6.

[2.4.2]

- Improvements
  - Deleted meaningless parameter in memory size conversion function.

[2.4.1]

- Bug Fixes
  - Fixed PSRAM A8 configuration issue, which should be 0x06U for PSRAM while pix mux bit width is 0x04U, based on different pix mux bit width.

[2.4.0]

- Improvements
  - Improved nor and sram timing configuration on sync mode.

[2.3.1]

- Bug Fixes
  - Updated refresh timer period(RT) timing setting, which updated into (RT+1)*(Prescaler period) for SDRAM.
  - Supported new DBI control register 2 to configure CSX interval time(CEITV).
  - Fixed violations of the MISRA C-2012 Rule 10.8.
  - Fixed doxygen warning.

[2.3.0]

- New Features
  - Limited burst length as 1 according to ERR050577, Auto-refresh command may possibly fail to be triggered during long time back-to-back write (or read) when SDRAM controller's burst length is greater than 1.
  - Supported 8 bits column address for SDRAM.

[2.2.1]

- New Features
  - Added queue weight control, which can control queue a/b is working or not.
  - Updated NAND FLASH configuration API which disables and enables SEMC between configure control registers.
  - Added ONFI parameter Integrity CRC check for SEMC flash component.

[2.2.0]

- New Features
  - Supported up to 4 PSRAM CS.
  - Added programmable delay line for DQS.
  - Added ready/wait feature for SRAM in asynchronous mode.

[2.1.0]

- Bug Fixes
  - MISRA C-2012 issue fixed: rule 10.3, 10.4, and 14.4.
  - Updated parameter type from uint16_t into uint32_t for send IP command API.

[2.0.4]

- Bug Fixes
  - Fixed the SEMC queueA and queueB weight configuration issue.
  - Fixed the wrong configuration of DBICR1 register in SEMC_ConfigureDBI.

[2.0.3]

- Bug Fixes
  - Added feature macro to control WDS&WDH bit setting for NOR synchronous transfer.

[2.0.2]

- Bug Fixes
  - Changed SEMC NAND configuration structure and verify SEMC NAND related APIs.
  - Added extended SEMC clock enable.

[2.0.1]

- Bug Fixes
  - Fixed data size mask configure in SEMC_ConfigureIPCommand API.
  - Updated the command mode in IP command type.

[2.0.0]

- Initial version.

SNVS_HP

[2.3.2]

- Make SNVS_HP_RTC_Init()/SNVS_HP_RTC_Deinit more transparent. Use function SNVS_HP_Init()/SNVS_HP_Deinit() instead of copy of this code in SNVS_HP_RTC_XXX() function.

[2.3.1]

- Fixed problem in SNVS_HP_RTC_Init(), which is clearing bits that should stay intact.

[2.3.0]

- Re-map Security Violation for RT11xx specific violations.

[2.2.0]

- Fixed doxygen issues.
- Add SNVS HP Set locks.

[2.1.4]

- Fix MISRA issues.

[2.1.3]

- Fixed IAR Pa082 warnings.

[2.1.2]

- Fixed problem with initialization of the periodic interrupt frequency.
- Fixed problem with SNVS entering into fail state when HAB enters closed mode.

[2.1.1]

- Added APIs for HP security violation status flags.

[2.1.0]

- Added APIs for High Assurance Counter (HAC), Zeroizable Master Key (ZMK) and Software Security Violation.

[2.0.0]

- Initial version.

SNVS_LP

[2.4.6]

- Fix a bug in SNVS_LP_EnableRxActiveTamper() where assignments to base->LPATRC2R were done wrongly to LPATRC1R.

[2.4.5]

- Fix a bug in SNVS_LP_EnableRxActiveTamper() where assignments to base->LPATRC1R would overwrite previously set bits.

[2.4.4]

- Make SNVS_LP_SRTC_Init()/SNVS_LP_SRTC_Deinit more transparent. Use function SNVS_LP_Init()/SNVS_LP_Deinit() instead of copy of this code in SNVS_LP_SRTC_XXX() function.

[2.4.3]

- Fixed problem in SNVS_LP_SRTC_Init(), which is clearing bits that should stay intact.

[2.4.2]

- Updated driver to match with new device header files.

[2.4.1]

- Fixed MISRA issues.

[2.4.0]

- Fix backward compatibility with version 2.2.x.

[2.3.0]

- Add active pin, clock, voltage and temperature tamper features.

[2.2.0]

- Fixed doxygen issues.
- Add Transition SNVS SSM state to Trusted/Non-secure from Check state.

[2.1.2]

- Fix MISRA issues.

[2.1.1]

- Fix IAR Pa082 warning.

[2.1.0]

- Added APIs for Zeroizable Master Key (ZMK) and Monotonic Counter (MC).

[2.0.0]

- Initial version.

SPDIF

[2.0.10]

- Improvements
  - Added SPDIF_CopyRxChannelStatusToTx() to share the interrupt transfer channel-status copy flow with the SPDIF_XCVR driver.

[2.0.9]

- New Features
  - Added common IRQ handler entry SPIDIF_CommonDriverIRQHandler.

[2.0.8]

- Bug fixes
  - Fixed MSG cert int30-c and int31-c findings

[2.0.7]

- Improvements
  - Add feature macro FSL_FEATURE_SPDIF_HAS_NO_SIC_REGISTER to handle nonexistent SIC register.

[2.0.6]

- Bug Fixes
  - Fixed the Q/U channel interrupt enabled unexpectly while Q/U transfer pointer is NULL.

[2.0.5]

- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 11.3.

[2.0.4]

- Bug Fixes
  - Added udata/qdata buffer address validation in driver IRQ handler to ensure that NULL pointer dereferences do not occur.

[2.0.3]

- Bug Fixes
  - MISRA C-2012 issue fixed: rule 10.3, 10.4, and 14.4.

[2.0.2]

- Bug Fixes
  - Corrected operator used for size value assertion in SPDIF_ReadBlocking/SPDIF_WriteBlocking.

[2.0.1]

- Bug Fixes
  - Corrected the feature macro name used to define s_edmaPrivateHandle.

[2.0.0]

- Initial version.

SPDIF DMA Driver

[2.0.9]

- Bug fixes
  - Fixed MSG cert int31-c findings

[2.0.8]

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

[2.0.7]

- Bug Fixes
  - Fixed the incompatibility issue with edma4 driver.

[2.0.6]

- Bug Fixes
  - Add feature macro to determine whether to use the API MEMORY_ConvertMemoryMapAddress to translate TCD addresses for DLAST_SGA.

[2.0.5]

- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 11.3.

[2.0.4]

- Bug Fixes
  - Added udata/qdata buffer address validation in driver IRQ handler to ensure that NULL pointer dereferences do not occur.

[2.0.3]

- Bug Fixes
  - MISRA C-2012 issue fixed: rule 10.3, 10.4, and 14.4.

[2.0.2]

- Bug Fixes
  - Corrected operator used for size value assertion in SPDIF_ReadBlocking/SPDIF_WriteBlocking.

[2.0.1]

- Bug Fixes
  - Corrected the feature macro name used to define s_edmaPrivateHandle.

[2.0.0]

- Initial version.

SRC

[2.0.1]

- Improvements
  - Updated SRC driver for adding SRC_SRSR_JTAG_SW_RST enumeration.

[2.0.0]

- Initial version.

TEMPMON

[2.2.1]

- Improvements
  - Move TEMPMON calibration data mask definitions from the driver source file to the header file, as the driver examples also need to use these definitions.

[2.2.0]

- Bug Fixes
  - Fixed the issue of inconsistency between data width and RM.

[2.1.1]

- Bug Fixes
  - Fixed the violations of MISRA C-2012 rules:
    - Rule 10.3 10.4.

[2.1.0]

- Bug Fixes
  - Supported minus value for alarm temperature setting.
  - Fixed wrong temperature calculation equation.

[2.0.3]

- Improvements
  - Added temperature threshold check for high/low/panic to avoid temperature overflow.

[2.0.2]

- Bug Fixes
  - Fixed wrong alarm value setting API, it need to clear it firstly and set a new value into it.

[2.0.1]

- Bug Fixes
  - Fixed the violations of MISRA C-2012 rules:
    - Rule 10.1 10.3 10.4 10.8 17.7.

[2.0.0]

- Initial version.

TRNG

[2.0.23]

- New features:
  - Updated TRNG default values for MCXA577 devices based on silicon characterization.

[2.0.22]

- New features:
  - Added support for KW43 and MCXW70 devices.

[2.0.21]

- New features:
  - Added support for MCXC devices.

[2.0.20]

- New features:
  - Added support for MCXA devices.

[2.0.19]

- New features:
  - Added support for MCXA and MCXL.

[2.0.18]

- Bug fix:
  - TRNG health checks now done in software on RT5xx and RT6xx.

[2.0.17]

- New features:
  - Add support for RT700.

[2.0.16]

- Improvements:
  - Added support for Dual oscillator mode.

[2.0.15]

- Other changes:
  - Changed TRNG_USER_CONFIG_DEFAULT_XXX values according to latest reccomended by design team.

[2.0.14]

- New features:
  - Add support for RW610 and RW612.

[2.0.13]

- Bug fix:
  - After deepsleep it might return error, added clearing bits in TRNG_GetRandomData() and generating new entropy.
  - Modified reloading entropy in TRNG_GetRandomData(), for some data length it doesn't reloading entropy correctly.

[2.0.12]

- Bug fix:
  - For KW34A4_SERIES, KW35A4_SERIES, KW36A4_SERIES set TRNG_USER_CONFIG_DEFAULT_OSC_DIV to kTRNG_RingOscDiv8.

[2.0.11]

- Bug fix:
  - Add clearing pending errors in TRNG_Init().

[2.0.10]

- Bug Fix:
  - Fixed doxygen issues.

[2.0.9]

- Bug Fix:
  - Fix HIS_CCM metrics issues.

[2.0.8]

- Bug fix:
  - For K32L2A41A_SERIES set TRNG_USER_CONFIG_DEFAULT_OSC_DIV to kTRNG_RingOscDiv4.

[2.0.7]

- Bug fix:
  - Fix MISRA 2004 issue rule 12.5.

[2.0.6]

- Bug fix:
  - For KW35Z4_SERIES set TRNG_USER_CONFIG_DEFAULT_OSC_DIV to kTRNG_RingOscDiv8.

[2.0.5]

- Improvements:
  - For FRQMIN, FRQMAX and OSCDIV, add possibility to use device specific preprocessor macro to define default value in TRNG user configuration structure.

[2.0.4]

- Bug Fix:
  - Fix MISRA-2012 issues.
    - Rule 10.1, rule 10.3, rule 13.5, rule 16.1.

[2.0.3]

- Improvements:
  - update TRNG_Init to restart new entropy generation.

[2.0.2]

- Improvements:
  - fix MISRA issues
    - Rule 14.4.

[2.0.1]

- New features:
  - Set default OSCDIV for Kinetis devices KL8x and KL28Z.
- Other changes:
  - Changed default OSCDIV for K81 to divide by 2.

[2.0.0]

- Initial version.

USDHC

[2.8.10]

- Bug Fixes
  - Fixed some quality issues.

[2.8.9]

- Bug Fixes
  - Fixed some quality issues.
  - Do not invalidate cache on non-blocking non-DMA transfer.
- New Features
  - Added common IRQ handler entry USDHC_DriverIRQHandler.

[2.8.8]

- Bug Fixes
  - Fixed build issue with armgcc O3.

[2.8.7]

- Bug Fixes
  - Disabled CMD error check for standard tuning per RM.

[2.8.6]

- Bug Fixes
  - Invalidate cache after blocking read.

[2.8.5]

- Improvements
  - Enable the driver to be AARCH64 compatible.

[2.8.4]

- Improvements
  - Add feature macro FSL_FEATURE_USDHC_HAS_NO_VS18.

[2.8.3]

- Improvements
  - Improved api USDHC_EnableAutoTuningForCmdAndData to adapt to new bit field name for USDHC_VEND_SPEC2 register.

[2.8.2]

- Improvements
  - Added feature macro FSL_FEATURE_USDHC_HAS_NO_VOLTAGE_SELECT.

[2.8.1]

- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 11.9.

[2.8.0]

- Improvements
  - Fixed the mmc boot transfer failed issue which is caused by the Dma complete interrupt not enabled.
  - Marked api USDHC_AdjustDelayForManualTuning as deprecated and added new api USDHC_SetTuingDelay/USDHC_GetTuningDelayStatus.
  - Improved the manual tuning flow accroding to specification.

  - Added memory address conversion to support buffers which could only be accessed using alias address by non-core masters.
  - Fixed violations of MISRA C-2012 rule 10.4.

[2.7.0]

- Improvements
  - Added api USDHC_TransferScatterGatherADMANonBlocking to support scatter gather transfer.
  - Added feature FSL_FEATURE_USDHC_REGISTER_HOST_CTRL_CAP_HAS_NO_RETUNING_TIME_COUNT for re-tuning time counter field in HOST_CTRL_CAP register.
- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 11.9, 10.1, 10.3, 10.4, 8.4.

[2.6.0]

- Improvements
  - Added api USDHC_SetStandardTuningCounter to support adjust tuning counter of Standard tuning.

[2.5.1]

- Improvements
  - Used different status code for command and data interrupt callback.
  - Added cache line invalidate for receive buffer in driver IRQ handler to fix CM7 speculative access issue.

[2.5.0]

- Improvements
  - Added new api USDHC_SetStrobeDllOverride for HS400 strobe dll override mode delay taps configurations.
  - Corrected the STROBE DLL configurations sequence.

[2.4.0]

- Improvements
  - Added feature macro for read/write burst length.
    - Disabled redundant interrupt per different transfer request.
    - Disabled interrupt and reset command/data pointer in handle when transfer completes.
- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 11.9, 15.7, 4.7, 16.4, 10.1, 10.3, 10.4, 11.3, 14.4, 10.6, 17.7, 16.1, 16.3.
  - Fixed PA082 build warning.
  - Fixed logically dead code Coverity issue.

[2.3.0]

- Improvements
  - Added USDHC_SetDataConfig API to support manual tuning.
  - Removed the limitaion that source clock must be bigger than the target in function USDHC_SetSdClock by using source clock frequency as target directly.
  - Added peripheral reset in USDHC_Init function.
  - Added tuning reset support in function USDHC_Reset function.

[2.2.8]

- Bug Fixes
  - Fixed out-of bounds write in function USDHC_ReceiveCommandResponse.

[2.2.7]

- Improvements
  - Added API USDHC_GetEnabledInterruptStatusFlags and used in USDHC_TransferHandleIRQ.
  - Removed useless member interruptFlags in usdhc_handle_t.

[2.2.6]

- Improvements
  - Added address align check for ADMA descriptor table address.
  - Changed USDHC_ADMA1_DESCRIPTOR_MAX_LENGTH_PER_ENTRY to (65536-4096) to make sure the data address is 4KB align for a transfer which need more than one ADMA1 descriptor.

[2.2.5]

- Bug Fixes
  - Fixed MDK 66-D warning.

[2.2.4]

- Bug Fixes
  - Fixed issue that real clock frequency wss mismatched with target clock frequency, which was caused by an incorrect prescaler calculation.
- New Features
  - Added control macro to enable/disable the CLOCK code in current driver.

[2.2.3]

- Bug Fixes
  - Fixed issue where AMDA did not disable with DMAEN clear.
- Improvements
  - Improved set clock function to check the output frequency range.

  - Dynamic set SDCLKFS during DDR enable or disable.

[2.2.2]

- Improvements
  - Improved read transfer cache maintain operation, combined clean, and invalidated them into one function.

[2.2.1]

- Bug Fixes
  - Disabled the invalidate cache operation for tuning.

[2.2.0]

- Improvements
  - Improved USDHC to support MMC boot feature.

[2.1.3]

- Bug Fixes
  - Fixed MISRA issue.

[2.1.2]

- Bug Fixes
  - Fixed Coverity issue.
  - Added base address and userData parameter for all callback functions.

[2.1.1]

- Improvements
  - Added cache maintain operation.
  - Added timeout status check for the DATA transfer which ignore error.
  - Added feature macro for SDR50/SDR104 mode.
  - Removed useless IRQ handler from different platforms.

[2.1.0]

- Improvements
  - Integrated tuning into transfer function.
  - Added strobe DLL feature.
  - Added enableAutoCommand23 in data structure.
  - Removed enable card clock function because the controller would handle the clock on/off.

[2.0.0]

- Initial version.

WDOG

[2.2.2]

- Bug Fixes
  - Added workaround for ERR051322: On affected devices, WDOG_GetDefaultConfig() now defaults enableTimeOutAssert to true, so that the WDOG_B pin is correctly asserted on watchdog timeout. This compensates for the ROM not configuring WCR[WDT] when the WDOG_B_PIN_EN fuse is blown.

[2.2.1]

- Bug Fixes
  - Fixed CERT INT31-C violations.

[2.2.0]

- Bug Fixes
  - Fixed the wrong behavior of workMode.enableWait, workMode.enableStop, workMode.enableDebug in configuration structure wdog_config_t. When set the items to true, WDOG will continues working in those modes.

[2.1.1]

- Bug Fixes
  - MISRA C-2012 issue fixed: rule 10.1, 10.3, 10.4, 10.6, 10.7 and 11.9.
  - Fixed the issue of the inseparable process interrupted by other interrupt source.
    - WDOG_Init
    - WDOG_Refresh

[2.1.0]

- New Features
  - Added new API "WDOG_TriggerSystemSoftwareReset()" to allow users to reset the system by software.
  - Added new API "WDOG_TriggerSoftwareSignal()" to allow users to trigger a WDOG_B signal by software.
  - Removed the parameter "softwareAssertion" and "softwareResetSignal" out of the wdog_config_t structure.
  - Added new parameter "enableTimeOutAssert" to the wdog_config_t structure. With this parameter enabled, when the WDOG timeout occurs, a WDOG_B signal will be asserted. This signal can be routed to external pin of the chip. Note that WDOG_B signal remains asserted until a power-on reset (POR) occurs.

[2.0.1]

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

[2.0.0]

- Initial version.

XBARA

[2.0.7]

- Bug Fixes
  - Fixed CERT-C issues.

[2.0.6]

- Bug Fixes
  - Fixed typo in kXBARA_RequestInterruptEnalbe item.

[2.0.5]

- Bug Fixes
  - Fixed IAR build warning Pa082.
  - Fixed violations of the MISRA C-2012 rules 10.1, 10.3, 10.4, 10.6, 10.7, 10.8, 12.1, 18.1, 20.7.

[2.0.4]

- Improvements
  - Optimized XBARA_SetOutputSignalConfig.

[2.0.3]

- Bug Fixes
  - Corrected configuration for function XBAR_SetOutputSignalConfig.

[2.0.2]

- Other Changes
  - Changed array clock name.

[2.0.1]

- Bug Fixes
  - Fixed w1c bits for XBARA_SetOutputSignalConfig function.

[2.0.0]

- Initial version.

XBARB

[2.0.3]

- Bug Fixes
  - Fixed CERT-C issues.

[2.0.2]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 12.2, 10.7

[2.0.1]

- Bug Fixes
  - Corrected XBARB_SetSignalsConnection function.
- Other Changes
  - Changed array clock name.

[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.

MIMXRT1024

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 FreeMASTER

freemaster

1.8.2 NXP Wi-Fi

Wi-Fi, Bluetooth, 802.15.4

1.8.3 FreeRTOS

FreeRTOS

1.8.4 lwIP

lwIP

1.8.5 File systemFatfs

FatFs

Chapter 2

MIMXRT1024

2.1 ADC: 12-bit Analog to Digital Converter Driver

void ADC_Init(ADC_Type *base, const *adc_config_t* *config)

Initialize the ADC module.

Parameters

- base – ADC peripheral base address.
- config – Pointer to “adc_config_t” structure.

void ADC_Deinit(ADC_Type *base)

De-initializes the ADC module.

Parameters

- base – ADC peripheral base address.

void ADC_GetDefaultConfig(*adc_config_t* *config)

Gets an available pre-defined settings for the converter’s configuration.

This function initializes the converter configuration structure with available settings. The default values are:

```
config->enableAsynchronousClockOutput = true;
config->enableOverWrite =               false;
config->enableContinuousConversion =    false;
config->enableHighSpeed =               false;
config->enableLowPower =                false;
config->enableLongSample =              false;
config->referenceVoltageSource =        kADC_ReferenceVoltageSourceAlt0;
config->samplePeriodMode =              kADC_SamplePeriod2or12Clocks;
config->clockSource =                   kADC_ClockSourceAD;
config->clockDriver =                   kADC_ClockDriver1;
config->resolution =                    kADC_Resolution12Bit;
```

Parameters

- config – Pointer to the configuration structure.

void ADC_SetChannelConfig(ADC_Type *base, uint32_t channelGroup, const *adc_channel_config_t* *config)

Configures the conversion channel.

This operation triggers the conversion when in software trigger mode. When in hardware trigger mode, this API configures the channel while the external trigger source helps to trigger the conversion.

Note that the "Channel Group" has a detailed description. To allow sequential conversions of the ADC to be triggered by internal peripherals, the ADC has more than one group of status and control registers, one for each conversion. The channel group parameter indicates which group of registers are used, for example channel group 0 is for Group A registers and channel group 1 is for Group B registers. The channel groups are used in a "ping-pong" approach to control the ADC operation. At any point, only one of the channel groups is actively controlling ADC conversions. The channel group 0 is used for both software and hardware trigger modes. Channel groups 1 and greater indicate potentially multiple channel group registers for use only in hardware trigger mode. See the chip configuration information in the appropriate MCU reference manual about the number of SC1n registers (channel groups) specific to this device. None of the channel groups 1 or greater are used for software trigger operation. Therefore, writing to these channel groups does not initiate a new conversion. Updating the channel group 0 while a different channel group is actively controlling a conversion is allowed and vice versa. Writing any of the channel group registers while that specific channel group is actively controlling a conversion aborts the current conversion.

Parameters

- base – ADC peripheral base address.
- channelGroup – Channel group index.
- config – Pointer to the "adc_channel_config_t" structure for the conversion channel.

static inline uint32_t ADC_GetChannelConversionValue(ADC_Type *base, uint32_t channelGroup)

Gets the conversion value.

Parameters

- base – ADC peripheral base address.
- channelGroup – Channel group index.

Returns

Conversion value.

static inline uint32_t ADC_GetChannelStatusFlags(ADC_Type *base, uint32_t channelGroup)

Gets the status flags of channel.

A conversion is completed when the result of the conversion is transferred into the data result registers. (provided the compare function & hardware averaging is disabled), this is indicated by the setting of COCOn. If hardware averaging is enabled, COCOn sets only, if the last of the selected number of conversions is complete. If the compare function is enabled, COCOn sets and conversion result data is transferred only if the compare condition is true. If both hardware averaging and compare functions are enabled, then COCOn sets only if the last of the selected number of conversions is complete and the compare condition is true.

Parameters

- base – ADC peripheral base address.
- channelGroup – Channel group index.

Returns

Status flags of channel.return 0 means COCO flag is 0,return 1 means COCOflag is 1.

status_t ADC_DoAutoCalibration(ADC_Type *base)

Automates the hardware calibration.

This auto calibration helps to adjust the plus/minus side gain automatically. Execute the calibration before using the converter. Note that the software trigger should be used during calibration.

Parameters

- base – ADC peripheral base address.

Return values

- kStatus_Success – Calibration is done successfully.
- kStatus_Fail – Calibration has failed.

Returns

Execution status.

void ADC_SetOffsetConfig(ADC_Type *base, const *adc_offest_config_t* *config)

Set user defined offset.

Parameters

- base – ADC peripheral base address.
- config – Pointer to "adc_offest_config_t" structure.

static inline void ADC_EnableDMA(ADC_Type *base, bool enable)

Enables generating the DMA trigger when the conversion is complete.

Parameters

- base – ADC peripheral base address.
- enable – Switcher of the DMA feature. "true" means enabled, "false" means not enabled.

void ADC_SetHardwareCompareConfig(ADC_Type *base, const *adc_hardware_compare_config_t* *config)

Enables the hardware trigger mode.

Configures the hardware compare mode.

The hardware compare mode provides a way to process the conversion result automatically by using hardware. Only the result in the compare range is available. To compare the range, see "adc_hardware_compare_mode_t" or the appopriate reference manual for more information.

Parameters

- base – ADC peripheral base address.
- enable – Switcher of the trigger mode. "true" means hardware tirgger mode,"false" means software mode.
- base – ADC peripheral base address.
- config – Pointer to "adc_hardware_compare_config_t" structure.

void ADC_SetHardwareAverageConfig(ADC_Type *base, *adc_hardware_average_mode_t* mode)

Configures the hardware average mode.

The hardware average mode provides a way to process the conversion result automatically by using hardware. The multiple conversion results are accumulated and averaged internally making them easier to read.

Parameters

- base – ADC peripheral base address.
- mode – Setting the hardware average mode. See “adc_hardware_average_mode_t”.

static inline uint32_t ADC_GetStatusFlags(ADC_Type *base)

Gets the converter’s status flags.

Parameters

- base – ADC peripheral base address.

Returns

Flags’ mask if indicated flags are asserted. See “adc_status_flags_t”.

void ADC_ClearStatusFlags(ADC_Type *base, uint32_t mask)

Clears the converter’s status falgs.

Parameters

- base – ADC peripheral base address.
- mask – Mask value for the cleared flags. See “adc_status_flags_t”.

enum _adc_status_flags

Converter’s status flags.

Values:

enumerator kADC_ConversionActiveFlag

Conversion is active,not support w1c.

enumerator kADC_CalibrationFailedFlag

Calibration is failed,support w1c.

enumerator kADC_AsynchronousWakeupInterruptFlag

Asynchronous wakeup interrupt occurred, support w1c.

enum _adc_reference_voltage_source

Reference voltage source.

Values:

enumerator kADC_ReferenceVoltageSourceAlt0

For external pins pair of VrefH and VrefL.

enum _adc_sample_period_mode

Sample time duration.

Values:

enumerator kADC_SamplePeriod2or12Clocks

Long sample 12 clocks or short sample 2 clocks.

enumerator kADC_SamplePeriod4or16Clocks

Long sample 16 clocks or short sample 4 clocks.

enumerator kADC_SamplePeriod6or20Clocks
Long sample 20 clocks or short sample 6 clocks.

enumerator kADC_SamplePeriod8or24Clocks
Long sample 24 clocks or short sample 8 clocks.

enumerator kADC_SamplePeriodLong12Clcoks
Long sample 12 clocks.

enumerator kADC_SamplePeriodLong16Clcoks
Long sample 16 clocks.

enumerator kADC_SamplePeriodLong20Clcoks
Long sample 20 clocks.

enumerator kADC_SamplePeriodLong24Clcoks
Long sample 24 clocks.

enumerator kADC_SamplePeriodShort2Clocks
Short sample 2 clocks.

enumerator kADC_SamplePeriodShort4Clocks
Short sample 4 clocks.

enumerator kADC_SamplePeriodShort6Clocks
Short sample 6 clocks.

enumerator kADC_SamplePeriodShort8Clocks
Short sample 8 clocks.

enum _adc_clock_source
Clock source.
Values:

enumerator kADC_ClockSourceIPG
Select IPG clock to generate ADCK.

enumerator kADC_ClockSourceIPGDiv2
Select IPG clock divided by 2 to generate ADCK.

enumerator kADC_ClockSourceAD
Select Asynchronous clock to generate ADCK.

enum _adc_clock_drvier
Clock divider for the converter.
Values:

enumerator kADC_ClockDriver1
For divider 1 from the input clock to the module.

enumerator kADC_ClockDriver2
For divider 2 from the input clock to the module.

enumerator kADC_ClockDriver4
For divider 4 from the input clock to the module.

enumerator kADC_ClockDriver8
For divider 8 from the input clock to the module.

enum _adc_resolution
Converter's resolution.

Values:

enumerator kADC_Resolution8Bit
Single End 8-bit resolution.

enumerator kADC_Resolution10Bit
Single End 10-bit resolution.

enumerator kADC_Resolution12Bit
Single End 12-bit resolution.

enum _adc_hardware_compare_mode
Converter hardware compare mode.

Values:

enumerator kADC_HardwareCompareMode0
Compare true if the result is less than the value1.

enumerator kADC_HardwareCompareMode1
Compare true if the result is greater than or equal to value1.

enumerator kADC_HardwareCompareMode2
Value1 <= Value2, compare true if the result is less than value1 Or the result is Greater than value2. Value1 > Value2, compare true if the result is less than value1 And the result is greater than value2

enumerator kADC_HardwareCompareMode3
Value1 <= Value2, compare true if the result is greater than or equal to value1 And the result is less than or equal to value2. Value1 > Value2, compare true if the result is greater than or equal to value1 Or the result is less than or equal to value2.

enum _adc_hardware_average_mode
Converter hardware average mode.

Values:

enumerator kADC_HardwareAverageCount4
For hardware average with 4 samples.

enumerator kADC_HardwareAverageCount8
For hardware average with 8 samples.

enumerator kADC_HardwareAverageCount16
For hardware average with 16 samples.

enumerator kADC_HardwareAverageCount32
For hardware average with 32 samples.

enumerator kADC_HardwareAverageDiasable
Disable the hardware average function.

typedef enum *_adc_status_flags* adc_status_flags_t
Converter's status flags.

typedef enum *_adc_reference_voltage_source* adc_reference_voltage_source_t
Reference voltage source.

typedef enum *_adc_sample_period_mode* adc_sample_period_mode_t
Sample time duration.

typedef enum *_adc_clock_source* adc_clock_source_t
Clock source.

typedef enum *_adc_clock_drvier* adc_clock_driver_t
Clock divider for the converter.

typedef enum *_adc_resolution* adc_resolution_t
Converter's resolution.

typedef enum *_adc_hardware_compare_mode* adc_hardware_compare_mode_t
Converter hardware compare mode.

typedef enum *_adc_hardware_average_mode* adc_hardware_average_mode_t
Converter hardware average mode.

typedef struct *_adc_config* adc_config_t
Converter configuration.

typedef struct *_adc_offest_config* adc_offest_config_t
Converter Offset configuration.

typedef struct *_adc_hardware_compare_config* adc_hardware_compare_config_t
ADC hardware compare configuration.

In kADC_HardwareCompareMode0, compare true if the result is less than the value1. In kADC_HardwareCompareMode1, compare true if the result is greater than or equal to

value1. In kADC_HardwareCompareMode2, Value1 <= Value2, compare true if the result is less than value1 Or the result is Greater than value2. Value1 > Value2, compare true if the result is less than value1 And the result is Greater than value2. In kADC_HardwareCompareMode3, Value1 <= Value2, compare true if the result is greater than or equal to value1 And the result is less than or equal to value2. Value1 > Value2, compare true if the result is greater than or equal to value1 Or the result is less than or equal to value2.

typedef struct *_adc_channel_config* adc_channel_config_t

ADC channel conversion configuration.

FSL_ADC_DRIVER_VERSION

ADC driver version.

Version 2.0.4.

struct _adc_config

#include <fsl_adc.h> Converter configuration.

Public Members

bool enableOverWrite

Enable the overwriting.

bool enableContinuousConversion

Enable the continuous conversion mode.

bool enableHighSpeed

Enable the high-speed mode.

bool enableLowPower

Enable the low power mode.

bool enableLongSample

Enable the long sample mode.

bool enableAsynchronousClockOutput

Enable the asynchronous clock output.

adc_reference_voltage_source_t referenceVoltageSource

Select the reference voltage source.

adc_sample_period_mode_t samplePeriodMode

Select the sample period in long sample mode or short mode.

adc_clock_source_t clockSource

Select the input clock source to generate the internal clock ADCK.

adc_clock_driver_t clockDriver

Select the divide ratio used by the ADC to generate the internal clock ADCK.

adc_resolution_t resolution

Select the ADC resolution mode.

struct _adc_offest_config

#include <fsl_adc.h> Converter Offset configuration.

Public Members

bool enableSigned

if false,The offset value is added with the raw result. if true,The offset value is subtracted from the raw converted value.

uint32_t offsetValue

User configurable offset value(0-4095).

struct _adc_hardware_compare_config

#include <fsl_adc.h> ADC hardware compare configuration.

In kADC_HardwareCompareMode0, compare true if the result is less than the value1. In kADC_HardwareCompareMode1, compare true if the result is greater than or equal to value1. In kADC_HardwareCompareMode2, Value1 <= Value2, compare true if the result is less than value1 Or the result is Greater than value2. Value1 > Value2, compare true if the result is less than value1 And the result is Greater than value2. In kADC_HardwareCompareMode3, Value1 <= Value2, compare true if the result is greater than or equal to value1 And the result is less than or equal to value2. Value1 > Value2, compare true if the result is greater than or equal to value1 Or the result is less than or equal to value2.

Public Members

adc_hardware_compare_mode_t hardwareCompareMode

Select the hardware compare mode. See “adc_hardware_compare_mode_t”.

uint16_t value1

Setting value1(0-4095) for hardware compare mode.

uint16_t value2

Setting value2(0-4095) for hardware compare mode.

struct _adc_channel_config

#include <fsl_adc.h> ADC channel conversion configuration.

Public Members

uint32_t channelNumber

Setting the conversion channel number. The available range is 0-31. See channel connection information for each chip in Reference Manual document.

bool enableInterruptOnConversionCompleted

Generate an interrupt request once the conversion is completed.

2.2 ADC_ETC: ADC External Trigger Control

void ADC_ETC_Init(ADC_ETC_Type *base, const *adc_etc_config_t* *config)

Initialize the ADC_ETC module.

Parameters

- base – ADC_ETC peripheral base address.
- config – Pointer to "adc_etc_config_t" structure.

void ADC_ETC_Deinit(ADC_ETC_Type *base)

De-Initialize the ADC_ETC module.

Parameters

- base – ADC_ETC peripheral base address.

void ADC_ETC_GetDefaultConfig(*adc_etc_config_t* *config)

Gets an available pre-defined settings for the ADC_ETC's configuration. This function initializes the ADC_ETC's configuration structure with available settings. The default values are:

```
config->enableTSCBypass = true;
config->enableTSC0Trigger = false;
config->enableTSC1Trigger = false;
config->TSC0triggerPriority = 0U;
config->TSC1triggerPriority = 0U;
config->clockPreDivider = 0U;
config->XBARtriggerMask = 0U;
```

Parameters

- config – Pointer to "adc_etc_config_t" structure.

void ADC_ETC_SetTriggerConfig(ADC_ETC_Type *base, uint32_t triggerGroup, const *adc_etc_trigger_config_t* *config)

Set the external XBAR trigger configuration.

Parameters

- base – ADC_ETC peripheral base address.
- triggerGroup – Trigger group index.
- config – Pointer to "adc_etc_trigger_config_t" structure.

void ADC_ETC_SetTriggerChainConfig(ADC_ETC_Type *base, uint32_t triggerGroup, uint32_t chainGroup, const *adc_etc_trigger_chain_config_t* *config)

Set the external XBAR trigger chain configuration. For example, if triggerGroup is set to 0U and chainGroup is set to 1U, which means Trigger0 source's chain1 would be configurated.

Parameters

- base – ADC_ETC peripheral base address.
- triggerGroup – Trigger group index. Available number is 0~7.
- chainGroup – Trigger chain group index. Available number is 0~7.

- config – Pointer to "adc_etc_trigger_chain_config_t" structure.

uint32_t ADC_ETC_GetInterruptStatusFlags(ADC_ETC_Type *base, *adc_etc_external_trigger_source_t* sourceIndex)

Gets the interrupt status flags of external XBAR and TSC triggers.

Parameters

- base – ADC_ETC peripheral base address.
- sourceIndex – trigger source index.

Returns

Status flags mask of trigger. Refer to "_adc_etc_status_flag_mask".

void ADC_ETC_ClearInterruptStatusFlags(ADC_ETC_Type *base, *adc_etc_external_trigger_source_t* sourceIndex, uint32_t mask)

Clears the ADC_ETC's interrupt status falgs.

Parameters

- base – ADC_ETC peripheral base address.
- sourceIndex – trigger source index.
- mask – Status flags mask of trigger. Refer to "_adc_etc_status_flag_mask".

static inline void ADC_ETC_EnableDMA(ADC_ETC_Type *base, uint32_t triggerGroup)

Enable the DMA corresponding to each trigger source.

Parameters

- base – ADC_ETC peripheral base address.
- triggerGroup – Trigger group index. Available number is 0~7.

static inline void ADC_ETC_DisableDMA(ADC_ETC_Type *base, uint32_t triggerGroup)

Disable the DMA corresponding to each trigger sources.

Parameters

- base – ADC_ETC peripheral base address.
- triggerGroup – Trigger group index. Available number is 0~7.

static inline uint32_t ADC_ETC_GetDMAStatusFlags(ADC_ETC_Type *base)

Get the DMA request status falgs. Only external XBAR sources support DMA request.

Parameters

- base – ADC_ETC peripheral base address.

Returns

Mask of external XBAR tirgger's DMA request asserted flags. Available range is trigger0:0x01 to trigger7:0x80.

static inline void ADC_ETC_ClearDMAStatusFlags(ADC_ETC_Type *base, uint32_t mask)

Clear the DMA request status falgs. Only external XBAR sources support DMA request.

Parameters

- base – ADC_ETC peripheral base address.
- mask – Mask of external XBAR tirgger's DMA request asserted flags. Available range is trigger0:0x01 to trigger7:0x80.

static inline void ADC_ETC_DoSoftwareReset(ADC_ETC_Type *base, bool enable)

When enable, all logical will be reset.

Parameters

- base – ADC_ETC peripheral base address.
- enable – Enable/Disable the software reset.

static inline void ADC_ETC_DoSoftwareTrigger(ADC_ETC_Type *base, uint32_t triggerGroup)

Do software trigger corresponding to each XBAR trigger sources. Each XBAR trigger sources can be configured as HW or SW trigger mode. In hardware trigger mode, trigger source is from XBAR. In software mode, trigger source is from software tigger. TSC trigger sources can only work in hardware trigger mode.

Parameters

- base – ADC_ETC peripheral base address.
- triggerGroup – Trigger group index. Available number is 0~7.

static inline void ADC_ETC_DoSoftwareTriggerBlocking(ADC_ETC_Type *base, uint32_t triggerGroup)

Do software trigger corresponding to each XBAR trigger sources.

Note: This function provides a workaround implementation for ERR052412 by using blocking way to implement SW trigger.

Parameters

- base – ADC_ETC peripheral base address.
- triggerGroup – Trigger group index. Available number is 0~7.

uint32_t ADC_ETC_GetADCConversionValue(ADC_ETC_Type *base, uint32_t triggerGroup, uint32_t chainGroup)

Get ADC conversion result from external XBAR sources. For example, if triggerGroup is set to 0U and chainGroup is set to 1U, which means the API would return Trigger0 source's chain1 conversion result.

Parameters

- base – ADC_ETC peripheral base address.
- triggerGroup – Trigger group index. Available number is 0~7.
- chainGroup – Trigger chain group index. Available number is 0~7.

Returns

ADC conversion result value.

enum _adc_etc_status_flag_mask

ADC_ETC customized status flags mask.

Values:

enumerator kADC_ETC_Done0StatusFlagMask

enumerator kADC_ETC_Done1StatusFlagMask

enumerator kADC_ETC_Done2StatusFlagMask

enumerator kADC_ETC_Done3StatusFlagMask

enumerator kADC_ETC_ErrorStatusFlagMask

enum _adc_etc_external_trigger_source

External triggers sources.

Values:

enumerator kADC_ETC_Trg0TriggerSource

enumerator kADC_ETC_Trg1TriggerSource

enumerator kADC_ETC_Trg2TriggerSource

enumerator kADC_ETC_Trg3TriggerSource

enumerator kADC_ETC_Trg4TriggerSource

enumerator kADC_ETC_Trg5TriggerSource

enumerator kADC_ETC_Trg6TriggerSource

enumerator kADC_ETC_Trg7TriggerSource

enumerator kADC_ETC_TSC0TriggerSource

enumerator kADC_ETC_TSC1TriggerSource

enum _adc_etc_interrupt_enable

Interrupt enable/disable mask.

Values:

enumerator kADC_ETC_Done0InterruptEnable

enumerator kADC_ETC_Done1InterruptEnable

enumerator kADC_ETC_Done2InterruptEnable

enumerator kADC_ETC_Done3InterruptEnable

enum _adc_etc_dma_mode_selection

DMA mode selection.

Values:

enumerator kADC_ETC_TrigDMAWithLatchedSignal

enumerator kADC_ETC_TrigDMAWithPulsedSignal

typedef enum *_adc_etc_external_trigger_source* adc_etc_external_trigger_source_t
External triggers sources.

typedef enum *_adc_etc_interrupt_enable* adc_etc_interrupt_enable_t
Interrupt enable/disable mask.

typedef enum *_adc_etc_dma_mode_selection* adc_etc_dma_mode_selection_t
DMA mode selection.

typedef struct *_adc_etc_config* adc_etc_config_t
ADC_ETC configuration.

typedef struct *_adc_etc_trigger_chain_config* adc_etc_trigger_chain_config_t
ADC_ETC trigger chain configuration.

typedef struct *_adc_etc_trigger_config* adc_etc_trigger_config_t
ADC_ETC trigger configuration.

FSL_ADC_ETC_DRIVER_VERSION
ADC_ETC driver version.
Version 2.3.2.

ADC_ETC_DMA_CTRL_TRGn_REQ_MASK
The mask of status flags cleared by writing 1.

struct _adc_etc_config
#include <fsl_adc_etc.h> ADC_ETC configuration.

struct _adc_etc_trigger_chain_config
#include <fsl_adc_etc.h> ADC_ETC trigger chain configuration.

struct _adc_etc_trigger_config
#include <fsl_adc_etc.h> ADC_ETC trigger configuration.

2.3 AIPSTZ: AHB to IP Bridge

void AIPSTZ_SetMasterPriviledgeLevel(AIPSTZ_Type *base, *aipstz_master_t* master, uint32_t privilegeConfig)
Configure the privilege level for master.

Parameters

- base – AIPSTZ peripheral base pointer
- master – Masters for AIPSTZ.
- privilegeConfig – Configuration is ORed from *aipstz_master_privilege_level_t*.

void AIPSTZ_SetPeripheralAccessControl(AIPSTZ_Type *base, *aipstz_peripheral_t* peripheral, uint32_t accessControl)

Configure the access for peripheral.

Parameters

- base – AIPSTZ peripheral base pointer
- peripheral – Peripheral for AIPSTZ.
- accessControl – Configuration is ORed from *aipstz_peripheral_access_control_t*.

FSL_AIPSTZ_DRIVER_VERSION

Version 2.0.1

enum _aipstz_master_privilege_level

List of AIPSTZ privilege configuration.

Values:

enumerator kAIPSTZ_MasterBufferedWriteEnable

Write accesses from this master are allowed to be buffered.

enumerator kAIPSTZ_MasterTrustedForReadEnable

This master is trusted for read accesses.

enumerator kAIPSTZ_MasterTrustedForWriteEnable

This master is trusted for write accesses.

enumerator kAIPSTZ_MasterForceUserModeEnable

Accesses from this master are forced to user-mode.

enum _aipstz_master

List of AIPSTZ masters. Organized by width for the 8-15 bits and shift for lower 8 bits.

Values:

enumerator kAIPSTZ_Master0

enumerator kAIPSTZ_Master1

enumerator kAIPSTZ_Master2

enumerator kAIPSTZ_Master3

enumerator kAIPSTZ_Master5

enum _aipstz_peripheral_access_control

List of AIPSTZ peripheral access control configuration.

Values:

enumerator kAIPSTZ_PeripheralAllowUntrustedMaster

enumerator kAIPSTZ_PeripheralWriteProtected

enumerator kAIPSTZ_PeripheralRequireSupervisor

enumerator kAIPSTZ_PeripheralAllowBufferedWrite

enum _aipstz_peripheral

List of AIPSTZ peripherals. Organized by register offset for higher 32 bits, width for the 8-15 bits and shift for lower 8 bits.

Values:

enumerator kAIPSTZ_Peripheral0

enumerator kAIPSTZ_Peripheral1

enumerator kAIPSTZ_Peripheral2

enumerator kAIPSTZ_Peripheral3

enumerator kAIPSTZ_Peripheral4

enumerator kAIPSTZ_Peripheral5

enumerator kAIPSTZ_Peripheral6

enumerator kAIPSTZ_Peripheral7

enumerator kAIPSTZ_Peripheral8

enumerator kAIPSTZ_Peripheral9

enumerator kAIPSTZ_Peripheral10

enumerator kAIPSTZ_Peripheral11

enumerator kAIPSTZ_Peripheral12

enumerator kAIPSTZ_Peripheral13

enumerator kAIPSTZ_Peripheral14

enumerator kAIPSTZ_Peripheral15

enumerator kAIPSTZ_Peripheral16

enumerator kAIPSTZ_Peripheral17

enumerator kAIPSTZ_Peripheral18

enumerator kAIPSTZ_Peripheral19

enumerator kAIPSTZ_Peripheral20

enumerator kAIPSTZ_Peripheral21

enumerator kAIPSTZ_Peripheral22

enumerator kAIPSTZ_Peripheral23

enumerator kAIPSTZ_Peripheral24

enumerator kAIPSTZ_Peripheral25

enumerator kAIPSTZ_Peripheral26

enumerator kAIPSTZ_Peripheral27

enumerator kAIPSTZ_Peripheral28

enumerator kAIPSTZ_Peripheral29

enumerator kAIPSTZ_Peripheral30

enumerator kAIPSTZ_Peripheral31

enumerator kAIPSTZ_Peripheral32

enumerator kAIPSTZ_Peripheral33

typedef enum *_aipstz_master_privilege_level* aipstz_master_privilege_level_t

List of AIPSTZ privilege configuration.

typedef enum *_aipstz_master* aipstz_master_t

List of AIPSTZ masters. Organized by width for the 8-15 bits and shift for lower 8 bits.

typedef enum *_aipstz_peripheral_access_control* aipstz_peripheral_access_control_t

List of AIPSTZ peripheral access control configuration.

typedef enum *_aipstz_peripheral* aipstz_peripheral_t

List of AIPSTZ peripherals. Organized by register offset for higher 32 bits, width for the 8-15 bits and shift for lower 8 bits.

2.4 AOI: Crossbar AND/OR/INVERT Driver

void AOI_Init(AOI_Type *base)

Initializes an AOI instance for operation.

This function un-gates the AOI clock.

Parameters

- base – AOI peripheral address.

void AOI_Deinit(AOI_Type *base)

Deinitializes an AOI instance for operation.

This function shutdowns AOI module.

Parameters

- base – AOI peripheral address.

void AOI_GetEventLogicConfig(AOI_Type *base, *aoi_event_t* event, *aoi_event_config_t* *config)

Gets the Boolean evaluation associated.

This function returns the Boolean evaluation associated.

Example:

```
aoi_event_config_t demoEventLogicStruct;

AOI_GetEventLogicConfig(AOI, kAOI_Event0, &demoEventLogicStruct);
```

Parameters

- base – AOI peripheral address.
- event – Index of the event which will be set of type aoi_event_t.
- config – Selected input configuration .

void AOI_SetEventLogicConfig(AOI_Type *base, *aoi_event_t* event, const *aoi_event_config_t* *eventConfig)

Configures an AOI event.

This function configures an AOI event according to the aoiEventConfig structure. This function configures all inputs (A, B, C, and D) of all product terms (0, 1, 2, and 3) of a desired event.

Example:

```
aoi_event_config_t demoEventLogicStruct;

demoEventLogicStruct.PT0AC = kAOI_InvInputSignal;
demoEventLogicStruct.PT0BC = kAOI_InputSignal;
demoEventLogicStruct.PT0CC = kAOI_LogicOne;
demoEventLogicStruct.PT0DC = kAOI_LogicOne;

demoEventLogicStruct.PT1AC = kAOI_LogicZero;
demoEventLogicStruct.PT1BC = kAOI_LogicOne;
demoEventLogicStruct.PT1CC = kAOI_LogicOne;
demoEventLogicStruct.PT1DC = kAOI_LogicOne;

demoEventLogicStruct.PT2AC = kAOI_LogicZero;
demoEventLogicStruct.PT2BC = kAOI_LogicOne;
demoEventLogicStruct.PT2CC = kAOI_LogicOne;
demoEventLogicStruct.PT2DC = kAOI_LogicOne;
```

(continues on next page)

(continued from previous page)

```
demoEventLogicStruct.PT3AC = kAOI_LogicZero;
demoEventLogicStruct.PT3BC = kAOI_LogicOne;
demoEventLogicStruct.PT3CC = kAOI_LogicOne;
demoEventLogicStruct.PT3DC = kAOI_LogicOne;

AOI_SetEventLogicConfig(AOI, kAOI_Event0, demoEventLogicStruct);
```

Parameters

- base – AOI peripheral address.
- event – Event which will be configured of type aoi_event_t.
- eventConfig – Pointer to type aoi_event_config_t structure. The user is responsible for filling out the members of this structure and passing the pointer to this function.

FSL_AOI_DRIVER_VERSION

Version 2.0.2.

enum _aoi_input_config

AOI input configurations.

The selection item represents the Boolean evaluations.

Values:

enumerator kAOI_LogicZero

Forces the input to logical zero.

enumerator kAOI_InputSignal

Passes the input signal.

enumerator kAOI_InvInputSignal

Inverts the input signal.

enumerator kAOI_LogicOne

Forces the input to logical one.

enum _aoi_event

AOI event indexes, where an event is the collection of the four product terms (0, 1, 2, and 3) and the four signal inputs (A, B, C, and D).

Values:

enumerator kAOI_Event0

Event 0 index

enumerator kAOI_Event1

Event 1 index

enumerator kAOI_Event2

Event 2 index

enumerator kAOI_Event3

Event 3 index

typedef enum *_aoi_input_config* aoi_input_config_t

AOI input configurations.

The selection item represents the Boolean evaluations.

typedef enum *_aoi_event* aoi_event_t

AOI event indexes, where an event is the collection of the four product terms (0, 1, 2, and 3) and the four signal inputs (A, B, C, and D).

typedef struct *_aoi_event_config* aoi_event_config_t

AOI event configuration structure.

Defines structure *_aoi_event_config* and use the *AOI_SetEventLogicConfig()* function to make whole event configuration.

AOI

AOI peripheral address

struct __aoi_event_config

#include <fsl_aoi.h> AOI event configuration structure.

Defines structure *_aoi_event_config* and use the *AOI_SetEventLogicConfig()* function to make whole event configuration.

Public Members

aoi_input_config_t PT0AC

Product term 0 input A

aoi_input_config_t PT0BC

Product term 0 input B

aoi_input_config_t PT0CC

Product term 0 input C

aoi_input_config_t PT0DC

Product term 0 input D

aoi_input_config_t PT1AC

Product term 1 input A

aoi_input_config_t PT1BC

Product term 1 input B

aoi_input_config_t PT1CC

Product term 1 input C

aoi_input_config_t PT1DC
Product term 1 input D

aoi_input_config_t PT2AC
Product term 2 input A

aoi_input_config_t PT2BC
Product term 2 input B

aoi_input_config_t PT2CC
Product term 2 input C

aoi_input_config_t PT2DC
Product term 2 input D

aoi_input_config_t PT3AC
Product term 3 input A

aoi_input_config_t PT3BC
Product term 3 input B

aoi_input_config_t PT3CC
Product term 3 input C

aoi_input_config_t PT3DC
Product term 3 input D

2.5 BEE: Bus Encryption Engine

FSL_BEE_DRIVER_VERSION
BEE driver version. Version 2.0.2.

Current version: 2.0.2

Change log:

- 2.0.2
  - Bug Fixes
    - Fixed MISRA issue.
- 2.0.1
  - Bug Fixes
    - Fixed bug in key user key loading sequence. BEE must be enabled during loading of user key.
    - Fixed typos in comments.
  - New Features

        * Added configuration setting for endian swap, access permission and region security level.
      - Improvements
        * Setting of AES nonce was moved from *BEE_SetRegionKey()* into separate *BEE_SetRegionNonce()* function.
          · Changed handling of region settings. Both regions are configured simultaneously by *BEE_SetConfig()* function. Configuration of FAC start and end address using IOMUXC_GPRs was moved to application.
        * Default value for region address offset was changed to 0.
    - 2.0.0
      - Initial version

enum _bee_aes_mode

BEE aes mode.

Values:

enumerator kBEE_AesEcbMode

AES ECB Mode

enumerator kBEE_AesCtrMode

AES CTR Mode

enum _bee_region

BEE region.

Values:

enumerator kBEE_Region0

BEE region 0

enumerator kBEE_Region1

BEE region 1

enum _bee_ac_prot_enable

BEE ac prot enable.

Values:

enumerator kBEE_AccessProtDisabled

BEE access permission control disabled

enumerator kBEE_AccessProtEnabled

BEE access permission control enabled

enum _bee_endian_swap_enable

BEE endian swap enable.

Values:

enumerator kBEE_EndianSwapDisabled
BEE endian swap disabled

enumerator kBEE_EndianSwapEnabled
BEE endian swap enabled

enum _bee_security_level
BEE security level.
Values:

enumerator kBEE_SecurityLevel0
BEE security level 0

enumerator kBEE_SecurityLevel1
BEE security level 1

enumerator kBEE_SecurityLevel2
BEE security level 2

enumerator kBEE_SecurityLevel3
BEE security level 3

enum _bee_status_flags
BEE status flags.
Values:

enumerator kBEE_DisableAbortFlag
Disable abort flag.

enumerator kBEE_Reg0ReadSecViolation
Region-0 read channel security violation

enumerator kBEE_ReadIllegalAccess
Read channel illegal access detected

enumerator kBEE_Reg1ReadSecViolation
Region-1 read channel security violation

enumerator kBEE_Reg0AccessViolation
Protected region-0 access violation

enumerator kBEE_Reg1AccessViolation
Protected region-1 access violation

enumerator kBEE_IdleFlag
Idle flag

typedef enum *_bee_aes_mode* bee_aes_mode_t

BEE aes mode.

typedef enum *_bee_region* bee_region_t

BEE region.

typedef enum *_bee_ac_prot_enable* bee_ac_prot_enable

BEE ac prot enable.

typedef enum *_bee_endian_swap_enable* bee_endian_swap_enable

BEE endian swap enable.

typedef enum *_bee_security_level* bee_security_level

BEE security level.

typedef enum *_bee_status_flags* bee_status_flags_t

BEE status flags.

typedef struct *_bee_region_config* bee_region_config_t

BEE region configuration structure.

void BEE_Init(BEE_Type *base)

Resets BEE module to factory default values.

This function performs hardware reset of BEE module. Attributes and keys from software for both regions are cleared.

Parameters

- base – BEE peripheral address.

void BEE_Deinit(BEE_Type *base)

Resets BEE module, clears keys for both regions and disables clock to the BEE.

This function performs hardware reset of BEE module and disables clocks. Attributes and keys from software for both regions are cleared.

Parameters

- base – BEE peripheral address.

static inline void BEE_Enable(BEE_Type *base)

Enables BEE decryption.

This function enables decryption using BEE.

Parameters

- base – BEE peripheral address.

static inline void BEE_Disable(BEE_Type *base)

Disables BEE decryption.

This function disables decryption using BEE.

Parameters

- base – BEE peripheral address.

void BEE_GetDefaultConfig(*bee_region_config_t* *config)

Loads default values to the BEE region configuration structure.

Loads default values to the BEE region configuration structure. The default values are as follows:

```
config->region0Mode = kBEE_AesCtrMode;
config->region1Mode = kBEE_AesCtrMode;
config->region0AddrOffset = 0U;
config->region1AddrOffset = 0U;
config->region0SecLevel = kBEE_SecurityLevel3;
config->region1SecLevel = kBEE_SecurityLevel3;
config->region1Bot = 0U;
config->region1Top = 0U;
config->accessPermission = kBEE_AccessProtDisabled;
config->endianSwapEn = kBEE_EndianSwapEnabled;
```

Parameters

- config – Configuration structure for BEE peripheral.

void BEE_SetConfig(BEE_Type *base, const *bee_region_config_t* *config)

Sets BEE configuration.

This function sets BEE peripheral and BEE region settings accorging to given configuration structure.

Parameters

- base – BEE peripheral address.
- config – Configuration structure for BEE.

status_t BEE_SetRegionKey(BEE_Type *base, *bee_region_t* region, const uint8_t *key, size_t keySize)

Loads the AES key for selected region into BEE key registers.

This function loads given AES key to BEE register for the given region. The key must be 32-bit aligned and stored in little-endian format.

Please note, that eFuse BEE_KEYx_SEL must be set accordingly to be able to load and use key loaded in BEE registers. Otherwise, key cannot loaded and BEE will use key from OTPMK or SW_GP2.

Parameters

- base – BEE peripheral address.
- region – Selection of the BEE region to be configured.
- key – AES key (in little-endian format).
- keySize – Size of AES key.

status_t BEE_SetRegionNonce(BEE_Type *base, *bee_region_t* region, const uint8_t *nonce, size_t nonceSize)

Loads the nonce for selected region into BEE nonce registers.

This function loads given nonce(only AES CTR mode) to BEE register for the given region. The nonce must be 32-bit aligned and stored in little-endian format.

Parameters

- base – BEE peripheral address.
- region – Selection of the BEE region to be configured.
- nonce – AES nonce (in little-endian format).

- nonceSize – Size of AES nonce.

uint32_t BEE_GetStatusFlags(BEE_Type *base)

Gets the BEE status flags.

This function returns status of BEE peripheral.

Parameters

- base – BEE peripheral address.

Returns

The status flags. This is the logical OR of members of the enumeration *bee_status_flags_t*

void BEE_ClearStatusFlags(BEE_Type *base, uint32_t mask)

Clears the BEE status flags.

Parameters

- base – BEE peripheral base address.
- mask – The status flags to clear. This is a logical OR of members of the enumeration *bee_status_flags_t*

struct __bee_region_config

#include <fsl_bee.h> BEE region configuration structure.

Public Members

bee_aes_mode_t region0Mode

AES mode used for encryption/decryption for region 0

bee_aes_mode_t region1Mode

AES mode used for encryption/decryption for region 1

uint32_t region0AddrOffset

Region 0 address offset

uint32_t region1AddrOffset

Region 1 address offset

bee_security_level region0SecLevel

Region 0 security level

bee_security_level region1SecLevel

Region 1 security level

uint32_t region1Bot

Region 1 bottom address

uint32_t region1Top

Region 1 top address

bee_ac_prot_enable accessPermission
Access permission control enable/disable

bee_endian_swap_enable endianSwapEn
Endian swap enable/disable

2.6 CACHE: ARMV7-M7 CACHE Memory Controller

static inline void L1CACHE_EnableICache(void)
Enables cortex-m7 L1 instruction cache.

static inline void L1CACHE_DisableICache(void)
Disables cortex-m7 L1 instruction cache.

static inline void L1CACHE_InvalidateICache(void)
Invalidate cortex-m7 L1 instruction cache.

void L1CACHE_InvalidateICacheByRange(uint32_t address, uint32_t size_byte)
Invalidate cortex-m7 L1 instruction cache by range.

Note: The start address and size_byte should be 32-byte(FSL_FEATURE_L1ICACHE_LINESIZE_BYTE) aligned. The startAddr here will be forced to align to L1 I-cache line size if startAddr is not aligned. For the size_byte, application should make sure the alignment or make sure the right operation order if the size_byte is not aligned.

Parameters

- address – The start address of the memory to be invalidated.
- size_byte – The memory size.

static inline void L1CACHE_EnableDCache(void)
Enables cortex-m7 L1 data cache.

static inline void L1CACHE_DisableDCache(void)
Disables cortex-m7 L1 data cache.

static inline void L1CACHE_InvalidateDCache(void)
Invalidates cortex-m7 L1 data cache.

static inline void L1CACHE_CleanDCache(void)
Cleans cortex-m7 L1 data cache.

static inline void L1CACHE_CleanInvalidateDCache(void)
Cleans and Invalidates cortex-m7 L1 data cache.

static inline void L1CACHE_InvalidateDCacheByRange(uint32_t address, uint32_t size_byte)
Invalidates cortex-m7 L1 data cache by range.

Note: The start address and size_byte should be 32-byte(FSL_FEATURE_L1DCACHE_LINESIZE_BYTE) aligned. The startAddr here will be forced to align to L1 D-cache line size if startAddr is not aligned. For the size_byte, application should make sure the alignment or make sure the right operation order if the size_byte is not aligned.

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 cortex-m7 L1 data cache by range.

Note: The start address and size_byte should be 32-byte(FSL_FEATURE_L1DCACHE_LINESIZE_BYTE) aligned. The startAddr here will be forced to align to L1 D-cache line size if startAddr is not aligned. For the size_byte, application should make sure the alignment or make sure the right operation order if the size_byte is not aligned.

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 cortex-m7 L1 data cache by range.

Note: The start address and size_byte should be 32-byte(FSL_FEATURE_L1DCACHE_LINESIZE_BYTE) aligned. The startAddr here will be forced to align to L1 D-cache line size if startAddr is not aligned. For the size_byte, application should make sure the alignment or make sure the right operation order if the size_byte is not aligned.

Parameters

- address – The start address of the memory to be clean and invalidated.
- size_byte – The memory size.

void ICACHE_InvalidateByRange(uint32_t address, uint32_t size_byte)

Invalidates all instruction caches by range.

Both cortex-m7 L1 cache line and L2 PL310 cache line length is 32-byte.

Note: address and size should be aligned to cache line size 32-Byte due to the cache operation unit is one cache line. The startAddr here will be forced to align to the cache line size if startAddr is not aligned. For the size_byte, application should make sure the alignment or make sure the right operation order if the size_byte is not aligned.

Parameters

- address – The physical address.
- size_byte – size of the memory to be invalidated.

void DCACHE_InvalidateByRange(uint32_t address, uint32_t size_byte)

Invalidates all data caches by range.

Both cortex-m7 L1 cache line and L2 PL310 cache line length is 32-byte.

Note: address and size should be aligned to cache line size 32-Byte due to the cache operation unit is one cache line. The startAddr here will be forced to align to the cache line size if startAddr is not aligned. For the size_byte, application should make sure the alignment or make sure the right operation order if the size_byte is not aligned.

Parameters

- address – The physical address.
- size_byte – size of the memory to be invalidated.

void DCACHE_CleanByRange(uint32_t address, uint32_t size_byte)

Cleans all data caches by range.

Both cortex-m7 L1 cache line and L2 PL310 cache line length is 32-byte.

Note: address and size should be aligned to cache line size 32-Byte due to the cache operation unit is one cache line. The startAddr here will be forced to align to the cache line size if startAddr is not aligned. For the size_byte, application should make sure the alignment or make sure the right operation order if the size_byte is not aligned.

Parameters

- address – The physical address.
- size_byte – size of the memory to be cleaned.

void DCACHE_CleanInvalidateByRange(uint32_t address, uint32_t size_byte)

Cleans and Invalidates all data caches by range.

Both cortex-m7 L1 cache line and L2 PL310 cache line length is 32-byte.

Note: address and size should be aligned to cache line size 32-Byte due to the cache operation unit is one cache line. The startAddr here will be forced to align to the cache line size if startAddr is not aligned. For the size_byte, application should make sure the alignment or make sure the right operation order if the size_byte is not aligned.

Parameters

- address – The physical address.
- size_byte – size of the memory to be cleaned and invalidated.

FSL_CACHE_DRIVER_VERSION

cache driver version 2.0.6.

2.7 Clock Driver

enum _clock_name

Clock name used to get clock frequency.

Values:

enumerator kCLOCK_CpuClk
CPU clock

enumerator kCLOCK_AhbClk
AHB clock

enumerator kCLOCK_SemcClk
SEMC clock

enumerator kCLOCK_IpgClk
IPG clock

enumerator kCLOCK_PerClk
PER clock

enumerator kCLOCK_OscClk
OSC clock selected by PMU_LOWPWR_CTRL[OSC_SEL].

enumerator kCLOCK_RtcClk
RTC clock. (RTCCLK)

enumerator kCLOCK_Usb1PllClk
USB1PLLCLK.

enumerator kCLOCK_Usb1PllPfd0Clk
USB1PLLPDF0CLK.

enumerator kCLOCK_Usb1PllPfd1Clk
USB1PLLPFD1CLK.

enumerator kCLOCK_Usb1PllPfd2Clk
USB1PLLPFD2CLK.

enumerator kCLOCK_Usb1PllPfd3Clk
USB1PLLPFD3CLK.

enumerator kCLOCK_Usb1SwClk
USB1PLLSWCLK

enumerator kCLOCK_Usb1Sw60MClk
USB1PLLSw60MCLK

enumerator kCLOCK_Usb1Sw80MClk
USB1PLLSw80MCLK

enumerator kCLOCK_SysPllClk
SYSPLLCLK.

enumerator kCLOCK_SysPllPfd0Clk
SYSPLLPDF0CLK.

enumerator kCLOCK_SysPllPfd1Clk
SYSPLLPFD1CLK.

enumerator kCLOCK_SysPllPfd2Clk
SYSPLLPFD2CLK.

enumerator kCLOCK_SysPllPfd3Clk
SYSPLLPFD3CLK.

enumerator kCLOCK_EnetPllClk
Enet PLLCLK ref_enetpll.

enumerator kCLOCK_EnetPll25MClk
Enet PLLCLK ref_enetpll25M.

enumerator kCLOCK_EnetPll500MClk
Enet PLLCLK ref_enetpll500M.

enumerator kCLOCK_AudioPllClk
Audio PLLCLK.

enumerator kCLOCK_NoneName
None Clock Name.

enum __clock_ip_name
CCM CCGR gate control for each module independently.
Values:

enumerator kCLOCK_IpInvalid

enumerator kCLOCK_Aips_tz1
CCGR0, CG0

enumerator kCLOCK_Aips_tz2
CCGR0, CG1

enumerator kCLOCK_Mqs
CCGR0, CG2, Reserved

enumerator kCLOCK_FlexSpiExsc
CCGR0, CG3

enumerator kCLOCK_Sim_m_clk_r
CCGR0, CG4

enumerator kCLOCK_Dcp
CCGR0, CG5

enumerator kCLOCK_Lpuart3
CCGR0, CG6

enumerator kCLOCK_Can1
CCGR0, CG7

enumerator kCLOCK_Can1S
CCGR0, CG8

enumerator kCLOCK_Can2
CCGR0, CG9

enumerator kCLOCK_Can2S
CCGR0, CG10

enumerator kCLOCK_Trace
CCGR0, CG11

enumerator kCLOCK_Gpt2
CCGR0, CG12

enumerator kCLOCK_Gpt2S
CCGR0, CG13

enumerator kCLOCK_Lpuart2
CCGR0, CG14

enumerator kCLOCK_Gpio2
CCGR0, CG15

enumerator kCLOCK_Lpspi1
CCGR1, CG0

enumerator kCLOCK_Lpspi2
CCGR1, CG1

enumerator kCLOCK_Lpspi3
CCGR1, CG2

enumerator kCLOCK_Lpspi4
CCGR1, CG3

enumerator kCLOCK_Adc2
CCGR1, CG4

enumerator kCLOCK_Enet
CCGR1, CG5

enumerator kCLOCK_Pit
CCGR1, CG6

enumerator kCLOCK_Adc1
CCGR1, CG8

enumerator kCLOCK_SemcExsc
CCGR1, CG9

enumerator kCLOCK_Gpt1
CCGR1, CG10

enumerator kCLOCK_Gpt1S
CCGR1, CG11

enumerator kCLOCK_Lpuart4
CCGR1, CG12

enumerator kCLOCK_Gpio1
CCGR1, CG13

enumerator kCLOCK_Csu
CCGR1, CG14

enumerator kCLOCK_Gpio5
CCGR1, CG15

enumerator kCLOCK_OcramExsc
CCGR2, CG0

enumerator kCLOCK_IomuxcSnvs
CCGR2, CG2

enumerator kCLOCK_Lpi2c1
CCGR2, CG3

enumerator kCLOCK_Lpi2c2
CCGR2, CG4

enumerator kCLOCK_Lpi2c3
CCGR2, CG5

enumerator kCLOCK_Ocotp
CCGR2, CG6

enumerator kCLOCK_Xbar1
CCGR2, CG11

enumerator kCLOCK_Xbar2
CCGR2, CG12

enumerator kCLOCK_Gpio3
CCGR2, CG13

enumerator kCLOCK_Lpuart5
CCGR3, CG1

enumerator kCLOCK_Semc
CCGR3, CG2

enumerator kCLOCK_Lpuart6
CCGR3, CG3

enumerator kCLOCK_Aoi
CCGR3, CG4

enumerator kCLOCK_Ewm0
CCGR3, CG7

enumerator kCLOCK_Wdog1
CCGR3, CG8

enumerator kCLOCK_FlexRam
CCGR3, CG9

enumerator kCLOCK_Acmp1
CCGR3, CG10

enumerator kCLOCK_Acmp2
CCGR3, CG11

enumerator kCLOCK_Acmp3
CCGR3, CG12

enumerator kCLOCK_Acmp4
CCGR3, CG13

enumerator kCLOCK_IomuxcSnvsGpr
CCGR3, CG15

enumerator kCLOCK_Sim_m7_clk_r
CCGR4, CG0

enumerator kCLOCK_Iomuxc
CCGR4, CG1

enumerator kCLOCK_IomuxcGpr
CCGR4, CG2

enumerator kCLOCK_Bee
CCGR4, CG3

enumerator kCLOCK_SimM7
CCGR4, CG4

enumerator kCLOCK_Tsc
CCGR4, CG5, Reserved

enumerator kCLOCK_SimM
CCGR4, CG6

enumerator kCLOCK_SimEms
CCGR4, CG7

enumerator kCLOCK_Pwm1
CCGR4, CG8

enumerator kCLOCK_Pwm2
CCGR4, CG9

enumerator kCLOCK_Enc1
CCGR4, CG12

enumerator kCLOCK_Enc2
CCGR4, CG13

enumerator kCLOCK_Rom
CCGR5, CG0

enumerator kCLOCK_Flexio1
CCGR5, CG1

enumerator kCLOCK_Wdog3
CCGR5, CG2

enumerator kCLOCK_Dma
CCGR5, CG3

enumerator kCLOCK_Kpp
CCGR5, CG4

enumerator kCLOCK_Wdog2
CCGR5, CG5

enumerator kCLOCK_Aips_tz4
CCGR5, CG6

enumerator kCLOCK_Spdif
CCGR5, CG7

enumerator kCLOCK_Sai1
CCGR5, CG9

enumerator kCLOCK_Sai2
CCGR5, CG10

enumerator kCLOCK_Sai3
CCGR5, CG11

enumerator kCLOCK_Lpuart1
CCGR5, CG12

enumerator kCLOCK_Lpuart7
CCGR5, CG13

enumerator kCLOCK_SnvsHp
CCGR5, CG14

enumerator kCLOCK_SnvsLp
CCGR5, CG15

enumerator kCLOCK_UsbOh3
CCGR6, CG0

enumerator kCLOCK_Usdhc1
CCGR6, CG1

enumerator kCLOCK_Usdhc2
CCGR6, CG2

enumerator kCLOCK_Dcdc
CCGR6, CG3

enumerator kCLOCK_Ipmux4
CCGR6, CG4

enumerator kCLOCK_FlexSpi
CCGR6, CG5

enumerator kCLOCK_Trng
CCGR6, CG6

enumerator kCLOCK_Lpuart8
CCGR6, CG7

enumerator kCLOCK_Timer4
CCGR6, CG8

enumerator kCLOCK_Aips_tz3
CCGR6, CG9

enumerator kCLOCK_SimPer
CCGR6, CG10

enumerator kCLOCK_Anadig
CCGR6, CG11

enumerator kCLOCK_Lpi2c4
CCGR6, CG12

enumerator kCLOCK_Timer1
CCGR6, CG13

enumerator kCLOCK_Timer2
CCGR6, CG14

enum _clock_osc
OSC 24M sorce select.
Values:

enumerator kCLOCK_RcOsc
On chip OSC.

enumerator kCLOCK_XtalOsc
24M Xtal OSC

enum _clock_gate_value
Clock gate value.
Values:

enumerator kCLOCK_ClockNotNeeded
Clock is off during all modes.

enumerator kCLOCK_ClockNeededRun
Clock is on in run mode, but off in WAIT and STOP modes

enumerator kCLOCK_ClockNeededRunWait
Clock is on during all modes, except STOP mode

enum _clock_mode_t
System clock mode.
Values:

enumerator kCLOCK_ModeRun
Remain in run mode.

enumerator kCLOCK_ModeWait
Transfer to wait mode.

enumerator kCLOCK_ModeStop
Transfer to stop mode.

enum _clock_mux
MUX control names for clock mux setting.
These constants define the mux control names for clock mux setting.

- 0:7: REG offset to CCM_BASE in bytes.
- 8:15: Root clock setting bit field shift.
- 16:31: Root clock setting bit field width.

Values:

enumerator kCLOCK_Pll3SwMux
pll3_sw_clk mux name

enumerator kCLOCK_PeriphMux
periph mux name

enumerator kCLOCK_SemcAltMux
semc mux name

enumerator kCLOCK_SemcMux
semc mux name

enumerator kCLOCK_PrePeriphMux
pre-periph mux name

enumerator kCLOCK_TraceMux
trace mux name

enumerator kCLOCK_PeriphClk2Mux
periph clock2 mux name

enumerator kCLOCK_LpspiMux
lpspi mux name

enumerator kCLOCK_FlexspiMux
flexspi mux name

enumerator kCLOCK_Usdhc2Mux
usdhc2 mux name

enumerator kCLOCK_Usdhc1Mux
usdhc1 mux name

enumerator kCLOCK_Sai3Mux
sai3 mux name

enumerator kCLOCK_Sai2Mux
sai2 mux name

enumerator kCLOCK_Sai1Mux
sai1 mux name

enumerator kCLOCK_PerclkMux
perclk mux name

enumerator kCLOCK_Flexio1Mux
flexio1 mux name

enumerator kCLOCK_CanMux
can mux name

enumerator kCLOCK_UartMux
uart mux name

enumerator kCLOCK_SpdifMux
spdif mux name

enumerator kCLOCK_Lpi2cMux
lpi2c mux name

enum _clock_div
DIV control names for clock div setting.

These constants define div control names for clock div setting.

- 0:7: REG offset to CCM_BASE in bytes.
- 8:15: Root clock setting bit field shift.
- 16:31: Root clock setting bit field width.

Values:

enumerator kCLOCK_ArmDiv
core div name

enumerator kCLOCK_PeriphClk2Div
periph clock2 div name

enumerator kCLOCK_SemcDiv
semc div name

enumerator kCLOCK_AhbDiv
ahb div name

enumerator kCLOCK_IpgDiv
ipg div name

enumerator kCLOCK_LpspiDiv
lpspi div name

enumerator kCLOCK_FlexspiDiv
flexspi div name

enumerator kCLOCK_PerclkDiv
perclk div name

enumerator kCLOCK_CanDiv
can div name

enumerator kCLOCK_TraceDiv
trace div name

enumerator kCLOCK_Usdhc2Div
usdhc2 div name

enumerator kCLOCK_Usdhc1Div
usdhc1 div name

enumerator kCLOCK_UartDiv
uart div name

enumerator kCLOCK_Flexio1Div
flexio1 pre div name

enumerator kCLOCK_Sai3PreDiv
sai3 pre div name

enumerator kCLOCK_Sai3Div
sai3 div name

enumerator kCLOCK_Flexio1PreDiv
flexio1 pre div name

enumerator kCLOCK_Sai1PreDiv
sai1 pre div name

enumerator kCLOCK_Sai1Div
sai1 div name

enumerator kCLOCK_Sai2PreDiv
sai2 pre div name

enumerator kCLOCK_Sai2Div
sai2 pre div name

enumerator kCLOCK_Spdif0PreDiv
spdif pre div name

enumerator kCLOCK_Spdif0Div
spdif div name

enumerator kCLOCK_Lpi2cDiv
lpi2c div name

enumerator kCLOCK_NonePreDiv
None Pre div.

enum _clock_div_value
Clock divider value.
Values:

enumerator kCLOCK_ArmDivBy1
ARM clock divider set to divided by 1.

enumerator kCLOCK_ArmDivBy2
ARM clock divider set to divided by 2.

enumerator kCLOCK_ArmDivBy3
ARM clock divider set to divided by 3.

enumerator kCLOCK_ArmDivBy4
ARM clock divider set to divided by 4.

enumerator kCLOCK_ArmDivBy5
ARM clock divider set to divided by 5.

enumerator kCLOCK_ArmDivBy6
ARM clock divider set to divided by 6.

enumerator kCLOCK_ArmDivBy7
ARM clock divider set to divided by 7.

enumerator kCLOCK_ArmDivBy8
ARM clock divider set to divided by 8.

enumerator kCLOCK_PeriphClk2DivBy1
PeriphClk2 divider set to divided by 1.

enumerator kCLOCK_PeriphClk2DivBy2
PeriphClk2 divider set to divided by 2.

enumerator kCLOCK_PeriphClk2DivBy3
PeriphClk2 divider set to divided by 3.

enumerator kCLOCK_PeriphClk2DivBy4
PeriphClk2 divider set to divided by 4.

enumerator kCLOCK_PeriphClk2DivBy5
PeriphClk2 divider set to divided by 5.

enumerator kCLOCK_PeriphClk2DivBy6
PeriphClk2 divider set to divided by 6.

enumerator kCLOCK_PeriphClk2DivBy7
PeriphClk2 divider set to divided by 7.

enumerator kCLOCK_PeriphClk2DivBy8
PeriphClk2 divider set to divided by 8.

enumerator kCLOCK_SemcDivBy1
SEMC divider set to divided by 1.

enumerator kCLOCK_SemcDivBy2
SEMC divider set to divided by 2.

enumerator kCLOCK_SemcDivBy3
SEMC divider set to divided by 3.

enumerator kCLOCK_SemcDivBy4
SEMC divider set to divided by 4.

enumerator kCLOCK_SemcDivBy5
SEMC divider set to divided by 5.

enumerator kCLOCK_SemcDivBy6
SEMC divider set to divided by 6.

enumerator kCLOCK_SemcDivBy7
SEMC divider set to divided by 7.

enumerator kCLOCK_SemcDivBy8
SEMC divider set to divided by 8.

enumerator kCLOCK_AhbDivBy1
AHB divider set to divided by 1.

enumerator kCLOCK_AhbDivBy2
AHB divider set to divided by 2.

enumerator kCLOCK_AhbDivBy3
AHB divider set to divided by 3.

enumerator kCLOCK_AhbDivBy4
AHB divider set to divided by 4.

enumerator kCLOCK_AhbDivBy5
AHB divider set to divided by 5.

enumerator kCLOCK_AhbDivBy6
AHB divider set to divided by 6.

enumerator kCLOCK_AhbDivBy7
AHB divider set to divided by 7.

enumerator kCLOCK_AhbDivBy8
AHB divider set to divided by 8.

enumerator kCLOCK_IpgDivBy1
Ipg divider set to divided by 1.

enumerator kCLOCK_IpgDivBy2
Ipg divider set to divided by 2.

enumerator kCLOCK_IpgDivBy3
Ipg divider set to divided by 3.

enumerator kCLOCK_IpgDivBy4
Ipg divider set to divided by 4.

enumerator kCLOCK_LpspiDivBy1
LPSPI divider set to divided by 1.

enumerator kCLOCK_LpspiDivBy2
LPSPI divider set to divided by 2.

enumerator kCLOCK_LpspiDivBy3
LPSPI divider set to divided by 3.

enumerator kCLOCK_LpspiDivBy4
LPSPI divider set to divided by 4.

enumerator kCLOCK_LpspiDivBy5
LPSPI divider set to divided by 5.

enumerator kCLOCK_LpspiDivBy6
LPSPI divider set to divided by 6.

enumerator kCLOCK_LpspiDivBy7
LPSPI divider set to divided by 7.

enumerator kCLOCK_LpspiDivBy8
LPSPI divider set to divided by 8.

enumerator kCLOCK_FlexspiDivBy1
FLEXSPI divider set to divided by 1.

enumerator kCLOCK_FlexspiDivBy2
FLEXSPI divider set to divided by 2.

enumerator kCLOCK_FlexspiDivBy3
FLEXSPI divider set to divided by 3.

enumerator kCLOCK_FlexspiDivBy4
FLEXSPI divider set to divided by 4.

enumerator kCLOCK_FlexspiDivBy5
FLEXSPI divider set to divided by 5.

enumerator kCLOCK_FlexspiDivBy6
FLEXSPI divider set to divided by 6.

enumerator kCLOCK_FlexspiDivBy7
FLEXSPI divider set to divided by 7.

enumerator kCLOCK_FlexspiDivBy8
FLEXSPI divider set to divided by 8.

enumerator kCLOCK_TraceDivBy1
TRACE divider set to divided by 1.

enumerator kCLOCK_TraceDivBy2
TRACE divider set to divided by 2.

enumerator kCLOCK_TraceDivBy3
TRACE divider set to divided by 3.

enumerator kCLOCK_TraceDivBy4
TRACE divider set to divided by 4.

enumerator kCLOCK_Usdhc2DivBy1
USDHC2 divider set to divided by 1.

enumerator kCLOCK_Usdhc2DivBy2
USDHC2 divider set to divided by 2.

enumerator kCLOCK_Usdhc2DivBy3
USDHC2 divider set to divided by 3.

enumerator kCLOCK_Usdhc2DivBy4
USDHC2 divider set to divided by 4.

enumerator kCLOCK_Usdhc2DivBy5
USDHC2 divider set to divided by 5.

enumerator kCLOCK_Usdhc2DivBy6
USDHC2 divider set to divided by 6.

enumerator kCLOCK_Usdhc2DivBy7
USDHC2 divider set to divided by 7.

enumerator kCLOCK_Usdhc2DivBy8
USDHC2 divider set to divided by 8.

enumerator kCLOCK_Usdhc1DivBy1
USDHC1 divider set to divided by 1.

enumerator kCLOCK_Usdhc1DivBy2
USDHC1 divider set to divided by 2.

enumerator kCLOCK_Usdhc1DivBy3
USDHC1 divider set to divided by 3.

enumerator kCLOCK_Usdhc1DivBy4
USDHC1 divider set to divided by 4.

enumerator kCLOCK_Usdhc1DivBy5
USDHC1 divider set to divided by 5.

enumerator kCLOCK_Usdhc1DivBy6
USDHC1 divider set to divided by 6.

enumerator kCLOCK_Usdhc1DivBy7
USDHC1 divider set to divided by 7.

enumerator kCLOCK_Usdhc1DivBy8
USDHC1 divider set to divided by 8.

enumerator kCLOCK_Flexio1DivBy1
Flexio1 divider set to divided by 1.

enumerator kCLOCK_Flexio1DivBy2
Flexio1 divider set to divided by 2.

enumerator kCLOCK_Flexio1DivBy3
Flexio1 divider set to divided by 3.

enumerator kCLOCK_Flexio1DivBy4
Flexio1 divider set to divided by 4.

enumerator kCLOCK_Flexio1DivBy5
Flexio1 divider set to divided by 5.

enumerator kCLOCK_Flexio1DivBy6
Flexio1 divider set to divided by 6.

enumerator kCLOCK_Flexio1DivBy7
Flexio1 divider set to divided by 7.

enumerator kCLOCK_Flexio1DivBy8
Flexio1 divider set to divided by 8.

enumerator kCLOCK_Sai3PreDivBy1
SAI3ClkPred divider set to divided by 1.

enumerator kCLOCK_Sai3PreDivBy2
SAI3ClkPred divider set to divided by 2.

enumerator kCLOCK_Sai3PreDivBy3
SAI3ClkPred divider set to divided by 3.

enumerator kCLOCK_Sai3PreDivBy4
SAI3ClkPred divider set to divided by 4.

enumerator kCLOCK_Sai3PreDivBy5
SAI3ClkPred divider set to divided by 5.

enumerator kCLOCK_Sai3PreDivBy6
SAI3ClkPred divider set to divided by 6.

enumerator kCLOCK_Sai3PreDivBy7
SAI3ClkPred divider set to divided by 7.

enumerator kCLOCK_Sai3PreDivBy8
SAI3ClkPred divider set to divided by 8.

enumerator kCLOCK_Flexio1PreDivBy1
Flexio1 pred divider set to divided by 1.

enumerator kCLOCK_Flexio1PreDivBy2
Flexio1 pred divider set to divided by 2.

enumerator kCLOCK_Flexio1PreDivBy3
Flexio1 pred divider set to divided by 3.

enumerator kCLOCK_Flexio1PreDivBy4
Flexio1 pred divider set to divided by 4.

enumerator kCLOCK_Flexio1PreDivBy5
Flexio1 pred divider set to divided by 5.

enumerator kCLOCK_Flexio1PreDivBy6
Flexio1 pred divider set to divided by 6.

enumerator kCLOCK_Flexio1PreDivBy7
Flexio1 pred divider set to divided by 7.

enumerator kCLOCK_Flexio1PreDivBy8
Flexio1 pred divider set to divided by 8.

enumerator kCLOCK_Sai1PreDivBy1
SAI1 pred divider set to divided by 1.

enumerator kCLOCK_Sai1PreDivBy2
SAI1 pred divider set to divided by 2.

enumerator kCLOCK_Sai1PreDivBy3
SAI1 pred divider set to divided by 3.

enumerator kCLOCK_Sai1PreDivBy4
SAI1 pred divider set to divided by 4.

enumerator kCLOCK_Sai1PreDivBy5
SAI1 pred divider set to divided by 5.

enumerator kCLOCK_Sai1PreDivBy6
SAI1 pred divider set to divided by 6.

enumerator kCLOCK_Sai1PreDivBy7
SAI1 pred divider set to divided by 7.

enumerator kCLOCK_Sai1PreDivBy8
SAI1 pred divider set to divided by 8.

enumerator kCLOCK_Sai2PreDivBy1
SAI2ClkPred divider set to divided by 1.

enumerator kCLOCK_Sai2PreDivBy2
SAI2ClkPred divider set to divided by 2.

enumerator kCLOCK_Sai2PreDivBy3
SAI2ClkPred divider set to divided by 3.

enumerator kCLOCK_Sai2PreDivBy4
SAI2ClkPred divider set to divided by 4.

enumerator kCLOCK_Sai2PreDivBy5
SAI2ClkPred divider set to divided by 5.

enumerator kCLOCK_Sai2PreDivBy6
SAI2ClkPred divider set to divided by 6.

enumerator kCLOCK_Sai2PreDivBy7
SAI2ClkPred divider set to divided by 7.

enumerator kCLOCK_Sai2PreDivBy8
SAI2ClkPred divider set to divided by 8.

enumerator kCLOCK_Spdif0PreDivBy1
SPDIF0ClkPred divider set to divided by 1.

enumerator kCLOCK_Spdif0PreDivBy2
SPDIF0ClkPred divider set to divided by 2.

enumerator kCLOCK_Spdif0PreDivBy3
SPDIF0ClkPred divider set to divided by 3.

enumerator kCLOCK_Spdif0PreDivBy4
SPDIF0ClkPred divider set to divided by 4.

enumerator kCLOCK_Spdif0PreDivBy5
SPDIF0ClkPred divider set to divided by 5.

enumerator kCLOCK_Spdif0PreDivBy6
SPDIF0ClkPred divider set to divided by 6.

enumerator kCLOCK_Spdif0PreDivBy7
SPDIF0ClkPred divider set to divided by 7.

enumerator kCLOCK_Spdif0PreDivBy8
SPDIF0ClkPred divider set to divided by 8.

enumerator kCLOCK_Spdif0DivBy1
SPDIF0ClkPodf divider set to divided by 1.

enumerator kCLOCK_Spdif0DivBy2
SPDIF0ClkPodf divider set to divided by 2.

enumerator kCLOCK_Spdif0DivBy3
SPDIF0ClkPodf divider set to divided by 3.

enumerator kCLOCK_Spdif0DivBy4
SPDIF0ClkPodf divider set to divided by 4.

enumerator kCLOCK_Spdif0DivBy5
SPDIF0ClkPodf divider set to divided by 5.

enumerator kCLOCK_Spdif0DivBy6
SPDIF0ClkPodf divider set to divided by 6.

enumerator kCLOCK_Spdif0DivBy7
SPDIF0ClkPodf divider set to divided by 7.

enumerator kCLOCK_Spdif0DivBy8
SPDIF0ClkPodf divider set to divided by 8.

enumerator kCLOCK_MiscDivBy1
Misc divider like LPI2C set to divided by 1.

enumerator kCLOCK_MiscDivBy2
Misc divider like LPI2C set to divided by 2.

enumerator kCLOCK_MiscDivBy3
Misc divider like LPI2C set to divided by 3.

enumerator kCLOCK_MiscDivBy4
Misc divider like LPI2C set to divided by 4.

enumerator kCLOCK_MiscDivBy5
Misc divider like LPI2C set to divided by 5.

enumerator kCLOCK_MiscDivBy6
Misc divider like LPI2C set to divided by 6.

enumerator kCLOCK_MiscDivBy7
Misc divider like LPI2C set to divided by 7.

enumerator kCLOCK_MiscDivBy8
Misc divider like LPI2C set to divided by 8.

enumerator kCLOCK_MiscDivBy9
Misc divider like LPI2C set to divided by 9.

enumerator kCLOCK_MiscDivBy10
Misc divider like LPI2C set to divided by 10.

enumerator kCLOCK_MiscDivBy11
Misc divider like LPI2C set to divided by 11.

enumerator kCLOCK_MiscDivBy12
Misc divider like LPI2C set to divided by 12.

enumerator kCLOCK_MiscDivBy13
Misc divider like LPI2C set to divided by 13.

enumerator kCLOCK_MiscDivBy14
Misc divider like LPI2C set to divided by 14.

enumerator kCLOCK_MiscDivBy15
Misc divider like LPI2C set to divided by 15.

enumerator kCLOCK_MiscDivBy16
Misc divider like LPI2C set to divided by 16.

enumerator kCLOCK_MiscDivBy17
Misc divider like LPI2C set to divided by 17.

enumerator kCLOCK_MiscDivBy18
Misc divider like LPI2C set to divided by 18.

enumerator kCLOCK_MiscDivBy19
Misc divider like LPI2C set to divided by 19.

enumerator kCLOCK_MiscDivBy20
Misc divider like LPI2C set to divided by 20.

enumerator kCLOCK_MiscDivBy21
Misc divider like LPI2C set to divided by 21.

enumerator kCLOCK_MiscDivBy22
Misc divider like LPI2C set to divided by 22.

enumerator kCLOCK_MiscDivBy23
Misc divider like LPI2C set to divided by 23.

enumerator kCLOCK_MiscDivBy24
Misc divider like LPI2C set to divided by 24.

enumerator kCLOCK_MiscDivBy25
Misc divider like LPI2C set to divided by 25.

enumerator kCLOCK_MiscDivBy26
Misc divider like LPI2C set to divided by 26.

enumerator kCLOCK_MiscDivBy27
Misc divider like LPI2C set to divided by 27.

enumerator kCLOCK_MiscDivBy28
Misc divider like LPI2C set to divided by 28.

enumerator kCLOCK_MiscDivBy29
Misc divider like LPI2C set to divided by 29.

enumerator kCLOCK_MiscDivBy30
Misc divider like LPI2C set to divided by 30.

enumerator kCLOCK_MiscDivBy31
Misc divider like LPI2C set to divided by 31.

enumerator kCLOCK_MiscDivBy32
Misc divider like LPI2C set to divided by 32.

enumerator kCLOCK_MiscDivBy33
Misc divider like LPI2C set to divided by 33.

enumerator kCLOCK_MiscDivBy34
Misc divider like LPI2C set to divided by 34.

enumerator kCLOCK_MiscDivBy35
Misc divider like LPI2C set to divided by 35.

enumerator kCLOCK_MiscDivBy36
Misc divider like LPI2C set to divided by 36.

enumerator kCLOCK_MiscDivBy37
Misc divider like LPI2C set to divided by 37.

enumerator kCLOCK_MiscDivBy38
Misc divider like LPI2C set to divided by 38.

enumerator kCLOCK_MiscDivBy39
Misc divider like LPI2C set to divided by 39.

enumerator kCLOCK_MiscDivBy40
Misc divider like LPI2C set to divided by 40.

enumerator kCLOCK_MiscDivBy41
Misc divider like LPI2C set to divided by 41.

enumerator kCLOCK_MiscDivBy42
Misc divider like LPI2C set to divided by 42.

enumerator kCLOCK_MiscDivBy43
Misc divider like LPI2C set to divided by 43.

enumerator kCLOCK_MiscDivBy44
Misc divider like LPI2C set to divided by 44.

enumerator kCLOCK_MiscDivBy45
Misc divider like LPI2C set to divided by 45.

enumerator kCLOCK_MiscDivBy46
Misc divider like LPI2C set to divided by 46.

enumerator kCLOCK_MiscDivBy47
Misc divider like LPI2C set to divided by 47.

enumerator kCLOCK_MiscDivBy48
Misc divider like LPI2C set to divided by 48.

enumerator kCLOCK_MiscDivBy49
Misc divider like LPI2C set to divided by 49.

enumerator kCLOCK_MiscDivBy50
Misc divider like LPI2C set to divided by 50.

enumerator kCLOCK_MiscDivBy51
Misc divider like LPI2C set to divided by 51.

enumerator kCLOCK_MiscDivBy52
Misc divider like LPI2C set to divided by 52.

enumerator kCLOCK_MiscDivBy53
Misc divider like LPI2C set to divided by 53.

enumerator kCLOCK_MiscDivBy54
Misc divider like LPI2C set to divided by 54.

enumerator kCLOCK_MiscDivBy55
Misc divider like LPI2C set to divided by 55.

enumerator kCLOCK_MiscDivBy56
Misc divider like LPI2C set to divided by 56.

enumerator kCLOCK_MiscDivBy57
Misc divider like LPI2C set to divided by 57.

enumerator kCLOCK_MiscDivBy58
Misc divider like LPI2C set to divided by 58.

enumerator kCLOCK_MiscDivBy59
Misc divider like LPI2C set to divided by 59.

enumerator kCLOCK_MiscDivBy60
Misc divider like LPI2C set to divided by 60.

enumerator kCLOCK_MiscDivBy61
Misc divider like LPI2C set to divided by 61.

enumerator kCLOCK_MiscDivBy62
Misc divider like LPI2C set to divided by 62.

enumerator kCLOCK_MiscDivBy63
Misc divider like LPI2C set to divided by 63.

enumerator kCLOCK_MiscDivBy64
Misc divider like LPI2C set to divided by 64.

enum _clock_usb_src
USB clock source definition.
Values:

enumerator kCLOCK_Usb480M
Use 480M.

enumerator kCLOCK_UsbSrcUnused
Used when the function does not care the clock source.

enum _clock_usb_phy_src
Source of the USB HS PHY.
Values:

enumerator kCLOCK_Usbphy480M
Use 480M.

enum _clock_pll_clk_src

PLL clock source, bypass cloco source also.

Values:

enumerator kCLOCK_PllClkSrc24M

Pll clock source 24M

enumerator kCLOCK_PllSrcClkPN

Pll clock source CLK1_P and CLK1_N

enum _clock_pll

PLL name.

Values:

enumerator kCLOCK_PllSys

PLL SYS

enumerator kCLOCK_PllUsb1

PLL USB1

enumerator kCLOCK_PllAudio

PLL Audio

enumerator kCLOCK_PllEnet

PLL Enet0

enumerator kCLOCK_PllEnet500M

PLL ENET

enumerator kCLOCK_PllEnet25M

PLL Enet1

enum _clock_pfd

PLL PFD name.

Values:

enumerator kCLOCK_Pfd0

PLL PFD0

enumerator kCLOCK_Pfd1

PLL PFD1

enumerator kCLOCK_Pfd2

PLL PFD2

enumerator kCLOCK_Pfd3

PLL PFD3

enum _clock_output1_selection

The enumerater of clock output1's clock source, such as USB1 PLL, SYS PLL and so on.

Values:

enumerator kCLOCK_OutputPllUsb1Sw

Selects USB1 PLL SW clock(Divided by 2) output.

enumerator kCLOCK_OutputPllSys

Selects SYS PLL clock(Divided by 2) output.

enumerator kCLOCK_OutputPllENET500M

Selects ENET PLL clock(Divided by 2) output.

enumerator kCLOCK_OutputSemcClk

Selects semc clock root output.

enumerator kCLOCK_OutputAhbClk

Selects AHB clock root output.

enumerator kCLOCK_OutputIpgClk

Selects IPG clock root output.

enumerator kCLOCK_OutputPerClk

Selects PERCLK clock root output.

enumerator kCLOCK_OutputPll4MainClk

Selects PLL4 main clock output.

enumerator kCLOCK_DisableClockOutput1

Disables CLKO1.

enum _clock_output2_selection

The enumerater of clock output2's clock source, such as USDHC1 clock root, LPI2C clock root and so on.

Values:

enumerator kCLOCK_OutputUsdhc1Clk

Selects USDHC1 clock root output.

enumerator kCLOCK_OutputLpi2cClk

Selects LPI2C clock root output.

enumerator kCLOCK_OutputOscClk

Selects OSC output.

enumerator kCLOCK_OutputLpspiClk

Selects LPSPI clock root output.

enumerator kCLOCK_OutputUsdhc2Clk
Selects USDHC2 clock root output.

enumerator kCLOCK_OutputSai1Clk
Selects SAI1 clock root output.

enumerator kCLOCK_OutputSai2Clk
Selects SAI2 clock root output.

enumerator kCLOCK_OutputSai3Clk
Selects SAI3 clock root output.

enumerator kCLOCK_OutputTraceClk
Selects Trace clock root output.

enumerator kCLOCK_OutputCanClk
Selects CAN clock root output.

enumerator kCLOCK_OutputFlexspiClk
Selects FLEXSPI clock root output.

enumerator kCLOCK_OutputUartClk
Selects UART clock root output.

enumerator kCLOCK_OutputSpdif0Clk
Selects SPDIF0 clock root output.

enumerator kCLOCK_DisableClockOutput2
Disables CLKO2.

enum _clock_output_divider
The enumerator of clock output's divider.
Values:

enumerator kCLOCK_DivideBy1
Output clock divided by 1.

enumerator kCLOCK_DivideBy2
Output clock divided by 2.

enumerator kCLOCK_DivideBy3
Output clock divided by 3.

enumerator kCLOCK_DivideBy4
Output clock divided by 4.

enumerator kCLOCK_DivideBy5
Output clock divided by 5.

enumerator kCLOCK_DivideBy6
Output clock divided by 6.

enumerator kCLOCK_DivideBy7
Output clock divided by 7.

enumerator kCLOCK_DivideBy8
Output clock divided by 8.

enum _clock_root
The enumerator of clock root.
Values:

enumerator kCLOCK_Usdhc1ClkRoot
USDHC1 clock root.

enumerator kCLOCK_Usdhc2ClkRoot
USDHC2 clock root.

enumerator kCLOCK_FlexspiClkRoot
FLEXSPI clock root.

enumerator kCLOCK_LpspiClkRoot
LPSPI clock root.

enumerator kCLOCK_TraceClkRoot
Trace clock root.

enumerator kCLOCK_Sai1ClkRoot
SAI1 clock root.

enumerator kCLOCK_Sai2ClkRoot
SAI2 clock root.

enumerator kCLOCK_Sai3ClkRoot
SAI3 clock root.

enumerator kCLOCK_Lpi2cClkRoot
LPI2C clock root.

enumerator kCLOCK_CanClkRoot
CAN clock root.

enumerator kCLOCK_UartClkRoot
UART clock root.

enumerator kCLOCK_SpdifClkRoot
SPDIF clock root.

enumerator kCLOCK_Flexio1ClkRoot
FLEXIO1 clock root.

typedef enum *_clock_name* clock_name_t
Clock name used to get clock frequency.

typedef enum *_clock_ip_name* clock_ip_name_t
CCM CCGR gate control for each module independently.

typedef enum *_clock_osc* clock_osc_t
OSC 24M sorce select.

typedef enum *_clock_gate_value* clock_gate_value_t
Clock gate value.

typedef enum *_clock_mode_t* clock_mode_t
System clock mode.

typedef enum *_clock_mux* clock_mux_t
MUX control names for clock mux setting.
These constants define the mux control names for clock mux setting.
- 0:7: REG offset to CCM_BASE in bytes.
- 8:15: Root clock setting bit field shift.
- 16:31: Root clock setting bit field width.

typedef enum *_clock_div* clock_div_t
DIV control names for clock div setting.
These constants define div control names for clock div setting.
- 0:7: REG offset to CCM_BASE in bytes.
- 8:15: Root clock setting bit field shift.
- 16:31: Root clock setting bit field width.

typedef enum *_clock_div_value* clock_div_value_t
Clock divider value.

typedef enum *_clock_usb_src* clock_usb_src_t
USB clock source definition.

typedef enum *_clock_usb_phy_src* clock_usb_phy_src_t
Source of the USB HS PHY.

typedef struct *_clock_usb_pll_config* clock_usb_pll_config_t
PLL configuration for USB.

typedef struct *_clock_sys_pll_config* clock_sys_pll_config_t
PLL configuration for System.

typedef struct *_clock_audio_pll_config* clock_audio_pll_config_t

PLL configuration for AUDIO and VIDEO.

typedef struct *_clock_enet_pll_config* clock_enet_pll_config_t

PLL configuration for ENET.

typedef enum *_clock_pll* clock_pll_t

PLL name.

typedef enum *_clock_pfd* clock_pfd_t

PLL PFD name.

typedef enum *_clock_output1_selection* clock_output1_selection_t

The enumerater of clock output1's clock source, such as USB1 PLL, SYS PLL and so on.

typedef enum *_clock_output2_selection* clock_output2_selection_t

The enumerater of clock output2's clock source, such as USDHC1 clock root, LPI2C clock root and so on.

typedef enum *_clock_output_divider* clock_output_divider_t

The enumerator of clock output's divider.

typedef enum *_clock_root* clock_root_t

The enumerator of clock root.

volatile uint32_t g_xtalFreq

External XTAL (24M OSC/SYSOSC) clock frequency.

The XTAL (24M OSC/SYSOSC) clock frequency in Hz, when the clock is setup, use the function CLOCK_SetXtalFreq to set the value in to clock driver. For example, if XTAL is 24MHz,

```
CLOCK_InitExternalClk(false);
CLOCK_SetXtalFreq(240000000);
```

volatile uint32_t g_rtcXtalFreq

External RTC XTAL (32K OSC) clock frequency.

The RTC XTAL (32K OSC) clock frequency in Hz, when the clock is setup, use the function CLOCK_SetRtcXtalFreq to set the value in to clock driver.

static inline void CLOCK_SetMux(*clock_mux_t* mux, uint32_t value)

Set CCM MUX node to certain value.

Parameters

- mux – Which mux node to set, see *clock_mux_t*.
- value – Clock mux value to set, different mux has different value range.

static inline uint32_t CLOCK_GetMux(*clock_mux_t* mux)

Get CCM MUX value.

Parameters

- mux – Which mux node to get, see *clock_mux_t*.

Returns
Clock mux value.

static inline void CLOCK_SetDiv(*clock_div_t* divider, uint32_t value)

Set clock divider value.

Example, set the ARM clock divider to divide by 2:

```
CLOCK_SetDiv(kCLOCK_ArmDiv, kCLOCK_ArmDivBy2);
```

Example, set the LPI2C clock divider to divide by 5.

```
CLOCK_SetDiv(kCLOCK_Lpi2cDiv, kCLOCK_MiscDivBy5);
```

Only *kCLOCK_PerclkDiv*, *kCLOCK_CanDiv*,*kCLOCK_UartDiv*, *kCLOCK_Sai3Div*, *kCLOCK_Sai1Div*, *kCLOCK_Sai2Div*, *kCLOCK_Lpi2cDiv* can use the divider kCLOCK_MiscDivByxxx.

Parameters

- divider – Which divider node to set.
- value – Clock div value to set, different divider has different value range. See *clock_div_value_t* for details. Divided clock frequency = Undivided clock frequency / (value + 1)

static inline uint32_t CLOCK_GetDiv(*clock_div_t* divider)

Get CCM DIV node value.

Parameters

- divider – Which div node to get, see *clock_div_t*.

static inline void CLOCK_ControlGate(*clock_ip_name_t* name, *clock_gate_value_t* value)

Control the clock gate for specific IP.

Parameters

- name – Which clock to enable, see *clock_ip_name_t*.
- value – Clock gate value to set, see *clock_gate_value_t*.

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_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_SetMode(*clock_mode_t* mode)

Setting the low power mode that system will enter on next assertion of dsm_request signal.

Parameters

- mode – Which mode to enter, see *clock_mode_t*.

static inline uint32_t CLOCK_GetOscFreq(void)

Gets the OSC clock frequency.

This function will return the external XTAL OSC frequency if it is selected as the source of OSC, otherwise internal 24MHz RC OSC frequency will be returned.

Returns
Clock frequency; If the clock is invalid, returns 0.

uint32_t CLOCK_GetAhbFreq(void)

Gets the AHB clock frequency.

Returns
The AHB clock frequency value in hertz.

uint32_t CLOCK_GetSemcFreq(void)

Gets the SEMC clock frequency.

Returns
The SEMC clock frequency value in hertz.

uint32_t CLOCK_GetIpgFreq(void)

Gets the IPG clock frequency.

Returns
The IPG clock frequency value in hertz.

uint32_t CLOCK_GetPerClkFreq(void)

Gets the PER clock frequency.

Returns
The PER clock frequency value in hertz.

uint32_t CLOCK_GetFreq(*clock_name_t* name)

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

- name – Clock names defined in clock_name_t

Returns
Clock frequency value in hertz

static inline uint32_t CLOCK_GetCpuClkFreq(void)

Get the CCM CPU/core/system frequency.

Returns
Clock frequency; If the clock is invalid, returns 0.

uint32_t CLOCK_GetClockRootFreq(*clock_root_t* clockRoot)

Gets the frequency of selected clock root.

Parameters

- clockRoot – The clock root used to get the frequency, please refer to *clock_root_t*.

Returns
The frequency of selected clock root.

FSL_CLOCK_DRIVER_VERSION

CLOCK driver version 2.5.4.

SDK_DEVICE_MAXIMUM_CPU_CLOCK_FREQUENCY

CCM_ANALOG_PLL_BYPASS_SHIFT

CCM_ANALOG_PLL_BYPASS_SHIFT

CCM_ANALOG_PLL_BYPASS_CLK_SRC_MASK

CCM_ANALOG_PLL_BYPASS_CLK_SRC_MASK

CCM_ANALOG_PLL_BYPASS_CLK_SRC_SHIFT

CCM_ANALOG_PLL_BYPASS_CLK_SRC_SHIFT

CCSR_OFFSET
CCM registers offset.

CBCDR_OFFSET

CBCMR_OFFSET

CSCMR1_OFFSET

CSCMR2_OFFSET

CSCDR1_OFFSET

CDCDR_OFFSET

CSCDR2_OFFSET

CACRR_OFFSET

CS1CDR_OFFSET

CS2CDR_OFFSET

PLL_SYS_OFFSET
CCM Analog registers offset.

PLL_USB1_OFFSET

PLL_AUDIO_OFFSET

PLL_ENET_OFFSET

CCM_TUPLE(reg, shift, mask, busyShift)

CCM_TUPLE_REG(base, tuple)

CCM_TUPLE_SHIFT(tuple)

CCM_TUPLE_MASK(tuple)

CCM_TUPLE_BUSY_SHIFT(tuple)

CCM_NO_BUSY_WAIT

CCM_ANALOG_TUPLE(reg, shift)
: CCM ANALOG tuple macros to map corresponding registers and bit fields.

CCM_ANALOG_TUPLE_SHIFT(tuple)

CCM_ANALOG_TUPLE_REG_OFF(base, tuple, off)

CCM_ANALOG_TUPLE_REG(base, tuple)

CAN_CLOCK_CHECK_NO_AFFECTS

CLKPN_FREQ
: clock1PN frequency.

CLOCK_SetXtal0Freq

CLOCK_SetXtal32Freq

ADC_CLOCKS
: Clock ip name array for ADC.

AOI_CLOCKS
: Clock ip name array for AOI.

BEE_CLOCKS
: Clock ip name array for BEE.

CMP_CLOCKS
: Clock ip name array for CMP.

DCDC_CLOCKS
: Clock ip name array for DCDC.

DCP_CLOCKS
: Clock ip name array for DCP.

DMAMUX_CLOCKS
: Clock ip name array for DMAMUX_CLOCKS.

EDMA_CLOCKS
: Clock ip name array for DMA.

ENC_CLOCKS
: Clock ip name array for ENC.

ENET_CLOCKS
: Clock ip name array for ENET.

EWM_CLOCKS
: Clock ip name array for EWM.

FLEXCAN_CLOCKS
: Clock ip name array for FLEXCAN.

FLEXCAN_PERIPH_CLOCKS
: Clock ip name array for FLEXCAN Peripheral clock.

FLEXIO_CLOCKS
: Clock ip name array for FLEXIO.

FLEXRAM_CLOCKS
: Clock ip name array for FLEXRAM.

FLEXSPI_CLOCKS
: Clock ip name array for FLEXSPI.

FLEXSPI_EXSC_CLOCKS
: Clock ip name array for FLEXSPI EXSC.

GPIO_CLOCKS
: Clock ip name array for GPIO.

GPT_CLOCKS
: Clock ip name array for GPT.

KPP_CLOCKS
: Clock ip name array for KPP.

LPI2C_CLOCKS
: Clock ip name array for LPI2C.

LPSPI_CLOCKS
: Clock ip name array for LPSPI.

LPUART_CLOCKS
: Clock ip name array for LPUART.

OCRAM_EXSC_CLOCKS
: Clock ip name array for OCRAM EXSC.

PIT_CLOCKS
: Clock ip name array for PIT.

PWM_CLOCKS
: Clock ip name array for PWM.

RTWDOG_CLOCKS
: Clock ip name array for RTWDOG.

SAI_CLOCKS
: Clock ip name array for SAI.

SEMC_CLOCKS
: Clock ip name array for SEMC.

SEMC_EXSC_CLOCKS
: Clock ip name array for SEMC EXSC.

TMR_CLOCKS
: Clock ip name array for QTIMER.

TRNG_CLOCKS
: Clock ip name array for TRNG.

WDOG_CLOCKS
: Clock ip name array for WDOG.

USDHC_CLOCKS
: Clock ip name array for USDHC.

SPDIF_CLOCKS
: Clock ip name array for SPDIF.

XBARA_CLOCKS
: Clock ip name array for XBARA.

XBARB_CLOCKS
: Clock ip name array for XBARB.

CLOCK_SOURCE_NONE

CLOCK_ROOT_SOUCE

CLOCK_ROOT_MUX_TUPLE

CLOCK_ROOT_NONE_PRE_DIV

CLOCK_ROOT_DIV_TUPLE

kCLOCK_CoreSysClk
: For compatible with other platforms without CCM.

CLOCK_GetCoreSysClkFreq

For compatible with other platforms without CCM.

void CLOCK_InitExternalClk(bool bypassXtalOsc)

Initialize the external 24MHz clock.

This function supports two modes:

a. Use external crystal oscillator.

b. Bypass the external crystal oscillator, using input source clock directly.

After this function, please call CLOCK_SetXtal0Freq to inform clock driver the external clock frequency.

Note: This device does not support bypass external crystal oscillator, so the input parameter should always be false.

Parameters

- bypassXtalOsc – Pass in true to bypass the external crystal oscillator.

void CLOCK_DeinitExternalClk(void)

Deinitialize the external 24MHz clock.

This function disables the external 24MHz clock.

After this function, please call CLOCK_SetXtal0Freq to set external clock frequency to 0.

void CLOCK_SwitchOsc(*clock_osc_t* osc)

Switch the OSC.

This function switches the OSC source for SoC.

Parameters

- osc – OSC source to switch to.

static inline uint32_t CLOCK_GetRtcFreq(void)

Gets the RTC clock frequency.

Returns

Clock frequency; If the clock is invalid, returns 0.

static inline void CLOCK_SetXtalFreq(uint32_t freq)

Set the XTAL (24M OSC) frequency based on board setting.

Parameters

- freq – The XTAL input clock frequency in Hz.

static inline void CLOCK_SetRtcXtalFreq(uint32_t freq)

Set the RTC XTAL (32K OSC) frequency based on board setting.

Parameters

- freq – The RTC XTAL input clock frequency in Hz.

void CLOCK_InitRcOsc24M(void)

Initialize the RC oscillator 24MHz clock.

void CLOCK_DeinitRcOsc24M(void)

Power down the RCOSC 24M clock.

bool CLOCK_EnableUsbhs0Clock(*clock_usb_src_t* src, uint32_t freq)

Enable USB HS clock.

This function only enables the access to USB HS prepheral, upper layer should first call the CLOCK_EnableUsbhs0PhyPllClock to enable the PHY clock to use USB HS.

Parameters

- src – USB HS does not care about the clock source, here must be *kCLOCK_UsbSrcUnused*.
- freq – USB HS does not care about the clock source, so this parameter is ignored.

Return values

- true – The clock is set successfully.
- false – The clock source is invalid to get proper USB HS clock.

static inline void CLOCK_SetPllBypass(CCM_ANALOG_Type *base, *clock_pll_t* pll, bool bypass)

PLL bypass setting.

Parameters

- base – CCM_ANALOG base pointer.
- pll – PLL control name (see ccm_analog_pll_control_t enumeration)
- bypass – Bypass the PLL.
  - true: Bypass the PLL.
  - false:Not bypass the PLL.

static inline bool CLOCK_IsPllBypassed(CCM_ANALOG_Type *base, *clock_pll_t* pll)

Check if PLL is bypassed.

Parameters

- base – CCM_ANALOG base pointer.
- pll – PLL control name (see ccm_analog_pll_control_t enumeration)

Returns

PLL bypass status.

- true: The PLL is bypassed.
- false: The PLL is not bypassed.

static inline bool CLOCK_IsPllEnabled(CCM_ANALOG_Type *base, *clock_pll_t* pll)

Check if PLL is enabled.

Parameters

- base – CCM_ANALOG base pointer.
- pll – PLL control name (see ccm_analog_pll_control_t enumeration)

Returns

PLL bypass status.

- true: The PLL is enabled.
- false: The PLL is not enabled.

static inline void CLOCK_SetPllBypassRefClkSrc(CCM_ANALOG_Type *base, *clock_pll_t* pll, uint32_t src)

PLL bypass clock source setting. Note: change the bypass clock source also change the pll reference clock source.

Parameters

- base – CCM_ANALOG base pointer.
- pll – PLL control name (see ccm_analog_pll_control_t enumeration)
- src – Bypass clock source, reference _clock_pll_bypass_clk_src.

static inline uint32_t CLOCK_GetPllBypassRefClk(CCM_ANALOG_Type *base, *clock_pll_t* pll)

Get PLL bypass clock value, it is PLL reference clock actually. If CLOCK1_P,CLOCK1_N is choose as the pll bypass clock source, please implement the CLKPN_FREQ define, otherwise 0 will be returned.

Parameters

- base – CCM_ANALOG base pointer.
- pll – PLL control name (see ccm_analog_pll_control_t enumeration)

Return values

bypass – reference clock frequency value.

void CLOCK_InitSysPll(const *clock_sys_pll_config_t* *config)

Initialize the System PLL.

This function initializes the System PLL with specific settings

Parameters

- config – Configuration to set to PLL.

void CLOCK_DeinitSysPll(void)

De-initialize the System PLL.

void CLOCK_InitUsb1Pll(const *clock_usb_pll_config_t* *config)

Initialize the USB1 PLL.

This function initializes the USB1 PLL with specific settings

Parameters

- config – Configuration to set to PLL.

void CLOCK_DeinitUsb1Pll(void)

Deinitialize the USB1 PLL.

void CLOCK_InitAudioPll(const *clock_audio_pll_config_t* *config)

Initializes the Audio PLL.

This function initializes the Audio PLL with specific settings

Parameters

- config – Configuration to set to PLL.

void CLOCK_DeinitAudioPll(void)

De-initialize the Audio PLL.

void CLOCK_InitEnetPll(const *clock_enet_pll_config_t* *config)

Initialize the ENET PLL.

This function initializes the ENET PLL with specific settings.

Parameters

- config – Configuration to set to PLL.

void CLOCK_DeinitEnetPll(void)

Deinitialize the ENET PLL.

This function disables the ENET PLL.

uint32_t CLOCK_GetPllFreq(*clock_pll_t* pll)

Get current PLL output frequency.

This function get current output frequency of specific PLL

Parameters

- pll – pll name to get frequency.

Returns

The PLL output frequency in hertz.

void CLOCK_InitSysPfd(*clock_pfd_t* pfd, uint8_t pfdFrac)

Initialize the System PLL PFD.

This function initializes the System PLL PFD. During new value setting, the clock output is disabled to prevent glitch.

Note: It is recommended that PFD settings are kept between 12-35.

Parameters

- pfd – Which PFD clock to enable.
- pfdFrac – The PFD FRAC value.

void CLOCK_DeinitSysPfd(*clock_pfd_t* pfd)

De-initialize the System PLL PFD.

This function disables the System PLL PFD.

Parameters

- pfd – Which PFD clock to disable.

bool CLOCK_IsSysPfdEnabled(*clock_pfd_t* pfd)

Check if Sys PFD is enabled.

Parameters

- pfd – PFD control name

Returns

PFD bypass status.

- true: power on.
- false: power off.

void CLOCK_InitUsb1Pfd(*clock_pfd_t* pfd, uint8_t pfdFrac)

Initialize the USB1 PLL PFD.

This function initializes the USB1 PLL PFD. During new value setting, the clock output is disabled to prevent glitch.

Note: It is recommended that PFD settings are kept between 12-35.

Parameters

- pfd – Which PFD clock to enable.

- pfdFrac – The PFD FRAC value.

void CLOCK_DeinitUsb1Pfd(*clock_pfd_t* pfd)

De-initialize the USB1 PLL PFD.

This function disables the USB1 PLL PFD.

Parameters

- pfd – Which PFD clock to disable.

bool CLOCK_IsUsb1PfdEnabled(*clock_pfd_t* pfd)

Check if Usb1 PFD is enabled.

Parameters

- pfd – PFD control name.

Returns

PFD bypass status.

- true: power on.
- false: power off.

uint32_t CLOCK_GetSysPfdFreq(*clock_pfd_t* pfd)

Get current System PLL PFD output frequency.

This function get current output frequency of specific System PLL PFD

Parameters

- pfd – pfd name to get frequency.

Returns

The PFD output frequency in hertz.

uint32_t CLOCK_GetUsb1PfdFreq(*clock_pfd_t* pfd)

Get current USB1 PLL PFD output frequency.

This function get current output frequency of specific USB1 PLL PFD

Parameters

- pfd – pfd name to get frequency.

Returns

The PFD output frequency in hertz.

bool CLOCK_EnableUsbhs0PhyPllClock(*clock_usb_phy_src_t* src, uint32_t freq)

Enable USB HS PHY PLL clock.

This function enables the internal 480MHz USB PHY PLL clock.

Parameters

- src – USB HS PHY PLL clock source.
- freq – The frequency specified by src.

Return values

- true – The clock is set successfully.
- false – The clock source is invalid to get proper USB HS clock.

void CLOCK_DisableUsbhs0PhyPllClock(void)

Disable USB HS PHY PLL clock.

This function disables USB HS PHY PLL clock.

void CLOCK_SetClockOutput1(*clock_output1_selection_t* selection, *clock_output_divider_t* divider)

Set the clock source and the divider of the clock output1.

Parameters

- selection – The clock source to be output, please refer to *clock_output1_selection_t*.
- divider – The divider of the output clock signal, please refer to *clock_output_divider_t*.

void CLOCK_SetClockOutput2(*clock_output2_selection_t* selection, *clock_output_divider_t* divider)

Set the clock source and the divider of the clock output2.

Parameters

- selection – The clock source to be output, please refer to *clock_output2_selection_t*.
- divider – The divider of the output clock signal, please refer to *clock_output_divider_t*.

uint32_t CLOCK_GetClockOutCLKO1Freq(void)

Get the frequency of clock output1 clock signal.

Returns
The frequency of clock output1 clock signal.

uint32_t CLOCK_GetClockOutClkO2Freq(void)

Get the frequency of clock output2 clock signal.

Returns
The frequency of clock output2 clock signal.

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.

---

struct _clock_usb_pll_config

#include <fsl_clock.h> PLL configuration for USB.

Public Members

uint8_t loopDivider

PLL loop divider. 0 - Fout=Fref*20; 1 - Fout=Fref*22

uint8_t src

Pll clock source, reference _clock_pll_clk_src

struct __clock__sys__pll__config

#include <fsl_clock.h> PLL configuration for System.

Public Members

uint8_t loopDivider

PLL loop divider. Intended to be 1 (528M). 0 - Fout=Fref*20; 1 - Fout=Fref*22

uint32_t numerator

30 bit numerator of fractional loop divider.

uint32_t denominator

30 bit denominator of fractional loop divider

uint8_t src

Pll clock source, reference _clock_pll_clk_src

uint16_t ss__stop

Stop value to get frequency change.

uint8_t ss__enable

Enable spread spectrum modulation

uint16_t ss__step

Step value to get frequency change step.

struct __clock__audio__pll__config

#include <fsl_clock.h> PLL configuration for AUDIO and VIDEO.

Public Members

uint8_t loopDivider

PLL loop divider. Valid range for DIV_SELECT divider value: 27~54.

uint8_t postDivider

Divider after the PLL, should only be 1, 2, 4, 8, 16.

uint32_t numerator

30 bit numerator of fractional loop divider.

uint32_t denominator

30 bit denominator of fractional loop divider

uint8_t src

Pll clock source, reference _clock_pll_clk_src

struct __clock__enet__pll__config

#include <fsl_clock.h> PLL configuration for ENET.

Public Members

bool enableClkOutput

Power on and enable PLL clock output for ENET0 (ref_enetpll0).

bool enableClkOutput500M

Power on and enable PLL clock output for ENET (ref_enetpll500M).

bool enableClkOutput25M

Power on and enable PLL clock output for ENET1 (ref_enetpll1).

uint8_t loopDivider

Controls the frequency of the ENET0 reference clock. b00 25MHz b01 50MHz b10 100MHz (not 50% duty cycle) b11 125MHz

uint8_t src

Pll clock source, reference _clock_pll_clk_src

2.8 CMP: Analog Comparator Driver

void CMP_Init(CMP_Type *base, const *cmp_config_t* *config)

Initializes the CMP.

This function initializes the CMP module. The operations included are as follows.

- Enabling the clock for CMP module.
- Configuring the comparator.
- Enabling the CMP module. Note that for some devices, multiple CMP instances share the same clock gate. In this case, to enable the clock for any instance enables all CMPs. See the appropriate MCU reference manual for the clock assignment of the CMP.

Parameters

- base – CMP peripheral base address.
- config – Pointer to the configuration structure.

void CMP_Deinit(CMP_Type *base)

De-initializes the CMP module.

This function de-initializes the CMP module. The operations included are as follows.

- Disabling the CMP module.
- Disabling the clock for CMP module.

This function disables the clock for the CMP. Note that for some devices, multiple CMP instances share the same clock gate. In this case, before disabling the clock for the CMP, ensure that all the CMP instances are not used.

Parameters

- base – CMP peripheral base address.

static inline void CMP_Enable(CMP_Type *base, bool enable)

Enables/disables the CMP module.

Parameters

- base – CMP peripheral base address.
- enable – Enables or disables the module.

void CMP_GetDefaultConfig(*cmp_config_t* *config)

Initializes the CMP user configuration structure.

This function initializes the user configuration structure to these default values.

```
config->enableCmp           = true;
config->hysteresisMode      = kCMP_HysteresisLevel0;
config->enableHighSpeed     = false;
config->enableInvertOutput  = false;
config->useUnfilteredOutput = false;
config->enablePinOut        = false;
config->enableTriggerMode   = false;
```

Parameters

- config – Pointer to the configuration structure.

void CMP_SetInputChannels(CMP_Type *base, uint8_t positiveChannel, uint8_t negativeChannel)

Sets the input channels for the comparator.

This function sets the input channels for the comparator. Note that two input channels cannot be set the same way in the application. When the user selects the same input from the analog mux to the positive and negative port, the comparator is disabled automatically.

Parameters

- base – CMP 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.

void CMP_EnableDMA(CMP_Type *base, bool enable)

Enables/disables the DMA request for rising/falling events.

This function enables/disables the DMA request for rising/falling events. Either event triggers the generation of the DMA request from CMP if the DMA feature is enabled. Both events are ignored for generating the DMA request from the CMP if the DMA is disabled.

Parameters

- base – CMP peripheral base address.
- enable – Enables or disables the feature.

static inline void CMP_EnableWindowMode(CMP_Type *base, bool enable)

Enables/disables the window mode.

Parameters

- base – CMP peripheral base address.
- enable – Enables or disables the feature.

static inline void CMP_EnablePassThroughMode(CMP_Type *base, bool enable)

Enables/disables the pass through mode.

Parameters

- base – CMP peripheral base address.
- enable – Enables or disables the feature.

void CMP_SetFilterConfig(CMP_Type *base, const *cmp_filter_config_t* *config)

Configures the filter.

Parameters

- base – CMP peripheral base address.
- config – Pointer to the configuration structure.

void CMP_SetDACConfig(CMP_Type *base, const *cmp_dac_config_t* *config)

Configures the internal DAC.

Parameters

- base – CMP peripheral base address.
- config – Pointer to the configuration structure. "NULL" disables the feature.

void CMP_EnableInterrupts(CMP_Type *base, uint32_t mask)

Enables the interrupts.

Parameters

- base – CMP peripheral base address.
- mask – Mask value for interrupts. See "_cmp_interrupt_enable".

void CMP_DisableInterrupts(CMP_Type *base, uint32_t mask)

Disables the interrupts.

Parameters

- base – CMP peripheral base address.
- mask – Mask value for interrupts. See "_cmp_interrupt_enable".

uint32_t CMP_GetStatusFlags(CMP_Type *base)

Gets the status flags.

Parameters

- base – CMP peripheral base address.

Returns

Mask value for the asserted flags. See "_cmp_status_flags".

void CMP_ClearStatusFlags(CMP_Type *base, uint32_t mask)

Clears the status flags.

Parameters

- base – CMP peripheral base address.
- mask – Mask value for the flags. See "_cmp_status_flags".

FSL_CMP_DRIVER_VERSION

CMP driver version 2.0.3.

enum _cmp_interrupt_enable

Interrupt enable/disable mask.

Values:

enumerator kCMP_OutputRisingInterruptEnable

Comparator interrupt enable rising.

enumerator kCMP_OutputFallingInterruptEnable

Comparator interrupt enable falling.

enum _cmp_status_flags

Status flags' mask.

Values:

enumerator kCMP_OutputRisingEventFlag

Rising-edge on the comparison output has occurred.

enumerator kCMP_OutputFallingEventFlag

Falling-edge on the comparison output has occurred.

enumerator kCMP_OutputAssertEventFlag

Return the current value of the analog comparator output.

enum _cmp_hysteresis_mode

CMP Hysteresis mode.

Values:

enumerator kCMP_HysteresisLevel0

Hysteresis level 0.

enumerator kCMP_HysteresisLevel1

Hysteresis level 1.

enumerator kCMP_HysteresisLevel2

Hysteresis level 2.

enumerator kCMP_HysteresisLevel3

Hysteresis level 3.

enum _cmp_reference_voltage_source

CMP Voltage Reference source.

Values:

enumerator kCMP_VrefSourceVin1

Vin1 is selected as a resistor ladder network supply reference Vin.

enumerator kCMP_VrefSourceVin2

Vin2 is selected as a resistor ladder network supply reference Vin.

typedef enum *_cmp_hysteresis_mode* cmp_hysteresis_mode_t

CMP Hysteresis mode.

typedef enum *_cmp_reference_voltage_source* cmp_reference_voltage_source_t

CMP Voltage Reference source.

typedef struct *_cmp_config* cmp_config_t

Configures the comparator.

typedef struct *_cmp_filter_config* cmp_filter_config_t

Configures the filter.

typedef struct *_cmp_dac_config* cmp_dac_config_t

Configures the internal DAC.

struct _cmp_config

#include <fsl_cmp.h> Configures the comparator.

Public Members

bool enableCmp

Enable the CMP module.

cmp_hysteresis_mode_t hysteresisMode

CMP Hysteresis mode.

bool enableHighSpeed

Enable High-speed (HS) comparison mode.

bool enableInvertOutput

Enable the inverted comparator output.

bool useUnfilteredOutput

Set the compare output(COUT) to equal COUTA(true) or COUT(false).

bool enablePinOut

The comparator output is available on the associated pin.

bool enableTriggerMode

Enable the trigger mode.

struct _cmp_filter_config

#include <fsl_cmp.h> Configures the filter.

Public Members

bool enableSample
Using the external SAMPLE as a sampling clock input or using a divided bus clock.

uint8_t filterCount
Filter Sample Count. Available range is 1-7; 0 disables the filter.

uint8_t filterPeriod
Filter Sample Period. The divider to the bus clock. Available range is 0-255.

struct _cmp_dac_config
#include <fsl_cmp.h> Configures the internal DAC.

Public Members

cmp_reference_voltage_source_t referenceVoltageSource
Supply voltage reference source.

uint8_t DACValue
Value for the DAC Output Voltage. Available range is 0-63.

2.9 DCDC: DCDC Converter

void DCDC_Init(DCDC_Type *base, *dcdc_config_t* *config)
Enable the access to DCDC registers.

Parameters

- base – DCDC peripheral base address.
- config – Pointer to the configuration structure.

void DCDC_Deinit(DCDC_Type *base)
Disable the access to DCDC registers.

Parameters

- base – DCDC peripheral base address.

void DCDC_GetDefaultConfig(DCDC_Type *base, *dcdc_config_t* *config)
brief Get the default setting for DCDC user configuration structure.

This function initializes the user configuration structure to a default value. The default values are: code config->controlMode = kDCDC_StaticControl; config->trimInputMode = kDCDC_SampleTrimInput; config->enableDcdcTimeout = false; config->enableSwitchingConverterOutput = false; endcode

param config Pointer to configuration structure. See to “dcdc_config_t”

uint32_t DCDC_GetstatusFlags(DCDC_Type *base)
Get DCDC status flags.

Parameters

- base – peripheral base address.

Returns
Mask of asserted status flags. See to “_dcdc_status_flags_t”.

void DCDC_EnterLowPowerMode(DCDC_Type *base, *dcdc_low_power_mode_t* mode)

Make DCDC enter into low power modes.

Parameters

- base – DCDC peripheral base address.
- mode – DCDC low power mode selection. See to "_dcdc_low_power_mode"

static inline void DCDC_EnableOutputRangeComparator(DCDC_Type *base, bool enable)

Enable the output range comparator.

The output range comparator is disabled by default.

Parameters

- base – DCDC peripheral base address.
- enable – Enable the feature or not.

void DCDC_SetClockSource(DCDC_Type *base, *dcdc_clock_source_t* clockSource)

Configure the DCDC clock source.

Parameters

- base – DCDC peripheral base address.
- clockSource – Clock source for DCDC. See to "dcdc_clock_source_t".

void DCDC_GetDefaultDetectionConfig(*dcdc_detection_config_t* *config)

Get the default setting for detection configuration.

The default configuration are set according to responding registers' setting when powered on. They are:

```
config->enableXtalokDetection = false;
config->powerDownOverVoltageDetection = true;
config->powerDownLowVlotageDetection = false;
config->powerDownOverCurrentDetection = true;
config->powerDownPeakCurrentDetection = true;
config->powerDownZeroCrossDetection = true;
config->OverCurrentThreshold = kDCDC_OverCurrentThresholdAlt0;
config->PeakCurrentThreshold = kDCDC_PeakCurrentThresholdAlt0;
```

Parameters

- config – Pointer to configuration structure. See to "dcdc_detection_config_t"

void DCDC_SetDetectionConfig(DCDC_Type *base, const *dcdc_detection_config_t* *config)

Configure the DCDC detection.

Parameters

- base – DCDC peripheral base address.
- config – Pointer to configuration structure. See to "dcdc_detection_config_t"

void DCDC_GetDefaultLowPowerConfig(*dcdc_low_power_config_t* *config)

Get the default setting for low power configuration.

The default configuration are set according to responding registers' setting when powered on. They are:

```
config->enableOverloadDetection = true;
config->enableAdjustHystereticValue = false;
config->countChargingTimePeriod = kDCDC_CountChargingTimePeriod8Cycle;
config->countChargingTimeThreshold = kDCDC_CountChargingTimeThreshold32;
```

Parameters

- config – Pointer to configuration structure. See to "dcdc_low_power_config_t"

void DCDC_SetLowPowerConfig(DCDC_Type *base, const *dcdc_low_power_config_t* *config)

Configure the DCDC low power.

Parameters

- base – DCDC peripheral base address.
- config – Pointer to configuration structure. See to "dcdc_low_power_config_t".

void DCDC_ResetCurrentAlertSignal(DCDC_Type *base, bool enable)

Reset current alert signal. Alert signal is generate by peak current detection.

Parameters

- base – DCDC peripheral base address.
- enable – Switcher to reset signal. True means reset signal. False means don't reset signal.

static inline void DCDC_SetBandgapVoltageTrimValue(DCDC_Type *base, uint32_t trimValue)

Set the bangap trim value to trim bandgap voltage.

Parameters

- base – DCDC peripheral base address.
- trimValue – The bangap trim value. Available range is 0U-31U.

void DCDC_GetDefaultLoopControlConfig(*dcdc_loop_control_config_t* *config)

Get the default setting for loop control configuration.

The default configuration are set according to responding registers' setting when powered on. They are:

```
config->enableCommonHysteresis = false;
config->enableCommonThresholdDetection = false;
config->enableInvertHysteresisSign = false;
config->enableRCThresholdDetection = false;
config->enableRCScaleCircuit = 0U;
config->complementFeedForwardStep = 0U;
```

Parameters

- config – Pointer to configuration structure. See to "dcdc_loop_control_config_t"

void DCDC_SetLoopControlConfig(DCDC_Type *base, const *dcdc_loop_control_config_t* *config)

Configure the DCDC loop control.

Parameters

- base – DCDC peripheral base address.
- config – Pointer to configuration structure. See to "dcdc_loop_control_config_t".

void DCDC_SetMinPowerConfig(DCDC_Type *base, const *dcdc_min_power_config_t* *config)

Configure for the min power.

Parameters

- base – DCDC peripheral base address.
- config – Pointer to configuration structure. See to "dcdc_min_power_config_t".

static inline void DCDC_SetLPComparatorBiasValue(DCDC_Type *base, *dcdc_comparator_current_bias_t* biasVaule)

Set the current bias of low power comparator.

Parameters

- base – DCDC peripheral base address.
- biasVaule – The current bias of low power comparator. Refer to "dcdc_comparator_current_bias_t".

static inline void DCDC_LockTargetVoltage(DCDC_Type *base)

Lock target voltage.

Parameters

- base – DCDC peripheral base address.

void DCDC_AdjustTargetVoltage(DCDC_Type *base, uint32_t VDDRun, uint32_t VDDStandby)

Adjust the target voltage of VDD_SOC in run mode and low power mode.

Deprecated:

Do not use this function. It has been superceded by *DCDC_AdjustRunTargetVoltage* and *DCDC_AdjustLowPowerTargetVoltage*

This function is to adjust the target voltage of DCDC output. Change them and finally wait until the output is stabled. Set the target value of run mode the same as low power mode before entering power save mode, because DCDC will switch back to run mode if it detects the current loading is larger than about 50 mA(typical value).

Parameters

- base – DCDC peripheral base address.
- VDDRun – Target value in run mode. 25 mV each step from 0x00 to 0x1F. 00 is for 0.8V, 0x1F is for 1.575V.
- VDDStandby – Target value in low power mode. 25 mV each step from 0x00 to 0x4. 00 is for 0.9V, 0x4 is for 1.0V.

void DCDC_AdjustRunTargetVoltage(DCDC_Type *base, uint32_t VDDRun)

Adjust the target voltage of VDD_SOC in run mode.

This function is to adjust the target voltage of DCDC output. Change them and finally wait until the output is stabled. Set the target value of run mode the same as low power mode before entering power save mode, because DCDC will switch back to run mode if it detects the current loading is larger than about 50 mA(typical value).

Parameters

- base – DCDC peripheral base address.
- VDDRun – Target value in run mode. 25 mV each step from 0x00 to 0x1F. 00 is for 0.8V, 0x1F is for 1.575V.

void DCDC_AdjustLowPowerTargetVoltage(DCDC_Type *base, uint32_t VDDStandby)

Adjust the target voltage of VDD_SOC in low power mode.

This function is to adjust the target voltage of DCDC output. Change them and finally wait until the output is stabled. Set the target value of run mode the same as low power mode before entering power save mode, because DCDC will switch back to run mode if it detects the current loading is larger than about 50 mA(typical value).

Parameters

- base – DCDC peripheral base address.
- VDDStandby – Target value in low power mode. 25 mV each step from 0x00 to 0x4. 00 is for 0.9V, 0x4 is for 1.0V.

void DCDC_SetInternalRegulatorConfig(DCDC_Type *base, const *dcdc_internal_regulator_config_t* *config)

Configure the DCDC internal regulator.

Parameters

- base – DCDC peripheral base address.
- config – Pointer to configuration structure. See to "dcdc_internal_regulator_config_t".

static inline void DCDC_EnableImproveTransition(DCDC_Type *base, bool enable)

Enable/Disable to improve the transition from heavy load to light load. It is valid while zero cross detection is enabled. If ouput exceeds the threshold, DCDC would return CCM from DCM.

Parameters

- base – DCDC peripheral base address.
- enable – Enable the feature or not.

void DCDC_BootIntoDCM(DCDC_Type *base)

Boot DCDC into DCM(discontinous conduction mode).

pwd_zcd=0x0; pwd_cmp_offset=0x0; dcdc_loopctrl_en_rcscale= 0x5; DCM_set_ctrl=1'b1;

Parameters

- base – DCDC peripheral base address.

void DCDC_BootIntoCCM(DCDC_Type *base)

Boot DCDC into CCM(continous conduction mode).

pwd_zcd=0x1; pwd_cmp_offset=0x0; dcdc_loopctrl_en_rcscale=0x3;

Parameters

- base – DCDC peripheral base address.

enum _dcdc_status_flags_t

DCDC status flags.

Values:

enumerator kDCDC_LockedOKStatus

Indicate DCDC status. 1'b1: DCDC already settled 1'b0: DCDC is settling.

enum _dcdc_comparator_current_bias

The current bias of low power comparator.

Values:

enumerator kDCDC_ComparatorCurrentBias50nA

The current bias of low power comparator is 50nA.

enumerator kDCDC_ComparatorCurrentBias100nA

The current bias of low power comparator is 100nA.

enumerator kDCDC_ComparatorCurrentBias200nA

The current bias of low power comparator is 200nA.

enumerator kDCDC_ComparatorCurrentBias400nA

The current bias of low power comparator is 400nA.

enum _dcdc_over_current_threshold

The threshold of over current detection.

Values:

enumerator kDCDC_OverCurrentThresholdAlt0

1A in the run mode, 0.25A in the power save mode.

enumerator kDCDC_OverCurrentThresholdAlt1

2A in the run mode, 0.25A in the power save mode.

enumerator kDCDC_OverCurrentThresholdAlt2

1A in the run mode, 0.2A in the power save mode.

enumerator kDCDC_OverCurrentThresholdAlt3

2A in the run mode, 0.2A in the power save mode.

enum _dcdc_peak_current_threshold

The threshold if peak current detection.

Values:

enumerator kDCDC_PeakCurrentThresholdAlt0

150mA peak current threshold.

enumerator kDCDC_PeakCurrentThresholdAlt1

250mA peak current threshold.

enumerator kDCDC_PeakCurrentThresholdAlt2

350mA peak current threshold.

enumerator kDCDC_PeakCurrentThresholdAlt3

450mA peak current threshold.

enumerator kDCDC_PeakCurrentThresholdAlt4
550mA peak current threshold.

enumerator kDCDC_PeakCurrentThresholdAlt5
650mA peak current threshold.

enum _dcdc_count_charging_time_period
The period of counting the charging times in power save mode.
Values:

enumerator kDCDC_CountChargingTimePeriod8Cycle
Eight 32k cycle.

enumerator kDCDC_CountChargingTimePeriod16Cycle
Sixteen 32k cycle.

enum _dcdc_count_charging_time_threshold
The threshold of the counting number of charging times.
Values:

enumerator kDCDC_CountChargingTimeThreshold32
0x0: 32.

enumerator kDCDC_CountChargingTimeThreshold64
0x1: 64.

enumerator kDCDC_CountChargingTimeThreshold16
0x2: 16.

enumerator kDCDC_CountChargingTimeThreshold8
0x3: 8.

enum _dcdc_clock_source
Oscillator clock option.
Values:

enumerator kDCDC_ClockAutoSwitch
Automatic clock switch from internal oscillator to external clock.

enumerator kDCDC_ClockInternalOsc
Use internal oscillator.

enumerator kDCDC_ClockExternalOsc
Use external 24M crystal oscillator.

enum _dcdc_low_power_mode
DCDC low power modes.
Values:

enumerator kDCDC_StandbyMode
Standby mode.

enumerator kDCDC_LowPowerMode
Low power mode.

enumerator kDCDC_GpcStandbyLowPowerMode
low power mode for GPC standby request.

enum _dcdc_control_mode
DCDC control mode.
Values:

enumerator kDCDC_StaticControl
Static control.

enumerator kDCDC_SetPointControl
Controlled by GPC set points.

enum _dcdc_trim_input_mode
DCDC trim input mode.
Values:

enumerator kDCDC_SampleTrimInput
Sample trim input.

enumerator kDCDC_HoldTrimInput
Hold trim input.

typedef enum *_dcdc_comparator_current_bias* dcdc_comparator_current_bias_t
The current bias of low power comparator.

typedef enum *_dcdc_over_current_threshold* dcdc_over_current_threshold_t
The threshold of over current detection.

typedef enum *_dcdc_peak_current_threshold* dcdc_peak_current_threshold_t
The threshold if peak current detection.

typedef enum *_dcdc_count_charging_time_period* dcdc_count_charging_time_period_t
The period of counting the charging times in power save mode.

typedef enum *_dcdc_count_charging_time_threshold* dcdc_count_charging_time_threshold_t
The threshold of the counting number of charging times.

typedef enum *_dcdc_clock_source* dcdc_clock_source_t
Oscillator clock option.

typedef enum *_dcdc_low_power_mode* dcdc_low_power_mode_t
DCDC low power modes.

typedef enum *_dcdc_control_mode* dcdc_control_mode_t
DCDC control mode.

typedef enum *_dcdc_trim_input_mode* dcdc_trim_input_mode_t
DCDC trim input mode.

typedef struct *_dcdc_config* dcdc_config_t
Configuration for DCDC.

typedef struct *_dcdc_detection_config* dcdc_detection_config_t
Configuration for DCDC detection.

typedef struct *_dcdc_loop_control_config* dcdc_loop_control_config_t
Configuration for the loop control.

typedef struct *_dcdc_low_power_config* dcdc_low_power_config_t
Configuration for DCDC low power.

typedef struct *_dcdc_internal_regulator_config* dcdc_internal_regulator_config_t
Configuration for DCDC internal regulator.

typedef struct *_dcdc_min_power_config* dcdc_min_power_config_t
Configuration for min power setting.

FSL_DCDC_DRIVER_VERSION
DCDC driver version.
Version 2.3.0.

struct _dcdc_config
#include <fsl_dcdc.h> Configuration for DCDC.

Public Members

dcdc_control_mode_t controlMode
DCDC control mode.

dcdc_trim_input_mode_t trimInputMode
Hold trim input.

bool enableDcdcTimeout
Enable internal count for DCDC_OK timeout.

bool enableSwitchingConverterOutput
Enable the VDDIO switching converter output.

struct __dcdc__detection__config

#include <fsl_dcdc.h> Configuration for DCDC detection.

Public Members

bool enableXtalokDetection

Enable xtalok detection circuit.

bool powerDownOverVoltageDetection

Power down over-voltage detection comparator.

bool powerDownLowVlotageDetection

Power down low-voltage detection comparator.

bool powerDownOverCurrentDetection

Power down over-current detection.

bool powerDownPeakCurrentDetection

Power down peak-current detection.

bool powerDownZeroCrossDetection

Power down the zero cross detection function for discontinuous conductor mode.

dcdc_over_current_threshold_t OverCurrentThreshold

The threshold of over current detection.

dcdc_peak_current_threshold_t PeakCurrentThreshold

The threshold of peak current detection.

struct __dcdc__loop__control__config

#include <fsl_dcdc.h> Configuration for the loop control.

Public Members

bool enableCommonHysteresis

Enable hysteresis in switching converter common mode analog comparators. This feature will improve transient supply ripple and efficiency.

bool enableCommonThresholdDetection

Increase the threshold detection for common mode analog comparator.

bool enableDifferentialHysteresis

Enable hysteresis in switching converter differential mode analog comparators. This feature will improve transient supply ripple and efficiency.

bool enableDifferentialThresholdDetection

Increase the threshold detection for differential mode analog comparators.

bool enableInvertHysteresisSign

Invert the sign of the hysteresis in DC-DC analog comparators.

bool enableRCThresholdDetection

Increase the threshold detection for RC scale circuit.

uint32_t enableRCScaleCircuit

Available range is 0~7. Enable analog circuit of DC-DC converter to respond faster under transient load conditions.

uint32_t complementFeedForwardStep

Available range is 0~7. Two's complement feed forward step in duty cycle in the switching DC-DC converter. Each time this field makes a transition from 0x0, the loop filter of the DC-DC converter is stepped once by a value proportional to the change. This can be used to force a certain control loop behavior, such as improving response under known heavy load transients.

struct _dcdc_low_power_config

#include <fsl_dcdc.h> Configuration for DCDC low power.

Public Members

bool enableAdjustHystereticValue

Adjust hysteretic value in low power from 12.5mV to 25mV.

dcdc_count_charging_time_period_t countChargingTimePeriod

The period of counting the charging times in power save mode.

dcdc_count_charging_time_threshold_t countChargingTimeThreshold

the threshold of the counting number of charging times during the period that lp_overload_freq_sel sets in power save mode.

struct _dcdc_internal_regulator_config

#include <fsl_dcdc.h> Configuration for DCDC internal regulator.

Public Members

bool enableLoadResistor

control the load resistor of the internal regulator of DCDC, the load resistor is connected as default "true", and need set to "false" to disconnect the load resistor.

uint32_t feedbackPoint

Available range is 0~3. Select the feedback point of the internal regulator.

struct _dcdc_min_power_config

#include <fsl_dcdc.h> Configuration for min power setting.

Public Members

bool enableUseHalfFreqForContinuous
: Set DCDC clock to half frequency for the continuous mode.

2.10 DCP: Data Co-Processor

FSL_DCP_DRIVER_VERSION
: DCP driver version. Version 2.1.8.

  Current version: 2.1.8

  Change log:

  - Version 2.1.8
    - Bug Fix
      * Fix missing OTP flag in DCP Control0 field when using OTP UNIQUE keys.
  - Version 2.1.7
    - Bug Fix
      * Reduce optimization level for critical functions working with SRF.
  - Version 2.1.6
    - Bug Fix
      * MISRA C-2012 issue fix.
  - Version 2.1.5
    - Improvements
      * Add support for DCACHE.
  - Version 2.1.4
    - Bug Fix
      * Fix CRC-32 computation issue on the code's block boundary size.
  - Version 2.1.3
    - Bug Fix
      * MISRA C-2012 issue fixed: rule 10.1, 10.3, 10.4, 11.9, 14.4, 16.4 and 17.7.
  - Version 2.1.2
    - Fix sign-compare warning in dcp_reverse_and_copy.
  - Version 2.1.1
    - Add DCP status clearing when channel operation is complete
  - 2.1.0
    - Add byte/word swap feature for key, input and output data
  - Version 2.0.0
    - Initial version

enum _dcp_status

DCP status return codes.

Values:

enumerator kStatus_DCP_Again

Non-blocking function shall be called again.

enum _dcp_ch_enable

DCP channel enable.

Values:

enumerator kDCP_chDisable

DCP channel disable

enumerator kDCP_ch0Enable

DCP channel 0 enable

enumerator kDCP_ch1Enable

DCP channel 1 enable

enumerator kDCP_ch2Enable

DCP channel 2 enable

enumerator kDCP_ch3Enable

DCP channel 3 enable

enumerator kDCP_chEnableAll

DCP channel enable all

enum _dcp_ch_int_enable

DCP interrupt enable.

Values:

enumerator kDCP_chIntDisable

DCP interrupts disable

enumerator kDCP_ch0IntEnable

DCP channel 0 interrupt enable

enumerator kDCP_ch1IntEnable

DCP channel 1 interrupt enable

enumerator kDCP_ch2IntEnable

DCP channel 2 interrupt enable

enumerator kDCP_ch3IntEnable

DCP channel 3 interrupt enable

enum _dcp_channel

DCP channel selection.

Values:

enumerator kDCP_Channel0

DCP channel 0.

enumerator kDCP_Channel1

DCP channel 1.

enumerator kDCP_Channel2

DCP channel 2.

enumerator kDCP_Channel3

DCP channel 3.

enum _dcp_key_slot

DCP key slot selection.

Values:

enumerator kDCP_KeySlot0

DCP key slot 0.

enumerator kDCP_KeySlot1

DCP key slot 1.

enumerator kDCP_KeySlot2

DCP key slot 2.

enumerator kDCP_KeySlot3

DCP key slot 3.

enumerator kDCP_OtpKey

DCP OTP key.

enumerator kDCP_OtpUniqueKey

DCP unique OTP key.

enumerator kDCP_PayloadKey

DCP payload key.

enum _dcp_swap

DCP key, input & output swap options.

Values:

enumerator kDCP_NoSwap

enumerator kDCP_KeyByteSwap

enumerator kDCP_KeyWordSwap

enumerator kDCP_InputByteSwap

enumerator kDCP_InputWordSwap

enumerator kDCP_OutputByteSwap

enumerator kDCP_OutputWordSwap

typedef enum *_dcp_ch_enable* _dcp_ch_enable_t
DCP channel enable.

typedef enum *_dcp_ch_int_enable* _dcp_ch_int_enable_t
DCP interrupt enable.

typedef enum *_dcp_channel* dcp_channel_t
DCP channel selection.

typedef enum *_dcp_key_slot* dcp_key_slot_t
DCP key slot selection.

typedef enum *_dcp_swap* dcp_swap_t
DCP key, input & output swap options.

typedef struct *_dcp_work_packet* dcp_work_packet_t
DCP's work packet.

typedef struct *_dcp_handle* dcp_handle_t
Specify DCP's key resource and DCP channel.

typedef struct *_dcp_context* dcp_context_t
DCP's context buffer, used by DCP for context switching between channels.

typedef struct *_dcp_config* dcp_config_t
DCP's configuration structure.

void DCP_Init(DCP_Type *base, const *dcp_config_t* *config)
Enables clock to and enables DCP.

Enable DCP clock and configure DCP.

Parameters

- base – DCP base address
- config – Pointer to configuration structure.

void DCP_Deinit(DCP_Type *base)

Disable DCP clock.

Reset DCP and Disable DCP clock.

Parameters

- base – DCP base address

void DCP_GetDefaultConfig(*dcp_config_t* *config)

Gets the default configuration structure.

This function initializes the DCP configuration structure to a default value. The default values are as follows. dcpConfig->gatherResidualWrites = true; dcpConfig->enableContextCaching = true; dcpConfig->enableContextSwitching = true; dcpConfig->enableChannnel = kDCP_chEnableAll; dcpConfig->enableChannelInterrupt = kDCP_chIntDisable;

Parameters

- config – **[out]** Pointer to configuration structure.

status_t DCP_WaitForChannelComplete(DCP_Type *base, *dcp_handle_t* *handle)

Poll and wait on DCP channel.

Polls the specified DCP channel until current it completes activity.

Parameters

- base – DCP peripheral base address.
- handle – Specifies DCP channel.

Returns

kStatus_Success When data processing completes without error.

Returns

kStatus_Fail When error occurs.

struct _dcp_work_packet

#include <fsl_dcp.h> DCP's work packet.

struct _dcp_handle

#include <fsl_dcp.h> Specify DCP's key resource and DCP channel.

Public Members

dcp_channel_t channel

Specify DCP channel.

dcp_key_slot_t keySlot

For operations with key (such as AES encryption/decryption), specify DCP key slot.

uint32_t swapConfig

For configuration of key, input, output byte/word swap options

struct _dcp_context

#include <fsl_dcp.h> DCP's context buffer, used by DCP for context switching between channels.

struct _dcp_config

#include <fsl_dcp.h> DCP's configuration structure.

Public Members

bool gatherResidualWrites

Enable the ragged writes to the unaligned buffers.

bool enableContextCaching

Enable the caching of contexts between the operations.

bool enableContextSwitching

Enable automatic context switching for the channels.

uint8_t enableChannel

DCP channel enable.

uint8_t enableChannelInterrupt

Per-channel interrupt enable.

2.11 DCP AES blocking driver

status_t DCP_AES_SetKey(DCP_Type *base, *dcp_handle_t* *handle, const uint8_t *key, size_t keySize)

Set AES key to dcp_handle_t struct and optionally to DCP.

Sets the AES key for encryption/decryption with the dcp_handle_t structure. The dcp_handle_t input argument specifies keySlot. If the keySlot is kDCP_OtpKey, the function will check the OTP_KEY_READY bit and will return it's ready to use status. For other keySlot selections, the function will copy and hold the key in dcp_handle_t struct. If the keySlot is one of the four DCP SRAM-based keys (one of kDCP_KeySlot0, kDCP_KeySlot1, kDCP_KeySlot2, kDCP_KeySlot3), this function will also load the supplied key to the specified keySlot in DCP.

Parameters

- base – DCP peripheral base address.
- handle – Handle used for the request.
- key – 0-mod-4 aligned pointer to AES key.
- keySize – AES key size in bytes. Shall equal 16.

Returns

status from set key operation

status_t DCP_AES_EncryptEcb(DCP_Type *base, *dcp_handle_t* *handle, const uint8_t *plaintext, uint8_t *ciphertext, size_t size)

Encrypts AES on one or multiple 128-bit block(s).

Encrypts AES. The source plaintext and destination ciphertext can overlap in system memory.

Parameters

- base – DCP peripheral base address
- handle – Handle used for this request.
- 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.

Returns
Status from encrypt operation

status_t DCP_AES_DecryptEcb(DCP_Type *base, *dcp_handle_t* *handle, const uint8_t *ciphertext, uint8_t *plaintext, size_t size)

Decrypts AES on one or multiple 128-bit block(s).

Decrypts AES. The source ciphertext and destination plaintext can overlap in system memory.

Parameters

- base – DCP peripheral base address
- handle – Handle used for this request.
- ciphertext – Input plain text to encrypt
- plaintext – **[out]** Output cipher text
- size – Size of input and output data in bytes. Must be multiple of 16 bytes.

Returns
Status from decrypt operation

status_t DCP_AES_EncryptCbc(DCP_Type *base, *dcp_handle_t* *handle, const uint8_t *plaintext, uint8_t *ciphertext, size_t size, const uint8_t iv[16])

Encrypts AES using CBC block mode.

Encrypts AES using CBC block mode. The source plaintext and destination ciphertext can overlap in system memory.

Parameters

- base – DCP peripheral base address
- handle – Handle used for this request.
- 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.

Returns
Status from encrypt operation

status_t DCP_AES_DecryptCbc(DCP_Type *base, *dcp_handle_t* *handle, const uint8_t *ciphertext, uint8_t *plaintext, size_t size, const uint8_t iv[16])

Decrypts AES using CBC block mode.

Decrypts AES using CBC block mode. The source ciphertext and destination plaintext can overlap in system memory.

Parameters

- base – DCP peripheral base address
- handle – Handle used for this request.

- 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.

Returns
Status from decrypt operation

DCP_AES_BLOCK_SIZE
AES block size in bytes

2.12 DCP HASH driver

enum _dcp_hash_algo_t
Supported cryptographic block cipher functions for HASH creation.

Values:

enumerator kDCP_Sha1
SHA_1

enumerator kDCP_Sha256
SHA_256

enumerator kDCP_Crc32
CRC_32

typedef enum *_dcp_hash_algo_t* dcp_hash_algo_t
Supported cryptographic block cipher functions for HASH creation.

typedef struct *_dcp_hash_ctx_t* dcp_hash_ctx_t
Storage type used to save hash context.

status_t DCP_HASH_Init(DCP_Type *base, *dcp_handle_t* *handle, *dcp_hash_ctx_t* *ctx, *dcp_hash_algo_t* algo)
Initialize HASH context.

This function initializes the HASH.

Parameters

- base – DCP peripheral base address
- handle – Specifies the DCP channel used for hashing.
- ctx – **[out]** Output hash context
- algo – Underlaying algorithm to use for hash computation.

Returns
Status of initialization

status_t DCP_HASH_Update(DCP_Type *base, *dcp_hash_ctx_t* *ctx, const uint8_t *input, size_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. The functions blocks. If it returns kStatus_Success, the running hash has been updated (DCP has processed the input data), so the memory at the input pointer can be released back to system. The DCP context buffer is updated with the running hash and with all necessary information to support possible context switch.

Parameters

- base – DCP peripheral base address
- ctx – **[inout]** HASH context
- input – Input data
- inputSize – Size of input data in bytes

Returns

Status of the hash update operation

status_t DCP_HASH_Finish(DCP_Type *base, *dcp_hash_ctx_t* *ctx, uint8_t *output, size_t *outputSize)

Finalize hashing.

Outputs the final hash (computed by *DCP_HASH_Update()*) and erases the context.

Parameters

- base – DCP peripheral base address
- ctx – **[inout]** Input hash context
- output – **[out]** Output hash data
- outputSize – **[inout]** Optional parameter (can be passed as NULL). On function entry, it specifies the size of output[] buffer. On function return, it stores the number of updated output bytes.

Returns

Status of the hash finish operation

status_t DCP_HASH(DCP_Type *base, *dcp_handle_t* *handle, *dcp_hash_algo_t* algo, const uint8_t *input, size_t inputSize, uint8_t *output, size_t *outputSize)

Create HASH on given data.

Perform the full SHA or CRC32 in one function call. The function is blocking.

Parameters

- base – DCP peripheral base address
- handle – Handle used for the request.
- algo – Underlaying algorithm to use for hash computation.
- input – Input data
- inputSize – Size of input data 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.

DCP_HASH_CAVP_COMPATIBLE

DCP_SHA_BLOCK_SIZE

DCP HASH Context size.

internal buffer block size

DCP_HASH_BLOCK_SIZE

DCP hash block size

DCP_HASH_CTX_SIZE

DCP HASH Context size.

struct _dcp_hash_ctx_t

#include <fsl_dcp.h> Storage type used to save hash context.

2.13 DCP AES non-blocking driver

status_t DCP_AES_EncryptEcbNonBlocking(DCP_Type *base, *dcp_handle_t* *handle, *dcp_work_packet_t* *dcpPacket, const uint8_t *plaintext, uint8_t *ciphertext, size_t size)

Encrypts AES using the ECB block mode.

Puts AES ECB encrypt work packet to DCP channel.

Parameters

- base – DCP peripheral base address
- handle – Handle used for this request.
- dcpPacket – **[out]** Memory for the DCP work packet.
- 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.

Returns

kStatus_Success The work packet has been scheduled at DCP channel.

Returns

kStatus_DCP_Again The DCP channel is busy processing previous request.

status_t DCP_AES_DecryptEcbNonBlocking(DCP_Type *base, *dcp_handle_t* *handle, *dcp_work_packet_t* *dcpPacket, const uint8_t *ciphertext, uint8_t *plaintext, size_t size)

Decrypts AES using ECB block mode.

Puts AES ECB decrypt dcpPacket to DCP input job ring.

Parameters

- base – DCP peripheral base address
- handle – Handle used for this request.
- dcpPacket – **[out]** Memory for the DCP work packet.
- 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.

Returns
kStatus_Success The work packet has been scheduled at DCP channel.

Returns
kStatus_DCP_Again The DCP channel is busy processing previous request.

status_t DCP_AES_EncryptCbcNonBlocking(DCP_Type *base, *dcp_handle_t* *handle, *dcp_work_packet_t* *dcpPacket, const uint8_t *plaintext, uint8_t *ciphertext, size_t size, const uint8_t *iv)

Encrypts AES using CBC block mode.

Puts AES CBC encrypt dcpPacket to DCP input job ring.

Parameters

- base – DCP peripheral base address
- handle – Handle used for this request. Specifies jobRing.
- dcpPacket – **[out]** Memory for the DCP work packet.
- 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.

Returns
kStatus_Success The work packet has been scheduled at DCP channel.

Returns
kStatus_DCP_Again The DCP channel is busy processing previous request.

status_t DCP_AES_DecryptCbcNonBlocking(DCP_Type *base, *dcp_handle_t* *handle, *dcp_work_packet_t* *dcpPacket, const uint8_t *ciphertext, uint8_t *plaintext, size_t size, const uint8_t *iv)

Decrypts AES using CBC block mode.

Puts AES CBC decrypt dcpPacket to DCP input job ring.

Parameters

- base – DCP peripheral base address
- handle – Handle used for this request. Specifies jobRing.
- dcpPacket – **[out]** Memory for the DCP work packet.
- 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.

Returns
kStatus_Success The work packet has been scheduled at DCP channel.

Returns
kStatus_DCP_Again The DCP channel is busy processing previous request.

2.14 DMAMUX: Direct Memory Access Multiplexer Driver

void DMAMUX_Init(DMAMUX_Type *base)

Initializes the DMAMUX peripheral.

This function ungates the DMAMUX clock.

Parameters

- base – DMAMUX peripheral base address.

void DMAMUX_Deinit(DMAMUX_Type *base)

Deinitializes the DMAMUX peripheral.

This function gates the DMAMUX clock.

Parameters

- base – DMAMUX peripheral base address.

static inline void DMAMUX_EnableChannel(DMAMUX_Type *base, uint32_t channel)

Enables the DMAMUX channel.

This function enables the DMAMUX channel.

Parameters

- base – DMAMUX peripheral base address.
- channel – DMAMUX channel number.

static inline void DMAMUX_DisableChannel(DMAMUX_Type *base, uint32_t channel)

Disables the DMAMUX channel.

This function disables the DMAMUX channel.

Note: The user must disable the DMAMUX channel before configuring it.

Parameters

- base – DMAMUX peripheral base address.
- channel – DMAMUX channel number.

static inline void DMAMUX_SetSource(DMAMUX_Type *base, uint32_t channel, int32_t source)

Configures the DMAMUX channel source.

Parameters

- base – DMAMUX peripheral base address.
- channel – DMAMUX channel number.
- source – Channel source, which is used to trigger the DMA transfer.User need to use the dma_request_source_t type as the input parameter.

static inline void DMAMUX_EnablePeriodTrigger(DMAMUX_Type *base, uint32_t channel)

Enables the DMAMUX period trigger.

This function enables the DMAMUX period trigger feature.

Parameters

- base – DMAMUX peripheral base address.
- channel – DMAMUX channel number.

static inline void DMAMUX_DisablePeriodTrigger(DMAMUX_Type *base, uint32_t channel)

Disables the DMAMUX period trigger.

This function disables the DMAMUX period trigger.

Parameters

- base – DMAMUX peripheral base address.
- channel – DMAMUX channel number.

static inline void DMAMUX_EnableAlwaysOn(DMAMUX_Type *base, uint32_t channel, bool enable)

Enables the DMA channel to be always ON.

This function enables the DMAMUX channel always ON feature.

Parameters

- base – DMAMUX peripheral base address.
- channel – DMAMUX channel number.
- enable – Switcher of the always ON feature. "true" means enabled, "false" means disabled.

FSL_DMAMUX_DRIVER_VERSION

DMAMUX driver version 2.1.4.

DMAMUX_CHANNEL_ENDIAN_CONVERTn(channel)

Macro used for dmamux channel endian convert.

2.15 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. All emda enabled request will be cleared in this function.

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.enableContinuousLinkMode = false;
config.enableHaltOnError = true;
config.enableRoundRobinArbitration = false;
config.enableDebugMode = false;
```

Parameters

- config – A pointer to the eDMA configuration structure.

static inline void EDMA_EnableContinuousChannelLinkMode(DMA_Type *base, bool enable)

Enable/Disable continuous channel link mode.

Note: Do not use continuous link mode with a channel linking to itself if there is only one minor loop iteration per service request, for example, if the channel's NBYTES value is the same as either the source or destination size. The same data transfer profile can be achieved by simply increasing the NBYTES value, which provides more efficient, faster processing.

Parameters

- base – EDMA peripheral base address.
- enable – true is enable, false is disable.

static inline void EDMA_EnableMinorLoopMapping(DMA_Type *base, bool enable)

Enable/Disable minor loop mapping.

The TCDn.word2 is redefined to include individual enable fields, an offset field, and the NBYTES field.

Parameters

- base – EDMA peripheral base address.
- enable – true is enable, false is disable.

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_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.

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.

void EDMA_SetChannelLink(DMA_Type *base, uint32_t channel, *edma_channel_link_type_t* linkType, 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.
- linkType – 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.

void EDMA_SetMajorOffsetConfig(DMA_Type *base, uint32_t channel, int32_t sourceOffset, int32_t destOffset)

Configures the eDMA channel TCD major offset feature.

Adjustment value added to the source address at the completion of the major iteration count

Parameters

- base – eDMA peripheral base address.
- channel – edma channel number.
- sourceOffset – source address offset will be applied to source address after major loop done.
- destOffset – destination address offset will be applied to source address after major loop done.

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 TCD 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_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* linkType, 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.
- linkType – 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.

void EDMA_TcdSetMajorOffsetConfig(*edma_tcd_t* *tcd, int32_t sourceOffset, int32_t destOffset)

Configures the eDMA TCD major offset feature.

Adjustment value added to the source address at the completion of the major iteration count

Parameters

- tcd – A point to the TCD structure.
- sourceOffset – source address offset wiil be applied to source address after major loop done.
- destOffset – destination address offset will be applied to source address after major loop done.

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. This function shall only be used while users need to use scatter gather mode. Scatter gather mode enables EDMA to load a new transfer control block (tcd) in hardware, and automatically reconfigure that DMA channel for a new transfer. Users need to prepare tcd memory and also configure tcds using interface EDMA_SubmitTransfer.

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. This function will be called every time one tcd finished transfer.

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_t.
- srcAddr – eDMA transfer source address.
- srcWidth – eDMA transfer source address width(bytes).

- srcOffset – source address offset.
- destAddr – eDMA transfer destination address.
- destWidth – eDMA transfer destination address width(bytes).
- destOffset – 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_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. In scatter gather mode, call this function will add a configured tcd to the circular list of tcd pool. The tcd pools is setup by call function EDMA_InstallTCDMemory before.

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.

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.

Note: For the case using TCD queue, when the major iteration count is exhausted, additional operations are performed. These include the final address adjustments and reloading of the BITER field into the CITER. Assertion of an optional interrupt request also occurs at this time, as does a possible fetch of a new TCD from memory using the scatter/gather address pointer included in the descriptor (if scatter/gather is enabled).

For instance, when the time interrupt of TCD[0] happens, the TCD[1] has already been loaded into the eDMA engine. As sga and sga_index are calculated based on the DLAST_SGA

bitfield lies in the TCD_CSR register, the sga_index in this case should be 2 (DLAST_SGA of TCD[1] stores the address of TCD[2]). Thus, the "tcdUsed" updated should be (tcdUsed - 2U) which indicates the number of TCDs can be loaded in the memory pool (because TCD[0] and TCD[1] have been loaded into the eDMA engine at this point already.).

For the last two continuous ISRs in a scatter/gather process, they both load the last TCD (The last ISR does not load a new TCD) from the memory pool to the eDMA engine when major loop completes. Therefore, ensure that the header and tcdUsed updated are identical for them. tcdUsed are both 0 in this case as no TCD to be loaded.

See the "eDMA basic data flow" in the eDMA Functional description section of the Reference Manual for further details.

Parameters

- handle – eDMA handle pointer.

void EDMA_DriverIRQHandler(uint32_t instance, uint32_t channel)

EDMA IRQ handler for a given instance and channel.

This is a common parameterized IRQ handler entry that dispatches to the appropriate channel handle based on the instance and channel number.

Parameters

- instance – EDMA instance number.
- channel – EDMA channel number.

FSL_EDMA_DRIVER_VERSION

eDMA driver version

Version 2.4.9.

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

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

_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_error_status_flags eDMA channel 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_ErrorChannelFlag
Error channel number of the cancelled channel number

enumerator kEDMA_ChannelPriorityErrorFlag
Channel priority is not unique.

enumerator kEDMA_TransferCanceledFlag
Transfer cancelled

enumerator kEDMA_ValidFlag
No error occurred, this bit is 0. Otherwise, it is 1.

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_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.

This callback function is called in the EDMA interrupt handle. In normal mode, run into callback function means the transfer users need is done. In scatter gather mode, run into callback function means a transfer control block (tcd) is finished. Not all transfer finished, users can get the finished tcd numbers using interface EDMA_GetUnusedTCDNumber.

Param handle
EDMA handle pointer, users shall not touch the values inside.

Param userData
The callback user parameter pointer. Users can use this parameter to involve things users need to change in EDMA callback function.

Param transferDone
If the current loaded transfer done. In normal mode it means if all transfer done. In scatter gather mode, this parameter shows is the current transfer block in EDMA register is done. As the load of core is different, it will be different if the new tcd loaded into EDMA registers while this callback called. If true, it always means new tcd still not loaded into registers, while false means new tcd already loaded into registers.

Param tcds
How many tcds are done from the last callback. This parameter only used in scatter gather mode. It tells user how many tcds are finished between the last callback and this.

typedef struct *_edma_handle* edma_handle_t

eDMA transfer handle structure

DMA_DCHPRI_INDEX(channel)

Compute the offset unit from DCHPRI3.

struct __edma_config

#include <fsl_edma.h> eDMA global configuration structure.

Public Members

bool enableContinuousLinkMode

Enable (true) continuous link mode. Upon minor loop completion, the channel activates again if that channel has a minor loop channel link enabled and the link channel is itself.

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 tcd 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. Should point to the next TCD to be loaded into the eDMA engine.

volatile int8_t tail
The last TCD index. Should point to the next TCD to be stored into the memory pool.

volatile int8_t tcdUsed
The number of used TCD slots. Should reflect the number of TCDs can be used/loaded in the memory.

volatile int8_t tcdSize
: The total number of TCD slots in the queue.

uint8_t flags
: The status of the current channel.

2.16 ENC: Quadrature Encoder/Decoder

void ENC_Init(ENC_Type *base, const *enc_config_t* *config)
: Initialization for the ENC module.

  This function is to make the initialization for the ENC module. It should be called firstly before any operation to the ENC with the operations like:

  - Enable the clock for ENC module.
  - Configure the ENC's working attributes.

  Parameters

  - base – ENC peripheral base address.
  - config – Pointer to configuration structure. See to "enc_config_t".

void ENC_Deinit(ENC_Type *base)
: De-initialization for the ENC module.

  This function is to make the de-initialization for the ENC module. It could be called when ENC is no longer used with the operations like:

  - Disable the clock for ENC module.

  Parameters

  - base – ENC peripheral base address.

void ENC_GetDefaultConfig(*enc_config_t* *config)
: Get an available pre-defined settings for ENC's configuration.

  This function initializes the ENC configuration structure with an available settings, the default value are:

```
config->enableReverseDirection              = false;
config->decoderWorkMode                     = kENC_DecoderWorkAsNormalMode;
config->HOMETriggerMode                     = kENC_HOMETriggerDisabled;
config->INDEXTriggerMode                    = kENC_INDEXTriggerDisabled;
config->enableTRIGGERClearPositionCounter   = false;
config->enableTRIGGERClearHoldPositionCounter = false;
config->enableWatchdog                      = false;
config->watchdogTimeoutValue                = 0U;
config->filterCount                         = 0U;
config->filterSamplePeriod                  = 0U;
config->positionMatchMode                   = kENC_
↪POSMATCHOnPositionCounterEqualToComapreValue;
config->positionCompareValue                = 0xFFFFFFFFU;
config->revolutionCountCondition            = kENC_RevolutionCountOnINDEXPulse;
config->enableModuloCountMode               = false;
config->positionModulusValue                = 0U;
config->positionInitialValue                = 0U;
```

(continues on next page)

(continued from previous page)

```
config->prescalerValue                    = kENC_ClockDiv1;
config->enablePeriodMeasurementFunction   = true;
```

Parameters

- config – Pointer to a variable of configuration structure. See to "enc_config_t".

void ENC_DoSoftwareLoadInitialPositionValue(ENC_Type *base)

Load the initial position value to position counter.

This function is to transfer the initial position value (UINIT and LINIT) contents to position counter (UPOS and LPOS), so that to provide the consistent operation the position counter registers.

Parameters

- base – ENC peripheral base address.

void ENC_SetSelfTestConfig(ENC_Type *base, const *enc_self_test_config_t* *config)

Enable and configure the self test function.

This function is to enable and configuration the self test function. It controls and sets the frequency of a quadrature signal generator. It provides a quadrature test signal to the inputs of the quadrature decoder module. It is a factory test feature; however, it may be useful to customers' software development and testing.

Parameters

- base – ENC peripheral base address.
- config – Pointer to configuration structure. See to "enc_self_test_config_t". Pass "NULL" to disable.

void ENC_EnableWatchdog(ENC_Type *base, bool enable)

Enable watchdog for ENC module.

Parameters

- base – ENC peripheral base address
- enable – Enables or disables the watchdog

void ENC_SetInitialPositionValue(ENC_Type *base, uint32_t value)

Set initial position value for ENC module.

Parameters

- base – ENC peripheral base address
- value – Positive initial value

uint32_t ENC_GetStatusFlags(ENC_Type *base)

Get the status flags.

Parameters

- base – ENC peripheral base address.

Returns

Mask value of status flags. For available mask, see to "_enc_status_flags".

void ENC_ClearStatusFlags(ENC_Type *base, uint32_t mask)

Clear the status flags.

Parameters

- base – ENC peripheral base address.
- mask – Mask value of status flags to be cleared. For available mask, see to "_enc_status_flags".

static inline uint16_t ENC_GetSignalStatusFlags(ENC_Type *base)

Get the signals' real-time status.

Parameters

- base – ENC peripheral base address.

Returns

Mask value of signals' real-time status. For available mask, see to "_enc_signal_status_flags"

void ENC_EnableInterrupts(ENC_Type *base, uint32_t mask)

Enable the interrupts.

Parameters

- base – ENC peripheral base address.
- mask – Mask value of interrupts to be enabled. For available mask, see to "_enc_interrupt_enable".

void ENC_DisableInterrupts(ENC_Type *base, uint32_t mask)

Disable the interrupts.

Parameters

- base – ENC peripheral base address.
- mask – Mask value of interrupts to be disabled. For available mask, see to "_enc_interrupt_enable".

uint32_t ENC_GetEnabledInterrupts(ENC_Type *base)

Get the enabled interrupts' flags.

Parameters

- base – ENC peripheral base address.

Returns

Mask value of enabled interrupts.

uint32_t ENC_GetPositionValue(ENC_Type *base)

Get the current position counter's value.

Parameters

- base – ENC peripheral base address.

Returns

Current position counter's value.

uint32_t ENC_GetHoldPositionValue(ENC_Type *base)

Get the hold position counter's value.

When any of the counter registers is read, the contents of each counter register is written to the corresponding hold register. Taking a snapshot of the counters' values provides a consistent view of a system position and a velocity to be attained.

Parameters

- base – ENC peripheral base address.

Returns

Hold position counter's value.

static inline uint16_t ENC_GetPositionDifferenceValue(ENC_Type *base)

Get the position difference counter's value.

Parameters

- base – ENC peripheral base address.

Returns

The position difference counter's value.

static inline uint16_t ENC_GetHoldPositionDifferenceValue(ENC_Type *base)

Get the hold position difference counter's value.

When any of the counter registers is read, the contents of each counter register is written to the corresponding hold register. Taking a snapshot of the counters' values provides a consistent view of a system position and a velocity to be attained.

Parameters

- base – ENC peripheral base address.

Returns

Hold position difference counter's value.

static inline uint16_t ENC_GetRevolutionValue(ENC_Type *base)

Get the position revolution counter's value.

Parameters

- base – ENC peripheral base address.

Returns

The position revolution counter's value.

static inline uint16_t ENC_GetHoldRevolutionValue(ENC_Type *base)

Get the hold position revolution counter's value.

When any of the counter registers is read, the contents of each counter register is written to the corresponding hold register. Taking a snapshot of the counters' values provides a consistent view of a system position and a velocity to be attained.

Parameters

- base – ENC peripheral base address.

Returns

Hold position revolution counter's value.

static inline uint16_t ENC_GetLastEdgeTimeValue(ENC_Type *base)

Get the last edge time value.

Parameters

- base – ENC peripheral base address.

Returns

The last edge time hold value.

static inline uint16_t ENC_GetHoldLastEdgeTimeValue(ENC_Type *base)

Get the last edge time hold value.

Parameters

- base – ENC peripheral base address.

Returns

The last edge time hold value.

static inline uint16_t ENC_GetPositionDifferencePeriodValue(ENC_Type *base)

Get the position difference period value.

Parameters

- base – ENC peripheral base address.

Returns

The position difference period hold value.

static inline uint16_t ENC_GetPositionDifferencePeriodBufferValue(ENC_Type *base)

Get the position difference period buffer value.

Parameters

- base – ENC peripheral base address.

Returns

The position difference period hold value.

static inline uint16_t ENC_GetHoldPositionDifferencePeriodValue(ENC_Type *base)

Get the position difference period hold value.

Parameters

- base – ENC peripheral base address.

Returns

The position difference period hold value.

enum _enc_interrupt_enable

Interrupt enable/disable mask.

Values:

enumerator kENC_HOMETransitionInterruptEnable

HOME interrupt enable.

enumerator kENC_INDEXPulseInterruptEnable

INDEX pulse interrupt enable.

enumerator kENC_WatchdogTimeoutInterruptEnable

Watchdog timeout interrupt enable.

enumerator kENC_PositionCompareInerruptEnable

Position compare interrupt enable.

enumerator kENC_PositionRollOverInterruptEnable

Roll-over interrupt enable.

enumerator kENC_PositionRollUnderInterruptEnable

Roll-under interrupt enable.

enum _enc_status_flags

Status flag mask.

These flags indicate the counter's events.

Values:

enumerator kENC_HOMETransitionFlag
HOME signal transition interrupt request.

enumerator kENC_INDEXPulseFlag
INDEX Pulse Interrupt Request.

enumerator kENC_WatchdogTimeoutFlag
Watchdog timeout interrupt request.

enumerator kENC_PositionCompareFlag
Position compare interrupt request.

enumerator kENC_PositionRollOverFlag
Roll-over interrupt request.

enumerator kENC_PositionRollUnderFlag
Roll-under interrupt request.

enumerator kENC_LastCountDirectionFlag
Last count was in the up direction, or the down direction.

enum _enc_signal_status_flags
Signal status flag mask.
These flags indicate the counter's signal.
Values:

enumerator kENC_RawHOMEStatusFlag
Raw HOME input.

enumerator kENC_RawINDEXStatusFlag
Raw INDEX input.

enumerator kENC_RawPHBStatusFlag
Raw PHASEB input.

enumerator kENC_RawPHAEXStatusFlag
Raw PHASEA input.

enumerator kENC_FilteredHOMEStatusFlag
The filtered version of HOME input.

enumerator kENC_FilteredINDEXStatusFlag
The filtered version of INDEX input.

enumerator kENC_FilteredPHBStatusFlag
The filtered version of PHASEB input.

enumerator kENC_FilteredPHAStatusFlag
The filtered version of PHASEA input.

enum _enc_home_trigger_mode
Define HOME signal's trigger mode.

The ENC would count the trigger from HOME signal line.

Values:

enumerator kENC_HOMETriggerDisabled
HOME signal's trigger is disabled.

enumerator kENC_HOMETriggerOnRisingEdge
Use positive going edge-to-trigger initialization of position counters.

enumerator kENC_HOMETriggerOnFallingEdge
Use negative going edge-to-trigger initialization of position counters.

enum _enc_index_trigger_mode
Define INDEX signal's trigger mode.

The ENC would count the trigger from INDEX signal line.

Values:

enumerator kENC_INDEXTriggerDisabled
INDEX signal's trigger is disabled.

enumerator kENC_INDEXTriggerOnRisingEdge
Use positive going edge-to-trigger initialization of position counters.

enumerator kENC_INDEXTriggerOnFallingEdge
Use negative going edge-to-trigger initialization of position counters.

enum _enc_decoder_work_mode
Define type for decoder work mode.

The normal work mode uses the standard quadrature decoder with PHASEA and PHASEB. When in signal phase count mode, a positive transition of the PHASEA input generates a count signal while the PHASEB input and the reverse direction control the counter direction. If the reverse direction is not enabled, PHASEB = 0 means counting up and PHASEB = 1 means counting down. Otherwise, the direction is reversed.

Values:

enumerator kENC_DecoderWorkAsNormalMode
Use standard quadrature decoder with PHASEA and PHASEB.

enumerator kENC_DecoderWorkAsSignalPhaseCountMode
PHASEA input generates a count signal while PHASEB input control the direction.

enum _enc_position_match_mode

Define type for the condition of POSMATCH pulses.

Values:

enumerator kENC_POSMATCHOnPositionCounterEqualToComapreValue

POSMATCH pulses when a match occurs between the position counters (POS) and the compare value (COMP).

enumerator kENC_POSMATCHOnReadingAnyPositionCounter

POSMATCH pulses when any position counter register is read.

enum _enc_revolution_count_condition

Define type for determining how the revolution counter (REV) is incremented/decremented.

Values:

enumerator kENC_RevolutionCountOnINDEXPulse

Use INDEX pulse to increment/decrement revolution counter.

enumerator kENC_RevolutionCountOnRollOverModulus

Use modulus counting roll-over/under to increment/decrement revolution counter.

enum _enc_self_test_direction

Define type for direction of self test generated signal.

Values:

enumerator kENC_SelfTestDirectionPositive

Self test generates the signal in positive direction.

enumerator kENC_SelfTestDirectionNegative

Self test generates the signal in negative direction.

enum _enc_prescaler

Define prescaler value for clock in CTRL3.

The clock is prescaled by a value of 2^PRSC which means that the prescaler logic can divide the clock by a minimum of 1 and a maximum of 32,768.

Values:

enumerator kENC_ClockDiv1

enumerator kENC_ClockDiv2

enumerator kENC_ClockDiv4

enumerator kENC_ClockDiv8

enumerator kENC_ClockDiv16

enumerator kENC_ClockDiv32

enumerator kENC_ClockDiv64

enumerator kENC_ClockDiv128

enumerator kENC_ClockDiv256

enumerator kENC_ClockDiv512

enumerator kENC_ClockDiv1024

enumerator kENC_ClockDiv2048

enumerator kENC_ClockDiv4096

enumerator kENC_ClockDiv8192

enumerator kENC_ClockDiv16384

enumerator kENC_ClockDiv32768

enum _enc_filter_prescaler

Define input filter prescaler value.

The input filter prescaler value is to prescale the IPBus clock. (Frequency of FILT clock) = (Frequency of IPBus clock) / 2^FILT_PRSC.

Values:

enumerator kENC_FilterPrescalerDiv1

Input filter prescaler is 1.

enumerator kENC_FilterPrescalerDiv2

Input filter prescaler is 2.

enumerator kENC_FilterPrescalerDiv4

Input filter prescaler is 4.

enumerator kENC_FilterPrescalerDiv8

Input filter prescaler is 8.

enumerator kENC_FilterPrescalerDiv16

Input filter prescaler is 16.

enumerator kENC_FilterPrescalerDiv32

Input filter prescaler is 32.

enumerator kENC_FilterPrescalerDiv64

Input filter prescaler is 64.

enumerator kENC_FilterPrescalerDiv128

Input filter prescaler is 128.

typedef enum *_enc_home_trigger_mode* enc_home_trigger_mode_t

Define HOME signal's trigger mode.

The ENC would count the trigger from HOME signal line.

typedef enum *_enc_index_trigger_mode* enc_index_trigger_mode_t

Define INDEX signal's trigger mode.

The ENC would count the trigger from INDEX signal line.

typedef enum *_enc_decoder_work_mode* enc_decoder_work_mode_t

Define type for decoder work mode.

The normal work mode uses the standard quadrature decoder with PHASEA and PHASEB. When in signal phase count mode, a positive transition of the PHASEA input generates a count signal while the PHASEB input and the reverse direction control the counter direction. If the reverse direction is not enabled, PHASEB = 0 means counting up and PHASEB = 1 means counting down. Otherwise, the direction is reversed.

typedef enum *_enc_position_match_mode* enc_position_match_mode_t

Define type for the condition of POSMATCH pulses.

typedef enum *_enc_revolution_count_condition* enc_revolution_count_condition_t

Define type for determining how the revolution counter (REV) is incremented/decremented.

typedef enum *_enc_self_test_direction* enc_self_test_direction_t

Define type for direction of self test generated signal.

typedef enum *_enc_prescaler* enc_prescaler_t

Define prescaler value for clock in CTRL3.

The clock is prescaled by a value of 2^PRSC which means that the prescaler logic can divide the clock by a minimum of 1 and a maximum of 32,768.

typedef enum *_enc_filter_prescaler* enc_filter_prescaler_t

Define input filter prescaler value.

The input filter prescaler value is to prescale the IPBus clock. (Frequency of FILT clock) = (Frequency of IPBus clock) / 2^FILT_PRSC.

typedef struct *_enc_config* enc_config_t

Define user configuration structure for ENC module.

typedef struct *_enc_self_test_config* enc_self_test_config_t

Define configuration structure for self test module.

The self test module provides a quadrature test signal to the inputs of the quadrature decoder module. This is a factory test feature. It is also useful to customers' software development and testing.

FSL_ENC_DRIVER_VERSION

struct _enc_config

#include <fsl_enc.h> Define user configuration structure for ENC module.

Public Members

bool enableReverseDirection

Enable reverse direction counting.

enc_decoder_work_mode_t decoderWorkMode

Enable signal phase count mode.

enc_home_trigger_mode_t HOMETriggerMode

Enable HOME to initialize position counters.

enc_index_trigger_mode_t INDEXTriggerMode

Enable INDEX to initialize position counters.

bool enableTRIGGERClearPositionCounter

Clear POSD, REV, UPOS and LPOS on rising edge of TRIGGER, or not.

bool enableTRIGGERClearHoldPositionCounter

Enable update of hold registers on rising edge of TRIGGER, or not.

bool enableWatchdog

Enable the watchdog to detect if the target is moving or not.

uint16_t watchdogTimeoutValue

Watchdog timeout count value. It stores the timeout count for the quadrature decoder module watchdog timer. This field is only available when "enableWatchdog" = true. The available value is a 16-bit unsigned number.

enc_filter_prescaler_t filterPrescaler

Input filter prescaler.

uint16_t filterCount

Input Filter Sample Count. This value should be chosen to reduce the probability of noisy samples causing an incorrect transition to be recognized. The value represent the number of consecutive samples that must agree prior to the input filter accepting an input transition. A value of 0x0 represents 3 samples. A value of 0x7 represents 10 samples. The Available range is 0 - 7.

uint16_t filterSamplePeriod

Input Filter Sample Period. This value should be set such that the sampling period is larger than the period of the expected noise. This value represents the sampling period (in IPBus clock cycles) of the decoder input signals. The available range is 0 - 255.

enc_position_match_mode_t positionMatchMode

The condition of POSMATCH pulses.

uint32_t positionCompareValue

Position compare value. The available value is a 32-bit number.

enc_revolution_count_condition_t revolutionCountCondition

Revolution Counter Modulus Enable.

bool enableModuloCountMode

Enable Modulo Counting.

uint32_t positionModulusValue

Position modulus value. This value would be available only when "enableModuloCountMode" = true. The available value is a 32-bit number.

uint32_t positionInitialValue

Position initial value. The available value is a 32-bit number.

bool enablePeriodMeasurementFunction

Enable period measurement function.

enc_prescaler_t prescalerValue

The value of prescaler.

struct __enc__self__test__config

#include <fsl_enc.h> Define configuration structure for self test module.

The self test module provides a quadrature test signal to the inputs of the quadrature decoder module. This is a factory test feature. It is also useful to customers' software development and testing.

Public Members

enc_self_test_direction_t signalDirection

Direction of self test generated signal.

uint16_t signalCount

Hold the number of quadrature advances to generate. The available range is 0 - 255.

uint16_t signalPeriod

Hold the period of quadrature phase in IPBus clock cycles. The available range is 0 - 31.

2.17 ENET: Ethernet MAC Driver

void ENET_GetDefaultConfig(*enet_config_t* *config)

Gets the ENET default configuration structure.

The purpose of this API is to get the default ENET MAC controller configure structure for *ENET_Init()*. User may use the initialized structure unchanged in *ENET_Init()*, or modify some fields of the structure before calling *ENET_Init()*. Example:

```
enet_config_t config;
ENET_GetDefaultConfig(&config);
```

Parameters

- config – The ENET mac controller configuration structure pointer.

status_t ENET_Up(ENET_Type *base, *enet_handle_t* *handle, const *enet_config_t* *config, const *enet_buffer_config_t* *bufferConfig, uint8_t *macAddr, uint32_t srcClock_Hz)

Initializes the ENET module.

This function initializes the module with the ENET configuration.

Note: ENET has two buffer descriptors legacy buffer descriptors and enhanced IEEE 1588 buffer descriptors. The legacy descriptor is used by default. To use the IEEE 1588 feature, use the enhanced IEEE 1588 buffer descriptor by defining "ENET_ENHANCEDBUFFERDESCRIPTOR_MODE" and calling ENET_Ptp1588Configure() to configure the 1588 feature and related buffers after calling *ENET_Up()*.

Parameters

- base – ENET peripheral base address.
- handle – ENET handler pointer.
- config – ENET mac configuration structure pointer. The "enet_config_t" type mac configuration return from ENET_GetDefaultConfig can be used directly. It is also possible to verify the Mac configuration using other methods.
- bufferConfig – ENET buffer configuration structure pointer. The buffer configuration should be prepared for ENET Initialization. It is the start address of "ringNum" enet_buffer_config structures. To support added multi-ring features in some soc and compatible with the previous enet driver version. For single ring supported, this bufferConfig is a buffer configure structure pointer, for multi-ring supported and used case, this bufferConfig pointer should be a buffer configure structure array pointer.
- macAddr – ENET mac address of Ethernet device. This MAC address should be provided.
- srcClock_Hz – The internal module clock source for MII clock.

Return values

- kStatus_Success – Succeed to initialize the ethernet driver.
- kStatus_ENET_InitMemoryFail – Init fails since buffer memory is not enough.

status_t ENET_Init(ENET_Type *base, *enet_handle_t* *handle, const *enet_config_t* *config, const *enet_buffer_config_t* *bufferConfig, uint8_t *macAddr, uint32_t srcClock_Hz)

Initializes the ENET module.

This function ungates the module clock and initializes it with the ENET configuration.

Note: ENET has two buffer descriptors legacy buffer descriptors and enhanced IEEE 1588 buffer descriptors. The legacy descriptor is used by default. To use the IEEE 1588 feature, use the enhanced IEEE 1588 buffer descriptor by defining "ENET_ENHANCEDBUFFERDESCRIPTOR_MODE" and calling ENET_Ptp1588Configure() to configure the 1588 feature and related buffers after calling *ENET_Init()*.

Parameters

- base – ENET peripheral base address.
- handle – ENET handler pointer.
- config – ENET mac configuration structure pointer. The "enet_config_t" type mac configuration return from ENET_GetDefaultConfig can be used directly. It is also possible to verify the Mac configuration using other methods.
- bufferConfig – ENET buffer configuration structure pointer. The buffer configuration should be prepared for ENET Initialization. It is the start address of "ringNum" enet_buffer_config structures. To support added multi-ring features in some soc and compatible with the previous enet driver version. For single ring supported, this bufferConfig is a buffer configure structure pointer, for multi-ring supported and used case, this bufferConfig pointer should be a buffer configure structure array pointer.
- macAddr – ENET mac address of Ethernet device. This MAC address should be provided.
- srcClock_Hz – The internal module clock source for MII clock.

Return values

- kStatus_Success – Succeed to initialize the ethernet driver.
- kStatus_ENET_InitMemoryFail – Init fails since buffer memory is not enough.

void ENET_Down(ENET_Type *base)

Stops the ENET module.

This function disables the ENET module.

Parameters

- base – ENET peripheral base address.

void ENET_Deinit(ENET_Type *base)

Deinitializes the ENET module.

This function gates the module clock, clears ENET interrupts, and disables the ENET module.

Parameters

- base – ENET peripheral base address.

static inline void ENET_Reset(ENET_Type *base)

Resets the ENET module.

This function restores the ENET module to reset state. Note that this function sets all registers to reset state. As a result, the ENET module can't work after calling this function.

Parameters

- base – ENET peripheral base address.

void ENET_SetMII(ENET_Type *base, *enet_mii_speed_t* speed, *enet_mii_duplex_t* duplex)

Sets the ENET MII speed and duplex.

This API is provided to dynamically change the speed and dulpex for MAC.

Parameters

- base – ENET peripheral base address.
- speed – The speed of the RMII mode.
- duplex – The duplex of the RMII mode.

void ENET_SetSMI(ENET_Type *base, uint32_t srcClock_Hz, bool isPreambleDisabled)

Sets the ENET SMI(serial management interface)- MII management interface.

Parameters

- base – ENET peripheral base address.
- srcClock_Hz – This is the ENET module clock frequency. See clock distribution.
- isPreambleDisabled – The preamble disable flag.
  - true Enables the preamble.
  - false Disables the preamble.

static inline bool ENET_GetSMI(ENET_Type *base)

Gets the ENET SMI- MII management interface configuration.

This API is used to get the SMI configuration to check whether the MII management interface has been set.

Parameters

- base – ENET peripheral base address.

Returns

The SMI setup status true or false.

static inline uint32_t ENET_ReadSMIData(ENET_Type *base)

Reads data from the PHY register through an SMI interface.

Parameters

- base – ENET peripheral base address.

Returns

The data read from PHY

static inline void ENET_StartSMIWrite(ENET_Type *base, uint8_t phyAddr, uint8_t regAddr, *enet_mii_write_t* operation, uint16_t data)

Sends the MDIO IEEE802.3 Clause 22 format write command.

After calling this function, need to check whether the transmission is over then do next MDIO operation. For ease of use, encapsulated *ENET_MDIOWrite()* can be called. For customized requirements, implement with combining separated APIs.

Parameters

- base – ENET peripheral base address.
- phyAddr – The PHY address. Range from 0 ~ 31.
- regAddr – The PHY register address. Range from 0 ~ 31.
- operation – The write operation.
- data – The data written to PHY.

static inline void ENET_StartSMIRead(ENET_Type *base, uint8_t phyAddr, uint8_t regAddr, *enet_mii_read_t* operation)

Sends the MDIO IEEE802.3 Clause 22 format read command.

After calling this function, need to check whether the transmission is over then do next MDIO operation. For ease of use, encapsulated *ENET_MDIORead()* can be called. For customized requirements, implement with combining separated APIs.

Parameters

- base – ENET peripheral base address.
- phyAddr – The PHY address. Range from 0 ~ 31.
- regAddr – The PHY register address. Range from 0 ~ 31.
- operation – The read operation.

status_t ENET_MDIOWrite(ENET_Type *base, uint8_t phyAddr, uint8_t regAddr, uint16_t data)

MDIO write with IEEE802.3 Clause 22 format.

Parameters

- base – ENET peripheral base address.
- phyAddr – The PHY address. Range from 0 ~ 31.
- regAddr – The PHY register. Range from 0 ~ 31.
- data – The data written to PHY.

Returns

kStatus_Success MDIO access succeeds.

Returns

kStatus_Timeout MDIO access timeout.

status_t ENET_MDIORead(ENET_Type *base, uint8_t phyAddr, uint8_t regAddr, uint16_t *pData)

MDIO read with IEEE802.3 Clause 22 format.

Parameters

- base – ENET peripheral base address.
- phyAddr – The PHY address. Range from 0 ~ 31.
- regAddr – The PHY register. Range from 0 ~ 31.
- pData – The data read from PHY.

Returns

kStatus_Success MDIO access succeeds.

Returns

kStatus_Timeout MDIO access timeout.

static inline void ENET_StartExtC45SMIWriteReg(ENET_Type *base, uint8_t portAddr, uint8_t devAddr, uint16_t regAddr)

Sends the MDIO IEEE802.3 Clause 45 format write register command.

After calling this function, need to check whether the transmission is over then do next MDIO operation. For ease of use, encapsulated *ENET_MDIOC45Write()*/ENET_MDIOC45Read() can be called. For customized requirements, implement with combining separated APIs.

Parameters

- base – ENET peripheral base address.
- portAddr – The MDIO port address(PHY address).
- devAddr – The device address.
- regAddr – The PHY register address.

static inline void ENET_StartExtC45SMIWriteData(ENET_Type *base, uint8_t portAddr, uint8_t devAddr, uint16_t data)

Sends the MDIO IEEE802.3 Clause 45 format write data command.

After calling this function, need to check whether the transmission is over then do next MDIO operation. For ease of use, encapsulated *ENET_MDIOC45Write()* can be called. For customized requirements, implement with combining separated APIs.

Parameters

- base – ENET peripheral base address.
- portAddr – The MDIO port address(PHY address).
- devAddr – The device address.
- data – The data written to PHY.

static inline void ENET_StartExtC45SMIReadData(ENET_Type *base, uint8_t portAddr, uint8_t devAddr)

Sends the MDIO IEEE802.3 Clause 45 format read data command.

After calling this function, need to check whether the transmission is over then do next MDIO operation. For ease of use, encapsulated *ENET_MDIOC45Read()* can be called. For customized requirements, implement with combining separated APIs.

Parameters

- base – ENET peripheral base address.
- portAddr – The MDIO port address(PHY address).
- devAddr – The device address.

status_t ENET_MDIOC45Write(ENET_Type *base, uint8_t portAddr, uint8_t devAddr, uint16_t regAddr, uint16_t data)

MDIO write with IEEE802.3 Clause 45 format.

Parameters

- base – ENET peripheral base address.
- portAddr – The MDIO port address(PHY address).
- devAddr – The device address.
- regAddr – The PHY register address.
- data – The data written to PHY.

Returns
kStatus_Success MDIO access succeeds.

Returns
kStatus_Timeout MDIO access timeout.

status_t ENET_MDIOC45Read(ENET_Type *base, uint8_t portAddr, uint8_t devAddr, uint16_t regAddr, uint16_t *pData)

MDIO read with IEEE802.3 Clause 45 format.

Parameters

- base – ENET peripheral base address.
- portAddr – The MDIO port address(PHY address).
- devAddr – The device address.
- regAddr – The PHY register address.
- pData – The data read from PHY.

Returns
kStatus_Success MDIO access succeeds.

Returns
kStatus_Timeout MDIO access timeout.

static inline void ENET_SetRGMIIClockDelay(ENET_Type *base, bool txEnabled, bool rxEnabled)

Control the usage of the delayed tx/rx RGMII clock.

Parameters

- base – ENET peripheral base address.
- txEnabled – Enable or disable to generate the delayed version of RGMII_TXC.
- rxEnabled – Enable or disable to use the delayed version of RGMII_RXC.

void ENET_SetMacAddr(ENET_Type *base, uint8_t *macAddr)

Sets the ENET module Mac address.

Parameters

- base – ENET peripheral base address.
- macAddr – The six-byte Mac address pointer. The pointer is allocated by application and input into the API.

void ENET_GetMacAddr(ENET_Type *base, uint8_t *macAddr)

Gets the ENET module Mac address.

Parameters

- base – ENET peripheral base address.
- macAddr – The six-byte Mac address pointer. The pointer is allocated by application and input into the API.

void ENET_AddMulticastGroup(ENET_Type *base, uint8_t *address)

Adds the ENET device to a multicast group.

Parameters

- base – ENET peripheral base address.
- address – The six-byte multicast group address which is provided by application.

void ENET_LeaveMulticastGroup(ENET_Type *base, uint8_t *address)

Moves the ENET device from a multicast group.

Parameters

- base – ENET peripheral base address.
- address – The six-byte multicast group address which is provided by application.

static inline void ENET_ActiveRead(ENET_Type *base)

Activates frame reception for multiple rings.

This function is to active the enet read process.

Note: This must be called after the MAC configuration and state are ready. It must be called after the *ENET_Init()*. This should be called when the frame reception is required.

Parameters

- base – ENET peripheral base address.

static inline void ENET_EnableSleepMode(ENET_Type *base, bool enable)

Enables/disables the MAC to enter sleep mode. This function is used to set the MAC enter sleep mode. When entering sleep mode, the magic frame wakeup interrupt should be enabled to wake up MAC from the sleep mode and reset it to normal mode.

Parameters

- base – ENET peripheral base address.
- enable – True enable sleep mode, false disable sleep mode.

static inline void ENET_GetAccelFunction(ENET_Type *base, uint32_t *txAccelOption, uint32_t *rxAccelOption)

Gets ENET transmit and receive accelerator functions from MAC controller.

Parameters

- base – ENET peripheral base address.
- txAccelOption – The transmit accelerator option. The "enet_tx_accelerator_t" is recommended to be used to as the mask to get the exact the accelerator option.
- rxAccelOption – The receive accelerator option. The "enet_rx_accelerator_t" is recommended to be used to as the mask to get the exact the accelerator option.

static inline void ENET_EnableInterrupts(ENET_Type *base, uint32_t mask)

Enables the ENET interrupt.

This function enables the ENET interrupt according to the provided mask. The mask is a logical OR of enumeration members. See *enet_interrupt_enable_t*. For example, to enable the TX frame interrupt and RX frame interrupt, do the following.

```
ENET_EnableInterrupts(ENET, kENET_TxFrameInterrupt | kENET_RxFrameInterrupt);
```

Parameters

- base – ENET peripheral base address.
- mask – ENET interrupts to enable. This is a logical OR of the enumeration *enet_interrupt_enable_t*.

static inline void ENET_DisableInterrupts(ENET_Type *base, uint32_t mask)

Disables the ENET interrupt.

This function disables the ENET interrupts according to the provided mask. The mask is a logical OR of enumeration members. See *enet_interrupt_enable_t*. For example, to disable the TX frame interrupt and RX frame interrupt, do the following.

```
ENET_DisableInterrupts(ENET, kENET_TxFrameInterrupt | kENET_RxFrameInterrupt);
```

Parameters

- base – ENET peripheral base address.
- mask – ENET interrupts to disable. This is a logical OR of the enumeration *enet_interrupt_enable_t*.

static inline uint32_t ENET_GetInterruptStatus(ENET_Type *base)

Gets the ENET interrupt status flag.

Parameters

- base – ENET peripheral base address.

Returns

The event status of the interrupt source. This is the logical OR of members of the enumeration *enet_interrupt_enable_t*.

static inline void ENET_ClearInterruptStatus(ENET_Type *base, uint32_t mask)

Clears the ENET interrupt events status flag.

This function clears enabled ENET interrupts according to the provided mask. The mask is a logical OR of enumeration members. See the *enet_interrupt_enable_t*. For example, to clear the TX frame interrupt and RX frame interrupt, do the following.

```
ENET_ClearInterruptStatus(ENET, kENET_TxFrameInterrupt | kENET_RxFrameInterrupt);
```

Parameters

- base – ENET peripheral base address.
- mask – ENET interrupt source to be cleared. This is the logical OR of members of the enumeration *enet_interrupt_enable_t*.

void ENET_SetRxISRHandler(ENET_Type *base, *enet_isr_t* ISRHandler)

Set the second level Rx IRQ handler.

Parameters

- base – ENET peripheral base address.
- ISRHandler – The handler to install.

void ENET_SetTxISRHandler(ENET_Type *base, *enet_isr_t* ISRHandler)

Set the second level Tx IRQ handler.

Parameters

- base – ENET peripheral base address.
- ISRHandler – The handler to install.

void ENET_SetErrISRHandler(ENET_Type *base, *enet_isr_t* ISRHandler)

Set the second level Err IRQ handler.

Parameters

- base – ENET peripheral base address.
- ISRHandler – The handler to install.

void ENET_GetRxErrBeforeReadFrame(*enet_handle_t* *handle, *enet_data_error_stats_t* *eErrorStatic, uint8_t ringId)

Gets the error statistics of a received frame for ENET specified ring.

This API must be called after the ENET_GetRxFrameSize and before the *ENET_ReadFrame()*. If the ENET_GetRxFrameSize returns kStatus_ENET_RxFrameError, the ENET_GetRxErrBeforeReadFrame can be used to get the exact error statistics. This is an example.

```
status = ENET_GetRxFrameSize(&g_handle, &length, 0);
if (status == kStatus_ENET_RxFrameError)
{
    Comments: Get the error information of the received frame.
    ENET_GetRxErrBeforeReadFrame(&g_handle, &eErrStatic, 0);
    Comments: update the receive buffer.
    ENET_ReadFrame(EXAMPLE_ENET, &g_handle, NULL, 0);
}
```

Parameters

- handle – The ENET handler structure pointer. This is the same handler pointer used in the ENET_Init.
- eErrorStatic – The error statistics structure pointer.
- ringId – The ring index, range from 0 ~ (FSL_FEATURE_ENET_INSTANCE_QUEUEn(x) - 1).

void ENET_EnableStatistics(ENET_Type *base, bool enable)

Enables/disables collection of transfer statistics.

Note that this function does not reset any of the already collected data, use the function ENET_ResetStatistics to clear the transfer statistics if needed.

Parameters

- base – ENET peripheral base address.
- enable – True enable statistics collection, false disable statistics collection.

void ENET_GetStatistics(ENET_Type *base, *enet_transfer_stats_t* *statistics)

Gets transfer statistics.

Copies the actual value of hardware counters into the provided structure. Calling this function does not reset the counters in hardware.

Parameters

- base – ENET peripheral base address.
- statistics – The statistics structure pointer.

void ENET_ResetStatistics(ENET_Type *base)

Resets transfer statistics.

Sets the value of hardware transfer counters to zero.

Parameters

- base – ENET peripheral base address.

status_t ENET_GetRxFrameSize(*enet_handle_t* *handle, uint32_t *length, uint8_t ringId)

Gets the size of the read frame for specified ring.

This function gets a received frame size from the ENET buffer descriptors.

Note: The FCS of the frame is automatically removed by MAC and the size is the length without the FCS. After calling ENET_GetRxFrameSize, *ENET_ReadFrame()* should be called to receive frame and update the BD if the result is not "kStatus_ENET_RxFrameEmpty".

Parameters

- handle – The ENET handler structure. This is the same handler pointer used in the ENET_Init.
- length – The length of the valid frame received.
- ringId – The ring index or ring number.

Return values

- kStatus_ENET_RxFrameEmpty – No frame received. Should not call ENET_ReadFrame to read frame.
- kStatus_ENET_RxFrameError – Data error happens. ENET_ReadFrame should be called with NULL data and NULL length to update the receive buffers.
- kStatus_Success – Receive a frame Successfully then the ENET_ReadFrame should be called with the right data buffer and the captured data length input.

status_t ENET_ReadFrame(ENET_Type *base, *enet_handle_t* *handle, uint8_t *data, uint32_t length, uint8_t ringId, uint32_t *ts)

Reads a frame from the ENET device. This function reads a frame (both the data and the length) from the ENET buffer descriptors. User can get timestamp through ts pointer if the ts is not NULL.

Note: It doesn't store the timestamp in the receive timestamp queue. The ENET_GetRxFrameSize should be used to get the size of the prepared data buffer. This API uses memcpy to copy data from DMA buffer to application buffer, 4 bytes aligned data buffer in 32 bits platforms provided by user may let compiler use optimization instruction to reduce time consumption. This is an example:

```
uint32_t length;
enet_handle_t g_handle;
Comments: Get the received frame size firstly.
status = ENET_GetRxFrameSize(&g_handle, &length, 0);
if (length != 0)
{
    Comments: Allocate memory here with the size of "length"
    uint8_t *data = memory allocate interface;
    if (!data)
    {
        ENET_ReadFrame(ENET, &g_handle, NULL, 0, 0, NULL);
        Comments: Add the console warning log.
    }
    else
    {
        status = ENET_ReadFrame(ENET, &g_handle, data, length, 0, NULL);
        Comments: Call stack input API to deliver the data to stack
```

(continues on next page)

(continued from previous page)

```
    }
}
else if (status == kStatus_ENET_RxFrameError)
{
    Comments: Update the received buffer when a error frame is received.
    ENET_ReadFrame(ENET, &g_handle, NULL, 0, 0, NULL);
}
```

Parameters

- base – ENET peripheral base address.
- handle – The ENET handler structure. This is the same handler pointer used in the ENET_Init.
- data – The data buffer provided by user to store the frame which memory size should be at least "length".
- length – The size of the data buffer which is still the length of the received frame.
- ringId – The ring index or ring number.
- ts – The timestamp address to store received timestamp.

Returns

The execute status, successful or failure.

status_t ENET_SendFrame(ENET_Type *base, *enet_handle_t* *handle, const uint8_t *data, uint32_t length, uint8_t ringId, bool tsFlag, void *context)

Transmits an ENET frame for specified ring.

Note: The CRC is automatically appended to the data. Input the data to send without the CRC. This API uses memcpy to copy data from DMA buffer to application buffer, 4 bytes aligned data buffer in 32 bits platforms provided by user may let compiler use optimization instruction to reduce time consumption.

Parameters

- base – ENET peripheral base address.
- handle – The ENET handler pointer. This is the same handler pointer used in the ENET_Init.
- data – The data buffer provided by user to send.
- length – The length of the data to send.
- ringId – The ring index or ring number.
- tsFlag – Timestamp enable flag.
- context – Used by user to handle some events after transmit over.

Return values

- kStatus_Success – Send frame succeed.
- kStatus_ENET_TxFrameBusy – Transmit buffer descriptor is busy under transmission. The transmit busy happens when the data send rate is over the MAC capacity. The waiting mechanism is recommended to be added after each call return with kStatus_ENET_TxFrameBusy.

status_t ENET_SetTxReclaim(*enet_handle_t* *handle, bool isEnable, uint8_t ringId)

Enable or disable tx descriptors reclaim mechanism.

Note: This function must be called when no pending send frame action. Set enable if you want to reclaim context or timestamp in interrupt.

Parameters

- handle – The ENET handler pointer. This is the same handler pointer used in the ENET_Init.
- isEnable – Enable or disable flag.
- ringId – The ring index or ring number.

Return values

- kStatus_Success – Succeed to enable/disable Tx reclaim.
- kStatus_Fail – Fail to enable/disable Tx reclaim.

void ENET_ReclaimTxDescriptor(ENET_Type *base, *enet_handle_t* *handle, uint8_t ringId)

Reclaim tx descriptors. This function is used to update the tx descriptor status and store the tx timestamp when the 1588 feature is enabled. This is called by the transmit interupt IRQ handler after the complete of a frame transmission.

Parameters

- base – ENET peripheral base address.
- handle – The ENET handler pointer. This is the same handler pointer used in the ENET_Init.
- ringId – The ring index or ring number.

status_t ENET_GetRxFrame(ENET_Type *base, *enet_handle_t* *handle, *enet_rx_frame_struct_t* *rxFrame, uint8_t ringId)

Receives one frame in specified BD ring with zero copy.

This function uses the user-defined allocation and free callbacks. Every time application gets one frame through this function, driver stores the buffer address(es) in enet_buffer_struct_t and allocate new buffer(s) for the BD(s). If there's no memory buffer in the pool, this function drops current one frame to keep the Rx frame in BD ring is as fresh as possible.

Note: Application must provide a memory pool including at least BD number + n buffers in order for this function to work properly, because each BD must always take one buffer while driver is running, then other extra n buffer(s) can be taken by application. Here n is the ceil(max_frame_length(set by RCR) / bd_rx_size(set by MRBR)). Application must also provide an array structure in rxFrame->rxBuffArray with n index to receive one complete frame in any case.

Parameters

- base – ENET peripheral base address.
- handle – The ENET handler pointer. This is the same handler pointer used in the ENET_Init.
- rxFrame – The received frame information structure provided by user.
- ringId – The ring index or ring number.

Return values

- kStatus_Success – Succeed to get one frame and allocate new memory for Rx buffer.
- kStatus_ENET_RxFrameEmpty – There's no Rx frame in the BD.
- kStatus_ENET_RxFrameError – There's issue in this receiving.
- kStatus_ENET_RxFrameDrop – There's no new buffer memory for BD, drop this frame.

status_t ENET_StartTxFrame(ENET_Type *base, *enet_handle_t* *handle, *enet_tx_frame_struct_t* *txFrame, uint8_t ringId)

Sends one frame in specified BD ring with zero copy.

This function supports scattered buffer transmit, user needs to provide the buffer array.

Note: Tx reclaim should be enabled to ensure the Tx buffer ownership can be given back to application after Tx is over.

Parameters

- base – ENET peripheral base address.
- handle – The ENET handler pointer. This is the same handler pointer used in the ENET_Init.
- txFrame – The Tx frame structure.
- ringId – The ring index or ring number.

Return values

- kStatus_Success – Succeed to send one frame.
- kStatus_ENET_TxFrameBusy – The BD is not ready for Tx or the reclaim operation still not finishs.
- kStatus_ENET_TxFrameOverLen – The Tx frame length is over max ethernet frame length.

void ENET_TransmitIRQHandler(ENET_Type *base, *enet_handle_t* *handle)

The transmit IRQ handler.

Parameters

- base – ENET peripheral base address.
- handle – The ENET handler pointer.

void ENET_ReceiveIRQHandler(ENET_Type *base, *enet_handle_t* *handle)

The receive IRQ handler.

Parameters

- base – ENET peripheral base address.
- handle – The ENET handler pointer.

void ENET_ErrorIRQHandler(ENET_Type *base, *enet_handle_t* *handle)

Some special IRQ handler including the error, mii, wakeup irq handler.

Parameters

- base – ENET peripheral base address.
- handle – The ENET handler pointer.

void ENET_Ptp1588IRQHandler(ENET_Type *base)

the common IRQ handler for the 1588 irq handler.

This is used for the 1588 timer interrupt.

Parameters

- base – ENET peripheral base address.

void ENET_CommonFrame0IRQHandler(ENET_Type *base)

the common IRQ handler for the tx/rx/error etc irq handler.

This is used for the combined tx/rx/error interrupt for single/mutli-ring (frame 0).

Parameters

- base – ENET peripheral base address.

void ENET_CommonFrame0DriverIRQHandler(uint32_t instance)

Parameterized frame-0 (tx/rx/error) common IRQ handler.

Takes the instance directly to avoid the *ENET_GetInstance()* lookup in the ISR path. Each SoC binds this from the interrupt lines that carry its frame-0 tx/rx/error events.

Parameters

- instance – ENET peripheral instance number.

void ENET_Ptp1588DriverIRQHandler(uint32_t instance)

Parameterized 1588 timer common IRQ handler.

Takes the instance directly to avoid the *ENET_GetInstance()* lookup in the ISR path. Each SoC binds this from the interrupt line that carries its 1588 timer event.

Parameters

- instance – ENET peripheral instance number.

FSL_ENET_DRIVER_VERSION

Defines the driver version.

ENET_BUFFDESCRIPTOR_RX_EMPTY_MASK

Empty bit mask.

ENET_BUFFDESCRIPTOR_RX_SOFTOWNER1_MASK

Software owner one mask.

ENET_BUFFDESCRIPTOR_RX_WRAP_MASK

Next buffer descriptor is the start address.

ENET_BUFFDESCRIPTOR_RX_SOFTOWNER2_Mask

Software owner two mask.

ENET_BUFFDESCRIPTOR_RX_LAST_MASK

Last BD of the frame mask.

ENET_BUFFDESCRIPTOR_RX_MISS_MASK

Received because of the promiscuous mode.

ENET_BUFFDESCRIPTOR_RX_BROADCAST_MASK
Broadcast packet mask.

ENET_BUFFDESCRIPTOR_RX_MULTICAST_MASK
Multicast packet mask.

ENET_BUFFDESCRIPTOR_RX_LENVLIOLATE_MASK
Length violation mask.

ENET_BUFFDESCRIPTOR_RX_NOOCTET_MASK
Non-octet aligned frame mask.

ENET_BUFFDESCRIPTOR_RX_CRC_MASK
CRC error mask.

ENET_BUFFDESCRIPTOR_RX_OVERRUN_MASK
FIFO overrun mask.

ENET_BUFFDESCRIPTOR_RX_TRUNC_MASK
Frame is truncated mask.

ENET_BUFFDESCRIPTOR_TX_READY_MASK
Ready bit mask.

ENET_BUFFDESCRIPTOR_TX_SOFTOWENER1_MASK
Software owner one mask.

ENET_BUFFDESCRIPTOR_TX_WRAP_MASK
Wrap buffer descriptor mask.

ENET_BUFFDESCRIPTOR_TX_SOFTOWENER2_MASK
Software owner two mask.

ENET_BUFFDESCRIPTOR_TX_LAST_MASK
Last BD of the frame mask.

ENET_BUFFDESCRIPTOR_TX_TRANMITCRC_MASK
Transmit CRC mask.

ENET_FRAME_MAX_FRAMELEN
Default maximum Ethernet frame size without VLAN tag.

ENET_FRAME_VLAN_TAGLEN
Ethernet single VLAN tag size.

ENET_FRAME_CRC_LEN
CRC size in a frame.

ENET_FRAME_TX_LEN_LIMITATION(x)

ENET_FIFO_MIN_RX_FULL
ENET minimum receive FIFO full.

ENET_RX_MIN_BUFFERSIZE
ENET minimum buffer size.

ENET_PHY_MAXADDRESS
Maximum PHY address.

ENET_TX_INTERRUPT
Enet Tx interrupt flag.

ENET_RX_INTERRUPT
Enet Rx interrupt flag.

ENET_TS_INTERRUPT
Enet timestamp interrupt flag.

ENET_ERR_INTERRUPT
Enet error interrupt flag.

Defines the status return codes for transaction.

Values:

enumerator kStatus_ENET_InitMemoryFail
Init fails since buffer memory is not enough.

enumerator kStatus_ENET_RxFrameError
A frame received but data error happen.

enumerator kStatus_ENET_RxFrameFail
Failed to receive a frame.

enumerator kStatus_ENET_RxFrameEmpty
No frame arrive.

enumerator kStatus_ENET_RxFrameDrop
Rx frame is dropped since no buffer memory.

enumerator kStatus_ENET_TxFrameOverLen
Tx frame over length.

enumerator kStatus_ENET_TxFrameBusy
Tx buffer descriptors are under process.

enumerator kStatus_ENET_TxFrameFail
Transmit frame fail.

enum _enet_mii_mode

Defines the MII/RMII/RGMII mode for data interface between the MAC and the PHY.

Values:

enumerator kENET_MiiMode

MII mode for data interface.

enumerator kENET_RmiiMode

RMII mode for data interface.

enumerator kENET_RgmiiMode

RGMII mode for data interface.

enum _enet_mii_speed

Defines the 10/100/1000 Mbps speed for the MII data interface.

Notice: “kENET_MiiSpeed1000M” only supported when mii mode is “kENET_RgmiiMode”.

Values:

enumerator kENET_MiiSpeed10M

Speed 10 Mbps.

enumerator kENET_MiiSpeed100M

Speed 100 Mbps.

enumerator kENET_MiiSpeed1000M

Speed 1000M bps.

enum _enet_mii_duplex

Defines the half or full duplex for the MII data interface.

Values:

enumerator kENET_MiiHalfDuplex

Half duplex mode.

enumerator kENET_MiiFullDuplex

Full duplex mode.

enum _enet_mii_write

Define the MII opcode for normal MDIO_CLAUSES_22 Frame.

Values:

enumerator kENET_MiiWriteNoCompliant

Write frame operation, but not MII-compliant.

enumerator kENET_MiiWriteValidFrame

Write frame operation for a valid MII management frame.

enum _enet_mii_read

Defines the read operation for the MII management frame.

Values:

enumerator kENET_MiiReadValidFrame

Read frame operation for a valid MII management frame.

enumerator kENET_MiiReadNoCompliant

Read frame operation, but not MII-compliant.

enum _enet_mii_extend_opcode

Define the MII opcode for extended MDIO_CLAUSES_45 Frame.

Values:

enumerator kENET_MiiAddrWrite_C45

Address Write operation.

enumerator kENET_MiiWriteFrame_C45

Write frame operation for a valid MII management frame.

enumerator kENET_MiiReadFrame_C45

Read frame operation for a valid MII management frame.

enum _enet_special_control_flag

Defines a special configuration for ENET MAC controller.

These control flags are provided for special user requirements. Normally, these control flags are unused for ENET initialization. For special requirements, set the flags to macSpecialConfig in the enet_config_t. The kENET_ControlStoreAndFwdDisable is used to disable the FIFO store and forward. FIFO store and forward means that the FIFO read/send is started when a complete frame is stored in TX/RX FIFO. If this flag is set, configure rxFifoFullThreshold and txFifoWatermark in the enet_config_t.

Values:

enumerator kENET_ControlFlowControlEnable

Enable ENET flow control: pause frame.

enumerator kENET_ControlRxPayloadCheckEnable

Enable ENET receive payload length check.

enumerator kENET_ControlRxPadRemoveEnable

Padding is removed from received frames.

enumerator kENET_ControlRxBroadCastRejectEnable

Enable broadcast frame reject.

enumerator kENET_ControlMacAddrInsert

Enable MAC address insert.

enumerator kENET_ControlStoreAndFwdDisable
Enable FIFO store and forward.

enumerator kENET_ControlSMIPreambleDisable
Enable SMI preamble.

enumerator kENET_ControlPromiscuousEnable
Enable promiscuous mode.

enumerator kENET_ControlMIILoopEnable
Enable ENET MII loop back.

enumerator kENET_ControlVLANTagEnable
Enable normal VLAN (single vlan tag).

enumerator kENET_ControlSVLANEnable
Enable S-VLAN.

enumerator kENET_ControlVLANUseSecondTag
Enable extracting the second vlan tag for further processing.

enum _enet_interrupt_enable
List of interrupts supported by the peripheral. This enumeration uses one-bit encoding to allow a logical OR of multiple members. Members usually map to interrupt enable bits in one or more peripheral registers.

Values:

enumerator kENET_BabrInterrupt
Babbling receive error interrupt source

enumerator kENET_BabtInterrupt
Babbling transmit error interrupt source

enumerator kENET_GraceStopInterrupt
Graceful stop complete interrupt source

enumerator kENET_TxFrameInterrupt
TX FRAME interrupt source

enumerator kENET_TxBufferInterrupt
TX BUFFER interrupt source

enumerator kENET_RxFrameInterrupt
RX FRAME interrupt source

enumerator kENET_RxBufferInterrupt
RX BUFFER interrupt source

enumerator kENET_MiiInterrupt
MII interrupt source

enumerator kENET_EBusERInterrupt
Ethernet bus error interrupt source

enumerator kENET_LateCollisionInterrupt
Late collision interrupt source

enumerator kENET_RetryLimitInterrupt
Collision Retry Limit interrupt source

enumerator kENET_UnderrunInterrupt
Transmit FIFO underrun interrupt source

enumerator kENET_PayloadRxInterrupt
Payload Receive error interrupt source

enumerator kENET_WakeupInterrupt
WAKEUP interrupt source

enumerator kENET_TsAvailInterrupt
TS AVAIL interrupt source for PTP

enumerator kENET_TsTimerInterrupt
TS WRAP interrupt source for PTP

enum _enet_event
Defines the common interrupt event for callback use.
Values:

enumerator kENET_RxEvent
Receive event.

enumerator kENET_TxEvent
Transmit event.

enumerator kENET_ErrEvent
Error event: BABR/BABT/EBERR/LC/RL/UN/PLR .

enumerator kENET_WakeUpEvent
Wake up from sleep mode event.

enumerator kENET_TimeStampEvent
Time stamp event.

enumerator kENET_TimeStampAvailEvent
Time stamp available event.

enum _enet_idle_slope

Defines certain idle slope for bandwidth fraction.

Values:

enumerator kENET_IdleSlope1

The bandwidth fraction is about 0.002.

enumerator kENET_IdleSlope2

The bandwidth fraction is about 0.003.

enumerator kENET_IdleSlope4

The bandwidth fraction is about 0.008.

enumerator kENET_IdleSlope8

The bandwidth fraction is about 0.02.

enumerator kENET_IdleSlope16

The bandwidth fraction is about 0.03.

enumerator kENET_IdleSlope32

The bandwidth fraction is about 0.06.

enumerator kENET_IdleSlope64

The bandwidth fraction is about 0.11.

enumerator kENET_IdleSlope128

The bandwidth fraction is about 0.20.

enumerator kENET_IdleSlope256

The bandwidth fraction is about 0.33.

enumerator kENET_IdleSlope384

The bandwidth fraction is about 0.43.

enumerator kENET_IdleSlope512

The bandwidth fraction is about 0.50.

enumerator kENET_IdleSlope640

The bandwidth fraction is about 0.56.

enumerator kENET_IdleSlope768

The bandwidth fraction is about 0.60.

enumerator kENET_IdleSlope896

The bandwidth fraction is about 0.64.

enumerator kENET_IdleSlope1024

The bandwidth fraction is about 0.67.

enumerator kENET_IdleSlope1152
The bandwidth fraction is about 0.69.

enumerator kENET_IdleSlope1280
The bandwidth fraction is about 0.71.

enumerator kENET_IdleSlope1408
The bandwidth fraction is about 0.73.

enumerator kENET_IdleSlope1536
The bandwidth fraction is about 0.75.

enum _enet_tx_accelerator
Defines the transmit accelerator configuration.

Note that the hardware does not insert ICMPv6 protocol checksums as mentioned in errata ERR052152.

Values:

enumerator kENET_TxAccelIsShift16Enabled
Transmit FIFO shift-16.

enumerator kENET_TxAccelIpCheckEnabled
Insert IP header checksum.

enumerator kENET_TxAccelProtoCheckEnabled
Insert protocol checksum (TCP, UDP, ICMPv4).

enum _enet_rx_accelerator
Defines the receive accelerator configuration.

Note that the hardware does not validate ICMPv6 protocol checksums as mentioned in errata ERR052152.

Values:

enumerator kENET_RxAccelPadRemoveEnabled
Padding removal for short IP frames.

enumerator kENET_RxAccelIpCheckEnabled
Discard with wrong IP header checksum.

enumerator kENET_RxAccelProtoCheckEnabled
Discard with wrong protocol checksum (TCP, UDP, ICMPv4).

enumerator kENET_RxAccelMacCheckEnabled
Discard with Mac layer errors.

enumerator kENET_RxAccelisShift16Enabled
Receive FIFO shift-16.

typedef enum *_enet_mii_mode* enet_mii_mode_t

Defines the MII/RMII/RGMII mode for data interface between the MAC and the PHY.

typedef enum *_enet_mii_speed* enet_mii_speed_t

Defines the 10/100/1000 Mbps speed for the MII data interface.

Notice: "kENET_MiiSpeed1000M" only supported when mii mode is "kENET_RgmiiMode".

typedef enum *_enet_mii_duplex* enet_mii_duplex_t

Defines the half or full duplex for the MII data interface.

typedef enum *_enet_mii_write* enet_mii_write_t

Define the MII opcode for normal MDIO_CLAUSES_22 Frame.

typedef enum *_enet_mii_read* enet_mii_read_t

Defines the read operation for the MII management frame.

typedef enum *_enet_mii_extend_opcode* enet_mii_extend_opcode

Define the MII opcode for extended MDIO_CLAUSES_45 Frame.

typedef enum *_enet_special_control_flag* enet_special_control_flag_t

Defines a special configuration for ENET MAC controller.

These control flags are provided for special user requirements. Normally, these control flags are unused for ENET initialization. For special requirements, set the flags to macSpecialConfig in the enet_config_t. The kENET_ControlStoreAndFwdDisable is used to disable the FIFO store and forward. FIFO store and forward means that the FIFO read/send is started when a complete frame is stored in TX/RX FIFO. If this flag is set, configure rxFifoFullThreshold and txFifoWatermark in the enet_config_t.

typedef enum *_enet_interrupt_enable* enet_interrupt_enable_t

List of interrupts supported by the peripheral. This enumeration uses one-bit encoding to allow a logical OR of multiple members. Members usually map to interrupt enable bits in one or more peripheral registers.

typedef enum *_enet_event* enet_event_t

Defines the common interrupt event for callback use.

typedef enum *_enet_idle_slope* enet_idle_slope_t

Defines certain idle slope for bandwidth fraction.

typedef enum *_enet_tx_accelerator* enet_tx_accelerator_t

Defines the transmit accelerator configuration.

Note that the hardware does not insert ICMPv6 protocol checksums as mentioned in errata ERR052152.

typedef enum *_enet_rx_accelerator* enet_rx_accelerator_t

Defines the receive accelerator configuration.

Note that the hardware does not validate ICMPv6 protocol checksums as mentioned in errata ERR052152.

typedef struct *_enet_rx_bd_struct* enet_rx_bd_struct_t

Defines the receive buffer descriptor structure for the little endian system.

typedef struct *_enet_tx_bd_struct* enet_tx_bd_struct_t

Defines the enhanced transmit buffer descriptor structure for the little endian system.

typedef struct *_enet_data_error_stats* enet_data_error_stats_t

Defines the ENET data error statistics structure.

typedef struct *_enet_rx_frame_error* enet_rx_frame_error_t

Defines the Rx frame error structure.

typedef struct *_enet_transfer_stats* enet_transfer_stats_t

Defines the ENET transfer statistics structure.

typedef struct *enet_frame_info* enet_frame_info_t

Defines the frame info structure.

typedef struct *_enet_tx_dirty_ring* enet_tx_dirty_ring_t

Defines the ENET transmit dirty addresses ring/queue structure.

typedef void *(*enet_rx_alloc_callback_t)(ENET_Type *base, void *userData, uint8_t ringId)

Defines the ENET Rx memory buffer alloc function pointer.

typedef void (*enet_rx_free_callback_t)(ENET_Type *base, void *buffer, void *userData, uint8_t ringId)

Defines the ENET Rx memory buffer free function pointer.

typedef struct *_enet_buffer_config* enet_buffer_config_t

Defines the receive buffer descriptor configuration structure.

Note that for the internal DMA requirements, the buffers have a corresponding alignment requirements.

a. The aligned receive and transmit buffer size must be evenly divisible by ENET_BUFF_ALIGNMENT. when the data buffers are in cacheable region when cache is enabled, all those size should be aligned to the maximum value of “ENET_BUFF_ALIGNMENT” and the cache line size.

b. The aligned transmit and receive buffer descriptor start address must be at least 64 bit aligned. However, it’s recommended to be evenly divisible by ENET_BUFF_ALIGNMENT. buffer descriptors should be put in non-cacheable region when cache is enabled.

c. The aligned transmit and receive data buffer start address must be evenly divisible by ENET_BUFF_ALIGNMENT. Receive buffers should be continuous with the total size equal to “rxBdNumber * rxBuffSizeAlign”. Transmit buffers should be continuous with the total size equal to “txBdNumber * txBuffSizeAlign”. when the data buffers are in cacheable region when cache is enabled, all those size should be aligned to the maximum value of “ENET_BUFF_ALIGNMENT” and the cache line size.

typedef struct *_enet_intcoalesce_config* enet_intcoalesce_config_t

Defines the interrupt coalescing configure structure.

typedef struct *_enet_avb_config* enet_avb_config_t

Defines the ENET AVB Configure structure.

This is used for to configure the extended ring 1 and ring 2.

a. The classification match format is (CMP3 « 12) | (CMP2 « 8) | (CMP1 « 4) | CMP0. composed of four 3-bit compared VLAN priority field cmp0~cmp3, cm0 ~ cmp3 are used in parallel.

If CMP1,2,3 are not unused, please set them to the same value as CMP0.

a. The idleSlope is used to calculate the Band Width fraction, BW fraction = 1 / (1 + 512/idleSlope). For avb configuration, the BW fraction of Class 1 and Class 2 combined must not exceed 0.75.

typedef struct *_enet_handle* enet_handle_t

typedef void (*enet_callback_t)(ENET_Type *base, *enet_handle_t* *handle, *enet_event_t* event, *enet_frame_info_t* *frameInfo, void *userData)

ENET callback function.

typedef struct *_enet_config* enet_config_t

Defines the basic configuration structure for the ENET device.

Note:

a. macSpecialConfig is used for a special control configuration, A logical OR of "enet_special_control_flag_t". For a special configuration for MAC, set this parameter to 0.

b. txWatermark is used for a cut-through operation. It is in steps of 64 bytes: 0/1 - 64 bytes written to TX FIFO before transmission of a frame begins. 2 - 128 bytes written to TX FIFO .... 3 - 192 bytes written to TX FIFO .... The maximum of txWatermark is 0x2F - 4032 bytes written to TX FIFO .... txWatermark allows minimizing the transmit latency to set the txWatermark to 0 or 1 or for larger bus access latency 3 or larger due to contention for the system bus.

c. rxFifoFullThreshold is similar to the txWatermark for cut-through operation in RX. It is in 64-bit words. The minimum is ENET_FIFO_MIN_RX_FULL and the maximum is 0xFF. If the end of the frame is stored in FIFO and the frame size if smaller than the txWatermark, the frame is still transmitted. The rule is the same for rxFifoFullThreshold in the receive direction.

d. When "kENET_ControlFlowControlEnable" is set in the macSpecialConfig, ensure that the pauseDuration, rxFifoEmptyThreshold, and rxFifoStatEmptyThreshold are set for flow control enabled case.

e. When "kENET_ControlStoreAndFwdDisabled" is set in the macSpecialConfig, ensure that the rxFifoFullThreshold and txFifoWatermark are set for store and forward disable.

f. The rxAccelerConfig and txAccelerConfig default setting with 0 - accelerator are disabled. The "enet_tx_accelerator_t" and "enet_rx_accelerator_t" are recommended to be used to enable the transmit and receive accelerator. After the accelerators are enabled, the store and forward feature should be enabled. As a result, kENET_ControlStoreAndFwdDisabled should not be set.

g. The intCoalesceCfg can be used in the rx or tx enabled cases to decrese the CPU loading.

typedef struct *_enet_tx_bd_ring* enet_tx_bd_ring_t

Defines the ENET transmit buffer descriptor ring/queue structure.

typedef struct *_enet_rx_bd_ring* enet_rx_bd_ring_t

Defines the ENET receive buffer descriptor ring/queue structure.

typedef struct *_enet_buffer_struct* enet_buffer_struct_t

typedef struct *_enet_rx_frame_attribute_struct* enet_rx_frame_attribute_t

typedef struct *_enet_rx_frame_struct* enet_rx_frame_struct_t

typedef struct *_enet_tx_frame_struct* enet_tx_frame_struct_t

typedef void (*enet_isr_t)(ENET_Type *base, *enet_handle_t* *handle)

Define interrupt IRQ handler.

const *clock_ip_name_t* s_enetClock[]

Pointers to enet clocks for each instance.

const *clock_ip_name_t* s_enetExtraClock[]

uint32_t ENET_GetInstance(ENET_Type *base)

Get the ENET instance from peripheral base address.

Parameters

- base – ENET peripheral base address.

Returns

ENET instance.

ENET_BUFFDESCRIPTOR_RX_ERR_MASK

Defines the receive error status flag mask.

struct _enet_rx_bd_struct

#include <fsl_enet.h> Defines the receive buffer descriptor structure for the little endian system.

Public Members

uint16_t length

Buffer descriptor data length.

uint16_t control

Buffer descriptor control and status.

uint32_t buffer

Data buffer pointer.

struct _enet_tx_bd_struct

#include <fsl_enet.h> Defines the enhanced transmit buffer descriptor structure for the little endian system.

Public Members

uint16_t length
Buffer descriptor data length.

uint16_t control
Buffer descriptor control and status.

uint32_t buffer
Data buffer pointer.

struct __enet__data__error__stats
#include <fsl_enet.h> Defines the ENET data error statistics structure.

Public Members

uint32_t statsRxLenGreaterErr
Receive length greater than RCR[MAX_FL].

uint32_t statsRxAlignErr
Receive non-octet alignment/

uint32_t statsRxFcsErr
Receive CRC error.

uint32_t statsRxOverRunErr
Receive over run.

uint32_t statsRxTruncateErr
Receive truncate.

struct __enet__rx__frame__error
#include <fsl_enet.h> Defines the Rx frame error structure.

Public Members

bool statsRxTruncateErr
Receive truncate.

bool statsRxOverRunErr
Receive over run.

bool statsRxFcsErr
Receive CRC error.

bool statsRxAlignErr
Receive non-octet alignment.

bool statsRxLenGreaterErr
Receive length greater than RCR[MAX_FL].

struct __enet__transfer__stats
#include <fsl_enet.h> Defines the ENET transfer statistics structure.

Public Members

uint32_t statsRxFrameCount
Rx frame number.

uint32_t statsRxFrameOk
Good Rx frame number.

uint32_t statsRxCrcErr
Rx frame number with CRC error.

uint32_t statsRxAlignErr
Rx frame number with alignment error.

uint32_t statsRxDropInvalidSFD
Dropped frame number due to invalid SFD.

uint32_t statsRxFifoOverflowErr
Rx FIFO overflow count.

uint32_t statsTxFrameCount
Tx frame number.

uint32_t statsTxFrameOk
Good Tx frame number.

uint32_t statsTxCrcAlignErr
The transmit frame is error.

uint32_t statsTxFifoUnderRunErr
Tx FIFO underrun count.

struct enet__frame__info
#include <fsl_enet.h> Defines the frame info structure.

Public Members

void *context
User specified data

struct __enet__tx__dirty__ring
#include <fsl_enet.h> Defines the ENET transmit dirty addresses ring/queue structure.

Public Members

enet_frame_info_t *txDirtyBase

Dirty buffer descriptor base address pointer.

uint16_t txGenIdx

tx generate index.

uint16_t txConsumIdx

tx consume index.

uint16_t txRingLen

tx ring length.

bool isFull

tx ring is full flag.

struct __enet_buffer_config

#include <fsl_enet.h> Defines the receive buffer descriptor configuration structure.

Note that for the internal DMA requirements, the buffers have a corresponding alignment requirements.

a. The aligned receive and transmit buffer size must be evenly divisible by ENET_BUFF_ALIGNMENT. when the data buffers are in cacheable region when cache is enabled, all those size should be aligned to the maximum value of "ENET_BUFF_ALIGNMENT" and the cache line size.

b. The aligned transmit and receive buffer descriptor start address must be at least 64 bit aligned. However, it's recommended to be evenly divisible by ENET_BUFF_ALIGNMENT. buffer descriptors should be put in non-cacheable region when cache is enabled.

c. The aligned transmit and receive data buffer start address must be evenly divisible by ENET_BUFF_ALIGNMENT. Receive buffers should be continuous with the total size equal to "rxBdNumber * rxBuffSizeAlign". Transmit buffers should be continuous with the total size equal to "txBdNumber * txBuffSizeAlign". when the data buffers are in cacheable region when cache is enabled, all those size should be aligned to the maximum value of "ENET_BUFF_ALIGNMENT" and the cache line size.

Public Members

uint16_t rxBdNumber

Receive buffer descriptor number.

uint16_t txBdNumber

Transmit buffer descriptor number.

uint16_t rxBuffSizeAlign

Aligned receive data buffer size.

uint16_t txBuffSizeAlign
: Aligned transmit data buffer size.

volatile *enet_rx_bd_struct_t* *rxBdStartAddrAlign
: Aligned receive buffer descriptor start address: should be non-cacheable.

volatile *enet_tx_bd_struct_t* *txBdStartAddrAlign
: Aligned transmit buffer descriptor start address: should be non-cacheable.

uint8_t *rxBufferAlign
: Receive data buffer start address.

uint8_t *txBufferAlign
: Transmit data buffer start address.

bool rxMaintainEnable
: Receive buffer cache maintain.

bool txMaintainEnable
: Transmit buffer cache maintain.

enet_frame_info_t *txFrameInfo
: Transmit frame information start address.

struct __enet__intcoalesce__config
: *#include <fsl_enet.h>* Defines the interrupt coalescing configure structure.

Public Members

uint8_t txCoalesceFrameCount[1]
: Transmit interrupt coalescing frame count threshold.

uint16_t txCoalesceTimeCount[1]
: Transmit interrupt coalescing timer count threshold.

uint8_t rxCoalesceFrameCount[1]
: Receive interrupt coalescing frame count threshold.

uint16_t rxCoalesceTimeCount[1]
: Receive interrupt coalescing timer count threshold.

struct __enet__avb__config
: *#include <fsl_enet.h>* Defines the ENET AVB Configure structure.

This is used for to configure the extended ring 1 and ring 2.

a. The classification match format is (CMP3 « 12) | (CMP2 « 8) | (CMP1 « 4) | CMP0. composed of four 3-bit compared VLAN priority field cmp0~cmp3, cm0 ~ cmp3 are used in parallel.

If CMP1,2,3 are not unused, please set them to the same value as CMP0.

a. The idleSlope is used to calculate the Band Width fraction, BW fraction = 1 / (1 + 512/idleSlope). For avb configuration, the BW fraction of Class 1 and Class 2 combined must not exceed 0.75.

Public Members

uint16_t rxClassifyMatch[1 - 1]

The classification match value for the ring.

enet_idle_slope_t idleSlope[1 - 1]

The idle slope for certian bandwidth fraction.

struct __enet__config

#include <fsl_enet.h> Defines the basic configuration structure for the ENET device.

Note:

a. macSpecialConfig is used for a special control configuration, A logical OR of "enet_special_control_flag_t". For a special configuration for MAC, set this parameter to 0.

b. txWatermark is used for a cut-through operation. It is in steps of 64 bytes: 0/1 - 64 bytes written to TX FIFO before transmission of a frame begins. 2 - 128 bytes written to TX FIFO .... 3 - 192 bytes written to TX FIFO .... The maximum of txWatermark is 0x2F - 4032 bytes written to TX FIFO .... txWatermark allows minimizing the transmit latency to set the txWatermark to 0 or 1 or for larger bus access latency 3 or larger due to contention for the system bus.

c. rxFifoFullThreshold is similar to the txWatermark for cut-through operation in RX. It is in 64-bit words. The minimum is ENET_FIFO_MIN_RX_FULL and the maximum is 0xFF. If the end of the frame is stored in FIFO and the frame size if smaller than the txWatermark, the frame is still transmitted. The rule is the same for rxFifoFullThreshold in the receive direction.

d. When "kENET_ControlFlowControlEnable" is set in the macSpecialConfig, ensure that the pauseDuration, rxFifoEmptyThreshold, and rxFifoStatEmptyThreshold are set for flow control enabled case.

e. When "kENET_ControlStoreAndFwdDisabled" is set in the macSpecialConfig, ensure that the rxFifoFullThreshold and txFifoWatermark are set for store and forward disable.

f. The rxAccelerConfig and txAccelerConfig default setting with 0 - accelerator are disabled. The "enet_tx_accelerator_t" and "enet_rx_accelerator_t" are recommended to be used to enable the transmit and receive accelerator. After the accelerators are enabled, the store and forward feature should be enabled. As a result, kENET_ControlStoreAndFwdDisabled should not be set.

g. The intCoalesceCfg can be used in the rx or tx enabled cases to decrese the CPU loading.

Public Members

uint32_t macSpecialConfig

Mac special configuration. A logical OR of "enet_special_control_flag_t".

uint32_t interrupt

Mac interrupt source. A logical OR of “enet_interrupt_enable_t”.

uint16_t rxMaxFrameLen

Receive maximum frame length.

enet_mii_mode_t miiMode

MII mode.

enet_mii_speed_t miiSpeed

MII Speed.

enet_mii_duplex_t miiDuplex

MII duplex.

uint8_t rxAccelerConfig

Receive accelerator, A logical OR of “enet_rx_accelerator_t”.

uint8_t txAccelerConfig

Transmit accelerator, A logical OR of “enet_rx_accelerator_t”.

uint16_t pauseDuration

For flow control enabled case: Pause duration.

uint8_t rxFifoEmptyThreshold

For flow control enabled case: when RX FIFO level reaches this value, it makes MAC generate XOFF pause frame.

uint8_t rxFifoStatEmptyThreshold

For flow control enabled case: number of frames in the receive FIFO, independent of size, that can be accept. If the limit is reached, reception continues and a pause frame is triggered.

uint8_t rxFifoFullThreshold

For store and forward disable case, the data required in RX FIFO to notify the MAC receive ready status.

uint8_t txFifoWatermark

For store and forward disable case, the data required in TX FIFO before a frame transmit start.

enet_intcoalesce_config_t *intCoalesceCfg

If the interrupt coalsecence is not required in the ring n(0,1,2), please set to NULL.

uint8_t ringNum

Number of used rings. default with 1 — single ring.

enet_rx_alloc_callback_t rxBuffAlloc

Callback function to alloc memory, must be provided for zero-copy Rx.

enet_rx_free_callback_t rxBuffFree
Callback function to free memory, must be provided for zero-copy Rx.

enet_callback_t callback
General callback function.

void *userData
Callback function parameter.

struct __enet_tx_bd_ring
#include <fsl_enet.h> Defines the ENET transmit buffer descriptor ring/queue structure.

Public Members

volatile *enet_tx_bd_struct_t* *txBdBase
Buffer descriptor base address pointer.

uint16_t txGenIdx
The current available transmit buffer descriptor pointer.

uint16_t txConsumIdx
Transmit consume index.

volatile uint16_t txDescUsed
Transmit descriptor used number.

uint16_t txRingLen
Transmit ring length.

struct __enet_rx_bd_ring
#include <fsl_enet.h> Defines the ENET receive buffer descriptor ring/queue structure.

Public Members

volatile *enet_rx_bd_struct_t* *rxBdBase
Buffer descriptor base address pointer.

uint16_t rxGenIdx
The current available receive buffer descriptor pointer.

uint16_t rxRingLen
Receive ring length.

struct __enet_handle
#include <fsl_enet.h> Defines the ENET handler structure.

Public Members

enet_rx_bd_ring_t rxBdRing[1]
Receive buffer descriptor.

enet_tx_bd_ring_t txBdRing[1]
Transmit buffer descriptor.

uint16_t rxBuffSizeAlign[1]
Receive buffer size alignment.

uint16_t txBuffSizeAlign[1]
Transmit buffer size alignment.

bool rxMaintainEnable[1]
Receive buffer cache maintain.

bool txMaintainEnable[1]
Transmit buffer cache maintain.

uint8_t ringNum
Number of used rings.

enet_callback_t callback
Callback function.

void *userData
Callback function parameter.

enet_tx_dirty_ring_t txDirtyRing[1]
Ring to store tx frame information.

bool txReclaimEnable[1]
Tx reclaim enable flag.

enet_rx_alloc_callback_t rxBuffAlloc
Callback function to alloc memory for zero copy Rx.

enet_rx_free_callback_t rxBuffFree
Callback function to free memory for zero copy Rx.

uint8_t multicastCount[64]
Multicast collisions counter

uint32_t enetClock
The clock of enet peripheral, to caculate core cycles for PTP timestamp.

uint32_t tsDelayCount
The count of core cycles for PTP timestamp capture delay.

struct __enet__buffer__struct

#include <fsl_enet.h>

Public Members

void *buffer

The buffer store the whole or partial frame.

uint16_t length

The byte length of this buffer.

struct __enet__rx__frame__attribute__struct

#include <fsl_enet.h>

Public Members

bool promiscuous

This frame is received because of promiscuous mode.

struct __enet__rx__frame__struct

#include <fsl_enet.h>

Public Members

enet_buffer_struct_t *rxBuffArray

Rx frame buffer structure.

uint16_t totLen

Rx frame total length.

enet_rx_frame_attribute_t rxAttribute

Rx frame attribute structure.

enet_rx_frame_error_t rxFrameError

Rx frame error.

struct __enet__tx__frame__struct

#include <fsl_enet.h>

Public Members

enet_buffer_struct_t *txBuffArray

Tx frame buffer structure.

uint32_t txBuffNum

Buffer number of this Tx frame.

void *context

Driver reclaims and gives it in Tx over callback, usually store network packet header.

2.18 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

2.19 FlexCAN: Flex Controller Area Network Driver

2.20 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_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 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_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.

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.

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_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_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_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_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.

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_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_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_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.

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_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.

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 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_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_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.

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 ___unnamed9___

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 ___unnamed11___

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 ___unnamed13___

Public Members

struct __flexcan_frame

struct __flexcan_frame

struct ___unnamed15___

Public Members

uint32_t dataWord0
: CAN Frame payload word0.

uint32_t dataWord1
: CAN Frame payload word1.

struct ___unnamed17___

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 ___unnamed19___

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 ___unnamed21___

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 ___unnamed23___

Public Members

struct __flexcan__fd__frame

struct __flexcan__fd__frame

struct ___unnamed25___

Public Members

uint32_t dataWord[16]

CAN FD Frame payload, 16 double word maximum.

struct ___unnamed27___

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 ___unnamed29___

Public Members

struct __flexcan__config

struct __flexcan__config

struct ___unnamed31___

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 ___unnamed33___

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 ___unnamed35___

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 ___unnamed39___

< 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 ___unnamed41___

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 ___unnamed37___

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 ___unnamed43___

< 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 ___unnamed45___

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.

2.21 FlexIO: FlexIO Driver

2.22 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

2.23 FlexIO eDMA I2S Driver

void FLEXIO_I2S_TransferTxCreateHandleEDMA(*FLEXIO_I2S_Type* *base, *flexio_i2s_edma_handle_t* *handle, *flexio_i2s_edma_callback_t* callback, void *userData, *edma_handle_t* *dmaHandle)

Initializes the FlexIO I2S eDMA handle.

This function initializes the FlexIO I2S master DMA handle which can be used for other FlexIO I2S master transactional APIs. Usually, for a specified FlexIO I2S instance, call this API once to get the initialized handle.

Parameters

- base – FlexIO I2S peripheral base address.
- handle – FlexIO I2S eDMA handle pointer.
- callback – FlexIO I2S eDMA callback function called while finished a block.
- userData – User parameter for callback.
- dmaHandle – eDMA handle for FlexIO I2S. This handle is a static value allocated by users.

void FLEXIO_I2S_TransferRxCreateHandleEDMA(*FLEXIO_I2S_Type* *base, *flexio_i2s_edma_handle_t* *handle, *flexio_i2s_edma_callback_t* callback, void *userData, *edma_handle_t* *dmaHandle)

Initializes the FlexIO I2S Rx eDMA handle.

This function initializes the FlexIO I2S slave DMA handle which can be used for other FlexIO I2S master transactional APIs. Usually, for a specified FlexIO I2S instance, call this API once to get the initialized handle.

Parameters

- base – FlexIO I2S peripheral base address.
- handle – FlexIO I2S eDMA handle pointer.
- callback – FlexIO I2S eDMA callback function called while finished a block.
- userData – User parameter for callback.
- dmaHandle – eDMA handle for FlexIO I2S. This handle is a static value allocated by users.

void FLEXIO_I2S_TransferSetFormatEDMA(*FLEXIO_I2S_Type* *base, *flexio_i2s_edma_handle_t* *handle, *flexio_i2s_format_t* *format, uint32_t srcClock_Hz)

Configures the FlexIO I2S Tx audio format.

Audio format can be changed in run-time of FlexIO I2S. This function configures the sample rate and audio data format to be transferred. This function also sets the eDMA parameter according to format.

Parameters

- base – FlexIO I2S peripheral base address.
- handle – FlexIO I2S eDMA handle pointer
- format – Pointer to FlexIO I2S audio data format structure.
- srcClock_Hz – FlexIO I2S clock source frequency in Hz, it should be 0 while in slave mode.

status_t FLEXIO_I2S_TransferSendEDMA(*FLEXIO_I2S_Type* *base, *flexio_i2s_edma_handle_t* *handle, *flexio_i2s_transfer_t* *xfer)

Performs a non-blocking FlexIO I2S transfer using DMA.

Note: This interface returned immediately after transfer initiates. Users should call FLEXIO_I2S_GetTransferStatus to poll the transfer status and check whether the FlexIO I2S transfer is finished.

Parameters

- base – FlexIO I2S peripheral base address.
- handle – FlexIO I2S DMA handle pointer.
- xfer – Pointer to DMA transfer structure.

Return values

- kStatus_Success – Start a FlexIO I2S eDMA send successfully.
- kStatus_InvalidArgument – The input arguments is invalid.
- kStatus_TxBusy – FlexIO I2S is busy sending data.

status_t FLEXIO_I2S_TransferReceiveEDMA(*FLEXIO_I2S_Type* *base, *flexio_i2s_edma_handle_t* *handle, *flexio_i2s_transfer_t* *xfer)

Performs a non-blocking FlexIO I2S receive using eDMA.

Note: This interface returned immediately after transfer initiates. Users should call FLEXIO_I2S_GetReceiveRemainingBytes to poll the transfer status and check whether the FlexIO I2S transfer is finished.

Parameters

- base – FlexIO I2S peripheral base address.
- handle – FlexIO I2S DMA handle pointer.
- xfer – Pointer to DMA transfer structure.

Return values

- kStatus_Success – Start a FlexIO I2S eDMA receive successfully.
- kStatus_InvalidArgument – The input arguments is invalid.
- kStatus_RxBusy – FlexIO I2S is busy receiving data.

void FLEXIO_I2S_TransferAbortSendEDMA(*FLEXIO_I2S_Type* *base, *flexio_i2s_edma_handle_t* *handle)

Aborts a FlexIO I2S transfer using eDMA.

Parameters

- base – FlexIO I2S peripheral base address.
- handle – FlexIO I2S DMA handle pointer.

void FLEXIO_I2S_TransferAbortReceiveEDMA(*FLEXIO_I2S_Type* *base, *flexio_i2s_edma_handle_t* *handle)

Aborts a FlexIO I2S receive using eDMA.

Parameters

- base – FlexIO I2S peripheral base address.
- handle – FlexIO I2S DMA handle pointer.

status_t FLEXIO_I2S_TransferGetSendCountEDMA(*FLEXIO_I2S_Type* *base, *flexio_i2s_edma_handle_t* *handle, size_t *count)

Gets the remaining bytes to be sent.

Parameters

- base – FlexIO I2S peripheral base address.
- handle – FlexIO I2S DMA handle pointer.
- 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_TransferGetReceiveCountEDMA(*FLEXIO_I2S_Type* *base, *flexio_i2s_edma_handle_t* *handle, size_t *count)

Get the remaining bytes to be received.

Parameters

- base – FlexIO I2S peripheral base address.
- handle – FlexIO I2S DMA handle pointer.
- count – Bytes received.

Return values

- kStatus_Success – Succeed get the transfer count.
- kStatus_NoTransferInProgress – There is not a non-blocking transaction currently in progress.

FSL_FLEXIO_I2S_EDMA_DRIVER_VERSION

FlexIO I2S EDMA driver version 2.1.9.

typedef struct *_flexio_i2s_edma_handle* flexio_i2s_edma_handle_t

typedef void (*flexio_i2s_edma_callback_t)(*FLEXIO_I2S_Type* *base, *flexio_i2s_edma_handle_t* *handle, *status_t* status, void *userData)

FlexIO I2S eDMA transfer callback function for finish and error.

struct _flexio_i2s_edma_handle

#include <fsl_flexio_i2s_edma.h> FlexIO I2S DMA transfer handle, users should not touch the content of the handle.

Public Members

edma_handle_t *dmaHandle

DMA handler for FlexIO I2S send

uint8_t bytesPerFrame
Bytes in a frame

uint8_t nbytes
eDMA minor byte transfer count initially configured.

uint32_t state
Internal state for FlexIO I2S eDMA transfer

flexio_i2s_edma_callback_t callback
Callback for users while transfer finish or error occurred

void *userData
User callback parameter

edma_tcd_t tcd[(4U) + 1U]
TCD pool for eDMA transfer.

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

2.24 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

2.25 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

2.26 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.

2.27 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

2.28 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.

2.29 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 ___unnamed100___

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.

2.30 FlexPWM: Enhanced Flex Pulse Width Modulator

void FLEXPWM_GetDefaultSubmoduleConfig(*flexpwm_submodule_config_t* *config)

Get default configuration for FlexPWM submodule.

This function initializes the FlexPWM submodule configuration structure with safe default values. The default configuration can be used directly or modified before calling *FLEXPWM_ConfigSubmodule()*.

Default values:

- Clock source: IPBus clock (kFLEXPWM_ClockSource_IPBusClock)
- Prescaler: Divide by 1 (no prescaling)
- Counter initial value: 0
- Counter modulo value: 0xFFFF (maximum range)
- Initialization source: Local sync
- Load mode: Opportunity (load at PWM reload opportunity)
- Load frequency: 0 (every PWM cycle)
- Half-cycle reload: Disabled
- Full-cycle reload: Enabled
- Reload source: Local reload
- Half-cycle value: 0
- Debug mode: Disabled (PWM stops in Debug mode)
- Wait mode: Disabled (PWM stops in Wait mode)

Parameters

- config – Pointer to the submodule configuration structure.

void FLEXPWM_ConfigSubmodule(PWM_Type *base, uint8_t submodule, const *flexpwm_submodule_config_t* *config)

Configure a FlexPWM submodule.

This function configures the basic parameters of a FlexPWM submodule, including:

- Clock source and prescaler (CTRL2[CLK_SEL], CTRL[PRSC])
- Counter initialization and modulo values (INIT, VAL1)
- Counter initialization source (CTRL2[INIT_SEL])
- Register reload mode, frequency, and timing (CTRL[LDMOD, LDFQ, HALF, FULL], CTRL2[RELOAD_SEL])

- Half-cycle reload point (VAL0)
- Debug and Wait mode behavior (CTRL2[DBGEN, WAITEN])

Note: Prerequisites:

- Call *FLEXPWM_Init()* before this function to enable the peripheral clock.

Note: Configuration write behavior:

- This function writes configuration values to buffered registers.
- The configuration does NOT take effect immediately.
- After calling this function, you must set MCTRL[LDOK] bit using a separate function to transfer buffered register values to active registers.
- The actual transfer timing depends on the configured reload mode and reload opportunities.

Note: This function does NOT:

- Set the MCTRL[LDOK] bit (you must call a separate LDOK function)
- Start the PWM counter (use a separate start API)
- Configure PWM waveform parameters (VAL2-VAL5, deadtime, polarity, etc.)
- Configure fault protection or capture functionality
- Configure FORCE_OUT behavior (use FLEXPWM_ConfigForceOut)

Parameters

- base – FlexPWM peripheral base address.
- submodule – Submodule number (0-3).
- config – Pointer to the submodule configuration structure.

void FLEXPWM_ConfigPWM(PWM_Type *base, uint8_t submodule, const *flexpwm_pwm_config_t* *config)

Configure PWM output for a FlexPWM submodule.

This function configures the PWM output parameters for a FlexPWM submodule, including:

- Compare values (VAL2, VAL3, VAL4, VAL5)
- Output polarity (POLA, POLB)
- Initial values (PWM23_INIT, PWM45_INIT)
- Operating mode: independent or complementary (INDEP)
- Complementary mode source selection (MCTRL[IPOL]) - only in complementary mode

---

Note: For complementary mode (complementary = true):

- Only the selected channel compare values (pwma if ipolSource=kFLEXPWM_IPOL_PWM23, or pwmb if ipolSource=kFLEXPWM_IPOL_PWM45) needs to be configured.
- But polarity still needs to be set for the complementary output to work correctly.

- The unselected channel will be ignored by the hardware.
- MCTRL[IPOL] will be set according to ipolSource parameter.

Note: For independent mode (complementary = false):

- Both pwma and pwmb must be fully configured.
- INDEP bit will be set to 1 in SMxCTRL2.
- MCTRL[IPOL] is ignored by the hardware in this mode.

Note: This function does NOT:

- Set the MCTRL[LDOK] bit (you must call a separate LDOK function)
- Trigger FORCE_OUT events (in complementary mode, MCTRL[IPOL] requires FORCE_OUT to take effect)
- Enable PWM_A Output or PWM_B Output (use a separate API to enable outputs)

Parameters

- base – FlexPWM peripheral base address.
- submodule – Submodule number (0-3).
- config – Pointer to the PWM configuration structure.

void FLEXPWM_ConfigPWMChannelX(PWM_Type *base, uint8_t submodule, const *flexpwm_pwm_channel_config_t* *pwmx)

Configure PWM_X output for a FlexPWM submodule.

This function configures the PWM_X output parameters for a FlexPWM submodule, including:

- Compare values (VAL0, VAL1)
- Output polarity (POLX)
- Initial values (PWMX_INIT)

Note: When submodule works in local sync mode, PWM_X polarity is inverted, because VAL1 determines both counter modulo and PWM_X turn-off edge. VAL0 determines PWM_X turn-on edge. Recommend setting submodule counter works in master sync mode if you want to use the PWM_X output, then both VAL0 and VAL1 can be used to control PWM_X waveform shape and polarity is normal.

Note: This function does NOT:

- Set the MCTRL[LDOK] bit (you must call a separate LDOK function)
- Enable PWM_X Output (use a separate API to enable outputs)

Parameters

- base – FlexPWM peripheral base address.
- submodule – Submodule number (0-3).

- pwmx – Pointer to the PWM_X configuration structure.

void FLEXPWM_GetDefaultForceOutConfig(*flexpwm_force_out_config_t* *config)

Get default configuration for force output.

This function initializes the force output configuration structure with safe default values. The default configuration can be used directly or modified before calling *FLEXPWM_ConfigForceOut()*.

Default values:

- FORCE_OUT trigger source: Local software force (kFLEXPWM_ForceOutputSource_LocalForce)
- Counter initialization on FORCE_OUT: Disabled (false)
- PWM_A source: Generated PWM signal (kFLEXPWM_ForcePwmSource_Generated)
- PWM_A software value: Logic 0 (false)
- PWM_B source: Generated PWM signal (kFLEXPWM_ForcePwmSource_Generated)
- PWM_B software value: Logic 0 (false)

Parameters

- config – Pointer to the force output configuration structure.

void FLEXPWM_ConfigForceOut(PWM_Type *base, uint8_t submodule, const *flexpwm_force_out_config_t* *config)

Configure FORCE_OUT for a FlexPWM submodule.

This function configures the FORCE_OUT event parameters for a FlexPWM submodule, including:

- FORCE_OUT trigger source (CTRL2[FORCE_SEL]): software, master, external, sync signals, etc.
- Counter initialization triggered by FORCE_OUT (CTRL2[FRCEN])
- PWM source selection for deadtime logic (DTSRCSEL[SMxSEL23/45]): generated, inverted, software-controlled, or external signals
- Software-controlled output values (SWCOUT[SMxOUT23/45]) when source is software-controlled

---

Note: Configuration write behavior:

- This function performs read-modify-write operations on global registers (DTSRCSEL, SWCOUT) to protect other submodules' configurations.
- Register writes are immediate and take effect when the corresponding FORCE_OUT event occurs.
- SWCOUT and DTSRCSEL are NOT affected by MCTRL[LDOK]; they are independent double-buffered registers that update on FORCE_OUT events.

Note: FORCE_OUT event effects: When a FORCE_OUT event occurs, the following updated values take effect simultaneously:

- PWM source routing from DTSRCSEL
- Software-controlled output values from SWCOUT
- Counter may be reinitialized if FRCEN is enabled

Note: This function does NOT:

- Trigger FORCE_OUT events (DTSRCSEL and SWCOUT settings require FORCE_OUT event to take effect)

Parameters

- base – FlexPWM peripheral base address.
- submodule – Submodule number (0-3).
- config – Pointer to the force output configuration structure.

void FLEXPWM_GetDefaultFaultConfig(*flexpwm_fault_config_t* *config)

Get default configuration for fault protection.

This function initializes the fault protection configuration structure with safe default values. The default configuration can be used directly or modified before calling *FLEXPWM_ConfigFaultProtection()*.

Default values:

- Fault Input Active Level (faultInputActiveLevel): Active high (true)
- Fault Clearing Mode (faultClearingMode): Automatic fault clearing
- Safety Mode (enableSafetyMode): Enabled (safe mode)
- Full Cycle Recovery (enableFullCycleRecovery): Enabled
- Half Cycle Recovery (enableHalfCycleRecovery): Disabled
- Combinational Path (enableCombinatorialPath): Enabled (allow combinational path for fastest response)

Parameters

- config – Pointer to the fault protection configuration structure.

void FLEXPWM_GetDefaultFaultSubmoduleConfig(*flexpwm_fault_submodule_config_t* *config)

Get default configuration for fault submodule protection.

This function initializes the fault submodule configuration structure with safe default values. The default configuration can be used directly or modified before calling *FLEXPWM_ConfigFaultSubmodule()*.

Default values:

- Disable Mask: All faults (0x0F) affect each output
- Output Behavior: High-Z (high impedance) for all outputs

For single-channel platforms: disableMask_ch0 = 0x0F For dual-channel platforms: disableMask_ch0 = 0x0F, disableMask_ch1 = 0x0F

Parameters

- config – Pointer to the fault submodule configuration structure.

void FLEXPWM_GetDefaultFaultFilterConfig(*flexpwm_fault_filter_config_t* *config)

Get default configuration for fault filter.

This function initializes the fault filter configuration structure with safe default values. The default configuration provides moderate debouncing suitable for typical applications.

Default values:

- Filter Period (FILT_PER): 0x08 (8 IPBus clock cycles)
- Filter Count (FILT_CNT): 0x04 (7 consecutive samples = 4+3)
- Glitch Stretching (GSTR): Enabled

Effective debounce window: ~56 IPBus clock cycles

Parameters

- config – Pointer to the fault filter configuration structure.

void FLEXPWM_ConfigFaultProtection(PWM_Type *base, uint8_t faultChannel, const *flexpwm_fault_config_t* *config, uint16_t faultMask)

Configure fault protection for a FlexPWM fault channel.

This function configures global fault protection parameters for one fault channel, including fault input active level, clearing mode, safety mode, and fault recovery timing.

Configuration details:

- Fault input active level (faultInputActiveLevel): Selects whether fault is triggered by high or low
- Fault clearing mode (faultClearingMode): Controls how and when PWM outputs are re-enabled
- Safety mode (enableSafetyMode): Adds extra requirement that fault input be de-asserted before re-enable
- Fault recovery timing (enableFullCycleRecovery, enableHalfCycleRecovery): Specifies during which PWM cycle phase outputs re-enable
- Combinational path (enableCombinatorialPath): Trades off response speed vs. latency

---

Note: Prerequisites:

- Call *FLEXPWM_Init()* before this function to enable the peripheral clock.

Note: Configuration applies to the entire fault channel (all submodules that reference it). For platforms with dual fault channels, call this function separately for each channel.

Note: For platforms with dual fault channels:

- faultChannel 0: Configures FAULT0-3 (FCTRL/FSTS/FFILT at offset 0x18C)
- faultChannel 1: Configures FAULT4-7 (FCTRL/FSTS/FFILT at offset 0x198)

Parameters

- base – FlexPWM peripheral base address.
- faultChannel – Fault channel number (0 or 1 for dual-channel platforms).
- config – Pointer to the fault protection configuration structure.
- faultMask – Fault input selection mask for the channel. Each bit corresponds to one fault input:
  - bit[0] = FAULT0 (or FAULT4 for channel 1)
  - bit[1] = FAULT1 (or FAULT5 for channel 1)

- bit[2] = FAULT2 (or FAULT6 for channel 1)
- bit[3] = FAULT3 (or FAULT7 for channel 1) Example: 0x05 = configure FAULT0 and FAULT2 with the same settings. Can use flexpwm_fault_mask_t enum values (kFLEXPWM_FaultMask_0/1/2/3) and OR them together for multiple faults: (kFLEXPWM_FaultMask_0 | kFLEXPWM_FaultMask_2) = 0x05.

void FLEXPWM_ConfigFaultSubmodule(PWM_Type *base, uint8_t submodule, const *flexpwm_fault_submodule_config_t* *config)

Configure fault submodule protection for a FlexPWM submodule.

This function configures fault-to-PWM-output mapping and output behavior for all three PWM outputs (PWM_A, PWM_B, PWM_X) of a single submodule. Each output can independently select which faults affect it and how it responds during a fault.

Configuration per output:

- Disable Mask: Selects which fault inputs can disable this output
  - disableMask_ch0: Bits [0-3] for FAULT0-3
  - disableMask_ch1: Bits [0-3] for FAULT4-7 (dual-channel platforms only)
- Output Behavior: Specifies pin state during fault (Force 0, Force 1, or High-Z)

Note: Configuration applies to SMxDISMAP and SMxOCTRL registers. These registers are not affected by LDOK or FORCE_OUT; configuration takes effect immediately upon register write.

Note: For dual-channel platforms, specify disableMask_ch0 and disableMask_ch1 to control which faults (from either channel) can disable each output.

Note: For single-channel platforms, only disableMask_ch0 is used; disableMask_ch1 is conditionally compiled out.

Parameters

- base – FlexPWM peripheral base address.
- submodule – Submodule number (0-3).
- config – Pointer to the fault submodule configuration structure.

void FLEXPWM_ConfigFaultFilter(PWM_Type *base, uint8_t faultChannel, const *flexpwm_fault_filter_config_t* *config)

Configure fault filter for a FlexPWM fault channel.

This function configures the input filter for fault protection pins, including sampling period, sample count requirement, and glitch stretching logic.

The filter helps reject electrical noise and false fault triggers by requiring the FAULTx signal to be stable for N consecutive samples before a fault is reported. With GSTR enabled, narrow fault glitches (< 2 IPBus cycles) are automatically stretched.

Note: Prerequisites:

- Call *FLEXPWM_Init()* before this function to enable the peripheral clock.

Note: Configuration applies to the entire fault channel (FFILT register). For platforms with dual fault channels, call this function separately for each channel.

Note: Filter delay calculation:

- Without GSTR: delay = FILT_PER * (FILT_CNT + 3) IPBus cycles
- With GSTR: may be reduced due to glitch stretching logic

Note: To disable filtering:

- Set filterPeriod = 0 (sampling disabled, filter bypassed)

Parameters

- base – FlexPWM peripheral base address.
- faultChannel – Fault channel number (0 or 1 for dual-channel platforms).
- config – Pointer to the fault filter configuration structure.

void FLEXPWM_GetDefaultOutputTriggerConfig(*flexpwm_output_trigger_config_t* *config)

Get default configuration for output trigger.

This function initializes the output trigger configuration structure with safe default values:

- outTriggerEnable: 0 (all triggers disabled)
- triggerFrequency: kFLEXPWM_TriggerFrequency_EveryCycle (trigger every PWM cycle)
- muxTrig0Source: kFLEXPWM_TriggerMuxSource_GeneratedTrigger (use PWM_OUT_TRIG0)
- muxTrig1Source: kFLEXPWM_TriggerMuxSource_GeneratedTrigger (use PWM_OUT_TRIG1)
- stretchPrescaler: kFLEXPWM_StretchPrescaler_None (no prescaling, if supported)

After calling this function, the user should modify the fields as needed before calling *FLEXPWM_ConfigOutputTrigger()*.

void FLEXPWM_ConfigOutputTrigger(PWM_Type *base, uint8_t submodule, const *flexpwm_output_trigger_config_t* *config)

Configure output trigger for a FlexPWM submodule.

This function configures the output trigger parameters for a FlexPWM submodule, including:

- Trigger source mapping from VAL0-5 register matches (SMxTCTRL[OUT_TRIG_EN])
- Trigger output frequency control (SMxTCTRL[TRGFRQ])
- PWM_MUX_TRIG0 port source selection (SMxTCTRL[PWAOT0])
- PWM_MUX_TRIG1 port source selection (SMxTCTRL[PWBOT1])

The output trigger mechanism allows PWM events (counter matches with VAL registers) to trigger external modules such as ADC, DMA, or other peripherals for synchronized operation.

Note: Trigger source mapping:

- VAL0, VAL2, VAL4 matches generate PWM_OUT_TRIG0
- VAL1, VAL3, VAL5 matches generate PWM_OUT_TRIG1 Multiple sources can be enabled simultaneously (OR logic).

Note: Trigger frequency control (TRGFRQ): This setting only takes effect when CTRL[LDFQ] is non-zero (reload frequency divider active).

- false: Trigger outputs every PWM cycle regardless of reload occurrence
- true: Trigger outputs only on the last PWM cycle before a reload opportunity

Note: Port routing: PWM_MUX_TRIG0/1 ports can be routed to either:

- Generated trigger signals (PWM_OUT_TRIG0/1) from VAL register matches
- Direct PWM outputs (PWM_A/PWM_B) for waveform monitoring Both ports are independently configurable.

Note: Configuration takes effect immediately upon register write. This function does NOT require setting MCTRL[LDOK].

Parameters

- base – FlexPWM peripheral base address.
- submodule – Submodule number (0-3).
- config – Pointer to the output trigger configuration structure.

void FLEXPWM_GetDefaultInputCaptureConfig(*flexpwm_input_capture_config_t* *config)

Get default configuration for input capture.

This function initializes the input capture configuration structure with safe default values. The default values are:

- All channels (A/B/X):
  - edge0 = kFLEXPWM_CaptureEdge_Disabled
  - edge1 = kFLEXPWM_CaptureEdge_Disabled
  - oneshot = false (free running mode)
  - inputSelect = kFLEXPWM_CaptureInput_RawSignal
  - edgeCompareValue = 0
  - fifoWatermark = kFLEXPWM_CaptureFifoWatermark_1

After calling this function, the user should modify the fields as needed before calling *FLEXPWM_ConfigInputCapture()*.

Parameters

- config – Pointer to the input capture configuration structure.

void FLEXPWM_GetDefaultCaptureFilterConfig(*flexpwm_capture_filter_config_t* *config)

Get default configuration for capture filter.

This function initializes the capture filter configuration structure with safe default values. The default values are:

- All channels (A/B/X):
  - filterPeriod = 0 (filter bypassed, sampling disabled)
  - filterCount = 0 (requires 3 consecutive samples)

After calling this function, the user should modify the fields as needed before calling *FLEXPWM_ConfigInputCaptureFilter()*.

Parameters

- config – Pointer to the capture filter configuration structure.

void FLEXPWM_ConfigInputCapture(PWM_Type *base, uint8_t submodule, const *flexpwm_input_capture_config_t* *config)

Configure input capture for a FlexPWM submodule.

This function configures the input capture parameters for up to three capture channels (Capture_A, Capture_B, Capture_X) of a FlexPWM submodule, including:

- Edge detection configuration for two edge detectors per channel (SMxCAPTCTRLA/B/X[EDGx0/1])
- One-shot or free-running mode (SMxCAPTCTRLA/B/X[ONESHOTx])
- Input source selection: raw signal or edge counter output (SMxCAPTCTRLA/B/X[INP_SELx])
- Edge counter compare value (SMxCAPTCOMPA/B/X[EDGCMPx])
- FIFO watermark level (SMxCAPTCTRLA/B/X[CFxWM])

Note: Edge counter behavior:

- When inputSelect = kFLEXPWM_CaptureInput_EdgeCounter, the edge counter is automatically enabled (EDGCNTx_EN = 1) by this function.
- When inputSelect = kFLEXPWM_CaptureInput_RawSignal, the edge counter remains disabled.

Note: Capture enable control:

- This function does NOT control the capture enable bits (ARMA/ARMB/ARMX).
- After calling this function, use a separate enable API to start capture operation.

Note: CVAL register mapping (NON-INTUITIVE):

- Capture_X: Edge0 -> CVAL0, Edge1 -> CVAL1
- Capture_A: Edge0 -> CVAL2, Edge1 -> CVAL3
- Capture_B: Edge0 -> CVAL4, Edge1 -> CVAL5

Note: Prerequisites:

- Call *FLEXPWM_Init()* before this function to enable the peripheral clock.

Parameters

- base – FlexPWM peripheral base address.
- submodule – Submodule number (0-3).
- config – Pointer to the input capture configuration structure.

void FLEXPWM_ConfigInputCaptureFilter(PWM_Type *base, uint8_t submodule, const *flexpwm_capture_filter_config_t* *config)

Configure input capture filter for a FlexPWM submodule.

This function configures the digital filter parameters for the three capture channels (Capture_A, Capture_B, Capture_X) of a FlexPWM submodule. The filter settings include:

- Filter sampling period (SMxCAPTFILTA/B/X[CAPTx_FILT_PER])
- Number of consecutive samples required (SMxCAPTFILTA/B/X[CAPTx_FILT_CNT])

The digital filter can eliminate glitches and noise on the capture input signals.

Note: Filter bypassing:

- Set filterPeriod = 0 to bypass the filter (no sampling, filter disabled).

Note: Filter delay calculation:

- Delay = filterPeriod * (filterCount + 3) IPBus clock cycles
- Example: filterPeriod=5, filterCount=2 -> delay = 5*(2+3) = 25 cycles

---

Note: Hardware behavior:

- This function always writes 0 to the filter register first, then writes the configuration value. This ensures correct behavior per hardware requirements.

Note: Prerequisites:

- Call *FLEXPWM_Init()* before this function to enable the peripheral clock.

Parameters

- base – FlexPWM peripheral base address.
- submodule – Submodule number (0-3).
- config – Pointer to the capture filter configuration structure.

uint16_t FLEXPWM_GetInputCaptureValue(PWM_Type *base, uint8_t submodule, *flexpwm_capture_index_t* captureIndex)

Read captured value from a FlexPWM capture channel.

This function reads the captured counter value from one of the six capture value registers (CVAL0-5) for the specified capture channel and edge detector.

Note: CVAL register mapping (NON-INTUITIVE):

- kFLEXPWM_Capture_X_Edge0 -> CVAL0
- kFLEXPWM_Capture_X_Edge1 -> CVAL1

- kFLEXPWM_Capture_A_Edge0 -> CVAL2
- kFLEXPWM_Capture_A_Edge1 -> CVAL3
- kFLEXPWM_Capture_B_Edge0 -> CVAL4
- kFLEXPWM_Capture_B_Edge1 -> CVAL5

Warning: Destructive read:

- Reading a CVAL register decrements the FIFO count by 1.
- If the FIFO is empty, the read value is undefined.
- Check the FIFO empty flag (CFx) before reading to avoid invalid data.

Parameters

- base – FlexPWM peripheral base address.
- submodule – Submodule number (0-3).
- captureIndex – Capture channel and edge detector index (use flexpwm_capture_index_t enum).

Returns

The captured counter value (16-bit).

uint16_t FLEXPWM_GetInputCaptureCycle(PWM_Type *base, uint8_t submodule, *flexpwm_capture_index_t* captureIndex)

Read captured cycle counter value from a FlexPWM capture channel.

This function reads the captured cycle counter value from one of the six capture cycle registers (CCYC0-5) for the specified capture channel and edge detector. The cycle counter value represents the upper part of the extended timestamp when a capture event occurs.

Note: CCYC register mapping (matches CVAL mapping):

- kFLEXPWM_Capture_X_Edge0 -> CCYC0
- kFLEXPWM_Capture_X_Edge1 -> CCYC1
- kFLEXPWM_Capture_A_Edge0 -> CCYC2
- kFLEXPWM_Capture_A_Edge1 -> CCYC3
- kFLEXPWM_Capture_B_Edge0 -> CCYC4
- kFLEXPWM_Capture_B_Edge1 -> CCYC5

Note: Combined timestamp:

- The full timestamp is {CCYC, CVAL} (cycle counter as upper 16 bits, capture value as lower 16 bits), forming a 32-bit extended timestamp.

Parameters

- base – FlexPWM peripheral base address.
- submodule – Submodule number (0-3).
- captureIndex – Capture channel and edge detector index (use flexpwm_capture_index_t enum).

Returns

The captured cycle counter value (16-bit).

void FLEXPWM_EnableInputCapture(PWM_Type *base, uint8_t submodule, *flexpwm_capture_channel_t* channel)

Enable input capture for a FlexPWM capture channel.

This function enables input capture operation for the specified capture channel by setting the corresponding ARM bit (ARMA/ARMB/ARMX) in the capture control register.

Note: ARM bit behavior in one-shot mode: When the capture channel is configured for one-shot mode (ONESHOT = 1), the ARM bit automatically clears after two capture events complete.

Note: Prerequisites: Before enabling capture, ensure:

a. Capture channel is configured (FLEXPWM_ConfigInputCapture)

b. Edge detectors are configured (EDGA0/EDGA1, EDGB0/EDGB1, or EDGX0/EDGX1)

c. Counter is running (FLEXPWM_EnableSubmoduleCounter)

Parameters

- base – FlexPWM peripheral base address.
- submodule – Submodule number (0-3).
- channel – Capture channel selection (use flexpwm_capture_channel_t enum).

void FLEXPWM_DisableInputCapture(PWM_Type *base, uint8_t submodule, *flexpwm_capture_channel_t* channel)

Disable input capture for a FlexPWM capture channel.

This function disables input capture operation for the specified capture channel by clearing the corresponding ARM bit (ARMA/ARMB/ARMX) in the capture control register.

Note: FIFO handling: Disabling capture does not flush the FIFO. Any previously captured values remain readable. Clear the capture flags (CFx0/CFx1) separately if needed.

Parameters

- base – FlexPWM peripheral base address.
- submodule – Submodule number (0-3).
- channel – Capture channel selection (use flexpwm_capture_channel_t enum).

static inline void FLEXPWM_SetINIT(PWM_Type *base, uint8_t submodule, uint16_t value)

Set the INIT register value for a FlexPWM submodule.

This function directly writes a value to the INIT register (initial count register). The INIT register defines the starting value of the counter when it is initialized.

Note: This function is intended for use in runtime scenarios such as interrupt handlers to dynamically adjust PWM parameters. It performs a direct register write without checking or managing MCTRL[LDOK]. Users are responsible for:

- Ensuring LDOK is clear before calling this function
- Setting LDOK after updating registers to trigger reload
- Managing reload timing and synchronization

Parameters

- base – FlexPWM peripheral base address.
- submodule – Submodule number (0-3).
- value – The value to write to the INIT register (16-bit).

static inline void FLEXPWM_SetVAL0(PWM_Type *base, uint8_t submodule, uint16_t value)

Set the VAL0 register value for a FlexPWM submodule.

This function directly writes a value to the VAL0 register. VAL0 defines the half-cycle reload point for the PWM counter.

Note: This function is intended for use in runtime scenarios such as interrupt handlers to dynamically adjust PWM parameters. It performs a direct register write without checking or managing MCTRL[LDOK]. Users are responsible for reload timing.

Parameters

- base – FlexPWM peripheral base address.
- submodule – Submodule number (0-3).
- value – The value to write to the VAL0 register (16-bit).

static inline void FLEXPWM_SetVAL1(PWM_Type *base, uint8_t submodule, uint16_t value)

Set the VAL1 register value for a FlexPWM submodule.

This function directly writes a value to the VAL1 register. VAL1 defines the modulo (period) value for the PWM counter.

Note: This function is intended for use in runtime scenarios such as interrupt handlers to dynamically adjust PWM period. It performs a direct register write without checking or managing MCTRL[LDOK]. Users are responsible for reload timing.

Parameters

- base – FlexPWM peripheral base address.
- submodule – Submodule number (0-3).
- value – The value to write to the VAL1 register (16-bit).

static inline void FLEXPWM_SetVAL2(PWM_Type *base, uint8_t submodule, uint16_t value)

Set the VAL2 register value for a FlexPWM submodule.

This function directly writes a value to the VAL2 register. VAL2 defines the count value at which PWM23 (PWM_A) is set high, controlling the turn-on edge of PWM_A output.

Note: This function is intended for use in runtime scenarios such as interrupt handlers to dynamically adjust PWM_A duty cycle. It performs a direct register write without checking or managing MCTRL[LDOK]. Users are responsible for reload timing.

Parameters

- base – FlexPWM peripheral base address.
- submodule – Submodule number (0-3).
- value – The value to write to the VAL2 register (16-bit).

static inline void FLEXPWM_SetVAL3(PWM_Type *base, uint8_t submodule, uint16_t value)

Set the VAL3 register value for a FlexPWM submodule.

This function directly writes a value to the VAL3 register. VAL3 defines the count value at which PWM23 (PWM_A) is set low, controlling the turn-off edge of PWM_A output.

Note: This function is intended for use in runtime scenarios such as interrupt handlers to dynamically adjust PWM_A duty cycle. It performs a direct register write without checking or managing MCTRL[LDOK]. Users are responsible for reload timing.

Parameters

- base – FlexPWM peripheral base address.
- submodule – Submodule number (0-3).
- value – The value to write to the VAL3 register (16-bit).

static inline void FLEXPWM_SetVAL4(PWM_Type *base, uint8_t submodule, uint16_t value)

Set the VAL4 register value for a FlexPWM submodule.

This function directly writes a value to the VAL4 register. VAL4 defines the count value at which PWM45 (PWM_B) is set high, controlling the turn-on edge of PWM_B output.

Note: This function is intended for use in runtime scenarios such as interrupt handlers to dynamically adjust PWM_B duty cycle. It performs a direct register write without checking or managing MCTRL[LDOK]. Users are responsible for reload timing.

Parameters

- base – FlexPWM peripheral base address.
- submodule – Submodule number (0-3).
- value – The value to write to the VAL4 register (16-bit).

static inline void FLEXPWM_SetVAL5(PWM_Type *base, uint8_t submodule, uint16_t value)

Set the VAL5 register value for a FlexPWM submodule.

This function directly writes a value to the VAL5 register. VAL5 defines the count value at which PWM45 (PWM_B) is set low, controlling the turn-off edge of PWM_B output.

Note: This function is intended for use in runtime scenarios such as interrupt handlers to dynamically adjust PWM_B duty cycle. It performs a direct register write without checking or managing MCTRL[LDOK]. Users are responsible for reload timing.

Parameters

- base – FlexPWM peripheral base address.
- submodule – Submodule number (0-3).
- value – The value to write to the VAL5 register (16-bit).

static inline void FLEXPWM_SetDTCNT0(PWM_Type *base, uint8_t submodule, uint16_t value)

Set the DTCNT0 register value for a FlexPWM submodule.

This function directly writes a value to the DTCNT0 register (deadtime count register 0). DTCNT0 specifies the number of IPBus clock cycles for deadtime delay insertion, independent of the PWM clock prescaler settings (CTRL[PRSC] and CTRL2[CLK_SEL]). DTCNT0 inserts software-selectable activation delays at PWM_A turn-on edge.

Note: DTCNT0 is an 11-bit register with valid range 0-0x7FF. Values exceeding this range will be masked to 11 bits.

Note: This function applies only to complementary PWM mode.

Parameters

- base – FlexPWM peripheral base address.
- submodule – Submodule number (0-3).
- value – The value to write to the DTCNT0 register (11-bit, range: 0-0x7FF).

static inline void FLEXPWM_SetDTCNT1(PWM_Type *base, uint8_t submodule, uint16_t value)

Set the DTCNT1 register value for a FlexPWM submodule.

This function directly writes a value to the DTCNT1 register (deadtime count register 1). DTCNT1 specifies the number of IPBus clock cycles for deadtime delay insertion, independent of the PWM clock prescaler settings (CTRL[PRSC] and CTRL2[CLK_SEL]). DTCNT1 inserts software-selectable activation delays at PWM_B turn-on edge.

Note: DTCNT1 is an 11-bit register with valid range 0-0x7FF. Values exceeding this range will be masked to 11 bits.

Note: This function applies only to complementary PWM mode.

Parameters

- base – FlexPWM peripheral base address.
- submodule – Submodule number (0-3).
- value – The value to write to the DTCNT1 register (11-bit, range: 0-0x7FF).

static inline void FLEXPWM_SetPhaseDelay(PWM_Type *base, uint8_t submodule, uint16_t value)

Set the Phase Delay register value for a FlexPWM submodule.

This function directly writes a value to the PHASEDLY register (phase delay register). PHASEDLY defines the time delay from the master sync signal (from submodule 0) to when this submodule recognizes the master sync, specified in PWM clock cycles.

Setting this register to a non-zero value and using the master sync signal as the initialization source allows this submodule's output to be delayed by a fixed number of cycles relative to submodule 0, enabling phase-shifted PWM outputs.

Note: PHASEDLY is only valid for submodules 1-3. Submodule 0 is the master sync source and cannot have a phase delay relative to itself.

Note: This function is intended for use in runtime scenarios such as interrupt handlers to dynamically adjust phase delay. It performs a direct register write without checking or managing MCTRL[LDOK]. Users are responsible for reload timing.

Parameters

- base – FlexPWM peripheral base address.
- submodule – Submodule number (1-3, must NOT be 0).
- value – The value to write to the PHASEDLY register (16-bit).

static inline void FLEXPWM_SetCompareMode(PWM_Type *base, uint8_t submodule, *flexpwm_compare_mode_t* compareMode)

Set Compare Mode for a FlexPWM submodule.

Parameters

- base – FlexPWM peripheral base address.
- submodule – Submodule number (0-3).
- compareMode – The compare mode to set for the submodule.

static inline void FLEXPWM_MaskPWMOutput(PWM_Type *base, uint16_t aMask, uint16_t bMask, uint16_t xMask)

Mask PWM_A, PWM_B, PWM_X outputs for specified submodules.

This function sets the MASK register bits to mask (disable) PWM outputs by forcing them to logic 0 before output polarity is applied. The mask takes effect when a FORCE_OUT event occurs on the corresponding submodule, unless immediate update is enabled.

Note: MASK register characteristics:

- Double-buffered register: writes take effect on next FORCE_OUT event
- Use *FLEXPWM_EnableUpdateMaskImmediately()* to bypass FORCE_OUT requirement
- Masking forces output to logic 0 before polarity inversion

Note: This function performs read-modify-write to preserve:

- Mask settings for other submodules not specified in parameters
- UPDATE_MASK bits (on platforms with FSL_FEATURE_PWM_MASK_HAS_UPDATE_MASK_BITFIELD)

Parameters

- base – FlexPWM peripheral base address.
- aMask – Submodule mask for PWM_A outputs (OR'ed combination of flexpwm_submodule_mask_t). bit[0]=SM0, bit[1]=SM1, bit[2]=SM2, bit[3]=SM3.
- bMask – Submodule mask for PWM_B outputs (OR'ed combination of flexpwm_submodule_mask_t). bit[0]=SM0, bit[1]=SM1, bit[2]=SM2, bit[3]=SM3.
- xMask – Submodule mask for PWM_X outputs (OR'ed combination of flexpwm_submodule_mask_t). bit[0]=SM0, bit[1]=SM1, bit[2]=SM2, bit[3]=SM3.

static inline void FLEXPWM_UnMaskPWMOutput(PWM_Type *base, uint16_t aMask, uint16_t bMask, uint16_t xMask)

Unmask PWM_A, PWM_B, PWM_X outputs for specified submodules.

This function clears the MASK register bits to unmask (enable) PWM outputs. The unmask takes effect when a FORCE_OUT event occurs on the corresponding submodule, unless immediate update is enabled.

Parameters

- base – FlexPWM peripheral base address.
- aMask – Submodule mask for PWM_A outputs (OR'ed combination of flexpwm_submodule_mask_t). bit[0]=SM0, bit[1]=SM1, bit[2]=SM2, bit[3]=SM3.
- bMask – Submodule mask for PWM_B outputs (OR'ed combination of flexpwm_submodule_mask_t). bit[0]=SM0, bit[1]=SM1, bit[2]=SM2, bit[3]=SM3.
- xMask – Submodule mask for PWM_X outputs (OR'ed combination of flexpwm_submodule_mask_t). bit[0]=SM0, bit[1]=SM1, bit[2]=SM2, bit[3]=SM3.

static inline void FLEXPWM_EnableUpdateMaskImmediately(PWM_Type *base, uint16_t submoduleMask)

Enable immediate MASK register update for specified submodules.

This function configures the specified submodules to update their MASK register values immediately, bypassing the FORCE_OUT event requirement. This is useful for emergency stop or fault protection scenarios where immediate output masking is required.

Note: MASK[UPDATE_MASK] register characteristics:

- Setting a bit to 1 enables immediate MASK update for that submodule
- Once enabled, MASK updates take effect immediately without waiting for FORCE_OUT

Parameters

- base – FlexPWM peripheral base address.
- submoduleMask – Submodule selection mask (OR'ed combination of flexpwm_submodule_mask_t). bit[0]=SM0, bit[1]=SM1, bit[2]=SM2, bit[3]=SM3.

static inline void FLEXPWM_SetPWMSource23(PWM_Type *base, uint8_t submodule, uint16_t source)

Set PWM source for PWM23 (PWM_A) of a submodule.

This function selects the signal source that feeds into the deadtime insertion logic for PWM23 (PWM_A) output by configuring the DTSRCSEL register. The setting takes effect when a FORCE_OUT event occurs on the submodule.

Available sources:

- Generated PWM: Normal PWM signal from PWM generator
- Inverted PWM: Inverted version of generated PWM
- Software: Software-controlled value from SWCOUT register
- External: External input signal

Note: DTSRCSEL register characteristics:

- Double-buffered: Changes take effect on next FORCE_OUT event
- NOT affected by MCTRL[LDOK]: Independent of reload mechanism
- Global register: This function performs read-modify-write to protect other submodules

---

Note: When source is set to kFLEXPWM_ForcePwmSource_Software, use *FLEXPWM_SetSoftwareControlOutput23()* to set the output value.

Parameters

- base – FlexPWM peripheral base address.
- submodule – Submodule number (0-3).
- source – PWM source selection (use flexpwm_force_pwm_source_t enum).

static inline void FLEXPWM_SetPWMSource45(PWM_Type *base, uint8_t submodule, uint16_t source)

Set PWM source for PWM45 (PWM_B) of a submodule.

This function selects the signal source that feeds into the deadtime insertion logic for PWM45 (PWM_B) output by configuring the DTSRCSEL register. The setting takes effect when a FORCE_OUT event occurs on the submodule.

Note: See *FLEXPWM_SetPWMSource23()* documentation for detailed information about source selection, register characteristics, and usage notes.

Parameters

- base – FlexPWM peripheral base address.
- submodule – Submodule number (0-3).
- source – PWM source selection (use flexpwm_force_pwm_source_t enum).

static inline void FLEXPWM_SetSoftwareControlOutput23(PWM_Type *base, uint8_t submodule, uint16_t softwareValue)

Set software-controlled output value for PWM23 (PWM_A) of a submodule.

This function sets the software-controlled PWM output value in the SWCOUT register. This value is used as the PWM source when DTSRCSEL is configured to select software control (kFLEXPWM_ForcePwmSource_Software). The setting takes effect when a FORCE_OUT event occurs on the submodule.

Note: SWCOUT register characteristics:

- Double-buffered: Changes take effect on next FORCE_OUT event
- NOT affected by MCTRL[LDOK]: Independent of reload mechanism
- Global register: This function performs read-modify-write to protect other submodules

---

Note: Typical usage sequence:

a. Call FLEXPWM_SetPWMSource23(base, kFLEXPWM_ForcePwmSource_Software, submodule)

b. Call FLEXPWM_SetSoftwareControlOutput23(base, value, submodule)

c. Trigger FORCE_OUT event to make both settings take effect

Parameters

- base – FlexPWM peripheral base address.
- submodule – Submodule number (0-3).
- softwareValue – Software control output value (0 or 1).
  - 0: Logic 0 provided to deadtime logic instead of PWM23
  - 1: Logic 1 provided to deadtime logic instead of PWM23

static inline void FLEXPWM_SetSoftwareControlOutput45(PWM_Type *base, uint8_t submodule, uint16_t softwareValue)

Set software-controlled output value for PWM45 (PWM_B) of a submodule.

This function sets the software-controlled PWM output value in the SWCOUT register. This value is used as the PWM source when DTSRCSEL is configured to select software control (kFLEXPWM_ForcePwmSource_Software).

Note: See *FLEXPWM_SetSoftwareControlOutput23()* documentation for detailed information about register characteristics, usage sequence, and notes.

Parameters

- base – FlexPWM peripheral base address.
- submodule – Submodule number (0-3).
- softwareValue – Software control output value (0 or 1).
  - 0: Logic 0 provided to deadtime logic instead of PWM45
  - 1: Logic 1 provided to deadtime logic instead of PWM45

static inline void FLEXPWM_SetComplementaryPWMSource23(PWM_Type *base, uint16_t submoduleMask)

Select PWM23 (PWM_A) as complementary PWM source for specified submodules.

This function configures the MCTRL[IPOL] register to select PWM23 (PWM_A) as the source for generating complementary PWM pair outputs in complementary mode. The setting takes effect when a FORCE_OUT event occurs on the submodule.

In complementary mode:

- PWM_A output: PWM23 signal (non-inverted)
- PWM_B output: ~PWM23 signal (inverted complement)

Note: MCTRL[IPOL] register characteristics:

- Takes effect on FORCE_OUT event (double-buffered behavior)
- Only applies when submodule is in complementary mode (INDEP = 0)
- Ignored in independent mode (INDEP = 1)

Note: This function clears IPOL bits for specified submodules (IPOL = 0 selects PWM23).

Parameters

- base – FlexPWM peripheral base address.
- submoduleMask – Submodule selection mask (OR'ed combination of flexpwm_submodule_mask_t). bit[0]=SM0, bit[1]=SM1, bit[2]=SM2, bit[3]=SM3.

static inline void FLEXPWM_SetComplementaryPWMSource45(PWM_Type *base, uint16_t submoduleMask)

Select PWM45 (PWM_B) as complementary PWM source for specified submodules.

This function configures the MCTRL[IPOL] register to select PWM45 (PWM_B) as the source for generating complementary PWM pair outputs in complementary mode. The setting takes effect when a FORCE_OUT event occurs on the submodule.

In complementary mode:

- PWM_A output: ~PWM45 signal (inverted complement)
- PWM_B output: PWM45 signal (non-inverted)

Note: See *FLEXPWM_SetComplementaryPWMSource23()* documentation for detailed information about MCTRL[IPOL] characteristics and complementary mode operation.

Note: This function sets IPOL bits for specified submodules (IPOL = 1 selects PWM45).

Parameters

- base – FlexPWM peripheral base address.
- submoduleMask – Submodule selection mask (OR'ed combination of flexpwm_submodule_mask_t). bit[0]=SM0, bit[1]=SM1, bit[2]=SM2, bit[3]=SM3.

static inline void FLEXPWM_SetLoadOkay(PWM_Type *base, uint16_t submoduleMask)

Set Load Okay (LDOK) for specified submodules.

This function sets the MCTRL[LDOK] bits to enable register reload for the specified submodules. When LDOK is set, buffered register values will be transferred to active registers according to the reload mode and timing configured in CTRL[LDMOD, LDFQ, HALF, FULL].

Note: LDOK behavior:

- Can only be set when the corresponding LDOK bit is currently 0
- Automatically cleared by hardware after reload completes
- Can be manually cleared using *FLEXPWM_ClearLoadOkay()*
- While LDOK=1, writes to buffered registers are blocked

Note: Typical usage flow:

a. Modify buffered registers (VAL0-5, INIT, etc.) using FLEXPWM_SetVALx() or similar

b. Call *FLEXPWM_SetLoadOkay()* to trigger reload

c. Hardware transfers buffered values at the configured reload opportunity

d. LDOK automatically clears after reload completes

Parameters

- base – FlexPWM peripheral base address.
- submoduleMask – Submodule selection mask (OR'ed combination of flexpwm_submodule_mask_t). bit[0]=SM0, bit[1]=SM1, bit[2]=SM2, bit[3]=SM3.

static inline void FLEXPWM_ClearLoadOkay(PWM_Type *base, uint16_t submoduleMask)

Clear Load Okay (LDOK) for specified submodules.

This function clears the MCTRL[LDOK] bits by writing to MCTRL[CLDOK] bits. This cancels a pending reload operation for the specified submodules.

Note: MCTRL[CLDOK] (Clear LDOK) mechanism:

- Writing 1 to CLDOK[x] clears the corresponding LDOK[x] bit
- CLDOK is write-only; reading always returns 0
- Use this when you need to cancel a pending reload

Parameters

- base – FlexPWM peripheral base address.
- submoduleMask – Submodule selection mask (OR'ed combination of flexpwm_submodule_mask_t). bit[0]=SM0, bit[1]=SM1, bit[2]=SM2, bit[3]=SM3.

static inline uint16_t FLEXPWM_GetLoadOkay(PWM_Type *base)

Get Load Okay (LDOK) status.

This function reads the MCTRL[LDOK] bits to check which submodules have pending reload operations.

Note: Return value format:

- Bit[0]: Submodule 0 LDOK status
- Bit[1]: Submodule 1 LDOK status
- Bit[2]: Submodule 2 LDOK status
- Bit[3]: Submodule 3 LDOK status

Note: Usage example:

```
uint16_t ldokStatus = FLEXPWM_GetLoadOkay(PWM0);
if (ldokStatus & kFLEXPWM_SubmoduleMask_0) {
}
```

Parameters

- base – FlexPWM peripheral base address.

Returns

LDOK status bits (bit[3:0] correspond to submodules 3-0). Use flexpwm_submodule_mask_t enum values to check specific submodules.

static inline void FLEXPWM_SetLocalForceOut(PWM_Type *base, uint8_t submodule)

Set local FORCE_OUT event for a submodule.

This function sets the SMxCTRL2[FORCE] bit to trigger a local FORCE_OUT event. When a FORCE_OUT event occurs, several double-buffered registers immediately update:

- DTSRCSEL (PWM source selection)
- SWCOUT (software-controlled output values)
- MASK (output masking)
- MCTRL[IPOL] (complementary PWM source selection)

Additionally, if CTRL2[FRCEN] is enabled and the counter is running, the counter will be initialized to the INIT register value.

Note: FORCE_OUT trigger source: This function triggers a local FORCE_OUT event by setting CTRL2[FORCE]. The FORCE_OUT event only occurs if CTRL2[FORCE_SEL] = 0 (local force selected). If FORCE_SEL selects a different source, setting FORCE has no effect.

Note: Counter initialization: The counter is reinitialized only if:

- CTRL2[FRCEN] = 1 (force initialization enabled), AND
- MCTRL[RUN] = 1 (counter running) OR CTRL2[CLK_SEL] = 2 (submodule 0 aux clock)

Parameters

- base – FlexPWM peripheral base address.
- submodule – Submodule number (0-3).

static inline void FLEXPWM_SetCounterForceInitialization(PWM_Type *base, uint8_t submodule, bool enable)

Enable or disable counter initialization on FORCE_OUT event.

This function controls whether FORCE_OUT events can trigger counter initialization by configuring the CTRL2[FRCEN] bit. When enabled, the counter will be reset to the INIT register value when local software forced event occurs.

Note: Counter initialization conditions: The counter is reinitialized only when ALL of the following conditions are met:

- CTRL2[FRCEN] = 1 (enabled via this function), AND
- A local FORCE_OUT event occurs (CTRL2[FORCE_SEL] = 0, CTRL2[FORCE] = 1)

Parameters

- base – FlexPWM peripheral base address.
- submodule – Submodule number (0-3).
- enable – Enable or disable counter force initialization:
  - true: FORCE_OUT events can trigger counter initialization (FRCEN = 1)
  - false: FORCE_OUT events do not affect counter initialization (FRCEN = 0)

static inline void FLEXPWM_EnableSubmoduleCounter(PWM_Type *base, uint16_t submoduleMask)

Enable PWM counter for specified submodules.

This function enables the PWM counters for the specified submodules by setting MCTRL[RUN] bits. When enabled, the counters start running and generating PWM waveforms.

Note: Effects of enabling counter (RUN=1):

- Counter begins incrementing/decrementing according to configuration
- PWM generation starts (if outputs are also enabled)
- Reload events can occur at configured opportunities
- FORCE_OUT can trigger counter initialization (if FRCEN enabled)

Parameters

- base – FlexPWM peripheral base address.
- submoduleMask – Submodule selection mask (OR'ed combination of flexpwm_submodule_mask_t). bit[0]=SM0, bit[1]=SM1, bit[2]=SM2, bit[3]=SM3.

static inline void FLEXPWM_DisableSubmoduleCounter(PWM_Type *base, uint16_t submoduleMask)

Disable PWM counter for specified submodules.

This function disables the PWM counters for the specified submodules by clearing MCTRL[RUN] bits. When disabled, the counters stop running but retain their configuration.

Note: Effects of disabling counter (RUN=0):

- Counter stops incrementing/decrementing
- PWM generation ceases (outputs hold last state or go to fault state)
- Configuration registers remain intact
- Fault clearing behavior changes (see fault protection documentation)

Parameters

- base – FlexPWM peripheral base address.
- submoduleMask – Submodule selection mask (OR'ed combination of flexpwm_submodule_mask_t). bit[0]=SM0, bit[1]=SM1, bit[2]=SM2, bit[3]=SM3.

static inline void FLEXPWM_EnablePWMOutput(PWM_Type *base, uint16_t aMask, uint16_t bMask, uint16_t xMask)

Enable PWM_A, PWM_B, PWM_X outputs for specified submodules.

This function enables PWM outputs by setting the corresponding bits in the OUTEN register. Each output (PWM_A, PWM_B, PWM_X) can be independently enabled for each submodule.

Note: OUTEN register layout (same as MASK register):

- bit[11:8]: PWMA_EN - PWM_A output enable for submodules 3-0
- bit[7:4]: PWMB_EN - PWM_B output enable for submodules 3-0
- bit[3:0]: PWMX_EN - PWM_X output enable for submodules 3-0

Note: Prerequisites: Before enabling outputs, ensure:

a. PWM waveform is configured (FLEXPWM_ConfigPWM)

b. Counter is running (FLEXPWM_EnableSubmoduleCounter)

c. Outputs are not masked (check MASK register)

Parameters

- base – FlexPWM peripheral base address.
- aMask – Submodule mask for PWM_A outputs (OR'ed combination of flexpwm_submodule_mask_t). bit[0]=SM0, bit[1]=SM1, bit[2]=SM2, bit[3]=SM3.
- bMask – Submodule mask for PWM_B outputs (OR'ed combination of flexpwm_submodule_mask_t). bit[0]=SM0, bit[1]=SM1, bit[2]=SM2, bit[3]=SM3.
- xMask – Submodule mask for PWM_X outputs (OR'ed combination of flexpwm_submodule_mask_t). bit[0]=SM0, bit[1]=SM1, bit[2]=SM2, bit[3]=SM3.

static inline void FLEXPWM_DisablePWMOutput(PWM_Type *base, uint16_t aMask, uint16_t bMask, uint16_t xMask)

Disable PWM_A, PWM_B, PWM_X outputs for specified submodules.

This function disables PWM outputs by clearing the corresponding bits in the OUTEN register. When an output is disabled, the pin is released from FlexPWM control and may be controlled by other peripherals or GPIO.

Note: Output disable effects:

- Pin is tri-stated or controlled by GPIO/other peripherals
- PWM waveform generation continues internally
- Re-enabling the output will resume PWM operation seamlessly

Parameters

- base – FlexPWM peripheral base address.
- aMask – Submodule mask for PWM_A outputs (OR'ed combination of flexpwm_submodule_mask_t). bit[0]=SM0, bit[1]=SM1, bit[2]=SM2, bit[3]=SM3.
- bMask – Submodule mask for PWM_B outputs (OR'ed combination of flexpwm_submodule_mask_t). bit[0]=SM0, bit[1]=SM1, bit[2]=SM2, bit[3]=SM3.
- xMask – Submodule mask for PWM_X outputs (OR'ed combination of flexpwm_submodule_mask_t). bit[0]=SM0, bit[1]=SM1, bit[2]=SM2, bit[3]=SM3.

static inline void FLEXPWM_SetWriteProtection(PWM_Type *base, *flexpwm_write_protect_t* config)

Set FlexPWM write protection mode.

This function configures the write protection for FlexPWM registers by setting MCTRL2[WRPROT]. Write protection prevents accidental modification of critical PWM configuration registers.

Note: Write protection modes:

- kFLEXPWM_WriteProtect_Off: Write protection is disabled (default after reset)
- kFLEXPWM_WriteProtect_On: Write protection is enabled but can be disabled
- kFLEXPWM_WriteProtect_OffLocked: Write protection is disabled and locked until chip reset
- kFLEXPWM_WriteProtect_OnLocked: Write protection is enabled and locked until chip reset

Note: Once a locked mode is set (OffLocked or OnLocked), the write protection state cannot be changed until the next chip reset. Use locked modes for safety-critical applications.

Parameters

- base – FlexPWM peripheral base address.
- config – Write protection mode selection.

static inline void FLEXPWM_EnableSubmoduleInterrupts(PWM_Type *base, uint8_t submodule, uint16_t mask)

Enable FlexPWM submodule interrupts.

This function enables the specified interrupt sources for a FlexPWM submodule by setting the corresponding bits in the SMxINTEN register.

Parameters

- base – FlexPWM peripheral base address.
- submodule – Submodule number (0-3).
- mask – Interrupt enable mask (OR'ed combination of _flexpwm_submodule_interrupt_enable flags).

static inline void FLEXPWM_DisableSubmoduleInterrupts(PWM_Type *base, uint8_t submodule, uint16_t mask)

Disable FlexPWM submodule interrupts.

This function disables the specified interrupt sources for a FlexPWM submodule by clearing the corresponding bits in the SMxINTEN register.

Parameters

- base – FlexPWM peripheral base address.
- submodule – Submodule number (0-3).
- mask – Interrupt disable mask (OR'ed combination of _flexpwm_submodule_interrupt_enable flags).

static inline uint16_t FLEXPWM_GetSubmoduleStatusFlags(PWM_Type *base, uint8_t submodule)

Get FlexPWM submodule status flags.

This function returns the current status flags from the SMxSTS register.

Parameters

- base – FlexPWM peripheral base address.
- submodule – Submodule number (0-3).

Returns

Current status flags (combination of _flexpwm_submodule_status_flag values).

static inline void FLEXPWM_ClearSubmoduleStatusFlags(PWM_Type *base, uint8_t submodule, uint16_t mask)

Clear FlexPWM submodule status flags.

This function clears the specified status flags in the SMxSTS register using the write-1-to-clear (W1C) mechanism.

Note: Do not attempt to clear kFLEXPWM_RegisterUpdatedFlag (RUF) as it is read-only.

Parameters

- base – FlexPWM peripheral base address.
- submodule – Submodule number (0-3).
- mask – Status flag clear mask (OR'ed combination of _flexpwm_submodule_status_flag values). To clear a flag, set the corresponding bit to 1 in the mask.

static inline void FLEXPWM_EnableFaultInterrupts(PWM_Type *base, uint8_t faultChannel, uint16_t mask)

Enable FlexPWM fault interrupts.

This function enables the specified fault interrupt sources by setting the corresponding bits in the FCTRL[FIE] field.

Note: For dual-channel platforms (FSL_FEATURE_PWM_FAULT_CH_COUNT > 1): The same mask bit values are used for both fault channels. The faultChannel parameter determines which set of fault inputs is controlled:

- faultChannel = 0: Controls FAULT0-3 interrupts
- faultChannel = 1: Controls FAULT4-7 interrupts

Parameters

- base – FlexPWM peripheral base address.
- faultChannel – Fault channel number:
  - 0: Fault channel 0 (FAULT0-3)
  - 1: Fault channel 1 (FAULT4-7, if supported)
- mask – Fault interrupt enable mask (OR'ed combination of _flexpwm_fault_interrupt_enable flags).

static inline void FLEXPWM_DisableFaultInterrupts(PWM_Type *base, uint8_t faultChannel, uint16_t mask)

Disable FlexPWM fault interrupts.

This function disables the specified fault interrupt sources by clearing the corresponding bits in the FCTRL[FIE] field.

Parameters

- base – FlexPWM peripheral base address.
- faultChannel – Fault channel number:
  - 0: Fault channel 0 (FAULT0-3)
  - 1: Fault channel 1 (FAULT4-7, if supported)
- mask – Fault interrupt disable mask (OR'ed combination of _flexpwm_fault_interrupt_enable flags).

static inline uint16_t FLEXPWM_GetFaultStatusFlags(PWM_Type *base, uint8_t faultChannel)

Get FlexPWM fault status flags.

This function returns the current fault status flags (FFLAG bits) from the FSTS register. These are latched flags that indicate a fault event has occurred.

Note: To check real-time hardware pin status (rather than latched flags), use *FLEXPWM_GetFaultFilteredPinStatusFlags()* instead.

Parameters

- base – FlexPWM peripheral base address.
- faultChannel – Fault channel number:
  - 0: Fault channel 0 (FAULT0-3)
  - 1: Fault channel 1 (FAULT4-7, if supported)

Returns

Current fault flags (combination of _flexpwm_fault_status_flag values).

static inline void FLEXPWM_ClearFaultStatusFlags(PWM_Type *base, uint8_t faultChannel, uint16_t mask)

Clear FlexPWM fault status flags.

This function clears the specified fault status flags (FFLAG bits) in the FSTS register using the write-1-to-clear (W1C) mechanism.

Note: Clearing FFLAG does not automatically re-enable PWM outputs. The re-enable timing depends on fault clearing mode (FAUTO), safety mode (FSAFE), recovery timing (FHALF/FFULL), and the state of FFPIN (filtered pin status).

Parameters

- base – FlexPWM peripheral base address.
- faultChannel – Fault channel number:
  - 0: Fault channel 0 (FAULT0-3)
  - 1: Fault channel 1 (FAULT4-7, if supported)
- mask – Fault flag clear mask (OR'ed combination of _flexpwm_fault_status_flag values). To clear a flag, set the corresponding bit to 1 in the mask.

static inline uint16_t FLEXPWM_GetFaultFilteredPinStatusFlags(PWM_Type *base, uint8_t faultChannel)

Get FlexPWM filtered fault pin status.

This function returns the real-time status of fault pins after filtering (FFPIN bits) from the FSTS register. These are read-only hardware status bits that reflect the current state of external fault signals.

Note: Key differences:

- FFLAG: Software-clearable latched flag (historical event - "a fault occurred")
- FFPIN: Read-only real-time status (current state - "fault is active now")

Parameters

- base – FlexPWM peripheral base address.
- faultChannel – Fault channel number:
  - 0: Fault channel 0 (FAULT0-3 pins)
  - 1: Fault channel 1 (FAULT4-7 pins, if supported)

Returns

Current filtered fault pin status (combination of _flexpwm_fault_pin_status values).

void FLEXPWM_ConfigDMA(PWM_Type *base, uint8_t submodule, const *flexpwm_dma_config_t* *config)

Configures the DMA control strategy for a FlexPWM submodule.

This function configures the DMA trigger source and FIFO watermark logic. These settings define the DMA request generation strategy and should typically be configured once during initialization.

Note: This function only configures CAPTDE[7:6] and FAND bits in SMxDMAEN register. To enable/disable individual DMA channels at runtime, use *FLEXPWM_EnableDMA()* and *FLEXPWM_DisableDMA()*.

Note: Prerequisites: Before calling this function, ensure:

a. FIFO watermark levels are configured in *FLEXPWM_ConfigInputCapture()*

b. Capture channels are properly configured if using capture DMA

Parameters

- base – FlexPWM peripheral base address
- submodule – FlexPWM submodule number (0-3)
- config – Pointer to DMA configuration structure

static inline void FLEXPWM_EnableDMA(PWM_Type *base, uint8_t submodule, uint16_t mask)

Enable FlexPWM submodule DMA requests.

This function enables DMA request generation for the specified channels. It controls the VALDE bit (VALx/FRACVALx write) and capture channel enables (CX0DE, CX1DE, CB0DE, CB1DE, CA0DE, CA1DE).

Note: Before calling this function, ensure *FLEXPWM_ConfigDMA()* has been called to set up the DMA trigger strategy.

Parameters

- base – FlexPWM peripheral base address
- submodule – FlexPWM submodule number (0-3)
- mask – Bit mask of DMA enable flags (OR'ed combination of _flexpwm_dma_enable)

static inline void FLEXPWM_DisableDMA(PWM_Type *base, uint8_t submodule, uint16_t mask)

Disable FlexPWM submodule DMA requests.

This function disables DMA request generation for the specified channels.

Parameters

- base – FlexPWM peripheral base address
- submodule – FlexPWM submodule number (0-3)
- mask – Bit mask of DMA enable flags (OR'ed combination of _flexpwm_dma_enable)

FSL_FLEXPWM_DRIVER_VERSION

Version 2.0.0

enum _flexpwm_clock_source

FlexPWM clock source selection.

Values:

enumerator kFLEXPWM_ClockSource_IPBusClock
IPBus clock

enumerator kFLEXPWM_ClockSource_ExtClock
External clock (EXT_CLK)

enumerator kFLEXPWM_ClockSource_Submodule0Clock
Submodule 0 auxiliary clock

enum _flexpwm_prescaler
FlexPWM clock prescaler selection.
Values:

enumerator kFLEXPWM_Prescale_Divide_1
Divide by 1

enumerator kFLEXPWM_Prescale_Divide_2
Divide by 2

enumerator kFLEXPWM_Prescale_Divide_4
Divide by 4

enumerator kFLEXPWM_Prescale_Divide_8
Divide by 8

enumerator kFLEXPWM_Prescale_Divide_16
Divide by 16

enumerator kFLEXPWM_Prescale_Divide_32
Divide by 32

enumerator kFLEXPWM_Prescale_Divide_64
Divide by 64

enumerator kFLEXPWM_Prescale_Divide_128
Divide by 128

enum _flexpwm_load_mode
FlexPWM load mode selection.
Values:

enumerator kFLEXPWM_LoadMode_Opportunity
Load at PWM reload opportunity

enumerator kFLEXPWM_LoadMode_Immediate
Load immediately after LDOK is set

enum _flexpwm_reload_source

FlexPWM reload source selection.

Values:

enumerator kFLEXPWM_ReloadSource_LocalReload

Local reload signal

enumerator kFLEXPWM_ReloadSource_MasterReload

Master reload signal from submodule 0

enum _flexpwm_init_source

FlexPWM counter initialization source selection.

Values:

enumerator kFLEXPWM_InitSource_LocalSync

Local sync signal

enumerator kFLEXPWM_InitSource_MasterReload

Master reload signal from submodule 0

enumerator kFLEXPWM_InitSource_MasterSync

Master sync signal from submodule 0

enumerator kFLEXPWM_InitSource_ExtSync

External sync signal (EXT_SYNC)

enum _flexpwm_compare_mode

FlexPWM PWM compare mode selection.

Values:

enumerator kFLEXPWM_CompareMode_Equal

Compare when counter equals compare value

enumerator kFLEXPWM_CompareMode_EqualGreater

Compare when counter is equal or greater than compare value

enum _flexpwm_submodule_mask

FlexPWM submodule mask for multi-submodule operations.

These flags can be OR'ed together to specify multiple submodules simultaneously. Used with functions that operate on multiple submodules at once.

Values:

enumerator kFLEXPWM_SubmoduleMask_0

Submodule 0

enumerator kFLEXPWM_SubmoduleMask_1

Submodule 1

enumerator kFLEXPWM_SubmoduleMask_2
Submodule 2

enum _flexpwm_pwm_polarity
FlexPWM PWM output polarity selection.
Values:

enumerator kFLEXPWM_Polarity_ActiveHigh
PWM signal is normal (active high)

enumerator kFLEXPWM_Polarity_ActiveLow
PWM signal is inverted (active low)

enum _flexpwm_ipol_source
FlexPWM PWM output source selection for complementary mode.
Values:

enumerator kFLEXPWM_IPOL_PWM23
Use PWM23 (PWM_A) as complementary PWM source

enumerator kFLEXPWM_IPOL_PWM45
Use PWM45 (PWM_B) as complementary PWM source

enum _flexpwm_force_output_source
FlexPWM FORCE_OUT event source selection.
Values:

enumerator kFLEXPWM_ForceOutputSource_LocalForce
Local software forced event

enumerator kFLEXPWM_ForceOutputSource_MasterForce
Master force signal from submodule 0

enumerator kFLEXPWM_ForceOutputSource_LocalReload
Local reload signal

enumerator kFLEXPWM_ForceOutputSource_MasterReload
Master reload signal from submodule 0

enumerator kFLEXPWM_ForceOutputSource_LocalSync
Local sync signal

enumerator kFLEXPWM_ForceOutputSource_MasterSync
Master sync signal from submodule 0

enumerator kFLEXPWM_ForceOutputSource_ExtForce
External force signal (EXT_FORCE)

enumerator kFLEXPWM_ForceOutputSource_ExtSync

External sync signal (EXT_SYNC)

enum _flexpwm_force_pwm_source

FlexPWM PWM source selection for deadtime logic (DTSRCSEL)

Values:

enumerator kFLEXPWM_ForcePwmSource_Generated

Generated PWM signal

enumerator kFLEXPWM_ForcePwmSource_Inverted

Inverted PWM signal

enumerator kFLEXPWM_ForcePwmSource_Software

Software controlled output (SWCOUT)

enumerator kFLEXPWM_ForcePwmSource_External

External signal

enum _flexpwm_fault_output_behavior

FlexPWM fault output behavior during fault condition.

Defines how PWM outputs respond when a fault is detected. Values map directly to SMxOCTRL[PWMAFS/PWMBFS/PWMXFS] bitfields.

Values:

enumerator kFLEXPWM_FaultOutput_Force0

0b00 - Force output to logic 0

enumerator kFLEXPWM_FaultOutput_Force1

0b01 - Force output to logic 1

enumerator kFLEXPWM_FaultOutput_HighZ

0b10/0b11 - High impedance (3-state) output

enum _flexpwm_fault_mask

FlexPWM fault input mask for fault protection configuration.

These flags can be OR'ed together to configure multiple fault inputs simultaneously with the same protection parameters. Used with *FLEXPWM_ConfigFaultProtection()* function.

Note: For dual-channel platforms (FSL_FEATURE_PWM_FAULT_CH_COUNT > 1):

- When faultChannel = 0: Use kFLEXPWM_FaultMask_0/1/2/3 for FAULT0-3
- When faultChannel = 1: Use kFLEXPWM_FaultMask_0/1/2/3 for FAULT4-7 The enum values are the same, but the channel parameter determines which physical fault inputs are configured.

Values:

enumerator kFLEXPWM_FaultMask_0
FAULT0 (ch0) or FAULT4 (ch1)

enumerator kFLEXPWM_FaultMask_1
FAULT1 (ch0) or FAULT5 (ch1)

enumerator kFLEXPWM_FaultMask_2
FAULT2 (ch0) or FAULT6 (ch1)

enumerator kFLEXPWM_FaultMask_3
FAULT3 (ch0) or FAULT7 (ch1)

enum _flexpwm_fault_clearing_mode
FlexPWM fault clearing mode selection.

Defines how fault conditions are cleared after a fault event.

Values:

enumerator kFLEXPWM_FaultClearingMode_Manual
Manual clearing - software must clear FSTS[FFLAGx]

enumerator kFLEXPWM_FaultClearingMode_Automatic
Automatic clearing when FAULTx de-asserts

enum _flexpwm_write_protect
FlexPWM write protection mode selection.

Values:

enumerator kFLEXPWM_WriteProtect_Off
Write protection off (default)

enumerator kFLEXPWM_WriteProtect_On
Write protection on

enumerator kFLEXPWM_WriteProtect_OffLocked
Write protection off and locked until chip reset

enumerator kFLEXPWM_WriteProtect_OnLocked
Write protection on and locked until chip reset

enum _flexpwm_stretch_prescaler
FlexPWM trigger signal stretch prescaler selection.

Stretch IPBus clock count prescaler for trigger signals including: mux0_trig, mux1_trig, out0_trig, out1_trig, pwma_trig, pwmb_trig

Values:

enumerator kFLEXPWM_StretchPrescaler_None
No stretch

enumerator kFLEXPWM_StretchPrescaler_2Clocks
Stretch for 2 IPBus clock periods

enumerator kFLEXPWM_StretchPrescaler_4Clocks
Stretch for 4 IPBus clock periods

enumerator kFLEXPWM_StretchPrescaler_8Clocks
Stretch for 8 IPBus clock periods

enum _flexpwm_trigger_mux_source
FlexPWM output trigger source selection for PWM_MUX_TRIG0/1 ports.

Selects which signal is routed to the PWM_MUX_TRIG0/1 output ports.

Values:

enumerator kFLEXPWM_TriggerMuxSource_GeneratedTrigger
Use PWM_OUT_TRIG0/1 (generated trigger signal)

enumerator kFLEXPWM_TriggerMuxSource_PWMOutput
Use PWM_A/PWM_B output directly

enum _flexpwm_trigger_frequency
FlexPWM output trigger frequency selection.

Controls the trigger output frequency when CTRL[LDFQ] is non-zero. Only takes effect when reload frequency divider is active (LDFQ > 0).

Values:

enumerator kFLEXPWM_TriggerFrequency_EveryCycle
Trigger every PWM cycle even if reload doesn't occur

enumerator kFLEXPWM_TriggerFrequency_OnReload
Trigger only on last cycle before reload opportunity

enum _flexpwm_output_trigger_mask
FlexPWM output trigger enable mask.

These flags can be OR'ed together to enable multiple trigger sources simultaneously. Used with outTriggerEnable field in flexpwm_output_trigger_config_t structure.

Values:

enumerator kFLEXPWM_OutputTriggerMask_VAL0
VAL0 match triggers PWM_OUT_TRIG0

enumerator kFLEXPWM_OutputTriggerMask_VAL1
VAL1 match triggers PWM_OUT_TRIG1

enumerator kFLEXPWM_OutputTriggerMask_VAL2
VAL2 match triggers PWM_OUT_TRIG0

enumerator kFLEXPWM_OutputTriggerMask_VAL3
VAL3 match triggers PWM_OUT_TRIG1

enumerator kFLEXPWM_OutputTriggerMask_VAL4
VAL4 match triggers PWM_OUT_TRIG0

enumerator kFLEXPWM_OutputTriggerMask_VAL5
VAL5 match triggers PWM_OUT_TRIG1

enum _flexpwm_capture_edge
FlexPWM input capture edge selection.

This enumeration defines the edge detection modes for input capture. The enum values directly map to hardware register bit values.

Values:

enumerator kFLEXPWM_CaptureEdge_Disabled
Disable capture (register bit value: 00b)

enumerator kFLEXPWM_CaptureEdge_Falling
Capture on falling edge (register bit value: 01b)

enumerator kFLEXPWM_CaptureEdge_Rising
Capture on rising edge (register bit value: 10b)

enumerator kFLEXPWM_CaptureEdge_Both
Capture on any edge (register bit value: 11b)

enum _flexpwm_capture_fifo_watermark
FlexPWM capture FIFO watermark selection.

This enumeration defines the FIFO watermark levels that determine when a watermark interrupt is triggered based on the number of valid samples in the FIFO.

Values:

enumerator kFLEXPWM_CaptureFifoWatermark_1
Trigger when FIFO has 1 or more samples (00b)

enumerator kFLEXPWM_CaptureFifoWatermark_2
Trigger when FIFO has 2 or more samples (01b)

enumerator kFLEXPWM_CaptureFifoWatermark_3
Trigger when FIFO has 3 or more samples (10b)

enumerator kFLEXPWM_CaptureFifoWatermark_4
Trigger when FIFO is full with 4 samples (11b)

enum _flexpwm_capture_input_select
FlexPWM capture input source selection.

This enumeration selects the source signal for input capture.

Values:

enumerator kFLEXPWM_CaptureInput_RawSignal
Use raw PWM input signal (register bit value: 0)

enumerator kFLEXPWM_CaptureInput_EdgeCounter
Use edge counter output (register bit value: 1). Used to reduce capture frequency by counting N edges

enum _flexpwm_capture_index
FlexPWM capture value index.

This enumeration defines the index for accessing capture value registers (CVAL0-5).

Note: Capture channel to CVAL register mapping (non-intuitive hardware mapping):

- Capture_X Edge0/Edge1 -> CVAL0/CVAL1 (and CVAL0CYC/CVAL1CYC)
- Capture_A Edge0/Edge1 -> CVAL2/CVAL3 (and CVAL2CYC/CVAL3CYC)
- Capture_B Edge0/Edge1 -> CVAL4/CVAL5 (and CVAL4CYC/CVAL5CYC)

Values:

enumerator kFLEXPWM_Capture_X_Edge0
CVAL0 - Capture_X Edge0

enumerator kFLEXPWM_Capture_X_Edge1
CVAL1 - Capture_X Edge1

enumerator kFLEXPWM_Capture_A_Edge0
CVAL2 - Capture_A Edge0

enumerator kFLEXPWM_Capture_A_Edge1
CVAL3 - Capture_A Edge1

enumerator kFLEXPWM_Capture_B_Edge0
CVAL4 - Capture_B Edge0

enumerator kFLEXPWM_Capture_B_Edge1
CVAL5 - Capture_B Edge1

enum _flexpwm_capture_channel
FlexPWM capture channel selection.

This enumeration defines the capture channel for enable/disable operations.

Values:

enumerator kFLEXPWM_Capture_A
Capture channel A (PWM_A)

enumerator kFLEXPWM_Capture_B
Capture channel B (PWM_B)

enumerator kFLEXPWM_Capture_X
Capture channel X (PWM_X)

enum _flexpwm_submodule_interrupt_enable
FlexPWM submodule interrupt enable flags.

These flags can be OR'ed together to enable multiple interrupt sources simultaneously. Use with *FLEXPWM_EnableSubmoduleInterrupts()* and *FLEXPWM_DisableSubmoduleInterrupts()*.

Values:

enumerator kFLEXPWM_Compare0InterruptEnable
VAL0 compare interrupt

enumerator kFLEXPWM_Compare1InterruptEnable
VAL1 compare interrupt

enumerator kFLEXPWM_Compare2InterruptEnable
VAL2 compare interrupt

enumerator kFLEXPWM_Compare3InterruptEnable
VAL3 compare interrupt

enumerator kFLEXPWM_Compare4InterruptEnable
VAL4 compare interrupt

enumerator kFLEXPWM_Compare5InterruptEnable
VAL5 compare interrupt

enumerator kFLEXPWM_CaptureX0InterruptEnable
Capture X0 interrupt

enumerator kFLEXPWM_CaptureX1InterruptEnable
Capture X1 interrupt

enumerator kFLEXPWM_CaptureB0InterruptEnable
Capture B0 interrupt

enumerator kFLEXPWM_CaptureB1InterruptEnable
Capture B1 interrupt

enumerator kFLEXPWM_CaptureA0InterruptEnable
Capture A0 interrupt

enumerator kFLEXPWM_CaptureA1InterruptEnable
Capture A1 interrupt

enumerator kFLEXPWM_ReloadInterruptEnable
Reload interrupt

enumerator kFLEXPWM_ReloadErrorInterruptEnable
Reload error interrupt

enum _flexpwm_submodule_status_flag
FlexPWM submodule status flags.

These flags represent the status of various events in a FlexPWM submodule. Use with *FLEXPWM_GetSubmoduleStatusFlags()* and *FLEXPWM_ClearSubmoduleStatusFlags()*.

Note: kFLEXPWM_RegisterUpdatedFlag is read-only and cannot be cleared by software.

Values:

enumerator kFLEXPWM_Compare0Flag
VAL0 compare flag

enumerator kFLEXPWM_Compare1Flag
VAL1 compare flag

enumerator kFLEXPWM_Compare2Flag
VAL2 compare flag

enumerator kFLEXPWM_Compare3Flag
VAL3 compare flag

enumerator kFLEXPWM_Compare4Flag
VAL4 compare flag

enumerator kFLEXPWM_Compare5Flag
VAL5 compare flag

enumerator kFLEXPWM_CaptureX0Flag
Capture X0 flag

enumerator kFLEXPWM_CaptureX1Flag
Capture X1 flag

enumerator kFLEXPWM_CaptureB0Flag
Capture B0 flag

enumerator kFLEXPWM_CaptureB1Flag
Capture B1 flag

enumerator kFLEXPWM_CaptureA0Flag
Capture A0 flag

enumerator kFLEXPWM_CaptureA1Flag
Capture A1 flag

enumerator kFLEXPWM_ReloadFlag

Reload flag

enumerator kFLEXPWM_ReloadErrorFlag

Reload error flag

enumerator kFLEXPWM_RegisterUpdatedFlag

Register updated flag (read-only)

enum _flexpwm_fault_interrupt_enable

FlexPWM fault interrupt enable flags.

These flags can be OR'ed together to enable multiple fault interrupt sources. Use with *FLEXPWM_EnableFaultInterrupts()* and *FLEXPWM_DisableFaultInterrupts()*.

Note: For dual-channel platforms (FSL_FEATURE_PWM_FAULT_CH_COUNT > 1): The same bit mask values are used for both fault channels:

- faultChannel = 0: Controls FAULT0-3
- faultChannel = 1: Controls FAULT4-7 The bit positions are the same, but the function parameter determines which fault channel is configured.

Values:

enumerator kFLEXPWM_Fault0InterruptEnable

FAULT0 (ch0) or FAULT4 (ch1) interrupt

enumerator kFLEXPWM_Fault1InterruptEnable

FAULT1 (ch0) or FAULT5 (ch1) interrupt

enumerator kFLEXPWM_Fault2InterruptEnable

FAULT2 (ch0) or FAULT6 (ch1) interrupt

enumerator kFLEXPWM_Fault3InterruptEnable

FAULT3 (ch0) or FAULT7 (ch1) interrupt

enum _flexpwm_fault_status_flag

FlexPWM fault status flags (FFLAG)

These flags represent the latched fault status (FFLAG bits in FSTS register). Use with *FLEXPWM_GetFaultStatusFlags()* and *FLEXPWM_ClearFaultStatusFlags()*.

Note: FFLAG characteristics:

- Write-1-to-clear: Software must write 1 to clear each flag
- Latched event: Set within 2 CPU cycles after FAULTx pin transitions to active state
- Historical record: Indicates a fault event has occurred (not current pin state)
- Interrupt capable: Can generate CPU interrupt requests when enabled
- Manual clearing: Does not auto-clear; requires explicit software action

---

Note: To check the real-time, filtered hardware pin status (not the latched event), use *FLEXPWM_GetFaultFilteredPinStatusFlags()* to read FFPIN instead.

Note: For dual-channel platforms: The same bit mask values apply to both channels.

- faultChannel = 0: FAULT0-3 flags
- faultChannel = 1: FAULT4-7 flags

Values:

enumerator kFLEXPWM_Fault0Flag
FAULT0 (ch0) or FAULT4 (ch1) flag

enumerator kFLEXPWM_Fault1Flag
FAULT1 (ch0) or FAULT5 (ch1) flag

enumerator kFLEXPWM_Fault2Flag
FAULT2 (ch0) or FAULT6 (ch1) flag

enumerator kFLEXPWM_Fault3Flag
FAULT3 (ch0) or FAULT7 (ch1) flag

enum _flexpwm_fault_pin_status
FlexPWM filtered fault pin status (FFPIN - read-only)

These flags represent the real-time status of fault pins after filtering (FFPIN bits in FSTS register). Use with *FLEXPWM_GetFaultFilteredPinStatusFlags()* to check if external fault signals are currently active.

Note: FFPIN characteristics:

- Read-only: Cannot be written or cleared by software
- Real-time status: Reflects current state of filtered FAULTx pin (not historical)
- Polarity converted: Always shows active-high logic (1 = fault present)
- Filter output: Shows fault filter output, not raw pin state
- No interrupt: Does not directly generate interrupts (use FFLAG for interrupts)
- Filter delay: May remain high briefly after raw pin clears due to filter latency

---

Note: Key differences from FFLAG:

- FFLAG: Software-clearable latched flag (historical event - “a fault occurred”)
- FFPIN: Read-only real-time status (current state - “fault is active now”)

Note: For dual-channel platforms: The same bit positions apply to both channels.

- faultChannel = 0: FAULT0-3 pin status

- faultChannel = 1: FAULT4-7 pin status

Values:

enumerator kFLEXPWM_Fault0PinActive
FAULT0/4 pin active (filtered)

enumerator kFLEXPWM_Fault1PinActive
FAULT1/5 pin active (filtered)

enumerator kFLEXPWM_Fault2PinActive
FAULT2/6 pin active (filtered)

enumerator kFLEXPWM_Fault3PinActive
FAULT3/7 pin active (filtered)

enum _flexpwm_dma_capture_source
FlexPWM DMA capture source selection.

This enumeration defines the trigger source for DMA read requests from capture FIFOs.

Values:

enumerator kFLEXPWM_DMADisable
Read DMA requests disabled

enumerator kFLEXPWM_DMAFIFOWatermark
Exceeding FIFO watermark sets read DMA request

enumerator kFLEXPWM_DMALocalSync
Local sync sets read DMA request

enumerator kFLEXPWM_DMALocalReload
Local reload sets read DMA request

enum _flexpwm_dma_enable
FlexPWM DMA enable flags.

These flags are used with *FLEXPWM_EnableDMA()* and *FLEXPWM_DisableDMA()* to control individual DMA channel enables. Multiple flags can be ORed together.

Values:

enumerator kFLEXPWM_DMA_CaptureX0Enable
Capture X0 FIFO DMA enable (CX0DE)

enumerator kFLEXPWM_DMA_CaptureX1Enable
Capture X1 FIFO DMA enable (CX1DE)

enumerator kFLEXPWM_DMA_CaptureB0Enable
Capture B0 FIFO DMA enable (CB0DE)

enumerator kFLEXPWM_DMA_CaptureB1Enable
Capture B1 FIFO DMA enable (CB1DE)

enumerator kFLEXPWM_DMA_CaptureA0Enable
Capture A0 FIFO DMA enable (CA0DE)

enumerator kFLEXPWM_DMA_CaptureA1Enable
Capture A1 FIFO DMA enable (CA1DE)

enumerator kFLEXPWM_DMA_ValueWriteEnable
Value registers DMA write enable (VALDE)

typedef enum *_flexpwm_clock_source* flexpwm_clock_source_t
FlexPWM clock source selection.

typedef enum *_flexpwm_prescaler* flexpwm_prescaler_t
FlexPWM clock prescaler selection.

typedef enum *_flexpwm_load_mode* flexpwm_load_mode_t
FlexPWM load mode selection.

typedef enum *_flexpwm_reload_source* flexpwm_reload_source_t
FlexPWM reload source selection.

typedef enum *_flexpwm_init_source* flexpwm_init_source_t
FlexPWM counter initialization source selection.

typedef enum *_flexpwm_compare_mode* flexpwm_compare_mode_t
FlexPWM PWM compare mode selection.

typedef enum *_flexpwm_submodule_mask* flexpwm_submodule_mask_t
FlexPWM submodule mask for multi-submodule operations.

These flags can be OR'ed together to specify multiple submodules simultaneously. Used with functions that operate on multiple submodules at once.

typedef struct *_flexpwm_counter_config* flexpwm_counter_config_t
FlexPWM counter configuration structure.

typedef struct *_flexpwm_reload_config* flexpwm_reload_config_t
FlexPWM reload configuration structure.

typedef struct *_flexpwm_submodule_config* flexpwm_submodule_config_t
FlexPWM submodule configuration structure.

This structure defines the basic configuration parameters for a FlexPWM submodule, including clock source, prescaler, counter behavior, reload mechanism, and operating modes.

Note: This configuration writes to buffered registers. After calling *FLEXPWM_ConfigSubmodule()*, you must set MCTRL[LDOK] using a separate function to transfer the configuration to active registers.

Note: This function does not start the PWM counter. Use a separate API to enable the counter.

typedef enum *_flexpwm_pwm_polarity* flexpwm_pwm_polarity_t

FlexPWM PWM output polarity selection.

typedef enum *_flexpwm_ipol_source* flexpwm_ipol_source_t

FlexPWM PWM output source selection for complementary mode.

typedef struct *_flexpwm_pwm_channel_config* flexpwm_pwm_channel_config_t

FlexPWM PWM channel configuration structure.

This structure defines the configuration for a PWM channel pair (PWM_A or PWM_B), including compare values, output polarity, and initial value.

typedef struct *_flexpwm_pwm_config* flexpwm_pwm_config_t

FlexPWM PWM configuration structure.

This structure defines the PWM output configuration for a FlexPWM submodule, including compare values, polarity, and operating mode (independent or complementary).

Note: For complementary mode: Only the selected channel (specified by ipolSource) needs to be configured. The other channel will be ignored by the hardware. For example, if ipolSource = kFLEXPWM_IPOL_PWM23, only pwma will be used.

Note: For independent mode: Both pwma and pwmb must be configured separately. The ipolSource field is ignored when complementary = false.

typedef enum *_flexpwm_force_output_source* flexpwm_force_output_source_t

FlexPWM FORCE_OUT event source selection.

typedef enum *_flexpwm_force_pwm_source* flexpwm_force_pwm_source_t

FlexPWM PWM source selection for deadtime logic (DTSRCSEL)

typedef struct *_flexpwm_force_out_channel_config* flexpwm_force_out_channel_config_t

FlexPWM PWM channel force output configuration structure.

This structure defines the force output configuration for a PWM channel (PWM_A or PWM_B), including the source selection and software-controlled output value.

typedef struct *_flexpwm_force_out_config* flexpwm_force_out_config_t

FlexPWM force output configuration structure.

This structure defines the force output (FORCE_OUT event) configuration for a FlexPWM submodule, including the trigger source, PWM logical initial value, and PWM source selection.

typedef enum *_flexpwm_fault_output_behavior* flexpwm_fault_output_behavior_t

FlexPWM fault output behavior during fault condition.

Defines how PWM outputs respond when a fault is detected. Values map directly to SMxOCTRL[PWMAFS/PWMBFS/PWMXFS] bitfields.

typedef enum *_flexpwm_fault_mask* flexpwm_fault_mask_t

FlexPWM fault input mask for fault protection configuration.

These flags can be OR'ed together to configure multiple fault inputs simultaneously with the same protection parameters. Used with *FLEXPWM_ConfigFaultProtection()* function.

Note: For dual-channel platforms (FSL_FEATURE_PWM_FAULT_CH_COUNT > 1):

- When faultChannel = 0: Use kFLEXPWM_FaultMask_0/1/2/3 for FAULT0-3
- When faultChannel = 1: Use kFLEXPWM_FaultMask_0/1/2/3 for FAULT4-7 The enum values are the same, but the channel parameter determines which physical fault inputs are configured.

---

typedef struct *_flexpwm_fault_output_config* flexpwm_fault_output_config_t

FlexPWM fault output configuration structure.

Configures fault protection for a single PWM output (PWM_A, PWM_B, or PWM_X). Supports both single-channel (FAULT0-3 only) and dual-channel (FAULT0-3 and FAULT4-7) platforms via conditional compilation. Can use flexpwm_fault_mask_t enum values (kFLEXPWM_FaultMask_0/1/2/3) and OR them together for multiple faults.

typedef struct *_flexpwm_fault_submodule_config* flexpwm_fault_submodule_config_t

FlexPWM fault submodule configuration structure.

Configures fault protection for all three PWM outputs (PWM_A, PWM_B, PWM_X) of a FlexPWM submodule. Includes fault disable mapping and output behavior for each output.

typedef enum *_flexpwm_fault_clearing_mode* flexpwm_fault_clearing_mode_t

FlexPWM fault clearing mode selection.

Defines how fault conditions are cleared after a fault event.

typedef struct *_flexpwm_fault_config* flexpwm_fault_config_t

FlexPWM fault protection configuration structure.

Configures global fault protection parameters for one fault channel. These settings apply to all submodules and their outputs that reference this channel. For platforms with dual fault channels (FAULT0-3 and FAULT4-7), configure each channel separately using *FLEXPWM_ConfigFaultProtection()*.

typedef struct *_flexpwm_fault_filter_config* flexpwm_fault_filter_config_t

FlexPWM fault filter configuration structure.

Configures input filter for fault protection pins to debounce electrical noise. The filter requires the input signal to be stable for N consecutive samples (where each sample is taken every T IPBus clock cycles) before reporting a fault. Helps avoid false fault triggers from noise on FAULTx pins.

typedef enum *_flexpwm_write_protect* flexpwm_write_protect_t

FlexPWM write protection mode selection.

typedef enum *_flexpwm_stretch_prescaler* flexpwm_stretch_prescaler_t

FlexPWM trigger signal stretch prescaler selection.

Stretch IPBus clock count prescaler for trigger signals including: mux0_trig, mux1_trig, out0_trig, out1_trig, pwma_trig, pwmb_trig

typedef enum *_flexpwm_trigger_mux_source* flexpwm_trigger_mux_source_t

FlexPWM output trigger source selection for PWM_MUX_TRIG0/1 ports.

Selects which signal is routed to the PWM_MUX_TRIG0/1 output ports.

typedef enum *_flexpwm_trigger_frequency* flexpwm_trigger_frequency_t

FlexPWM output trigger frequency selection.

Controls the trigger output frequency when CTRL[LDFQ] is non-zero. Only takes effect when reload frequency divider is active (LDFQ > 0).

typedef enum *_flexpwm_output_trigger_mask* flexpwm_output_trigger_mask_t

FlexPWM output trigger enable mask.

These flags can be OR'ed together to enable multiple trigger sources simultaneously. Used with outTriggerEnable field in flexpwm_output_trigger_config_t structure.

typedef struct *_flexpwm_output_trigger_config* flexpwm_output_trigger_config_t

FlexPWM output trigger configuration structure.

This structure defines the output trigger configuration for a FlexPWM submodule, including trigger source mapping from VAL registers, trigger frequency control, and output port routing.

Note: outTriggerEnable is a 6-bit mask corresponding to OUT_TRIG_EN[5:0], where each bit enables a trigger source from VAL0-5 register matches:

- bit[0]: VAL0 match triggers PWM_OUT_TRIG0
- bit[1]: VAL1 match triggers PWM_OUT_TRIG1
- bit[2]: VAL2 match triggers PWM_OUT_TRIG0
- bit[3]: VAL3 match triggers PWM_OUT_TRIG1
- bit[4]: VAL4 match triggers PWM_OUT_TRIG0
- bit[5]: VAL5 match triggers PWM_OUT_TRIG1 Multiple bits can be set to combine trigger sources (OR logic).

Note: triggerFrequency only takes effect when CTRL[LDFQ] is non-zero.

- false: Triggers output every PWM cycle even if reload doesn't occur
- true: Triggers output only on the last PWM cycle before a reload opportunity

---

typedef enum *_flexpwm_capture_edge* flexpwm_capture_edge_t

FlexPWM input capture edge selection.

This enumeration defines the edge detection modes for input capture. The enum values directly map to hardware register bit values.

typedef enum *_flexpwm_capture_fifo_watermark* flexpwm_capture_fifo_watermark_t

FlexPWM capture FIFO watermark selection.

This enumeration defines the FIFO watermark levels that determine when a watermark interrupt is triggered based on the number of valid samples in the FIFO.

typedef enum *_flexpwm_capture_input_select* flexpwm_capture_input_select_t

FlexPWM capture input source selection.

This enumeration selects the source signal for input capture.

typedef enum *_flexpwm_capture_index* flexpwm_capture_index_t

FlexPWM capture value index.

This enumeration defines the index for accessing capture value registers (CVAL0-5).

Note: Capture channel to CVAL register mapping (non-intuitive hardware mapping):

- Capture_X Edge0/Edge1 -> CVAL0/CVAL1 (and CVAL0CYC/CVAL1CYC)
- Capture_A Edge0/Edge1 -> CVAL2/CVAL3 (and CVAL2CYC/CVAL3CYC)
- Capture_B Edge0/Edge1 -> CVAL4/CVAL5 (and CVAL4CYC/CVAL5CYC)

---

typedef enum *_flexpwm_capture_channel* flexpwm_capture_channel_t

FlexPWM capture channel selection.

This enumeration defines the capture channel for enable/disable operations.

typedef struct *_flexpwm_capture_channel_config* flexpwm_capture_channel_config_t

FlexPWM single capture channel configuration structure.

This structure defines the configuration parameters for a single input capture channel.

typedef struct *_flexpwm_input_capture_config* flexpwm_input_capture_config_t

FlexPWM input capture configuration structure.

This structure contains the configuration for three independent input capture channels. Users can configure only the channels they need; unused channels should be left at default values.

Note: Capture channel to CVAL/CVALxCYC register mapping (important for reading capture values):

- captureA -> CVAL2/CVAL3 (CVAL2CYC/CVAL3CYC)
- captureB -> CVAL4/CVAL5 (CVAL4CYC/CVAL5CYC)
- captureX -> CVAL0/CVAL1 (CVAL0CYC/CVAL1CYC)

---

typedef struct *_flexpwm_capture_filter_channel_config* flexpwm_capture_filter_channel_config_t

FlexPWM capture filter channel configuration structure.

This structure defines the digital filter parameters for a single capture input pin.

typedef struct *_flexpwm_capture_filter_config* flexpwm_capture_filter_config_t

FlexPWM input capture filter configuration structure.

This structure contains the filter configuration for three independent capture input pins.

Note: Capture filter to register mapping:

- captureA -> SMxCAPTFILTA register
- captureB -> SMxCAPTFILTB register
- captureX -> SMxCAPTFILTX register

---

typedef enum *_flexpwm_dma_capture_source* flexpwm_dma_capture_source_t

FlexPWM DMA capture source selection.

This enumeration defines the trigger source for DMA read requests from capture FIFOs.

typedef struct *_flexpwm_dma_config* flexpwm_dma_config_t

FlexPWM DMA configuration structure.

This structure holds the DMA control strategy configuration. These settings should be configured during initialization and typically do not need to be modified at runtime.

uint32_t FLEXPWM_GetInstance(PWM_Type *base)

Get the instance for FlexPWM module.

Parameters

- base – FlexPWM base address

Returns

Instance number if valid base address is provided, otherwise returns *ARRAY_SIZE(s_flexpwmBases)*

status_t FLEXPWM_Init(PWM_Type *base)

Initialize the FlexPWM module.

This function initializes the FlexPWM peripheral by:

- Enabling clocks for all submodules (SM0-SM3)
- Releasing peripheral reset (if supported by platform)

After calling this function, the FlexPWM module is ready for configuration. Additional configuration functions (FLEXPWM_ConfigSubmodule, FLEXPWM_ConfigPWM, etc.) should be called to set up the desired PWM behavior.

Note: This function does not configure any PWM parameters. It only enables the peripheral.

Note: This function does not start PWM generation. Use *FLEXPWM_EnableSubmoduleCounter()* and *FLEXPWM_EnablePWMOutput()* to start PWM output.

Parameters

- base – FlexPWM peripheral base address.

Returns

kStatus_Success if initialization is successful. kStatus_InvalidArgument if the base address is invalid.

status_t FLEXPWM_DeInit(PWM_Type *base)

Deinitialize the FlexPWM module.

This function deinitializes the FlexPWM peripheral by:

- Disabling clocks for all submodules (SM0-SM3)
- Asserting peripheral reset (if supported by platform)

After calling this function, all FlexPWM registers are reset.

Note: To re-enable PWM operation, call *FLEXPWM_Init()* again.

Parameters

- base – FlexPWM peripheral base address.

Returns

kStatus_Success if deinitialization is successful. kStatus_InvalidArgument if the base address is invalid.

FLEXPWM_GET_FCTRL_REG(base, ch)

FLEXPWM_GET_FSTS_REG(base, ch)

FLEXPWM_GET_FFILT_REG(base, ch)

FLEXPWM_GET_FCTRL2_REG(base, ch)

struct __flexpwm_counter_config

#include <fsl_flexpwm.h> FlexPWM counter configuration structure.

Public Members

uint16_t initValue

Counter initial value (INIT register).

uint16_t modValue

Counter modulo value (VAL1 register). Defines the PWM period upper limit.

flexpwm_init_source_t initSource

Counter initialization source selection. Determines when the counter resets to initValue.

struct __flexpwm_reload_config

#include <fsl_flexpwm.h> FlexPWM reload configuration structure.

Public Members

flexpwm_load_mode_t loadMode

Register load mode selection.

- kFLEXPWM_LoadMode_Opportunity: Load at next reload opportunity (PWM cycle)
- kFLEXPWM_LoadMode_Immediate: Load immediately after LDOK is set

uint8_t loadFrequency

Load frequency divider (LDFQ field, 0-15). Buffered registers reload every (loadFrequency + 1) PWM cycles.

- 0: Every PWM cycle
- 1: Every 2 PWM cycles
- 15: Every 16 PWM cycles

bool enableHalfCycleReload

Enable reload at half-cycle point (VAL0 match).

- true: Reload can occur at VAL0 match
- false: No reload at half-cycle

bool enableFullCycleReload

Enable reload at full-cycle point (VAL1 match).

- true: Reload can occur at VAL1 match
- false: No reload at full-cycle

flexpwm_reload_source_t reloadSource

Reload signal source selection.

- kFLEXPWM_ReloadSource_LocalReload: Use local reload signal
- kFLEXPWM_ReloadSource_MasterReload: Use master reload from submodule 0

uint16_t halfCycleValue

Half-cycle reload point (VAL0 register). This value determines when half-cycle reload occurs. Note: Does not need to be exactly half the PWM period.

struct __flexpwm_submodule_config

#include <fsl_flexpwm.h> FlexPWM submodule configuration structure.

This structure defines the basic configuration parameters for a FlexPWM submodule, including clock source, prescaler, counter behavior, reload mechanism, and operating modes.

Note: This configuration writes to buffered registers. After calling *FLEXPWM_ConfigSubmodule()*, you must set MCTRL[LDOK] using a separate function to transfer the configuration to active registers.

Note: This function does not start the PWM counter. Use a separate API to enable the counter.

Public Members

flexpwm_clock_source_t clockSource

Clock source for the submodule PWM counter.

flexpwm_prescaler_t prescaler

Clock prescaler divider.

flexpwm_counter_config_t counterConfig

Counter initialization and behavior configuration.

flexpwm_reload_config_t reloadConfig

Register reload mechanism configuration.

bool enableDebugMode

PWM behavior in Debug mode.

- true: PWM continues running when MCU enters Debug mode
- false: PWM stops when MCU enters Debug mode

struct __flexpwm__pwm__channel__config

#include <fsl_flexpwm.h> FlexPWM PWM channel configuration structure.

This structure defines the configuration for a PWM channel pair (PWM_A or PWM_B), including compare values, output polarity, and initial value.

Public Members

uint16_t compareValue__ON

Compare value that defines the PWM turn-on edge (in normal polarity). Mapping per channel:

- PWM_A: written to VAL2
- PWM_B: written to VAL4
- PWM_X: written to VAL0

uint16_t compareValue__OFF

Compare value that defines the PWM turn-off edge (in normal polarity). Mapping per channel:

- PWM_A: written to VAL3
- PWM_B: written to VAL5
- PWM_X: written to VAL1

flexpwm_pwm_polarity_t polarity

Output polarity.

- kFLEXPWM_Polarity_ActiveLow: PWM signal is inverted (active low)
- kFLEXPWM_Polarity_ActiveHigh: PWM signal is normal (active high)

struct __flexpwm__pwm__config

#include <fsl_flexpwm.h> FlexPWM PWM configuration structure.

This structure defines the PWM output configuration for a FlexPWM submodule, including compare values, polarity, and operating mode (independent or complementary).

Note: For complementary mode: Only the selected channel (specified by ipolSource) needs to be configured. The other channel will be ignored by the hardware. For example, if ipolSource = kFLEXPWM_IPOL_PWM23, only pwma will be used.

Note: For independent mode: Both pwma and pwmb must be configured separately. The ipolSource field is ignored when complementary = false.

Public Members

flexpwm_pwm_channel_config_t pwma

PWM_A (PWM23) channel configuration

flexpwm_pwm_channel_config_t pwmb

PWM_B (PWM45) channel configuration

bool complementary

Operating mode.

- true: Complementary mode (PWM_A and PWM_B form a complementary pair)
- false: Independent mode (PWM_A and PWM_B are independent channels)

flexpwm_ipol_source_t ipolSource

MCTRL[IPOL] - Select which channel pair as complementary source. Only used when complementary = true.

- kFLEXPWM_IPOL_PWM23: Use PWM23 as source
- kFLEXPWM_IPOL_PWM45: Use PWM45 as source

struct __flexpwm__force__out__channel__config

#include <fsl_flexpwm.h> FlexPWM PWM channel force output configuration structure.

This structure defines the force output configuration for a PWM channel (PWM_A or PWM_B), including the source selection and software-controlled output value.

Public Members

flexpwm_force_pwm_source_t source

PWM source selection for this channel (DTSRCSEL).

bool softwareValue

When source=kFLEXPWM_ForcePwmSource_Software, this value (0 or 1) is output to deadtime logic (SWCOUT).

struct _flexpwm_force_out_config

#include <fsl_flexpwm.h> FlexPWM force output configuration structure.

This structure defines the force output (FORCE_OUT event) configuration for a FlexPWM submodule, including the trigger source, PWM logical initial value, and PWM source selection.

Public Members

flexpwm_force_output_source_t forceSource

FORCE_OUT trigger source (CTRL2[FORCE_SEL]).

flexpwm_force_out_channel_config_t pwma

PWM_A (PWM23) force output configuration.

flexpwm_force_out_channel_config_t pwmb

PWM_B (PWM45) force output configuration.

bool pwma_initialValue

PWM_A (PWM23) logical initial value in normal polarity.

- true: PWM_A output starts at logic 1
- false: PWM_A output starts at logic 0

bool pwmb_initialValue

PWM_B (PWM45) logical initial value in normal polarity.

- true: PWM_B output starts at logic 1
- false: PWM_B output starts at logic 0

bool pwmx_initialValue

PWM_X logical initial value in normal polarity.

- true: PWM_X output starts at logic 1
- false: PWM_X output starts at logic 0

struct _flexpwm_fault_output_config

#include <fsl_flexpwm.h> FlexPWM fault output configuration structure.

Configures fault protection for a single PWM output (PWM_A, PWM_B, or PWM_X). Supports both single-channel (FAULT0-3 only) and dual-channel (FAULT0-3 and FAULT4-7) platforms via conditional compilation. Can use flexpwm_fault_mask_t enum values (kFLEXPWM_FaultMask_0/1/2/3) and OR them together for multiple faults.

Public Members

uint16_t disableMask_ch0

Fault disable mask for channel 0 (FAULT0-3). Each bit selects one fault input:

- bit[0] = FAULT0
- bit[1] = FAULT1
- bit[2] = FAULT2
- bit[3] = FAULT3 Maps to SMxDISMAP[0] register.

flexpwm_fault_output_behavior_t outputBehavior

Output behavior when fault is detected Maps to SMxOCTRL[PWMAFS/PWMBFS/PWMXFS].

struct _flexpwm_fault_submodule_config

#include <fsl_flexpwm.h> FlexPWM fault submodule configuration structure.

Configures fault protection for all three PWM outputs (PWM_A, PWM_B, PWM_X) of a FlexPWM submodule. Includes fault disable mapping and output behavior for each output.

Public Members

flexpwm_fault_output_config_t pwma

PWM_A (PWM23) fault configuration. Controls which faults disable PWM_A output and how PWM_A behaves during fault.

flexpwm_fault_output_config_t pwmb

PWM_B (PWM45) fault configuration. Controls which faults disable PWM_B output and how PWM_B behaves during fault.

flexpwm_fault_output_config_t pwmx

PWM_X fault configuration. Controls which faults disable PWM_X output and how PWM_X behaves during fault.

struct _flexpwm_fault_config

#include <fsl_flexpwm.h> FlexPWM fault protection configuration structure.

Configures global fault protection parameters for one fault channel. These settings apply to all submodules and their outputs that reference this channel. For platforms with dual fault channels (FAULT0-3 and FAULT4-7), configure each channel separately using *FLEXPWM_ConfigFaultProtection()*.

Public Members

bool faultInputActiveLevel

FAULTx input active level (FCTRL[FLVL])

- false: Logic 0 on FAULTx indicates fault (active low)
- true: Logic 1 on FAULTx indicates fault (active high)

flexpwm_fault_clearing_mode_t faultClearingMode

Fault clearing mode (FCTRL[FAUTO])

bool enableSafetyMode

Safety Mode (FCTRL[FSAFE]) Only useful for Manual fault clearing mode

- false: Normal mode (PWM outputs enabled when FFLAG clears)
- true: Safe mode (PWM outputs enabled only when both FFLAG and FFPIN are clear)

bool enableFullCycleRecovery

Full Cycle Recovery (FSTS[FFULL])

- true: PWM outputs can be re-enabled at PWM full cycle start
- false: No re-enable at full cycle

bool enableHalfCycleRecovery

Half Cycle Recovery (FSTS[FHALF])

- true: PWM outputs can be re-enabled at PWM half cycle start
- false: No re-enable at half cycle

bool enableCombinatorialPath

Combinational Path Control (FCTRL2[NOCOMB])

- true: Combinational path active - fault input directly disables PWM outputs (fastest response)
- false: Combinational path disabled - only latched fault signal disables PWM outputs (adds latency)

struct __flexpwm_fault_filter_config

#include <fsl_flexpwm.h> FlexPWM fault filter configuration structure.

Configures input filter for fault protection pins to debounce electrical noise. The filter requires the input signal to be stable for N consecutive samples (where each sample is taken every T IPBus clock cycles) before reporting a fault. Helps avoid false fault triggers from noise on FAULTx pins.

Public Members

uint8_t filterPeriod

Fault Filter Period (FFILT[FILT_PER]) Sampling period in IPBus clock cycles. Range: 0-255

- 0: Input filter bypassed
- N: FAULTx sampled every N IPBus cycles

uint8_t filterCount

Fault Filter Count (FFILT[FILT_CNT]) Number of consecutive stable samples required to accept input transition. Range: 0-7 (actual samples = value + 3)

- 0: 3 samples

- 1: 4 samples
- 7: 10 samples

bool enableGlitchStretch

Fault Glitch Stretching (FFILT[GSTR])

- false: Glitch stretching disabled
- true: Input fault signals stretched to min 2 IPBus cycles Prevents false triggers from narrow glitches

struct __flexpwm_output_trigger_config

#include <fsl_flexpwm.h> FlexPWM output trigger configuration structure.

This structure defines the output trigger configuration for a FlexPWM submodule, including trigger source mapping from VAL registers, trigger frequency control, and output port routing.

Note: outTriggerEnable is a 6-bit mask corresponding to OUT_TRIG_EN[5:0], where each bit enables a trigger source from VAL0-5 register matches:

- bit[0]: VAL0 match triggers PWM_OUT_TRIG0
- bit[1]: VAL1 match triggers PWM_OUT_TRIG1
- bit[2]: VAL2 match triggers PWM_OUT_TRIG0
- bit[3]: VAL3 match triggers PWM_OUT_TRIG1
- bit[4]: VAL4 match triggers PWM_OUT_TRIG0
- bit[5]: VAL5 match triggers PWM_OUT_TRIG1 Multiple bits can be set to combine trigger sources (OR logic).

Note: triggerFrequency only takes effect when CTRL[LDFQ] is non-zero.

- false: Triggers output every PWM cycle even if reload doesn't occur
- true: Triggers output only on the last PWM cycle before a reload opportunity

---

Public Members

uint16_t outTriggerEnable

Output trigger enable mask (OUT_TRIG_EN[5:0]). Bit mask for VAL0-5 trigger sources. OR'ed combination of flexpwm_output_trigger_mask_t Example: 0x05 enables VAL0 and VAL2 for TRIG0.

flexpwm_trigger_mux_source_t muxTrig0Source

PWM_MUX_TRIG0 port source selection (PWAOT0).

- kFLEXPWM_TriggerMuxSource_GeneratedTrigger: Route PWM_OUT_TRIG0 signal
- kFLEXPWM_TriggerMuxSource_PWMOutput: Route PWM_A output directly

flexpwm_trigger_mux_source_t muxTrig1Source

PWM_MUX_TRIG1 port source selection (PWBOT1).

- kFLEXPWM_TriggerMuxSource_GeneratedTrigger: Route PWM_OUT_TRIG1 signal
- kFLEXPWM_TriggerMuxSource_PWMOutput: Route PWM_B output directly

flexpwm_trigger_frequency_t triggerFrequency

Trigger frequency control (TRGFRQ). Controls trigger output frequency when CTRL[LDFQ] > 0.

- kFLEXPWM_TriggerFrequency_EveryCycle: Trigger every PWM cycle
- kFLEXPWM_TriggerFrequency_OnReload: Trigger only on last cycle before reload

flexpwm_stretch_prescaler_t stretchPrescaler

Trigger signal stretch prescaler (MCTRL2[STRETCH_CNT_PRSC]). Stretches trigger signal pulse width by specified IPBus clock cycles.

struct __flexpwm_capture_channel_config

#include <fsl_flexpwm.h> FlexPWM single capture channel configuration structure.

This structure defines the configuration parameters for a single input capture channel.

Public Members

flexpwm_capture_edge_t edge0

Edge0 capture edge selection

flexpwm_capture_edge_t edge1

Edge1 capture edge selection

bool oneshot

Capture mode.

- true: One-shot mode
- false: Free-running mode

flexpwm_capture_input_select_t inputSelect

Input signal source selection.

- kFLEXPWM_CaptureInput_RawSignal: Raw PWM input
- kFLEXPWM_CaptureInput_EdgeCounter: Edge counter output

bool enableEdgeCounter

Edge counter enable control.

- true: Enable edge counter (EDGCNT_EN = 1)
- false: Disable edge counter (EDGCNT_EN = 0)

uint8_t edgeCompareValue

Edge counter compare value (0-255). Capture occurs when edge count reaches this value. Only valid when enableEdgeCounter = true

flexpwm_capture_fifo_watermark_t fifoWatermark

FIFO watermark level

struct __flexpwm_input_capture_config

#include <fsl_flexpwm.h> FlexPWM input capture configuration structure.

This structure contains the configuration for three independent input capture channels. Users can configure only the channels they need; unused channels should be left at default values.

Note: Capture channel to CVAL/CVALxCYC register mapping (important for reading capture values):

- captureA -> CVAL2/CVAL3 (CVAL2CYC/CVAL3CYC)
- captureB -> CVAL4/CVAL5 (CVAL4CYC/CVAL5CYC)
- captureX -> CVAL0/CVAL1 (CVAL0CYC/CVAL1CYC)

Public Members

flexpwm_capture_channel_config_t captureA

Capture_A (PWM_A) channel configuration

flexpwm_capture_channel_config_t captureB

Capture_B (PWM_B) channel configuration

flexpwm_capture_channel_config_t captureX

Capture_X (PWM_X) channel configuration

struct __flexpwm_capture_filter_channel_config

#include <fsl_flexpwm.h> FlexPWM capture filter channel configuration structure.

This structure defines the digital filter parameters for a single capture input pin.

Public Members

uint8_t filterPeriod

Filter sampling period in IPBus clock cycles (0-255).

- 0: Bypass filter (default)
- Non-zero: Enable filter with specified sampling period

uint8_t filterCount

Number of consecutive consistent samples required (0-7). Actual sample count = filterCount + 3 (i.e., 3-10 samples)

struct _flexpwm_capture_filter_config

#include <fsl_flexpwm.h> FlexPWM input capture filter configuration structure.

This structure contains the filter configuration for three independent capture input pins.

Note: Capture filter to register mapping:

- captureA -> SMxCAPTFILTA register
- captureB -> SMxCAPTFILTB register
- captureX -> SMxCAPTFILTX register

Public Members

flexpwm_capture_filter_channel_config_t captureA

Capture_A filter configuration

flexpwm_capture_filter_channel_config_t captureB

Capture_B filter configuration

flexpwm_capture_filter_channel_config_t captureX

Capture_X filter configuration

struct _flexpwm_dma_config

#include <fsl_flexpwm.h> FlexPWM DMA configuration structure.

This structure holds the DMA control strategy configuration. These settings should be configured during initialization and typically do not need to be modified at runtime.

Public Members

flexpwm_dma_capture_source_t captureSource

Capture DMA enable source selection (CAPTDE[7:6])

bool fifoWatermarkAND

FIFO watermark AND/OR control (FAND).

- true: Selected FIFO watermarks are ANDed
- false: Selected FIFO watermarks are ORed

2.31 FLEXRAM: on-chip RAM manager

void FLEXRAM_Init(FLEXRAM_Type *base)

FLEXRAM module initialization function.

Parameters

- base – FLEXRAM base address.

void FLEXRAM_Deinit(FLEXRAM_Type *base)
De-initializes the FLEXRAM.

static inline uint32_t FLEXRAM_GetInterruptStatus(FLEXRAM_Type *base)
FLEXRAM module gets interrupt status.

Parameters

- base – FLEXRAM base address.

static inline void FLEXRAM_ClearInterruptStatus(FLEXRAM_Type *base, uint32_t status)
FLEXRAM module clears interrupt status.

Parameters

- base – FLEXRAM base address.
- status – Status to be cleared.

static inline void FLEXRAM_EnableInterruptStatus(FLEXRAM_Type *base, uint32_t status)
FLEXRAM module enables interrupt status.

Parameters

- base – FLEXRAM base address.
- status – Status to be enabled.

static inline void FLEXRAM_DisableInterruptStatus(FLEXRAM_Type *base, uint32_t status)
FLEXRAM module disable interrupt status.

Parameters

- base – FLEXRAM base address.
- status – Status to be disabled.

static inline void FLEXRAM_EnableInterruptSignal(FLEXRAM_Type *base, uint32_t status)
FLEXRAM module enables interrupt.

Parameters

- base – FLEXRAM base address.
- status – Status interrupt to be enabled.

static inline void FLEXRAM_DisableInterruptSignal(FLEXRAM_Type *base, uint32_t status)
FLEXRAM module disables interrupt.

Parameters

- base – FLEXRAM base address.
- status – Status interrupt to be disabled.

FSL_FLEXRAM_DRIVER_VERSION
Driver version.

Flexram write/read selection.

Values:

enumerator kFLEXRAM_Read
read

enumerator kFLEXRAM_Write
write

Interrupt status flag mask.

Values:

enumerator kFLEXRAM_OCRAMAccessError
OCRAM accesses unallocated address

enumerator kFLEXRAM_DTCMAccessError
DTCM accesses unallocated address

enumerator kFLEXRAM_ITCMAccessError
ITCM accesses unallocated address

enumerator kFLEXRAM_OCRAMMagicAddrMatch
OCRAM magic address match

enumerator kFLEXRAM_DTCMMagicAddrMatch
DTCM magic address match

enumerator kFLEXRAM_ITCMMagicAddrMatch
ITCM magic address match

enumerator kFLEXRAM_OCRAMECCMultiError

enumerator kFLEXRAM_OCRAMECCSingleError

enumerator kFLEXRAM_ITCMECCMultiError

enumerator kFLEXRAM_ITCMECCSingleError

enumerator kFLEXRAM_D0TCMECCMultiError

enumerator kFLEXRAM_D0TCMECCSingleError

enumerator kFLEXRAM_D1TCMECCMultiError

enumerator kFLEXRAM_D1TCMECCSingleError

enumerator kFLEXRAM_InterruptStatusAll

enum __flexram_tcm_access_mode
FLEXRAM TCM access mode. Fast access mode expected to be finished in 1-cycle; Wait access mode expected to be finished in 2-cycle. Wait access mode is a feature of the flexram and it should be used when the CPU clock is too fast to finish TCM access in 1-cycle. Normally, fast mode is the default mode, the efficiency of the TCM access will better.

Values:

enumerator kFLEXRAM_TCMAccessFastMode
fast access mode

enumerator kFLEXRAM_TCMAccessWaitMode
wait access mode

FLEXRAM TCM support size.

Values:

enumerator kFLEXRAM_TCMSize32KB
TCM total size be 32KB

enumerator kFLEXRAM_TCMSize64KB
TCM total size be 64KB

enumerator kFLEXRAM_TCMSize128KB
TCM total size be 128KB

enumerator kFLEXRAM_TCMSize256KB
TCM total size be 256KB

enumerator kFLEXRAM_TCMSize512KB
TCM total size be 512KB

enum _flexram_memory_type
FLEXRAM memory type, such as OCRAM/ITCM/D0TCM/D1TCM.

Values:

enumerator kFLEXRAM_OCRAM
Memory type OCRAM

enumerator kFLEXRAM_ITCM
Memory type ITCM

enumerator kFLEXRAM_D0TCM
Memory type D0TCM

enumerator kFLEXRAM_D1TCM
Memory type D1TCM

typedef enum *_flexram_tcm_access_mode* flexram_tcm_access_mode_t
FLEXRAM TCM access mode. Fast access mode expected to be finished in 1-cycle; Wait access mode expected to be finished in 2-cycle. Wait access mode is a feature of the flexram and it should be used when the CPU clock is too fast to finish TCM access in 1-cycle. Normally, fast mode is the default mode, the efficiency of the TCM access will better.

typedef enum *_flexram_memory_type* flexram_memory_type_t

FLEXRAM memory type, such as OCRAM/ITCM/D0TCM/D1TCM.

typedef struct *_flexram_ecc_error_type* flexram_ecc_error_type_t

FLEXRAM error type, such as single bit error position, multi-bit error position.

typedef struct *_flexram_ocram_ecc_single_error_info* flexram_ocram_ecc_single_error_info_t

FLEXRAM ocram ecc single error information, including single error information, error address, error data.

typedef struct *_flexram_ocram_ecc_multi_error_info* flexram_ocram_ecc_multi_error_info_t

FLEXRAM ocram ecc multiple error information, including multiple error information, error address, error data.

typedef struct *_flexram_itcm_ecc_single_error_info* flexram_itcm_ecc_single_error_info_t

FLEXRAM itcm ecc single error information, including single error information, error address, error data.

typedef struct *_flexram_itcm_ecc_multi_error_info* flexram_itcm_ecc_multi_error_info_t

FLEXRAM itcm ecc multiple error information, including multiple error information, error address, error data.

typedef struct *_flexram_dtcm_ecc_single_error_info* flexram_dtcm_ecc_single_error_info_t

FLEXRAM dtcm ecc single error information, including single error information, error address, error data.

typedef struct *_flexram_dtcm_ecc_multi_error_info* flexram_dtcm_ecc_multi_error_info_t

FLEXRAM dtcm ecc multiple error information, including multiple error information, error address, error data.

static inline void FLEXRAM_SetTCMReadAccessMode(FLEXRAM_Type *base, *flexram_tcm_access_mode_t* mode)

FLEXRAM module sets TCM read access mode.

Parameters

- base – FLEXRAM base address.
- mode – Access mode.

static inline void FLEXRAM_SetTCMWriteAccessMode(FLEXRAM_Type *base, *flexram_tcm_access_mode_t* mode)

FLEXRAM module set TCM write access mode.

Parameters

- base – FLEXRAM base address.
- mode – Access mode.

static inline void FLEXRAM_EnableForceRamClockOn(FLEXRAM_Type *base, bool enable)

FLEXRAM module force ram clock on.

Parameters

- base – FLEXRAM base address.
- enable – Enable or disable clock force on.

static inline void FLEXRAM_SetOCRAMMagicAddr(FLEXRAM_Type *base, uint16_t magicAddr, uint32_t rwSel)

FLEXRAM OCRAM magic addr configuration. When read/write access hit magic address, it will generate interrupt.

Parameters

- base – FLEXRAM base address.
- magicAddr – Magic address, the actual address bits [18:3] is corresponding to the register field [16:1].
- rwSel – Read/write selection. 0 for read access while 1 for write access.

static inline void FLEXRAM_SetDTCMMagicAddr(FLEXRAM_Type *base, uint16_t magicAddr, uint32_t rwSel)

FLEXRAM DTCM magic addr configuration. When read/write access hits magic address, it will generate interrupt.

Parameters

- base – FLEXRAM base address.
- magicAddr – Magic address, the actual address bits [18:3] is corresponding to the register field [16:1].
- rwSel – Read/write selection. 0 for read access while 1 write access.

static inline void FLEXRAM_SetITCMMagicAddr(FLEXRAM_Type *base, uint16_t magicAddr, uint32_t rwSel)

FLEXRAM ITCM magic addr configuration. When read/write access hits magic address, it will generate interrupt.

Parameters

- base – FLEXRAM base address.
- magicAddr – Magic address, the actual address bits [18:3] is corresponding to the register field [16:1].
- rwSel – Read/write selection. 0 for read access while 1 for write access.

void FLEXRAM_EnableECC(FLEXRAM_Type *base, bool OcramECCEnable, bool TcmECCEnable)

FLEXRAM get ocram ecc single error information.

Parameters

- base – FLEXRAM base address.
- OcramECCEnable – ocram ecc enablement.
- TcmECCEnable – tcm(itcm/d0tcm/d1tcm) ecc enablement.

void FLEXRAM_ErrorInjection(FLEXRAM_Type *base, *flexram_memory_type_t* memory, *flexram_ecc_error_type_t* *error)

FLEXRAM ECC error injection.

Parameters

- base – FLEXRAM base address.
- memory – memory type, such as OCRAM/ITCM/DTCM.
- error – ECC error type.

void FLEXRAM_GetOcramSingleErroInfo(FLEXRAM_Type *base, *flexram_ocram_ecc_single_error_info_t* *info)

FLEXRAM get ocram ecc single error information.

Parameters

- base – FLEXRAM base address.
- info – ecc error information.

void FLEXRAM_GetOcramMultiErroInfo(FLEXRAM_Type *base, *flexram_ocram_ecc_multi_error_info_t* *info)

FLEXRAM get ocram ecc multiple error information.

Parameters

- base – FLEXRAM base address.
- info – ecc error information.

void FLEXRAM_GetItcmSingleErroInfo(FLEXRAM_Type *base, *flexram_itcm_ecc_single_error_info_t* *info)

FLEXRAM get itcm ecc single error information.

Parameters

- base – FLEXRAM base address.
- info – ecc error information.

void FLEXRAM_GetItcmMultiErroInfo(FLEXRAM_Type *base, *flexram_itcm_ecc_multi_error_info_t* *info)

FLEXRAM get itcm ecc multiple error information.

Parameters

- base – FLEXRAM base address.
- info – ecc error information.

void FLEXRAM_GetDtcmSingleErroInfo(FLEXRAM_Type *base, *flexram_dtcm_ecc_single_error_info_t* *info, uint8_t bank)

FLEXRAM get d0tcm ecc single error information.

Parameters

- base – FLEXRAM base address.
- info – ecc error information.
- bank – DTCM bank, 0 is D0TCM, 1 is D1TCM.

void FLEXRAM_GetDtcmMultiErroInfo(FLEXRAM_Type *base, *flexram_dtcm_ecc_multi_error_info_t* *info, uint8_t bank)

FLEXRAM get d0tcm ecc multiple error information.

Parameters

- base – FLEXRAM base address.
- info – ecc error information.
- bank – DTCM bank, 0 is D0TCM, 1 is D1TCM.

FLEXRAM_ECC_ERROR_DETAILED_INFO

Get ECC error detailed information.

struct __flexram__ecc__error__type

#include <fsl_flexram.h> FLEXRAM error type, such as single bit error position, multi-bit error position.

Public Members

uint8_t SingleBitPos

Bit position of the bit to inject ECC Error.

uint8_t SecondBitPos

Bit position of the second bit to inject multi-bit ECC Error

bool Fource1BitDataInversion

Force One 1-Bit Data Inversion (single-bit ECC error) on memory write access

bool FourceOneNCDataInversion

Force One Non-correctable Data Inversion(multi-bit ECC error) on memory write access

bool FourceConti1BitDataInversion

Force Continuous 1-Bit Data Inversions (single-bit ECC error) on memory write access

bool FourceContiNCDataInversion

Force Continuous Non-correctable Data Inversions (multi-bit ECC error) on memory write access

struct __flexram__ocram__ecc__single__error__info

#include <fsl_flexram.h> FLEXRAM ocram ecc single error information, including single error information, error address, error data.

Public Members

uint32_t OcramSingleErrorInfo

Ocram single error information, user should parse it by themself.

uint32_t OcramSingleErrorAddr

Ocram single error address

uint32_t OcramSingleErrorDataLSB

Ocram single error data LSB

uint32_t OcramSingleErrorDataMSB

Ocram single error data MSB

struct __flexram__ocram__ecc__multi__error__info

#include <fsl_flexram.h> FLEXRAM ocram ecc multiple error information, including multiple error information, error address, error data.

Public Members

uint32_t OcramMultiErrorInfo
: Ocram single error information, user should parse it by themself.

uint32_t OcramMultiErrorAddr
: Ocram multiple error address

uint32_t OcramMultiErrorDataLSB
: Ocram multiple error data LSB

uint32_t OcramMultiErrorDataMSB
: Ocram multiple error data MSB

struct __flexram__itcm__ecc__single__error__info
: *#include <fsl_flexram.h>* FLEXRAM itcm ecc single error information, including single error information, error address, error data.

Public Members

uint32_t ItcmSingleErrorInfo
: itcm single error information, user should parse it by themself.

uint32_t ItcmSingleErrorAddr
: itcm single error address

uint32_t ItcmSingleErrorDataLSB
: itcm single error data LSB

uint32_t ItcmSingleErrorDataMSB
: itcm single error data MSB

struct __flexram__itcm__ecc__multi__error__info
: *#include <fsl_flexram.h>* FLEXRAM itcm ecc multiple error information, including multiple error information, error address, error data.

Public Members

uint32_t ItcmMultiErrorInfo
: itcm multiple error information, user should parse it by themself.

uint32_t ItcmMultiErrorAddr
: itcm multiple error address

uint32_t ItcmMultiErrorDataLSB
: itcm multiple error data LSB

uint32_t ItcmMultiErrorDataMSB
: itcm multiple error data MSB

struct __flexram_dtcm_ecc_single_error_info
: *#include <fsl_flexram.h>* FLEXRAM dtcm ecc single error information, including single error information, error address, error data.

Public Members

uint32_t DtcmSingleErrorInfo
: dtcm single error information, user should parse it by themself.

uint32_t DtcmSingleErrorAddr
: dtcm single error address

uint32_t DtcmSingleErrorData
: dtcm single error data

struct __flexram_dtcm_ecc_multi_error_info
: *#include <fsl_flexram.h>* FLEXRAM dtcm ecc multiple error information, including multiple error information, error address, error data.

Public Members

uint32_t DtcmMultiErrorInfo
: dtcm multiple error information, user should parse it by themself.

uint32_t DtcmMultiErrorAddr
: dtcm multiple error address

uint32_t DtcmMultiErrorData
: dtcm multiple error data

2.32 FLEXSPI: Flexible Serial Peripheral Interface Driver

uint32_t FLEXSPI_GetInstance(FLEXSPI_Type *base)
: Get the instance number for FLEXSPI.

  Parameters

  - base – FLEXSPI base pointer.

status_t FLEXSPI_CheckAndClearError(FLEXSPI_Type *base, uint32_t status)
: Check and clear IP command execution errors.

  Parameters

  - base – FLEXSPI base pointer.
  - status – interrupt status.

void FLEXSPI_Init(FLEXSPI_Type *base, const *flexspi_config_t* *config)

Initializes the FLEXSPI module and internal state.

This function enables the clock for FLEXSPI and also configures the FLEXSPI with the input configure parameters. Users should call this function before any FLEXSPI operations.

Parameters

- base – FLEXSPI peripheral base address.
- config – FLEXSPI configure structure.

void FLEXSPI_GetDefaultConfig(*flexspi_config_t* *config)

Gets default settings for FLEXSPI.

Parameters

- config – FLEXSPI configuration structure.

void FLEXSPI_Deinit(FLEXSPI_Type *base)

Deinitializes the FLEXSPI module.

Clears the FLEXSPI state and FLEXSPI module registers.

Parameters

- base – FLEXSPI peripheral base address.

void FLEXSPI_UpdateDllValue(FLEXSPI_Type *base, *flexspi_device_config_t* *config, *flexspi_port_t* port)

Update FLEXSPI DLL value depending on currently flexspi root clock.

Parameters

- base – FLEXSPI peripheral base address.
- config – Flash configuration parameters.
- port – FLEXSPI Operation port.

void FLEXSPI_SetFlashConfig(FLEXSPI_Type *base, *flexspi_device_config_t* *config, *flexspi_port_t* port)

Configures the connected device parameter.

This function configures the connected device relevant parameters, such as the size, command, and so on. The flash configuration value cannot have a default value. The user needs to configure it according to the connected device.

Parameters

- base – FLEXSPI peripheral base address.
- config – Flash configuration parameters.
- port – FLEXSPI Operation port.

void FLEXSPI_SoftwareReset(FLEXSPI_Type *base)

Software reset for the FLEXSPI logic.

This function sets the software reset flags for both AHB and buffer domain and resets both AHB buffer and also IP FIFOs.

Parameters

- base – FLEXSPI peripheral base address.

static inline void FLEXSPI_Enable(FLEXSPI_Type *base, bool enable)

Enables or disables the FLEXSPI module.

Parameters

- base – FLEXSPI peripheral base address.
- enable – True means enable FLEXSPI, false means disable.

void FLEXSPI_UpdateAhbBuffersSettings(FLEXSPI_Type *base, *flexspi_ahbBuffers_ctrl_t* *ptrAhbBufferCtrl)

Update all AHB buffers' settings, including buffer size, master ID.

Parameters

- base – FLEXSPI peripheral base address.
- ptrAhbBufferCtrl – Pointer to structure *flexspi_ahbBuffers_ctrl_t* which store all AHB buffers' settings.

static inline void FLEXSPI_EnableInterrupts(FLEXSPI_Type *base, uint32_t mask)

Enables the FLEXSPI interrupts.

Parameters

- base – FLEXSPI peripheral base address.
- mask – FLEXSPI interrupt source.

static inline void FLEXSPI_DisableInterrupts(FLEXSPI_Type *base, uint32_t mask)

Disable the FLEXSPI interrupts.

Parameters

- base – FLEXSPI peripheral base address.
- mask – FLEXSPI interrupt source.

static inline void FLEXSPI_EnableTxDMA(FLEXSPI_Type *base, bool enable)

Enables or disables FLEXSPI IP Tx FIFO DMA requests.

Parameters

- base – FLEXSPI peripheral base address.
- enable – Enable flag for transmit DMA request. Pass true for enable, false for disable.

static inline void FLEXSPI_EnableRxDMA(FLEXSPI_Type *base, bool enable)

Enables or disables FLEXSPI IP Rx FIFO DMA requests.

Parameters

- base – FLEXSPI peripheral base address.
- enable – Enable flag for receive DMA request. Pass true for enable, false for disable.

static inline uint32_t FLEXSPI_GetTxFifoAddress(FLEXSPI_Type *base)

Gets FLEXSPI IP tx fifo address for DMA transfer.

Parameters

- base – FLEXSPI peripheral base address.

Return values

The – tx fifo address.

static inline uint32_t FLEXSPI_GetRxFifoAddress(FLEXSPI_Type *base)

Gets FLEXSPI IP rx fifo address for DMA transfer.

Parameters

- base – FLEXSPI peripheral base address.

Return values
The – rx fifo address.

static inline void FLEXSPI_ResetFifos(FLEXSPI_Type *base, bool txFifo, bool rxFifo)

Clears the FLEXSPI IP FIFO logic.

Parameters

- base – FLEXSPI peripheral base address.
- txFifo – Pass true to reset TX FIFO.
- rxFifo – Pass true to reset RX FIFO.

static inline void FLEXSPI_GetFifoCounts(FLEXSPI_Type *base, size_t *txCount, size_t *rxCount)

Gets the valid data entries in the FLEXSPI FIFOs.

Parameters

- base – FLEXSPI 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 uint32_t FLEXSPI_GetInterruptStatusFlags(FLEXSPI_Type *base)

Get the FLEXSPI interrupt status flags.

Parameters

- base – FLEXSPI peripheral base address.

Return values
interrupt – status flag, use status flag to AND *flexspi_flags_t* could get the related status.

static inline void FLEXSPI_ClearInterruptStatusFlags(FLEXSPI_Type *base, uint32_t mask)

Get the FLEXSPI interrupt status flags.

Parameters

- base – FLEXSPI peripheral base address.
- mask – FLEXSPI interrupt source.

static inline *flexspi_arb_command_source_t* FLEXSPI_GetArbitratorCommandSource(FLEXSPI_Type *base)

Gets the trigger source of current command sequence granted by arbitrator.

Parameters

- base – FLEXSPI peripheral base address.

Return values
trigger – source of current command sequence.

static inline *flexspi_ip_error_code_t* FLEXSPI_GetIPCommandErrorCode(FLEXSPI_Type *base, uint8_t *index)

Gets the error code when IP command error detected.

Parameters

- base – FLEXSPI peripheral base address.
- index – Pointer to a uint8_t type variable to receive the sequence index when error detected.

Return values
error – code when IP command error detected.

static inline *flexspi_ahb_error_code_t* FLEXSPI_GetAHBCommandErrorCode(FLEXSPI_Type *base, uint8_t *index)

Gets the error code when AHB command error detected.

Parameters

- base – FLEXSPI peripheral base address.
- index – Pointer to a uint8_t type variable to receive the sequence index when error detected.

Return values
error – code when AHB command error detected.

static inline bool FLEXSPI_GetBusIdleStatus(FLEXSPI_Type *base)

Returns whether the bus is idle.

Parameters

- base – FLEXSPI peripheral base address.

Return values

- true – Bus is idle.
- false – Bus is busy.

void FLEXSPI_UpdateRxSampleClock(FLEXSPI_Type *base, *flexspi_read_sample_clock_t* clockSource)

Update read sample clock source.

Parameters

- base – FLEXSPI peripheral base address.
- clockSource – clockSource of type *flexspi_read_sample_clock_t*

void FLEXSPI_UpdateLUT(FLEXSPI_Type *base, uint32_t index, const uint32_t *cmd, uint32_t count)

Updates the LUT table.

Parameters

- base – FLEXSPI peripheral base address.
- index – From which index start to update. It could be any index of the LUT table, which also allows user to update command content inside a command. Each command consists of up to 8 instructions and occupy 4*32-bit memory.
- cmd – Command sequence array.
- count – Number of sequences.

static inline void FLEXSPI_WriteData(FLEXSPI_Type *base, uint32_t data, uint8_t fifoIndex)

Writes data into FIFO.

Parameters

- base – FLEXSPI peripheral base address
- data – The data bytes to send
- fifoIndex – Destination fifo index.

static inline uint32_t FLEXSPI_ReadData(FLEXSPI_Type *base, uint8_t fifoIndex)

Receives data from data FIFO.

Parameters

- base – FLEXSPI peripheral base address
- fifoIndex – Source fifo index.

Returns

The data in the FIFO.

status_t FLEXSPI_WriteBlocking(FLEXSPI_Type *base, uint8_t *buffer, size_t size)

Sends a buffer of data bytes using blocking method.

Note: This function blocks via polling until all bytes have been sent.

Parameters

- base – FLEXSPI peripheral base address
- buffer – The data bytes to send
- size – The number of data bytes to send

Return values

- kStatus_Success – write success without error
- kStatus_FLEXSPI_SequenceExecutionTimeout – sequence execution timeout
- kStatus_FLEXSPI_IpCommandSequenceError – IP command sequence error detected
- kStatus_FLEXSPI_IpCommandGrantTimeout – IP command grant timeout detected

status_t FLEXSPI_ReadBlocking(FLEXSPI_Type *base, uint8_t *buffer, size_t size)

Receives a buffer of data bytes using a blocking method.

Note: This function blocks via polling until all bytes have been sent.

Parameters

- base – FLEXSPI peripheral base address
- buffer – The data bytes to send
- size – The number of data bytes to receive

Return values

- kStatus_Success – read success without error
- kStatus_FLEXSPI_SequenceExecutionTimeout – sequence execution timeout
- kStatus_FLEXSPI_IpCommandSequenceError – IP command sequencen error detected
- kStatus_FLEXSPI_IpCommandGrantTimeout – IP command grant timeout detected

status_t FLEXSPI_TransferBlocking(FLEXSPI_Type *base, *flexspi_transfer_t* *xfer)

Execute command to transfer a buffer data bytes using a blocking method.

Parameters

- base – FLEXSPI peripheral base address
- xfer – pointer to the transfer structure.

Return values

- kStatus_Success – command transfer success without error
- kStatus_FLEXSPI_SequenceExecutionTimeout – sequence execution timeout
- kStatus_FLEXSPI_IpCommandSequenceError – IP command sequence error detected
- kStatus_FLEXSPI_IpCommandGrantTimeout – IP command grant timeout detected

void FLEXSPI_TransferCreateHandle(FLEXSPI_Type *base, *flexspi_handle_t* *handle, *flexspi_transfer_callback_t* callback, void *userData)

Initializes the FLEXSPI handle which is used in transactional functions.

Parameters

- base – FLEXSPI peripheral base address.
- handle – pointer to flexspi_handle_t structure to store the transfer state.
- callback – pointer to user callback function.
- userData – user parameter passed to the callback function.

status_t FLEXSPI_TransferNonBlocking(FLEXSPI_Type *base, *flexspi_handle_t* *handle, *flexspi_transfer_t* *xfer)

Performs a interrupt non-blocking transfer on the FLEXSPI bus.

Note: Calling the API returns immediately after transfer initiates. The user needs to call FLEXSPI_GetTransferCount to poll the transfer status to check whether the transfer is finished. If the return status is not kStatus_FLEXSPI_Busy, the transfer is finished. For FLEXSPI_Read, the dataSize should be multiple of rx watermark level, or FLEXSPI could not read data properly.

Parameters

- base – FLEXSPI peripheral base address.
- handle – pointer to flexspi_handle_t structure which stores the transfer state.
- xfer – pointer to flexspi_transfer_t structure.

Return values

- kStatus_Success – Successfully start the data transmission.
- kStatus_FLEXSPI_Busy – Previous transmission still not finished.

status_t FLEXSPI_TransferGetCount(FLEXSPI_Type *base, *flexspi_handle_t* *handle, size_t *count)

Gets the master transfer status during a interrupt non-blocking transfer.

Parameters

- base – FLEXSPI peripheral base address.
- handle – pointer to flexspi_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_Success – Successfully return the count.

void FLEXSPI_TransferAbort(FLEXSPI_Type *base, *flexspi_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 – FLEXSPI peripheral base address.
- handle – pointer to flexspi_handle_t structure which stores the transfer state

void FLEXSPI_TransferHandleIRQ(FLEXSPI_Type *base, *flexspi_handle_t* *handle)

Master interrupt handler.

Parameters

- base – FLEXSPI peripheral base address.
- handle – pointer to flexspi_handle_t structure.

FSL_FLEXSPI_DRIVER_VERSION

FLEXSPI driver version.

Status structure of FLEXSPI.

Values:

enumerator kStatus_FLEXSPI_Busy

FLEXSPI is busy

enumerator kStatus_FLEXSPI_SequenceExecutionTimeout

Sequence execution timeout error occurred during FLEXSPI transfer.

enumerator kStatus_FLEXSPI_IpCommandSequenceError

IP command Sequence execution timeout error occurred during FLEXSPI transfer.

enumerator kStatus_FLEXSPI_IpCommandGrantTimeout

IP command grant timeout error occurred during FLEXSPI transfer.

CMD definition of FLEXSPI, use to form LUT instruction, _flexspi_command.

Values:

enumerator kFLEXSPI_Command_STOP
Stop execution, deassert CS.

enumerator kFLEXSPI_Command_SDR
Transmit Command code to Flash, using SDR mode.

enumerator kFLEXSPI_Command_RADDR_SDR
Transmit Row Address to Flash, using SDR mode.

enumerator kFLEXSPI_Command_CADDR_SDR
Transmit Column Address to Flash, using SDR mode.

enumerator kFLEXSPI_Command_MODE1_SDR
Transmit 1-bit Mode bits to Flash, using SDR mode.

enumerator kFLEXSPI_Command_MODE2_SDR
Transmit 2-bit Mode bits to Flash, using SDR mode.

enumerator kFLEXSPI_Command_MODE4_SDR
Transmit 4-bit Mode bits to Flash, using SDR mode.

enumerator kFLEXSPI_Command_MODE8_SDR
Transmit 8-bit Mode bits to Flash, using SDR mode.

enumerator kFLEXSPI_Command_WRITE_SDR
Transmit Programming Data to Flash, using SDR mode.

enumerator kFLEXSPI_Command_READ_SDR
Receive Read Data from Flash, using SDR mode.

enumerator kFLEXSPI_Command_LEARN_SDR
Receive Read Data or Preamble bit from Flash, SDR mode.

enumerator kFLEXSPI_Command_DATSZ_SDR
Transmit Read/Program Data size (byte) to Flash, SDR mode.

enumerator kFLEXSPI_Command_DUMMY_SDR
Leave data lines undriven by FlexSPI controller.

enumerator kFLEXSPI_Command_DUMMY_RWDS_SDR
Leave data lines undriven by FlexSPI controller, dummy cycles decided by RWDS.

enumerator kFLEXSPI_Command_DDR
Transmit Command code to Flash, using DDR mode.

enumerator kFLEXSPI_Command_RADDR_DDR
Transmit Row Address to Flash, using DDR mode.

enumerator kFLEXSPI_Command_CADDR_DDR
Transmit Column Address to Flash, using DDR mode.

enumerator kFLEXSPI_Command_MODE1_DDR
Transmit 1-bit Mode bits to Flash, using DDR mode.

enumerator kFLEXSPI_Command_MODE2_DDR
Transmit 2-bit Mode bits to Flash, using DDR mode.

enumerator kFLEXSPI_Command_MODE4_DDR
Transmit 4-bit Mode bits to Flash, using DDR mode.

enumerator kFLEXSPI_Command_MODE8_DDR
Transmit 8-bit Mode bits to Flash, using DDR mode.

enumerator kFLEXSPI_Command_WRITE_DDR
Transmit Programming Data to Flash, using DDR mode.

enumerator kFLEXSPI_Command_READ_DDR
Receive Read Data from Flash, using DDR mode.

enumerator kFLEXSPI_Command_LEARN_DDR
Receive Read Data or Preamble bit from Flash, DDR mode.

enumerator kFLEXSPI_Command_DATSZ_DDR
Transmit Read/Program Data size (byte) to Flash, DDR mode.

enumerator kFLEXSPI_Command_DUMMY_DDR
Leave data lines undriven by FlexSPI controller.

enumerator kFLEXSPI_Command_DUMMY_RWDS_DDR
Leave data lines undriven by FlexSPI controller, dummy cycles decided by RWDS.

enumerator kFLEXSPI_Command_JUMP_ON_CS
Stop execution, deassert CS and save operand[7:0] as the instruction start pointer for next sequence

enum _flexspi_pad
pad definition of FLEXSPI, use to form LUT instruction.

Values:

enumerator kFLEXSPI_1PAD
Transmit command/address and transmit/receive data only through DATA0/DATA1.

enumerator kFLEXSPI_2PAD
Transmit command/address and transmit/receive data only through DATA[1:0].

enumerator kFLEXSPI_4PAD
Transmit command/address and transmit/receive data only through DATA[3:0].

enumerator kFLEXSPI_8PAD
Transmit command/address and transmit/receive data only through DATA[7:0].

enum _flexspi_flags
FLEXSPI interrupt status flags.
Values:

enumerator kFLEXSPI_SequenceExecutionTimeoutFlag
Sequence execution timeout.

enumerator kFLEXSPI_AhbBusErrorFlag
AHB Bus error flag.

enumerator kFLEXSPI_SckStoppedBecauseTxEmptyFlag
SCK is stopped during command sequence because Async TX FIFO empty.

enumerator kFLEXSPI_SckStoppedBecauseRxFullFlag
SCK is stopped during command sequence because Async RX FIFO full.

enumerator kFLEXSPI_IpTxFifoWatermarkEmptyFlag
IP TX FIFO WaterMark empty.

enumerator kFLEXSPI_IpRxFifoWatermarkAvailableFlag
IP RX FIFO WaterMark available.

enumerator kFLEXSPI_AhbCommandSequenceErrorFlag
AHB triggered Command Sequences Error.

enumerator kFLEXSPI_IpCommandSequenceErrorFlag
IP triggered Command Sequences Error.

enumerator kFLEXSPI_AhbCommandGrantTimeoutFlag
AHB triggered Command Sequences Grant Timeout.

enumerator kFLEXSPI_IpCommandGrantTimeoutFlag
IP triggered Command Sequences Grant Timeout.

enumerator kFLEXSPI_IpCommandExecutionDoneFlag
IP triggered Command Sequences Execution finished.

enumerator kFLEXSPI_AllInterruptFlags
All flags.

enum _flexspi_read_sample_clock
FLEXSPI sample clock source selection for Flash Reading.
Values:

enumerator kFLEXSPI_ReadSampleClkLoopbackInternally
Dummy Read strobe generated by FlexSPI Controller and loopback internally.

enumerator kFLEXSPI_ReadSampleClkLoopbackFromDqsPad
Dummy Read strobe generated by FlexSPI Controller and loopback from DQS pad.

enumerator kFLEXSPI_ReadSampleClkLoopbackFromSckPad
SCK output clock and loopback from SCK pad.

enumerator kFLEXSPI_ReadSampleClkExternalInputFromDqsPad
Flash provided Read strobe and input from DQS pad.

enum _flexspi_cs_interval_cycle_unit
FLEXSPI interval unit for flash device select.
Values:

enumerator kFLEXSPI_CsIntervalUnit1SckCycle
Chip selection interval: CSINTERVAL * 1 serial clock cycle.

enumerator kFLEXSPI_CsIntervalUnit256SckCycle
Chip selection interval: CSINTERVAL * 256 serial clock cycle.

enum _flexspi_ahb_write_wait_unit
FLEXSPI AHB wait interval unit for writing.
Values:

enumerator kFLEXSPI_AhbWriteWaitUnit2AhbCycle
AWRWAIT unit is 2 ahb clock cycle.

enumerator kFLEXSPI_AhbWriteWaitUnit8AhbCycle
AWRWAIT unit is 8 ahb clock cycle.

enumerator kFLEXSPI_AhbWriteWaitUnit32AhbCycle
AWRWAIT unit is 32 ahb clock cycle.

enumerator kFLEXSPI_AhbWriteWaitUnit128AhbCycle
AWRWAIT unit is 128 ahb clock cycle.

enumerator kFLEXSPI_AhbWriteWaitUnit512AhbCycle
AWRWAIT unit is 512 ahb clock cycle.

enumerator kFLEXSPI_AhbWriteWaitUnit2048AhbCycle
AWRWAIT unit is 2048 ahb clock cycle.

enumerator kFLEXSPI_AhbWriteWaitUnit8192AhbCycle
AWRWAIT unit is 8192 ahb clock cycle.

enumerator kFLEXSPI_AhbWriteWaitUnit32768AhbCycle
AWRWAIT unit is 32768 ahb clock cycle.

enum _flexspi_ip_error_code
Error Code when IP command Error detected.
Values:

enumerator kFLEXSPI_IpCmdErrorNoError
No error.

enumerator kFLEXSPI_IpCmdErrorJumpOnCsInIpCmd
IP command with JMP_ON_CS instruction used.

enumerator kFLEXSPI_IpCmdErrorUnknownOpCode
Unknown instruction opcode in the sequence.

enumerator kFLEXSPI_IpCmdErrorSdrDummyInDdrSequence
Instruction DUMMY_SDR/DUMMY_RWDS_SDR used in DDR sequence.

enumerator kFLEXSPI_IpCmdErrorDdrDummyInSdrSequence
Instruction DUMMY_DDR/DUMMY_RWDS_DDR used in SDR sequence.

enumerator kFLEXSPI_IpCmdErrorInvalidAddress
Flash access start address exceed the whole flash address range (A1/A2/B1/B2).

enumerator kFLEXSPI_IpCmdErrorSequenceExecutionTimeout
Sequence execution timeout.

enumerator kFLEXSPI_IpCmdErrorFlashBoundaryAcrosss
Flash boundary crossed.

enum _flexspi_ahb_error_code
Error Code when AHB command Error detected.
Values:

enumerator kFLEXSPI_AhbCmdErrorNoError
No error.

enumerator kFLEXSPI_AhbCmdErrorJumpOnCsInWriteCmd
AHB Write command with JMP_ON_CS instruction used in the sequence.

enumerator kFLEXSPI_AhbCmdErrorUnknownOpCode
Unknown instruction opcode in the sequence.

enumerator kFLEXSPI_AhbCmdErrorSdrDummyInDdrSequence
Instruction DUMMY_SDR/DUMMY_RWDS_SDR used in DDR sequence.

enumerator kFLEXSPI_AhbCmdErrorDdrDummyInSdrSequence
Instruction DUMMY_DDR/DUMMY_RWDS_DDR used in SDR sequence.

enumerator kFLEXSPI_AhbCmdSequenceExecutionTimeout
Sequence execution timeout.

enum _flexspi_port
FLEXSPI operation port select.
Values:

enumerator kFLEXSPI_PortA1
Access flash on A1 port.

enumerator kFLEXSPI_PortA2
Access flash on A2 port.

enumerator kFLEXSPI_PortCount

enum _flexspi_arb_command_source
Trigger source of current command sequence granted by arbitrator.
Values:

enumerator kFLEXSPI_AhbReadCommand

enumerator kFLEXSPI_AhbWriteCommand

enumerator kFLEXSPI_IpCommand

enumerator kFLEXSPI_SuspendedCommand

enum _flexspi_command_type
Command type.
Values:

enumerator kFLEXSPI_Command
FlexSPI operation: Only command, both TX and Rx buffer are ignored.

enumerator kFLEXSPI_Config
FlexSPI operation: Configure device mode, the TX fifo size is fixed in LUT.

enumerator kFLEXSPI_Read

enumerator kFLEXSPI_Write

typedef enum *_flexspi_pad* flexspi_pad_t
pad definition of FLEXSPI, use to form LUT instruction.

typedef enum *_flexspi_flags* flexspi_flags_t
FLEXSPI interrupt status flags.

typedef enum *_flexspi_read_sample_clock* flexspi_read_sample_clock_t
FLEXSPI sample clock source selection for Flash Reading.

typedef enum *_flexspi_cs_interval_cycle_unit* flexspi_cs_interval_cycle_unit_t
FLEXSPI interval unit for flash device select.

typedef enum *_flexspi_ahb_write_wait_unit* flexspi_ahb_write_wait_unit_t
FLEXSPI AHB wait interval unit for writing.

typedef enum *_flexspi_ip_error_code* flexspi_ip_error_code_t
Error Code when IP command Error detected.

typedef enum *_flexspi_ahb_error_code* flexspi_ahb_error_code_t
Error Code when AHB command Error detected.

typedef enum *_flexspi_port* flexspi_port_t
FLEXSPI operation port select.

typedef enum *_flexspi_arb_command_source* flexspi_arb_command_source_t
Trigger source of current command sequence granted by arbitrator.

typedef enum *_flexspi_command_type* flexspi_command_type_t
Command type.

typedef struct *_flexspi_ahbBuffer_config* flexspi_ahbBuffer_config_t

typedef struct *_flexspi_ahbBuffers_ctrl* flexspi_ahbBuffers_ctrl_t
Structure to control all AHB buffers.

typedef struct *_flexspi_config* flexspi_config_t
FLEXSPI configuration structure.

typedef struct *_flexspi_device_config* flexspi_device_config_t
External device configuration items.

typedef struct *_flexspi_transfer* flexspi_transfer_t
Transfer structure for FLEXSPI.

typedef struct *_flexspi_handle* flexspi_handle_t

typedef void (*flexspi_transfer_callback_t)(FLEXSPI_Type *base, *flexspi_handle_t* *handle, *status_t* status, void *userData)
FLEXSPI transfer callback function.

typedef struct *_flexspi_addr_map_config* flexspi_addr_map_config_t
Address mapping configuration structure.

FSL_FEATURE_FLEXSPI_AHB_BUFFER_COUNT

FLEXSPI_LUT_SEQ(cmd0, pad0, op0, cmd1, pad1, op1)

Formula to form FLEXSPI instructions in LUT table.

struct _flexspi_ahbBuffer_config

#include <fsl_flexspi.h>

Public Members

uint8_t priority

This priority for AHB Master Read which this AHB RX Buffer is assigned.

uint8_t masterIndex

AHB Master ID the AHB RX Buffer is assigned.

uint16_t bufferSize

AHB buffer size in byte.

bool enablePrefetch

AHB Read Prefetch Enable for current AHB RX Buffer corresponding Master, allows prefetch disable/enable separately for each master.

struct _flexspi_ahbBuffers_ctrl

#include <fsl_flexspi.h> Structure to control all AHB buffers.

Public Members

flexspi_ahbBuffer_config_t buffer[FSL_FEATURE_FLEXSPI_AHB_BUFFER_COUNTn(0)]

Configurations of all AHB buffers.

struct _flexspi_config

#include <fsl_flexspi.h> FLEXSPI configuration structure.

Public Members

uint8_t clockDiv

FLEXSPI serial root clock divider.

flexspi_read_sample_clock_t rxSampleClock

Sample Clock source selection for Flash Reading.

bool enableSckFreeRunning

Enable/disable SCK output free-running.

bool enableDoze

Enable/disable doze mode support.

bool enableHalfSpeedAccess

Enable/disable divide by 2 of the clock for half speed commands.

flexspi_read_sample_clock_t rxSampleClockPortB

Sample Clock source_b selection for Flash Reading.

bool rxSampleClockDiff

Sample Clock source or source_b selection for Flash Reading.

bool enableSameConfigForAll

Enable/disable same configuration for all connected devices when enabled, same configuration in FLASHA1CRx is applied to all.

uint16_t seqTimeoutCycle

Timeout wait cycle for command sequence execution, timeout after ahbGrantTimeoutCyle*1024 serial root clock cycles.

uint8_t ipGrantTimeoutCycle

Timeout wait cycle for IP command grant, timeout after ipGrantTimeoutCycle*1024 AHB clock cycles.

uint8_t txWatermark

FLEXSPI IP transmit watermark value.

uint8_t rxWatermark

FLEXSPI receive watermark value.

struct __flexspi__device__config

#include <fsl_flexspi.h> External device configuration items.

Public Members

uint32_t flexspiRootClk

FLEXSPI serial root clock.

bool isSck2Enabled

FLEXSPI use SCK2.

uint32_t flashSize

Flash size in KByte.

bool addressShift

Address shift.

flexspi_cs_interval_cycle_unit_t CSIntervalUnit
: CS interval unit, 1 or 256 cycle.

uint16_t CSInterval
: CS line assert interval, multiply CS interval unit to get the CS line assert interval cycles.

uint8_t CSHoldTime
: CS line hold time.

uint8_t CSSetupTime
: CS line setup time.

uint8_t dataValidTime
: Data valid time for external device.

uint8_t columnspace
: Column space size.

bool enableWordAddress
: If enable word address.

uint8_t AWRSeqIndex
: Sequence ID for AHB write command.

uint8_t AWRSeqNumber
: Sequence number for AHB write command.

uint8_t ARDSeqIndex
: Sequence ID for AHB read command.

uint8_t ARDSeqNumber
: Sequence number for AHB read command.

flexspi_ahb_write_wait_unit_t AHBWriteWaitUnit
: AHB write wait unit.

uint16_t AHBWriteWaitInterval
: AHB write wait interval, multiply AHB write interval unit to get the AHB write wait cycles.

bool enableWriteMask
: Enable/Disable FLEXSPI drive DQS pin as write mask when writing to external device.

bool isFroClockSource
: Is FRO clock source or not.

struct __flexspi__transfer
: *#include <fsl_flexspi.h>* Transfer structure for FLEXSPI.

Public Members

uint32_t deviceAddress
: Operation device address.

flexspi_port_t port
: Operation port.

flexspi_command_type_t cmdType
: Execution command type.

uint8_t seqIndex
: Sequence ID for command.

uint8_t SeqNumber
: Sequence number for command.

uint32_t *data
: Data buffer.

size_t dataSize
: Data size in bytes.

struct __flexspi_handle
: *#include <fsl_flexspi.h>* Transfer handle structure for FLEXSPI.

Public Members

uint32_t state
: Internal state for FLEXSPI transfer

uint8_t *data
: Data buffer.

size_t dataSize
: Remaining Data size in bytes.

size_t transferTotalSize
: Total Data size in bytes.

flexspi_transfer_callback_t completionCallback
: Callback for users while transfer finish or error occurred

void *userData
: FLEXSPI callback function parameter.

struct __flexspi_addr_map_config
: *#include <fsl_flexspi.h>* Address mapping configuration structure.

Public Members

uint32_t addrStart

Remapping start address.

uint32_t addrEnd

Remapping end address.

uint32_t addrOffset

Address offset.

bool remapEnable

Enable address remapping.

struct ahbConfig

Public Members

uint8_t ahbGrantTimeoutCycle

Timeout wait cycle for AHB command grant, timeout after ahbGrantTimeoutCyle*1024 AHB clock cycles.

uint16_t ahbBusTimeoutCycle

Timeout wait cycle for AHB read/write access, timeout after ahbBusTimeoutCycle*1024 AHB clock cycles.

uint8_t resumeWaitCycle

Wait cycle for idle state before suspended command sequence resume, timeout after ahbBusTimeoutCycle AHB clock cycles.

bool disableAhbReadResume

True: Suspended AHB read prefetch does not resume once aborted; **False:** Suspended AHB read prefetch resumes when AHB is IDLE.

flexspi_ahbBuffer_config_t buffer[FSL_FEATURE_FLEXSPI_AHB_BUFFER_COUNTn(0)]

AHB buffer size.

bool enableClearAHBBufferOpt

Enable/disable automatically clean AHB RX Buffer and TX Buffer when FLEXSPI returns STOP mode ACK.

bool enableReadAddressOpt

Enable/disable remove AHB read burst start address alignment limitation. when enable, there is no AHB read burst start address alignment limitation.

bool enableAHBPrefetch

Enable/disable AHB read prefetch feature, when enabled, FLEXSPI will fetch more data than current AHB burst.

bool enableAHBBufferable

Enable/disable AHB bufferable write access support, when enabled, FLEXSPI return before waiting for command execution finished.

bool enableAHBCachable

Enable AHB bus cachable read access support.

2.33 FLEXSPI eDMA Driver

void FLEXSPI_TransferCreateHandleEDMA(FLEXSPI_Type *base, *flexspi_edma_handle_t* *handle, *flexspi_edma_callback_t* callback, void *userData, *edma_handle_t* *txDmaHandle, *edma_handle_t* *rxDmaHandle)

Initializes the FLEXSPI handle for transfer which is used in transactional functions and set the callback.

Parameters

- base – FLEXSPI peripheral base address
- handle – Pointer to flexspi_edma_handle_t structure
- callback – FLEXSPI callback, NULL means no callback.
- userData – User callback function data.
- txDmaHandle – User requested DMA handle for TX DMA transfer.
- rxDmaHandle – User requested DMA handle for RX DMA transfer.

void FLEXSPI_TransferUpdateSizeEDMA(FLEXSPI_Type *base, *flexspi_edma_handle_t* *handle, *flexspi_edma_transfer_nsize_t* nsize)

Update FLEXSPI EDMA transfer source data transfer size(SSIZE) and destination data transfer size(DSIZE).

See also:

flexspi_edma_transfer_nsize_t .

Parameters

- base – FLEXSPI peripheral base address
- handle – Pointer to flexspi_edma_handle_t structure
- nsize – FLEXSPI DMA transfer data transfer size(SSIZE/DSIZE), by default the size is *kFLEXPSI_EDMAnSize1Bytes(one byte)*.

status_t FLEXSPI_TransferEDMA(FLEXSPI_Type *base, *flexspi_edma_handle_t* *handle, *flexspi_transfer_t* *xfer)

Transfers FLEXSPI data using an eDMA non-blocking method.

This function writes/receives data to/from the FLEXSPI transmit/receive FIFO. This function is non-blocking.

Parameters

- base – FLEXSPI peripheral base address.
- handle – Pointer to flexspi_edma_handle_t structure

- xfer – FLEXSPI transfer structure.

Return values

- kStatus_FLEXSPI_Busy – FLEXSPI is busy transfer.
- kStatus_InvalidArgument – The watermark configuration is invalid, the watermark should be power of 2 to do successfully EDMA transfer.
- kStatus_Success – FLEXSPI successfully start edma transfer.

void FLEXSPI_TransferAbortEDMA(FLEXSPI_Type *base, *flexspi_edma_handle_t* *handle)

Aborts the transfer data using eDMA.

This function aborts the transfer data using eDMA.

Parameters

- base – FLEXSPI peripheral base address.
- handle – Pointer to flexspi_edma_handle_t structure

status_t FLEXSPI_TransferGetTransferCountEDMA(FLEXSPI_Type *base, *flexspi_edma_handle_t* *handle, size_t *count)

Gets the transferred counts of transfer.

Parameters

- base – FLEXSPI peripheral base address.
- handle – Pointer to flexspi_edma_handle_t structure.
- count – Bytes transfer.

Return values

- kStatus_Success – Succeed get the transfer count.
- kStatus_NoTransferInProgress – There is not a non-blocking transaction currently in progress.

FSL_FLEXSPI_EDMA_DRIVER_VERSION

FLEXSPI EDMA driver version.

enum _flexspi_edma_ntransfer_size

eDMA transfer configuration

Values:

enumerator kFLEXPSI_EDMAnSize1Bytes

Source/Destination data transfer size is 1 byte every time

enumerator kFLEXPSI_EDMAnSize2Bytes

Source/Destination data transfer size is 2 bytes every time

enumerator kFLEXPSI_EDMAnSize4Bytes

Source/Destination data transfer size is 4 bytes every time

enumerator kFLEXPSI_EDMAnSize8Bytes

Source/Destination data transfer size is 8 bytes every time

enumerator kFLEXPSI_EDMAnSize32Bytes

Source/Destination data transfer size is 32 bytes every time

typedef struct *_flexspi_edma_handle* flexspi_edma_handle_t

typedef void (*flexspi_edma_callback_t)(FLEXSPI_Type *base, *flexspi_edma_handle_t* *handle, *status_t* status, void *userData)

FLEXSPI eDMA transfer callback function for finish and error.

typedef enum *_flexspi_edma_ntransfer_size* flexspi_edma_transfer_nsize_t

eDMA transfer configuration

struct _flexspi_edma_handle

#include <fsl_flexspi_edma.h> FLEXSPI DMA transfer handle, users should not touch the content of the handle.

Public Members

edma_handle_t *txDmaHandle

eDMA handler for FLEXSPI Tx.

edma_handle_t *rxDmaHandle

eDMA handler for FLEXSPI Rx.

size_t transferSize

Bytes need to transfer.

flexspi_edma_transfer_nsize_t nsize

eDMA SSIZE/DSIZE in each transfer.

uint8_t nbytes

eDMA minor byte transfer count initially configured.

uint8_t count

The transfer data count in a DMA request.

uint32_t state

Internal state for FLEXSPI eDMA transfer.

flexspi_edma_callback_t completionCallback

A callback function called after the eDMA transfer is finished.

void *userData

User callback parameter

2.34 GPC: General Power Controller Driver

FSL_GPC_DRIVER_VERSION

GPC driver version 2.1.1.

static inline void GPC_AllowIRQs(GPC_Type *base)

Allow all the IRQ/Events within the charge of GPC.

Parameters

- base – GPC peripheral base address.

static inline void GPC_DisallowIRQs(GPC_Type *base)

Disallow all the IRQ/Events within the charge of GPC.

Parameters

- base – GPC peripheral base address.

void GPC_EnableIRQ(GPC_Type *base, uint32_t irqId)

Enable the IRQ.

Parameters

- base – GPC peripheral base address.
- irqId – ID number of IRQ to be enabled, available range is 32-159. 0-31 is available in some platforms.

void GPC_DisableIRQ(GPC_Type *base, uint32_t irqId)

Disable the IRQ.

Parameters

- base – GPC peripheral base address.
- irqId – ID number of IRQ to be disabled, available range is 32-159. 0-31 is available in some platforms.

bool GPC_GetIRQStatusFlag(GPC_Type *base, uint32_t irqId)

Get the IRQ/Event flag.

Parameters

- base – GPC peripheral base address.
- irqId – ID number of IRQ to be enabled, available range is 32-159. 0-31 is available in some platforms.

Returns

Indicated IRQ/Event is asserted or not.

static inline void GPC_RequestL2CachePowerDown(GPC_Type *base, bool enable)

L2 Cache Power Gate Enable.

This function configures the L2 cache if it will keep power when in low power mode. When the L2 cache power is OFF, L2 cache will be power down once when CPU core is power down and will be hardware invalidated automatically when CPU core is re-power up. When the L2 cache power is ON, L2 cache will keep power on even if CPU core is power down and will not be hardware invalidated. When CPU core is re-power up, the default setting is OFF.

Parameters

- base – GPC peripheral base address.
- enable – Enable the request or not.

static inline void GPC_RequestPdram0PowerDown(GPC_Type *base, bool enable)

FLEXRAM PDRAM0 Power Gate Enable.

This function configures the FLEXRAM PDRAM0 if it will keep power when cpu core is power down. When the PDRAM0 Power is 1, PDRAM0 will be power down once when CPU core is power down. When the PDRAM0 Power is 0, PDRAM0 will keep power on even if CPU core is power down. When CPU core is re-power up, the default setting is 1.

Parameters

- base – GPC peripheral base address.
- enable – Enable the request or not.

static inline void GPC_RequestVADCPowerDown(GPC_Type *base, bool enable)

VADC power down.

This function requests the VADC power down.

Parameters

- base – GPC peripheral base address.
- enable – Enable the request or not.

static inline bool GPC_GetVADCPowerDownFlag(GPC_Type *base)

Checks if the VADC is power off.

Parameters

- base – GPC peripheral base address.

Returns

Whether the VADC is power off or not.

static inline bool GPC_HasDVFS0ChangeRequest(GPC_Type *base)

Checks if the DVFS0 is requesting for frequency/voltage update.

Parameters

- base – GPC peripheral base address.

Returns

Whether the DVFS0 is requesting for frequency/voltage update.

static inline void GPC_RequestDisplayPowerOn(GPC_Type *base, bool enable)

Requests the display power switch sequence.

Parameters

- base – GPC peripheral base address.
- enable – Enable the power on sequence, or the power down sequence.

static inline void GPC_RequestMEGAPowerOn(GPC_Type *base, bool enable)

Requests the MEGA power switch sequence.

Parameters

- base – GPC peripheral base address.
- enable – Enable the power on sequence, or the power down sequence.

2.35 GPIO: General-Purpose Input/Output Driver

void GPIO_PinInit(GPIO_Type *base, uint32_t pin, const *gpio_pin_config_t* *Config)

Initializes the GPIO peripheral according to the specified parameters in the initConfig.

Parameters

- base – GPIO base pointer.
- pin – Specifies the pin number
- Config – pointer to a *gpio_pin_config_t* structure that contains the configuration information.

void GPIO_PinWrite(GPIO_Type *base, uint32_t pin, uint8_t output)

Sets the output level of the individual GPIO pin to logic 1 or 0.

Parameters

- base – GPIO base pointer.
- pin – GPIO port pin number.
- output – GPIOpin output logic level.
  - 0: corresponding pin output low-logic level.
  - 1: corresponding pin output high-logic level.

static inline void GPIO_WritePinOutput(GPIO_Type *base, uint32_t pin, uint8_t output)

Sets the output level of the individual GPIO pin to logic 1 or 0.

Deprecated:

Do not use this function. It has been superceded by *GPIO_PinWrite*.

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 (GPIO1, GPIO2, GPIO3, and so on.)
- mask – GPIO pin number macro

static inline void GPIO_SetPinsOutput(GPIO_Type *base, uint32_t mask)

Sets the output level of the multiple GPIO pins to the logic 1.

Deprecated:

Do not use this function. It has been superceded by *GPIO_PortSet*.

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 (GPIO1, GPIO2, GPIO3, and so on.)
- mask – GPIO pin number macro

static inline void GPIO_ClearPinsOutput(GPIO_Type *base, uint32_t mask)

Sets the output level of the multiple GPIO pins to the logic 0.

Deprecated:

Do not use this function. It has been superceded by *GPIO_PortClear*.

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 (GPIO1, GPIO2, GPIO3, 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 base pointer.
- pin – GPIO port pin number.

Return values

GPIO – port input value.

static inline uint32_t GPIO_ReadPinInput(GPIO_Type *base, uint32_t pin)

Reads the current input value of the GPIO port.

Deprecated:

Do not use this function. It has been superceded by *GPIO_PinRead*.

static inline uint8_t GPIO_PinReadPadStatus(GPIO_Type *base, uint32_t pin)

Reads the current GPIO pin pad status.

Parameters

- base – GPIO base pointer.
- pin – GPIO port pin number.

Return values

GPIO – pin pad status value.

static inline uint8_t GPIO_ReadPadStatus(GPIO_Type *base, uint32_t pin)

Reads the current GPIO pin pad status.

Deprecated:

Do not use this function. It has been superceded by *GPIO_PinReadPadStatus*.

void GPIO_PinSetInterruptConfig(GPIO_Type *base, uint32_t pin, *gpio_interrupt_mode_t* pinInterruptMode)

Sets the current pin interrupt mode.

Parameters

- base – GPIO base pointer.
- pin – GPIO port pin number.
- pinInterruptMode – pointer to a *gpio_interrupt_mode_t* structure that contains the interrupt mode information.

static inline void GPIO_SetPinInterruptConfig(GPIO_Type *base, uint32_t pin, *gpio_interrupt_mode_t* pinInterruptMode)

Sets the current pin interrupt mode.

Deprecated:

Do not use this function. It has been superceded by *GPIO_PinSetInterruptConfig*.

static inline void GPIO_PortEnableInterrupts(GPIO_Type *base, uint32_t mask)

Enables the specific pin interrupt.

Parameters

- base – GPIO base pointer.

- mask – GPIO pin number macro.

static inline void GPIO_EnableInterrupts(GPIO_Type *base, uint32_t mask)

Enables the specific pin interrupt.

Parameters

- base – GPIO base pointer.
- mask – GPIO pin number macro.

static inline void GPIO_PortDisableInterrupts(GPIO_Type *base, uint32_t mask)

Disables the specific pin interrupt.

Parameters

- base – GPIO base pointer.
- mask – GPIO pin number macro.

static inline void GPIO_DisableInterrupts(GPIO_Type *base, uint32_t mask)

Disables the specific pin interrupt.

Deprecated:

Do not use this function. It has been superceded by *GPIO_PortDisableInterrupts*.

static inline uint32_t GPIO_PortGetInterruptFlags(GPIO_Type *base)

Reads individual pin interrupt status.

Parameters

- base – GPIO base pointer.

Return values

current – pin interrupt status flag.

static inline uint32_t GPIO_GetPinsInterruptFlags(GPIO_Type *base)

Reads individual pin interrupt status.

Parameters

- base – GPIO base pointer.

Return values

current – pin interrupt status flag.

static inline void GPIO_PortClearInterruptFlags(GPIO_Type *base, uint32_t mask)

Clears pin interrupt flag. Status flags are cleared by writing a 1 to the corresponding bit position.

Parameters

- base – GPIO base pointer.
- mask – GPIO pin number macro.

static inline void GPIO_ClearPinsInterruptFlags(GPIO_Type *base, uint32_t mask)

Clears pin interrupt flag. Status flags are cleared by writing a 1 to the corresponding bit position.

Parameters

- base – GPIO base pointer.
- mask – GPIO pin number macro.

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_interrupt_mode
GPIO interrupt mode definition.
Values:

enumerator kGPIO_NoIntmode
Set current pin general IO functionality.

enumerator kGPIO_IntLowLevel
Set current pin interrupt is low-level sensitive.

enumerator kGPIO_IntHighLevel
Set current pin interrupt is high-level sensitive.

enumerator kGPIO_IntRisingEdge
Set current pin interrupt is rising-edge sensitive.

enumerator kGPIO_IntFallingEdge
Set current pin interrupt is falling-edge sensitive.

enumerator kGPIO_IntRisingOrFallingEdge
Enable the edge select bit to override the ICR register's configuration.

typedef enum *_gpio_pin_direction* gpio_pin_direction_t
GPIO direction definition.

typedef enum *_gpio_interrupt_mode* gpio_interrupt_mode_t
GPIO interrupt mode definition.

typedef struct *_gpio_pin_config* gpio_pin_config_t
GPIO Init structure definition.

struct _gpio_pin_config
#include <fsl_gpio.h> GPIO Init structure definition.

Public Members

gpio_pin_direction_t direction
Specifies the pin direction.

uint8_t outputLogic
Set a default output logic, which has no use in input

gpio_interrupt_mode_t interruptMode
Specifies the pin interrupt mode, a value of *gpio_interrupt_mode_t*.

2.36 GPT: General Purpose Timer

void GPT_Init(GPT_Type *base, const *gpt_config_t* *initConfig)
Initialize GPT to reset state and initialize running mode.

Parameters

- base – GPT peripheral base address.
- initConfig – GPT mode setting configuration.

void GPT_Deinit(GPT_Type *base)
Disables the module and gates the GPT clock.

Parameters

- base – GPT peripheral base address.

void GPT_GetDefaultConfig(*gpt_config_t* *config)
Fills in the GPT configuration structure with default settings.

The default values are:

```
config->clockSource = kGPT_ClockSource_Periph;
config->divider = 1U;
config->enableRunInStop = true;
config->enableRunInWait = true;
config->enableRunInDoze = false;
config->enableRunInDbg = false;
config->enableFreeRun = false;
config->enableMode  = true;
```

Parameters

- config – Pointer to the user configuration structure.

static inline void GPT_SoftwareReset(GPT_Type *base)
Software reset of GPT module.

Parameters

- base – GPT peripheral base address.

static inline void GPT_SetClockSource(GPT_Type *base, *gpt_clock_source_t* gptClkSource)
Set clock source of GPT.

Parameters

- base – GPT peripheral base address.

- gptClkSource – Clock source (see *gpt_clock_source_t* typedef enumeration).

static inline *gpt_clock_source_t* GPT_GetClockSource(GPT_Type *base)

Get clock source of GPT.

Parameters

- base – GPT peripheral base address.

Returns

clock source (see *gpt_clock_source_t* typedef enumeration).

static inline void GPT_SetClockDivider(GPT_Type *base, uint32_t divider)

Set pre scaler of GPT.

Parameters

- base – GPT peripheral base address.
- divider – Divider of GPT (1-4096).

static inline uint32_t GPT_GetClockDivider(GPT_Type *base)

Get clock divider in GPT module.

Parameters

- base – GPT peripheral base address.

Returns

clock divider in GPT module (1-4096).

static inline void GPT_SetOscClockDivider(GPT_Type *base, uint32_t divider)

OSC 24M pre-scaler before selected by clock source.

Parameters

- base – GPT peripheral base address.
- divider – OSC Divider(1-16).

static inline uint32_t GPT_GetOscClockDivider(GPT_Type *base)

Get OSC 24M clock divider in GPT module.

Parameters

- base – GPT peripheral base address.

Returns

OSC clock divider in GPT module (1-16).

static inline void GPT_StartTimer(GPT_Type *base)

Start GPT timer.

Parameters

- base – GPT peripheral base address.

static inline void GPT_StopTimer(GPT_Type *base)

Stop GPT timer.

Parameters

- base – GPT peripheral base address.

static inline uint32_t GPT_GetCurrentTimerCount(GPT_Type *base)

Reads the current GPT counting value.

Parameters

- base – GPT peripheral base address.

Returns
Current GPT counter value.

static inline void GPT_SetInputOperationMode(GPT_Type *base, *gpt_input_capture_channel_t* channel, *gpt_input_operation_mode_t* mode)

Set GPT operation mode of input capture channel.

Parameters

- base – GPT peripheral base address.
- channel – GPT capture channel (see *gpt_input_capture_channel_t* typedef enumeration).
- mode – GPT input capture operation mode (see *gpt_input_operation_mode_t* typedef enumeration).

static inline *gpt_input_operation_mode_t* GPT_GetInputOperationMode(GPT_Type *base, *gpt_input_capture_channel_t* channel)

Get GPT operation mode of input capture channel.

Parameters

- base – GPT peripheral base address.
- channel – GPT capture channel (see *gpt_input_capture_channel_t* typedef enumeration).

Returns
GPT input capture operation mode (see *gpt_input_operation_mode_t* typedef enumeration).

static inline uint32_t GPT_GetInputCaptureValue(GPT_Type *base, *gpt_input_capture_channel_t* channel)

Get GPT input capture value of certain channel.

Parameters

- base – GPT peripheral base address.
- channel – GPT capture channel (see *gpt_input_capture_channel_t* typedef enumeration).

Returns
GPT input capture value.

static inline void GPT_SetOutputOperationMode(GPT_Type *base, *gpt_output_compare_channel_t* channel, *gpt_output_operation_mode_t* mode)

Set GPT operation mode of output compare channel.

Parameters

- base – GPT peripheral base address.
- channel – GPT output compare channel (see *gpt_output_compare_channel_t* typedef enumeration).
- mode – GPT output operation mode (see *gpt_output_operation_mode_t* typedef enumeration).

static inline *gpt_output_operation_mode_t* GPT_GetOutputOperationMode(GPT_Type *base, *gpt_output_compare_channel_t* channel)

Get GPT operation mode of output compare channel.

Parameters

- base – GPT peripheral base address.
- channel – GPT output compare channel (see *gpt_output_compare_channel_t* typedef enumeration).

Returns
GPT output operation mode (see *gpt_output_operation_mode_t* typedef enumeration).

static inline void GPT_SetOutputCompareValue(GPT_Type *base, *gpt_output_compare_channel_t* channel, uint32_t value)

Set GPT output compare value of output compare channel.

Parameters

- base – GPT peripheral base address.
- channel – GPT output compare channel (see *gpt_output_compare_channel_t* typedef enumeration).
- value – GPT output compare value.

static inline uint32_t GPT_GetOutputCompareValue(GPT_Type *base, *gpt_output_compare_channel_t* channel)

Get GPT output compare value of output compare channel.

Parameters

- base – GPT peripheral base address.
- channel – GPT output compare channel (see *gpt_output_compare_channel_t* typedef enumeration).

Returns
GPT output compare value.

static inline void GPT_ForceOutput(GPT_Type *base, *gpt_output_compare_channel_t* channel)

Force GPT output action on output compare channel, ignoring comparator.

Parameters

- base – GPT peripheral base address.
- channel – GPT output compare channel (see *gpt_output_compare_channel_t* typedef enumeration).

static inline void GPT_EnableInterrupts(GPT_Type *base, uint32_t mask)

Enables the selected GPT interrupts.

Parameters

- base – GPT peripheral base address.
- mask – The interrupts to enable. This is a logical OR of members of the enumeration *gpt_interrupt_enable_t*

static inline void GPT_DisableInterrupts(GPT_Type *base, uint32_t mask)

Disables the selected GPT interrupts.

Parameters

- base – GPT peripheral base address
- mask – The interrupts to disable. This is a logical OR of members of the enumeration *gpt_interrupt_enable_t*

static inline uint32_t GPT_GetEnabledInterrupts(GPT_Type *base)

Gets the enabled GPT interrupts.

Parameters

- base – GPT peripheral base address

Returns

The enabled interrupts. This is the logical OR of members of the enumeration *gpt_interrupt_enable_t*

static inline uint32_t GPT_GetStatusFlags(GPT_Type *base, *gpt_status_flag_t* flags)

Get GPT status flags.

Parameters

- base – GPT peripheral base address.
- flags – GPT status flag mask (see *gpt_status_flag_t* for bit definition).

Returns

GPT status, each bit represents one status flag.

static inline void GPT_ClearStatusFlags(GPT_Type *base, *gpt_status_flag_t* flags)

Clears the GPT status flags.

Parameters

- base – GPT peripheral base address.
- flags – GPT status flag mask (see *gpt_status_flag_t* for bit definition).

FSL_GPT_DRIVER_VERSION

enum _gpt_clock_source

List of clock sources.

Note: Actual number of clock sources is SoC dependent

Values:

enumerator kGPT_ClockSource_Off

GPT Clock Source Off.

enumerator kGPT_ClockSource_Periph

GPT Clock Source from Peripheral Clock.

enumerator kGPT_ClockSource_HighFreq

GPT Clock Source from High Frequency Reference Clock.

enumerator kGPT_ClockSource_Ext

GPT Clock Source from external pin.

enumerator kGPT_ClockSource_LowFreq

GPT Clock Source from Low Frequency Reference Clock.

enumerator kGPT_ClockSource_Osc

GPT Clock Source from Crystal oscillator.

enum _gpt_input_capture_channel

List of input capture channel number.

Values:

enumerator kGPT_InputCapture_Channel1

GPT Input Capture Channel1.

enumerator kGPT_InputCapture_Channel2

GPT Input Capture Channel2.

enum _gpt_input_operation_mode

List of input capture operation mode.

Values:

enumerator kGPT_InputOperation_Disabled

Don't capture.

enumerator kGPT_InputOperation_RiseEdge

Capture on rising edge of input pin.

enumerator kGPT_InputOperation_FallEdge

Capture on falling edge of input pin.

enumerator kGPT_InputOperation_BothEdge

Capture on both edges of input pin.

enum _gpt_output_compare_channel

List of output compare channel number.

Values:

enumerator kGPT_OutputCompare_Channel1

Output Compare Channel1.

enumerator kGPT_OutputCompare_Channel2

Output Compare Channel2.

enumerator kGPT_OutputCompare_Channel3

Output Compare Channel3.

enum _gpt_output_operation_mode

List of output compare operation mode.

Values:

enumerator kGPT_OutputOperation_Disconnected

Don't change output pin.

enumerator kGPT_OutputOperation_Toggle

Toggle output pin.

enumerator kGPT_OutputOperation_Clear
Set output pin low.

enumerator kGPT_OutputOperation_Set
Set output pin high.

enumerator kGPT_OutputOperation_Activelow
Generate a active low pulse on output pin.

enum _gpt_interrupt_enable
List of GPT interrupts.
Values:

enumerator kGPT_OutputCompare1InterruptEnable
Output Compare Channel1 interrupt enable

enumerator kGPT_OutputCompare2InterruptEnable
Output Compare Channel2 interrupt enable

enumerator kGPT_OutputCompare3InterruptEnable
Output Compare Channel3 interrupt enable

enumerator kGPT_InputCapture1InterruptEnable
Input Capture Channel1 interrupt enable

enumerator kGPT_InputCapture2InterruptEnable
Input Capture Channel1 interrupt enable

enumerator kGPT_RollOverFlagInterruptEnable
Counter rolled over interrupt enable

enum _gpt_status_flag
Status flag.
Values:

enumerator kGPT_OutputCompare1Flag
Output compare channel 1 event.

enumerator kGPT_OutputCompare2Flag
Output compare channel 2 event.

enumerator kGPT_OutputCompare3Flag
Output compare channel 3 event.

enumerator kGPT_InputCapture1Flag
Input Capture channel 1 event.

enumerator kGPT_InputCapture2Flag
Input Capture channel 2 event.

enumerator kGPT_RollOverFlag
Counter reaches maximum value and rolled over to 0 event.

typedef enum *_gpt_clock_source* gpt_clock_source_t
List of clock sources.

Note: Actual number of clock sources is SoC dependent

typedef enum *_gpt_input_capture_channel* gpt_input_capture_channel_t
List of input capture channel number.

typedef enum *_gpt_input_operation_mode* gpt_input_operation_mode_t
List of input capture operation mode.

typedef enum *_gpt_output_compare_channel* gpt_output_compare_channel_t
List of output compare channel number.

typedef enum *_gpt_output_operation_mode* gpt_output_operation_mode_t
List of output compare operation mode.

typedef enum *_gpt_interrupt_enable* gpt_interrupt_enable_t
List of GPT interrupts.

typedef enum *_gpt_status_flag* gpt_status_flag_t
Status flag.

typedef struct *_gpt_init_config* gpt_config_t
Structure to configure the running mode.

struct __gpt_init_config
#include <fsl_gpt.h> Structure to configure the running mode.

Public Members

gpt_clock_source_t clockSource
clock source for GPT module.

uint32_t divider
clock divider (prescaler+1) from clock source to counter.

bool enableFreeRun
true: FreeRun mode, false: Restart mode.

bool enableRunInWait
GPT enabled in wait mode.

bool enableRunInStop
GPT enabled in stop mode.

bool enableRunInDoze
GPT enabled in doze mode.

bool enableRunInDbg
GPT enabled in debug mode.

bool enableMode
true: counter reset to 0 when enabled; false: counter retain its value when enabled.

2.37 IOMUXC: IOMUX Controller

enum _iomuxc_gpr_mode
Values:

enumerator kIOMUXC_GPR_GlobalInterruptRequest

enumerator kIOMUXC_GPR_ENET1RefClkMode

enumerator kIOMUXC_GPR_ENET1TxClkOutputDir

enumerator kIOMUXC_GPR_SAI1MClkOutputDir

enumerator kIOMUXC_GPR_SAI2MClkOutputDir

enumerator kIOMUXC_GPR_SAI3MClkOutputDir

enumerator kIOMUXC_GPR_ExcMonitorSlavErrResponse

enumerator kIOMUXC_GPR_AHBClockEnable

enum _iomuxc_gpr_saimclk
Values:

enumerator kIOMUXC_GPR_SAI1MClk1Sel

enumerator kIOMUXC_GPR_SAI1MClk2Sel

enumerator kIOMUXC_GPR_SAI1MClk3Sel

enumerator kIOMUXC_GPR_SAI2MClk3Sel

enumerator kIOMUXC_GPR_SAI3MClk3Sel

enum __iomuxc_mqs_pwm_oversample_rate

Values:

enumerator kIOMUXC_MqsPwmOverSampleRate32

enumerator kIOMUXC_MqsPwmOverSampleRate64

typedef enum *_iomuxc_gpr_mode* iomuxc_gpr_mode_t

typedef enum *_iomuxc_gpr_saimclk* iomuxc_gpr_saimclk_t

typedef enum *_iomuxc_mqs_pwm_oversample_rate* iomuxc_mqs_pwm_oversample_rate_t

IOMUXC_GPIO_EMC_00_SEMC_DATA00

IOMUXC_GPIO_EMC_00_QTIMER2_TIMER0

IOMUXC_GPIO_EMC_00_LPUART4_CTS_B

IOMUXC_GPIO_EMC_00_SPDIF_SR_CLK

IOMUXC_GPIO_EMC_00_LPSPI2_SCK

IOMUXC_GPIO_EMC_00_GPIO2_IO00

IOMUXC_GPIO_EMC_00_FLEXCAN1_TX

IOMUXC_GPIO_EMC_00_PIT_TRIGGER02

IOMUXC_GPIO_EMC_01_SEMC_DATA01

IOMUXC_GPIO_EMC_01_QTIMER2_TIMER1

IOMUXC_GPIO_EMC_01_LPUART4_RTS_B

IOMUXC_GPIO_EMC_01_SPDIF_OUT

IOMUXC_GPIO_EMC_01_LPSPI2_PCS0

IOMUXC_GPIO_EMC_01_GPIO2_IO01

IOMUXC_GPIO_EMC_01_FLEXCAN1_RX

IOMUXC_GPIO_EMC_01_PIT_TRIGGER03

IOMUXC_GPIO_EMC_02_SEMC_DATA02

IOMUXC_GPIO_EMC_02_QTIMER2_TIMER2

IOMUXC_GPIO_EMC_02_LPUART4_TX

IOMUXC_GPIO_EMC_02_SPDIF_LOCK

IOMUXC_GPIO_EMC_02_LPSPI2_SDO

IOMUXC_GPIO_EMC_02_GPIO2_IO02

IOMUXC_GPIO_EMC_02_LPI2C1_SCL

IOMUXC_GPIO_EMC_03_SEMC_DATA03

IOMUXC_GPIO_EMC_03_QTIMER2_TIMER3

IOMUXC_GPIO_EMC_03_LPUART4_RX

IOMUXC_GPIO_EMC_03_SPDIF_EXT_CLK

IOMUXC_GPIO_EMC_03_LPSPI2_SDI

IOMUXC_GPIO_EMC_03_GPIO2_IO03

IOMUXC_GPIO_EMC_03_LPI2C1_SDA

IOMUXC_GPIO_EMC_04_SEMC_DATA04

IOMUXC_GPIO_EMC_04_XBAR1_INOUT04

IOMUXC_GPIO_EMC_04_SPDIF_OUT

IOMUXC_GPIO_EMC_04_SAI2_TX_BCLK

IOMUXC_GPIO_EMC_04_FLEXIO1_FLEXIO16

IOMUXC_GPIO_EMC_04_GPIO2_IO04

IOMUXC_GPIO_EMC_05_SEMC_DATA05

IOMUXC_GPIO_EMC_05_XBAR1_INOUT05

IOMUXC_GPIO_EMC_05_SPDIF_IN

IOMUXC_GPIO_EMC_05_SAI2_TX_SYNC

IOMUXC_GPIO_EMC_05_FLEXIO1_FLEXIO17

IOMUXC_GPIO_EMC_05_GPIO2_IO05

IOMUXC_GPIO_EMC_06_SEMC_DATA06

IOMUXC_GPIO_EMC_06_XBAR1_INOUT06

IOMUXC_GPIO_EMC_06_LPUART3_TX

IOMUXC_GPIO_EMC_06_SAI2_TX_DATA

IOMUXC_GPIO_EMC_06_FLEXIO1_FLEXIO18

IOMUXC_GPIO_EMC_06_GPIO2_IO06

IOMUXC_GPIO_EMC_07_SEMC_DATA07

IOMUXC_GPIO_EMC_07_XBAR1_INOUT07

IOMUXC_GPIO_EMC_07_LPUART3_RX

IOMUXC_GPIO_EMC_07_SAI2_RX_SYNC

IOMUXC_GPIO_EMC_07_FLEXIO1_FLEXIO19

IOMUXC_GPIO_EMC_07_GPIO2_IO07

IOMUXC_GPIO_EMC_08_SEMC_DM00

IOMUXC_GPIO_EMC_08_XBAR1_INOUT08

IOMUXC_GPIO_EMC_08_FLEXCAN2_TX

IOMUXC_GPIO_EMC_08_SAI2_RX_DATA

IOMUXC_GPIO_EMC_08_FLEXIO1_FLEXIO20

IOMUXC_GPIO_EMC_08_GPIO2_IO08

IOMUXC_GPIO_EMC_09_SEMC_WE

IOMUXC_GPIO_EMC_09_XBAR1_INOUT09

IOMUXC_GPIO_EMC_09_FLEXCAN2_RX

IOMUXC_GPIO_EMC_09_SAI2_RX_BCLK

IOMUXC_GPIO_EMC_09_FLEXIO1_FLEXIO21

IOMUXC_GPIO_EMC_09_GPIO2_IO09

IOMUXC_GPIO_EMC_10_SEMC_CAS

IOMUXC_GPIO_EMC_10_XBAR1_INOUT10

IOMUXC_GPIO_EMC_10_LPI2C4_SDA

IOMUXC_GPIO_EMC_10_SAI1_TX_SYNC

IOMUXC_GPIO_EMC_10_LPSPI2_SCK

IOMUXC_GPIO_EMC_10_GPIO2_IO10

IOMUXC_GPIO_EMC_10_FLEXPWM2_PWMX00

IOMUXC_GPIO_EMC_11_SEMC_RAS

IOMUXC_GPIO_EMC_11_XBAR1_INOUT11

IOMUXC_GPIO_EMC_11_LPI2C4_SCL

IOMUXC_GPIO_EMC_11_SAI1_TX_BCLK

IOMUXC_GPIO_EMC_11_LPSPI2_PCS0

IOMUXC_GPIO_EMC_11_GPIO2_IO11

IOMUXC_GPIO_EMC_11_FLEXPWM2_PWMX01

IOMUXC_GPIO_EMC_12_SEMC_CS0

IOMUXC_GPIO_EMC_12_XBAR1_INOUT12

IOMUXC_GPIO_EMC_12_LPUART6_TX

IOMUXC_GPIO_EMC_12_SAI1_TX_DATA00

IOMUXC_GPIO_EMC_12_LPSPI2_SDO

IOMUXC_GPIO_EMC_12_GPIO2_IO12

IOMUXC_GPIO_EMC_12_FLEXPWM2_PWMX02

IOMUXC_GPIO_EMC_13_SEMC_BA0

IOMUXC_GPIO_EMC_13_XBAR1_INOUT13

IOMUXC_GPIO_EMC_13_LPUART6_RX

IOMUXC_GPIO_EMC_13_SAI1_RX_DATA00

IOMUXC_GPIO_EMC_13_LPSPI2_SDI

IOMUXC_GPIO_EMC_13_GPIO2_IO13

IOMUXC_GPIO_EMC_13_FLEXPWM2_PWMX03

IOMUXC_GPIO_EMC_13_CCM_PMIC_RDY

IOMUXC_GPIO_EMC_14_SEMC_BA1

IOMUXC_GPIO_EMC_14_XBAR1_INOUT14

IOMUXC_GPIO_EMC_14_LPUART6_CTS_B

IOMUXC_GPIO_EMC_14_SAI1_RX_BCLK

IOMUXC_GPIO_EMC_14_LPSPI2_PCS1

IOMUXC_GPIO_EMC_14_GPIO2_IO14

IOMUXC_GPIO_EMC_14_FLEXCAN1_TX

IOMUXC_GPIO_EMC_15_SEMC_ADDR10

IOMUXC_GPIO_EMC_15_XBAR1_INOUT15

IOMUXC_GPIO_EMC_15_LPUART6_RTS_B

IOMUXC_GPIO_EMC_15_SAI1_RX_SYNC

IOMUXC_GPIO_EMC_15_WDOG1_B

IOMUXC_GPIO_EMC_15_GPIO2_IO15

IOMUXC_GPIO_EMC_15_FLEXCAN1_RX

IOMUXC_GPIO_EMC_16_SEMC_ADDR00

IOMUXC_GPIO_EMC_16_MQS_RIGHT

IOMUXC_GPIO_EMC_16_SAI2_MCLK

IOMUXC_GPIO_EMC_16_GPIO2_IO16

IOMUXC_GPIO_EMC_16_SRC_BOOT_MODE00

IOMUXC_GPIO_EMC_17_SEMC_ADDR01

IOMUXC_GPIO_EMC_17_MQS_LEFT

IOMUXC_GPIO_EMC_17_SAI3_MCLK

IOMUXC_GPIO_EMC_17_GPIO2_IO17

IOMUXC_GPIO_EMC_17_SRC_BOOT_MODE01

IOMUXC_GPIO_EMC_18_SEMC_ADDR02

IOMUXC_GPIO_EMC_18_XBAR1_INOUT16

IOMUXC_GPIO_EMC_18_LPI2C2_SDA

IOMUXC_GPIO_EMC_18_SAI1_RX_SYNC

IOMUXC_GPIO_EMC_18_FLEXIO1_FLEXIO22

IOMUXC_GPIO_EMC_18_GPIO2_IO18

IOMUXC_GPIO_EMC_18_SRC_BT_CFG00

IOMUXC_GPIO_EMC_19_SEMC_ADDR03

IOMUXC_GPIO_EMC_19_XBAR1_INOUT17

IOMUXC_GPIO_EMC_19_LPI2C2_SCL

IOMUXC_GPIO_EMC_19_SAI1_RX_BCLK

IOMUXC_GPIO_EMC_19_FLEXIO1_FLEXIO23

IOMUXC_GPIO_EMC_19_GPIO2_IO19

IOMUXC_GPIO_EMC_19_SRC_BT_CFG01

IOMUXC_GPIO_EMC_20_SEMC_ADDR04

IOMUXC_GPIO_EMC_20_FLEXPWM1_PWMA03

IOMUXC_GPIO_EMC_20_LPUART2_CTS_B

IOMUXC_GPIO_EMC_20_SAI1_MCLK

IOMUXC_GPIO_EMC_20_FLEXIO1_FLEXIO24

IOMUXC_GPIO_EMC_20_GPIO2_IO20

IOMUXC_GPIO_EMC_20_SRC_BT_CFG02

IOMUXC_GPIO_EMC_21_SEMC_ADDR05

IOMUXC_GPIO_EMC_21_FLEXPWM1_PWMB03

IOMUXC_GPIO_EMC_21_LPUART2_RTS_B

IOMUXC_GPIO_EMC_21_SAI1_RX_DATA00

IOMUXC_GPIO_EMC_21_FLEXIO1_FLEXIO25

IOMUXC_GPIO_EMC_21_GPIO2_IO21

IOMUXC_GPIO_EMC_21_SRC_BT_CFG03

IOMUXC_GPIO_EMC_22_SEMC_ADDR06

IOMUXC_GPIO_EMC_22_FLEXPWM1_PWMA02

IOMUXC_GPIO_EMC_22_LPUART2_TX

IOMUXC_GPIO_EMC_22_SAI1_TX_DATA03

IOMUXC_GPIO_EMC_22_FLEXIO1_FLEXIO26

IOMUXC_GPIO_EMC_22_GPIO2_IO22

IOMUXC_GPIO_EMC_22_SRC_BT_CFG04

IOMUXC_GPIO_EMC_23_SEMC_ADDR07

IOMUXC_GPIO_EMC_23_FLEXPWM1_PWMB02

IOMUXC_GPIO_EMC_23_LPUART2_RX

IOMUXC_GPIO_EMC_23_SAI1_TX_DATA02

IOMUXC_GPIO_EMC_23_FLEXIO1_FLEXIO27

IOMUXC_GPIO_EMC_23_GPIO2_IO23

IOMUXC_GPIO_EMC_23_SRC_BT_CFG05

IOMUXC_GPIO_EMC_24_SEMC_ADDR08

IOMUXC_GPIO_EMC_24_FLEXPWM1_PWMA01

IOMUXC_GPIO_EMC_24_LPUART8_CTS_B

IOMUXC_GPIO_EMC_24_SAI1_TX_DATA01

IOMUXC_GPIO_EMC_24_FLEXIO1_FLEXIO28

IOMUXC_GPIO_EMC_24_GPIO2_IO24

IOMUXC_GPIO_EMC_24_SRC_BT_CFG06

IOMUXC_GPIO_EMC_25_SEMC_ADDR09

IOMUXC_GPIO_EMC_25_FLEXPWM1_PWMB01

IOMUXC_GPIO_EMC_25_LPUART8_RTS_B

IOMUXC_GPIO_EMC_25_SAI1_TX_DATA00

IOMUXC_GPIO_EMC_25_FLEXIO1_FLEXIO29

IOMUXC_GPIO_EMC_25_GPIO2_IO25

IOMUXC_GPIO_EMC_25_SRC_BT_CFG07

IOMUXC_GPIO_EMC_26_SEMC_ADDR11

IOMUXC_GPIO_EMC_26_FLEXPWM1_PWMA00

IOMUXC_GPIO_EMC_26_LPUART8_TX

IOMUXC_GPIO_EMC_26_SAI1_TX_BCLK

IOMUXC_GPIO_EMC_26_FLEXIO1_FLEXIO30

IOMUXC_GPIO_EMC_26_GPIO2_IO26

IOMUXC_GPIO_EMC_26_SRC_BT_CFG08

IOMUXC_GPIO_EMC_27_SEMC_ADDR12

IOMUXC_GPIO_EMC_27_FLEXPWM1_PWMB00

IOMUXC_GPIO_EMC_27_LPUART8_RX

IOMUXC_GPIO_EMC_27_SAI1_TX_SYNC

IOMUXC_GPIO_EMC_27_FLEXIO1_FLEXIO31

IOMUXC_GPIO_EMC_27_GPIO2_IO27

IOMUXC_GPIO_EMC_27_SRC_BT_CFG09

IOMUXC_GPIO_EMC_28_SEMC_DQS

IOMUXC_GPIO_EMC_28_FLEXPWM2_PWMA03

IOMUXC_GPIO_EMC_28_XBAR1_INOUT18

IOMUXC_GPIO_EMC_28_SAI3_MCLK

IOMUXC_GPIO_EMC_28_EWM_OUT_B

IOMUXC_GPIO_EMC_28_GPIO2_IO28

IOMUXC_GPIO_EMC_28_GPT2_CAPTURE2

IOMUXC_GPIO_EMC_28_FLEXPWM1_PWMX00

IOMUXC_GPIO_EMC_29_SEMC_CKE

IOMUXC_GPIO_EMC_29_FLEXPWM2_PWMB03

IOMUXC_GPIO_EMC_29_XBAR1_INOUT19

IOMUXC_GPIO_EMC_29_SAI3_RX_BCLK

IOMUXC_GPIO_EMC_29_WDOG2_RST_B_DEB

IOMUXC_GPIO_EMC_29_GPIO2_IO29

IOMUXC_GPIO_EMC_29_GPT2_COMPARE2

IOMUXC_GPIO_EMC_29_FLEXPWM1_PWMX01

IOMUXC_GPIO_EMC_30_SEMC_CLK

IOMUXC_GPIO_EMC_30_FLEXPWM2_PWMA02

IOMUXC_GPIO_EMC_30_LPUART4_CTS_B

IOMUXC_GPIO_EMC_30_SAI3_RX_SYNC

IOMUXC_GPIO_EMC_30_WDOG1_RST_B_DEB

IOMUXC_GPIO_EMC_30_GPIO2_IO30

IOMUXC_GPIO_EMC_30_GPT2_COMPARE3

IOMUXC_GPIO_EMC_30_FLEXPWM1_PWMX02

IOMUXC_GPIO_EMC_31_SEMC_DM01

IOMUXC_GPIO_EMC_31_FLEXPWM2_PWMB02

IOMUXC_GPIO_EMC_31_LPUART4_RTS_B

IOMUXC_GPIO_EMC_31_SAI3_RX_DATA

IOMUXC_GPIO_EMC_31_WDOG2_B

IOMUXC_GPIO_EMC_31_GPIO2_IO31

IOMUXC_GPIO_EMC_31_GPT2_CLK

IOMUXC_GPIO_EMC_31_FLEXPWM1_PWMX03

IOMUXC_GPIO_EMC_32_SEMC_DATA08

IOMUXC_GPIO_EMC_32_QTIMER1_TIMER0

IOMUXC_GPIO_EMC_32_LPUART4_TX

IOMUXC_GPIO_EMC_32_SAI3_TX_DATA

IOMUXC_GPIO_EMC_32_LPSPI4_SCK

IOMUXC_GPIO_EMC_32_GPIO3_IO00

IOMUXC_GPIO_EMC_32_REF_24M_OUT

IOMUXC_GPIO_EMC_33_SEMC_DATA09

IOMUXC_GPIO_EMC_33_QTIMER1_TIMER1

IOMUXC_GPIO_EMC_33_LPUART4_RX

IOMUXC_GPIO_EMC_33_SAI3_TX_BCLK

IOMUXC_GPIO_EMC_33_LPSPI4_PCS0

IOMUXC_GPIO_EMC_33_GPIO3_IO01

IOMUXC_GPIO_EMC_34_SEMC_DATA10

IOMUXC_GPIO_EMC_34_QTIMER1_TIMER2

IOMUXC_GPIO_EMC_34_LPUART7_TX

IOMUXC_GPIO_EMC_34_SAI3_TX_SYNC

IOMUXC_GPIO_EMC_34_LPSPI4_SDO

IOMUXC_GPIO_EMC_34_GPIO3_IO02

IOMUXC_GPIO_EMC_34_ENET_CRS

IOMUXC_GPIO_EMC_35_SEMC_DATA11

IOMUXC_GPIO_EMC_35_QTIMER1_TIMER3

IOMUXC_GPIO_EMC_35_LPUART7_RX

IOMUXC_GPIO_EMC_35_USDHC2_WP

IOMUXC_GPIO_EMC_35_LPSPI4_SDI

IOMUXC_GPIO_EMC_35_GPIO3_IO03

IOMUXC_GPIO_EMC_35_ENET_COL

IOMUXC_GPIO_EMC_36_SEMC_DATA12

IOMUXC_GPIO_EMC_36_FLEXPWM2_PWMA01

IOMUXC_GPIO_EMC_36_LPUART5_CTS_B

IOMUXC_GPIO_EMC_36_CCM_PMIC_RDY

IOMUXC_GPIO_EMC_36_LPSPI4_PCS1

IOMUXC_GPIO_EMC_36_GPIO3_IO04

IOMUXC_GPIO_EMC_36_ENET_RX_CLK

IOMUXC_GPIO_EMC_36_USDHC1_WP

IOMUXC_GPIO_EMC_37_SEMC_DATA13

IOMUXC_GPIO_EMC_37_FLEXPWM2_PWMB01

IOMUXC_GPIO_EMC_37_LPUART5_RTS_B

IOMUXC_GPIO_EMC_37_MQS_RIGHT

IOMUXC_GPIO_EMC_37_LPSPI4_PCS2

IOMUXC_GPIO_EMC_37_GPIO3_IO05

IOMUXC_GPIO_EMC_37_ENET_RDATA03

IOMUXC_GPIO_EMC_37_USDHC1_VSELECT

IOMUXC_GPIO_EMC_38_SEMC_DATA14

IOMUXC_GPIO_EMC_38_FLEXPWM2_PWMA00

IOMUXC_GPIO_EMC_38_LPUART5_TX

IOMUXC_GPIO_EMC_38_MQS_LEFT

IOMUXC_GPIO_EMC_38_LPSPI4_PCS3

IOMUXC_GPIO_EMC_38_GPIO3_IO06

IOMUXC_GPIO_EMC_38_ENET_RDATA02

IOMUXC_GPIO_EMC_38_USDHC1_CD_B

IOMUXC_GPIO_EMC_39_SEMC_DATA15

IOMUXC_GPIO_EMC_39_FLEXPWM2_PWMB00

IOMUXC_GPIO_EMC_39_LPUART5_RX

IOMUXC_GPIO_EMC_39_USB_OTG1_OC

IOMUXC_GPIO_EMC_39_WDOG1_B

IOMUXC_GPIO_EMC_39_GPIO3_IO07

IOMUXC_GPIO_EMC_39_ENET_TX_ER

IOMUXC_GPIO_EMC_39_GPT1_CLK

IOMUXC_GPIO_EMC_40_SEMC_CSX00

IOMUXC_GPIO_EMC_40_XBAR1_INOUT18

IOMUXC_GPIO_EMC_40_SPDIF_OUT

IOMUXC_GPIO_EMC_40_USB_OTG1_ID

IOMUXC_GPIO_EMC_40_ENET_MDIO

IOMUXC_GPIO_EMC_40_GPIO3_IO08

IOMUXC_GPIO_EMC_40_ENET_TDATA03

IOMUXC_GPIO_EMC_40_GPT1_COMPARE3

IOMUXC_GPIO_EMC_41_SEMC_READY

IOMUXC_GPIO_EMC_41_XBAR1_INOUT19

IOMUXC_GPIO_EMC_41_SPDIF_IN

IOMUXC_GPIO_EMC_41_USB_OTG1_PWR

IOMUXC_GPIO_EMC_41_ENET_MDC

IOMUXC_GPIO_EMC_41_GPIO3_IO09

IOMUXC_GPIO_EMC_41_ENET_TDATA02

IOMUXC_GPIO_EMC_41_GPT1_COMPARE2

IOMUXC_GPIO_AD_B0_00_JTAG_TMS

IOMUXC_GPIO_AD_B0_00_GPIO1_IO00

IOMUXC_GPIO_AD_B0_00_GPT1_COMPARE1

IOMUXC_GPIO_AD_B0_01_JTAG_TCK

IOMUXC_GPIO_AD_B0_01_GPIO1_IO01

IOMUXC_GPIO_AD_B0_01_GPT1_CAPTURE2

IOMUXC_GPIO_AD_B0_02_JTAG_MOD

IOMUXC_GPIO_AD_B0_02_GPIO1_IO02

IOMUXC_GPIO_AD_B0_02_GPT1_CAPTURE1

IOMUXC_GPIO_AD_B0_03_JTAG_TDI

IOMUXC_GPIO_AD_B0_03_USDHC2_CD_B

IOMUXC_GPIO_AD_B0_03_WDOG1_B

IOMUXC_GPIO_AD_B0_03_SAI1_MCLK

IOMUXC_GPIO_AD_B0_03_USDHC1_WP

IOMUXC_GPIO_AD_B0_03_GPIO1_IO03

IOMUXC_GPIO_AD_B0_03_USB_OTG1_OC

IOMUXC_GPIO_AD_B0_03_CCM_PMIC_RDY

IOMUXC_GPIO_AD_B0_04_JTAG_TDO

IOMUXC_GPIO_AD_B0_04_FLEXCAN1_TX

IOMUXC_GPIO_AD_B0_04_USDHC1_WP

IOMUXC_GPIO_AD_B0_04_QTIMER2_TIMER0

IOMUXC_GPIO_AD_B0_04_ENET_MDIO

IOMUXC_GPIO_AD_B0_04_GPIO1_IO04

IOMUXC_GPIO_AD_B0_04_USB_OTG1_PWR

IOMUXC_GPIO_AD_B0_04_EWM_OUT_B

IOMUXC_GPIO_AD_B0_05_JTAG_TRSTB

IOMUXC_GPIO_AD_B0_05_FLEXCAN1_RX

IOMUXC_GPIO_AD_B0_05_USDHC1_CD_B

IOMUXC_GPIO_AD_B0_05_QTIMER2_TIMER1

IOMUXC_GPIO_AD_B0_05_ENET_MDC

IOMUXC_GPIO_AD_B0_05_GPIO1_IO05

IOMUXC_GPIO_AD_B0_05_USB_OTG1_ID

IOMUXC_GPIO_AD_B0_05_NMI_GLUE_NMI

IOMUXC_GPIO_AD_B0_06_PIT_TRIGGER00

IOMUXC_GPIO_AD_B0_06_MQS_RIGHT

IOMUXC_GPIO_AD_B0_06_LPUART1_TX

IOMUXC_GPIO_AD_B0_06_QTIMER2_TIMER2

IOMUXC_GPIO_AD_B0_06_FLEXPWM2_PWMA03

IOMUXC_GPIO_AD_B0_06_GPIO1_IO06

IOMUXC_GPIO_AD_B0_06_REF_32K_OUT

IOMUXC_GPIO_AD_B0_07_PIT_TRIGGER01

IOMUXC_GPIO_AD_B0_07_MQS_LEFT

IOMUXC_GPIO_AD_B0_07_LPUART1_RX

IOMUXC_GPIO_AD_B0_07_QTIMER2_TIMER3

IOMUXC_GPIO_AD_B0_07_FLEXPWM2_PWMB03

IOMUXC_GPIO_AD_B0_07_GPIO1_IO07

IOMUXC_GPIO_AD_B0_07_REF_24M_OUT

IOMUXC_GPIO_AD_B0_08_ENET_TX_CLK

IOMUXC_GPIO_AD_B0_08_LPI2C3_SCL

IOMUXC_GPIO_AD_B0_08_LPUART1_CTS_B

IOMUXC_GPIO_AD_B0_08_KPP_COL00

IOMUXC_GPIO_AD_B0_08_ENET_REF_CLK1

IOMUXC_GPIO_AD_B0_08_GPIO1_IO08

IOMUXC_GPIO_AD_B0_08_ARM_CM7_TXEV

IOMUXC_GPIO_AD_B0_09_ENET_RDATA01

IOMUXC_GPIO_AD_B0_09_LPI2C3_SDA

IOMUXC_GPIO_AD_B0_09_LPUART1_RTS_B

IOMUXC_GPIO_AD_B0_09_KPP_ROW00

IOMUXC_GPIO_AD_B0_09_GPIO1_IO09

IOMUXC_GPIO_AD_B0_09_ARM_CM7_RXEV

IOMUXC_GPIO_AD_B0_10_ENET_RDATA00

IOMUXC_GPIO_AD_B0_10_LPSPI1_SCK

IOMUXC_GPIO_AD_B0_10_LPUART5_TX

IOMUXC_GPIO_AD_B0_10_KPP_COL01

IOMUXC_GPIO_AD_B0_10_FLEXPWM2_PWMA02

IOMUXC_GPIO_AD_B0_10_GPIO1_IO10

IOMUXC_GPIO_AD_B0_10_ARM_CM7_TRACE_CLK

IOMUXC_GPIO_AD_B0_11_ENET_RX_EN

IOMUXC_GPIO_AD_B0_11_LPSPI1_PCS0

IOMUXC_GPIO_AD_B0_11_LPUART5_RX

IOMUXC_GPIO_AD_B0_11_KPP_ROW01

IOMUXC_GPIO_AD_B0_11_FLEXPWM2_PWMB02

IOMUXC_GPIO_AD_B0_11_GPIO1_IO11

IOMUXC_GPIO_AD_B0_11_ARM_CM7_TRACE_SWO

IOMUXC_GPIO_AD_B0_12_ENET_RX_ER

IOMUXC_GPIO_AD_B0_12_LPSPI1_SDO

IOMUXC_GPIO_AD_B0_12_LPUART3_CTS_B

IOMUXC_GPIO_AD_B0_12_KPP_COL02

IOMUXC_GPIO_AD_B0_12_FLEXPWM2_PWMA01

IOMUXC_GPIO_AD_B0_12_GPIO1_IO12

IOMUXC_GPIO_AD_B0_12_ARM_CM7_TRACE00

IOMUXC_GPIO_AD_B0_12_SNVS_HP_VIO_5_CTL

IOMUXC_GPIO_AD_B0_13_ENET_TX_EN

IOMUXC_GPIO_AD_B0_13_LPSPI1_SDI

IOMUXC_GPIO_AD_B0_13_LPUART3_RTS_B

IOMUXC_GPIO_AD_B0_13_KPP_ROW02

IOMUXC_GPIO_AD_B0_13_FLEXPWM2_PWMB01

IOMUXC_GPIO_AD_B0_13_GPIO1_IO13

IOMUXC_GPIO_AD_B0_13_ARM_CM7_TRACE01

IOMUXC_GPIO_AD_B0_13_SNVS_HP_VIO_5_B

IOMUXC_GPIO_AD_B0_14_ENET_TDATA00

IOMUXC_GPIO_AD_B0_14_FLEXCAN2_TX

IOMUXC_GPIO_AD_B0_14_LPUART3_TX

IOMUXC_GPIO_AD_B0_14_KPP_COL03

IOMUXC_GPIO_AD_B0_14_FLEXPWM2_PWMA00

IOMUXC_GPIO_AD_B0_14_GPIO1_IO14

IOMUXC_GPIO_AD_B0_14_ARM_CM7_TRACE02

IOMUXC_GPIO_AD_B0_14_WDOG1_ANY

IOMUXC_GPIO_AD_B0_15_ENET_TDATA01

IOMUXC_GPIO_AD_B0_15_FLEXCAN2_RX

IOMUXC_GPIO_AD_B0_15_LPUART3_RX

IOMUXC_GPIO_AD_B0_15_KPP_ROW03

IOMUXC_GPIO_AD_B0_15_FLEXPWM2_PWMB00

IOMUXC_GPIO_AD_B0_15_GPIO1_IO15

IOMUXC_GPIO_AD_B0_15_ARM_CM7_TRACE03

IOMUXC_GPIO_AD_B1_06_USDHC1_RESET_B

IOMUXC_GPIO_AD_B1_06_FLEXPWM1_PWMA00

IOMUXC_GPIO_AD_B1_06_LPUART2_CTS_B

IOMUXC_GPIO_AD_B1_06_SAI1_RX_BCLK

IOMUXC_GPIO_AD_B1_06_FLEXIO1_FLEXIO09

IOMUXC_GPIO_AD_B1_06_GPIO1_IO22

IOMUXC_GPIO_AD_B1_06_LPSPI1_PCS3

IOMUXC_GPIO_AD_B1_06_KPP_COL07

IOMUXC_GPIO_AD_B1_07_USDHC1_VSELECT

IOMUXC_GPIO_AD_B1_07_FLEXPWM1_PWMB00

IOMUXC_GPIO_AD_B1_07_LPUART2_RTS_B

IOMUXC_GPIO_AD_B1_07_SAI1_TX_DATA01

IOMUXC_GPIO_AD_B1_07_FLEXIO1_FLEXIO08

IOMUXC_GPIO_AD_B1_07_GPIO1_IO23

IOMUXC_GPIO_AD_B1_07_LPSPI3_PCS3

IOMUXC_GPIO_AD_B1_07_KPP_ROW07

IOMUXC_GPIO_AD_B1_08_LPI2C2_SCL

IOMUXC_GPIO_AD_B1_08_FLEXPWM1_PWMA01

IOMUXC_GPIO_AD_B1_08_LPUART2_TX

IOMUXC_GPIO_AD_B1_08_SAI1_TX_DATA02

IOMUXC_GPIO_AD_B1_08_FLEXIO1_FLEXIO07

IOMUXC_GPIO_AD_B1_08_GPIO1_IO24

IOMUXC_GPIO_AD_B1_08_LPSPI3_PCS2

IOMUXC_GPIO_AD_B1_08_XBAR1_INOUT12

IOMUXC_GPIO_AD_B1_09_LPI2C2_SDA

IOMUXC_GPIO_AD_B1_09_FLEXPWM1_PWMB01

IOMUXC_GPIO_AD_B1_09_LPUART2_RX

IOMUXC_GPIO_AD_B1_09_SAI1_TX_DATA03

IOMUXC_GPIO_AD_B1_09_FLEXIO1_FLEXIO06

IOMUXC_GPIO_AD_B1_09_GPIO1_IO25

IOMUXC_GPIO_AD_B1_09_LPSPI3_PCS1

IOMUXC_GPIO_AD_B1_09_XBAR1_INOUT13

IOMUXC_GPIO_AD_B1_10_USB_OTG1_PWR

IOMUXC_GPIO_AD_B1_10_FLEXPWM1_PWMA02

IOMUXC_GPIO_AD_B1_10_LPUART4_TX

IOMUXC_GPIO_AD_B1_10_USDHC1_CD_B

IOMUXC_GPIO_AD_B1_10_FLEXIO1_FLEXIO05

IOMUXC_GPIO_AD_B1_10_GPIO1_IO26

IOMUXC_GPIO_AD_B1_10_GPT2_CAPTURE1

IOMUXC_GPIO_AD_B1_11_USB_OTG1_ID

IOMUXC_GPIO_AD_B1_11_FLEXPWM1_PWMB02

IOMUXC_GPIO_AD_B1_11_LPUART4_RX

IOMUXC_GPIO_AD_B1_11_USDHC1_WP

IOMUXC_GPIO_AD_B1_11_FLEXIO1_FLEXIO04

IOMUXC_GPIO_AD_B1_11_GPIO1_IO27

IOMUXC_GPIO_AD_B1_11_GPT2_COMPARE1

IOMUXC_GPIO_AD_B1_12_USB_OTG1_OC

IOMUXC_GPIO_AD_B1_12_ACMP1_OUT

IOMUXC_GPIO_AD_B1_12_LPSPI3_SCK

IOMUXC_GPIO_AD_B1_12_USDHC2_CD_B

IOMUXC_GPIO_AD_B1_12_FLEXIO1_FLEXIO03

IOMUXC_GPIO_AD_B1_12_GPIO1_IO28

IOMUXC_GPIO_AD_B1_12_FLEXPWM1_PWMA03

IOMUXC_GPIO_AD_B1_13_LPI2C1_HREQ

IOMUXC_GPIO_AD_B1_13_ACMP2_OUT

IOMUXC_GPIO_AD_B1_13_LPSPI3_PCS0

IOMUXC_GPIO_AD_B1_13_USDHC2_WP

IOMUXC_GPIO_AD_B1_13_FLEXIO1_FLEXIO02

IOMUXC_GPIO_AD_B1_13_GPIO1_IO29

IOMUXC_GPIO_AD_B1_13_FLEXPWM1_PWMB03

IOMUXC_GPIO_AD_B1_14_LPI2C1_SCL

IOMUXC_GPIO_AD_B1_14_ACMP3_OUT

IOMUXC_GPIO_AD_B1_14_LPSPI3_SDO

IOMUXC_GPIO_AD_B1_14_ENET_1588_EVENT0_OUT

IOMUXC_GPIO_AD_B1_14_FLEXIO1_FLEXIO01

IOMUXC_GPIO_AD_B1_14_GPIO1_IO30

IOMUXC_GPIO_AD_B1_15_LPI2C1_SDA

IOMUXC_GPIO_AD_B1_15_ACMP4_OUT

IOMUXC_GPIO_AD_B1_15_LPSPI3_SDI

IOMUXC_GPIO_AD_B1_15_ENET_1588_EVENT0_IN

IOMUXC_GPIO_AD_B1_15_FLEXIO1_FLEXIO00

IOMUXC_GPIO_AD_B1_15_GPIO1_IO31

IOMUXC_GPIO_SD_B0_00_USDHC1_DATA2

IOMUXC_GPIO_SD_B0_00_QTIMER1_TIMER0

IOMUXC_GPIO_SD_B0_00_SAI1_MCLK

IOMUXC_GPIO_SD_B0_00_SAI2_MCLK

IOMUXC_GPIO_SD_B0_00_LPI2C3_SCL

IOMUXC_GPIO_SD_B0_00_GPIO3_IO13

IOMUXC_GPIO_SD_B0_00_FLEXSPI_A_SS1_B

IOMUXC_GPIO_SD_B0_00_XBAR1_INOUT14

IOMUXC_GPIO_SD_B0_01_USDHC1_DATA3

IOMUXC_GPIO_SD_B0_01_QTIMER1_TIMER1

IOMUXC_GPIO_SD_B0_01_REF_24M_OUT

IOMUXC_GPIO_SD_B0_01_SAI2_RX_SYNC

IOMUXC_GPIO_SD_B0_01_LPI2C3_SDA

IOMUXC_GPIO_SD_B0_01_GPIO3_IO14

IOMUXC_GPIO_SD_B0_01_FLEXSPI_B_SS1_B

IOMUXC_GPIO_SD_B0_01_XBAR1_INOUT15

IOMUXC_GPIO_SD_B0_02_USDHC1_CMD

IOMUXC_GPIO_SD_B0_02_QTIMER1_TIMER2

IOMUXC_GPIO_SD_B0_02_LPUART7_CTS_B

IOMUXC_GPIO_SD_B0_02_SAI2_RX_BCLK

IOMUXC_GPIO_SD_B0_02_LPSPI1_SCK

IOMUXC_GPIO_SD_B0_02_GPIO3_IO15

IOMUXC_GPIO_SD_B0_02_ENET_MDIO

IOMUXC_GPIO_SD_B0_02_XBAR1_INOUT16

IOMUXC_GPIO_SD_B0_03_USDHC1_CLK

IOMUXC_GPIO_SD_B0_03_QTIMER1_TIMER3

IOMUXC_GPIO_SD_B0_03_LPUART7_RTS_B

IOMUXC_GPIO_SD_B0_03_SAI2_RX_DATA

IOMUXC_GPIO_SD_B0_03_LPSPI1_PCS0

IOMUXC_GPIO_SD_B0_03_GPIO3_IO16

IOMUXC_GPIO_SD_B0_03_ENET_MDC

IOMUXC_GPIO_SD_B0_04_USDHC1_DATA0

IOMUXC_GPIO_SD_B0_04_FLEXCAN2_TX

IOMUXC_GPIO_SD_B0_04_LPUART7_TX

IOMUXC_GPIO_SD_B0_04_SAI2_TX_DATA

IOMUXC_GPIO_SD_B0_04_LPSPI1_SDO

IOMUXC_GPIO_SD_B0_04_GPIO3_IO17

IOMUXC_GPIO_SD_B0_04_FLEXSPI_B_SS0_B

IOMUXC_GPIO_SD_B0_05_USDHC1_DATA1

IOMUXC_GPIO_SD_B0_05_FLEXCAN2_RX

IOMUXC_GPIO_SD_B0_05_LPUART7_RX

IOMUXC_GPIO_SD_B0_05_SAI2_TX_BCLK

IOMUXC_GPIO_SD_B0_05_LPSPI1_SDI

IOMUXC_GPIO_SD_B0_05_GPIO3_IO18

IOMUXC_GPIO_SD_B0_05_FLEXSPI_B_DQS

IOMUXC_GPIO_SD_B0_06_USDHC1_CD_B

IOMUXC_GPIO_SD_B0_06_USDHC1_RESET_B

IOMUXC_GPIO_SD_B0_06_REF_32K_OUT

IOMUXC_GPIO_SD_B0_06_SAI2_TX_SYNC

IOMUXC_GPIO_SD_B0_06_WDOG1_B

IOMUXC_GPIO_SD_B0_06_GPIO3_IO19

IOMUXC_GPIO_SD_B0_06_XBAR1_INOUT17

IOMUXC_GPIO_SD_B1_00_USDHC2_DATA2

IOMUXC_GPIO_SD_B1_00_FLEXSPI_B_DATA03

IOMUXC_GPIO_SD_B1_00_LPUART6_TX

IOMUXC_GPIO_SD_B1_00_XBAR1_INOUT10

IOMUXC_GPIO_SD_B1_00_FLEXCAN1_TX

IOMUXC_GPIO_SD_B1_00_GPIO3_IO20

IOMUXC_GPIO_SD_B1_01_USDHC2_DATA3

IOMUXC_GPIO_SD_B1_01_FLEXSPI_B_SCLK

IOMUXC_GPIO_SD_B1_01_LPUART6_RX

IOMUXC_GPIO_SD_B1_01_FLEXSPI_A_SS1_B

IOMUXC_GPIO_SD_B1_01_FLEXCAN1_RX

IOMUXC_GPIO_SD_B1_01_GPIO3_IO21

IOMUXC_GPIO_SD_B1_02_USDHC2_CMD

IOMUXC_GPIO_SD_B1_02_FLEXSPI_B_DATA00

IOMUXC_GPIO_SD_B1_02_LPUART8_TX

IOMUXC_GPIO_SD_B1_02_LPI2C4_SCL

IOMUXC_GPIO_SD_B1_02_ENET_1588_EVENT1_OUT

IOMUXC_GPIO_SD_B1_02_GPIO3_IO22

IOMUXC_GPIO_SD_B1_02_CCM_CLKO1

IOMUXC_GPIO_SD_B1_03_USDHC2_CLK

IOMUXC_GPIO_SD_B1_03_FLEXSPI_B_DATA02

IOMUXC_GPIO_SD_B1_03_LPUART8_RX

IOMUXC_GPIO_SD_B1_03_LPI2C4_SDA

IOMUXC_GPIO_SD_B1_03_ENET_1588_EVENT1_IN

IOMUXC_GPIO_SD_B1_03_GPIO3_IO23

IOMUXC_GPIO_SD_B1_03_CCM_CLKO2

IOMUXC_GPIO_SD_B1_04_USDHC2_DATA0

IOMUXC_GPIO_SD_B1_04_FLEXSPI_B_DATA01

IOMUXC_GPIO_SD_B1_04_ENET_TX_CLK

IOMUXC_GPIO_SD_B1_04_ENET_REF_CLK1

IOMUXC_GPIO_SD_B1_04_EWM_OUT_B

IOMUXC_GPIO_SD_B1_04_GPIO3_IO24

IOMUXC_GPIO_SD_B1_04_CCM_WAIT

IOMUXC_GPIO_SD_B1_05_USDHC2_DATA1

IOMUXC_GPIO_SD_B1_05_FLEXSPI_A_DQS

IOMUXC_GPIO_SD_B1_05_ENET_RDATA01

IOMUXC_GPIO_SD_B1_05_SAI3_MCLK

IOMUXC_GPIO_SD_B1_05_FLEXSPI_B_SS0_B

IOMUXC_GPIO_SD_B1_05_GPIO3_IO25

IOMUXC_GPIO_SD_B1_05_CCM_PMIC_RDY

IOMUXC_GPIO_SD_B1_06_USDHC2_CD_B

IOMUXC_GPIO_SD_B1_06_FLEXSPI_A_DATA03

IOMUXC_GPIO_SD_B1_06_ENET_RDATA00

IOMUXC_GPIO_SD_B1_06_SAI3_TX_BCLK

IOMUXC_GPIO_SD_B1_06_LPSPI2_PCS0

IOMUXC_GPIO_SD_B1_06_GPIO3_IO26

IOMUXC_GPIO_SD_B1_06_CCM_STOP

IOMUXC_GPIO_SD_B1_07_USDHC2_RESET_B

IOMUXC_GPIO_SD_B1_07_FLEXSPI_A_SCLK

IOMUXC_GPIO_SD_B1_07_ENET_RX_EN

IOMUXC_GPIO_SD_B1_07_SAI3_TX_SYNC

IOMUXC_GPIO_SD_B1_07_LPSPI2_SCK

IOMUXC_GPIO_SD_B1_07_GPIO3_IO27

IOMUXC_GPIO_SD_B1_08_USDHC2_DATA4

IOMUXC_GPIO_SD_B1_08_FLEXSPI_A_DATA00

IOMUXC_GPIO_SD_B1_08_ENET_RX_ER

IOMUXC_GPIO_SD_B1_08_SAI3_TX_DATA

IOMUXC_GPIO_SD_B1_08_LPSPI2_SDO

IOMUXC_GPIO_SD_B1_08_GPIO3_IO28

IOMUXC_GPIO_SD_B1_09_USDHC2_DATA5

IOMUXC_GPIO_SD_B1_09_FLEXSPI_A_DATA02

IOMUXC_GPIO_SD_B1_09_ENET_TX_EN

IOMUXC_GPIO_SD_B1_09_SAI3_RX_BCLK

IOMUXC_GPIO_SD_B1_09_LPSPI2_SDI

IOMUXC_GPIO_SD_B1_09_GPIO3_IO29

IOMUXC_GPIO_SD_B1_10_USDHC2_DATA6

IOMUXC_GPIO_SD_B1_10_FLEXSPI_A_DATA01

IOMUXC_GPIO_SD_B1_10_ENET_TDATA00

IOMUXC_GPIO_SD_B1_10_SAI3_RX_SYNC

IOMUXC_GPIO_SD_B1_10_LPSPI2_PCS2

IOMUXC_GPIO_SD_B1_10_GPIO3_IO30

IOMUXC_GPIO_SD_B1_11_USDHC2_DATA7

IOMUXC_GPIO_SD_B1_11_FLEXSPI_A_SS0_B

IOMUXC_GPIO_SD_B1_11_ENET_TDATA01

IOMUXC_GPIO_SD_B1_11_SAI3_RX_DATA

IOMUXC_GPIO_SD_B1_11_LPSPI2_PCS3

IOMUXC_GPIO_SD_B1_11_GPIO3_IO31

IOMUXC_SNVS_WAKEUP_GPIO5_IO00

IOMUXC_SNVS_WAKEUP_NMI_GLUE_NMI

IOMUXC_SNVS_PMIC_ON_REQ_SNVS_LP_PMIC_ON_REQ

IOMUXC_SNVS_PMIC_ON_REQ_GPIO5_IO01

IOMUXC_SNVS_PMIC_STBY_REQ_CCM_PMIC_VSTBY_REQ

IOMUXC_SNVS_PMIC_STBY_REQ_GPIO5_IO02

IOMUXC_SNVS_TEST_MODE

IOMUXC_SNVS_POR_B

IOMUXC_SNVS_ONOFF

IOMUXC_GPR_SAIMCLK_LOWBITMASK

IOMUXC_GPR_SAIMCLK_HIGHBITMASK

static inline void IOMUXC_SetPinMux(uint32_t muxRegister, uint32_t muxMode, uint32_t inputRegister, uint32_t inputDaisy, uint32_t configRegister, uint32_t inputOnfield)

Sets the IOMUXC pin mux mode.

This is an example to set the PTA6 as the lpuart0_tx:

```
IOMUXC_SetPinMux(IOMUXC_PTA6_LPUART0_TX, 0);
```

This is an example to set the PTA0 as GPIOA0:

```
IOMUXC_SetPinMux(IOMUXC_PTA0_GPIOA0, 0);
```

Note: The first five parameters can be filled with the pin function ID macros.

Parameters

- muxRegister – The pin mux register.
- muxMode – The pin mux mode.
- inputRegister – The select input register.
- inputDaisy – The input daisy.
- configRegister – The config register.
- inputOnfield – Software input on field.

static inline void IOMUXC_SetPinConfig(uint32_t muxRegister, uint32_t muxMode, uint32_t inputRegister, uint32_t inputDaisy, uint32_t configRegister, uint32_t configValue)

Sets the IOMUXC pin configuration.

This is an example to set pin configuration for IOMUXC_PTA3_LPI2C0_SCLS:

```
IOMUXC_SetPinConfig(IOMUXC_PTA3_LPI2C0_SCLS,IOMUXC_SW_PAD_CTL_PAD_PUS_
↪MASK|IOMUXC_SW_PAD_CTL_PAD_PUS(2U))
```

Note: The previous five parameters can be filled with the pin function ID macros.

Parameters

- muxRegister – The pin mux register.
- muxMode – The pin mux mode.
- inputRegister – The select input register.
- inputDaisy – The input daisy.
- configRegister – The config register.
- configValue – The pin config value.

static inline void IOMUXC_EnableMode(IOMUXC_GPR_Type *base, uint32_t mode, bool enable)

Sets IOMUXC general configuration for some mode.

Parameters

- base – The IOMUXC GPR base address.
- mode – The mode for setting. the mode is the logical OR of "iomuxc_gpr_mode"
- enable – True enable false disable.

static inline void IOMUXC_SetSaiMClkClockSource(IOMUXC_GPR_Type *base, *iomuxc_gpr_saimclk_t* mclk, uint8_t clkSrc)

Sets IOMUXC general configuration for SAI MCLK selection.

Parameters

- base – The IOMUXC GPR base address.
- mclk – The SAI MCLK.
- clkSrc – The clock source. Take refer to register setting details for the clock source in RM.

static inline void IOMUXC_MQSEnterSoftwareReset(IOMUXC_GPR_Type *base, bool enable)

Enters or exit MQS software reset.

Parameters

- base – The IOMUXC GPR base address.
- enable – Enter or exit MQS software reset.

static inline void IOMUXC_MQSEnable(IOMUXC_GPR_Type *base, bool enable)

Enables or disables MQS.

Parameters

- base – The IOMUXC GPR base address.
- enable – Enable or disable the MQS.

static inline void IOMUXC_MQSConfig(IOMUXC_GPR_Type *base, *iomuxc_mqs_pwm_oversample_rate_t* rate, uint8_t divider)

Configure MQS PWM oversampling rate compared with mclk and divider ratio control for mclk from hmclk.

Parameters

- base – The IOMUXC GPR base address.
- rate – The MQS PWM oversampling rate, refer to "iomuxc_mqs_pwm_oversample_rate_t".
- divider – The divider ratio control for mclk from hmclk. mclk freq = 1 /(divider + 1) * hmclk freq.

FSL_IOMUXC_DRIVER_VERSION

IOMUXC driver version 2.0.2.

FSL_COMPONENT_ID

2.38 KPP: KeyPad Port Driver

void KPP_Init(KPP_Type *base, *kpp_config_t* *configure)

KPP initialize. This function ungates the KPP clock and initializes KPP. This function must be called before calling any other KPP driver functions.

Parameters

- base – KPP peripheral base address.
- configure – The KPP configuration structure pointer.

void KPP_Deinit(KPP_Type *base)

Deinitializes the KPP module and gates the clock. This function gates the KPP clock. As a result, the KPP module doesn't work after calling this function.

Parameters

- base – KPP peripheral base address.

static inline void KPP_EnableInterrupts(KPP_Type *base, uint16_t mask)

Enable the interrupt.

Parameters

- base – KPP peripheral base address.
- mask – KPP interrupts to enable. This is a logical OR of the enumeration :: kpp_interrupt_enable_t.

static inline void KPP_DisableInterrupts(KPP_Type *base, uint16_t mask)

Disable the interrupt.

Parameters

- base – KPP peripheral base address.
- mask – KPP interrupts to disable. This is a logical OR of the enumeration :: kpp_interrupt_enable_t.

static inline uint16_t KPP_GetStatusFlag(KPP_Type *base)

Gets the KPP interrupt event status.

Parameters

- base – KPP peripheral base address.

Returns

The status of the KPP. Application can use the enum type in the "kpp_interrupt_enable_t" to get the right status of the related event.

static inline void KPP_ClearStatusFlag(KPP_Type *base, uint16_t mask)

Clears KPP status flag.

Parameters

- base – KPP peripheral base address.
- mask – KPP mask to be cleared. This is a logical OR of the enumeration :: kpp_interrupt_enable_t.

static inline void KPP_SetSynchronizeChain(KPP_Type *base, uint16_t mask)

Set KPP synchronization chain.

Parameters

- base – KPP peripheral base address.

- mask – KPP mask to be cleared. This is a logical OR of the enumeration :: kpp_sync_operation_t.

status_t KPP_keyPressScanning(KPP_Type *base, uint8_t *data, uint32_t clockSrc_Hz)

Keypad press scanning.

This function will scanning all columns and rows. so all scanning data will be stored in the data pointer.

Parameters

- base – KPP peripheral base address.
- data – KPP key press scanning data. The data buffer should be prepared with length at least equal to KPP_KEYPAD_COLUMNNUM_MAX * KPP_KEYPAD_ROWNUM_MAX. the data pointer is recommended to be a array like uint8_t data[KPP_KEYPAD_COLUMNNUM_MAX]. for example the data[2] = 4, that means in column 1 row 2 has a key press event.
- clockSrc_Hz – Source clock.

Return values

kStatus_Success – kpp press scan succeed.

FSL_KPP_DRIVER_VERSION

KPP driver version.

enum _kpp_interrupt_enable

List of interrupts supported by the peripheral. This enumeration uses one-bot encoding to allow a logical OR of multiple members. Members usually map to interrupt enable bits in one or more peripheral registers.

Values:

enumerator kKPP_keyDepressInterrupt

Keypad depress interrupt source

enumerator kKPP_keyReleaseInterrupt

Keypad release interrupt source

enum _kpp_sync_operation

Lists of KPP synchronize chain operation.

Values:

enumerator kKPP_ClearKeyDepressSyncChain

Keypad depress interrupt status.

enumerator kKPP_SetKeyReleasesSyncChain

Keypad release interrupt status.

typedef enum *_kpp_interrupt_enable* kpp_interrupt_enable_t

List of interrupts supported by the peripheral. This enumeration uses one-bot encoding to allow a logical OR of multiple members. Members usually map to interrupt enable bits in one or more peripheral registers.

typedef enum *_kpp_sync_operation* kpp_sync_operation_t
Lists of KPP synchronize chain operation.

typedef struct *_kpp_config* kpp_config_t
Lists of KPP status.

KPP_KEYPAD_COLUMNNUM_MAX

KPP_KEYPAD_ROWNUM_MAX

struct _kpp_config
#include <fsl_kpp.h> Lists of KPP status.

Public Members

uint8_t activeRow
The row number: bit 7 ~ 0 represents the row 7 ~ 0.

uint8_t activeColumn
The column number: bit 7 ~ 0 represents the column 7 ~ 0.

uint16_t interrupt
KPP interrupt source. A logical OR of "kpp_interrupt_enable_t".

2.39 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_L1DCACHE_ALIGN(var)

Macro to define a variable with L1 d-cache line size alignment

SDK_SIZEALIGN(var, alignbytes)

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*.

MCUX_CS

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.

CACHE_LINE_DATA

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.

2.40 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.

2.41 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.

2.42 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.

2.43 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.

2.44 LPSPI: Low Power Serial Peripheral Interface

2.45 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.

2.46 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

2.47 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 OCOTP: On Chip One-Time Programmable controller.

FSL_OCOTP_DRIVER_VERSION
OCOTP driver version.

_ocotp_status Error codes for the OCOTP driver.

Values:

enumerator kStatus_OCOTP_AccessError
eFuse and shadow register access error.

enumerator kStatus_OCOTP_CrcFail
CRC check failed.

enumerator kStatus_OCOTP_ReloadError
Error happens during reload shadow register.

enumerator kStatus_OCOTP_ProgramFail
Fuse programming failed.

enumerator kStatus_OCOTP_Locked
Fuse is locked and cannot be programmed.

typedef struct *_ocotp_timing* ocotp_timing_t

OCOTP timing structure. Note that, these value are used for calcalating the read/write timings. And the values should statisfy below rules:

Tsp_rd=(WAIT+1)/ipg_clk_freq should be >= 150ns; Tsp_pgm=(RELAX+1)/ipg_clk_freq should be >= 100ns; Trd = ((STROBE_READ+1)- 2*(RELAX_READ+1)) /ipg_clk_freq, The Trd is required to be larger than 40 ns. Tpgm = ((STROBE_PROG+1)- 2*(RELAX_PROG+1)) /ipg_clk_freq; The Tpgm should be configured within the range of 9000 ns < Tpgm < 11000 ns;

void OCOTP_Init(OCOTP_Type *base, uint32_t srcClock_Hz)

Initializes OCOTP controller.

Parameters

- base – OCOTP peripheral base address.
- srcClock_Hz – source clock frequency in unit of Hz. When the macro FSL_FEATURE_OCOTP_HAS_TIMING_CTRL is defined as 0, this parameter is not used, application could pass in 0 in this case.

void OCOTP_Deinit(OCOTP_Type *base)

De-initializes OCOTP controller.

Return values

kStatus_Success – upon successful execution, error status otherwise.

static inline bool OCOTP_CheckBusyStatus(OCOTP_Type *base)

Checking the BUSY bit in CTRL register. Checking this BUSY bit will help confirm if the OCOTP controller is ready for access.

Parameters

- base – OCOTP peripheral base address.

Return values

true – for bit set and false for cleared.

static inline bool OCOTP_CheckErrorStatus(OCOTP_Type *base)

Checking the ERROR bit in CTRL register.

Parameters

- base – OCOTP peripheral base address.

Return values

true – for bit set and false for cleared.

static inline void OCOTP_ClearErrorStatus(OCOTP_Type *base)

Clear the error bit if this bit is set.

Parameters

- base – OCOTP peripheral base address.

status_t OCOTP_ReloadShadowRegister(OCOTP_Type *base)

Reload the shadow register. This function will help reload the shadow register without reseting the OCOTP module. Please make sure the OCOTP has been initialized before calling this API.

Parameters

- base – OCOTP peripheral base addess.

Return values

- kStatus_Success – Reload success.

- kStatus_OCOTP_ReloadError – Reload failed.

uint32_t OCOTP_ReadFuse(OCOTP_Type *base, uint32_t address)

Read the fuse word with the fuse address.

Deprecated:

Use *OCOTP_ReadFuseExt* instead of this function.

Parameters

- base – OCOTP peripheral base address.
- address – the fuse address to be read from.

Returns

The read out data.

status_t OCOTP_ReadFuseExt(OCOTP_Type *base, uint32_t address, uint32_t *data, uint8_t fuseWords)

Read the fuse word from the fuse address.

This function reads fuse from address, how many words to read is specified by the parameter fuseWords. This function could read at most OCOTP_READ_FUSE_DATA_COUNT fuse word one time.

Parameters

- base – OCOTP peripheral base address.
- address – the fuse address to be read from.
- data – Data array to save the readout fuse value.
- fuseWords – How many words to read.

Return values

- kStatus_Success – Read success.
- kStatus_Fail – Error occurs during read.

status_t OCOTP_WriteFuse(OCOTP_Type *base, uint32_t address, uint32_t data)

Write the fuse word with the fuse address and data. Please make sure the write address is not locked while calling this API.

Warning: This function burns fuses permanently. Once fuses are burned, they cannot be reversed.

Parameters

- base – OCOTP peripheral base address.
- address – the fuse address to be written.
- data – the value will be writen to fuse address.

Return values

write – status, kStatus_Success for success and kStatus_Fail for failed.

status_t OCOTP_WriteFuseWithLock(OCOTP_Type *base, uint32_t address, uint32_t data, bool lock)

Write the fuse word and lock it.

Please make sure the write address is not locked while calling this API.

Some OCOTP controller supports ECC mode and redundancy mode (see reference mananual for more details). OCOTP controller will auto select ECC or redundancy mode to program the fuse word according to fuse map definition. In ECC mode, the 32 fuse bits in one word can only be written once. In redundancy mode, the word can be written more than once as long as they are different fuse bits. Set parameter lock as true to force use ECC mode.

Warning: This function burns fuses permanently. Once fuses are burned, they cannot be reversed.

Parameters

- base – OCOTP peripheral base address.
- address – The fuse address to be written.
- data – The value will be writen to fuse address.
- lock – Lock or unlock write fuse operation.

Return values

- kStatus_Success – Program and reload success.
- kStatus_OCOTP_Locked – The eFuse word is locked and cannot be programmed.
- kStatus_OCOTP_ProgramFail – eFuse word programming failed.
- kStatus_OCOTP_ReloadError – eFuse word programming success, but error happens during reload the values.
- kStatus_OCOTP_AccessError – Cannot access eFuse word.

static inline uint32_t OCOTP_GetVersion(OCOTP_Type *base)

Get the OCOTP controller version from the register.

Parameters

- base – OCOTP peripheral base address.

Return values

return – the version value.

OCOTP_READ_FUSE_DATA_COUNT

OCOTP_ReadFuseShadowRegister(base, address)

OCOTP_ReadFuseShadowRegisterExt(base, address, data, fuseWords)

OCOTP_WriteFuseShadowRegister(base, address, data)

OCOTP_WriteFuseShadowRegisterWithLock(base, address, data, lock)

struct _ocotp_timing

#include <fsl_ocotp.h> OCOTP timing structure. Note that, these value are used for calcalating the read/write timings. And the values should statisfy below rules:

Tsp_rd=(WAIT+1)/ipg_clk_freq should be >= 150ns; Tsp_pgm=(RELAX+1)/ipg_clk_freq should be >= 100ns; Trd = ((STROBE_READ+1)- 2*(RELAX_READ+1)) /ipg_clk_freq, The Trd is required to be larger than 40 ns. Tpgm = ((STROBE_PROG+1)- 2*(RELAX_PROG+1))

/ipg_clk_freq; The Tpgm should be configured within the range of 9000 ns < Tpgm < 11000 ns;

Public Members

uint32_t wait

Wait time value to fill in the TIMING register.

uint32_t relax

Relax time value to fill in the TIMING register.

uint32_t strobe_prog

Storbe program time value to fill in the TIMING register.

uint32_t strobe_read

Storbe read time value to fill in the TIMING register.

2.51 PIT: Periodic Interrupt Timer

void PIT_Init(PIT_Type *base, const *pit_config_t* *config)

Ungates the PIT clock, enables the PIT module, and configures the peripheral for basic operations.

Note: This API should be called at the beginning of the application using the PIT driver.

Parameters

- base – PIT peripheral base address
- config – Pointer to the user's PIT config structure

void PIT_Deinit(PIT_Type *base)

Gates the PIT clock and disables the PIT module.

Parameters

- base – PIT peripheral base address

static inline void PIT_GetDefaultConfig(*pit_config_t* *config)

Fills in the PIT configuration structure with the default settings.

The default values are as follows.

```
config->enableRunInDebug = false;
```

Parameters

- config – Pointer to the configuration structure.

static inline void PIT_SetTimerChainMode(PIT_Type *base, *pit_chnl_t* channel, bool enable)

Enables or disables chaining a timer with the previous timer.

When a timer has a chain mode enabled, it only counts after the previous timer has expired. If the timer n-1 has counted down to 0, counter n decrements the value by one. Each timer is 32-bits, which allows the developers to chain timers together and form a longer timer (64-bits and larger). The first timer (timer 0) can't be chained to any other timer.

Parameters

- base – PIT peripheral base address
- channel – Timer channel number which is chained with the previous timer
- enable – Enable or disable chain. true: Current timer is chained with the previous timer. false: Timer doesn't chain with other timers.

static inline void PIT_EnableInterrupts(PIT_Type *base, *pit_chnl_t* channel, uint32_t mask)

Enables the selected PIT interrupts.

Parameters

- base – PIT peripheral base address
- channel – Timer channel number
- mask – The interrupts to enable. This is a logical OR of members of the enumeration *pit_interrupt_enable_t*

static inline void PIT_DisableInterrupts(PIT_Type *base, *pit_chnl_t* channel, uint32_t mask)

Disables the selected PIT interrupts.

Parameters

- base – PIT peripheral base address
- channel – Timer channel number
- mask – The interrupts to disable. This is a logical OR of members of the enumeration *pit_interrupt_enable_t*

static inline uint32_t PIT_GetEnabledInterrupts(PIT_Type *base, *pit_chnl_t* channel)

Gets the enabled PIT interrupts.

Parameters

- base – PIT peripheral base address
- channel – Timer channel number

Returns

The enabled interrupts. This is the logical OR of members of the enumeration *pit_interrupt_enable_t*

static inline uint32_t PIT_GetStatusFlags(PIT_Type *base, *pit_chnl_t* channel)

Gets the PIT status flags.

Parameters

- base – PIT peripheral base address
- channel – Timer channel number

Returns

The status flags. This is the logical OR of members of the enumeration *pit_status_flags_t*

static inline void PIT_ClearStatusFlags(PIT_Type *base, *pit_chnl_t* channel, uint32_t mask)

Clears the PIT status flags.

Parameters

- base – PIT peripheral base address
- channel – Timer channel number
- mask – The status flags to clear. This is a logical OR of members of the enumeration *pit_status_flags_t*

static inline void PIT_SetTimerPeriod(PIT_Type *base, *pit_chnl_t* channel, uint32_t count)

Sets the timer period in units of count.

Timers begin counting from the value set by this function until it reaches 0, then it generates an interrupt and load 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: Users can call the utility macros provided in fsl_common.h to convert to ticks.

Parameters

- base – PIT peripheral base address
- channel – Timer channel number
- count – Timer period in units of ticks

static inline uint32_t PIT_GetCurrentTimerCount(PIT_Type *base, *pit_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: Users can call the utility macros provided in fsl_common.h to convert ticks to usec or msec.

Parameters

- base – PIT peripheral base address
- channel – Timer channel number

Returns

Current timer counting value in ticks

static inline void PIT_StartTimer(PIT_Type *base, *pit_chnl_t* channel)

Starts the timer counting.

After calling this function, timers load period value, count down to 0 and then load the respective start value again. Each time a timer reaches 0, it generates a trigger pulse and sets the timeout interrupt flag.

Parameters

- base – PIT peripheral base address
- channel – Timer channel number.

static inline void PIT_StopTimer(PIT_Type *base, *pit_chnl_t* channel)

Stops the timer counting.

This function stops every timer counting. Timers reload their periods respectively after the next time they call the PIT_DRV_StartTimer.

Parameters

- base – PIT peripheral base address
- channel – Timer channel number.

FSL_PIT_DRIVER_VERSION

PIT Driver Version 2.2.0.

enum _pit_chnl

List of PIT channels.

Note: Actual number of available channels is SoC dependent

Values:

enumerator kPIT_Chnl_0

PIT channel number 0

enumerator kPIT_Chnl_1

PIT channel number 1

enumerator kPIT_Chnl_2

PIT channel number 2

enumerator kPIT_Chnl_3

PIT channel number 3

enum _pit_interrupt_enable

List of PIT interrupts.

Values:

enumerator kPIT_TimerInterruptEnable

Timer interrupt enable

enum _pit_status_flags

List of PIT status flags.

Values:

enumerator kPIT_TimerFlag

Timer flag

typedef enum *_pit_chnl* pit_chnl_t

List of PIT channels.

Note: Actual number of available channels is SoC dependent

typedef enum *_pit_interrupt_enable* pit_interrupt_enable_t

List of PIT interrupts.

typedef enum *_pit_status_flags* pit_status_flags_t

List of PIT status flags.

typedef struct *_pit_config* pit_config_t

PIT configuration structure.

This structure holds the configuration settings for the PIT peripheral. To initialize this structure to reasonable defaults, call the *PIT_GetDefaultConfig()* function and pass a pointer to your config structure instance.

The configuration structure can be made constant so it resides in flash.

uint64_t PIT_GetLifetimeTimerCount(PIT_Type *base)

Reads the current lifetime counter value.

The lifetime timer is a 64-bit timer which chains timer 0 and timer 1 together. Timer 0 and 1 are chained by calling the PIT_SetTimerChainMode before using this timer. The period of lifetime timer is equal to the "period of timer 0 * period of timer 1". For the 64-bit value, the higher 32-bit has the value of timer 1, and the lower 32-bit has the value of timer 0.

Parameters

- base – PIT peripheral base address

Returns

Current lifetime timer value

struct _pit_config

#include <fsl_pit.h> PIT configuration structure.

This structure holds the configuration settings for the PIT peripheral. To initialize this structure to reasonable defaults, call the *PIT_GetDefaultConfig()* function and pass a pointer to your config structure instance.

The configuration structure can be made constant so it resides in flash.

Public Members

bool enableRunInDebug

true: Timers run in debug mode; false: Timers stop in debug mode

2.52 PMU: Power Management Unit

uint32_t PMU_GetStatusFlags(PMU_Type *base)

Get PMU status flags.

Parameters

- base – PMU peripheral base address.

Returns

PMU status flags.It indicate if regulator output of 1P1,3P0 and 2P5 is ok and brownout output of 1P1,3P0 and 2P5 is detected.

static inline void PMU_1P1SetWeakReferenceSource(PMU_Type *base, *pmu_1p1_weak_reference_source_t* option)

Selects the source for the reference voltage of the weak 1P1 regulator.

Parameters

- base – PMU peripheral base address.
- option – The option for reference voltage source, see to *pmu_1p1_weak_reference_source_t*.

static inline void PMU_1P1EnableWeakRegulator(PMU_Type *base, bool enable)

Enables the weak 1P1 regulator.

This regulator can be used when the main 1P1 regulator is disabled, under low-power conditions.

Parameters

- base – PMU peripheral base address.
- enable – Enable the feature or not.

static inline void PMU_1P1SetRegulatorOutputVoltage(PMU_Type *base, uint32_t value)

Adjust the 1P1 regulator output voltage.

Each LSB is worth 25mV. Programming examples are detailed below. Other output target voltages may be interpolated from these examples. Choices must be in this range:

- 0x1b(1.375V) >= output_trg >= 0x04(0.8V)
- 0x04 : 0.8V
- 0x10 : 1.1V (typical)
- 0x1b : 1.375V NOTE: There may be reduced chip functionality or reliability at the extremes of the programming range.

Parameters

- base – PMU peripheral base address.
- value – Setting value for the output.

static inline void PMU_1P1SetBrownoutOffsetVoltage(PMU_Type *base, uint32_t value)

Adjust the 1P1 regulator brownout offset voltage.

Control bits to adjust the regulator brownout offset voltage in 25mV steps. The reset brownoffset is 175mV below the programmed target code. Brownout target = OUTPUT_TRG - BO_OFFSET. Some steps may be irrelevant because of input supply limitations or load operation.

Parameters

- base – PMU peripheral base address.
- value – Setting value for the brownout offset. The available range is in 3-bit.

static inline void PMU_1P1EnablePullDown(PMU_Type *base, bool enable)

Enable the pull-down circuitry in the regulator.

Parameters

- base – PMU peripheral base address.
- enable – Enable the feature or not.

static inline void PMU_1P1EnableCurrentLimit(PMU_Type *base, bool enable)

Enable the current-limit circuitry in the regulator.

Parameters

- base – PMU peripheral base address.

- enable – Enable the feature or not.

static inline void PMU_1P1EnableBrownout(PMU_Type *base, bool enable)

Enable the brownout circuitry in the regulator.

Parameters

- base – PMU peripheral base address.
- enable – Enable the feature or not.

static inline void PMU_1P1EnableOutput(PMU_Type *base, bool enable)

Enable the regulator output.

Parameters

- base – PMU peripheral base address.
- enable – Enable the feature or not.

static inline void PMU_3P0SetRegulatorOutputVoltage(PMU_Type *base, uint32_t value)

Adjust the 3P0 regulator output voltage.

Each LSB is worth 25mV. Programming examples are detailed below. Other output target voltages may be interpolated from these examples. Choices must be in this range:

- 0x00(2.625V) >= output_trg >= 0x1f(3.4V)
- 0x00 : 2.625V
- 0x0f : 3.0V (typical)
- 0x1f : 3.4V

Parameters

- base – PMU peripheral base address.
- value – Setting value for the output.

static inline void PMU_3P0SetVBusVoltageSource(PMU_Type *base, *pmu_3p0_vbus_voltage_source_t* option)

Select input voltage source for LDO_3P0.

Select input voltage source for LDO_3P0 from either USB_OTG1_VBUS or USB_OTG2_VBUS. If only one of the two VBUS voltages is present, it is automatically selected.

Parameters

- base – PMU peripheral base address.
- option – User-defined input voltage source for LDO_3P0.

static inline void PMU_3P0SetBrownoutOffsetVoltage(PMU_Type *base, uint32_t value)

Adjust the 3P0 regulator brownout offset voltage.

Control bits to adjust the 3P0 regulator brownout offset voltage in 25mV steps. The reset brown-offset is 175mV below the programmed target code. Brownout target = OUTPUT_TRG - BO_OFFSET. Some steps may be irrelevant because of input supply limitations or load operation.

Parameters

- base – PMU peripheral base address.
- value – Setting value for the brownout offset. The available range is in 3-bit.

static inline void PMU_3P0EnableCurrentLimit(PMU_Type *base, bool enable)

Enable the current-limit circuitry in the 3P0 regulator.

Parameters

- base – PMU peripheral base address.
- enable – Enable the feature or not.

static inline void PMU_3P0EnableBrownout(PMU_Type *base, bool enable)

Enable the brownout circuitry in the 3P0 regulator.

Parameters

- base – PMU peripheral base address.
- enable – Enable the feature or not.

static inline void PMU_3P0EnableOutput(PMU_Type *base, bool enable)

Enable the 3P0 regulator output.

Parameters

- base – PMU peripheral base address.
- enable – Enable the feature or not.

static inline void PMU_2P5EnableWeakRegulator(PMU_Type *base, bool enable)

Enables the weak 2P5 regulator.

This low power regulator is used when the main 2P5 regulator is disabled to keep the 2.5V output roughly at 2.5V. Scales directly with the value of VDDHIGH_IN.

Parameters

- base – PMU peripheral base address.
- enable – Enable the feature or not.

static inline void PMU_2P5SetRegulatorOutputVoltage(PMU_Type *base, uint32_t value)

Adjust the 1P1 regulator output voltage.

Each LSB is worth 25mV. Programming examples are detailed below. Other output target voltages may be interpolated from these examples. Choices must be in this range:

- 0x00(2.1V) >= output_trg >= 0x1f(2.875V)
- 0x00 : 2.1V
- 0x10 : 2.5V (typical)
- 0x1f : 2.875V NOTE: There may be reduced chip functionality or reliability at the extremes of the programming range.

Parameters

- base – PMU peripheral base address.
- value – Setting value for the output.

static inline void PMU_2P5SetBrownoutOffsetVoltage(PMU_Type *base, uint32_t value)

Adjust the 2P5 regulator brownout offset voltage.

Adjust the regulator brownout offset voltage in 25mV steps. The reset brown-offset is 175mV below the programmed target code. Brownout target = OUTPUT_TRG - BO_OFFSET. Some steps may be irrelevant because of input supply limitations or load operation.

Parameters

- base – PMU peripheral base address.

- value – Setting value for the brownout offset. The available range is in 3-bit.

static inline void PMU_2P5EnablePullDown(PMU_Type *base, bool enable)

Enable the pull-down circuitry in the 2P5 regulator.

Parameters

- base – PMU peripheral base address.
- enable – Enable the feature or not.

static inline void PMU_2P1EnablePullDown(PMU_Type *base, bool enable)

Enable the pull-down circuitry in the 2P5 regulator.

Deprecated:

Do not use this function. It has been superceded by *PMU_2P5EnablePullDown*.

static inline void PMU_2P5EnableCurrentLimit(PMU_Type *base, bool enable)

Enable the current-limit circuitry in the 2P5 regulator.

Parameters

- base – PMU peripheral base address.
- enable – Enable the feature or not.

static inline void PMU_2P5nableBrownout(PMU_Type *base, bool enable)

Enable the brownout circuitry in the 2P5 regulator.

Parameters

- base – PMU peripheral base address.
- enable – Enable the feature or not.

static inline void PMU_2P5EnableOutput(PMU_Type *base, bool enable)

Enable the 2P5 regulator output.

Parameters

- base – PMU peripheral base address.
- enable – Enable the feature or not.

static inline void PMU_CoreEnableIncreaseGateDrive(PMU_Type *base, bool enable)

Increase the gate drive on power gating FETs.

If set, increases the gate drive on power gating FETs to reduce leakage in the off state. Care must be taken to apply this bit only when the input supply voltage to the power FET is less than 1.1V. NOTE: This bit should only be used in low-power modes where the external input supply voltage is nominally 0.9V.

Parameters

- base – PMU peripheral base address.
- enable – Enable the feature or not.

static inline void PMU_CoreSetRegulatorVoltageRampRate(PMU_Type *base, *pmu_core_reg_voltage_ramp_rate_t* option)

Set the CORE regulator voltage ramp rate.

Parameters

- base – PMU peripheral base address.

- option – User-defined option for voltage ramp rate, see to *pmu_core_reg_voltage_ramp_rate_t*.

static inline void PMU_CoreSetSOCDomainVoltage(PMU_Type *base, uint32_t value)

Define the target voltage for the SOC power domain.

Define the target voltage for the SOC power domain. Single-bit increments reflect 25mV core voltage steps. Some steps may not be relevant because of input supply limitations or load operation.

- 0x00 : Power gated off.
- 0x01 : Target core voltage = 0.725V
- 0x02 : Target core voltage = 0.750V
- ...
- 0x10 : Target core voltage = 1.100V
- ...
- 0x1e : Target core voltage = 1.450V
- 0x1F : Power FET switched full on. No regulation. NOTE: This register is capable of programming an over-voltage condition on the device. Consult the datasheet Operating Ranges table for the allowed voltages.

Parameters

- base – PMU peripheral base address.
- value – Setting value for target voltage. 5-bit available

static inline void PMU_CoreSetARMCoreDomainVoltage(PMU_Type *base, uint32_t value)

Define the target voltage for the ARM Core power domain.

Define the target voltage for the ARM Core power domain. Single-bit increments reflect 25mV core voltage steps. Some steps may not be relevant because of input supply limitations or load operation.

- 0x00 : Power gated off.
- 0x01 : Target core voltage = 0.725V
- 0x02 : Target core voltage = 0.750V
- ...
- 0x10 : Target core voltage = 1.100V
- ...
- 0x1e : Target core voltage = 1.450V
- 0x1F : Power FET switched full on. No regulation. NOTE: This register is capable of programming an over-voltage condition on the device. Consult the datasheet Operating Ranges table for the allowed voltages.

Parameters

- base – PMU peripheral base address.
- value – Setting value for target voltage. 5-bit available

FSL_PMU_DRIVER_VERSION

PMU driver version.

Version 2.1.1.

PMU Status flags.

Values:

enumerator kPMU_1P1RegulatorOutputOK

Status bit that signals when the 1p1 regulator output is ok. 1 = regulator output > brownout target.

enumerator kPMU_1P1BrownoutOnOutput

Status bit that signals when a 1p1 brownout is detected on the regulator output.

enumerator kPMU_3P0RegulatorOutputOK

Status bit that signals when the 3p0 regulator output is ok. 1 = regulator output > brownout target.

enumerator kPMU_3P0BrownoutOnOutput

Status bit that signals when a 3p0 brownout is detected on the regulator output.

enumerator kPMU_2P5RegulatorOutputOK

Status bit that signals when the 2p5 regulator output is ok. 1 = regulator output > brownout target.

enumerator kPMU_2P5BrownoutOnOutput

Status bit that signals when a 2p5 brownout is detected on the regulator output.

enum _pmu_1p1_weak_reference_source

The source for the reference voltage of the weak 1P1 regulator.

Values:

enumerator kPMU_1P1WeakReferenceSourceAlt0

Weak-linreg output tracks low-power-bandgap voltage.

enumerator kPMU_1P1WeakReferenceSourceAlt1

Weak-linreg output tracks VDD_SOC_CAP voltage.

enum _pmu_3p0_vbus_voltage_source

Input voltage source for LDO_3P0 from USB VBus.

Values:

enumerator kPMU_3P0VBusVoltageSourceAlt0

USB_OTG1_VBUS - Utilize VBUS OTG1 for power.

enumerator kPMU_3P0VBusVoltageSourceAlt1

USB_OTG2_VBUS - Utilize VBUS OTG2 for power.

enum _pmu_core_reg_voltage_ramp_rate

Regulator voltage ramp rate.

Values:

enumerator kPMU_CoreRegVoltageRampRateFast
Fast.

enumerator kPMU_CoreRegVoltageRampRateMediumFast
Medium Fast.

enumerator kPMU_CoreRegVoltageRampRateMediumSlow
Medium Slow.

enumerator kPMU_CoreRegVoltageRampRateSlow
Slow.

enum _pmu_power_bandgap
Bandgap select.
Values:

enumerator kPMU_NormalPowerBandgap
Normal power bandgap.

enumerator kPMU_LowPowerBandgap
Low power bandgap.

typedef enum *_pmu_1p1_weak_reference_source* pmu_1p1_weak_reference_source_t
The source for the reference voltage of the weak 1P1 regulator.

typedef enum *_pmu_3p0_vbus_voltage_source* pmu_3p0_vbus_voltage_source_t
Input voltage source for LDO_3P0 from USB VBus.

typedef enum *_pmu_core_reg_voltage_ramp_rate* pmu_core_reg_voltage_ramp_rate_t
Regulator voltage ramp rate.

typedef enum *_pmu_power_bandgap* pmu_power_bandgap_t
Bandgap select.

2.53 PWM: Pulse Width Modulator

status_t PWM_Init(PWM_Type *base, *pwm_submodule_t* subModule, const *pwm_config_t* *config)
Ungates the PWM submodule clock and configures the peripheral for basic operation.

This API should be called at the beginning of the application using the PWM driver. When user select PWMX, user must choose edge aligned output, becasue there are some limitation on center aligned PWMX output. When output PWMX in center aligned mode, VAL1 register controls both PWM period and PWMX duty cycle, PWMA and PWMB output will be corrupted. But edge aligned PWMX output do not have such limit. In master reload counter initialization mode, PWM period is depended by period of set LDOK in submodule 0 because this operation will reload register. Submodule 0 counter initialization cannot be master sync or master reload.

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- config – Pointer to user's PWM config structure.

Returns
kStatus_Success means success; else failed.

void PWM_Deinit(PWM_Type *base, *pwm_submodule_t* subModule)

Gate the PWM submodule clock.

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to deinitialize

void PWM_GetDefaultConfig(*pwm_config_t* *config)

Fill in the PWM config struct with the default settings.

The default values are:

```
config->enableDebugMode = false;
config->enableWait = false;
config->reloadSelect = kPWM_LocalReload;
config->clockSource = kPWM_BusClock;
config->prescale = kPWM_Prescale_Divide_1;
config->initializationControl = kPWM_Initialize_LocalSync;
config->forceTrigger = kPWM_Force_Local;
config->reloadFrequency = kPWM_LoadEveryOportunity;
config->reloadLogic = kPWM_ReloadImmediate;
config->pairOperation = kPWM_Independent;
```

Parameters

- config – Pointer to user's PWM config structure.

status_t PWM_SetupPwm(PWM_Type *base, *pwm_submodule_t* subModule, const *pwm_signal_param_t* *chnlParams, uint8_t numOfChnls, *pwm_mode_t* mode, uint32_t pwmFreq_Hz, uint32_t srcClock_Hz)

Sets up the PWM signals for a PWM submodule.

The function initializes the submodule according to the parameters passed in by the user. The function also sets up the value compare registers to match the PWM signal requirements. If the dead time insertion logic is enabled, the pulse period is reduced by the dead time period specified by the user. When user select PWMX, user must choose edge aligned output, becasue there are some limitation on center aligned PWMX output. Due to edge aligned PWMX is negative true signal, need to configure PWMX active low true level to get correct duty cycle. The half cycle point will not be exactly in the middle of the PWM cycle when PWMX enabled.

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- 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. Array size should not be more than 3 as each submodule has 3 pins to output PWM.
- mode – PWM operation mode, options available in enumeration *pwm_mode_t*

- pwmFreq_Hz – PWM signal frequency in Hz
- srcClock_Hz – PWM source clock of correspond submodule in Hz. If source clock of submodule1,2,3 is from submodule0 AUX_CLK, its source clock is submodule0 source clock divided with submodule0 prescaler value instead of submodule0 source clock.

Returns

Returns kStatus_Fail if there was error setting up the signal; kStatus_Success otherwise

status_t PWM_SetupPwmPhaseShift(PWM_Type *base, *pwm_submodule_t* subModule, *pwm_channels_t* pwmChannel, uint32_t pwmFreq_Hz, uint32_t srcClock_Hz, uint8_t shiftvalue, bool doSync)

Set PWM phase shift for PWM channel running on channel PWM_A, PWM_B which with 50% duty cycle.

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- pwmChannel – PWM channel to configure
- pwmFreq_Hz – PWM signal frequency in Hz
- srcClock_Hz – PWM main counter clock in Hz.
- shiftvalue – Phase shift value, range in 0 ~ 50
- doSync – true: Set LDOK bit for the submodule list; false: LDOK bit don't set, need to call PWM_SetPwmLdok to sync update.

Returns

Returns kStatus_Fail if there was error setting up the signal; kStatus_Success otherwise

void PWM_UpdatePwmDutycycle(PWM_Type *base, *pwm_submodule_t* subModule, *pwm_channels_t* pwmSignal, *pwm_mode_t* currPwmMode, uint8_t dutyCyclePercent)

Updates the PWM signal's dutycycle.

The function updates the PWM dutycyle to the new value that is passed in. If the dead time insertion logic is enabled then the pulse period is reduced by the dead time period specified by the user.

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- pwmSignal – Signal (PWM A, PWM B, PWM X) to update
- currPwmMode – 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)

void PWM_UpdatePwmDutycycleHighAccuracy(PWM_Type *base, *pwm_submodule_t* subModule, *pwm_channels_t* pwmSignal, *pwm_mode_t* currPwmMode, uint16_t dutyCycle)

Updates the PWM signal's dutycycle with 16-bit accuracy.

The function updates the PWM dutycyle to the new value that is passed in. If the dead time insertion logic is enabled then the pulse period is reduced by the dead time period specified by the user.

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- pwmSignal – Signal (PWM A, PWM B, PWM X) to update
- currPwmMode – The current PWM mode set during PWM setup
- dutyCycle – New PWM pulse width, value should be between 0 to 65535 0=inactive signal(0% duty cycle)... 65535=active signal (100% duty cycle)

void PWM_UpdatePwmPeriodAndDutycycle(PWM_Type *base, *pwm_submodule_t* subModule, *pwm_channels_t* pwmSignal, *pwm_mode_t* currPwmMode, uint16_t pulseCnt, uint16_t dutyCycle)

Update the PWM signal's period and dutycycle for a PWM submodule.

The function updates PWM signal period generated by a specific submodule according to the parameters passed in by the user. This function can also set dutycycle weather you want to keep original dutycycle or update new dutycycle. Call this function in local sync control mode because PWM period is depended by

INIT and VAL1 register of each submodule. In master sync initialization control mode, call this function to update INIT and VAL1 register of all submodule because PWM period is depended by INIT and VAL1 register in submodule0. If the dead time insertion logic is enabled, the pulse period is reduced by the dead time period specified by the user. PWM signal will not be generated if its period is less than dead time duration.

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- pwmSignal – Signal (PWM A or PWM B) to update
- currPwmMode – The current PWM mode set during PWM setup, options available in enumeration *pwm_mode_t*
- pulseCnt – New PWM period, value should be between 0 to 65535 0=minimum PWM period... 65535=maximum PWM period
- dutyCycle – New PWM pulse width of channel, value should be between 0 to 65535 0=inactive signal(0% duty cycle)... 65535=active signal (100% duty cycle) You can keep original duty cycle or update new duty cycle

static inline void PWM_EnableInterrupts(PWM_Type *base, *pwm_submodule_t* subModule, uint32_t mask)

Enables the selected PWM interrupts.

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- mask – The interrupts to enable. This is a logical OR of members of the enumeration *pwm_interrupt_enable_t*

static inline void PWM_DisableInterrupts(PWM_Type *base, *pwm_submodule_t* subModule, uint32_t mask)

Disables the selected PWM interrupts.

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- mask – The interrupts to enable. This is a logical OR of members of the enumeration *pwm_interrupt_enable_t*

static inline uint32_t PWM_GetEnabledInterrupts(PWM_Type *base, *pwm_submodule_t* subModule)

Gets the enabled PWM interrupts.

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure

Returns

The enabled interrupts. This is the logical OR of members of the enumeration *pwm_interrupt_enable_t*

static inline void PWM_DMAFIFOWatermarkControl(PWM_Type *base, *pwm_submodule_t* subModule, *pwm_watermark_control_t* pwm_watermark_control)

Capture DMA Enable Source Select.

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- pwm_watermark_control – PWM FIFO watermark and control

static inline void PWM_DMACaptureSourceSelect(PWM_Type *base, *pwm_submodule_t* subModule, *pwm_dma_source_select_t* pwm_dma_source_select)

Capture DMA Enable Source Select.

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- pwm_dma_source_select – PWM capture DMA enable source select

static inline void PWM_EnableDMACapture(PWM_Type *base, *pwm_submodule_t* subModule, uint16_t mask, bool activate)

Enables or disables the selected PWM DMA Capture read request.

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- mask – The DMA to enable or disable. This is a logical OR of members of the enumeration *pwm_dma_enable_t*
- activate – true: Enable DMA read request; false: Disable DMA read request

static inline void PWM_EnableDMAWrite(PWM_Type *base, *pwm_submodule_t* subModule, bool activate)

Enables or disables the PWM DMA write request.

Parameters

- base – PWM peripheral base address

- subModule – PWM submodule to configure
- activate – true: Enable DMA write request; false: Disable DMA write request

static inline uint32_t PWM_GetStatusFlags(PWM_Type *base, *pwm_submodule_t* subModule)

Gets the PWM status flags.

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure

Returns

The status flags. This is the logical OR of members of the enumeration *pwm_status_flags_t*

static inline void PWM_ClearStatusFlags(PWM_Type *base, *pwm_submodule_t* subModule, uint32_t mask)

Clears the PWM status flags.

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- mask – The status flags to clear. This is a logical OR of members of the enumeration *pwm_status_flags_t*

static inline void PWM_StartTimer(PWM_Type *base, uint8_t subModulesToStart)

Starts the PWM counter for a single or multiple submodules.

Sets the Run bit which enables the clocks to the PWM submodule. This function can start multiple submodules at the same time.

Parameters

- base – PWM peripheral base address
- subModulesToStart – PWM submodules to start. This is a logical OR of members of the enumeration *pwm_module_control_t*

static inline void PWM_StopTimer(PWM_Type *base, uint8_t subModulesToStop)

Stops the PWM counter for a single or multiple submodules.

Clears the Run bit which resets the submodule's counter. This function can stop multiple submodules at the same time.

Parameters

- base – PWM peripheral base address
- subModulesToStop – PWM submodules to stop. This is a logical OR of members of the enumeration *pwm_module_control_t*

FSL_PWM_DRIVER_VERSION

Version 2.10.1

enum _pwm_submodule

List of PWM submodules.

Values:

enumerator kPWM_Module_0
Submodule 0

enumerator kPWM_Module_1
Submodule 1

enumerator kPWM_Module_2
Submodule 2

enum _pwm_channels
List of PWM channels in each module.
Values:

enumerator kPWM_PwmB

enumerator kPWM_PwmA

enumerator kPWM_PwmX

enum _pwm_value_register
List of PWM value registers.
Values:

enumerator kPWM_ValueRegister_0
PWM Value0 register

enumerator kPWM_ValueRegister_1
PWM Value1 register

enumerator kPWM_ValueRegister_2
PWM Value2 register

enumerator kPWM_ValueRegister_3
PWM Value3 register

enumerator kPWM_ValueRegister_4
PWM Value4 register

enumerator kPWM_ValueRegister_5
PWM Value5 register

enum _pwm_value_register_mask
List of PWM value registers mask.
Values:

enumerator kPWM_ValueRegisterMask_0
PWM Value0 register mask

enumerator kPWM_ValueRegisterMask_1
PWM Value1 register mask

enumerator kPWM_ValueRegisterMask_2
PWM Value2 register mask

enumerator kPWM_ValueRegisterMask_3
PWM Value3 register mask

enumerator kPWM_ValueRegisterMask_4
PWM Value4 register mask

enumerator kPWM_ValueRegisterMask_5
PWM Value5 register mask

enum __pwm_clock_source
PWM clock source selection.
Values:

enumerator kPWM_BusClock
Device specific IPBus clock, refer reference manual for frequency

enumerator kPWM_ExternalClock
EXT_CLK is used as the clock

enumerator kPWM_Submodule0Clock
Clock of the submodule 0 (AUX_CLK) is used as the source clock

enum __pwm_clock_prescale
PWM prescaler factor selection for clock source.
Values:

enumerator kPWM_Prescale_Divide_1
PWM clock frequency = fclk/1

enumerator kPWM_Prescale_Divide_2
PWM clock frequency = fclk/2

enumerator kPWM_Prescale_Divide_4
PWM clock frequency = fclk/4

enumerator kPWM_Prescale_Divide_8
PWM clock frequency = fclk/8

enumerator kPWM_Prescale_Divide_16
PWM clock frequency = fclk/16

enumerator kPWM_Prescale_Divide_32
PWM clock frequency = fclk/32

enumerator kPWM_Prescale_Divide_64
PWM clock frequency = fclk/64

enumerator kPWM_Prescale_Divide_128
PWM clock frequency = fclk/128

enum __pwm_force_output_trigger
Options that can trigger a PWM FORCE_OUT.
Values:

enumerator kPWM_Force_Local
The local force signal, CTRL2[FORCE], from the submodule is used to force updates

enumerator kPWM_Force_Master
The master force signal from submodule 0 is used to force updates

enumerator kPWM_Force_LocalReload
The local reload signal from this submodule is used to force updates without regard to the state of LDOK

enumerator kPWM_Force_MasterReload
The master reload signal from submodule 0 is used to force updates if LDOK is set

enumerator kPWM_Force_LocalSync
The local sync signal from this submodule is used to force updates

enumerator kPWM_Force_MasterSync
The master sync signal from submodule0 is used to force updates

enumerator kPWM_Force_External
The external force signal, EXT_FORCE, from outside the PWM module causes updates

enumerator kPWM_Force_ExternalSync
The external sync signal, EXT_SYNC, from outside the PWM module causes updates

enum __pwm_output_state
PWM channel output status.
Values:

enumerator kPWM_HighState
The output state of PWM channel is high

enumerator kPWM_LowState
The output state of PWM channel is low

enumerator kPWM_NormalState
The output state of PWM channel is normal

enumerator kPWM_InvertState
The output state of PWM channel is invert

enumerator kPWM_MaskState
The output state of PWM channel is mask

enum __pwm_init_source
PWM counter initialization options.
Values:

enumerator kPWM_Initialize_LocalSync
Local sync causes initialization

enumerator kPWM_Initialize_MasterReload
Master reload from submodule 0 causes initialization

enumerator kPWM_Initialize_MasterSync
Master sync from submodule 0 causes initialization

enumerator kPWM_Initialize_ExtSync
EXT_SYNC causes initialization

enum __pwm_load_frequency
PWM load frequency selection.
Values:

enumerator kPWM_LoadEveryOportunity
Every PWM opportunity

enumerator kPWM_LoadEvery2Oportunity
Every 2 PWM opportunities

enumerator kPWM_LoadEvery3Oportunity
Every 3 PWM opportunities

enumerator kPWM_LoadEvery4Oportunity
Every 4 PWM opportunities

enumerator kPWM_LoadEvery5Oportunity
Every 5 PWM opportunities

enumerator kPWM_LoadEvery6Oportunity
Every 6 PWM opportunities

enumerator kPWM_LoadEvery7Oportunity
Every 7 PWM opportunities

enumerator kPWM_LoadEvery8Oportunity
Every 8 PWM opportunities

enumerator kPWM_LoadEvery9Oportunity
Every 9 PWM opportunities

enumerator kPWM_LoadEvery10Oportunity
Every 10 PWM opportunities

enumerator kPWM_LoadEvery11Oportunity
Every 11 PWM opportunities

enumerator kPWM_LoadEvery12Oportunity
Every 12 PWM opportunities

enumerator kPWM_LoadEvery13Oportunity
Every 13 PWM opportunities

enumerator kPWM_LoadEvery14Oportunity
Every 14 PWM opportunities

enumerator kPWM_LoadEvery15Oportunity
Every 15 PWM opportunities

enumerator kPWM_LoadEvery16Oportunity
Every 16 PWM opportunities

enum __pwm_fault_input
List of PWM fault selections.
Values:

enumerator kPWM_Fault_0
Fault 0 input pin

enumerator kPWM_Fault_1
Fault 1 input pin

enumerator kPWM_Fault_2
Fault 2 input pin

enumerator kPWM_Fault_3
Fault 3 input pin

enum __pwm_fault_disable
List of PWM fault disable mapping selections.
Values:

enumerator kPWM_FaultDisable_0
Fault 0 disable mapping

enumerator kPWM_FaultDisable_1
Fault 1 disable mapping

enumerator kPWM_FaultDisable_2
Fault 2 disable mapping

enumerator kPWM_FaultDisable_3
Fault 3 disable mapping

enum _pwm_fault_channels
List of PWM fault channels.
Values:

enumerator kPWM_faultchannel_0

enum _pwm_input_capture_edge
PWM capture edge select.
Values:

enumerator kPWM_Disable
Disabled

enumerator kPWM_FallingEdge
Capture on falling edge only

enumerator kPWM_RisingEdge
Capture on rising edge only

enumerator kPWM_RiseAndFallEdge
Capture on rising or falling edge

enum _pwm_force_signal
PWM output options when a FORCE_OUT signal is asserted.
Values:

enumerator kPWM_UsePwm
Generated PWM signal is used by the deadtime logic.

enumerator kPWM_InvertedPwm
Inverted PWM signal is used by the deadtime logic.

enumerator kPWM_SoftwareControl
Software controlled value is used by the deadtime logic.

enumerator kPWM_UseExternal
PWM_EXTA signal is used by the deadtime logic.

enum _pwm_chnl_pair_operation
Options available for the PWM A & B pair operation.
Values:

enumerator kPWM_Independent
PWM A & PWM B operate as 2 independent channels

enumerator kPWM_ComplementaryPwmA
PWM A & PWM B are complementary channels, PWM A generates the signal

enumerator kPWM_ComplementaryPwmB
PWM A & PWM B are complementary channels, PWM B generates the signal

enum _pwm_register_reload
Options available on how to load the buffered-registers with new values.
Values:

enumerator kPWM_ReloadImmediate
Buffered-registers get loaded with new values as soon as LDOK bit is set

enumerator kPWM_ReloadPwmHalfCycle
Registers loaded on a PWM half cycle

enumerator kPWM_ReloadPwmFullCycle
Registers loaded on a PWM full cycle

enumerator kPWM_ReloadPwmHalfAndFullCycle
Registers loaded on a PWM half & full cycle

enum _pwm_fault_recovery_mode
Options available on how to re-enable the PWM output when recovering from a fault.
Values:

enumerator kPWM_NoRecovery
PWM output will stay inactive

enumerator kPWM_RecoverHalfCycle
PWM output re-enabled at the first half cycle

enumerator kPWM_RecoverFullCycle
PWM output re-enabled at the first full cycle

enumerator kPWM_RecoverHalfAndFullCycle
PWM output re-enabled at the first half or full cycle

enum _pwm_interrupt_enable

List of PWM interrupt options.

Values:

enumerator kPWM_CompareVal0InterruptEnable

PWM VAL0 compare interrupt

enumerator kPWM_CompareVal1InterruptEnable

PWM VAL1 compare interrupt

enumerator kPWM_CompareVal2InterruptEnable

PWM VAL2 compare interrupt

enumerator kPWM_CompareVal3InterruptEnable

PWM VAL3 compare interrupt

enumerator kPWM_CompareVal4InterruptEnable

PWM VAL4 compare interrupt

enumerator kPWM_CompareVal5InterruptEnable

PWM VAL5 compare interrupt

enumerator kPWM_CaptureX0InterruptEnable

PWM capture X0 interrupt

enumerator kPWM_CaptureX1InterruptEnable

PWM capture X1 interrupt

enumerator kPWM_CaptureB0InterruptEnable

PWM capture B0 interrupt

enumerator kPWM_CaptureB1InterruptEnable

PWM capture B1 interrupt

enumerator kPWM_CaptureA0InterruptEnable

PWM capture A0 interrupt

enumerator kPWM_CaptureA1InterruptEnable

PWM capture A1 interrupt

enumerator kPWM_ReloadInterruptEnable

PWM reload interrupt

enumerator kPWM_ReloadErrorInterruptEnable

PWM reload error interrupt

enumerator kPWM_Fault0InterruptEnable

PWM fault 0 interrupt

enumerator kPWM_Fault1InterruptEnable
PWM fault 1 interrupt

enumerator kPWM_Fault2InterruptEnable
PWM fault 2 interrupt

enumerator kPWM_Fault3InterruptEnable
PWM fault 3 interrupt

enum __pwm_status_flags
List of PWM status flags.
Values:

enumerator kPWM_CompareVal0Flag
PWM VAL0 compare flag

enumerator kPWM_CompareVal1Flag
PWM VAL1 compare flag

enumerator kPWM_CompareVal2Flag
PWM VAL2 compare flag

enumerator kPWM_CompareVal3Flag
PWM VAL3 compare flag

enumerator kPWM_CompareVal4Flag
PWM VAL4 compare flag

enumerator kPWM_CompareVal5Flag
PWM VAL5 compare flag

enumerator kPWM_CaptureX0Flag
PWM capture X0 flag

enumerator kPWM_CaptureX1Flag
PWM capture X1 flag

enumerator kPWM_CaptureB0Flag
PWM capture B0 flag

enumerator kPWM_CaptureB1Flag
PWM capture B1 flag

enumerator kPWM_CaptureA0Flag
PWM capture A0 flag

enumerator kPWM_CaptureA1Flag
PWM capture A1 flag

enumerator kPWM_ReloadFlag
PWM reload flag

enumerator kPWM_ReloadErrorFlag
PWM reload error flag

enumerator kPWM_RegUpdatedFlag
PWM registers updated flag

enumerator kPWM_Fault0Flag
PWM fault 0 flag

enumerator kPWM_Fault1Flag
PWM fault 1 flag

enumerator kPWM_Fault2Flag
PWM fault 2 flag

enumerator kPWM_Fault3Flag
PWM fault 3 flag

enum _pwm_dma_enable
List of PWM DMA options.
Values:

enumerator kPWM_CaptureX0DMAEnable
PWM capture X0 DMA

enumerator kPWM_CaptureX1DMAEnable
PWM capture X1 DMA

enumerator kPWM_CaptureB0DMAEnable
PWM capture B0 DMA

enumerator kPWM_CaptureB1DMAEnable
PWM capture B1 DMA

enumerator kPWM_CaptureA0DMAEnable
PWM capture A0 DMA

enumerator kPWM_CaptureA1DMAEnable
PWM capture A1 DMA

enum _pwm_dma_source_select
List of PWM capture DMA enable source select.
Values:

enumerator kPWM_DMARequestDisable
Read DMA requests disabled

enumerator kPWM_DMAWatermarksEnable
Exceeding a FIFO watermark sets the DMA read request

enumerator kPWM_DMALocalSync
A local sync (VAL1 matches counter) sets the read DMA request

enumerator kPWM_DMALocalReload
A local reload (STS[RF] being set) sets the read DMA request

enum _pwm_watermark_control
PWM FIFO Watermark AND Control.
Values:

enumerator kPWM_FIFOWatermarksOR
Selected FIFO watermarks are OR'ed together

enumerator kPWM_FIFOWatermarksAND
Selected FIFO watermarks are AND'ed together

enum _pwm_mode
PWM operation mode.
Values:

enumerator kPWM_SignedCenterAligned
Signed center-aligned

enumerator kPWM_CenterAligned
Unsigned cente-aligned

enumerator kPWM_SignedEdgeAligned
Signed edge-aligned

enumerator kPWM_EdgeAligned
Unsigned edge-aligned

enum _pwm_level_select
PWM output pulse mode, high-true or low-true.
Values:

enumerator kPWM_HighTrue
High level represents "on" or "active" state

enumerator kPWM_LowTrue
Low level represents "on" or "active" state

enum _pwm_fault_state

PWM output fault status.

Values:

enumerator kPWM_PwmFaultState0

Output is forced to logic 0 state prior to consideration of output polarity control.

enumerator kPWM_PwmFaultState1

Output is forced to logic 1 state prior to consideration of output polarity control.

enumerator kPWM_PwmFaultState2

Output is tristated.

enumerator kPWM_PwmFaultState3

Output is tristated.

enum _pwm_reload_source_select

PWM reload source select.

Values:

enumerator kPWM_LocalReload

The local reload signal is used to reload registers

enumerator kPWM_MasterReload

The master reload signal (from submodule 0) is used to reload

enum _pwm_fault_clear

PWM fault clearing options.

Values:

enumerator kPWM_Automatic

Automatic fault clearing

enumerator kPWM_ManualNormal

Manual fault clearing with no fault safety mode

enumerator kPWM_ManualSafety

Manual fault clearing with fault safety mode

enum _pwm_module_control

Options for submodule master control operation.

Values:

enumerator kPWM_Control_Module_0

Control submodule 0's start/stop,buffer reload operation

enumerator kPWM_Control_Module_1

Control submodule 1's start/stop,buffer reload operation

enumerator kPWM_Control_Module_2
Control submodule 2's start/stop,buffer reload operation

enumerator kPWM_Control_Module_3
Control submodule 3's start/stop,buffer reload operation

typedef enum *_pwm_submodule* pwm_submodule_t
List of PWM submodules.

typedef enum *_pwm_channels* pwm_channels_t
List of PWM channels in each module.

typedef enum *_pwm_value_register* pwm_value_register_t
List of PWM value registers.

typedef enum *_pwm_clock_source* pwm_clock_source_t
PWM clock source selection.

typedef enum *_pwm_clock_prescale* pwm_clock_prescale_t
PWM prescaler factor selection for clock source.

typedef enum *_pwm_force_output_trigger* pwm_force_output_trigger_t
Options that can trigger a PWM FORCE_OUT.

typedef enum *_pwm_output_state* pwm_output_state_t
PWM channel output status.

typedef enum *_pwm_init_source* pwm_init_source_t
PWM counter initialization options.

typedef enum *_pwm_load_frequency* pwm_load_frequency_t
PWM load frequency selection.

typedef enum *_pwm_fault_input* pwm_fault_input_t
List of PWM fault selections.

typedef enum *_pwm_fault_disable* pwm_fault_disable_t
List of PWM fault disable mapping selections.

typedef enum *_pwm_fault_channels* pwm_fault_channels_t
List of PWM fault channels.

typedef enum *_pwm_input_capture_edge* pwm_input_capture_edge_t
PWM capture edge select.

typedef enum *_pwm_force_signal* pwm_force_signal_t
PWM output options when a FORCE_OUT signal is asserted.

typedef enum *_pwm_chnl_pair_operation* pwm_chnl_pair_operation_t
Options available for the PWM A & B pair operation.

typedef enum *_pwm_register_reload* pwm_register_reload_t
Options available on how to load the buffered-registers with new values.

typedef enum *_pwm_fault_recovery_mode* pwm_fault_recovery_mode_t
Options available on how to re-enable the PWM output when recovering from a fault.

typedef enum *_pwm_interrupt_enable* pwm_interrupt_enable_t
List of PWM interrupt options.

typedef enum *_pwm_status_flags* pwm_status_flags_t
List of PWM status flags.

typedef enum *_pwm_dma_enable* pwm_dma_enable_t
List of PWM DMA options.

typedef enum *_pwm_dma_source_select* pwm_dma_source_select_t
List of PWM capture DMA enable source select.

typedef enum *_pwm_watermark_control* pwm_watermark_control_t
PWM FIFO Watermark AND Control.

typedef enum *_pwm_mode* pwm_mode_t
PWM operation mode.

typedef enum *_pwm_level_select* pwm_level_select_t
PWM output pulse mode, high-true or low-true.

typedef enum *_pwm_fault_state* pwm_fault_state_t
PWM output fault status.

typedef enum *_pwm_reload_source_select* pwm_reload_source_select_t
PWM reload source select.

typedef enum *_pwm_fault_clear* pwm_fault_clear_t
PWM fault clearing options.

typedef enum *_pwm_module_control* pwm_module_control_t
Options for submodule master control operation.

typedef struct *_pwm_signal_param* pwm_signal_param_t
Structure for the user to define the PWM signal characteristics.

typedef struct *_pwm_config* pwm_config_t
PWM config structure.

This structure holds the configuration settings for the PWM peripheral. To initialize this structure to reasonable defaults, call the *PWM_GetDefaultConfig()* function and pass a pointer to your config structure instance.

The config struct can be made const so it resides in flash

typedef struct *_pwm_fault_input_filter_param* pwm_fault_input_filter_param_t

Structure for the user to configure the fault input filter.

typedef struct *_pwm_fault_param* pwm_fault_param_t

Structure is used to hold the parameters to configure a PWM fault.

typedef struct *_pwm_input_capture_param* pwm_input_capture_param_t

Structure is used to hold parameters to configure the capture capability of a signal pin.

void PWM_SetupInputCapture(PWM_Type *base, *pwm_submodule_t* subModule, *pwm_channels_t* pwmChannel, const *pwm_input_capture_param_t* *inputCaptureParams)

Sets up the PWM input capture.

Each PWM submodule has 3 pins that can be configured for use as input capture pins. This function sets up the capture parameters for each pin and enables the pin for input capture operation.

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- pwmChannel – Channel in the submodule to setup
- inputCaptureParams – Parameters passed in to set up the input pin

status_t PWM_GetInputCaptureValue(PWM_Type *base, *pwm_submodule_t* subModule, *pwm_channels_t* pwmChannel, uint8_t captureIndex, uint16_t *captureValue)

Read the capture value.

This function reads the capture value stored in channel's capture value register. It should be called when a valid edge is detected on the input capture pin(related capture flag is set). The capture circuit has two input capture registers per channel for first edge and second edge capture.

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- pwmChannel – PWM channel to read from (PWM A, PWM B, or PWM X)
- captureIndex – Capture register to read (0 for first edge capture, 1 for second edge capture)

Returns

Returns kStatus_InvalidArgument if pwmChannel does not support capture feature; kStatus_Success otherwise

void PWM_SetupFaultInputFilter(PWM_Type *base, const *pwm_fault_input_filter_param_t* *faultInputFilterParams)

Sets up the PWM fault channel 0 input filter.

Parameters

- base – PWM peripheral base address
- faultInputFilterParams – Parameters passed in to set up the fault input filter.

void PWM_SetupFaultInputFilterExt(PWM_Type *base, *pwm_fault_channels_t* faultChannel, const *pwm_fault_input_filter_param_t* *faultInputFilterParams)

Sets up the PWM fault input filter.

Parameters

- base – PWM peripheral base address
- faultChannel – PWM fault channel to configure.
- faultInputFilterParams – Parameters passed in to set up the fault input filter.

void PWM_SetupFaults(PWM_Type *base, *pwm_fault_input_t* faultNum, const *pwm_fault_param_t* *faultParams)

Sets up the PWM fault channel 0 protection.

Parameters

- base – PWM peripheral base address
- faultNum – PWM fault to configure.
- faultParams – Pointer to the PWM fault config structure

void PWM_SetupFaultsExt(PWM_Type *base, *pwm_fault_channels_t* faultChannel, *pwm_fault_input_t* faultNum, const *pwm_fault_param_t* *faultParams)

Sets up the PWM fault protection.

Parameters

- base – PWM peripheral base address
- faultChannel – PWM fault channel to configure.
- faultNum – PWM fault to configure.
- faultParams – Pointer to the PWM fault config structure

void PWM_FaultDefaultConfig(*pwm_fault_param_t* *config)

Fill in the PWM fault config struct with the default settings.

The default values are:

```
config->faultClearingMode = kPWM_Automatic;
config->faultLevel = false;
config->enableCombinationalPath = true;
config->recoverMode = kPWM_NoRecovery;
```

Parameters

- config – Pointer to user's PWM fault config structure.

void PWM_SetupForceSignal(PWM_Type *base, *pwm_submodule_t* subModule, *pwm_channels_t* pwmChannel, *pwm_force_signal_t* mode)

Selects the signal to output on a PWM pin when a FORCE_OUT signal is asserted.

The user specifies which channel to configure by supplying the submodule number and whether to modify PWM A or PWM B within that submodule.

Parameters

- base – PWM peripheral base address

- subModule – PWM submodule to configure
- pwmChannel – Channel to configure
- mode – Signal to output when a FORCE_OUT is triggered

static inline void PWM_EnableLocalForce(PWM_Type *base, *pwm_submodule_t* subModule)

Enables local software force initialization on a PWM submodule.

This function performs a software-controlled initialization, causes a FORCE_OUT event which latches all double-buffered fields (DTSRCSEL, MCTRL[IPOL], SWCOUT) into their active registers and, if MCTRL[LDOK] is set, also triggers a register reload.

Call this function after updating the desired buffered registers (e.g. after *PWM_SetupForceSignal()*) to apply the new values synchronously.

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to enable local force on

void PWM_UpdateCurrentPolarity(PWM_Type *base, *pwm_submodule_t* subModule, *pwm_chnl_pair_operation_t* polarity)

Updates the current polarity (MCTRL[IPOL]) for a PWM submodule.

MCTRL[IPOL] is a double-buffered field. This function only writes the shadow register; the value does NOT take effect until a FORCE_OUT event occurs. Call *PWM_EnableLocalForce()* after this function to apply all pending changes atomically in one FORCE_OUT event.

Only meaningful when the submodule operates in complementary mode (CTRL2[INDEP] = 0).

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- polarity – kPWM_ComplementaryPwmA: PWM23 (VAL2/VAL3) drives the complementary pair kPWM_ComplementaryPwmB: PWM45 (VAL4/VAL5) drives the complementary pair

static inline void PWM_SetVALxValue(PWM_Type *base, *pwm_submodule_t* subModule, *pwm_value_register_t* valueRegister, uint16_t value)

Set the PWM VALx registers.

This function allows the user to write value into VAL registers directly. And it will destroying the PWM clock period set by the *PWM_SetupPwm()*/PWM_SetupPwmPhaseShift() functions. Due to VALx registers are bufferd, the new value will not active uless call *PWM_SetPwmLdok()* and the reload point is reached.

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- valueRegister – VALx register that will be writen new value
- value – Value that will been write into VALx register

static inline uint16_t PWM_GetVALxValue(PWM_Type *base, *pwm_submodule_t* subModule, *pwm_value_register_t* valueRegister)

Get the PWM VALx registers.

Parameters

- base – PWM peripheral base address

- subModule – PWM submodule to configure
- valueRegister – VALx register that will be read value

Returns
The VALx register value

static inline void PWM_OutputTriggerEnable(PWM_Type *base, *pwm_submodule_t* subModule, *pwm_value_register_t* valueRegister, bool activate)

Enables or disables the PWM output trigger.

This function allows the user to enable or disable the PWM trigger. The PWM has 2 triggers. Trigger 0 is activated when the counter matches VAL 0, VAL 2, or VAL 4 register. Trigger 1 is activated when the counter matches VAL 1, VAL 3, or VAL 5 register.

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- valueRegister – Value register that will activate the trigger
- activate – true: Enable the trigger; false: Disable the trigger

static inline void PWM_ActivateOutputTrigger(PWM_Type *base, *pwm_submodule_t* subModule, uint16_t valueRegisterMask)

Enables the PWM output trigger.

This function allows the user to enable one or more (VAL0-5) PWM trigger.

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- valueRegisterMask – Value register mask that will activate one or more (VAL0-5) trigger enumeration *_pwm_value_register_mask*

static inline void PWM_DeactivateOutputTrigger(PWM_Type *base, *pwm_submodule_t* subModule, uint16_t valueRegisterMask)

Disables the PWM output trigger.

This function allows the user to disables one or more (VAL0-5) PWM trigger.

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- valueRegisterMask – Value register mask that will Deactivate one or more (VAL0-5) trigger enumeration *_pwm_value_register_mask*

static inline void PWM_SetupSwCtrlOut(PWM_Type *base, *pwm_submodule_t* subModule, *pwm_channels_t* pwmChannel, bool value)

Sets the software control output for a pin to high or low.

The user specifies which channel to modify by supplying the submodule number and whether to modify PWM A or PWM B within that submodule.

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- pwmChannel – Channel to configure
- value – true: Supply a logic 1, false: Supply a logic 0.

static inline void PWM_SetPwmLdok(PWM_Type *base, uint8_t subModulesToUpdate, bool value)

Sets or clears the PWM LDOK bit on a single or multiple submodules.

Set LDOK bit to load buffered values into CTRL[PRSC] and the INIT, FRACVAL and VAL registers. The values are loaded immediately if kPWM_ReloadImmediate option was choosen during config. Else the values are loaded at the next PWM reload point. This function can issue the load command to multiple submodules at the same time.

Parameters

- base – PWM peripheral base address
- subModulesToUpdate – PWM submodules to update with buffered values. This is a logical OR of members of the enumeration *pwm_module_control_t*
- value – true: Set LDOK bit for the submodule list; false: Clear LDOK bit

static inline void PWM_SetPwmFaultState(PWM_Type *base, *pwm_submodule_t* subModule, *pwm_channels_t* pwmChannel, *pwm_fault_state_t* faultState)

Set PWM output fault status.

These bits determine the fault state for the PWM_A output in fault conditions and STOP mode. It may also define the output state in WAIT and DEBUG modes depending on the settings of CTRL2[WAITEN] and CTRL2[DBGEN]. This function can update PWM output fault status.

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- pwmChannel – Channel to configure
- faultState – PWM output fault status

static inline void PWM_SetupFaultDisableMap(PWM_Type *base, *pwm_submodule_t* subModule, *pwm_channels_t* pwmChannel, *pwm_fault_channels_t* pwm_fault_channels, uint16_t value)

Set PWM fault disable mapping.

Each of the four bits of this read/write field is one-to-one associated with the four FAULTx inputs of fault channel 0/1. The PWM output will be turned off if there is a logic 1 on an FAULTx input and a 1 in the corresponding bit of this field. A reset sets all bits in this field.

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- pwmChannel – PWM channel to configure
- pwm_fault_channels – PWM fault channel to configure
- value – Fault disable mapping mask value enumeration *pwm_fault_disable_t*

static inline void PWM_OutputEnable(PWM_Type *base, *pwm_channels_t* pwmChannel, *pwm_submodule_t* subModule)

Set PWM output enable.

This feature allows the user to enable the PWM Output. Recommend to invoke this API after PWM and fault configuration. But invoke this API before configure MCTRL register is okay, such as set LDOK or start timer.

Parameters

- base – PWM peripheral base address
- pwmChannel – PWM channel to configure
- subModule – PWM submodule to configure

static inline void PWM_OutputDisable(PWM_Type *base, *pwm_channels_t* pwmChannel, *pwm_submodule_t* subModule)

Set PWM output disable.

This feature allows the user to disable the PWM output. Recommend to invoke this API after PWM and fault configuration. But invoke this API before configure MCTRL register is okay, such as set LDOK or start timer.

Parameters

- base – PWM peripheral base address
- pwmChannel – PWM channel to configure
- subModule – PWM submodule to configure

uint8_t PWM_GetPwmChannelState(PWM_Type *base, *pwm_submodule_t* subModule, *pwm_channels_t* pwmChannel)

Get the dutycycle value.

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- pwmChannel – PWM channel to configure

Returns

Current channel dutycycle value.

status_t PWM_SetOutputToIdle(PWM_Type *base, *pwm_channels_t* pwmChannel, *pwm_submodule_t* subModule, bool idleStatus)

Set PWM output in idle status (high or low).

Note: This API should call after *PWM_SetupPwm()* APIs, and PWMX submodule is not supported.

Parameters

- base – PWM peripheral base address
- pwmChannel – PWM channel to configure
- subModule – PWM submodule to configure
- idleStatus – True: PWM output is high in idle status; false: PWM output is low in idle status.

Returns

kStatus_Fail if there was error setting up the signal; kStatus_Success if set output idle success

void PWM_SetClockMode(PWM_Type *base, *pwm_submodule_t* subModule, *pwm_clock_prescale_t* prescaler)

Set the pwm submodule prescaler.

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- prescaler – Set prescaler value

void PWM_SetPwmForceOutputToZero(PWM_Type *base, *pwm_submodule_t* subModule, *pwm_channels_t* pwmChannel, bool forcetozero)

This function enables-disables the forcing of the output of a given eFlexPwm channel to logic 0.

Parameters

- base – PWM peripheral base address
- pwmChannel – PWM channel to configure
- subModule – PWM submodule to configure
- forcetozero – True: Enable the pwm force output to zero; False: Disable the pwm output resumes normal function.

void PWM_SetChannelOutput(PWM_Type *base, *pwm_submodule_t* subModule, *pwm_channels_t* pwmChannel, *pwm_output_state_t* outputstate)

This function set the output state of the PWM pin as requested for the current cycle.

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- pwmChannel – PWM channel to configure
- outputstate – Set pwm output state, see *pwm_output_state_t*.

status_t PWM_SetPhaseDelay(PWM_Type *base, *pwm_channels_t* pwmChannel, *pwm_submodule_t* subModule, uint16_t delayCycles)

This function set the phase delay from the master sync signal of submodule 0.

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- pwmChannel – PWM channel to configure
- delayCycles – Number of cycles delayed from submodule 0.

Returns

kStatus_Fail if the number of delay cycles is set larger than the period defined in submodule 0; kStatus_Success if set phase delay success

static inline void PWM_SetFilterSampleCount(PWM_Type *base, *pwm_channels_t* pwmChannel, *pwm_submodule_t* subModule, uint8_t filterSampleCount)

This function set the number of consecutive samples that must agree prior to the input filter.

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- pwmChannel – PWM channel to configure
- filterSampleCount – Number of consecutive samples.

static inline void PWM_SetFilterSamplePeriod(PWM_Type *base, *pwm_channels_t* pwmChannel, *pwm_submodule_t* subModule, uint8_t filterSamplePeriod)

This function set the sampling period of the fault pin input filter.

Parameters

- base – PWM peripheral base address
- subModule – PWM submodule to configure
- pwmChannel – PWM channel to configure
- filterSamplePeriod – Sampling period of input filter.

PWM_SUBMODULE_SWCONTROL_WIDTH

Number of bits per submodule for software output control

PWM_SUBMODULE_CHANNEL

Submodule channels include PWMA, PWMB, PWMX.

struct __pwm_signal_param

#include <fsl_pwm.h> Structure for the user to define the PWM signal characteristics.

Public Members

pwm_channels_t pwmChannel

PWM channel being configured; PWM A or PWM B

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)

pwm_level_select_t level

PWM output active level select

uint16_t deadtimeValue

The deadtime value; only used if channel pair is operating in complementary mode

pwm_fault_state_t faultState

PWM output fault status

bool pwmchannelenable

Enable PWM output

struct __pwm_config

#include <fsl_pwm.h> PWM config structure.

This structure holds the configuration settings for the PWM peripheral. To initialize this structure to reasonable defaults, call the *PWM_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 enableDebugMode
: true: PWM continues to run in debug mode; false: PWM is paused in debug mode

pwm_init_source_t initializationControl
: Option to initialize the counter

pwm_clock_source_t clockSource
: Clock source for the counter

pwm_clock_prescale_t prescale
: Pre-scaler to divide down the clock

pwm_chnl_pair_operation_t pairOperation
: Channel pair in indepedent or complementary mode

pwm_register_reload_t reloadLogic
: PWM Reload logic setup

pwm_reload_source_select_t reloadSelect
: Reload source select

pwm_load_frequency_t reloadFrequency
: Specifies when to reload, used when user's choice is not immediate reload

pwm_force_output_trigger_t forceTrigger
: Specify which signal will trigger a FORCE_OUT

struct __pwm_fault_input_filter_param
: *#include <fsl_pwm.h>* Structure for the user to configure the fault input filter.

Public Members

uint8_t faultFilterCount
: Fault filter count

uint8_t faultFilterPeriod
: Fault filter period;value of 0 will bypass the filter

bool faultGlitchStretch
: Fault Glitch Stretch Enable: A logic 1 means that input fault signals will be stretched to at least 2 IPBus clock cycles

struct __pwm_fault_param
: *#include <fsl_pwm.h>* Structure is used to hold the parameters to configure a PWM fault.

Public Members

pwm_fault_clear_t faultClearingMode

Fault clearing mode to use

bool faultLevel

true: Logic 1 indicates fault; false: Logic 0 indicates fault

bool enableCombinationalPath

true: Combinational Path from fault input is enabled; false: No combination path is available

pwm_fault_recovery_mode_t recoverMode

Specify when to re-enable the PWM output

struct __pwm__input__capture__param

#include <fsl_pwm.h> Structure is used to hold parameters to configure the capture capability of a signal pin.

Public Members

bool captureInputSel

true: Use the edge counter signal as source false: Use the raw input signal from the pin as source

uint8_t edgeCompareValue

Compare value, used only if edge counter is used as source

pwm_input_capture_edge_t edge0

Specify which edge causes a capture for input circuitry 0

pwm_input_capture_edge_t edge1

Specify which edge causes a capture for input circuitry 1

bool enableOneShotCapture

true: Use one-shot capture mode; false: Use free-running capture mode

uint8_t fifoWatermark

Watermark level for capture FIFO. The capture flags in the status register will set if the word count in the FIFO is greater than this watermark level

2.54 QTMR: Quad Timer Driver

void QTMR_Init(TMR_Type *base, *qtmr_channel_selection_t* channel, const *qtmr_config_t* *config)

Ungates the Quad Timer clock and configures the peripheral for basic operation.

Note: This API should be called at the beginning of the application using the Quad Timer driver.

Parameters

- base – Quad Timer peripheral base address
- channel – Quad Timer channel number
- config – Pointer to user's Quad Timer config structure

void QTMR_Deinit(TMR_Type *base, *qtmr_channel_selection_t* channel)

Stops the counter and gates the Quad Timer clock.

Parameters

- base – Quad Timer peripheral base address
- channel – Quad Timer channel number

void QTMR_GetDefaultConfig(*qtmr_config_t* *config)

Fill in the Quad Timer config struct with the default settings.

The default values are:

```
config->debugMode = kQTMR_RunNormalInDebug;
config->enableExternalForce = false;
config->enableMasterMode = false;
config->faultFilterCount = 0;
config->faultFilterPeriod = 0;
config->primarySource = kQTMR_ClockDivide_2;
config->secondarySource = kQTMR_Counter0InputPin;
```

Parameters

- config – Pointer to user's Quad Timer config structure.

void QTMR_EnableInterrupts(TMR_Type *base, *qtmr_channel_selection_t* channel, uint32_t mask)

Enables the selected Quad Timer interrupts.

Parameters

- base – Quad Timer peripheral base address
- channel – Quad Timer channel number
- mask – The interrupts to enable. This is a logical OR of members of the enumeration *qtmr_interrupt_enable_t*

void QTMR_DisableInterrupts(TMR_Type *base, *qtmr_channel_selection_t* channel, uint32_t mask)

Disables the selected Quad Timer interrupts.

Parameters

- base – Quad Timer peripheral base addres
- channel – Quad Timer channel number
- mask – The interrupts to enable. This is a logical OR of members of the enumeration *qtmr_interrupt_enable_t*

uint32_t QTMR_GetEnabledInterrupts(TMR_Type *base, *qtmr_channel_selection_t* channel)

Gets the enabled Quad Timer interrupts.

Parameters

- base – Quad Timer peripheral base address
- channel – Quad Timer channel number

Returns

The enabled interrupts. This is the logical OR of members of the enumeration *qtmr_interrupt_enable_t*

uint32_t QTMR_GetStatus(TMR_Type *base, *qtmr_channel_selection_t* channel)

Gets the Quad Timer status flags.

Parameters

- base – Quad Timer peripheral base address
- channel – Quad Timer channel number

Returns

The status flags. This is the logical OR of members of the enumeration *qtmr_status_flags_t*

void QTMR_ClearStatusFlags(TMR_Type *base, *qtmr_channel_selection_t* channel, uint32_t mask)

Clears the Quad Timer status flags.

Parameters

- base – Quad Timer peripheral base address
- channel – Quad Timer channel number
- mask – The status flags to clear. This is a logical OR of members of the enumeration *qtmr_status_flags_t*

void QTMR_SetTimerPeriod(TMR_Type *base, *qtmr_channel_selection_t* channel, uint16_t ticks)

Sets the timer period in ticks.

Timers counts from initial value till it equals the count value set here. The counter will then reinitialize to the value specified in the Load register.

Note:

a. This function will write the time period in ticks to COMP1 or COMP2 register depending on the count direction

b. User can call the utility macros provided in fsl_common.h to convert to ticks

c. This function supports cases, providing only primary source clock without secondary source clock.

Parameters

- base – Quad Timer peripheral base address
- channel – Quad Timer channel number
- ticks – Timer period in units of ticks

void QTMR_SetCompareValue(TMR_Type *base, *qtmr_channel_selection_t* channel, uint16_t ticks)

Set compare value.

This function sets the value used for comparison with the counter value.

Parameters

- base – Quad Timer peripheral base address
- channel – Quad Timer channel number
- ticks – Timer period in units of ticks.

static inline void QTMR_SetLoadValue(TMR_Type *base, *qtmr_channel_selection_t* channel, uint16_t value)

Set load value.

This function sets the value used to initialize the counter after a counter comparison.

Parameters

- base – Quad Timer peripheral base address
- channel – Quad Timer channel number
- value – Load register initialization value.

static inline uint16_t QTMR_GetCurrentTimerCount(TMR_Type *base, *qtmr_channel_selection_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 usec or msec

---

Parameters

- base – Quad Timer peripheral base address
- channel – Quad Timer channel number

Returns

Current counter value in ticks

static inline void QTMR_StartTimer(TMR_Type *base, *qtmr_channel_selection_t* channel, *qtmr_counting_mode_t* clockSource)

Starts the Quad Timer counter.

Parameters

- base – Quad Timer peripheral base address
- channel – Quad Timer channel number
- clockSource – Quad Timer clock source

static inline void QTMR_StopTimer(TMR_Type *base, *qtmr_channel_selection_t* channel)

Stops the Quad Timer counter.

Parameters

- base – Quad Timer peripheral base address
- channel – Quad Timer channel number

void QTMR_EnableDma(TMR_Type *base, *qtmr_channel_selection_t* channel, uint32_t mask)

Enable the Quad Timer DMA.

Parameters

- base – Quad Timer peripheral base address
- channel – Quad Timer channel number
- mask – The DMA to enable. This is a logical OR of members of the enumeration *qtmr_dma_enable_t*

void QTMR_DisableDma(TMR_Type *base, *qtmr_channel_selection_t* channel, uint32_t mask)

Disable the Quad Timer DMA.

Parameters

- base – Quad Timer peripheral base address
- channel – Quad Timer channel number
- mask – The DMA to enable. This is a logical OR of members of the enumeration *qtmr_dma_enable_t*

FSL_QTMR_DRIVER_VERSION

Version

enum _qtmr_primary_count_source

Quad Timer primary clock source selection.

Values:

enumerator kQTMR_ClockCounter0InputPin

Use counter 0 input pin

enumerator kQTMR_ClockCounter1InputPin

Use counter 1 input pin

enumerator kQTMR_ClockCounter2InputPin

Use counter 2 input pin

enumerator kQTMR_ClockCounter3InputPin

Use counter 3 input pin

enumerator kQTMR_ClockCounter0Output

Use counter 0 output

enumerator kQTMR_ClockCounter1Output

Use counter 1 output

enumerator kQTMR_ClockCounter2Output

Use counter 2 output

enumerator kQTMR_ClockCounter3Output

Use counter 3 output

enumerator kQTMR_ClockDivide_1
IP bus clock divide by 1 prescaler

enumerator kQTMR_ClockDivide_2
IP bus clock divide by 2 prescaler

enumerator kQTMR_ClockDivide_4
IP bus clock divide by 4 prescaler

enumerator kQTMR_ClockDivide_8
IP bus clock divide by 8 prescaler

enumerator kQTMR_ClockDivide_16
IP bus clock divide by 16 prescaler

enumerator kQTMR_ClockDivide_32
IP bus clock divide by 32 prescaler

enumerator kQTMR_ClockDivide_64
IP bus clock divide by 64 prescaler

enumerator kQTMR_ClockDivide_128
IP bus clock divide by 128 prescaler

enum _qtmr_input_source
Quad Timer input sources selection.
Values:

enumerator kQTMR_Counter0InputPin
Use counter 0 input pin

enumerator kQTMR_Counter1InputPin
Use counter 1 input pin

enumerator kQTMR_Counter2InputPin
Use counter 2 input pin

enumerator kQTMR_Counter3InputPin
Use counter 3 input pin

enum _qtmr_counting_mode
Quad Timer counting mode selection.
Values:

enumerator kQTMR_NoOperation
No operation

enumerator kQTMR_PriSrcRiseEdge
Count rising edges of primary source

enumerator kQTMR_PriSrcRiseAndFallEdge
Count rising and falling edges of primary source

enumerator kQTMR_PriSrcRiseEdgeSecInpHigh
Count rise edges of pri SRC while sec inp high active

enumerator kQTMR_QuadCountMode
Quadrature count mode, uses pri and sec sources

enumerator kQTMR_PriSrcRiseEdgeSecDir
Count rising edges of pri SRC; sec SRC specifies dir

enumerator kQTMR_SecSrcTrigPriCnt
Edge of sec SRC trigger primary count until compare

enumerator kQTMR_CascadeCount
Cascaded count mode (up/down)

enum _qtmr_pwm_out_state
Quad Timer PWM output state.
Values:

enumerator kQTMR_PwmLow
The output state of PWM channel is low

enumerator kQTMR_PwmHigh
The output state of PWM channel is low

enum _qtmr_output_mode
Quad Timer output mode selection.
Values:

enumerator kQTMR_AssertWhenCountActive
Assert OFLAG while counter is active

enumerator kQTMR_ClearOnCompare
Clear OFLAG on successful compare

enumerator kQTMR_SetOnCompare
Set OFLAG on successful compare

enumerator kQTMR_ToggleOnCompare
Toggle OFLAG on successful compare

enumerator kQTMR_ToggleOnAltCompareReg
Toggle OFLAG using alternating compare registers

enumerator kQTMR_SetOnCompareClearOnSecSrcInp
Set OFLAG on compare, clear on sec SRC input edge

enumerator kQTMR_SetOnCompareClearOnCountRoll
Set OFLAG on compare, clear on counter rollover

enumerator kQTMR_EnableGateClock
Enable gated clock output while count is active

enum _qtmr_input_capture_edge
Quad Timer input capture edge mode, rising edge, or falling edge.
Values:

enumerator kQTMR_NoCapture
Capture is disabled

enumerator kQTMR_RisingEdge
Capture on rising edge (IPS=0) or falling edge (IPS=1)

enumerator kQTMR_FallingEdge
Capture on falling edge (IPS=0) or rising edge (IPS=1)

enumerator kQTMR_RisingAndFallingEdge
Capture on both edges

enum _qtmr_preload_control
Quad Timer input capture edge mode, rising edge, or falling edge.
Values:

enumerator kQTMR_NoPreload
Never preload

enumerator kQTMR_LoadOnComp1
Load upon successful compare with value in COMP1

enumerator kQTMR_LoadOnComp2
Load upon successful compare with value in COMP2

enum _qtmr_debug_action
List of Quad Timer run options when in Debug mode.
Values:

enumerator kQTMR_RunNormalInDebug
Continue with normal operation

enumerator kQTMR_HaltCounter
Halt counter

enumerator kQTMR_ForceOutToZero
Force output to logic 0

enumerator kQTMR_HaltCountForceOutZero
Halt counter and force output to logic 0

enum _qtmr_interrupt_enable
List of Quad Timer interrupts.
Values:

enumerator kQTMR_CompareInterruptEnable
Compare interrupt.

enumerator kQTMR_Compare1InterruptEnable
Compare 1 interrupt.

enumerator kQTMR_Compare2InterruptEnable
Compare 2 interrupt.

enumerator kQTMR_OverflowInterruptEnable
Timer overflow interrupt.

enumerator kQTMR_EdgeInterruptEnable
Input edge interrupt.

enum _qtmr_status_flags
List of Quad Timer flags.
Values:

enumerator kQTMR_CompareFlag
Compare flag

enumerator kQTMR_Compare1Flag
Compare 1 flag

enumerator kQTMR_Compare2Flag
Compare 2 flag

enumerator kQTMR_OverflowFlag
Timer overflow flag

enumerator kQTMR_EdgeFlag
Input edge flag

enum _qtmr_channel_selection

List of channel selection.

Values:

enumerator kQTMR_Channel_0

TMR Channel 0

enumerator kQTMR_Channel_1

TMR Channel 1

enumerator kQTMR_Channel_2

TMR Channel 2

enumerator kQTMR_Channel_3

TMR Channel 3

enum _qtmr_dma_enable

List of Quad Timer DMA enable.

Values:

enumerator kQTMR_InputEdgeFlagDmaEnable

Input Edge Flag DMA Enable.

enumerator kQTMR_ComparatorPreload1DmaEnable

Comparator Preload Register 1 DMA Enable.

enumerator kQTMR_ComparatorPreload2DmaEnable

Comparator Preload Register 2 DMA Enable.

typedef uint16_t qtmrRegType

typedef enum *_qtmr_primary_count_source* qtmr_primary_count_source_t

Quad Timer primary clock source selection.

typedef enum *_qtmr_input_source* qtmr_input_source_t

Quad Timer input sources selection.

typedef enum *_qtmr_counting_mode* qtmr_counting_mode_t

Quad Timer counting mode selection.

typedef enum *_qtmr_pwm_out_state* qtmr_pwm_out_state_t

Quad Timer PWM output state.

typedef enum *_qtmr_output_mode* qtmr_output_mode_t

Quad Timer output mode selection.

typedef enum *_qtmr_input_capture_edge* qtmr_input_capture_edge_t

Quad Timer input capture edge mode, rising edge, or falling edge.

typedef enum *_qtmr_preload_control* qtmr_preload_control_t

Quad Timer input capture edge mode, rising edge, or falling edge.

typedef enum *_qtmr_debug_action* qtmr_debug_action_t

List of Quad Timer run options when in Debug mode.

typedef enum *_qtmr_interrupt_enable* qtmr_interrupt_enable_t

List of Quad Timer interrupts.

typedef enum *_qtmr_status_flags* qtmr_status_flags_t

List of Quad Timer flags.

typedef enum *_qtmr_channel_selection* qtmr_channel_selection_t

List of channel selection.

typedef enum *_qtmr_dma_enable* qtmr_dma_enable_t

List of Quad Timer DMA enable.

typedef struct *_qtmr_config* qtmr_config_t

Quad Timer config structure.

This structure holds the configuration settings for the Quad Timer peripheral. To initialize this structure to reasonable defaults, call the *QTMR_GetDefaultConfig()* function and pass a pointer to your config structure instance.

The config struct can be made const so it resides in flash

status_t QTMR_SetupPwm(TMR_Type *base, *qtmr_channel_selection_t* channel, uint32_t pwmFreqHz, uint8_t dutyCyclePercent, bool outputPolarity, uint32_t srcClock_Hz)

Sets up Quad timer module for PWM signal output.

The function initializes the timer module according to the parameters passed in by the user. The function also sets up the value compare registers to match the PWM signal requirements.

Parameters

- base – Quad Timer peripheral base address
- channel – Quad Timer channel number
- pwmFreqHz – PWM signal frequency in Hz
- dutyCyclePercent – PWM pulse width, value should be between 0 to 100 0=inactive signal(0% duty cycle)... 100=active signal (100% duty cycle)
- outputPolarity – true: invert polarity of the output signal, false: no inversion
- srcClock_Hz – Main counter clock in Hz.

Returns

Returns an error if there was error setting up the signal.

void QTMR_SetupInputCapture(TMR_Type *base, *qtmr_channel_selection_t* channel, *qtmr_input_source_t* capturePin, bool inputPolarity, bool reloadOnCapture, *qtmr_input_capture_edge_t* captureMode)

Allows the user to count the source clock cycles until a capture event arrives.

The count is stored in the capture register.

Parameters

- base – Quad Timer peripheral base address
- channel – Quad Timer channel number
- capturePin – Pin through which we receive the input signal to trigger the capture
- inputPolarity – true: invert polarity of the input signal, false: no inversion
- reloadOnCapture – true: reload the counter when an input capture occurs, false: no reload
- captureMode – Specifies which edge of the input signal triggers a capture

void QTMR_SetPwmOutputToIdle(TMR_Type *base, *qtmr_channel_selection_t* channel, bool idleStatus)

Set PWM output in idle status (high or low).

Note: When the PWM is set again, the counting needs to be restarted.

Parameters

- base – Quad Timer peripheral base address
- channel – Quad Timer channel number
- idleStatus – True: PWM output is high in idle status; false: PWM output is low in idle status.

static inline *qtmr_pwm_out_state_t* QTMR_GetPwmOutputStatus(TMR_Type *base, *qtmr_channel_selection_t* channel)

Get the channel output status.

Parameters

- base – Quad Timer peripheral base address
- channel – Quad Timer channel number

Returns

Current channel output status.

uint8_t QTMR_GetPwmChannelStatus(TMR_Type *base, *qtmr_channel_selection_t* channel)

Get the PWM channel dutycycle value.

Parameters

- base – Quad Timer peripheral base address
- channel – Quad Timer channel number

Returns

Current channel dutycycle value.

void QTMR_SetPwmClockMode(TMR_Type *base, *qtmr_channel_selection_t* channel, *qtmr_primary_count_source_t* prescaler)

This function set the value of the prescaler on QTimer channels.

Parameters

- base – Quad Timer peripheral base address
- channel – Quad Timer channel number

- prescaler – Set prescaler value

TMR_CSCTRL_OFLAG_MASK

TMR_CSCTRL_OFLAG_SHIFT

struct _qtmr_config

#include <fsl_qtmr.h> Quad Timer config structure.

This structure holds the configuration settings for the Quad Timer peripheral. To initialize this structure to reasonable defaults, call the *QTMR_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

qtmr_primary_count_source_t primarySource

Specify the primary count source

qtmr_input_source_t secondarySource

Specify the secondary count source

bool enableMasterMode

true: Broadcast compare function output to other counters; false no broadcast

bool enableExternalForce

true: Compare from another counter force state of OFLAG signal false: OFLAG controlled by local counter

uint8_t faultFilterCount

Fault filter count

uint8_t faultFilterPeriod

Fault filter period;value of 0 will bypass the filter

qtmr_debug_action_t debugMode

Operation in Debug mode

2.55 RTWDOG: 32-bit Watchdog Timer

void RTWDOG_GetDefaultConfig(*rtwdog_config_t* *config)

Initializes the RTWDOG configuration structure.

This function initializes the RTWDOG configuration structure to default values. The default values are:

```
rtwdogConfig->enableRtwdog = true;
rtwdogConfig->clockSource = kRTWDOG_ClockSource1;
rtwdogConfig->prescaler = kRTWDOG_ClockPrescalerDivide1;
rtwdogConfig->workMode.enableWait = true;
rtwdogConfig->workMode.enableStop = false;
rtwdogConfig->workMode.enableDebug = false;
rtwdogConfig->testMode = kRTWDOG_TestModeDisabled;
rtwdogConfig->enableUpdate = true;
rtwdogConfig->enableInterrupt = false;
rtwdogConfig->enableWindowMode = false;
rtwdogConfig->windowValue = 0U;
rtwdogConfig->timeoutValue = 0xFFFFU;
```

See also:

rtwdog_config_t

Parameters

- config – Pointer to the RTWDOG configuration structure.

void RTWDOG_Init(RTWDOG_Type *base, const *rtwdog_config_t* *config)

Initializes the RTWDOG module.

This function initializes the RTWDOG. To reconfigure the RTWDOG without forcing a reset first, enableUpdate must be set to true in the configuration.

Example:

```
rtwdog_config_t config;
RTWDOG_GetDefaultConfig(&config);
config.timeoutValue = 0x7ffU;
config.enableUpdate = true;
RTWDOG_Init(wdog_base,&config);
```

Parameters

- base – RTWDOG peripheral base address.
- config – The configuration of the RTWDOG.

void RTWDOG_Deinit(RTWDOG_Type *base)

De-initializes the RTWDOG module.

This function shuts down the RTWDOG. Ensure that the WDOG_CS.UPDATE is 1, which means that the register update is enabled.

Parameters

- base – RTWDOG peripheral base address.

static inline void RTWDOG_Enable(RTWDOG_Type *base)

Enables the RTWDOG module.

This function writes a value into the WDOG_CS register to enable the RTWDOG. The WDOG_CS register is a write-once register. 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 – RTWDOG peripheral base address.

static inline void RTWDOG_Disable(RTWDOG_Type *base)

Disables the RTWDOG module.

This function writes a value into the WDOG_CS register to disable the RTWDOG. The WDOG_CS register is a write-once register. 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 – RTWDOG peripheral base address

static inline void RTWDOG_EnableInterrupts(RTWDOG_Type *base, uint32_t mask)

Enables the RTWDOG interrupt.

This function writes a value into the WDOG_CS register to enable the RTWDOG interrupt. The WDOG_CS register is a write-once register. 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 – RTWDOG peripheral base address.
- mask – The interrupts to enable. The parameter can be a combination of the following source if defined:
  - kRTWDOG_InterruptEnable

static inline void RTWDOG_DisableInterrupts(RTWDOG_Type *base, uint32_t mask)

Disables the RTWDOG interrupt.

This function writes a value into the WDOG_CS register to disable the RTWDOG interrupt. The WDOG_CS register is a write-once register. 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 – RTWDOG peripheral base address.
- mask – The interrupts to disabled. The parameter can be a combination of the following source if defined:
  - kRTWDOG_InterruptEnable

static inline uint32_t RTWDOG_GetStatusFlags(RTWDOG_Type *base)

Gets the RTWDOG all status flags.

This function gets all status flags.

Example to get the running flag:

```
uint32_t status;
status = RTWDOG_GetStatusFlags(wdog_base) & kRTWDOG_RunningFlag;
```

See also:

_rtwdog_status_flags_t

- true: related status flag has been set.
- false: related status flag is not set.

Parameters

- base – RTWDOG peripheral base address

Returns

State of the status flag: asserted (true) or not-asserted (false).

static inline void RTWDOG_EnableWindowMode(RTWDOG_Type *base, bool enable)

Enables/disables the window mode.

Parameters

- base – RTWDOG peripheral base address.
- enable – Enables(true) or disables(false) the feature.

static inline uint32_t RTWDOG_CountToMesec(RTWDOG_Type *base, uint32_t count, uint32_t clockFreqInHz)

Converts raw count value to millisecond.

Note that if the clock frequency is too high the timeout period can be less than 1 ms. In this case this api will return 0 value.

Parameters

- base – RTWDOG peripheral base address.
- count – Raw count value.
- clockFreqInHz – The frequency of the clock source RTWDOG uses.

Returns

Return converted time. Will return 0 if result is larger than 0xFFFFFFFF.

void RTWDOG_ClearStatusFlags(RTWDOG_Type *base, uint32_t mask)

Clears the RTWDOG flag.

This function clears the RTWDOG status flag.

Example to clear an interrupt flag:

```
RTWDOG_ClearStatusFlags(wdog_base,kRTWDOG_InterruptFlag);
```

Parameters

- base – RTWDOG peripheral base address.
- mask – The status flags to clear. The parameter can be any combination of the following values:
  - kRTWDOG_InterruptFlag

static inline void RTWDOG_SetTimeoutValue(RTWDOG_Type *base, uint16_t timeoutCount)

Sets the RTWDOG timeout value.

This function writes a timeout value into the WDOG_TOVAL register. The WDOG_TOVAL register is a write-once register. 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 – RTWDOG peripheral base address
- timeoutCount – RTWDOG timeout value, count of RTWDOG clock ticks.

static inline void RTWDOG_SetWindowValue(RTWDOG_Type *base, uint16_t windowValue)

Sets the RTWDOG window value.

This function writes a window value into the WDOG_WIN register. The WDOG_WIN register is a write-once register. 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 – RTWDOG peripheral base address.

- windowValue – RTWDOG window value.

___STATIC_FORCEINLINE void RTWDOG_Unlock (RTWDOG_Type *base)

Unlocks the RTWDOG register written.

This function unlocks the RTWDOG 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 – RTWDOG peripheral base address

static inline void RTWDOG_Refresh(RTWDOG_Type *base)

Refreshes the RTWDOG timer.

This function feeds the RTWDOG. This function should be called before the Watchdog timer is in timeout. Otherwise, a reset is asserted.

Parameters

- base – RTWDOG peripheral base address

static inline uint32_t RTWDOG_GetCounterValue(RTWDOG_Type *base)

Gets the RTWDOG counter value.

This function gets the RTWDOG counter value.

Parameters

- base – RTWDOG peripheral base address.

Returns

Current RTWDOG 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_RTWDOG_DRIVER_VERSION

RTWDOG driver version.

enum _rtwdog_clock_source

Describes RTWDOG clock source.

Values:

enumerator kRTWDOG_ClockSource0

Clock source 0

enumerator kRTWDOG_ClockSource1
Clock source 1

enumerator kRTWDOG_ClockSource2
Clock source 2

enumerator kRTWDOG_ClockSource3
Clock source 3

enum _rtwdog_clock_prescaler
Describes the selection of the clock prescaler.
Values:

enumerator kRTWDOG_ClockPrescalerDivide1
Divided by 1

enumerator kRTWDOG_ClockPrescalerDivide256
Divided by 256

enum _rtwdog_test_mode
Describes RTWDOG test mode.
Values:

enumerator kRTWDOG_TestModeDisabled
Test Mode disabled

enumerator kRTWDOG_UserModeEnabled
User Mode enabled

enumerator kRTWDOG_LowByteTest
Test Mode enabled, only low byte is used

enumerator kRTWDOG_HighByteTest
Test Mode enabled, only high byte is used

enum _rtwdog_interrupt_enable_t
RTWDOG interrupt configuration structure.
This structure contains the settings for all of the RTWDOG interrupt configurations.
Values:

enumerator kRTWDOG_InterruptEnable
Interrupt is generated before forcing a reset

enum _rtwdog_status_flags_t
RTWDOG status flags.
This structure contains the RTWDOG status flags for use in the RTWDOG functions.
Values:

enumerator kRTWDOG_RunningFlag
Running flag, set when RTWDOG is enabled

enumerator kRTWDOG_InterruptFlag
Interrupt flag, set when interrupt occurs

typedef enum *_rtwdog_clock_source* rtwdog_clock_source_t
Describes RTWDOG clock source.

typedef enum *_rtwdog_clock_prescaler* rtwdog_clock_prescaler_t
Describes the selection of the clock prescaler.

typedef struct *_rtwdog_work_mode* rtwdog_work_mode_t
Defines RTWDOG work mode.

typedef enum *_rtwdog_test_mode* rtwdog_test_mode_t
Describes RTWDOG test mode.

typedef struct *_rtwdog_config* rtwdog_config_t
Describes RTWDOG configuration structure.

struct _rtwdog_work_mode
#include <fsl_rtwdog.h> Defines RTWDOG work mode.

Public Members

bool enableWait
Enables or disables RTWDOG in wait mode

bool enableStop
Enables or disables RTWDOG in stop mode

bool enableDebug
Enables or disables RTWDOG in debug mode

struct _rtwdog_config
#include <fsl_rtwdog.h> Describes RTWDOG configuration structure.

Public Members

bool enableRtwdog
Enables or disables RTWDOG

rtwdog_clock_source_t clockSource
Clock source select

rtwdog_clock_prescaler_t prescaler
Clock prescaler value

rtwdog_work_mode_t workMode
Configures RTWDOG work mode in debug stop and wait mode

rtwdog_test_mode_t testMode
Configures RTWDOG test mode

bool enableUpdate
Update write-once register enable

bool enableInterrupt
Enables or disables RTWDOG interrupt

bool enableWindowMode
Enables or disables RTWDOG window mode

uint16_t windowValue
Window value

uint16_t timeoutValue
Timeout value

2.56 SAI: Serial Audio Interface

2.57 SAI Driver

void SAI_Init(I2S_Type *base)
Initializes the SAI peripheral.

This API gates the SAI clock. The SAI module can't operate unless SAI_Init is called to enable the clock.

Parameters

- base – SAI base pointer.

void SAI_Deinit(I2S_Type *base)
De-initializes the SAI peripheral.

This API gates the SAI clock. The SAI module can't operate unless SAI_TxInit or SAI_RxInit is called to enable the clock.

Parameters

- base – SAI base pointer.

void SAI_TxReset(I2S_Type *base)
Resets the SAI Tx.

This function enables the software reset and FIFO reset of SAI Tx. After reset, clear the reset bit.

Parameters

- base – SAI base pointer

void SAI_RxReset(I2S_Type *base)

Resets the SAI Rx.

This function enables the software reset and FIFO reset of SAI Rx. After reset, clear the reset bit.

Parameters

- base – SAI base pointer

void SAI_TxEnable(I2S_Type *base, bool enable)

Enables/disables the SAI Tx.

Parameters

- base – SAI base pointer.
- enable – True means enable SAI Tx, false means disable.

void SAI_RxEnable(I2S_Type *base, bool enable)

Enables/disables the SAI Rx.

Parameters

- base – SAI base pointer.
- enable – True means enable SAI Rx, false means disable.

static inline void SAI_TxSetBitClockDirection(I2S_Type *base, *sai_master_slave_t* masterSlave)

Set Rx bit clock direction.

Select bit clock direction, master or slave.

Parameters

- base – SAI base pointer.
- masterSlave – reference sai_master_slave_t.

static inline void SAI_RxSetBitClockDirection(I2S_Type *base, *sai_master_slave_t* masterSlave)

Set Rx bit clock direction.

Select bit clock direction, master or slave.

Parameters

- base – SAI base pointer.
- masterSlave – reference sai_master_slave_t.

static inline void SAI_RxSetFrameSyncDirection(I2S_Type *base, *sai_master_slave_t* masterSlave)

Set Rx frame sync direction.

Select frame sync direction, master or slave.

Parameters

- base – SAI base pointer.
- masterSlave – reference sai_master_slave_t.

static inline void SAI_TxSetFrameSyncDirection(I2S_Type *base, *sai_master_slave_t* masterSlave)

Set Tx frame sync direction.

Select frame sync direction, master or slave.

Parameters

- base – SAI base pointer.

- masterSlave – reference sai_master_slave_t.

void SAI_TxSetBitClockRate(I2S_Type *base, uint32_t sourceClockHz, uint32_t sampleRate, uint32_t bitWidth, uint32_t channelNumbers)

Transmitter bit clock rate configurations.

Parameters

- base – SAI base pointer.
- sourceClockHz – Bit clock source frequency.
- sampleRate – Audio data sample rate.
- bitWidth – Audio data bitWidth.
- channelNumbers – Audio channel numbers.

void SAI_RxSetBitClockRate(I2S_Type *base, uint32_t sourceClockHz, uint32_t sampleRate, uint32_t bitWidth, uint32_t channelNumbers)

Receiver bit clock rate configurations.

Parameters

- base – SAI base pointer.
- sourceClockHz – Bit clock source frequency.
- sampleRate – Audio data sample rate.
- bitWidth – Audio data bitWidth.
- channelNumbers – Audio channel numbers.

void SAI_TxSetBitclockConfig(I2S_Type *base, *sai_master_slave_t* masterSlave, *sai_bit_clock_t* *config)

Transmitter Bit clock configurations.

Parameters

- base – SAI base pointer.
- masterSlave – master or slave.
- config – bit clock other configurations, can be NULL in slave mode.

void SAI_RxSetBitclockConfig(I2S_Type *base, *sai_master_slave_t* masterSlave, *sai_bit_clock_t* *config)

Receiver Bit clock configurations.

Parameters

- base – SAI base pointer.
- masterSlave – master or slave.
- config – bit clock other configurations, can be NULL in slave mode.

void SAI_SetMasterClockConfig(I2S_Type *base, *sai_master_clock_t* *config)

Master clock configurations.

Parameters

- base – SAI base pointer.
- config – master clock configurations.

void SAI_TxSetFifoConfig(I2S_Type *base, *sai_fifo_t* *config)

SAI transmitter fifo configurations.

Parameters

- base – SAI base pointer.
- config – fifo configurations.

void SAI_RxSetFifoConfig(I2S_Type *base, *sai_fifo_t* *config)

SAI receiver fifo configurations.

Parameters

- base – SAI base pointer.
- config – fifo configurations.

void SAI_TxSetFrameSyncConfig(I2S_Type *base, *sai_master_slave_t* masterSlave, *sai_frame_sync_t* *config)

SAI transmitter Frame sync configurations.

Parameters

- base – SAI base pointer.
- masterSlave – master or slave.
- config – frame sync configurations, can be NULL in slave mode.

void SAI_RxSetFrameSyncConfig(I2S_Type *base, *sai_master_slave_t* masterSlave, *sai_frame_sync_t* *config)

SAI receiver Frame sync configurations.

Parameters

- base – SAI base pointer.
- masterSlave – master or slave.
- config – frame sync configurations, can be NULL in slave mode.

void SAI_TxSetSerialDataConfig(I2S_Type *base, *sai_serial_data_t* *config)

SAI transmitter Serial data configurations.

Parameters

- base – SAI base pointer.
- config – serial data configurations.

void SAI_RxSetSerialDataConfig(I2S_Type *base, *sai_serial_data_t* *config)

SAI receiver Serial data configurations.

Parameters

- base – SAI base pointer.
- config – serial data configurations.

void SAI_TxSetConfig(I2S_Type *base, *sai_transceiver_t* *config)

SAI transmitter configurations.

Parameters

- base – SAI base pointer.
- config – transmitter configurations.

void SAI_RxSetConfig(I2S_Type *base, *sai_transceiver_t* *config)

SAI receiver configurations.

Parameters

- base – SAI base pointer.
- config – receiver configurations.

void SAI_GetClassicI2SConfig(*sai_transceiver_t* *config, *sai_word_width_t* bitWidth, *sai_mono_stereo_t* mode, uint32_t saiChannelMask)

Get classic I2S mode configurations.

Parameters

- config – transceiver configurations.
- bitWidth – audio data bitWidth.
- mode – audio data channel.
- saiChannelMask – mask value of the channel to be enable.

void SAI_GetLeftJustifiedConfig(*sai_transceiver_t* *config, *sai_word_width_t* bitWidth, *sai_mono_stereo_t* mode, uint32_t saiChannelMask)

Get left justified mode configurations.

Parameters

- config – transceiver configurations.
- bitWidth – audio data bitWidth.
- mode – audio data channel.
- saiChannelMask – mask value of the channel to be enable.

void SAI_GetRightJustifiedConfig(*sai_transceiver_t* *config, *sai_word_width_t* bitWidth, *sai_mono_stereo_t* mode, uint32_t saiChannelMask)

Get right justified mode configurations.

Parameters

- config – transceiver configurations.
- bitWidth – audio data bitWidth.
- mode – audio data channel.
- saiChannelMask – mask value of the channel to be enable.

void SAI_GetTDMConfig(*sai_transceiver_t* *config, *sai_frame_sync_len_t* frameSyncWidth, *sai_word_width_t* bitWidth, uint32_t dataWordNum, uint32_t saiChannelMask)

Get TDM mode configurations.

Parameters

- config – transceiver configurations.
- frameSyncWidth – length of frame sync.
- bitWidth – audio data word width.
- dataWordNum – word number in one frame.
- saiChannelMask – mask value of the channel to be enable.

void SAI_GetDSPConfig(*sai_transceiver_t* *config, *sai_frame_sync_len_t* frameSyncWidth, *sai_word_width_t* bitWidth, *sai_mono_stereo_t* mode, uint32_t saiChannelMask)

Get DSP mode configurations.

DSP/PCM MODE B configuration flow for TX. RX is similiar but uses SAI_RxSetConfig instead of SAI_TxSetConfig:

```
SAI_GetDSPConfig(config, kSAI_FrameSyncLenOneBitClk, bitWidth, kSAI_Stereo, channelMask)
SAI_TxSetConfig(base, config)
```

Note: DSP mode is also called PCM mode which support MODE A and MODE B, DSP/PCM MODE A configuration flow. RX is similiar but uses SAI_RxSetConfig instead of SAI_TxSetConfig:

```
SAI_GetDSPConfig(config, kSAI_FrameSyncLenOneBitClk, bitWidth, kSAI_Stereo, channelMask)
config->frameSync.frameSyncEarly    = true;
SAI_TxSetConfig(base, config)
```

Parameters

- config – transceiver configurations.
- frameSyncWidth – length of frame sync.
- bitWidth – audio data bitWidth.
- mode – audio data channel.
- saiChannelMask – mask value of the channel to enable.

static inline uint32_t SAI_TxGetStatusFlag(I2S_Type *base)

Gets the SAI Tx status flag state.

Parameters

- base – SAI base pointer

Returns

SAI Tx status flag value. Use the Status Mask to get the status value needed.

static inline void SAI_TxClearStatusFlags(I2S_Type *base, uint32_t mask)

Clears the SAI Tx status flag state.

Parameters

- base – SAI base pointer
- mask – State mask. It can be a combination of the following source if defined:
  - kSAI_WordStartFlag
  - kSAI_SyncErrorFlag
  - kSAI_FIFOErrorFlag

static inline uint32_t SAI_RxGetStatusFlag(I2S_Type *base)

Gets the SAI Tx status flag state.

Parameters

- base – SAI base pointer

Returns

SAI Rx status flag value. Use the Status Mask to get the status value needed.

static inline void SAI_RxClearStatusFlags(I2S_Type *base, uint32_t mask)

Clears the SAI Rx status flag state.

Parameters

- base – SAI base pointer

- mask – State mask. It can be a combination of the following sources if defined.
  - kSAI_WordStartFlag
  - kSAI_SyncErrorFlag
  - kSAI_FIFOErrorFlag

void SAI_TxSoftwareReset(I2S_Type *base, *sai_reset_type_t* resetType)

Do software reset or FIFO reset .

FIFO reset means clear all the data in the FIFO, and make the FIFO pointer both to 0. Software reset means clear the Tx internal logic, including the bit clock, frame count etc. But software reset will not clear any configuration registers like TCR1~TCR5. This function will also clear all the error flags such as FIFO error, sync error etc.

Parameters

- base – SAI base pointer
- resetType – Reset type, FIFO reset or software reset

void SAI_RxSoftwareReset(I2S_Type *base, *sai_reset_type_t* resetType)

Do software reset or FIFO reset .

FIFO reset means clear all the data in the FIFO, and make the FIFO pointer both to 0. Software reset means clear the Rx internal logic, including the bit clock, frame count etc. But software reset will not clear any configuration registers like RCR1~RCR5. This function will also clear all the error flags such as FIFO error, sync error etc.

Parameters

- base – SAI base pointer
- resetType – Reset type, FIFO reset or software reset

void SAI_TxSetChannelFIFOMask(I2S_Type *base, uint8_t mask)

Set the Tx channel FIFO enable mask.

Parameters

- base – SAI base pointer
- mask – Channel enable mask, 0 means all channel FIFO disabled, 1 means channel 0 enabled, 3 means both channel 0 and channel 1 enabled.

void SAI_RxSetChannelFIFOMask(I2S_Type *base, uint8_t mask)

Set the Rx channel FIFO enable mask.

Parameters

- base – SAI base pointer
- mask – Channel enable mask, 0 means all channel FIFO disabled, 1 means channel 0 enabled, 3 means both channel 0 and channel 1 enabled.

void SAI_TxSetDataOrder(I2S_Type *base, *sai_data_order_t* order)

Set the Tx data order.

Parameters

- base – SAI base pointer
- order – Data order MSB or LSB

void SAI_RxSetDataOrder(I2S_Type *base, *sai_data_order_t* order)

Set the Rx data order.

Parameters

- base – SAI base pointer
- order – Data order MSB or LSB

void SAI_TxSetBitClockPolarity(I2S_Type *base, *sai_clock_polarity_t* polarity)

Set the Tx data order.

Parameters

- base – SAI base pointer
- polarity –

void SAI_RxSetBitClockPolarity(I2S_Type *base, *sai_clock_polarity_t* polarity)

Set the Rx data order.

Parameters

- base – SAI base pointer
- polarity –

void SAI_TxSetFrameSyncPolarity(I2S_Type *base, *sai_clock_polarity_t* polarity)

Set the Tx data order.

Parameters

- base – SAI base pointer
- polarity –

void SAI_RxSetFrameSyncPolarity(I2S_Type *base, *sai_clock_polarity_t* polarity)

Set the Rx data order.

Parameters

- base – SAI base pointer
- polarity –

void SAI_TxSetFIFOPacking(I2S_Type *base, *sai_fifo_packing_t* pack)

Set Tx FIFO packing feature.

Parameters

- base – SAI base pointer.
- pack – FIFO pack type. It is element of sai_fifo_packing_t.

void SAI_RxSetFIFOPacking(I2S_Type *base, *sai_fifo_packing_t* pack)

Set Rx FIFO packing feature.

Parameters

- base – SAI base pointer.
- pack – FIFO pack type. It is element of sai_fifo_packing_t.

static inline void SAI_TxSetFIFOErrorContinue(I2S_Type *base, bool isEnabled)

Set Tx FIFO error continue.

FIFO error continue mode means SAI will keep running while FIFO error occurred. If this feature not enabled, SAI will hang and users need to clear FEF flag in TCSR register.

Parameters

- base – SAI base pointer.
- isEnabled – Is FIFO error continue enabled, true means enable, false means disable.

static inline void SAI_RxSetFIFOErrorContinue(I2S_Type *base, bool isEnabled)

Set Rx FIFO error continue.

FIFO error continue mode means SAI will keep running while FIFO error occurred. If this feature not enabled, SAI will hang and users need to clear FEF flag in RCSR register.

Parameters

- base – SAI base pointer.
- isEnabled – Is FIFO error continue enabled, true means enable, false means disable.

static inline void SAI_TxEnableInterrupts(I2S_Type *base, uint32_t mask)

Enables the SAI Tx interrupt requests.

Parameters

- base – SAI base pointer
- mask – interrupt source The parameter can be a combination of the following sources if defined.
  - kSAI_WordStartInterruptEnable
  - kSAI_SyncErrorInterruptEnable
  - kSAI_FIFOWarningInterruptEnable
  - kSAI_FIFORequestInterruptEnable
  - kSAI_FIFOErrorInterruptEnable

static inline void SAI_RxEnableInterrupts(I2S_Type *base, uint32_t mask)

Enables the SAI Rx interrupt requests.

Parameters

- base – SAI base pointer
- mask – interrupt source The parameter can be a combination of the following sources if defined.
  - kSAI_WordStartInterruptEnable
  - kSAI_SyncErrorInterruptEnable
  - kSAI_FIFOWarningInterruptEnable
  - kSAI_FIFORequestInterruptEnable
  - kSAI_FIFOErrorInterruptEnable

static inline void SAI_TxDisableInterrupts(I2S_Type *base, uint32_t mask)

Disables the SAI Tx interrupt requests.

Parameters

- base – SAI base pointer
- mask – interrupt source The parameter can be a combination of the following sources if defined.
  - kSAI_WordStartInterruptEnable
  - kSAI_SyncErrorInterruptEnable
  - kSAI_FIFOWarningInterruptEnable
  - kSAI_FIFORequestInterruptEnable
  - kSAI_FIFOErrorInterruptEnable

static inline void SAI_RxDisableInterrupts(I2S_Type *base, uint32_t mask)

Disables the SAI Rx interrupt requests.

Parameters

- base – SAI base pointer
- mask – interrupt source The parameter can be a combination of the following sources if defined.
  - kSAI_WordStartInterruptEnable
  - kSAI_SyncErrorInterruptEnable
  - kSAI_FIFOWarningInterruptEnable
  - kSAI_FIFORequestInterruptEnable
  - kSAI_FIFOErrorInterruptEnable

static inline void SAI_TxEnableDMA(I2S_Type *base, uint32_t mask, bool enable)

Enables/disables the SAI Tx DMA requests.

Parameters

- base – SAI base pointer
- mask – DMA source The parameter can be combination of the following sources if defined.
  - kSAI_FIFOWarningDMAEnable
  - kSAI_FIFORequestDMAEnable
- enable – True means enable DMA, false means disable DMA.

static inline void SAI_RxEnableDMA(I2S_Type *base, uint32_t mask, bool enable)

Enables/disables the SAI Rx DMA requests.

Parameters

- base – SAI base pointer
- mask – DMA source The parameter can be a combination of the following sources if defined.
  - kSAI_FIFOWarningDMAEnable
  - kSAI_FIFORequestDMAEnable
- enable – True means enable DMA, false means disable DMA.

static inline uintptr_t SAI_TxGetDataRegisterAddress(I2S_Type *base, uint32_t channel)

Gets the SAI Tx data register address.

This API is used to provide a transfer address for the SAI DMA transfer configuration.

Parameters

- base – SAI base pointer.
- channel – Which data channel used.

Returns

data register address.

static inline uintptr_t SAI_RxGetDataRegisterAddress(I2S_Type *base, uint32_t channel)

Gets the SAI Rx data register address.

This API is used to provide a transfer address for the SAI DMA transfer configuration.

Parameters

- base – SAI base pointer.
- channel – Which data channel used.

Returns
data register address.

void SAI_WriteBlocking(I2S_Type *base, uint32_t channel, uint32_t bitWidth, uint8_t *buffer, uint32_t size)

Sends data using a blocking method.

Note: This function blocks by polling until data is ready to be sent.

Parameters

- base – SAI base pointer.
- channel – Data channel used.
- bitWidth – How many bits in an audio word; usually 8/16/24/32 bits.
- buffer – Pointer to the data to be written.
- size – Bytes to be written.

void SAI_WriteMultiChannelBlocking(I2S_Type *base, uint32_t channel, uint32_t channelMask, uint32_t bitWidth, uint8_t *buffer, uint32_t size)

Sends data to multi channel using a blocking method.

Note: This function blocks by polling until data is ready to be sent.

Parameters

- base – SAI base pointer.
- channel – Data channel used.
- channelMask – channel mask.
- bitWidth – How many bits in an audio word; usually 8/16/24/32 bits.
- buffer – Pointer to the data to be written.
- size – Bytes to be written.

static inline void SAI_WriteData(I2S_Type *base, uint32_t channel, uint32_t data)

Writes data into SAI FIFO.

Parameters

- base – SAI base pointer.
- channel – Data channel used.
- data – Data needs to be written.

void SAI_ReadBlocking(I2S_Type *base, uint32_t channel, uint32_t bitWidth, uint8_t *buffer, uint32_t size)

Receives data using a blocking method.

Note: This function blocks by polling until data is ready to be sent.

Parameters

- base – SAI base pointer.
- channel – Data channel used.
- bitWidth – How many bits in an audio word; usually 8/16/24/32 bits.
- buffer – Pointer to the data to be read.
- size – Bytes to be read.

void SAI_ReadMultiChannelBlocking(I2S_Type *base, uint32_t channel, uint32_t channelMask, uint32_t bitWidth, uint8_t *buffer, uint32_t size)

Receives multi channel data using a blocking method.

Note: This function blocks by polling until data is ready to be sent.

Parameters

- base – SAI base pointer.
- channel – Data channel used.
- channelMask – channel mask.
- bitWidth – How many bits in an audio word; usually 8/16/24/32 bits.
- buffer – Pointer to the data to be read.
- size – Bytes to be read.

static inline uint32_t SAI_ReadData(I2S_Type *base, uint32_t channel)

Reads data from the SAI FIFO.

Parameters

- base – SAI base pointer.
- channel – Data channel used.

Returns

Data in SAI FIFO.

void SAI_TransferTxCreateHandle(I2S_Type *base, *sai_handle_t* *handle, *sai_transfer_callback_t* callback, void *userData)

Initializes the SAI Tx handle.

This function initializes the Tx handle for the SAI Tx transactional APIs. Call this function once to get the handle initialized.

Parameters

- base – SAI base pointer
- handle – SAI handle pointer.
- callback – Pointer to the user callback function.
- userData – User parameter passed to the callback function

void SAI_TransferRxCreateHandle(I2S_Type *base, *sai_handle_t* *handle, *sai_transfer_callback_t* callback, void *userData)

Initializes the SAI Rx handle.

This function initializes the Rx handle for the SAI Rx transactional APIs. Call this function once to get the handle initialized.

Parameters

- base – SAI base pointer.
- handle – SAI handle pointer.
- callback – Pointer to the user callback function.
- userData – User parameter passed to the callback function.

void SAI_TransferTxSetConfig(I2S_Type *base, *sai_handle_t* *handle, *sai_transceiver_t* *config)

SAI transmitter transfer configurations.

This function initializes the Tx, include bit clock, frame sync, master clock, serial data and fifo configurations.

Parameters

- base – SAI base pointer.
- handle – SAI handle pointer.
- config – tranmitter configurations.

void SAI_TransferRxSetConfig(I2S_Type *base, *sai_handle_t* *handle, *sai_transceiver_t* *config)

SAI receiver transfer configurations.

This function initializes the Rx, include bit clock, frame sync, master clock, serial data and fifo configurations.

Parameters

- base – SAI base pointer.
- handle – SAI handle pointer.
- config – receiver configurations.

status_t SAI_TransferSendNonBlocking(I2S_Type *base, *sai_handle_t* *handle, *sai_transfer_t* *xfer)

Performs an interrupt non-blocking send transfer on SAI.

Note: This API returns immediately after the transfer initiates. Call the SAI_TxGetTransferStatusIRQ to poll the transfer status and check whether the transfer is finished. If the return status is not kStatus_SAI_Busy, the transfer is finished.

Parameters

- base – SAI base pointer.
- handle – Pointer to the sai_handle_t structure which stores the transfer state.
- xfer – Pointer to the sai_transfer_t structure.

Return values

- kStatus_Success – Successfully started the data receive.
- kStatus_SAI_TxBusy – Previous receive still not finished.
- kStatus_InvalidArgument – The input parameter is invalid.

status_t SAI_TransferReceiveNonBlocking(I2S_Type *base, *sai_handle_t* *handle, *sai_transfer_t* *xfer)

Performs an interrupt non-blocking receive transfer on SAI.

Note: This API returns immediately after the transfer initiates. Call the SAI_RxGetTransferStatusIRQ to poll the transfer status and check whether the transfer is finished. If the return status is not kStatus_SAI_Busy, the transfer is finished.

Parameters

- base – SAI base pointer
- handle – Pointer to the sai_handle_t structure which stores the transfer state.
- xfer – Pointer to the sai_transfer_t structure.

Return values

- kStatus_Success – Successfully started the data receive.
- kStatus_SAI_RxBusy – Previous receive still not finished.
- kStatus_InvalidArgument – The input parameter is invalid.

status_t SAI_TransferGetSendCount(I2S_Type *base, *sai_handle_t* *handle, size_t *count)

Gets a set byte count.

Parameters

- base – SAI base pointer.
- handle – Pointer to the sai_handle_t structure which stores the transfer state.
- count – Bytes count sent.

Return values

- kStatus_Success – Succeed get the transfer count.
- kStatus_NoTransferInProgress – There is not a non-blocking transaction currently in progress.

status_t SAI_TransferGetReceiveCount(I2S_Type *base, *sai_handle_t* *handle, size_t *count)

Gets a received byte count.

Parameters

- base – SAI base pointer.
- handle – Pointer to the sai_handle_t structure which stores the transfer state.
- count – Bytes count received.

Return values

- kStatus_Success – Succeed get the transfer count.
- kStatus_NoTransferInProgress – There is not a non-blocking transaction currently in progress.

void SAI_TransferAbortSend(I2S_Type *base, *sai_handle_t* *handle)

Aborts the current send.

Note: This API can be called any time when an interrupt non-blocking transfer initiates to abort the transfer early.

Parameters

- base – SAI base pointer.
- handle – Pointer to the sai_handle_t structure which stores the transfer state.

void SAI_TransferAbortReceive(I2S_Type *base, *sai_handle_t* *handle)

Aborts the current IRQ receive.

Note: This API can be called when an interrupt non-blocking transfer initiates to abort the transfer early.

Parameters

- base – SAI base pointer
- handle – Pointer to the sai_handle_t structure which stores the transfer state.

void SAI_TransferTerminateSend(I2S_Type *base, *sai_handle_t* *handle)

Terminate all SAI send.

This function will clear all transfer slots buffered in the sai queue. If users only want to abort the current transfer slot, please call SAI_TransferAbortSend.

Parameters

- base – SAI base pointer.
- handle – SAI eDMA handle pointer.

void SAI_TransferTerminateReceive(I2S_Type *base, *sai_handle_t* *handle)

Terminate all SAI receive.

This function will clear all transfer slots buffered in the sai queue. If users only want to abort the current transfer slot, please call SAI_TransferAbortReceive.

Parameters

- base – SAI base pointer.
- handle – SAI eDMA handle pointer.

void SAI_TransferTxHandleIRQ(I2S_Type *base, *sai_handle_t* *handle)

Tx interrupt handler.

Parameters

- base – SAI base pointer.
- handle – Pointer to the sai_handle_t structure.

void SAI_TransferRxHandleIRQ(I2S_Type *base, *sai_handle_t* *handle)

Tx interrupt handler.

Parameters

- base – SAI base pointer.
- handle – Pointer to the sai_handle_t structure.

void SAI_DriverIRQHandler(uint32_t instance)

SAI driver IRQ handler common entry.

This function provides the common IRQ request entry for SAI.

Parameters

- instance – SAI instance.

FSL_SAI_DRIVER_VERSION

Version 2.4.11

_sai_status_t, SAI return status.

Values:

enumerator kStatus_SAI_TxBusy

SAI Tx is busy.

enumerator kStatus_SAI_RxBusy

SAI Rx is busy.

enumerator kStatus_SAI_TxError

SAI Tx FIFO error.

enumerator kStatus_SAI_RxError

SAI Rx FIFO error.

enumerator kStatus_SAI_QueueFull

SAI transfer queue is full.

enumerator kStatus_SAI_TxIdle

SAI Tx is idle

enumerator kStatus_SAI_RxIdle

SAI Rx is idle

_sai_channel_mask,.sai channel mask value, actual channel numbers is depend soc specific

Values:

enumerator kSAI_Channel0Mask

channel 0 mask value

enumerator kSAI_Channel1Mask

channel 1 mask value

enumerator kSAI_Channel2Mask

channel 2 mask value

enumerator kSAI_Channel3Mask
channel 3 mask value

enumerator kSAI_Channel4Mask
channel 4 mask value

enumerator kSAI_Channel5Mask
channel 5 mask value

enumerator kSAI_Channel6Mask
channel 6 mask value

enumerator kSAI_Channel7Mask
channel 7 mask value

enum _sai_protocol
Define the SAI bus type.
Values:

enumerator kSAI_BusLeftJustified
Uses left justified format.

enumerator kSAI_BusRightJustified
Uses right justified format.

enumerator kSAI_BusI2S
Uses I2S format.

enumerator kSAI_BusPCMA
Uses I2S PCM A format.

enumerator kSAI_BusPCMB
Uses I2S PCM B format.

enum _sai_master_slave
Master or slave mode.
Values:

enumerator kSAI_Master
Master mode include bclk and frame sync

enumerator kSAI_Slave
Slave mode include bclk and frame sync

enumerator kSAI_Bclk_Master_FrameSync_Slave
bclk in master mode, frame sync in slave mode

enumerator kSAI_Bclk_Slave_FrameSync_Master
bclk in slave mode, frame sync in master mode

enum _sai_mono_stereo
Mono or stereo audio format.
Values:

enumerator kSAI_Stereo
Stereo sound.

enumerator kSAI_MonoRight
Only Right channel have sound.

enumerator kSAI_MonoLeft
Only left channel have sound.

enum _sai_data_order
SAI data order, MSB or LSB.
Values:

enumerator kSAI_DataLSB
LSB bit transferred first

enumerator kSAI_DataMSB
MSB bit transferred first

enum _sai_clock_polarity
SAI clock polarity, active high or low.
Values:

enumerator kSAI_PolarityActiveHigh
Drive outputs on rising edge

enumerator kSAI_PolarityActiveLow
Drive outputs on falling edge

enumerator kSAI_SampleOnFallingEdge
Sample inputs on falling edge

enumerator kSAI_SampleOnRisingEdge
Sample inputs on rising edge

enum _sai_sync_mode
Synchronous or asynchronous mode.
Values:

enumerator kSAI_ModeAsync
Asynchronous mode

enumerator kSAI_ModeSync
Synchronous mode (with receiver or transmit)

enumerator kSAI_ModeSyncWithOtherTx
Synchronous with another SAI transmit

enumerator kSAI_ModeSyncWithOtherRx
Synchronous with another SAI receiver

enum _sai_bclk_source
Bit clock source.
Values:

enumerator kSAI_BclkSourceBusclk
Bit clock using bus clock

enumerator kSAI_BclkSourceMclkOption1
Bit clock MCLK option 1

enumerator kSAI_BclkSourceMclkOption2
Bit clock MCLK option2

enumerator kSAI_BclkSourceMclkOption3
Bit clock MCLK option3

enumerator kSAI_BclkSourceMclkDiv
Bit clock using master clock divider

enumerator kSAI_BclkSourceOtherSai0
Bit clock from other SAI device

enumerator kSAI_BclkSourceOtherSai1
Bit clock from other SAI device

_sai_interrupt_enable_t, The SAI interrupt enable flag
Values:

enumerator kSAI_WordStartInterruptEnable
Word start flag, means the first word in a frame detected

enumerator kSAI_SyncErrorInterruptEnable
Sync error flag, means the sync error is detected

enumerator kSAI_FIFOWarningInterruptEnable
FIFO warning flag, means the FIFO is empty

enumerator kSAI_FIFOErrorInterruptEnable
FIFO error flag

enumerator kSAI_FIFORequestInterruptEnable
FIFO request, means reached watermark

_sai_dma_enable_t, The DMA request sources

Values:

enumerator kSAI_FIFOWarningDMAEnable
FIFO warning caused by the DMA request

enumerator kSAI_FIFORequestDMAEnable
FIFO request caused by the DMA request

_sai_flags, The SAI status flag

Values:

enumerator kSAI_WordStartFlag
Word start flag, means the first word in a frame detected

enumerator kSAI_SyncErrorFlag
Sync error flag, means the sync error is detected

enumerator kSAI_FIFOErrorFlag
FIFO error flag

enumerator kSAI_FIFORequestFlag
FIFO request flag.

enumerator kSAI_FIFOWarningFlag
FIFO warning flag

enum __sai_reset_type
The reset type.

Values:

enumerator kSAI_ResetTypeSoftware
Software reset, reset the logic state

enumerator kSAI_ResetTypeFIFO
FIFO reset, reset the FIFO read and write pointer

enumerator kSAI_ResetAll
All reset.

enum _sai_fifo_packing

The SAI packing mode The mode includes 8 bit and 16 bit packing.

Values:

enumerator kSAI_FifoPackingDisabled

Packing disabled

enumerator kSAI_FifoPacking8bit

8 bit packing enabled

enumerator kSAI_FifoPacking16bit

16bit packing enabled

enum _sai_sample_rate

Audio sample rate.

Values:

enumerator kSAI_SampleRate8KHz

Sample rate 8000 Hz

enumerator kSAI_SampleRate11025Hz

Sample rate 11025 Hz

enumerator kSAI_SampleRate12KHz

Sample rate 12000 Hz

enumerator kSAI_SampleRate16KHz

Sample rate 16000 Hz

enumerator kSAI_SampleRate22050Hz

Sample rate 22050 Hz

enumerator kSAI_SampleRate24KHz

Sample rate 24000 Hz

enumerator kSAI_SampleRate32KHz

Sample rate 32000 Hz

enumerator kSAI_SampleRate44100Hz

Sample rate 44100 Hz

enumerator kSAI_SampleRate48KHz

Sample rate 48000 Hz

enumerator kSAI_SampleRate96KHz

Sample rate 96000 Hz

enumerator kSAI_SampleRate192KHz
Sample rate 192000 Hz

enumerator kSAI_SampleRate384KHz
Sample rate 384000 Hz

enum _sai_word_width
Audio word width.
Values:

enumerator kSAI_WordWidth8bits
Audio data width 8 bits

enumerator kSAI_WordWidth16bits
Audio data width 16 bits

enumerator kSAI_WordWidth24bits
Audio data width 24 bits

enumerator kSAI_WordWidth32bits
Audio data width 32 bits

enum _sai_data_pin_state
sai data pin state definition
Values:

enumerator kSAI_DataPinStateTriState
transmit data pins are tri-stated when slots are masked or channels are disabled

enumerator kSAI_DataPinStateOutputZero
transmit data pins are never tri-stated and will output zero when slots are masked or channel disabled

enum _sai_fifo_combine
sai fifo combine mode definition
Values:

enumerator kSAI_FifoCombineDisabled
sai TX/RX fifo combine mode disabled

enumerator kSAI_FifoCombineModeEnabledOnRead
sai TX fifo combine mode enabled on FIFO reads

enumerator kSAI_FifoCombineModeEnabledOnWrite
sai TX fifo combine mode enabled on FIFO write

enumerator kSAI_RxFifoCombineModeEnabledOnWrite
sai RX fifo combine mode enabled on FIFO write

enumerator kSAI_RXFifoCombineModeEnabledOnRead
sai RX fifo combine mode enabled on FIFO reads

enumerator kSAI_FifoCombineModeEnabledOnReadWrite
sai TX/RX fifo combined mode enabled on FIFO read/writes

enum _sai_transceiver_type
sai transceiver type
Values:

enumerator kSAI_Transmitter
sai transmitter

enumerator kSAI_Receiver
sai receiver

enum _sai_frame_sync_len
sai frame sync len
Values:

enumerator kSAI_FrameSyncLenOneBitClk
1 bit clock frame sync len for DSP mode

enumerator kSAI_FrameSyncLenPerWordWidth
Frame sync length decided by word width

typedef enum *_sai_protocol* sai_protocol_t
Define the SAI bus type.

typedef enum *_sai_master_slave* sai_master_slave_t
Master or slave mode.

typedef enum *_sai_mono_stereo* sai_mono_stereo_t
Mono or stereo audio format.

typedef enum *_sai_data_order* sai_data_order_t
SAI data order, MSB or LSB.

typedef enum *_sai_clock_polarity* sai_clock_polarity_t
SAI clock polarity, active high or low.

typedef enum *_sai_sync_mode* sai_sync_mode_t
Synchronous or asynchronous mode.

typedef enum *_sai_bclk_source* sai_bclk_source_t
Bit clock source.

typedef enum *_sai_reset_type* sai_reset_type_t
: The reset type.

typedef enum *_sai_fifo_packing* sai_fifo_packing_t
: The SAI packing mode The mode includes 8 bit and 16 bit packing.

typedef struct *_sai_config* sai_config_t
: SAI user configuration structure.

typedef enum *_sai_sample_rate* sai_sample_rate_t
: Audio sample rate.

typedef enum *_sai_word_width* sai_word_width_t
: Audio word width.

typedef enum *_sai_data_pin_state* sai_data_pin_state_t
: sai data pin state definition

typedef enum *_sai_fifo_combine* sai_fifo_combine_t
: sai fifo combine mode definition

typedef enum *_sai_transceiver_type* sai_transceiver_type_t
: sai transceiver type

typedef enum *_sai_frame_sync_len* sai_frame_sync_len_t
: sai frame sync len

typedef struct *_sai_transfer_format* sai_transfer_format_t
: sai transfer format

typedef struct *_sai_master_clock* sai_master_clock_t
: master clock configurations

typedef struct *_sai_fifo* sai_fifo_t
: sai fifo configurations

typedef struct *_sai_bit_clock* sai_bit_clock_t
: sai bit clock configurations

typedef struct *_sai_frame_sync* sai_frame_sync_t
: sai frame sync configurations

typedef struct *_sai_serial_data* sai_serial_data_t
: sai serial data configurations

typedef struct *_sai_transceiver* sai_transceiver_t
: sai transceiver configurations

typedef struct *_sai_transfer* sai_transfer_t

SAI transfer structure.

typedef struct *_sai_handle* sai_handle_t

typedef void (*sai_transfer_callback_t)(I2S_Type *base, *sai_handle_t* *handle, *status_t* status, void *userData)

SAI transfer callback prototype.

MCUX_SDK_SAI_ALLOW_NULL_FIFO_WATERMARK

Used to control whether *SAI_RxSetFifoConfig()*/SAI_TxSetFifoConfig() allows a NULL FIFO watermark.

If this macro is set to 0 then *SAI_RxSetFifoConfig()*/SAI_TxSetFifoConfig() will set the watermark to half of the FIFO's depth if passed a NULL watermark.

MCUX_SDK_SAI_DISABLE_IMPLICIT_CHAN_CONFIG

Disable implicit channel data configuration within *SAI_TxSetConfig()*/SAI_RxSetConfig().

Use this macro to control whether *SAI_RxSetConfig()*/SAI_TxSetConfig() will attempt to implicitly configure the channel data. By channel data we mean the startChannel, channelMask, endChannel, and channelNums fields from the sai_transciever_t structure. By default, *SAI_TxSetConfig()*/SAI_RxSetConfig() will attempt to compute these fields, which may not be desired in cases where the user wants to set them before the call to said functions.

SAI_XFER_QUEUE_SIZE

SAI transfer queue size, user can refine it according to use case.

FSL_SAI_HAS_FIFO_EXTEND_FEATURE

sai fifo feature

struct _sai_config

#include <fsl_sai.h> SAI user configuration structure.

Public Members

sai_protocol_t protocol

Audio bus protocol in SAI

sai_sync_mode_t syncMode

SAI sync mode, control Tx/Rx clock sync

bool mclkOutputEnable

Master clock output enable, true means master clock divider enabled

sai_bclk_source_t bclkSource

Bit Clock source

sai_master_slave_t masterSlave

Master or slave

struct _sai_transfer_format

#include <fsl_sai.h> sai transfer format

Public Members

uint32_t sampleRate_Hz

Sample rate of audio data

uint32_t bitWidth

Data length of audio data, usually 8/16/24/32 bits

sai_mono_stereo_t stereo

Mono or stereo

uint32_t masterClockHz

Master clock frequency in Hz

uint8_t watermark

Watermark value

uint8_t channel

Transfer start channel

uint8_t channelMask

enabled channel mask value, reference _sai_channel_mask

uint8_t endChannel

end channel number

uint8_t channelNums

Total enabled channel numbers

sai_protocol_t protocol

Which audio protocol used

bool isFrameSyncCompact

True means Frame sync length is configurable according to bitWidth, false means frame sync length is 64 times of bit clock.

struct _sai_master_clock

#include <fsl_sai.h> master clock configurations

Public Members

bool mclkOutputEnable

master clock output enable

uint32_t mclkHz
: target mclk frequency

uint32_t mclkSourceClkHz
: mclk source frequency

struct _sai_fifo
: *#include <fsl_sai.h>* sai fifo configurations

Public Members

bool fifoContinueOneError
: fifo continues when error occur

sai_fifo_combine_t fifoCombine
: fifo combine mode

sai_fifo_packing_t fifoPacking
: fifo packing mode

uint8_t fifoWatermark
: fifo watermark

struct _sai_bit_clock
: *#include <fsl_sai.h>* sai bit clock configurations

Public Members

bool bclkSrcSwap
: bit clock source swap

bool bclkInputDelay
: bit clock actually used by the transmitter is delayed by the pad output delay, this has effect of decreasing the data input setup time, but increasing the data output valid time .

sai_clock_polarity_t bclkPolarity
: bit clock polarity

sai_bclk_source_t bclkSource
: bit Clock source

struct _sai_frame_sync
: *#include <fsl_sai.h>* sai frame sync configurations

Public Members

uint8_t frameSyncWidth
: frame sync width in number of bit clocks

bool frameSyncEarly
: TRUE is frame sync assert one bit before the first bit of frame FALSE is frame sync assert with the first bit of the frame

bool frameSyncGenerateOnDemand
: internal frame sync is generated when FIFO waring flag is clear

sai_clock_polarity_t frameSyncPolarity
: frame sync polarity

struct __sai__serial__data
: *#include <fsl_sai.h>* sai serial data configurations

Public Members

sai_data_pin_state_t dataMode
: sai data pin state when slots masked or channel disabled

sai_data_order_t dataOrder
: configure whether the LSB or MSB is transmitted first

uint8_t dataWord0Length
: configure the number of bits in the first word in each frame

uint8_t dataWordNLength
: configure the number of bits in the each word in each frame, except the first word

uint8_t dataWordLength
: used to record the data length for dma transfer

uint8_t dataFirstBitShifted
: Configure the bit index for the first bit transmitted for each word in the frame

uint8_t dataWordNum
: configure the number of words in each frame

uint32_t dataMaskedWord
: configure whether the transmit word is masked

struct __sai__transceiver
: *#include <fsl_sai.h>* sai transceiver configurations

Public Members

sai_serial_data_t serialData
serial data configurations

sai_frame_sync_t frameSync
ws configurations

sai_bit_clock_t bitClock
bit clock configurations

sai_fifo_t fifo
fifo configurations

sai_master_slave_t masterSlave
transceiver is master or slave

sai_sync_mode_t syncMode
transceiver sync mode

uint8_t startChannel
Transfer start channel

uint8_t channelMask
enabled channel mask value, reference _sai_channel_mask

uint8_t endChannel
end channel number

uint8_t channelNums
Total enabled channel numbers

struct __sai__transfer
#include <fsl_sai.h> SAI transfer structure.

Public Members

uint8_t *data
Data start address to transfer.

size_t dataSize
Transfer size.

struct __sai__handle
#include <fsl_sai.h> SAI handle structure.

Public Members

I2S_Type *base
: base address

uint32_t state
: Transfer status

sai_transfer_callback_t callback
: Callback function called at transfer event

void *userData
: Callback parameter passed to callback function

uint8_t bitWidth
: Bit width for transfer, 8/16/24/32 bits

uint8_t channel
: Transfer start channel

uint8_t channelMask
: enabled channel mask value, refernece _sai_channel_mask

uint8_t endChannel
: end channel number

uint8_t channelNums
: Total enabled channel numbers

sai_transfer_t saiQueue[(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

uint8_t watermark
: Watermark value

2.58 SAI EDMA Driver

void SAI_TransferTxCreateHandleEDMA(I2S_Type *base, *sai_edma_handle_t* *handle, *sai_edma_callback_t* callback, void *userData, *edma_handle_t* *txDmaHandle)

Initializes the SAI eDMA handle.

This function initializes the SAI master DMA handle, which can be used for other SAI master transactional APIs. Usually, for a specified SAI instance, call this API once to get the initialized handle.

Parameters

- base – SAI base pointer.
- handle – SAI eDMA handle pointer.
- callback – Pointer to user callback function.
- userData – User parameter passed to the callback function.
- txDmaHandle – eDMA handle pointer, this handle shall be static allocated by users.

void SAI_TransferRxCreateHandleEDMA(I2S_Type *base, *sai_edma_handle_t* *handle, *sai_edma_callback_t* callback, void *userData, *edma_handle_t* *rxDmaHandle)

Initializes the SAI Rx eDMA handle.

This function initializes the SAI slave DMA handle, which can be used for other SAI master transactional APIs. Usually, for a specified SAI instance, call this API once to get the initialized handle.

Parameters

- base – SAI base pointer.
- handle – SAI eDMA handle pointer.
- callback – Pointer to user callback function.
- userData – User parameter passed to the callback function.
- rxDmaHandle – eDMA handle pointer, this handle shall be static allocated by users.

void SAI_TransferSetInterleaveType(*sai_edma_handle_t* *handle, *sai_edma_interleave_t* interleaveType)

Initializes the SAI interleave type.

This function initializes the SAI DMA handle member interleaveType, it shall be called only when application would like to use type kSAI_EDMAInterleavePerChannelBlock, since the default interleaveType is kSAI_EDMAInterleavePerChannelSample always

Parameters

- handle – SAI eDMA handle pointer.
- interleaveType – Multi channel interleave type.

void SAI_TransferTxSetConfigEDMA(I2S_Type *base, *sai_edma_handle_t* *handle, *sai_transceiver_t* *saiConfig)

Configures the SAI Tx.

Note: SAI eDMA supports data transfer in a multiple SAI channels if the FIFO Combine feature is supported. To activate the multi-channel transfer enable SAI channels by filling the channelMask of sai_transceiver_t with the corresponding values of _sai_channel_mask enum, enable the FIFO Combine mode by assigning kSAI_FifoCombineModeEnabledOnWrite to the fifoCombine member of sai_fifo_combine_t

which is a member of sai_transceiver_t. This is an example of multi-channel data transfer configuration step.

```
sai_transceiver_t config;
SAI_GetClassicI2SConfig(&config, kSAI_WordWidth16bits, kSAI_Stereo, kSAI_Channel0Mask|kSAI_
↪Channel1Mask);
config.fifo.fifoCombine = kSAI_FifoCombineModeEnabledOnWrite;
SAI_TransferTxSetConfigEDMA(I2S0, &edmaHandle, &config);
```

Parameters

- base – SAI base pointer.
- handle – SAI eDMA handle pointer.
- saiConfig – sai configurations.

void SAI_TransferRxSetConfigEDMA(I2S_Type *base, *sai_edma_handle_t* *handle, *sai_transceiver_t* *saiConfig)

Configures the SAI Rx.

Note: SAI eDMA supports data transfer in a multiple SAI channels if the FIFO Combine feature is supported. To activate the multi-channel transfer enable SAI channels by filling the channelMask of sai_transceiver_t with the corresponding values of _sai_channel_mask enum, enable the FIFO Combine mode by assigning kSAI_FifoCombineModeEnabledOnRead to the fifoCombine member of sai_fifo_combine_t which is a member of sai_transceiver_t. This is an example of multi-channel data transfer configuration step.

```
sai_transceiver_t config;
SAI_GetClassicI2SConfig(&config, kSAI_WordWidth16bits, kSAI_Stereo, kSAI_Channel0Mask|kSAI_
↪Channel1Mask);
config.fifo.fifoCombine = kSAI_FifoCombineModeEnabledOnRead;
SAI_TransferRxSetConfigEDMA(I2S0, &edmaHandle, &config);
```

Parameters

- base – SAI base pointer.
- handle – SAI eDMA handle pointer.
- saiConfig – sai configurations.

status_t SAI_TransferSendEDMA(I2S_Type *base, *sai_edma_handle_t* *handle, *sai_transfer_t* *xfer)

Performs a non-blocking SAI transfer using DMA.

This function support multi channel transfer,

a. for the sai IP support fifo combine mode, application should enable the fifo combine mode, no limitation on channel numbers

b. for the sai IP not support fifo combine mode, sai edma provide another solution which using EDMA modulo feature, but support 2 or 4 channels only.

Note: This interface returns immediately after the transfer initiates. Call

SAI_GetTransferStatus to poll the transfer status and check whether the SAI transfer is finished.

Parameters

- base – SAI base pointer.
- handle – SAI eDMA handle pointer.
- xfer – Pointer to the DMA transfer structure.

Return values

- kStatus_Success – Start a SAI eDMA send successfully.
- kStatus_InvalidArgument – The input argument is invalid.
- kStatus_TxBusy – SAI is busy sending data.

status_t SAI_TransferReceiveEDMA(I2S_Type *base, *sai_edma_handle_t* *handle, *sai_transfer_t* *xfer)

Performs a non-blocking SAI receive using eDMA.

This function support multi channel transfer,

a. for the sai IP support fifo combine mode, application should enable the fifo combine mode, no limitation on channel numbers
b. for the sai IP not support fifo combine mode, sai edma provide another solution which using EDMA modulo feature, but support 2 or 4 channels only.

Note: This interface returns immediately after the transfer initiates. Call the SAI_GetReceiveRemainingBytes to poll the transfer status and check whether the SAI transfer is finished.

Parameters

- base – SAI base pointer
- handle – SAI eDMA handle pointer.
- xfer – Pointer to DMA transfer structure.

Return values

- kStatus_Success – Start a SAI eDMA receive successfully.
- kStatus_InvalidArgument – The input argument is invalid.
- kStatus_RxBusy – SAI is busy receiving data.

status_t SAI_TransferSendLoopEDMA(I2S_Type *base, *sai_edma_handle_t* *handle, *sai_transfer_t* *xfer, uint32_t loopTransferCount)

Performs a non-blocking SAI loop transfer using eDMA.

Once the loop transfer start, application can use function SAI_TransferAbortSendEDMA to stop the loop transfer.

Note: This function support loop transfer only,such as A->B->...->A, application must be aware of that the more counts of the loop transfer, then more tcd memory required,

as the function use the tcd pool in sai_edma_handle_t, so application could redefine the SAI_XFER_QUEUE_SIZE to determine the proper TCD pool size. This function support one sai channel only.

Parameters

- base – SAI base pointer.
- handle – SAI eDMA handle pointer.
- xfer – Pointer to the DMA transfer structure, should be a array with elements counts >=1(loopTransferCount).
- loopTransferCount – the counts of xfer array.

Return values

- kStatus_Success – Start a SAI eDMA send successfully.
- kStatus_InvalidArgument – The input argument is invalid.

status_t SAI_TransferReceiveLoopEDMA(I2S_Type *base, *sai_edma_handle_t* *handle, *sai_transfer_t* *xfer, uint32_t loopTransferCount)

Performs a non-blocking SAI loop transfer using eDMA.

Once the loop transfer start, application can use function SAI_TransferAbortReceiveEDMA to stop the loop transfer.

Note: This function support loop transfer only,such as A->B->...->A, application must be aware of that the more counts of the loop transfer, then more tcd memory required, as the function use the tcd pool in sai_edma_handle_t, so application could redefine the SAI_XFER_QUEUE_SIZE to determine the proper TCD pool size. This function support one sai channel only.

Parameters

- base – SAI base pointer.
- handle – SAI eDMA handle pointer.
- xfer – Pointer to the DMA transfer structure, should be a array with elements counts >=1(loopTransferCount).
- loopTransferCount – the counts of xfer array.

Return values

- kStatus_Success – Start a SAI eDMA receive successfully.
- kStatus_InvalidArgument – The input argument is invalid.

void SAI_TransferTerminateSendEDMA(I2S_Type *base, *sai_edma_handle_t* *handle)

Terminate all SAI send.

This function will clear all transfer slots buffered in the sai queue. If users only want to abort the current transfer slot, please call SAI_TransferAbortSendEDMA.

Parameters

- base – SAI base pointer.
- handle – SAI eDMA handle pointer.

void SAI_TransferTerminateReceiveEDMA(I2S_Type *base, *sai_edma_handle_t* *handle)

Terminate all SAI receive.

This function will clear all transfer slots buffered in the sai queue. If users only want to abort the current transfer slot, please call SAI_TransferAbortReceiveEDMA.

Parameters

- base – SAI base pointer.
- handle – SAI eDMA handle pointer.

void SAI_TransferAbortSendEDMA(I2S_Type *base, *sai_edma_handle_t* *handle)

Aborts a SAI transfer using eDMA.

This function only aborts the current transfer slots, the other transfer slots' information still kept in the handler. If users want to terminate all transfer slots, just call SAI_TransferTerminateSendEDMA.

Parameters

- base – SAI base pointer.
- handle – SAI eDMA handle pointer.

void SAI_TransferAbortReceiveEDMA(I2S_Type *base, *sai_edma_handle_t* *handle)

Aborts a SAI receive using eDMA.

This function only aborts the current transfer slots, the other transfer slots' information still kept in the handler. If users want to terminate all transfer slots, just call SAI_TransferTerminateReceiveEDMA.

Parameters

- base – SAI base pointer
- handle – SAI eDMA handle pointer.

status_t SAI_TransferGetSendCountEDMA(I2S_Type *base, *sai_edma_handle_t* *handle, size_t *count)

Gets byte count sent by SAI.

Parameters

- base – SAI base pointer.
- handle – SAI eDMA handle pointer.
- count – Bytes count sent by SAI.

Return values

- kStatus_Success – Succeed get the transfer count.
- kStatus_NoTransferInProgress – There is no non-blocking transaction in progress.

status_t SAI_TransferGetReceiveCountEDMA(I2S_Type *base, *sai_edma_handle_t* *handle, size_t *count)

Gets byte count received by SAI.

Parameters

- base – SAI base pointer
- handle – SAI eDMA handle pointer.
- count – Bytes count received by SAI.

Return values

- kStatus_Success – Succeed get the transfer count.
- kStatus_NoTransferInProgress – There is no non-blocking transaction in progress.

uint32_t SAI_TransferGetValidTransferSlotsEDMA(I2S_Type *base, *sai_edma_handle_t* *handle)

Gets valid transfer slot.

This function can be used to query the valid transfer request slot that the application can submit. It should be called in the critical section, that means the application could call it in the corresponding callback function or disable IRQ before calling it in the application, otherwise, the returned value may not correct.

Parameters

- base – SAI base pointer
- handle – SAI eDMA handle pointer.

Return values

valid – slot count that application submit.

FSL_SAI_EDMA_DRIVER_VERSION

Version 2.7.4

enum _sai_edma_interleave

sai interleave type

Values:

enumerator kSAI_EDMAInterleavePerChannelSample

enumerator kSAI_EDMAInterleavePerChannelBlock

typedef struct *sai_edma_handle* sai_edma_handle_t

typedef void (*sai_edma_callback_t)(I2S_Type *base, *sai_edma_handle_t* *handle, *status_t* status, void *userData)

SAI eDMA transfer callback function for finish and error.

typedef enum *_sai_edma_interleave* sai_edma_interleave_t

sai interleave type

MCUX_SDK_SAI_EDMA_RX_ENABLE_INTERNAL

the SAI enable position When calling SAI_TransferReceiveEDMA

MCUX_SDK_SAI_EDMA_TX_ENABLE_INTERNAL

the SAI enable position When calling SAI_TransferSendEDMA

struct sai_edma_handle

#include <fsl_sai_edma.h> SAI DMA transfer handle, users should not touch the content of the handle.

Public Members

edma_handle_t *dmaHandle
DMA handler for SAI send

uint8_t nbytes
eDMA minor byte transfer count initially configured.

uint8_t bytesPerFrame
Bytes in a frame

uint8_t channelMask
Enabled channel mask value, reference _sai_channel_mask

uint8_t channelNums
total enabled channel nums

uint8_t channel
Which data channel

uint8_t count
The transfer data count in a DMA request

uint32_t state
Internal state for SAI eDMA transfer

sai_edma_callback_t callback
Callback for users while transfer finish or error occurs

void *userData
User callback parameter

uint8_t tcd[((4U) + 1U) * sizeof(*edma_tcd_t*)]
TCD pool for eDMA transfer.

sai_transfer_t saiQueue[(4U)]
Transfer queue storing queued transfer.

size_t transferSize[(4U)]
Data bytes need to transfer

sai_edma_interleave_t interleaveType
Transfer interleave type

volatile uint8_t queueUser
Index for user to queue transfer.

volatile uint8_t queueDriver
Index for driver to get the transfer data and size

2.59 SEMC: Smart External DRAM Controller Driver

void SEMC_GetDefaultConfig(*semc_config_t* *config)

Gets the SEMC default basic configuration structure.

The purpose of this API is to get the default SEMC configure structure for *SEMC_Init()*. User may use the initialized structure unchanged in *SEMC_Init()*, or modify some fields of the structure before calling *SEMC_Init()*. Example:

```
semc_config_t config;
SEMC_GetDefaultConfig(&config);
```

Parameters

- config – The SEMC configuration structure pointer.

void SEMC_Init(SEMC_Type *base, *semc_config_t* *configure)

Initializes SEMC. This function ungates the SEMC clock and initializes SEMC. This function must be called before calling any other SEMC driver functions.

Parameters

- base – SEMC peripheral base address.
- configure – The SEMC configuration structure pointer.

void SEMC_Deinit(SEMC_Type *base)

Deinitializes the SEMC module and gates the clock.

This function gates the SEMC clock. As a result, the SEMC module doesn't work after calling this function, for some IDE, calling this API may cause the next downloading operation failed. so, please call this API cautiously. Additional, users can using "#define FSL_SDK_DISABLE_DRIVER_CLOCK_CONTROL (1)" to disable the clock control operation in drivers.

Parameters

- base – SEMC peripheral base address.

status_t SEMC_ConfigureSDRAM(SEMC_Type *base, *semc_sdram_cs_t* cs, *semc_sdram_config_t* *config, uint32_t clkSrc_Hz)

Configures SDRAM controller in SEMC.

Parameters

- base – SEMC peripheral base address.
- cs – The chip selection.
- config – The sdram configuration.
- clkSrc_Hz – The SEMC clock frequency.

status_t SEMC_ConfigureNAND(SEMC_Type *base, *semc_nand_config_t* *config, uint32_t clkSrc_Hz)

Configures NAND controller in SEMC.

Parameters

- base – SEMC peripheral base address.
- config – The nand configuration.
- clkSrc_Hz – The SEMC clock frequency.

status_t SEMC_ConfigureNOR(SEMC_Type *base, *semc_nor_config_t* *config, uint32_t clkSrc_Hz)

Configures NOR controller in SEMC.

Parameters

- base – SEMC peripheral base address.
- config – The nor configuration.
- clkSrc_Hz – The SEMC clock frequency.

status_t SEMC_ConfigureSRAMWithChipSelection(SEMC_Type *base, *semc_sram_cs_t* cs, *semc_sram_config_t* *config, uint32_t clkSrc_Hz)

Configures SRAM controller in SEMC.

Parameters

- base – SEMC peripheral base address.
- cs – The chip selection.
- config – The sram configuration.
- clkSrc_Hz – The SEMC clock frequency.

status_t SEMC_ConfigureSRAM(SEMC_Type *base, *semc_sram_config_t* *config, uint32_t clkSrc_Hz)

Configures SRAM controller in SEMC.

Deprecated:

Do not use this function. It has been superceded by *SEMC_ConfigureSRAMWithChipSelection*.

Parameters

- base – SEMC peripheral base address.
- config – The sram configuration.
- clkSrc_Hz – The SEMC clock frequency.

status_t SEMC_ConfigureDBI(SEMC_Type *base, *semc_dbi_config_t* *config, uint32_t clkSrc_Hz)

Configures DBI controller in SEMC.

Parameters

- base – SEMC peripheral base address.
- config – The dbi configuration.
- clkSrc_Hz – The SEMC clock frequency.

static inline void SEMC_EnableInterrupts(SEMC_Type *base, uint32_t mask)

Enables the SEMC interrupt.

This function enables the SEMC interrupts according to the provided mask. The mask is a logical OR of enumeration members. See *semc_interrupt_enable_t*. For example, to enable the IP command done and error interrupt, do the following.

```
SEMC_EnableInterrupts(ENET, kSEMC_IPCmdDoneInterrupt | kSEMC_IPCmdErrInterrupt);
```

Parameters

- base – SEMC peripheral base address.

- mask – SEMC interrupts to enable. This is a logical OR of the enumeration :: semc_interrupt_enable_t.

static inline void SEMC_DisableInterrupts(SEMC_Type *base, uint32_t mask)

Disables the SEMC interrupt.

This function disables the SEMC interrupts according to the provided mask. The mask is a logical OR of enumeration members. See *semc_interrupt_enable_t*. For example, to disable the IP command done and error interrupt, do the following.

```
SEMC_DisableInterrupts(ENET, kSEMC_IPCmdDoneInterrupt | kSEMC_IPCmdErrInterrupt);
```

Parameters

- base – SEMC peripheral base address.
- mask – SEMC interrupts to disable. This is a logical OR of the enumeration :: semc_interrupt_enable_t.

static inline bool SEMC_GetStatusFlag(SEMC_Type *base)

Gets the SEMC status.

This function gets the SEMC interrupts event status. User can use the a logical OR of enumeration member as a mask. See *semc_interrupt_enable_t*.

Parameters

- base – SEMC peripheral base address.

Returns

status flag, use status flag in semc_interrupt_enable_t to get the related status.

static inline void SEMC_ClearStatusFlags(SEMC_Type *base, uint32_t mask)

Clears the SEMC status flag state.

The following status register flags can be cleared SEMC interrupt status.

Parameters

- base – SEMC base pointer
- mask – The status flag mask, a logical OR of enumeration member *semc_interrupt_enable_t*.

static inline bool SEMC_IsInIdle(SEMC_Type *base)

Check if SEMC is in idle.

Parameters

- base – SEMC peripheral base address.

Returns

True SEMC is in idle, false is not in idle.

status_t SEMC_SendIPCommand(SEMC_Type *base, *semc_mem_type_t* memType, uint32_t address, uint32_t command, uint32_t write, uint32_t *read)

SEMC IP command access.

Parameters

- base – SEMC peripheral base address.
- memType – SEMC memory type. refer to “semc_mem_type_t”
- address – SEMC device address.

- command – SEMC IP command. For NAND device, we should use the SEMC_BuildNandIPCommand to get the right nand command. For NOR/DBI device, take refer to "semc_ipcmd_nor_dbi_t". For SRAM device, take refer to "semc_ipcmd_sram_t". For SDRAM device, take refer to "semc_ipcmd_sdram_t".
- write – Data for write access.
- read – Data pointer for read data out.

static inline uint16_t SEMC_BuildNandIPCommand(uint8_t userCommand, *semc_ipcmd_nand_addrmode_t* addrMode, *semc_ipcmd_nand_cmdmode_t* cmdMode)

Build SEMC IP command for NAND.

This function build SEMC NAND IP command. The command is build of user command code, SEMC address mode and SEMC command mode.

Parameters

- userCommand – NAND device normal command.
- addrMode – NAND address mode. Refer to "semc_ipcmd_nand_addrmode_t".
- cmdMode – NAND command mode. Refer to "semc_ipcmd_nand_cmdmode_t".

static inline bool SEMC_IsNandReady(SEMC_Type *base)

Check if the NAND device is ready.

Parameters

- base – SEMC peripheral base address.

Returns

True NAND is ready, false NAND is not ready.

status_t SEMC_IPCommandNandWrite(SEMC_Type *base, uint32_t address, uint8_t *data, uint32_t size_bytes)

SEMC NAND device memory write through IP command.

Parameters

- base – SEMC peripheral base address.
- address – SEMC NAND device address.
- data – Data for write access.
- size_bytes – Data length.

status_t SEMC_IPCommandNandRead(SEMC_Type *base, uint32_t address, uint8_t *data, uint32_t size_bytes)

SEMC NAND device memory read through IP command.

Parameters

- base – SEMC peripheral base address.
- address – SEMC NAND device address.
- data – Data pointer for data read out.
- size_bytes – Data length.

status_t SEMC_IPCommandNorWrite(SEMC_Type *base, uint32_t address, uint8_t *data, uint32_t size_bytes)

SEMC NOR device memory write through IP command.

Parameters

- base – SEMC peripheral base address.
- address – SEMC NOR device address.
- data – Data for write access.
- size_bytes – Data length.

status_t SEMC_IPCommandNorRead(SEMC_Type *base, uint32_t address, uint8_t *data, uint32_t size_bytes)

SEMC NOR device memory read through IP command.

Parameters

- base – SEMC peripheral base address.
- address – SEMC NOR device address.
- data – Data pointer for data read out.
- size_bytes – Data length.

FSL_SEMC_DRIVER_VERSION

SEMC driver version.

SEMC status, _semc_status.

Values:

enumerator kStatus_SEMC_InvalidDeviceType

Invalid device type.

enumerator kStatus_SEMC_IpCommandExecutionError

IP command execution error.

enumerator kStatus_SEMC_AxiCommandExecutionError

AXI command execution error.

enumerator kStatus_SEMC_InvalidMemorySize

Invalid memory sie.

enumerator kStatus_SEMC_InvalidIpcmdDataSize

Invalid IP command data size.

enumerator kStatus_SEMC_InvalidAddressPortWidth

Invalid address port width.

enumerator kStatus_SEMC_InvalidDataPortWidth

Invalid data port width.

enumerator kStatus_SEMC_InvalidSwPinmuxSelection

Invalid SW pinmux selection.

enumerator kStatus_SEMC_InvalidBurstLength
Invalid burst length

enumerator kStatus_SEMC_InvalidColumnAddressBitWidth
Invalid column address bit width.

enumerator kStatus_SEMC_InvalidBaseAddress
Invalid base address.

enumerator kStatus_SEMC_InvalidTimerSetting
Invalid timer setting.

enum _semc_mem_type
SEMC memory device type.
Values:

enumerator kSEMC_MemType_SDRAM
SDRAM

enumerator kSEMC_MemType_SRAM
SRAM

enumerator kSEMC_MemType_NOR
NOR

enumerator kSEMC_MemType_NAND
NAND

enumerator kSEMC_MemType_8080
i.

enum _semc_waitready_polarity
SEMC WAIT/RDY polarity.
Values:

enumerator kSEMC_LowActive
Low active.

enumerator kSEMC_HighActive
High active.

enum _semc_sdram_cs
SEMC SDRAM Chip selection .
Values:

enumerator kSEMC_SDRAM_CS0
SEMC SDRAM CS0.

enumerator kSEMC_SDRAM_CS1
SEMC SDRAM CS1.

enumerator kSEMC_SDRAM_CS2
SEMC SDRAM CS2.

enumerator kSEMC_SDRAM_CS3
SEMC SDRAM CS3.

enum _semc_sram_cs
SEMC SRAM Chip selection .
Values:

enumerator kSEMC_SRAM_CS0
SEMC SRAM CS0.

enum _semc_nand_access_type
SEMC NAND device type.
Values:

enumerator kSEMC_NAND_ACCESS_BY_AXI
Access to NAND flash by AXI bus.

enumerator kSEMC_NAND_ACCESS_BY_IPCMD
Access to NAND flash by IP bus.

enum _semc_interrupt_enable
SEMC interrupts .
Values:

enumerator kSEMC_IPCmdDoneInterrupt
Ip command done interrupt.

enumerator kSEMC_IPCmdErrInterrupt
Ip command error interrupt.

enumerator kSEMC_AXICmdErrInterrupt
AXI command error interrupt.

enumerator kSEMC_AXIBusErrInterrupt
AXI bus error interrupt.

enum _semc_ipcmd_datasize
SEMC IP command data size in bytes.
Values:

enumerator kSEMC_IPcmdDataSize_1bytes
The IP command data size 1 byte.

enumerator kSEMC_IPcmdDataSize_2bytes
The IP command data size 2 byte.

enumerator kSEMC_IPcmdDataSize_3bytes
The IP command data size 3 byte.

enumerator kSEMC_IPcmdDataSize_4bytes
The IP command data size 4 byte.

enum _semc_refresh_time
SEMC auto-refresh timing.
Values:

enumerator kSEMC_RefreshThreeClocks
The refresh timing with three bus clocks.

enumerator kSEMC_RefreshSixClocks
The refresh timing with six bus clocks.

enumerator kSEMC_RefreshNineClocks
The refresh timing with nine bus clocks.

enum _semc_caslatency
CAS latency.
Values:

enumerator kSEMC_LatencyOne
Latency 1.

enumerator kSEMC_LatencyTwo
Latency 2.

enumerator kSEMC_LatencyThree
Latency 3.

enum _semc_sdram_column_bit_num
SEMC sdram column address bit number.
Values:

enumerator kSEMC_SdramColunm_12bit
12 bit.

enumerator kSEMC_SdramColunm_11bit
11 bit.

enumerator kSEMC_SdramColunm_10bit
10 bit.

enumerator kSEMC_SdramColunm_9bit
9 bit.

enumerator kSEMC_SdramColunm_8bit
8 bit.

enum _semc_sdram_burst_len
SEMC sdram burst length.

Values:

enumerator kSEMC_Sdram_BurstLen1
According to ERR050577, Auto-refresh command may possibly fail to be triggered during long time back-to-back write (or read) when SDRAM controller's burst length is greater than 1. Burst length 1

enum _semc_nand_column_bit_num
SEMC nand column address bit number.

Values:

enumerator kSEMC_NandColum_16bit
16 bit.

enumerator kSEMC_NandColum_15bit
15 bit.

enumerator kSEMC_NandColum_14bit
14 bit.

enumerator kSEMC_NandColum_13bit
13 bit.

enumerator kSEMC_NandColum_12bit
12 bit.

enumerator kSEMC_NandColum_11bit
11 bit.

enumerator kSEMC_NandColum_10bit
10 bit.

enumerator kSEMC_NandColum_9bit
9 bit.

enum _semc_nand_burst_len
SEMC nand burst length.

Values:

enumerator kSEMC_Nand_BurstLen1
Burst length 1

enumerator kSEMC_Nand_BurstLen2
Burst length 2

enumerator kSEMC_Nand_BurstLen4
Burst length 4

enumerator kSEMC_Nand_BurstLen8
Burst length 8

enumerator kSEMC_Nand_BurstLen16
Burst length 16

enumerator kSEMC_Nand_BurstLen32
Burst length 32

enumerator kSEMC_Nand_BurstLen64
Burst length 64

enum _semc_norsram_column_bit_num
SEMC nor/sram column address bit number.
Values:

enumerator kSEMC_NorColum_12bit
12 bit.

enumerator kSEMC_NorColum_11bit
11 bit.

enumerator kSEMC_NorColum_10bit
10 bit.

enumerator kSEMC_NorColum_9bit
9 bit.

enumerator kSEMC_NorColum_8bit
8 bit.

enumerator kSEMC_NorColum_7bit
7 bit.

enumerator kSEMC_NorColum_6bit
6 bit.

enumerator kSEMC_NorColum_5bit
5 bit.

enumerator kSEMC_NorColum_4bit
4 bit.

enumerator kSEMC_NorColum_3bit
3 bit.

enumerator kSEMC_NorColum_2bit
2 bit.

enum _semc_norsram_burst_len
SEMC nor/sram burst length.
Values:

enumerator kSEMC_Nor_BurstLen1
Burst length 1

enumerator kSEMC_Nor_BurstLen2
Burst length 2

enumerator kSEMC_Nor_BurstLen4
Burst length 4

enumerator kSEMC_Nor_BurstLen8
Burst length 8

enumerator kSEMC_Nor_BurstLen16
Burst length 16

enumerator kSEMC_Nor_BurstLen32
Burst length 32

enumerator kSEMC_Nor_BurstLen64
Burst length 64

enum _semc_dbi_column_bit_num
SEMC dbi column address bit number.
Values:

enumerator kSEMC_Dbi_Colum_12bit
12 bit.

enumerator kSEMC_Dbi_Colum_11bit
11 bit.

enumerator kSEMC_Dbi_Colum_10bit
10 bit.

enumerator kSEMC_Dbi_Colum_9bit
9 bit.

enumerator kSEMC_Dbi_Colum_8bit
8 bit.

enumerator kSEMC_Dbi_Colum_7bit
7 bit.

enumerator kSEMC_Dbi_Colum_6bit
6 bit.

enumerator kSEMC_Dbi_Colum_5bit
5 bit.

enumerator kSEMC_Dbi_Colum_4bit
4 bit.

enumerator kSEMC_Dbi_Colum_3bit
3 bit.

enumerator kSEMC_Dbi_Colum_2bit
2 bit.

enum _semc_dbi_burst_len
SEMC dbi burst length.
Values:

enumerator kSEMC_Dbi_BurstLen1
Burst length 1

enumerator kSEMC_Dbi_BurstLen2
Burst length 2

enumerator kSEMC_Dbi_Dbi_BurstLen4
Burst length 4

enumerator kSEMC_Dbi_BurstLen8
Burst length 8

enumerator kSEMC_Dbi_BurstLen16
Burst length 16

enumerator kSEMC_Dbi_BurstLen32
Burst length 32

enumerator kSEMC_Dbi_BurstLen64
Burst length 64

enum _semc_iomux_pin

SEMC IOMUXC.

Values:

enumerator kSEMC_MUXA8

MUX A8 pin.

enumerator kSEMC_MUXCSX0

MUX CSX0 pin

enumerator kSEMC_MUXCSX1

MUX CSX1 Pin.

enumerator kSEMC_MUXCSX2

MUX CSX2 Pin.

enumerator kSEMC_MUXCSX3

MUX CSX3 Pin.

enumerator kSEMC_MUXRDY

MUX RDY pin.

enum _semc_iomux_nora27_pin

SEMC NOR/PSRAM Address bit 27 A27.

Values:

enumerator kSEMC_MORA27_NONE

No NOR/SRAM A27 pin.

enumerator kSEMC_NORA27_MUXCSX3

MUX CSX3 Pin.

enumerator kSEMC_NORA27_MUXRDY

MUX RDY pin.

enum _semc_port_size

SEMC port size.

Values:

enumerator kSEMC_PortSize8Bit

8-Bit port size.

enumerator kSEMC_PortSize16Bit

16-Bit port size.

enum _semc_addr_mode

SEMC address mode.

Values:

enumerator kSEMC_AddrDataMux
SEMC address/data mux mode.

enumerator kSEMC_AdvAddrdataMux
Advanced address/data mux mode.

enumerator kSEMC_AddrDataNonMux
Address/data non-mux mode.

enum _semc_dqs_mode
SEMC DQS read strobe mode.
Values:

enumerator kSEMC_Loopbackinternal
Dummy read strobe loopbacked internally.

enumerator kSEMC_Loopbackdqspad
Dummy read strobe loopbacked from DQS pad.

enum _semc_adv_polarity
SEMC ADV signal active polarity.
Values:

enumerator kSEMC_AdvActiveLow
Adv active low.

enumerator kSEMC_AdvActiveHigh
Adv active high.

enum _semc_sync_mode
SEMC sync mode.
Values:

enumerator kSEMC_AsyncMode
Async mode.

enumerator kSEMC_SyncMode
Sync mode.

enum _semc_adv_level_control
SEMC ADV signal level control.
Values:

enumerator kSEMC_AdvHigh
Adv is high during address hold state.

enumerator kSEMC_AdvLow
Adv is low during address hold state.

enum _semc_rdy_polarity

SEMC RDY signal active polarity.

Values:

enumerator kSEMC_RdyActiveLow

Adv active low.

enumerator kSEMC_RdyActivehigh

Adv active low.

enum _semc_ipcmd_nand_addrmode

SEMC IP command for NAND: address mode.

Values:

enumerator kSEMC_NANDAM_ColumnRow

Address mode: column and row address(5Byte-CA0/CA1/RA0/RA1/RA2).

enumerator kSEMC_NANDAM_ColumnCA0

Address mode: column address only(1 Byte-CA0).

enumerator kSEMC_NANDAM_ColumnCA0CA1

Address mode: column address only(2 Byte-CA0/CA1).

enumerator kSEMC_NANDAM_RawRA0

Address mode: row address only(1 Byte-RA0).

enumerator kSEMC_NANDAM_RawRA0RA1

Address mode: row address only(2 Byte-RA0/RA1).

enumerator kSEMC_NANDAM_RawRA0RA1RA2

Address mode: row address only(3 Byte-RA0).

enum _semc_ipcmd_nand_cmdmode

SEMC IP command for NAND command mode.

Values:

enumerator kSEMC_NANDCM_Command

command.

enumerator kSEMC_NANDCM_CommandHold

Command hold.

enumerator kSEMC_NANDCM_CommandAddress

Command address.

enumerator kSEMC_NANDCM_CommandAddressHold

Command address hold.

enumerator kSEMC_NANDCM_CommandAddressRead
Command address read.

enumerator kSEMC_NANDCM_CommandAddressWrite
Command address write.

enumerator kSEMC_NANDCM_CommandRead
Command read.

enumerator kSEMC_NANDCM_CommandWrite
Command write.

enumerator kSEMC_NANDCM_Read
Read.

enumerator kSEMC_NANDCM_Write
Write.

enum _semc_nand_address_option
SEMC NAND address option.
Values:

enumerator kSEMC_NandAddrOption_5byte_CA2RA3
CA0+CA1+RA0+RA1+RA2

enumerator kSEMC_NandAddrOption_4byte_CA2RA2
CA0+CA1+RA0+RA1

enumerator kSEMC_NandAddrOption_3byte_CA2RA1
CA0+CA1+RA0

enumerator kSEMC_NandAddrOption_4byte_CA1RA3
CA0+RA0+RA1+RA2

enumerator kSEMC_NandAddrOption_3byte_CA1RA2
CA0+RA0+RA1

enumerator kSEMC_NandAddrOption_2byte_CA1RA1
CA0+RA0

enum _semc_ipcmd_nor_dbi
SEMC IP command for NOR.
Values:

enumerator kSEMC_NORDBICM_Read
NOR read.

enumerator kSEMC_NORDBICM_Write
NOR write.

enum _semc_ipcmd_sram
SEMC IP command for SRAM.
Values:

enumerator kSEMC_SRAMCM_ArrayRead
SRAM memory array read.

enumerator kSEMC_SRAMCM_ArrayWrite
SRAM memory array write.

enumerator kSEMC_SRAMCM_RegRead
SRAM memory register read.

enumerator kSEMC_SRAMCM_RegWrite
SRAM memory register write.

enum _semc_ipcmd_sdram
SEMC IP command for SDARM.
Values:

enumerator kSEMC_SDRAMCM_Read
SDRAM memory read.

enumerator kSEMC_SDRAMCM_Write
SDRAM memory write.

enumerator kSEMC_SDRAMCM_Modeset
SDRAM MODE SET.

enumerator kSEMC_SDRAMCM_Active
SDRAM active.

enumerator kSEMC_SDRAMCM_AutoRefresh
SDRAM auto-refresh.

enumerator kSEMC_SDRAMCM_SelfRefresh
SDRAM self-refresh.

enumerator kSEMC_SDRAMCM_Precharge
SDRAM precharge.

enumerator kSEMC_SDRAMCM_Prechargeall
SDRAM precharge all.

typedef enum *_semc_mem_type* semc_mem_type_t
SEMC memory device type.

typedef enum *_semc_waitready_polarity* semc_waitready_polarity_t
SEMC WAIT/RDY polarity.

typedef enum *_semc_sdram_cs* semc_sdram_cs_t
SEMC SDRAM Chip selection .

typedef enum *_semc_sram_cs* semc_sram_cs_t
SEMC SRAM Chip selection .

typedef enum *_semc_nand_access_type* semc_nand_access_type_t
SEMC NAND device type.

typedef enum *_semc_interrupt_enable* semc_interrupt_enable_t
SEMC interrupts .

typedef enum *_semc_ipcmd_datasize* semc_ipcmd_datasize_t
SEMC IP command data size in bytes.

typedef enum *_semc_refresh_time* semc_refresh_time_t
SEMC auto-refresh timing.

typedef enum *_semc_caslatency* semc_caslatency_t
CAS latency.

typedef enum *_semc_sdram_column_bit_num* semc_sdram_column_bit_num_t
SEMC sdram column address bit number.

typedef enum *_semc_sdram_burst_len* sem_sdram_burst_len_t
SEMC sdram burst length.

typedef enum *_semc_nand_column_bit_num* semc_nand_column_bit_num_t
SEMC nand column address bit number.

typedef enum *_semc_nand_burst_len* sem_nand_burst_len_t
SEMC nand burst length.

typedef enum *_semc_norsram_column_bit_num* semc_norsram_column_bit_num_t
SEMC nor/sram column address bit number.

typedef enum *_semc_norsram_burst_len* sem_norsram_burst_len_t
SEMC nor/sram burst length.

typedef enum *_semc_dbi_column_bit_num* semc_dbi_column_bit_num_t
SEMC dbi column address bit number.

typedef enum *_semc_dbi_burst_len* sem_dbi_burst_len_t
SEMC dbi burst length.

typedef enum *_semc_iomux_pin* semc_iomux_pin
SEMC IOMUXC.

typedef enum *_semc_iomux_nora27_pin* semc_iomux_nora27_pin
SEMC NOR/PSRAM Address bit 27 A27.

typedef enum *_semc_port_size* smec_port_size_t
SEMC port size.

typedef enum *_semc_addr_mode* semc_addr_mode_t
SEMC address mode.

typedef enum *_semc_dqs_mode* semc_dqs_mode_t
SEMC DQS read strobe mode.

typedef enum *_semc_adv_polarity* semc_adv_polarity_t
SEMC ADV signal active polarity.

typedef enum *_semc_sync_mode* semc_sync_mode_t
SEMC sync mode.

typedef enum *_semc_adv_level_control* semc_adv_level_control_t
SEMC ADV signal level control.

typedef enum *_semc_rdy_polarity* semc_rdy_polarity_t
SEMC RDY signal active polarity.

typedef enum *_semc_ipcmd_nand_addrmode* semc_ipcmd_nand_addrmode_t
SEMC IP command for NAND: address mode.

typedef enum *_semc_ipcmd_nand_cmdmode* semc_ipcmd_nand_cmdmode_t
SEMC IP command for NAND command mode.

typedef enum *_semc_nand_address_option* semc_nand_address_option_t
SEMC NAND address option.

typedef enum *_semc_ipcmd_nor_dbi* semc_ipcmd_nor_dbi_t
SEMC IP command for NOR.

typedef enum *_semc_ipcmd_sram* semc_ipcmd_sram_t
SEMC IP command for SRAM.

typedef enum *_semc_ipcmd_sdram* semc_ipcmd_sdram_t
SEMC IP command for SDARM.

typedef struct *_semc_sdram_config* semc_sdram_config_t

SEMC SDRAM configuration structure.

a. The memory size in the configuration is in the unit of KB. So memsize_kbytes should be set as 2^2, 2^3, 2^4 .etc which is base 2KB exponential function. Take refer to BR0~BR3 register in RM for details.

b. The prescalePeriod_N16Cycle is in unit of 16 clock cycle. It is a exception for prescaleTimer_n16cycle = 0, it means the prescaler timer period is 256 * 16 clock cycles. For precalerIf precalerTimer_n16cycle not equal to 0, The prescaler timer period is prescalePeriod_N16Cycle * 16 clock cycles. idleTimeout_NprescalePeriod, refreshUrgThreshold_NprescalePeriod, refreshPeriod_NprescalePeriod are similar to prescalePeriod_N16Cycle.

typedef struct *_semc_nand_timing_config* semc_nand_timing_config_t

SEMC NAND device timing configuration structure.

typedef struct *_semc_nand_config* semc_nand_config_t

SEMC NAND configuration structure.

typedef struct *_semc_nor_config* semc_nor_config_t

SEMC NOR configuration structure.

typedef struct *_semc_sram_config* semc_sram_config_t

SEMC SRAM configuration structure.

typedef struct *_semc_dbi_config* semc_dbi_config_t

SEMC DBI configuration structure.

typedef struct *_semc_queuea_weight_struct* semc_queuea_weight_struct_t

SEMC AXI queue a weight setting structure.

typedef union *_semc_queuea_weight* semc_queuea_weight_t

SEMC AXI queue a weight setting union.

typedef struct *_semc_queueb_weight_struct* semc_queueb_weight_struct_t

SEMC AXI queue b weight setting structure.

typedef union *_semc_queueb_weight* semc_queueb_weight_t

SEMC AXI queue b weight setting union.

typedef struct *_semc_axi_queueweight* semc_axi_queueweight_t

SEMC AXI queue weight setting.

typedef struct *_semc_config_t* semc_config_t

SEMC configuration structure.

busTimeoutCycles: when busTimeoutCycles is zero, the bus timeout cycle is 255*1024. otherwise the bus timeout cycles is busTimeoutCycles*1024. cmdTimeoutCycles: is used for command execution timeout cycles. it's similar to the busTimeoutCycles.

struct __semc__sdram__config

#include <fsl_semc.h> SEMC SDRAM configuration structure.

a. The memory size in the configuration is in the unit of KB. So memsize_kbytes should be set as 2^2, 2^3, 2^4 .etc which is base 2KB exponential function. Take refer to BR0~BR3 register in RM for details.

b. The prescalePeriod_N16Cycle is in unit of 16 clock cycle. It is a exception for prescaleTimer_n16cycle = 0, it means the prescaler timer period is 256 * 16 clock cycles. For precalerIf precalerTimer_n16cycle not equal to 0, The prescaler timer period is prescalePeriod_N16Cycle * 16 clock cycles. idleTimeout_NprescalePeriod, refreshUrgThreshold_NprescalePeriod, refreshPeriod_NprescalePeriod are similar to prescalePeriod_N16Cycle.

Public Members

semc_iomux_pin csxPinMux

CS pin mux. The kSEMC_MUXA8 is not valid in sdram pin mux setting.

uint32_t address

The base address.

uint32_t memsize__kbytes

The memory size in unit of kbytes.

smec_port_size_t portSize

Port size.

sem_sdram_burst_len_t burstLen

Burst length.

semc_sdram_column_bit_num_t columnAddrBitNum

Column address bit number.

semc_caslatency_t casLatency

CAS latency.

uint8_t tPrecharge2Act__Ns

Precharge to active wait time in unit of nanosecond.

uint8_t tAct2ReadWrite__Ns

Act to read/write wait time in unit of nanosecond.

uint8_t tRefreshRecovery__Ns

Refresh recovery time in unit of nanosecond.

uint8_t tWriteRecovery__Ns

write recovery time in unit of nanosecond.

uint8_t tCkeOff_Ns
: CKE off minimum time in unit of nanosecond.

uint8_t tAct2Prechage_Ns
: Active to precharge in unit of nanosecond.

uint8_t tSelfRefRecovery_Ns
: Self refresh recovery time in unit of nanosecond.

uint8_t tRefresh2Refresh_Ns
: Refresh to refresh wait time in unit of nanosecond.

uint8_t tAct2Act_Ns
: Active to active wait time in unit of nanosecond.

uint32_t tPrescalePeriod_Ns
: Prescaler timer period should not be larger than 256 * 16 * clock cycle.

uint32_t tIdleTimeout_Ns
: Idle timeout in unit of prescale time period.

uint32_t refreshPeriod_nsPerRow
: Refresh timer period like 64ms * 1000000/8192 .

uint32_t refreshUrgThreshold
: Refresh urgent threshold.

uint8_t refreshBurstLen
: Refresh burst length.

uint8_t delayChain
: Delay chain, which adds delays on DQS clock to compensate timings while DQS is faster than read data.

uint8_t autofreshTimes
: Auto Refresh cycles times.

struct _semc_nand_timing_config
: *#include <fsl_semc.h>* SEMC NAND device timing configuration structure.

Public Members

uint8_t tCeSetup_Ns
: CE setup time: tCS.

uint8_t tCeHold_Ns
: CE hold time: tCH.

uint8_t tCeInterval_Ns
CE interval time:tCEITV.

uint8_t tWeLow_Ns
WE low time: tWP.

uint8_t tWeHigh_Ns
WE high time: tWH.

uint8_t tReLow_Ns
RE low time: tRP.

uint8_t tReHigh_Ns
RE high time: tREH.

uint8_t tTurnAround_Ns
Turnaround time for async mode: tTA.

uint8_t tWehigh2Relow_Ns
WE# high to RE# wait time: tWHR.

uint8_t tRehigh2Welow_Ns
RE# high to WE# low wait time: tRHW.

uint8_t tAle2WriteStart_Ns
ALE to write start wait time: tADL.

uint8_t tReady2Relow_Ns
Ready to RE# low min wait time: tRR.

uint8_t tWehigh2Busy_Ns
WE# high to busy wait time: tWB.

struct _semc_nand_config
#include <fsl_semc.h> SEMC NAND configuration structure.

Public Members

semc_iomux_pin cePinMux
The CE pin mux setting. The kSEMC_MUXRDY is not valid for CE pin setting.

uint32_t axiAddress
The base address for AXI nand.

uint32_t axiMemsize_kbytes
The memory size in unit of kbytes for AXI nand.

uint32_t ipgAddress
: The base address for IPG nand .

uint32_t ipgMemsize_kbytes
: The memory size in unit of kbytes for IPG nand.

semc_rdy_polarity_t rdyactivePolarity
: Wait ready polarity.

bool edoModeEnabled
: EDO mode enabled.

semc_nand_column_bit_num_t columnAddrBitNum
: Column address bit number.

semc_nand_address_option_t arrayAddrOption
: Address option.

sem_nand_burst_len_t burstLen
: Burst length.

smec_port_size_t portSize
: Port size.

semc_nand_timing_config_t *timingConfig
: SEMC nand timing configuration.

struct __semc_nor_config
: *#include <fsl_semc.h>* SEMC NOR configuration structure.

Public Members

semc_iomux_pin cePinMux
: The CE# pin mux setting.

semc_iomux_nora27_pin addr27
: The Addr bit 27 pin mux setting.

uint32_t address
: The base address.

uint32_t memsize_kbytes
: The memory size in unit of kbytes.

uint8_t addrPortWidth
: The address port width.

semc_rdy_polarity_t rdyactivePolarity
Wait ready polarity.

semc_adv_polarity_t advActivePolarity
ADV# polarity.

semc_norsram_column_bit_num_t columnAddrBitNum
Column address bit number.

semc_addr_mode_t addrMode
Address mode.

sem_norsram_burst_len_t burstLen
Burst length.

smec_port_size_t portSize
Port size.

uint8_t tCeSetup_Ns
The CE setup time.

uint8_t tCeHold_Ns
The CE hold time.

uint8_t tCeInterval_Ns
CE interval minimum time.

uint8_t tAddrSetup_Ns
The address setup time.

uint8_t tAddrHold_Ns
The address hold time.

uint8_t tWeLow_Ns
WE low time for async mode.

uint8_t tWeHigh_Ns
WE high time for async mode.

uint8_t tReLow_Ns
RE low time for async mode.

uint8_t tReHigh_Ns
RE high time for async mode.

uint8_t tTurnAround_Ns
Turnaround time for async mode.

uint8_t tAddr2WriteHold_Ns
: Address to write data hold time for async mode.

uint8_t tWriteSetup_Ns
: Write data setup time for sync mode.

uint8_t tWriteHold_Ns
: Write hold time for sync mode.

uint8_t latencyCount
: Latency count for sync mode.

uint8_t readCycle
: Read cycle time for sync mode.

uint8_t delayChain
: Delay chain, which adds delays on DQS clock to compensate timings while DQS is faster than read data.

struct _semc_sram_config
: *#include <fsl_semc.h>* SEMC SRAM configuration structure.

Public Members

semc_iomux_pin cePinMux
: The CE# pin mux setting.

semc_iomux_nora27_pin addr27
: The Addr bit 27 pin mux setting.

uint32_t address
: The base address.

uint32_t memsize_kbytes
: The memory size in unit of kbytes.

uint8_t addrPortWidth
: The address port width.

semc_adv_polarity_t advActivePolarity
: ADV# polarity 1: active high, 0: active low.

semc_addr_mode_t addrMode
: Address mode.

sem_norsram_burst_len_t burstLen
: Burst length.

smec_port_size_t portSize
: Port size.

semc_sync_mode_t syncMode
: Sync mode.

bool waitEnable
: Wait enable.

uint8_t waitSample
: Wait sample.

semc_adv_level_control_t advLevelCtrl
: ADV# level control during address hold state, 1: low, 0: high.

uint32_t tCeSetup_Ns
: The CE setup time.

uint32_t tCeHold_Ns
: The CE hold time.

uint32_t tCeInterval_Ns
: CE interval minimum time.

uint32_t readHoldTime_Ns
: read hold time.

uint32_t tAddrSetup_Ns
: The address setup time.

uint32_t tAddrHold_Ns
: The address hold time.

uint32_t tWeLow_Ns
: WE low time for async mode.

uint32_t tWeHigh_Ns
: WE high time for async mode.

uint32_t tReLow_Ns
: RE low time for async mode.

uint32_t tReHigh_Ns
: RE high time for async mode.

uint32_t tTurnAround_Ns
: Turnaround time for async mode.

uint32_t tAddr2WriteHold_Ns
: Address to write data hold time for async mode.

uint32_t tWriteSetup_Ns
: Write data setup time for sync mode.

uint32_t tWriteHold_Ns
: Write hold time for sync mode.

uint8_t latencyCount
: Latency count for sync mode.

uint8_t readCycle
: Read cycle time for sync mode.

uint8_t delayChain
: Delay chain, which adds delays on DQS clock to compensate timings while DQS is faster than read data.

struct _semc_dbi_config
: *#include <fsl_semc.h>* SEMC DBI configuration structure.

Public Members

semc_iomux_pin csxPinMux
: The CE# pin mux.

uint32_t address
: The base address.

uint32_t memsize_kbytes
: The memory size in unit of 4kbytes.

semc_dbi_column_bit_num_t columnAddrBitNum
: Column address bit number.

sem_dbi_burst_len_t burstLen
: Burst length.

smec_port_size_t portSize
: Port size.

uint8_t tCsxSetup_Ns
: The CSX setup time.

uint8_t tCsxHold_Ns
: The CSX hold time.

uint8_t tWexLow_Ns
WEX low time.

uint8_t tWexHigh_Ns
WEX high time.

uint8_t tRdxLow_Ns
RDX low time.

uint8_t tRdxHigh_Ns
RDX high time.

uint8_t tCsxInterval_Ns
Write data setup time.

struct __semc_queuea_weight_struct
#include <fsl_semc.h> SEMC AXI queue a weight setting structure.

Public Members

uint32_t qos
weight of qos for queue 0 .

uint32_t aging
weight of aging for queue 0.

uint32_t slaveHitNoswitch
weight of read/write no switch for queue 0 .

uint32_t slaveHitSwitch
weight of read/write switch for queue 0.

union __semc_queuea_weight
#include <fsl_semc.h> SEMC AXI queue a weight setting union.

Public Members

semc_queuea_weight_struct_t queueaConfig
Structure configuration for queueA.

uint32_t queueaValue
Configuration value for queueA which could directly write to the reg.

struct __semc_queueb_weight_struct
#include <fsl_semc.h> SEMC AXI queue b weight setting structure.

Public Members

uint32_t qos
: weight of qos for queue 1.

uint32_t aging
: weight of aging for queue 1.

uint32_t weightPagehit
: weight of page hit for queue 1 only .

uint32_t slaveHitNoswitch
: weight of read/write no switch for queue 1.

uint32_t bankRotation
: weight of bank rotation for queue 1 only .

union __semc__queueb__weight
: *#include <fsl_semc.h>* SEMC AXI queue b weight setting union.

Public Members

semc_queueb_weight_struct_t queuebConfig
: Structure configuration for queueB.

uint32_t queuebValue
: Configuration value for queueB which could directly write to the reg.

struct __semc__axi__queueweight
: *#include <fsl_semc.h>* SEMC AXI queue weight setting.

Public Members

bool queueaEnable
: Enable queue a.

semc_queuea_weight_t queueaWeight
: Weight settings for queue a.

bool queuebEnable
: Enable queue b.

semc_queueb_weight_t queuebWeight
: Weight settings for queue b.

struct __semc__config__t

#include <fsl_semc.h> SEMC configuration structure.

busTimeoutCycles: when busTimeoutCycles is zero, the bus timeout cycle is 255*1024. otherwise the bus timeout cycles is busTimeoutCycles*1024. cmdTimeoutCycles: is used for command execution timeout cycles. it's similar to the busTimeoutCycles.

Public Members

semc_dqs_mode_t dqsMode

Dummy read strobe mode: use enum in "semc_dqs_mode_t".

uint8_t cmdTimeoutCycles

Command execution timeout cycles.

uint8_t busTimeoutCycles

Bus timeout cycles.

semc_axi_queueweight_t queueWeight

AXI queue weight.

2.60 SNVS: Secure Non-Volatile Storage

2.61 Secure Non-Volatile Storage High-Power

void SNVS_HP_Init(SNVS_Type *base)

Initialize the SNVS.

Note: This API should be called at the beginning of the application using the SNVS driver.

Parameters

- base – SNVS peripheral base address

void SNVS_HP_Deinit(SNVS_Type *base)

Deinitialize the SNVS.

Parameters

- base – SNVS peripheral base address

void SNVS_HP_RTC_Init(SNVS_Type *base, const *snvs_hp_rtc_config_t* *config)

Ungates the SNVS clock and configures the peripheral for basic operation.

Note: This API should be called at the beginning of the application using the SNVS driver.

Parameters

- base – SNVS peripheral base address

- config – Pointer to the user's SNVS configuration structure.

void SNVS_HP_RTC_Deinit(SNVS_Type *base)

Stops the RTC and SRTC timers.

Parameters

- base – SNVS peripheral base address

void SNVS_HP_RTC_GetDefaultConfig(*snvs_hp_rtc_config_t* *config)

Fills in the SNVS config struct with the default settings.

The default values are as follows.

```
config->rtccalenable = false;
config->rtccalvalue = 0U;
config->PIFreq = 0U;
```

Parameters

- config – Pointer to the user's SNVS configuration structure.

status_t SNVS_HP_RTC_SetDatetime(SNVS_Type *base, const *snvs_hp_rtc_datetime_t* *datetime)

Sets the SNVS RTC date and time according to the given time structure.

Parameters

- base – SNVS 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 SNVS RTC kStatus_InvalidArgument: Error because the datetime format is incorrect

void SNVS_HP_RTC_GetDatetime(SNVS_Type *base, *snvs_hp_rtc_datetime_t* *datetime)

Gets the SNVS RTC time and stores it in the given time structure.

Parameters

- base – SNVS peripheral base address
- datetime – Pointer to the structure where the date and time details are stored.

status_t SNVS_HP_RTC_SetAlarm(SNVS_Type *base, const *snvs_hp_rtc_datetime_t* *alarmTime)

Sets the SNVS RTC alarm time.

The function sets the RTC alarm. It also 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 – SNVS peripheral base address
- alarmTime – Pointer to the structure where the alarm time is stored.

Returns

kStatus_Success: success in setting the SNVS RTC alarm kStatus_InvalidArgument: Error because the alarm datetime format is incorrect kStatus_Fail: Error because the alarm time has already passed

void SNVS_HP_RTC_GetAlarm(SNVS_Type *base, *snvs_hp_rtc_datetime_t* *datetime)

Returns the SNVS RTC alarm time.

Parameters

- base – SNVS peripheral base address
- datetime – Pointer to the structure where the alarm date and time details are stored.

void SNVS_HP_RTC_TimeSynchronize(SNVS_Type *base)

The function synchronizes RTC counter value with SRTC.

Parameters

- base – SNVS peripheral base address

static inline void SNVS_HP_RTC_EnableInterrupts(SNVS_Type *base, uint32_t mask)

Enables the selected SNVS interrupts.

Parameters

- base – SNVS peripheral base address
- mask – The interrupts to enable. This is a logical OR of members of the enumeration :: _snvs_hp_interrupts_t

static inline void SNVS_HP_RTC_DisableInterrupts(SNVS_Type *base, uint32_t mask)

Disables the selected SNVS interrupts.

Parameters

- base – SNVS peripheral base address
- mask – The interrupts to disable. This is a logical OR of members of the enumeration :: _snvs_hp_interrupts_t

uint32_t SNVS_HP_RTC_GetEnabledInterrupts(SNVS_Type *base)

Gets the enabled SNVS interrupts.

Parameters

- base – SNVS peripheral base address

Returns

The enabled interrupts. This is the logical OR of members of the enumeration :: _snvs_hp_interrupts_t

uint32_t SNVS_HP_RTC_GetStatusFlags(SNVS_Type *base)

Gets the SNVS status flags.

Parameters

- base – SNVS peripheral base address

Returns

The status flags. This is the logical OR of members of the enumeration :: _snvs_hp_status_flags_t

static inline void SNVS_HP_RTC_ClearStatusFlags(SNVS_Type *base, uint32_t mask)

Clears the SNVS status flags.

Parameters

- base – SNVS peripheral base address
- mask – The status flags to clear. This is a logical OR of members of the enumeration :: _snvs_hp_status_flags_t

static inline void SNVS_HP_RTC_StartTimer(SNVS_Type *base)

Starts the SNVS RTC time counter.

Parameters

- base – SNVS peripheral base address

static inline void SNVS_HP_RTC_StopTimer(SNVS_Type *base)

Stops the SNVS RTC time counter.

Parameters

- base – SNVS peripheral base address

static inline void SNVS_HP_EnableHighAssuranceCounter(SNVS_Type *base, bool enable)

Enable or disable the High Assurance Counter (HAC)

Parameters

- base – SNVS peripheral base address
- enable – Pass true to enable, false to disable.

static inline void SNVS_HP_StartHighAssuranceCounter(SNVS_Type *base, bool start)

Start or stop the High Assurance Counter (HAC)

Parameters

- base – SNVS peripheral base address
- start – Pass true to start, false to stop.

static inline void SNVS_HP_SetHighAssuranceCounterInitialValue(SNVS_Type *base, uint32_t value)

Set the High Assurance Counter (HAC) initialize value.

Parameters

- base – SNVS peripheral base address
- value – The initial value to set.

static inline void SNVS_HP_LoadHighAssuranceCounter(SNVS_Type *base)

Load the High Assurance Counter (HAC)

This function loads the HAC initialize value to counter register.

Parameters

- base – SNVS peripheral base address

static inline uint32_t SNVS_HP_GetHighAssuranceCounter(SNVS_Type *base)

Get the current High Assurance Counter (HAC) value.

Parameters

- base – SNVS peripheral base address

Returns

HAC currnet value.

static inline void SNVS_HP_ClearHighAssuranceCounter(SNVS_Type *base)

Clear the High Assurance Counter (HAC)

This function can be called in a functional or soft fail state. When the HAC is enabled:

- If the HAC is cleared in the soft fail state, the SSM transitions to the hard fail state immediately;
- If the HAC is cleared in functional state, the SSM will transition to hard fail immediately after transitioning to soft fail.

Parameters

- base – SNVS peripheral base address

static inline void SNVS_HP_LockHighAssuranceCounter(SNVS_Type *base)

Lock the High Assurance Counter (HAC)

Once locked, the HAC initialize value could not be changed, the HAC enable status could not be changed. This could only be unlocked by system reset.

Parameters

- base – SNVS peripheral base address

FSL_SNVS_HP_DRIVER_VERSION

Version 2.3.2

enum _snvs_hp_interrupts

List of SNVS interrupts.

Values:

enumerator kSNVS_RTC_AlarmInterrupt

RTC time alarm

enumerator kSNVS_RTC_PeriodicInterrupt

RTC periodic interrupt

enum _snvs_hp_status_flags

List of SNVS flags.

Values:

enumerator kSNVS_RTC_AlarmInterruptFlag

RTC time alarm flag

enumerator kSNVS_RTC_PeriodicInterruptFlag

RTC periodic interrupt flag

enumerator kSNVS_ZMK_ZeroFlag

The ZMK is zero

enumerator kSNVS_OTPMK_ZeroFlag

The OTPMK is zero

enum _snvs_hp_sv_status_flags

List of SNVS security violation flags.

Values:

enumerator kSNVS_LP_ViolationFlag

Low Power section Security Violation

enumerator kSNVS_ZMK_EccFailFlag

Zeroizable Master Key Error Correcting Code Check Failure

enumerator kSNVS_LP_SoftwareViolationFlag
LP Software Security Violation

enumerator kSNVS_FatalSoftwareViolationFlag
Software Fatal Security Violation

enumerator kSNVS_SoftwareViolationFlag
Software Security Violation

enumerator kSNVS_Violation0Flag
Security Violation 0

enumerator kSNVS_Violation1Flag
Security Violation 1

enumerator kSNVS_Violation2Flag
Security Violation 2

enumerator kSNVS_Violation4Flag
Security Violation 4

enumerator kSNVS_Violation5Flag
Security Violation 5

enum _snvs_hp_ssm_state
List of SNVS Security State Machine State.
Values:

enumerator kSNVS_SSMInit
Init

enumerator kSNVS_SSMHardFail
Hard Fail

enumerator kSNVS_SSMSoftFail
Soft Fail

enumerator kSNVS_SSMInitInter
Init Intermediate (transition state between Init and Check)

enumerator kSNVS_SSMCheck
Check

enumerator kSNVS_SSMNonSecure
Non-Secure

enumerator kSNVS_SSMTrusted
Trusted

enumerator kSNVS_SSMSecure

Secure

typedef enum *_snvs_hp_interrupts* snvs_hp_interrupts_t

List of SNVS interrupts.

typedef enum *_snvs_hp_status_flags* snvs_hp_status_flags_t

List of SNVS flags.

typedef enum *_snvs_hp_sv_status_flags* snvs_hp_sv_status_flags_t

List of SNVS security violation flags.

typedef struct *_snvs_hp_rtc_datetime* snvs_hp_rtc_datetime_t

Structure is used to hold the date and time.

typedef struct *_snvs_hp_rtc_config* snvs_hp_rtc_config_t

SNVS config structure.

This structure holds the configuration settings for the SNVS peripheral. To initialize this structure to reasonable defaults, call the SNVS_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 *_snvs_hp_ssm_state* snvs_hp_ssm_state_t

List of SNVS Security State Machine State.

static inline void SNVS_HP_EnableMasterKeySelection(SNVS_Type *base, bool enable)

Enable or disable master key selection.

Parameters

- base – SNVS peripheral base address
- enable – Pass true to enable, false to disable.

static inline void SNVS_HP_ProgramZeroizableMasterKey(SNVS_Type *base)

Trigger to program Zeroizable Master Key.

Parameters

- base – SNVS peripheral base address

static inline void SNVS_HP_ChangeSSMState(SNVS_Type *base)

Trigger SSM State Transition.

Trigger state transition of the system security monitor (SSM). It results only the following transitions of the SSM:

- Check State -> Non-Secure (when Non-Secure Boot and not in Fab Configuration)
- Check State –> Trusted (when Secure Boot or in Fab Configuration)
- Trusted State –> Secure
- Secure State –> Trusted
- Soft Fail –> Non-Secure

Parameters

- base – SNVS peripheral base address

static inline void SNVS_HP_SetSoftwareFatalSecurityViolation(SNVS_Type *base)

Trigger Software Fatal Security Violation.

The result SSM state transition is:

- Check State -> Soft Fail
- Non-Secure State -> Soft Fail
- Trusted State -> Soft Fail
- Secure State -> Soft Fail

Parameters

- base – SNVS peripheral base address

static inline void SNVS_HP_SetSoftwareSecurityViolation(SNVS_Type *base)

Trigger Software Security Violation.

The result SSM state transition is:

- Check -> Non-Secure
- Trusted -> Soft Fail
- Secure -> Soft Fail

Parameters

- base – SNVS peripheral base address

static inline *snvs_hp_ssm_state_t* SNVS_HP_GetSSMState(SNVS_Type *base)

Get current SSM State.

Parameters

- base – SNVS peripheral base address

Returns

Current SSM state

static inline void SNVS_HP_ResetLP(SNVS_Type *base)

Reset the SNVS LP section.

Reset the LP section except SRTC and Time alarm.

Parameters

- base – SNVS peripheral base address

static inline uint32_t SNVS_HP_GetStatusFlags(SNVS_Type *base)

Get the SNVS HP status flags.

The flags are returned as the OR'ed value f the enumeration :: _snvs_hp_status_flags_t.

Parameters

- base – SNVS peripheral base address

Returns

The OR'ed value of status flags.

static inline void SNVS_HP_ClearStatusFlags(SNVS_Type *base, uint32_t mask)

Clear the SNVS HP status flags.

The flags to clear are passed in as the OR'ed value of the enumeration :: _snvs_hp_status_flags_t. Only these flags could be cleared using this API.

- *kSNVS_RTC_PeriodicInterruptFlag*

- *kSNVS_RTC_AlarmInterruptFlag*

Parameters

- base – SNVS peripheral base address
- mask – OR'ed value of the flags to clear.

static inline uint32_t SNVS_HP_GetSecurityViolationStatusFlags(SNVS_Type *base)

Get the SNVS HP security violation status flags.

The flags are returned as the OR'ed value of the enumeration :: _snvs_hp_sv_status_flags_t.

Parameters

- base – SNVS peripheral base address

Returns

The OR'ed value of security violation status flags.

static inline void SNVS_HP_ClearSecurityViolationStatusFlags(SNVS_Type *base, uint32_t mask)

Clear the SNVS HP security violation status flags.

The flags to clear are passed in as the OR'ed value of the enumeration :: _snvs_hp_sv_status_flags_t. Only these flags could be cleared using this API.

- *kSNVS_ZMK_EccFailFlag*
- *kSNVS_Violation0Flag*
- *kSNVS_Violation1Flag*
- *kSNVS_Violation2Flag*
- kSNVS_Violation3Flag
- *kSNVS_Violation4Flag*
- *kSNVS_Violation5Flag*

Parameters

- base – SNVS peripheral base address
- mask – OR'ed value of the flags to clear.

SNVS_HPSVSR_SV0_MASK

SNVS_HPSVSR_SV1_MASK

SNVS_HPSVSR_SV2_MASK

SNVS_HPSVSR_SV4_MASK

SNVS_HPSVSR_SV5_MASK

SNVS_MAKE_HP_SV_FLAG(x)

Macro to make security violation flag.

Macro help to make security violation flag kSNVS_Violation0Flag to kSNVS_Violation5Flag, For example, *SNVS_MAKE_HP_SV_FLAG(0)* is kSNVS_Violation0Flag.

struct _snvs_hp_rtc_datetime

#include <fsl_snvs_hp.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 _snvs_hp_rtc_config

#include <fsl_snvs_hp.h> SNVS config structure.

This structure holds the configuration settings for the SNVS peripheral. To initialize this structure to reasonable defaults, call the SNVS_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 rtcCalEnable

true: RTC calibration mechanism is enabled; false:No calibration is used

uint32_t rtcCalValue

Defines signed calibration value for nonsecure RTC; This is a 5-bit 2's complement value, range from -16 to +15

uint32_t periodicInterruptFreq

Defines frequency of the periodic interrupt; Range from 0 to 15

2.62 Secure Non-Volatile Storage Low-Power

void SNVS_LP_Init(SNVS_Type *base)

Ungates the SNVS clock and configures the peripheral for basic operation.

Note: This API should be called at the beginning of the application using the SNVS driver.

Parameters

- base – SNVS peripheral base address

void SNVS_LP_Deinit(SNVS_Type *base)

Deinit the SNVS LP section.

Parameters

- base – SNVS peripheral base address

status_t SNVS_LP_SRTC_SetDatetime(SNVS_Type *base, const *snvs_lp_srtc_datetime_t* *datetime)

Sets the SNVS SRTC date and time according to the given time structure.

Parameters

- base – SNVS 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 SNVS SRTC kStatus_InvalidArgument: Error because the datetime format is incorrect

void SNVS_LP_SRTC_GetDatetime(SNVS_Type *base, *snvs_lp_srtc_datetime_t* *datetime)

Gets the SNVS SRTC time and stores it in the given time structure.

Parameters

- base – SNVS peripheral base address
- datetime – Pointer to the structure where the date and time details are stored.

status_t SNVS_LP_SRTC_SetAlarm(SNVS_Type *base, const *snvs_lp_srtc_datetime_t* *alarmTime)

Sets the SNVS SRTC alarm time.

The function sets the SRTC alarm. It also 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. Please note, that SRTC alarm has limited resolution because only 32 most significant bits of SRTC counter are compared to SRTC Alarm register. If the alarm time is beyond SRTC resolution, the function does not set the alarm and returns an error.

Parameters

- base – SNVS peripheral base address
- alarmTime – Pointer to the structure where the alarm time is stored.

Returns

kStatus_Success: success in setting the SNVS SRTC alarm kStatus_InvalidArgument: Error because the alarm datetime format is incorrect kStatus_Fail: Error because the alarm time has already passed or is beyond resolution

void SNVS_LP_SRTC_GetAlarm(SNVS_Type *base, *snvs_lp_srtc_datetime_t* *datetime)

Returns the SNVS SRTC alarm time.

Parameters

- base – SNVS peripheral base address
- datetime – Pointer to the structure where the alarm date and time details are stored.

static inline void SNVS_LP_SRTC_EnableInterrupts(SNVS_Type *base, uint32_t mask)

Enables the selected SNVS interrupts.

Parameters

- base – SNVS peripheral base address
- mask – The interrupts to enable. This is a logical OR of members of the enumeration :: _snvs_lp_srtc_interrupts

static inline void SNVS_LP_SRTC_DisableInterrupts(SNVS_Type *base, uint32_t mask)

Disables the selected SNVS interrupts.

Parameters

- base – SNVS peripheral base address
- mask – The interrupts to enable. This is a logical OR of members of the enumeration :: _snvs_lp_srtc_interrupts

uint32_t SNVS_LP_SRTC_GetEnabledInterrupts(SNVS_Type *base)

Gets the enabled SNVS interrupts.

Parameters

- base – SNVS peripheral base address

Returns

The enabled interrupts. This is the logical OR of members of the enumeration :: _snvs_lp_srtc_interrupts

uint32_t SNVS_LP_SRTC_GetStatusFlags(SNVS_Type *base)

Gets the SNVS status flags.

Parameters

- base – SNVS peripheral base address

Returns

The status flags. This is the logical OR of members of the enumeration :: _snvs_lp_srtc_status_flags

static inline void SNVS_LP_SRTC_ClearStatusFlags(SNVS_Type *base, uint32_t mask)

Clears the SNVS status flags.

Parameters

- base – SNVS peripheral base address
- mask – The status flags to clear. This is a logical OR of members of the enumeration :: _snvs_lp_srtc_status_flags

static inline void SNVS_LP_SRTC_StartTimer(SNVS_Type *base)

Starts the SNVS SRTC time counter.

Parameters

- base – SNVS peripheral base address

static inline void SNVS_LP_SRTC_StopTimer(SNVS_Type *base)

Stops the SNVS SRTC time counter.

Parameters

- base – SNVS peripheral base address

void SNVS_LP_EnablePassiveTamper(SNVS_Type *base, *snvs_lp_external_tamper_t* pin, *snvs_lp_passive_tamper_t* config)

Enables the specified SNVS external tamper.

Parameters

- base – SNVS peripheral base address
- pin – SNVS external tamper pin
- config – Configuration structure of external passive tamper

status_t SNVS_LP_EnableTxActiveTamper(SNVS_Type *base, *snvs_lp_active_tx_tamper_t* pin, *tamper_active_tx_config_t* config)

Enable active tamper tx external pad.

Parameters

- base – SNVS peripheral base address
- pin – SNVS active tamper pin
- config – Configuration structure of external active tamper

status_t SNVS_LP_EnableRxActiveTamper(SNVS_Type *base, *snvs_lp_external_tamper_t* rx, *tamper_active_rx_config_t* config)

Enable active tamper rx external pad.

Parameters

- base – SNVS peripheral base address
- rx – SNVS external RX tamper pin
- config – SNVS RX tamper config structure

status_t SNVS_LP_SetVoltageTamper(SNVS_Type *base, bool enable)

Sets voltage tamper detect.

Parameters

- base – SNVS peripheral base address
- enable – True if enable false if disable

status_t SNVS_LP_SetTemperatureTamper(SNVS_Type *base, bool enable)

Sets temperature tamper detect.

Parameters

- base – SNVS peripheral base address
- enable – True if enable false if disable

status_t SNVS_LP_SetClockTamper(SNVS_Type *base, bool enable)

Sets clock tamper detect.

Parameters

- base – SNVS peripheral base address
- enable – True if enable false if disable

snvs_lp_external_tamper_status_t SNVS_LP_CheckVoltageTamper(SNVS_Type *base)

brief Check voltage tamper

param base SNVS peripheral base address

snvs_lp_external_tamper_status_t SNVS_LP_CheckTemperatureTamper(SNVS_Type *base)

Check temperature tamper.

Parameters

- base – SNVS peripheral base address

snvs_lp_external_tamper_status_t SNVS_LP_CheckClockTamper(SNVS_Type *base)

brief Check clock tamper

param base SNVS peripheral base address

void SNVS_LP_TamperPinTx_GetDefaultConfig(*tamper_active_tx_config_t* *config)

Fills in the SNVS tamper pin config struct with the default settings.

The default values are as follows. code config->clock = kSNVS_ActiveTamper16HZ; config->seed = 0U; config->polynomial = 0U; endcode

Parameters

- config – Pointer to the user's SNVS configuration structure.

void SNVS_LP_TamperPinRx_GetDefaultConfig(*tamper_active_rx_config_t* *config)

brief Fills in the SNVS tamper pin config struct with the default settings.

The default values are as follows. code config->filterenable = 0U; config->filter = 0U; config->tx = kSNVS_ActiveTamper1; endcode param config Pointer to the user's SNVS configuration structure.

void SNVS_LP_PassiveTamperPin_GetDefaultConfig(*snvs_lp_passive_tamper_t* *config)

Fills in the SNVS tamper pin config struct with the default settings.

The default values are as follows. code config->polarity = 0U; config->filterenable = 0U; if available on SoC config->filter = 0U; if available on SoC endcode

Parameters

- config – Pointer to the user's SNVS configuration structure.

void SNVS_LP_DisableExternalTamper(SNVS_Type *base, *snvs_lp_external_tamper_t* pin)

Disables the specified SNVS external tamper.

Parameters

- base – SNVS peripheral base address
- pin – SNVS external tamper pin

void SNVS_LP_DisableAllExternalTamper(SNVS_Type *base)

Disable all external tamper.

Parameters

- base – SNVS peripheral base address

snvs_lp_external_tamper_status_t SNVS_LP_GetExternalTamperStatus(SNVS_Type *base, *snvs_lp_external_tamper_t* pin)

Returns status of the specified external tamper.

Parameters

- base – SNVS peripheral base address

- pin – SNVS external tamper pin

Returns
The status flag. This is the enumeration :: _snvs_lp_external_tamper_status

void SNVS_LP_ClearExternalTamperStatus(SNVS_Type *base, *snvs_lp_external_tamper_t* pin)

Clears status of the specified external tamper.

Parameters

- base – SNVS peripheral base address
- pin – SNVS external tamper pin

void SNVS_LP_ClearAllExternalTamperStatus(SNVS_Type *base)

Clears status of the all external tamper.

Parameters

- base – SNVS peripheral base address

static inline void SNVS_LP_EnableMonotonicCounter(SNVS_Type *base, bool enable)

Enable or disable the Monotonic Counter.

Parameters

- base – SNVS peripheral base address
- enable – Pass true to enable, false to disable.

uint64_t SNVS_LP_GetMonotonicCounter(SNVS_Type *base)

Get the current Monotonic Counter.

Parameters

- base – SNVS peripheral base address

Returns
Current Monotonic Counter value.

static inline void SNVS_LP_IncreaseMonotonicCounter(SNVS_Type *base)

Increase the Monotonic Counter.

Increase the Monotonic Counter by 1.

Parameters

- base – SNVS peripheral base address

void SNVS_LP_WriteZeroizableMasterKey(SNVS_Type *base, uint32_t ZMKey[8U])

Write Zeroizable Master Key (ZMK) to the SNVS registers.

Parameters

- base – SNVS peripheral base address
- ZMKey – The ZMK write to the SNVS register.

static inline void SNVS_LP_SetZeroizableMasterKeyValid(SNVS_Type *base, bool valid)

Set Zeroizable Master Key valid.

This API could only be called when using software programming mode. After writing ZMK using *SNVS_LP_WriteZeroizableMasterKey*, call this API to make the ZMK valid.

Parameters

- base – SNVS peripheral base address
- valid – Pass true to set valid, false to set invalid.

static inline bool SNVS_LP_GetZeroizableMasterKeyValid(SNVS_Type *base)

Get Zeroizable Master Key valid status.

In hardware programming mode, call this API to check whether the ZMK is valid.

Parameters

- base – SNVS peripheral base address

Returns

true if valid, false if invalid.

static inline void SNVS_LP_SetZeroizableMasterKeyProgramMode(SNVS_Type *base, *snvs_lp_zmk_program_mode_t* mode)

Set Zeroizable Master Key programming mode.

Parameters

- base – SNVS peripheral base address
- mode – ZMK programming mode.

static inline void SNVS_LP_EnableZeroizableMasterKeyECC(SNVS_Type *base, bool enable)

Enable or disable Zeroizable Master Key ECC.

Parameters

- base – SNVS peripheral base address
- enable – Pass true to enable, false to disable.

static inline void SNVS_LP_SetMasterKeyMode(SNVS_Type *base, *snvs_lp_master_key_mode_t* mode)

Set SNVS Master Key mode.

Note: When *kSNVS_ZMK* or *kSNVS_CMK* used, the SNVS_HP must be configured to enable the master key selection.

Parameters

- base – SNVS peripheral base address
- mode – Master Key mode.

FSL_SNVS_LP_DRIVER_VERSION

Version 2.4.6

enum _snvs_lp_srtc_interrupts

List of SNVS_LP interrupts.

Values:

enumerator kSNVS_SRTC_AlarmInterrupt

SRTC time alarm.

enum _snvs_lp_srtc_status_flags

List of SNVS_LP flags.

Values:

enumerator kSNVS_SRTC_AlarmInterruptFlag
SRTC time alarm flag

enum _snvs_lp_external_tamper
List of SNVS_LP external tampers.
Values:

enumerator kSNVS_ExternalTamper1

enum _snvs_lp_active_tamper
List of SNVS_LP active tampers.
Values:

enumerator kSNVS_ActiveTamper1

enumerator kSNVS_ActiveTamper2

enumerator kSNVS_ActiveTamper3

enumerator kSNVS_ActiveTamper4

enumerator kSNVS_ActiveTamper5

enum _snvs_lp_active_clock
List of SNVS_LP external tampers.
Values:

enumerator kSNVS_ActiveTamper16HZ

enumerator kSNVS_ActiveTamper8HZ

enumerator kSNVS_ActiveTamper4HZ

enumerator kSNVS_ActiveTamper2HZ

enum _snvs_lp_external_tamper_status
List of SNVS_LP external tampers status.
Values:

enumerator kSNVS_TamperNotDetected

enumerator kSNVS_TamperDetected

enum _snvs_lp_external_tamper_polarity
SNVS_LP external tamper polarity.
Values:

enumerator kSNVS_ExternalTamperActiveLow

enumerator kSNVS_ExternalTamperActiveHigh

enum __snvs_lp_zmk_program_mode

SNVS_LP Zeroizable Master Key programming mode.

Values:

enumerator kSNVS_ZMKSoftwareProgram

Software programming mode.

enumerator kSNVS_ZMKHardwareProgram

Hardware programming mode.

enum __snvs_lp_master_key_mode

SNVS_LP Master Key mode.

Values:

enumerator kSNVS_OTPMK

One Time Programmable Master Key.

enumerator kSNVS_ZMK

Zeroizable Master Key.

enumerator kSNVS_CMK

Combined Master Key, it is XOR of OPTMK and ZMK.

typedef enum *_snvs_lp_srtc_interrupts* snvs_lp_srtc_interrupts_t

List of SNVS_LP interrupts.

typedef enum *_snvs_lp_srtc_status_flags* snvs_lp_srtc_status_flags_t

List of SNVS_LP flags.

typedef enum *_snvs_lp_external_tamper* snvs_lp_external_tamper_t

List of SNVS_LP external tampers.

typedef enum *_snvs_lp_active_tamper* snvs_lp_active_tx_tamper_t

List of SNVS_LP active tampers.

typedef enum *_snvs_lp_active_clock* snvs_lp_active_clock_t

List of SNVS_LP external tampers.

typedef enum *_snvs_lp_external_tamper_status* snvs_lp_external_tamper_status_t

List of SNVS_LP external tampers status.

typedef enum *_snvs_lp_external_tamper_polarity* snvs_lp_external_tamper_polarity_t

SNVS_LP external tamper polarity.

typedef struct *_snvs_lp_srtc_datetime* snvs_lp_srtc_datetime_t

Structure is used to hold the date and time.

typedef struct *_snvs_lp_srtc_config* snvs_lp_srtc_config_t

SNVS_LP config structure.

This structure holds the configuration settings for the SNVS_LP peripheral. To initialize this structure to reasonable defaults, call the SNVS_LP_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 *_snvs_lp_zmk_program_mode* snvs_lp_zmk_program_mode_t

SNVS_LP Zeroizable Master Key programming mode.

typedef enum *_snvs_lp_master_key_mode* snvs_lp_master_key_mode_t

SNVS_LP Master Key mode.

void SNVS_LP_SRTC_Init(SNVS_Type *base, const *snvs_lp_srtc_config_t* *config)

Ungates the SNVS clock and configures the peripheral for basic operation.

Note: This API should be called at the beginning of the application using the SNVS driver.

Parameters

- base – SNVS peripheral base address
- config – Pointer to the user's SNVS configuration structure.

void SNVS_LP_SRTC_Deinit(SNVS_Type *base)

Stops the SRTC timer.

Parameters

- base – SNVS peripheral base address

void SNVS_LP_SRTC_GetDefaultConfig(*snvs_lp_srtc_config_t* *config)

Fills in the SNVS_LP config struct with the default settings.

The default values are as follows.

```
config->srtccalenable = false;
config->srtccalvalue = 0U;
```

Parameters

- config – Pointer to the user's SNVS configuration structure.

SNVS_ZMK_REG_COUNT

Define of SNVS_LP Zeroizable Master Key registers.

SNVS_LP_MAX_TAMPER

Define of SNVS_LP Max possible tamper.

struct tamper_active_tx_config_t

#include <fsl_snvs_lp.h> Structure is used to configure SNVS LP active TX tamper pins.

struct tamper_active_rx_config_t

#include <fsl_snvs_lp.h> Structure is used to configure SNVS LP active RX tamper pins.

struct snvs_lp_passive_tamper_t

#include <fsl_snvs_lp.h> Structure is used to configure SNVS LP passive tamper pins.

struct _snvs_lp_srtc_datetime

#include <fsl_snvs_lp.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 _snvs_lp_srtc_config

#include <fsl_snvs_lp.h> SNVS_LP config structure.

This structure holds the configuration settings for the SNVS_LP peripheral. To initialize this structure to reasonable defaults, call the SNVS_LP_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 srtcCalEnable

true: SRTC calibration mechanism is enabled; false: No calibration is used

uint32_t srtcCalValue

Defines signed calibration value for SRTC; This is a 5-bit 2's complement value, range from -16 to +15

2.63 SPDIF: Sony/Philips Digital Interface

void SPDIF_Init(SPDIF_Type *base, const *spdif_config_t* *config)

Initializes the SPDIF peripheral.

Ungates the SPDIF clock, resets the module, and configures SPDIF with a configuration structure. The configuration structure can be custom filled or set with default values by *SPDIF_GetDefaultConfig()*.

Note: This API should be called at the beginning of the application to use the SPDIF driver. Otherwise, accessing the SPDIF module can cause a hard fault because the clock is not enabled.

Parameters

- base – SPDIF base pointer
- config – SPDIF configuration structure.

void SPDIF_GetDefaultConfig(*spdif_config_t* *config)

Sets the SPDIF configuration structure to default values.

This API initializes the configuration structure for use in SPDIF_Init. The initialized structure can remain unchanged in SPDIF_Init, or it can be modified before calling SPDIF_Init. This is an example.

```
spdif_config_t config;
SPDIF_GetDefaultConfig(&config);
```

Parameters

- config – pointer to master configuration structure

void SPDIF_Deinit(SPDIF_Type *base)

De-initializes the SPDIF peripheral.

This API gates the SPDIF clock. The SPDIF module can't operate unless SPDIF_Init is called to enable the clock.

Parameters

- base – SPDIF base pointer

uint32_t SPDIF_GetInstance(SPDIF_Type *base)

Get the instance number for SPDIF.

Parameters

- base – SPDIF base pointer.

static inline void SPDIF_TxFIFOReset(SPDIF_Type *base)

Resets the SPDIF Tx.

This function makes Tx FIFO in reset mode.

Parameters

- base – SPDIF base pointer

static inline void SPDIF_RxFIFOReset(SPDIF_Type *base)

Resets the SPDIF Rx.

This function enables the software reset and FIFO reset of SPDIF Rx. After reset, clear the reset bit.

Parameters

- base – SPDIF base pointer

void SPDIF_TxEnable(SPDIF_Type *base, bool enable)

Enables/disables the SPDIF Tx.

Parameters

- base – SPDIF base pointer
- enable – True means enable SPDIF Tx, false means disable.

static inline void SPDIF_RxEnable(SPDIF_Type *base, bool enable)

Enables/disables the SPDIF Rx.

Parameters

- base – SPDIF base pointer
- enable – True means enable SPDIF Rx, false means disable.

static inline uint32_t SPDIF_GetStatusFlag(SPDIF_Type *base)

Gets the SPDIF status flag state.

Parameters

- base – SPDIF base pointer

Returns

SPDIF status flag value. Use the _spdif_interrupt_enable_t to get the status value needed.

static inline void SPDIF_ClearStatusFlags(SPDIF_Type *base, uint32_t mask)

Clears the SPDIF status flag state.

Parameters

- base – SPDIF base pointer
- mask – State mask. It can be a combination of the _spdif_interrupt_enable_t member. Notice these members cannot be included, as these flags cannot be cleared by writing 1 to these bits:
  - kSPDIF_UChannelReceiveRegisterFull
  - kSPDIF_QChannelReceiveRegisterFull
  - kSPDIF_TxFIFOEmpty
  - kSPDIF_RxFIFOFull

static inline void SPDIF_EnableInterrupts(SPDIF_Type *base, uint32_t mask)

Enables the SPDIF Tx interrupt requests.

Parameters

- base – SPDIF base pointer
- mask – interrupt source The parameter can be a combination of the following sources if defined.
  - kSPDIF_WordStartInterruptEnable
  - kSPDIF_SyncErrorInterruptEnable

- kSPDIF_FIFOWarningInterruptEnable
- kSPDIF_FIFORequestInterruptEnable
- kSPDIF_FIFOErrorInterruptEnable

static inline void SPDIF_DisableInterrupts(SPDIF_Type *base, uint32_t mask)

Disables the SPDIF Tx interrupt requests.

Parameters

- base – SPDIF base pointer
- mask – interrupt source The parameter can be a combination of the following sources if defined.
  - kSPDIF_WordStartInterruptEnable
  - kSPDIF_SyncErrorInterruptEnable
  - kSPDIF_FIFOWarningInterruptEnable
  - kSPDIF_FIFORequestInterruptEnable
  - kSPDIF_FIFOErrorInterruptEnable

static inline void SPDIF_EnableDMA(SPDIF_Type *base, uint32_t mask, bool enable)

Enables/disables the SPDIF DMA requests.

Parameters

- base – SPDIF base pointer
- mask – SPDIF DMA enable mask, The parameter can be a combination of the following sources if defined
  - kSPDIF_RxDMAEnable
  - kSPDIF_TxDMAEnable
- enable – True means enable DMA, false means disable DMA.

static inline uint32_t SPDIF_TxGetLeftDataRegisterAddress(SPDIF_Type *base)

Gets the SPDIF Tx left data register address.

This API is used to provide a transfer address for the SPDIF DMA transfer configuration.

Parameters

- base – SPDIF base pointer.

Returns

data register address.

static inline uint32_t SPDIF_TxGetRightDataRegisterAddress(SPDIF_Type *base)

Gets the SPDIF Tx right data register address.

This API is used to provide a transfer address for the SPDIF DMA transfer configuration.

Parameters

- base – SPDIF base pointer.

Returns

data register address.

static inline uint32_t SPDIF_RxGetLeftDataRegisterAddress(SPDIF_Type *base)

Gets the SPDIF Rx left data register address.

This API is used to provide a transfer address for the SPDIF DMA transfer configuration.

Parameters

- base – SPDIF base pointer.

Returns
data register address.

static inline uint32_t SPDIF_RxGetRightDataRegisterAddress(SPDIF_Type *base)

Gets the SPDIF Rx right data register address.

This API is used to provide a transfer address for the SPDIF DMA transfer configuration.

Parameters

- base – SPDIF base pointer.

Returns
data register address.

void SPDIF_TxSetSampleRate(SPDIF_Type *base, uint32_t sampleRate_Hz, uint32_t sourceClockFreq_Hz)

Configures the SPDIF Tx sample rate.

The audio format can be changed at run-time. This function configures the sample rate.

Parameters

- base – SPDIF base pointer.
- sampleRate_Hz – SPDIF sample rate frequency in Hz.
- sourceClockFreq_Hz – SPDIF tx clock source frequency in Hz.

uint32_t SPDIF_GetRxSampleRate(SPDIF_Type *base, uint32_t clockSourceFreq_Hz)

Configures the SPDIF Rx audio format.

The audio format can be changed at run-time. This function configures the sample rate and audio data format to be transferred.

Parameters

- base – SPDIF base pointer.
- clockSourceFreq_Hz – SPDIF system clock frequency in hz.

void SPDIF_WriteBlocking(SPDIF_Type *base, uint8_t *buffer, uint32_t size)

Sends data using a blocking method.

Note: This function blocks by polling until data is ready to be sent.

Parameters

- base – SPDIF base pointer.
- buffer – Pointer to the data to be written.
- size – Bytes to be written.

static inline void SPDIF_WriteLeftData(SPDIF_Type *base, uint32_t data)

Writes data into SPDIF FIFO.

Parameters

- base – SPDIF base pointer.
- data – Data needs to be written.

static inline void SPDIF_WriteRightData(SPDIF_Type *base, uint32_t data)

Writes data into SPDIF FIFO.

Parameters

- base – SPDIF base pointer.
- data – Data needs to be written.

static inline void SPDIF_WriteChannelStatusHigh(SPDIF_Type *base, uint32_t data)

Writes data into SPDIF FIFO.

Parameters

- base – SPDIF base pointer.
- data – Data needs to be written.

static inline void SPDIF_WriteChannelStatusLow(SPDIF_Type *base, uint32_t data)

Writes data into SPDIF FIFO.

Parameters

- base – SPDIF base pointer.
- data – Data needs to be written.

void SPDIF_ReadBlocking(SPDIF_Type *base, uint8_t *buffer, uint32_t size)

Receives data using a blocking method.

Note: This function blocks by polling until data is ready to be sent.

Parameters

- base – SPDIF base pointer.
- buffer – Pointer to the data to be read.
- size – Bytes to be read.

static inline uint32_t SPDIF_ReadLeftData(SPDIF_Type *base)

Reads data from the SPDIF FIFO.

Parameters

- base – SPDIF base pointer.

Returns

Data in SPDIF FIFO.

static inline uint32_t SPDIF_ReadRightData(SPDIF_Type *base)

Reads data from the SPDIF FIFO.

Parameters

- base – SPDIF base pointer.

Returns

Data in SPDIF FIFO.

static inline uint32_t SPDIF_ReadChannelStatusHigh(SPDIF_Type *base)

Reads data from the SPDIF FIFO.

Parameters

- base – SPDIF base pointer.

Returns
Data in SPDIF FIFO.

static inline uint32_t SPDIF_ReadChannelStatusLow(SPDIF_Type *base)

Reads data from the SPDIF FIFO.

Parameters

- base – SPDIF base pointer.

Returns
Data in SPDIF FIFO.

static inline uint32_t SPDIF_ReadQChannel(SPDIF_Type *base)

Reads data from the SPDIF FIFO.

Parameters

- base – SPDIF base pointer.

Returns
Data in SPDIF FIFO.

static inline uint32_t SPDIF_ReadUChannel(SPDIF_Type *base)

Reads data from the SPDIF FIFO.

Parameters

- base – SPDIF base pointer.

Returns
Data in SPDIF FIFO.

void SPDIF_CopyRxChannelStatusToTx(SPDIF_Type *base)

Copies received channel-status words into the transmit channel-status registers.

Parameters

- base – SPDIF base pointer.

void SPDIF_TransferTxCreateHandle(SPDIF_Type *base, *spdif_handle_t* *handle, *spdif_transfer_callback_t* callback, void *userData)

Initializes the SPDIF Tx handle.

This function initializes the Tx handle for the SPDIF Tx transactional APIs. Call this function once to get the handle initialized.

Parameters

- base – SPDIF base pointer
- handle – SPDIF handle pointer.
- callback – Pointer to the user callback function.
- userData – User parameter passed to the callback function

void SPDIF_TransferRxCreateHandle(SPDIF_Type *base, *spdif_handle_t* *handle, *spdif_transfer_callback_t* callback, void *userData)

Initializes the SPDIF Rx handle.

This function initializes the Rx handle for the SPDIF Rx transactional APIs. Call this function once to get the handle initialized.

Parameters

- base – SPDIF base pointer.
- handle – SPDIF handle pointer.
- callback – Pointer to the user callback function.

- userData – User parameter passed to the callback function.

status_t SPDIF_TransferSendNonBlocking(SPDIF_Type *base, *spdif_handle_t* *handle, *spdif_transfer_t* *xfer)

Performs an interrupt non-blocking send transfer on SPDIF.

Note: This API returns immediately after the transfer initiates. Call the SPDIF_TxGetTransferStatusIRQ to poll the transfer status and check whether the transfer is finished. If the return status is not kStatus_SPDIF_Busy, the transfer is finished.

Parameters

- base – SPDIF base pointer.
- handle – Pointer to the spdif_handle_t structure which stores the transfer state.
- xfer – Pointer to the spdif_transfer_t structure.

Return values

- kStatus_Success – Successfully started the data receive.
- kStatus_SPDIF_TxBusy – Previous receive still not finished.
- kStatus_InvalidArgument – The input parameter is invalid.

status_t SPDIF_TransferReceiveNonBlocking(SPDIF_Type *base, *spdif_handle_t* *handle, *spdif_transfer_t* *xfer)

Performs an interrupt non-blocking receive transfer on SPDIF.

Note: This API returns immediately after the transfer initiates. Call the SPDIF_RxGetTransferStatusIRQ to poll the transfer status and check whether the transfer is finished. If the return status is not kStatus_SPDIF_Busy, the transfer is finished.

Parameters

- base – SPDIF base pointer
- handle – Pointer to the spdif_handle_t structure which stores the transfer state.
- xfer – Pointer to the spdif_transfer_t structure.

Return values

- kStatus_Success – Successfully started the data receive.
- kStatus_SPDIF_RxBusy – Previous receive still not finished.
- kStatus_InvalidArgument – The input parameter is invalid.

status_t SPDIF_TransferGetSendCount(SPDIF_Type *base, *spdif_handle_t* *handle, size_t *count)

Gets a set byte count.

Parameters

- base – SPDIF base pointer.
- handle – Pointer to the spdif_handle_t structure which stores the transfer state.
- count – Bytes count sent.

Return values

- kStatus_Success – Succeed get the transfer count.
- kStatus_NoTransferInProgress – There is not a non-blocking transaction currently in progress.

status_t SPDIF_TransferGetReceiveCount(SPDIF_Type *base, *spdif_handle_t* *handle, size_t *count)

Gets a received byte count.

Parameters

- base – SPDIF base pointer.
- handle – Pointer to the spdif_handle_t structure which stores the transfer state.
- count – Bytes count received.

Return values

- kStatus_Success – Succeed get the transfer count.
- kStatus_NoTransferInProgress – There is not a non-blocking transaction currently in progress.

void SPDIF_TransferAbortSend(SPDIF_Type *base, *spdif_handle_t* *handle)

Aborts the current send.

Note: This API can be called any time when an interrupt non-blocking transfer initiates to abort the transfer early.

Parameters

- base – SPDIF base pointer.
- handle – Pointer to the spdif_handle_t structure which stores the transfer state.

void SPDIF_TransferAbortReceive(SPDIF_Type *base, *spdif_handle_t* *handle)

Aborts the current IRQ receive.

Note: This API can be called when an interrupt non-blocking transfer initiates to abort the transfer early.

Parameters

- base – SPDIF base pointer
- handle – Pointer to the spdif_handle_t structure which stores the transfer state.

void SPDIF_TransferTxHandleIRQ(SPDIF_Type *base, *spdif_handle_t* *handle)

Tx interrupt handler.

Parameters

- base – SPDIF base pointer.
- handle – Pointer to the spdif_handle_t structure.

void SPDIF_TransferRxHandleIRQ(SPDIF_Type *base, *spdif_handle_t* *handle)

Tx interrupt handler.

Parameters

- base – SPDIF base pointer.
- handle – Pointer to the spdif_handle_t structure.

FSL_SPDIF_DRIVER_VERSION

Version 2.0.10

SPDIF return status.

Values:

enumerator kStatus_SPDIF_RxDPLLLocked

SPDIF Rx PLL locked.

enumerator kStatus_SPDIF_TxFIFOError

SPDIF Tx FIFO error.

enumerator kStatus_SPDIF_TxFIFOResync

SPDIF Tx left and right FIFO resync.

enumerator kStatus_SPDIF_RxCnew

SPDIF Rx status channel value updated.

enumerator kStatus_SPDIF_ValidatyNoGood

SPDIF validaty flag not good.

enumerator kStatus_SPDIF_RxIllegalSymbol

SPDIF Rx receive illegal symbol.

enumerator kStatus_SPDIF_RxParityBitError

SPDIF Rx parity bit error.

enumerator kStatus_SPDIF_UChannelOverrun

SPDIF receive U channel overrun.

enumerator kStatus_SPDIF_QChannelOverrun

SPDIF receive Q channel overrun.

enumerator kStatus_SPDIF_UQChannelSync

SPDIF U/Q channel sync found.

enumerator kStatus_SPDIF_UQChannelFrameError

SPDIF U/Q channel frame error.

enumerator kStatus_SPDIF_RxFIFOError

SPDIF Rx FIFO error.

enumerator kStatus_SPDIF_RxFIFOResync
SPDIF Rx left and right FIFO resync.

enumerator kStatus_SPDIF_LockLoss
SPDIF Rx PLL clock lock loss.

enumerator kStatus_SPDIF_TxIdle
SPDIF Tx is idle

enumerator kStatus_SPDIF_RxIdle
SPDIF Rx is idle

enumerator kStatus_SPDIF_QueueFull
SPDIF queue full

enum _spdif_rxfull_select
SPDIF Rx FIFO full falg select, it decides when assert the rx full flag.
Values:

enumerator kSPDIF_RxFull1Sample
Rx full at least 1 sample in left and right FIFO

enumerator kSPDIF_RxFull4Samples
Rx full at least 4 sample in left and right FIFO

enumerator kSPDIF_RxFull8Samples
Rx full at least 8 sample in left and right FIFO

enumerator kSPDIF_RxFull16Samples
Rx full at least 16 sample in left and right FIFO

enum _spdif_txempty_select
SPDIF tx FIFO EMPTY falg select, it decides when assert the tx empty flag.
Values:

enumerator kSPDIF_TxEmpty0Sample
Tx empty at most 0 sample in left and right FIFO

enumerator kSPDIF_TxEmpty4Samples
Tx empty at most 4 sample in left and right FIFO

enumerator kSPDIF_TxEmpty8Samples
Tx empty at most 8 sample in left and right FIFO

enumerator kSPDIF_TxEmpty12Samples
Tx empty at most 12 sample in left and right FIFO

enum _spdif_uchannel_source

SPDIF U channel source.

Values:

enumerator kSPDIF_NoUChannel

No embedded U channel

enumerator kSPDIF_UChannelFromRx

U channel from receiver, it is CD mode

enumerator kSPDIF_UChannelFromTx

U channel from on chip tx

enum _spdif_gain_select

SPDIF clock gain.

Values:

enumerator kSPDIF_GAIN_24

Gain select is 24

enumerator kSPDIF_GAIN_16

Gain select is 16

enumerator kSPDIF_GAIN_12

Gain select is 12

enumerator kSPDIF_GAIN_8

Gain select is 8

enumerator kSPDIF_GAIN_6

Gain select is 6

enumerator kSPDIF_GAIN_4

Gain select is 4

enumerator kSPDIF_GAIN_3

Gain select is 3

enum _spdif_tx_source

SPDIF tx data source.

Values:

enumerator kSPDIF_txFromReceiver

Tx data directly through SPDIF receiver

enumerator kSPDIF_txNormal

Normal operation, data from processor

enum _spdif_validity_config

SPDIF tx data source.

Values:

enumerator kSPDIF_validityFlagAlwaysSet

Outgoing validity flags always set

enumerator kSPDIF_validityFlagAlwaysClear

Outgoing validity flags always clear

The SPDIF interrupt enable flag.

Values:

enumerator kSPDIF_RxDPLLLocked

SPDIF DPLL locked

enumerator kSPDIF_TxFIFOError

Tx FIFO underrun or overrun

enumerator kSPDIF_TxFIFOResync

Tx FIFO left and right channel resync

enumerator kSPDIF_RxControlChannelChange

SPDIF Rx control channel value changed

enumerator kSPDIF_ValidityFlagNoGood

SPDIF validity flag no good

enumerator kSPDIF_RxIllegalSymbol

SPDIF receiver found illegal symbol

enumerator kSPDIF_RxParityBitError

SPDIF receiver found parity bit error

enumerator kSPDIF_UChannelReceiveRegisterFull

SPDIF U channel revceive register full

enumerator kSPDIF_UChannelReceiveRegisterOverrun

SPDIF U channel receive register overrun

enumerator kSPDIF_QChannelReceiveRegisterFull

SPDIF Q channel receive reigster full

enumerator kSPDIF_QChannelReceiveRegisterOverrun

SPDIF Q channel receive register overrun

enumerator kSPDIF_UQChannelSync
SPDIF U/Q channel sync found

enumerator kSPDIF_UQChannelFrameError
SPDIF U/Q channel frame error

enumerator kSPDIF_RxFIFOError
SPDIF Rx FIFO underrun/overrun

enumerator kSPDIF_RxFIFOResync
SPDIF Rx left and right FIFO resync

enumerator kSPDIF_LockLoss
SPDIF receiver loss of lock

enumerator kSPDIF_TxFIFOEmpty
SPDIF Tx FIFO empty

enumerator kSPDIF_RxFIFOFull
SPDIF Rx FIFO full

enumerator kSPDIF_AllInterrupt
all interrupt

The DMA request sources.

Values:

enumerator kSPDIF_RxDMAEnable
Rx FIFO full

enumerator kSPDIF_TxDMAEnable
Tx FIFO empty

typedef enum *_spdif_rxfull_select* spdif_rxfull_select_t
SPDIF Rx FIFO full falg select, it decides when assert the rx full flag.

typedef enum *_spdif_txempty_select* spdif_txempty_select_t
SPDIF tx FIFO EMPTY falg select, it decides when assert the tx empty flag.

typedef enum *_spdif_uchannel_source* spdif_uchannel_source_t
SPDIF U channel source.

typedef enum *_spdif_gain_select* spdif_gain_select_t
SPDIF clock gain.

typedef enum *_spdif_tx_source* spdif_tx_source_t
SPDIF tx data source.

typedef enum *_spdif_validity_config* spdif_validity_config_t

SPDIF tx data source.

typedef struct *_spdif_config* spdif_config_t

SPDIF user configuration structure.

typedef struct *_spdif_transfer* spdif_transfer_t

SPDIF transfer structure.

typedef struct *_spdif_handle* spdif_handle_t

typedef void (*spdif_transfer_callback_t)(SPDIF_Type *base, *spdif_handle_t* *handle, *status_t* status, void *userData)

SPDIF transfer callback prototype.

SPDIF_XFER_QUEUE_SIZE

SPDIF transfer queue size, user can refine it according to use case.

struct _spdif_config

#include <fsl_spdif.h> SPDIF user configuration structure.

Public Members

bool isTxAutoSync

If auto sync mechanism open

bool isRxAutoSync

If auto sync mechanism open

uint8_t DPLLClkSource

SPDIF DPLL clock source, range from 0~15, meaning is chip-specific

uint8_t txClkSource

SPDIF tx clock source, range from 0~7, meaning is chip-specific

spdif_rxfull_select_t rxFullSelect

SPDIF rx buffer full select

spdif_txempty_select_t txFullSelect

SPDIF tx buffer empty select

spdif_uchannel_source_t uChannelSrc

U channel source

spdif_tx_source_t txSource

SPDIF tx data source

spdif_validity_config_t validityConfig
Validity flag config

spdif_gain_select_t gain
Rx receive clock measure gain parameter.

struct _spdif_transfer
#include <fsl_spdif.h> SPDIF transfer structure.

Public Members

uint8_t *data
Data start address to transfer.

uint8_t *qdata
Data buffer for Q channel

uint8_t *udata
Data buffer for C channel

size_t dataSize
Transfer size.

struct _spdif_handle
#include <fsl_spdif.h> SPDIF handle structure.

Public Members

uint32_t state
Transfer status

spdif_transfer_callback_t callback
Callback function called at transfer event

void *userData
Callback parameter passed to callback function

spdif_transfer_t spdifQueue[(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

uint8_t watermark

Watermark value

2.64 SPDIF eDMA Driver

void SPDIF_TransferTxCreateHandleEDMA(SPDIF_Type *base, *spdif_edma_handle_t* *handle, *spdif_edma_callback_t* callback, void *userData, *edma_handle_t* *dmaLeftHandle, *edma_handle_t* *dmaRightHandle)

Initializes the SPDIF eDMA handle.

This function initializes the SPDIF master DMA handle, which can be used for other SPDIF master transactional APIs. Usually, for a specified SPDIF instance, call this API once to get the initialized handle.

Parameters

- base – SPDIF base pointer.
- handle – SPDIF eDMA handle pointer.
- callback – Pointer to user callback function.
- userData – User parameter passed to the callback function.
- dmaLeftHandle – eDMA handle pointer for left channel, this handle shall be static allocated by users.
- dmaRightHandle – eDMA handle pointer for right channel, this handle shall be static allocated by users.

void SPDIF_TransferRxCreateHandleEDMA(SPDIF_Type *base, *spdif_edma_handle_t* *handle, *spdif_edma_callback_t* callback, void *userData, *edma_handle_t* *dmaLeftHandle, *edma_handle_t* *dmaRightHandle)

Initializes the SPDIF Rx eDMA handle.

This function initializes the SPDIF slave DMA handle, which can be used for other SPDIF master transactional APIs. Usually, for a specified SPDIF instance, call this API once to get the initialized handle.

Parameters

- base – SPDIF base pointer.
- handle – SPDIF eDMA handle pointer.
- callback – Pointer to user callback function.
- userData – User parameter passed to the callback function.
- dmaLeftHandle – eDMA handle pointer for left channel, this handle shall be static allocated by users.
- dmaRightHandle – eDMA handle pointer for right channel, this handle shall be static allocated by users.

status_t SPDIF_TransferSendEDMA(SPDIF_Type *base, *spdif_edma_handle_t* *handle, *spdif_edma_transfer_t* *xfer)

Performs a non-blocking SPDIF transfer using DMA.

Note: This interface returns immediately after the transfer initiates. Call SPDIF_GetTransferStatus to poll the transfer status and check whether the SPDIF transfer is finished.

Parameters

- base – SPDIF base pointer.
- handle – SPDIF eDMA handle pointer.
- xfer – Pointer to the DMA transfer structure.

Return values

- kStatus_Success – Start a SPDIF eDMA send successfully.
- kStatus_InvalidArgument – The input argument is invalid.
- kStatus_TxBusy – SPDIF is busy sending data.

status_t SPDIF_TransferReceiveEDMA(SPDIF_Type *base, *spdif_edma_handle_t* *handle, *spdif_edma_transfer_t* *xfer)

Performs a non-blocking SPDIF receive using eDMA.

Note: This interface returns immediately after the transfer initiates. Call the SPDIF_GetReceiveRemainingBytes to poll the transfer status and check whether the SPDIF transfer is finished.

Parameters

- base – SPDIF base pointer
- handle – SPDIF eDMA handle pointer.
- xfer – Pointer to DMA transfer structure.

Return values

- kStatus_Success – Start a SPDIF eDMA receive successfully.
- kStatus_InvalidArgument – The input argument is invalid.
- kStatus_RxBusy – SPDIF is busy receiving data.

void SPDIF_TransferAbortSendEDMA(SPDIF_Type *base, *spdif_edma_handle_t* *handle)

Aborts a SPDIF transfer using eDMA.

Parameters

- base – SPDIF base pointer.
- handle – SPDIF eDMA handle pointer.

void SPDIF_TransferAbortReceiveEDMA(SPDIF_Type *base, *spdif_edma_handle_t* *handle)

Aborts a SPDIF receive using eDMA.

Parameters

- base – SPDIF base pointer

- handle – SPDIF eDMA handle pointer.

status_t SPDIF_TransferGetSendCountEDMA(SPDIF_Type *base, *spdif_edma_handle_t* *handle, size_t *count)

Gets byte count sent by SPDIF.

Parameters

- base – SPDIF base pointer.
- handle – SPDIF eDMA handle pointer.
- count – Bytes count sent by SPDIF.

Return values

- kStatus_Success – Succeed get the transfer count.
- kStatus_NoTransferInProgress – There is no non-blocking transaction in progress.

status_t SPDIF_TransferGetReceiveCountEDMA(SPDIF_Type *base, *spdif_edma_handle_t* *handle, size_t *count)

Gets byte count received by SPDIF.

Parameters

- base – SPDIF base pointer
- handle – SPDIF eDMA handle pointer.
- count – Bytes count received by SPDIF.

Return values

- kStatus_Success – Succeed get the transfer count.
- kStatus_NoTransferInProgress – There is no non-blocking transaction in progress.

FSL_SPDIF_EDMA_DRIVER_VERSION

Version 2.0.9

typedef struct *_spdif_edma_handle* spdif_edma_handle_t

typedef void (*spdif_edma_callback_t)(SPDIF_Type *base, *spdif_edma_handle_t* *handle, *status_t* status, void *userData)

SPDIF eDMA transfer callback function for finish and error.

typedef struct *_spdif_edma_transfer* spdif_edma_transfer_t

SPDIF transfer structure.

struct _spdif_edma_transfer

#include <fsl_spdif_edma.h> SPDIF transfer structure.

Public Members

uint8_t *leftData

Data start address to transfer.

uint8_t *rightData

Data start address to transfer.

size_t dataSize

Transfer size.

struct _spdif_edma_handle

#include <fsl_spdif_edma.h> SPDIF DMA transfer handle, users should not touch the content of the handle.

Public Members

edma_handle_t *dmaLeftHandle

DMA handler for SPDIF left channel

edma_handle_t *dmaRightHandle

DMA handler for SPDIF right channel

uint8_t nbytes

eDMA minor byte transfer count initially configured.

uint8_t count

The transfer data count in a DMA request

uint32_t state

Internal state for SPDIF eDMA transfer

spdif_edma_callback_t callback

Callback for users while transfer finish or error occurs

void *userData

User callback parameter

edma_tcd_t leftTcd[(4U) + 1U]

TCD pool for eDMA transfer.

edma_tcd_t rightTcd[(4U) + 1U]

TCD pool for eDMA transfer.

spdif_edma_transfer_t spdifQueue[(4U)]

Transfer queue storing queued transfer.

size_t transferSize[(4U)]

Data bytes need to transfer, left and right are the same, so use one

volatile uint8_t queueUser

Index for user to queue transfer.

volatile uint8_t queueDriver

Index for driver to get the transfer data and size

2.65 SRC: System Reset Controller Driver

FSL_SRC_DRIVER_VERSION

SRC driver version 2.0.1.

enum _src_reset_status_flags

SRC reset status flags.

Values:

enumerator kSRC_ResetOutputEnableFlag

This bit indicates if RESET status is driven out on PTE0 pin.

enumerator kSRC_TemperatureSensorResetFlag

Indicates whether the reset was the result of software reset from on-chip Temperature Sensor. Temperature Sensor Interrupt needs to be served before this bit can be cleaned.

enumerator kSRC_Wdog3ResetFlag

IC Watchdog3 Time-out reset. Indicates whether the reset was the result of the watchdog3 time-out event.

enumerator kSRC_SoftwareResetFlag

Indicates a reset has been caused by software setting of SYSRESETREQ bit in Application Interrupt and Reset Control Register in the ARM core.

enumerator kSRC_JTAGSystemResetFlag

Indicates whether the reset was the result of software reset form JTAG

enumerator kSRC_JTAGSoftwareResetFlag

Indicates whether the reset was the result of setting SJC_GPCCR bit 31.

enumerator kSRC_JTAGGeneratedResetFlag

Indicates a reset has been caused by JTAG selection of certain IR codes: EXTEST or HIGHZ.

enumerator kSRC_WatchdogResetFlag

Indicates a reset has been caused by the watchdog timer timing out. This reset source can be blocked by disabling the watchdog.

enumerator kSRC_IppUserResetFlag

Indicates whether the reset was the result of the ipp_user_reset_b qualified reset.

enumerator kSRC_SNVSFailResetFlag

SNVS hardware failure will always cause a cold reset. This flag indicates whether the reset is a result of SNVS hardware failure.

enumerator kSRC_CsuResetFlag

Indicates whether the reset was the result of the csu_reset_b input.

enumerator kSRC_CoreLockupResetFlag

Indicates a reset has been caused by the ARM core indication of a LOCKUP event.

enumerator kSRC_PowerOnResetFlag

Indicates a reset has been caused by the power-on detection logic.

enumerator kSRC_LockupSysResetFlag

Indicates a reset has been caused by CPU lockup or software setting of SYSRESETREQ bit in Application Interrupt and Reset Control Register of the ARM core.

enumerator kSRC_IppResetPinFlag

Indicates whether reset was the result of ipp_reset_b pin (Power-up sequence).

enum _src_status_flags

SRC interrupt status flag.

Values:

enumerator kSRC_Core0WdogResetReqFlag

WDOG reset request from core0. Read-only status bit.

enum _src_mix_reset_stretch_cycles

Selection of SoC mix power reset stretch.

This type defines the SoC mix (Audio, ENET, uSDHC, EIM, QSPI, OCRAM, MMDC, etc) power up reset stretch mix reset width with the optional count of cycles

Values:

enumerator kSRC_MixResetStretchCycleAlt0

mix reset width is 1 x 88 ipg_cycle cycles.

enumerator kSRC_MixResetStretchCycleAlt1

mix reset width is 2 x 88 ipg_cycle cycles.

enumerator kSRC_MixResetStretchCycleAlt2

mix reset width is 3 x 88 ipg_cycle cycles.

enumerator kSRC_MixResetStretchCycleAlt3

mix reset width is 4 x 88 ipg_cycle cycles.

enum _src_wdog3_reset_option

Selection of WDOG3 reset option.

Values:

enumerator kSRC_Wdog3ResetOptionAlt0

Wdog3_rst_b asserts M4 reset (default).

enumerator kSRC_Wdog3ResetOptionAlt1

Wdog3_rst_b asserts global reset.

enum _src_warm_reset_bypass_count

Selection of WARM reset bypass count.

This type defines the 32KHz clock cycles to count before bypassing the MMDC acknowledge for WARM reset. If the MMDC acknowledge is not asserted before this counter is elapsed, a COLD reset will be initiated.

Values:

enumerator kSRC_WarmResetWaitAlways

System will wait until MMDC acknowledge is asserted.

enumerator kSRC_WarmResetWaitClk16

Wait 16 32KHz clock cycles before switching the reset.

enumerator kSRC_WarmResetWaitClk32

Wait 32 32KHz clock cycles before switching the reset.

enumerator kSRC_WarmResetWaitClk64

Wait 64 32KHz clock cycles before switching the reset.

typedef enum *_src_mix_reset_stretch_cycles* src_mix_reset_stretch_cycles_t

Selection of SoC mix power reset stretch.

This type defines the SoC mix (Audio, ENET, uSDHC, EIM, QSPI, OCRAM, MMDC, etc) power up reset stretch mix reset width with the optional count of cycles

typedef enum *_src_wdog3_reset_option* src_wdog3_reset_option_t

Selection of WDOG3 reset option.

typedef enum *_src_warm_reset_bypass_count* src_warm_reset_bypass_count_t

Selection of WARM reset bypass count.

This type defines the 32KHz clock cycles to count before bypassing the MMDC acknowledge for WARM reset. If the MMDC acknowledge is not asserted before this counter is elapsed, a COLD reset will be initiated.

static inline void SRC_EnableWDOG3Reset(SRC_Type *base, bool enable)

Enable the WDOG3 reset.

The WDOG3 reset is enabled by default.

Parameters

- base – SRC peripheral base address.
- enable – Enable the reset or not.

static inline void SRC_SetMixResetStretchCycles(SRC_Type *base, *src_mix_reset_stretch_cycles_t* option)

Set the mix power up reset stretch mix reset width.

Parameters

- base – SRC peripheral base address.

- option – Setting option, see to *src_mix_reset_stretch_cycles_t*.

static inline void SRC_EnableCoreDebugResetAfterPowerGate(SRC_Type *base, bool enable)

Debug reset would be asserted after power gating event.

Parameters

- base – SRC peripheral base address.
- enable – Enable the reset or not.

static inline void SRC_SetWdog3ResetOption(SRC_Type *base, *src_wdog3_reset_option_t* option)

Set the Wdog3_rst_b option.

Parameters

- base – SRC peripheral base address.
- option – Setting option, see to *src_wdog3_reset_option_t*.

static inline void SRC_DoSoftwareResetARMCoreDebug(SRC_Type *base)

Software reset for debug of arm platform only.

Parameters

- base – SRC peripheral base address.

static inline bool SRC_GetSoftwareResetARMCoreDebugDone(SRC_Type *base)

Check if the software reset for debug of arm platform only is done.

Parameters

- base – SRC peripheral base address.

static inline void SRC_EnableTemperatureSensorReset(SRC_Type *base, bool enable)

Enable the temperature sensor reset.

The temperature sersor reset is enabled by default. When the sensor reset happens, an flag bit would be asserted. This flag bit can be cleared only by the hardware reset.

Parameters

- base – SRC peripheral base address.
- enable – Enable the reset or not.

static inline void SRC_DoSoftwareResetARMCore0(SRC_Type *base)

Do software reset the ARM core0 only.

Parameters

- base – SRC peripheral base address.

static inline bool SRC_GetSoftwareResetARMCore0Done(SRC_Type *base)

Check if the software for ARM core0 is done.

Parameters

- base – SRC peripheral base address.

Returns

If the reset is done.

static inline void SRC_DoSoftwareResetARMCore(SRC_Type *base)

Do software reset for ARM core.

This function can be used to assert the ARM core reset. Once it is called, the reset process will begin. After the reset process is finished, the command bit would be self cleared.

Parameters

- base – SRC peripheral base address.

static inline bool SRC_GetSoftwareResetARMCoreDone(SRC_Type *base)

Check if the software for ARM core is done.

Parameters

- base – SRC peripheral base address.

Returns

If the reset is done.

static inline void SRC_AssertEIMReset(SRC_Type *base, bool enable)

Assert the EIM reset.

EIM reset is needed in order to reconfigure the EIM chip select. The software reset bit must de-asserted since this is not self-refresh.

Parameters

- base – SRC peripheral base address.
- enable – Make the assertion or not.

static inline void SRC_EnableWDOGReset(SRC_Type *base, bool enable)

Enable the WDOG Reset in SRC.

WDOG Reset is enabled in SRC by default. If the WDOG event to SRC is masked, it would not create a reset to the chip. During the time the WDOG event is masked, when the WDOG event flag is asserted, it would remain asserted regardless of servicing the WDOG module. The only way to clear that bit is the hardware reset.

Parameters

- base – SRC peripheral base address.
- enable – Enable the reset or not.

static inline void SRC_EnableLockupReset(SRC_Type *base, bool enable)

Enable the lockup reset.

Parameters

- base – SRC peripheral base address.
- enable – Enable the reset or not.

static inline void SRC_EnableCoreLockupReset(SRC_Type *base, bool enable)

Enable the core lockup reset.

When enable the core luckup reset, the system would be reset when core luckup event happens.

Parameters

- base – SRC peripheral base address.
- enable – Enable the reset or not.

static inline uint32_t SRC_GetStatusFlags(SRC_Type *base)

Get interrupt status flags.

Parameters

- base – SRC peripheral base address.

Returns

Mask value of status flags. See to $_src_status_flags.

static inline uint32_t SRC_GetBootModeWord1(SRC_Type *base)

Get the boot mode register 1 value.

The Boot Mode register contains bits that reflect the status of BOOT_CFGx pins of the chip. See to chip-specific document for detail information about value.

Parameters

- base – SRC peripheral base address.

Returns

status of BOOT_CFGx pins of the chip.

static inline uint32_t SRC_GetBootModeWord2(SRC_Type *base)

Get the boot mode register 2 value.

The Boot Mode register contains bits that reflect the status of BOOT_MODEx Pins and fuse values that controls boot of the chip. See to chip-specific document for detail information about value.

Parameters

- base – SRC peripheral base address.

Returns

status of BOOT_MODEx Pins and fuse values that controls boot of the chip.

static inline uint32_t SRC_GetResetStatusFlags(SRC_Type *base)

Get the status flags of SRC.

Parameters

- base – SRC peripheral base address.

Returns

Mask value of status flags, see to *_src_reset_status_flags*.

void SRC_ClearResetStatusFlags(SRC_Type *base, uint32_t flags)

Clear the status flags of SRC.

Parameters

- base – SRC peripheral base address.
- flags – value of status flags to be cleared, see to *_src_reset_status_flags*.

static inline void SRC_SetGeneralPurposeRegister(SRC_Type *base, uint32_t index, uint32_t value)

Set value to general purpose registers.

General purpose registers (GPRx) would hold the value during reset process. Wakeup function could be kept in these register. For example, the GPR1 holds the entry function for waking-up from Partial SLEEP mode while the GPR2 holds the argument. Other GPRx register would store the arbitray values.

Parameters

- base – SRC peripheral base address.
- index – The index of GPRx register array. Note index 0 reponses the GPR1 register.
- value – Setting value for GPRx register.

static inline uint32_t SRC_GetGeneralPurposeRegister(SRC_Type *base, uint32_t index)

Get the value from general purpose registers.

Parameters

- base – SRC peripheral base address.

- index – The index of GPRx register array. Note index 0 reponses the GPR1 register.

Returns
The setting value for GPRx register.

2.66 TEMPMON: Temperature Monitor Module

FSL_TEMPMON_DRIVER_VERSION
TEMPMON driver version.

enum _tempmon_alarm_mode
TEMPMON alarm mode.

Values:

enumerator kTEMPMON_HighAlarmMode
The high alarm temperature interrupt mode.

enumerator kTEMPMON_PanicAlarmMode
The panic alarm temperature interrupt mode.

enumerator kTEMPMON_LowAlarmMode
The low alarm temperature interrupt mode.

typedef struct *_tempmon_config* tempmon_config_t
TEMPMON temperature structure.

typedef enum *_tempmon_alarm_mode* tempmon_alarm_mode
TEMPMON alarm mode.

void TEMPMON_Init(TEMPMON_Type *base, const *tempmon_config_t* *config)
Initializes the TEMPMON module.

Parameters

- base – TEMPMON base pointer
- config – Pointer to configuration structure.

void TEMPMON_Deinit(TEMPMON_Type *base)
Deinitializes the TEMPMON module.

Parameters

- base – TEMPMON base pointer

void TEMPMON_GetDefaultConfig(*tempmon_config_t* *config)
Gets the default configuration structure.

This function initializes the TEMPMON configuration structure to a default value. The default values are: tempmonConfig->frequency = 0x02U; tempmonConfig->highAlarmTemp = 44U; tempmonConfig->panicAlarmTemp = 90U; tempmonConfig->lowAlarmTemp = 39U;

Parameters

- config – Pointer to a configuration structure.

static inline void TEMPMON_StartMeasure(TEMPMON_Type *base)

start the temperature measurement process.

Parameters

- base – TEMPMON base pointer.

static inline void TEMPMON_StopMeasure(TEMPMON_Type *base)

stop the measurement process.

Parameters

- base – TEMPMON base pointer

float TEMPMON_GetCurrentTemperature(TEMPMON_Type *base)

Get current temperature with the fused temperature calibration data.

Parameters

- base – TEMPMON base pointer

Returns

current temperature with degrees Celsius.

void TEMPMON_SetTempAlarm(TEMPMON_Type *base, int16_t tempVal, *tempmon_alarm_mode* alarmMode)

Set the temperature count (raw sensor output) that will generate an alarm interrupt.

Parameters

- base – TEMPMON base pointer
- tempVal – The alarm temperature with degrees Celsius
- alarmMode – The alarm mode.

TEMPMON_HOTTEMPMASK

TEMPMON calibration data mask.

TEMPMON_HOTTEMPSHIFT

TEMPMON_HOTCOUNTMASK

TEMPMON_HOTCOUNTSHIFT

TEMPMON_ROOMCOUNTMASK

TEMPMON_ROOMCOUNTSHIFT

struct _tempmon_config

#include <fsl_tempmon.h> TEMPMON temperature structure.

Public Members

uint16_t frequency

The temperature measure frequency.

int16_t highAlarmTemp
: The high alarm temperature.

int16_t panicAlarmTemp
: The panic alarm temperature.

int16_t lowAlarmTemp
: The low alarm temperature.

2.67 TRNG: True Random Number Generator

FSL_TRNG_DRIVER_VERSION
: TRNG driver version 2.0.23.

Current version: 2.0.23

Change log:

- version 2.0.23
  - Updated TRNG default values for MCXA577 devices based on silicon characterization.
- version 2.0.22
  - Added support for KW43 and MCXW70 devices.
- version 2.0.21
  - Added support for MCXC devices.
- version 2.0.20
  - Added support for MCXA devices.
- version 2.0.19
  - Added support for MCXA and MCXL.
- version 2.0.18
  - TRNG health checks now done in software on RT5xx and RT6xx.
- version 2.0.17
  - Added support for RT700.
- version 2.0.16
  - Added support for Dual oscillator mode.
- version 2.0.15
  - Changed TRNG_USER_CONFIG_DEFAULT_XXX values according to latest reccomended by design team.
- version 2.0.14
  - add support for RW610 and RW612
- version 2.0.13
  - After deepsleep it might return error, added clearing bits in *TRNG_GetRandomData()* and generating new entropy.

  - Modified reloading entropy in *TRNG_GetRandomData()*, for some data length it doesn't reloading entropy correctly.
- version 2.0.12
  - For KW34A4_SERIES, KW35A4_SERIES, KW36A4_SERIES set TRNG_USER_CONFIG_DEFAULT_OSC_DIV to kTRNG_RingOscDiv8.
- version 2.0.11
  - Add clearing pending errors in *TRNG_Init()*.
- version 2.0.10
  - Fixed doxygen issues.
- version 2.0.9
  - Fix HIS_CCM metrics issues.
- version 2.0.8
  - For K32L2A41A_SERIES set TRNG_USER_CONFIG_DEFAULT_OSC_DIV to kTRNG_RingOscDiv4.
- version 2.0.7
  - Fix MISRA 2004 issue rule 12.5.
- version 2.0.6
  - For KW35Z4_SERIES set TRNG_USER_CONFIG_DEFAULT_OSC_DIV to kTRNG_RingOscDiv8.
- version 2.0.5
  - Add possibility to define default TRNG configuration by device specific preprocessor macros for FRQMIN, FRQMAX and OSCDIV.
- version 2.0.4
  - Fix MISRA-2012 issues.
- Version 2.0.3
  - update TRNG_Init to restart entropy generation
- Version 2.0.2
  - fix MISRA issues
- Version 2.0.1
  - add support for KL8x and KL28Z
  - update default OSCDIV for K81 to divide by 2

enum _trng_sample_mode

TRNG sample mode. Used by trng_config_t.

Values:

enumerator kTRNG_SampleModeVonNeumann

Use von Neumann data in both Entropy shifter and Statistical Checker.

enumerator kTRNG_SampleModeRaw

Use raw data into both Entropy shifter and Statistical Checker.

enumerator kTRNG_SampleModeVonNeumannRaw

Use von Neumann data in Entropy shifter. Use raw data into Statistical Checker.

enum _trng_clock_mode

TRNG clock mode. Used by trng_config_t.

Values:

enumerator kTRNG_ClockModeRingOscillator

Ring oscillator is used to operate the TRNG (default).

enumerator kTRNG_ClockModeSystem

System clock is used to operate the TRNG. This is for test use only, and indeterminate results may occur.

enum _trng_ring_osc_div

TRNG ring oscillator divide. Used by trng_config_t.

Values:

enumerator kTRNG_RingOscDiv0

Ring oscillator with no divide

enumerator kTRNG_RingOscDiv2

Ring oscillator divided-by-2.

enumerator kTRNG_RingOscDiv4

Ring oscillator divided-by-4.

enumerator kTRNG_RingOscDiv8

Ring oscillator divided-by-8.

typedef enum *_trng_sample_mode* trng_sample_mode_t

TRNG sample mode. Used by trng_config_t.

typedef enum *_trng_clock_mode* trng_clock_mode_t

TRNG clock mode. Used by trng_config_t.

typedef enum *_trng_ring_osc_div* trng_ring_osc_div_t

TRNG ring oscillator divide. Used by trng_config_t.

typedef struct *_trng_statistical_check_limit* trng_statistical_check_limit_t

Data structure for definition of statistical check limits. Used by trng_config_t.

typedef struct *_trng_user_config* trng_config_t

Data structure for the TRNG initialization.

This structure initializes the TRNG by calling the *TRNG_Init()* function. It contains all TRNG configurations.

status_t TRNG_GetDefaultConfig(*trng_config_t* *userConfig)

Initializes the user configuration structure to default values.

This function initializes the configuration structure to default values. The default values are platform dependent.

Parameters

- userConfig – User configuration structure.

Returns

If successful, returns the kStatus_TRNG_Success. Otherwise, it returns an error.

status_t TRNG_Init(TRNG_Type *base, const *trng_config_t* *userConfig)

Initializes the TRNG.

This function initializes the TRNG. When called, the TRNG entropy generation starts immediately.

Parameters

- base – TRNG base address
- userConfig – Pointer to the initialization configuration structure.

Returns

If successful, returns the kStatus_TRNG_Success. Otherwise, it returns an error.

void TRNG_Deinit(TRNG_Type *base)

Shuts down the TRNG.

This function shuts down the TRNG.

Parameters

- base – TRNG base address.

status_t TRNG_GetRandomData(TRNG_Type *base, void *data, size_t dataSize)

Gets random data.

This function gets random data from the TRNG.

Parameters

- base – TRNG base address.
- data – Pointer address used to store random data.
- dataSize – Size of the buffer pointed by the data parameter.

Returns

random data

struct __trng_statistical_check_limit

#include <fsl_trng.h> Data structure for definition of statistical check limits. Used by trng_config_t.

Public Members

uint32_t maximum

Maximum limit.

int32_t minimum

Minimum limit.

struct __trng__user__config

#include <fsl_trng.h> Data structure for the TRNG initialization.

This structure initializes the TRNG by calling the *TRNG_Init()* function. It contains all TRNG configurations.

Public Members

bool lock

Disable programmability of TRNG registers.

trng_clock_mode_t clockMode

Clock mode used to operate TRNG.

trng_ring_osc_div_t ringOscDiv

Ring oscillator divide used by TRNG.

trng_sample_mode_t sampleMode

Sample mode of the TRNG ring oscillator.

uint16_t entropyDelay

Entropy Delay. Defines the length (in system clocks) of each Entropy sample taken.

uint16_t sampleSize

Sample Size. Defines the total number of Entropy samples that will be taken during Entropy generation.

uint16_t sparseBitLimit

Sparse Bit Limit which defines the maximum number of consecutive samples that may be discarded before an error is generated. This limit is used only for during von Neumann sampling (enabled by TRNG_HAL_SetSampleMode()). Samples are discarded if two consecutive raw samples are both 0 or both 1. If this discarding occurs for a long period of time, it indicates that there is insufficient Entropy.

uint8_t retryCount

Retry count. It defines the number of times a statistical check may fails during the TRNG Entropy Generation before generating an error.

uint8_t longRunMaxLimit

Largest allowable number of consecutive samples of all 1, or all 0, that is allowed during the Entropy generation.

trng_statistical_check_limit_t monobitLimit

Maximum and minimum limits for statistical check of number of ones/zero detected during entropy generation.

trng_statistical_check_limit_t runBit1Limit

Maximum and minimum limits for statistical check of number of runs of length 1 detected during entropy generation.

trng_statistical_check_limit_t runBit2Limit

Maximum and minimum limits for statistical check of number of runs of length 2 detected during entropy generation.

trng_statistical_check_limit_t runBit3Limit

Maximum and minimum limits for statistical check of number of runs of length 3 detected during entropy generation.

trng_statistical_check_limit_t runBit4Limit

Maximum and minimum limits for statistical check of number of runs of length 4 detected during entropy generation.

trng_statistical_check_limit_t runBit5Limit

Maximum and minimum limits for statistical check of number of runs of length 5 detected during entropy generation.

trng_statistical_check_limit_t runBit6PlusLimit

Maximum and minimum limits for statistical check of number of runs of length 6 or more detected during entropy generation.

trng_statistical_check_limit_t pokerLimit

Maximum and minimum limits for statistical check of "Poker Test".

trng_statistical_check_limit_t frequencyCountLimit

Maximum and minimum limits for statistical check of entropy sample frequency count.

2.68 USDHC: Ultra Secured Digital Host Controller Driver

void USDHC_Init(USDHC_Type *base, const *usdhc_config_t* *config)

USDHC module initialization function.

Configures the USDHC according to the user configuration.

Example:

```
usdhc_config_t config;
config.cardDetectDat3 = false;
config.endianMode = kUSDHC_EndianModeLittle;
config.dmaMode = kUSDHC_DmaModeAdma2;
config.readWatermarkLevel = 128U;
config.writeWatermarkLevel = 128U;
USDHC_Init(USDHC, &config);
```

Parameters

- base – USDHC peripheral base address.
- config – USDHC configuration information.

Return values

kStatus_Success – Operate successfully.

void USDHC_Deinit(USDHC_Type *base)

Deinitializes the USDHC.

Parameters

- base – USDHC peripheral base address.

bool USDHC_Reset(USDHC_Type *base, uint32_t mask, uint32_t timeout)

Resets the USDHC.

Parameters

- base – USDHC peripheral base address.
- mask – The reset type mask(*_usdhc_reset*).
- timeout – Timeout for reset.

Return values

- true – Reset successfully.
- false – Reset failed.

status_t USDHC_SetAdmaTableConfig(USDHC_Type *base, *usdhc_adma_config_t* *dmaConfig, *usdhc_data_t* *dataConfig, uint32_t flags)

Sets the DMA descriptor table configuration. A high level DMA descriptor configuration function.

Parameters

- base – USDHC peripheral base address.
- dmaConfig – ADMA configuration
- dataConfig – Data descriptor
- flags – ADAM descriptor flag, used to indicate to create multiple or single descriptor, please refer to enum *_usdhc_adma_flag*.

Return values

- kStatus_OutOfRange – ADMA descriptor table length isn't enough to describe data.
- kStatus_Success – Operate successfully.

status_t USDHC_SetInternalDmaConfig(USDHC_Type *base, *usdhc_adma_config_t* *dmaConfig, const uint32_t *dataAddr, bool enAutoCmd23)

Internal DMA configuration. This function is used to config the USDHC DMA related registers.

Parameters

- base – USDHC peripheral base address.
- dmaConfig – ADMA configuration.
- dataAddr – Transfer data address, a simple DMA parameter, if ADMA is used, leave it to NULL.
- enAutoCmd23 – Flag to indicate Auto CMD23 is enable or not, a simple DMA parameter, if ADMA is used, leave it to false.

Return values

- kStatus_OutOfRange – ADMA descriptor table length isn't enough to describe data.

- kStatus_Success – Operate successfully.

status_t USDHC_SetADMA2Descriptor(uint32_t *admaTable, uint32_t admaTableWords, const uint32_t *dataBufferAddr, uint32_t dataBytes, uint32_t flags)

Sets the ADMA2 descriptor table configuration.

Parameters

- admaTable – ADMA table address.
- admaTableWords – ADMA table length.
- dataBufferAddr – Data buffer address.
- dataBytes – Data Data length.
- flags – ADAM descriptor flag, used to indicate to create multiple or single descriptor, please refer to enum *_usdhc_adma_flag*.

Return values

- kStatus_OutOfRange – ADMA descriptor table length isn't enough to describe data.
- kStatus_Success – Operate successfully.

status_t USDHC_SetADMA1Descriptor(uint32_t *admaTable, uint32_t admaTableWords, const uint32_t *dataBufferAddr, uint32_t dataBytes, uint32_t flags)

Sets the ADMA1 descriptor table configuration.

Parameters

- admaTable – ADMA table address.
- admaTableWords – ADMA table length.
- dataBufferAddr – Data buffer address.
- dataBytes – Data length.
- flags – ADAM descriptor flag, used to indicate to create multiple or single descriptor, please refer to enum *_usdhc_adma_flag*.

Return values

- kStatus_OutOfRange – ADMA descriptor table length isn't enough to describe data.
- kStatus_Success – Operate successfully.

static inline void USDHC_EnableInternalDMA(USDHC_Type *base, bool enable)

Enables internal DMA.

Parameters

- base – USDHC peripheral base address.
- enable – enable or disable flag

static inline void USDHC_EnableInterruptStatus(USDHC_Type *base, uint32_t mask)

Enables the interrupt status.

Parameters

- base – USDHC peripheral base address.
- mask – Interrupt status flags mask(*_usdhc_interrupt_status_flag*).

static inline void USDHC_DisableInterruptStatus(USDHC_Type *base, uint32_t mask)

Disables the interrupt status.

Parameters

- base – USDHC peripheral base address.
- mask – The interrupt status flags mask(*_usdhc_interrupt_status_flag*).

static inline void USDHC_EnableInterruptSignal(USDHC_Type *base, uint32_t mask)

Enables the interrupt signal corresponding to the interrupt status flag.

Parameters

- base – USDHC peripheral base address.
- mask – The interrupt status flags mask(*_usdhc_interrupt_status_flag*).

static inline void USDHC_DisableInterruptSignal(USDHC_Type *base, uint32_t mask)

Disables the interrupt signal corresponding to the interrupt status flag.

Parameters

- base – USDHC peripheral base address.
- mask – The interrupt status flags mask(*_usdhc_interrupt_status_flag*).

static inline uint32_t USDHC_GetEnabledInterruptStatusFlags(USDHC_Type *base)

Gets the enabled interrupt status.

Parameters

- base – USDHC peripheral base address.

Returns

Current interrupt status flags mask(*_usdhc_interrupt_status_flag*).

static inline uint32_t USDHC_GetInterruptStatusFlags(USDHC_Type *base)

Gets the current interrupt status.

Parameters

- base – USDHC peripheral base address.

Returns

Current interrupt status flags mask(*_usdhc_interrupt_status_flag*).

static inline void USDHC_ClearInterruptStatusFlags(USDHC_Type *base, uint32_t mask)

Clears a specified interrupt status. write 1 clears.

Parameters

- base – USDHC peripheral base address.
- mask – The interrupt status flags mask(*_usdhc_interrupt_status_flag*).

static inline uint32_t USDHC_GetAutoCommand12ErrorStatusFlags(USDHC_Type *base)

Gets the status of auto command 12 error.

Parameters

- base – USDHC peripheral base address.

Returns

Auto command 12 error status flags mask(*_usdhc_auto_command12_error_status_flag*).

static inline uint32_t USDHC_GetAdmaErrorStatusFlags(USDHC_Type *base)

Gets the status of the ADMA error.

Parameters

- base – USDHC peripheral base address.

Returns
ADMA error status flags mask(*_usdhc_adma_error_status_flag*).

static inline uint32_t USDHC_GetPresentStatusFlags(USDHC_Type *base)

Gets a present status.

This function gets the present USDHC's status except for an interrupt status and an error status.

Parameters

- base – USDHC peripheral base address.

Returns
Present USDHC's status flags mask(*_usdhc_present_status_flag*).

void USDHC_GetCapability(USDHC_Type *base, *usdhc_capability_t* *capability)

Gets the capability information.

Parameters

- base – USDHC peripheral base address.
- capability – Structure to save capability information.

static inline void USDHC_ForceClockOn(USDHC_Type *base, bool enable)

Forces the card clock on.

Parameters

- base – USDHC peripheral base address.
- enable – enable/disable flag

uint32_t USDHC_SetSdClock(USDHC_Type *base, uint32_t srcClock_Hz, uint32_t busClock_Hz)

Sets the SD bus clock frequency.

Parameters

- base – USDHC peripheral base address.
- srcClock_Hz – USDHC source clock frequency united in Hz.
- busClock_Hz – SD bus clock frequency united in Hz.

Returns
The nearest frequency of busClock_Hz configured for SD bus.

bool USDHC_SetCardActive(USDHC_Type *base, uint32_t timeout)

Sends 80 clocks to the card to set it to the active state.

This function must be called each time the card is inserted to ensure that the card can receive the command correctly.

Parameters

- base – USDHC peripheral base address.
- timeout – Timeout to initialize card.

Return values

- true – Set card active successfully.
- false – Set card active failed.

static inline void USDHC_AssertHardwareReset(USDHC_Type *base, bool high)

Triggers a hardware reset.

Parameters

- base – USDHC peripheral base address.
- high – 1 or 0 level

static inline void USDHC_SetDataBusWidth(USDHC_Type *base, *usdhc_data_bus_width_t* width)

Sets the data transfer width.

Parameters

- base – USDHC peripheral base address.
- width – Data transfer width.

static inline void USDHC_WriteData(USDHC_Type *base, uint32_t data)

Fills the data port.

This function is used to implement the data transfer by Data Port instead of DMA.

Parameters

- base – USDHC peripheral base address.
- data – The data about to be sent.

static inline uint32_t USDHC_ReadData(USDHC_Type *base)

Retrieves the data from the data port.

This function is used to implement the data transfer by Data Port instead of DMA.

Parameters

- base – USDHC peripheral base address.

Returns

The data has been read.

void USDHC_SendCommand(USDHC_Type *base, *usdhc_command_t* *command)

Sends command function.

Parameters

- base – USDHC peripheral base address.
- command – configuration

static inline void USDHC_EnableWakeupEvent(USDHC_Type *base, uint32_t mask, bool enable)

Enables or disables a wakeup event in low-power mode.

Parameters

- base – USDHC peripheral base address.
- mask – Wakeup events mask(*_usdhc_wakeup_event*).
- enable – True to enable, false to disable.

static inline void USDHC_CardDetectByData3(USDHC_Type *base, bool enable)

Detects card insert status.

Parameters

- base – USDHC peripheral base address.
- enable – enable/disable flag

static inline bool USDHC_DetectCardInsert(USDHC_Type *base)

Detects card insert status.

Parameters

- base – USDHC peripheral base address.

static inline void USDHC_EnableSdioControl(USDHC_Type *base, uint32_t mask, bool enable)

Enables or disables the SDIO card control.

Parameters

- base – USDHC peripheral base address.
- mask – SDIO card control flags mask(*_usdhc_sdio_control_flag*).
- enable – True to enable, false to disable.

static inline void USDHC_SetContinueRequest(USDHC_Type *base)

Restarts a transaction which has stopped at the block GAP for the SDIO card.

Parameters

- base – USDHC peripheral base address.

static inline void USDHC_RequestStopAtBlockGap(USDHC_Type *base, bool enable)

Request stop at block gap function.

Parameters

- base – USDHC peripheral base address.
- enable – True to stop at block gap, false to normal transfer.

void USDHC_SetMmcBootConfig(USDHC_Type *base, const *usdhc_boot_config_t* *config)

Configures the MMC boot feature.

Example:

```
usdhc_boot_config_t config;
config.ackTimeoutCount = 4;
config.bootMode = kUSDHC_BootModeNormal;
config.blockCount = 5;
config.enableBootAck = true;
config.enableBoot = true;
config.enableAutoStopAtBlockGap = true;
USDHC_SetMmcBootConfig(USDHC, &config);
```

Parameters

- base – USDHC peripheral base address.
- config – The MMC boot configuration information.

static inline void USDHC_EnableMmcBoot(USDHC_Type *base, bool enable)

Enables or disables the mmc boot mode.

Parameters

- base – USDHC peripheral base address.
- enable – True to enable, false to disable.

static inline void USDHC_SetForceEvent(USDHC_Type *base, uint32_t mask)

Forces generating events according to the given mask.

Parameters

- base – USDHC peripheral base address.

- mask – The force events bit posistion (_usdhc_force_event).

static inline bool USDHC_RequestTuningForSDR50(USDHC_Type *base)

Checks the SDR50 mode request tuning bit. When this bit set, application shall perform tuning for SDR50 mode.

Parameters

- base – USDHC peripheral base address.

static inline bool USDHC_RequestReTuning(USDHC_Type *base)

Checks the request re-tuning bit. When this bit is set, user should do manual tuning or standard tuning function.

Parameters

- base – USDHC peripheral base address.

static inline void USDHC_EnableAutoTuning(USDHC_Type *base, bool enable)

The SDR104 mode auto tuning enable and disable. This function should be called after tuning function execute pass, auto tuning will handle by hardware.

Parameters

- base – USDHC peripheral base address.
- enable – enable/disable flag

void USDHC_EnableAutoTuningForCmdAndData(USDHC_Type *base)

The auto tuning enbale for CMD/DATA line.

Parameters

- base – USDHC peripheral base address.

void USDHC_EnableManualTuning(USDHC_Type *base, bool enable)

Manual tuning trigger or abort. User should handle the tuning cmd and find the boundary of the delay then calucate a average value which will be configured to the **CLK_TUNE_CTRL_STATUS** This function should be called before function *USDHC_AdjustDelayForManualTuning*.

Parameters

- base – USDHC peripheral base address.
- enable – tuning enable flag

static inline uint32_t USDHC_GetTuningDelayStatus(USDHC_Type *base)

Get the tuning delay cell setting.

Parameters

- base – USDHC peripheral base address.

Return values

CLK – Tuning Control and Status register value.

status_t USDHC_SetTuningDelay(USDHC_Type *base, uint32_t preDelay, uint32_t outDelay, uint32_t postDelay)

The tuning delay cell setting.

Parameters

- base – USDHC peripheral base address.
- preDelay – Set the number of delay cells on the feedback clock between the feedback clock and CLK_PRE.

- outDelay – Set the number of delay cells on the feedback clock between CLK_PRE and CLK_OUT.
- postDelay – Set the number of delay cells on the feedback clock between CLK_OUT and CLK_POST.

Return values

- kStatus_Fail – config the delay setting fail
- kStatus_Success – config the delay setting success

status_t USDHC_AdjustDelayForManualTuning(USDHC_Type *base, uint32_t delay)

Adjusts delay for mannual tuning.

Deprecated:

Do not use this function. It has been superceded by USDHC_SetTuingDelay

Parameters

- base – USDHC peripheral base address.
- delay – setting configuration

Return values

- kStatus_Fail – config the delay setting fail
- kStatus_Success – config the delay setting success

static inline void USDHC_SetStandardTuningCounter(USDHC_Type *base, uint8_t counter)

set tuning counter tuning.

Parameters

- base – USDHC peripheral base address.
- counter – tuning counter

Return values

- kStatus_Fail – config the delay setting fail
- kStatus_Success – config the delay setting success

void USDHC_EnableStandardTuning(USDHC_Type *base, uint32_t tuningStartTap, uint32_t step, bool enable)

The enable standard tuning function. The standard tuning window and tuning counter using the default config tuning cmd is sent by the software, user need to check whether the tuning result can be used for SDR50, SDR104, and HS200 mode tuning.

Parameters

- base – USDHC peripheral base address.
- tuningStartTap – start tap
- step – tuning step
- enable – enable/disable flag

static inline uint32_t USDHC_GetExecuteStdTuningStatus(USDHC_Type *base)

Gets execute STD tuning status.

Parameters

- base – USDHC peripheral base address.

static inline uint32_t USDHC_CheckStdTuningResult(USDHC_Type *base)

Checks STD tuning result.

Parameters

- base – USDHC peripheral base address.

static inline uint32_t USDHC_CheckTuningError(USDHC_Type *base)

Checks tuning error.

Parameters

- base – USDHC peripheral base address.

void USDHC_EnableDDRMode(USDHC_Type *base, bool enable, uint32_t nibblePos)

The enable/disable DDR mode.

Parameters

- base – USDHC peripheral base address.
- enable – enable/disable flag
- nibblePos – nibble position

static inline void USDHC_EnableHS400Mode(USDHC_Type *base, bool enable)

The enable/disable HS400 mode.

Parameters

- base – USDHC peripheral base address.
- enable – enable/disable flag

static inline void USDHC_ResetStrobeDLL(USDHC_Type *base)

Resets the strobe DLL.

Parameters

- base – USDHC peripheral base address.

static inline void USDHC_EnableStrobeDLL(USDHC_Type *base, bool enable)

Enables/disables the strobe DLL.

Parameters

- base – USDHC peripheral base address.
- enable – enable/disable flag

void USDHC_ConfigStrobeDLL(USDHC_Type *base, uint32_t delayTarget, uint32_t updateInterval)

Configs the strobe DLL delay target and update interval.

Parameters

- base – USDHC peripheral base address.
- delayTarget – delay target
- updateInterval – update interval

static inline void USDHC_SetStrobeDllOverride(USDHC_Type *base, uint32_t delayTaps)

Enables manual override for slave delay chain using **STROBE_SLV_OVERRIDE_VAL**.

Parameters

- base – USDHC peripheral base address.
- delayTaps – Valid delay taps range from 1 - 128 taps. A value of 0 selects tap 1, and a value of 0x7F selects tap 128.

static inline uint32_t USDHC_GetStrobeDLLStatus(USDHC_Type *base)

Gets the strobe DLL status.

Parameters

- base – USDHC peripheral base address.

void USDHC_SetDataConfig(USDHC_Type *base, *usdhc_transfer_direction_t* dataDirection, uint32_t blockCount, uint32_t blockSize)

USDHC data configuration.

Parameters

- base – USDHC peripheral base address.
- dataDirection – Data direction, tx or rx.
- blockCount – Data block count.
- blockSize – Data block size.

void USDHC_TransferCreateHandle(USDHC_Type *base, *usdhc_handle_t* *handle, const *usdhc_transfer_callback_t* *callback, void *userData)

Creates the USDHC handle.

Parameters

- base – USDHC peripheral base address.
- handle – USDHC handle pointer.
- callback – Structure pointer to contain all callback functions.
- userData – Callback function parameter.

status_t USDHC_TransferNonBlocking(USDHC_Type *base, *usdhc_handle_t* *handle, *usdhc_adma_config_t* *dmaConfig, *usdhc_transfer_t* *transfer)

Transfers the command/data using an interrupt and an asynchronous method.

This function sends a command and data and returns immediately. It doesn't wait for the transfer to complete or to encounter an error. The application must not call this API in multiple threads at the same time. Because of that this API doesn't support the re-entry mechanism.

Note: Call API *USDHC_TransferCreateHandle* when calling this API.

Parameters

- base – USDHC peripheral base address.
- handle – USDHC handle.
- dmaConfig – ADMA configuration.
- transfer – Transfer content.

Return values

- kStatus_InvalidArgument – Argument is invalid.
- kStatus_USDHC_BusyTransferring – Busy transferring.
- kStatus_USDHC_PrepareAdmaDescriptorFailed – Prepare ADMA descriptor failed.
- kStatus_Success – Operate successfully.

status_t USDHC_TransferBlocking(USDHC_Type *base, *usdhc_adma_config_t* *dmaConfig, *usdhc_transfer_t* *transfer)

Transfers the command/data using a blocking method.

This function waits until the command response/data is received or the USDHC encounters an error by polling the status flag.

The application must not call this API in multiple threads at the same time. Because this API doesn't support the re-entry mechanism.

Note: There is no need to call API *USDHC_TransferCreateHandle* when calling this API.

Parameters

- base – USDHC peripheral base address.
- dmaConfig – adma configuration
- transfer – Transfer content.

Return values

- kStatus_InvalidArgument – Argument is invalid.
- kStatus_USDHC_PrepareAdmaDescriptorFailed – Prepare ADMA descriptor failed.
- kStatus_USDHC_SendCommandFailed – Send command failed.
- kStatus_USDHC_TransferDataFailed – Transfer data failed.
- kStatus_Success – Operate successfully.

void USDHC_TransferHandleIRQ(USDHC_Type *base, *usdhc_handle_t* *handle)

IRQ handler for the USDHC.

This function deals with the IRQs on the given host controller.

Parameters

- base – USDHC peripheral base address.
- handle – USDHC handle.

void USDHC_DriverIRQHandler(uint32_t instance)

FSL_USDHC_DRIVER_VERSION

Driver version.

Enum _usdhc_status. USDHC status.

Values:

enumerator kStatus_USDHC_BusyTransferring

Transfer is on-going.

enumerator kStatus_USDHC_PrepareAdmaDescriptorFailed

Set DMA descriptor failed.

enumerator kStatus_USDHC_SendCommandFailed
Send command failed.

enumerator kStatus_USDHC_TransferDataFailed
Transfer data failed.

enumerator kStatus_USDHC_DMADataAddrNotAlign
Data address not aligned.

enumerator kStatus_USDHC_ReTuningRequest
Re-tuning request.

enumerator kStatus_USDHC_TuningError
Tuning error.

enumerator kStatus_USDHC_NotSupport
Not support.

enumerator kStatus_USDHC_TransferDataComplete
Transfer data complete.

enumerator kStatus_USDHC_SendCommandSuccess
Transfer command complete.

enumerator kStatus_USDHC_TransferDMAComplete
Transfer DMA complete.

Enum _usdhc_capability_flag. Host controller capabilities flag mask. .

Values:

enumerator kUSDHC_SupportAdmaFlag
Support ADMA.

enumerator kUSDHC_SupportHighSpeedFlag
Support high-speed.

enumerator kUSDHC_SupportDmaFlag
Support DMA.

enumerator kUSDHC_SupportSuspendResumeFlag
Support suspend/resume.

enumerator kUSDHC_SupportV330Flag
Support voltage 3.3V.

enumerator kUSDHC_SupportV300Flag
Support voltage 3.0V.

enumerator kUSDHC_Support4BitFlag
Flag in HTCAPBLT_MBL's position, supporting 4-bit mode.

enumerator kUSDHC_Support8BitFlag
Flag in HTCAPBLT_MBL's position, supporting 8-bit mode.

enumerator kUSDHC_SupportDDR50Flag
SD version 3.0 new feature, supporting DDR50 mode.

enumerator kUSDHC_SupportSDR104Flag
Support SDR104 mode.

enumerator kUSDHC_SupportSDR50Flag
Support SDR50 mode.

Enum _usdhc_wakeup_event. Wakeup event mask. .

Values:

enumerator kUSDHC_WakeupEventOnCardInt
Wakeup on card interrupt.

enumerator kUSDHC_WakeupEventOnCardInsert
Wakeup on card insertion.

enumerator kUSDHC_WakeupEventOnCardRemove
Wakeup on card removal.

enumerator kUSDHC_WakeupEventsAll
All wakeup events

Enum _usdhc_reset. Reset type mask. .

Values:

enumerator kUSDHC_ResetAll
Reset all except card detection.

enumerator kUSDHC_ResetCommand
Reset command line.

enumerator kUSDHC_ResetData
Reset data line.

enumerator kUSDHC_ResetTuning
Reset tuning circuit.

enumerator kUSDHC_ResetsAll
All reset types

Enum _usdhc_transfer_flag. Transfer flag mask.
Values:

enumerator kUSDHC_EnableDmaFlag
Enable DMA.

enumerator kUSDHC_CommandTypeSuspendFlag
Suspend command.

enumerator kUSDHC_CommandTypeResumeFlag
Resume command.

enumerator kUSDHC_CommandTypeAbortFlag
Abort command.

enumerator kUSDHC_EnableBlockCountFlag
Enable block count.

enumerator kUSDHC_EnableAutoCommand12Flag
Enable auto CMD12.

enumerator kUSDHC_DataReadFlag
Enable data read.

enumerator kUSDHC_MultipleBlockFlag
Multiple block data read/write.

enumerator kUSDHC_EnableAutoCommand23Flag
Enable auto CMD23.

enumerator kUSDHC_ResponseLength136Flag
136-bit response length.

enumerator kUSDHC_ResponseLength48Flag
48-bit response length.

enumerator kUSDHC_ResponseLength48BusyFlag
48-bit response length with busy status.

enumerator kUSDHC_EnableCrcCheckFlag
Enable CRC check.

enumerator kUSDHC_EnableIndexCheckFlag
Enable index check.

enumerator kUSDHC_DataPresentFlag
Data present flag.

Enum _usdhc_present_status_flag. Present status flag mask. .

Values:

enumerator kUSDHC_CommandInhibitFlag
Command inhibit.

enumerator kUSDHC_DataInhibitFlag
Data inhibit.

enumerator kUSDHC_DataLineActiveFlag
Data line active.

enumerator kUSDHC_SdClockStableFlag
SD bus clock stable.

enumerator kUSDHC_WriteTransferActiveFlag
Write transfer active.

enumerator kUSDHC_ReadTransferActiveFlag
Read transfer active.

enumerator kUSDHC_BufferWriteEnableFlag
Buffer write enable.

enumerator kUSDHC_BufferReadEnableFlag
Buffer read enable.

enumerator kUSDHC_ReTuningRequestFlag
Re-tuning request flag, only used for SDR104 mode.

enumerator kUSDHC_DelaySettingFinishedFlag
Delay setting finished flag.

enumerator kUSDHC_CardInsertedFlag
Card inserted.

enumerator kUSDHC_CommandLineLevelFlag
Command line signal level.

enumerator kUSDHC_Data0LineLevelFlag
Data0 line signal level.

enumerator kUSDHC_Data1LineLevelFlag
Data1 line signal level.

enumerator kUSDHC_Data2LineLevelFlag
Data2 line signal level.

enumerator kUSDHC_Data3LineLevelFlag
Data3 line signal level.

enumerator kUSDHC_Data4LineLevelFlag
Data4 line signal level.

enumerator kUSDHC_Data5LineLevelFlag
Data5 line signal level.

enumerator kUSDHC_Data6LineLevelFlag
Data6 line signal level.

enumerator kUSDHC_Data7LineLevelFlag
Data7 line signal level.

Enum _usdhc_interrupt_status_flag. Interrupt status flag mask. .

Values:

enumerator kUSDHC_CommandCompleteFlag
Command complete.

enumerator kUSDHC_DataCompleteFlag
Data complete.

enumerator kUSDHC_BlockGapEventFlag
Block gap event.

enumerator kUSDHC_DmaCompleteFlag
DMA interrupt.

enumerator kUSDHC_BufferWriteReadyFlag
Buffer write ready.

enumerator kUSDHC_BufferReadReadyFlag
Buffer read ready.

enumerator kUSDHC_CardInsertionFlag
Card inserted.

enumerator kUSDHC_CardRemovalFlag
Card removed.

enumerator kUSDHC_CardInterruptFlag
Card interrupt.

enumerator kUSDHC_ReTuningEventFlag
Re-Tuning event, only for SD3.0 SDR104 mode.

enumerator kUSDHC_TuningPassFlag
SDR104 mode tuning pass flag.

enumerator kUSDHC_TuningErrorFlag
SDR104 tuning error flag.

enumerator kUSDHC_CommandTimeoutFlag
Command timeout error.

enumerator kUSDHC_CommandCrcErrorFlag
Command CRC error.

enumerator kUSDHC_CommandEndBitErrorFlag
Command end bit error.

enumerator kUSDHC_CommandIndexErrorFlag
Command index error.

enumerator kUSDHC_DataTimeoutFlag
Data timeout error.

enumerator kUSDHC_DataCrcErrorFlag
Data CRC error.

enumerator kUSDHC_DataEndBitErrorFlag
Data end bit error.

enumerator kUSDHC_AutoCommand12ErrorFlag
Auto CMD12 error.

enumerator kUSDHC_DmaErrorFlag
DMA error.

enumerator kUSDHC_CommandErrorFlag
Command error

enumerator kUSDHC_DataErrorFlag
Data error

enumerator kUSDHC_ErrorFlag
All error

enumerator kUSDHC_DataFlag
Data interrupts

enumerator kUSDHC_DataDMAFlag
Data interrupts

enumerator kUSDHC_CommandFlag
Command interrupts

enumerator kUSDHC_CardDetectFlag
Card detection interrupts

enumerator kUSDHC_SDR104TuningFlag
SDR104 tuning flag.

enumerator kUSDHC_AllInterruptFlags
All flags mask

Enum _usdhc_auto_command12_error_status_flag. Auto CMD12 error status flag mask. .
Values:

enumerator kUSDHC_AutoCommand12NotExecutedFlag
Not executed error.

enumerator kUSDHC_AutoCommand12TimeoutFlag
Timeout error.

enumerator kUSDHC_AutoCommand12EndBitErrorFlag
End bit error.

enumerator kUSDHC_AutoCommand12CrcErrorFlag
CRC error.

enumerator kUSDHC_AutoCommand12IndexErrorFlag
Index error.

enumerator kUSDHC_AutoCommand12NotIssuedFlag
Not issued error.

Enum _usdhc_standard_tuning. Standard tuning flag.
Values:

enumerator kUSDHC_ExecuteTuning
Used to start tuning procedure.

enumerator kUSDHC_TuningSampleClockSel
When **std_tuning_en** bit is set, this bit is used to select sampleing clock.

Enum _usdhc_adma_error_status_flag. ADMA error status flag mask. .

Values:

enumerator kUSDHC_AdmaLenghMismatchFlag
Length mismatch error.

enumerator kUSDHC_AdmaDescriptorErrorFlag
Descriptor error.

Enum _usdhc_adma_error_state. ADMA error state.

This state is the detail state when ADMA error has occurred.

Values:

enumerator kUSDHC_AdmaErrorStateStopDma
Stop DMA, previous location set in the ADMA system address is errored address.

enumerator kUSDHC_AdmaErrorStateFetchDescriptor
Fetch descriptor, current location set in the ADMA system address is errored address.

enumerator kUSDHC_AdmaErrorStateChangeAddress
Change address, no DMA error has occurred.

enumerator kUSDHC_AdmaErrorStateTransferData
Transfer data, previous location set in the ADMA system address is errored address.

enumerator kUSDHC_AdmaErrorStateInvalidLength
Invalid length in ADMA descriptor.

enumerator kUSDHC_AdmaErrorStateInvalidDescriptor
Invalid descriptor fetched by ADMA.

enumerator kUSDHC_AdmaErrorState
ADMA error state

Enum _usdhc_force_event. Force event bit position. .

Values:

enumerator kUSDHC_ForceEventAutoCommand12NotExecuted
Auto CMD12 not executed error.

enumerator kUSDHC_ForceEventAutoCommand12Timeout
Auto CMD12 timeout error.

enumerator kUSDHC_ForceEventAutoCommand12CrcError
Auto CMD12 CRC error.

enumerator kUSDHC_ForceEventEndBitError
Auto CMD12 end bit error.

enumerator kUSDHC_ForceEventAutoCommand12IndexError
Auto CMD12 index error.

enumerator kUSDHC_ForceEventAutoCommand12NotIssued
Auto CMD12 not issued error.

enumerator kUSDHC_ForceEventCommandTimeout
Command timeout error.

enumerator kUSDHC_ForceEventCommandCrcError
Command CRC error.

enumerator kUSDHC_ForceEventCommandEndBitError
Command end bit error.

enumerator kUSDHC_ForceEventCommandIndexError
Command index error.

enumerator kUSDHC_ForceEventDataTimeout
Data timeout error.

enumerator kUSDHC_ForceEventDataCrcError
Data CRC error.

enumerator kUSDHC_ForceEventDataEndBitError
Data end bit error.

enumerator kUSDHC_ForceEventAutoCommand12Error
Auto CMD12 error.

enumerator kUSDHC_ForceEventCardInt
Card interrupt.

enumerator kUSDHC_ForceEventDmaError
Dma error.

enumerator kUSDHC_ForceEventTuningError
Tuning error.

enumerator kUSDHC_ForceEventsAll
All force event flags mask.

enum _usdhc_transfer_direction
Data transfer direction.

Values:

enumerator kUSDHC_TransferDirectionReceive
USDHC transfer direction receive.

enumerator kUSDHC_TransferDirectionSend
USDHC transfer direction send.

enum _usdhc_data_bus_width
Data transfer width.
Values:

enumerator kUSDHC_DataBusWidth1Bit
1-bit mode

enumerator kUSDHC_DataBusWidth4Bit
4-bit mode

enumerator kUSDHC_DataBusWidth8Bit
8-bit mode

enum _usdhc_endian_mode
Endian mode.
Values:

enumerator kUSDHC_EndianModeBig
Big endian mode.

enumerator kUSDHC_EndianModeHalfWordBig
Half word big endian mode.

enumerator kUSDHC_EndianModeLittle
Little endian mode.

enum _usdhc_dma_mode
DMA mode.
Values:

enumerator kUSDHC_DmaModeSimple
External DMA.

enumerator kUSDHC_DmaModeAdma1
ADMA1 is selected.

enumerator kUSDHC_DmaModeAdma2
ADMA2 is selected.

enumerator kUSDHC_ExternalDMA
External DMA mode selected.

Enum _usdhc_sdio_control_flag. SDIO control flag mask. .

Values:

enumerator kUSDHC_StopAtBlockGapFlag
Stop at block gap.

enumerator kUSDHC_ReadWaitControlFlag
Read wait control.

enumerator kUSDHC_InterruptAtBlockGapFlag
Interrupt at block gap.

enumerator kUSDHC_ReadDoneNo8CLK
Read done without 8 clk for block gap.

enumerator kUSDHC_ExactBlockNumberReadFlag
Exact block number read.

enum _usdhc_boot_mode
MMC card boot mode.

Values:

enumerator kUSDHC_BootModeNormal
Normal boot

enumerator kUSDHC_BootModeAlternative
Alternative boot

enum _usdhc_card_command_type
The command type.

Values:

enumerator kCARD_CommandTypeNormal
Normal command

enumerator kCARD_CommandTypeSuspend
Suspend command

enumerator kCARD_CommandTypeResume
Resume command

enumerator kCARD_CommandTypeAbort
Abort command

enumerator kCARD_CommandTypeEmpty
Empty command

enum _usdhc_card_response_type

The command response type.

Defines the command response type from card to host controller.

Values:

enumerator kCARD_ResponseTypeNone

Response type: none

enumerator kCARD_ResponseTypeR1

Response type: R1

enumerator kCARD_ResponseTypeR1b

Response type: R1b

enumerator kCARD_ResponseTypeR2

Response type: R2

enumerator kCARD_ResponseTypeR3

Response type: R3

enumerator kCARD_ResponseTypeR4

Response type: R4

enumerator kCARD_ResponseTypeR5

Response type: R5

enumerator kCARD_ResponseTypeR5b

Response type: R5b

enumerator kCARD_ResponseTypeR6

Response type: R6

enumerator kCARD_ResponseTypeR7

Response type: R7

Enum _usdhc_adma1_descriptor_flag. The mask for the control/status field in ADMA1 descriptor.

Values:

enumerator kUSDHC_Adma1DescriptorValidFlag

Valid flag.

enumerator kUSDHC_Adma1DescriptorEndFlag

End flag.

enumerator kUSDHC_Adma1DescriptorInterrupFlag

Interrupt flag.

enumerator kUSDHC_Adma1DescriptorActivity1Flag
Activity 1 flag.

enumerator kUSDHC_Adma1DescriptorActivity2Flag
Activity 2 flag.

enumerator kUSDHC_Adma1DescriptorTypeNop
No operation.

enumerator kUSDHC_Adma1DescriptorTypeTransfer
Transfer data.

enumerator kUSDHC_Adma1DescriptorTypeLink
Link descriptor.

enumerator kUSDHC_Adma1DescriptorTypeSetLength
Set data length.

Enum _usdhc_adma2_descriptor_flag. ADMA1 descriptor control and status mask.

Values:

enumerator kUSDHC_Adma2DescriptorValidFlag
Valid flag.

enumerator kUSDHC_Adma2DescriptorEndFlag
End flag.

enumerator kUSDHC_Adma2DescriptorInterruptFlag
Interrupt flag.

enumerator kUSDHC_Adma2DescriptorActivity1Flag
Activity 1 mask.

enumerator kUSDHC_Adma2DescriptorActivity2Flag
Activity 2 mask.

enumerator kUSDHC_Adma2DescriptorTypeNop
No operation.

enumerator kUSDHC_Adma2DescriptorTypeReserved
Reserved.

enumerator kUSDHC_Adma2DescriptorTypeTransfer
Transfer type.

enumerator kUSDHC_Adma2DescriptorTypeLink
Link type.

Enum _usdhc_adma_flag. ADMA descriptor configuration flag. .

Values:

enumerator kUSDHC_AdmaDescriptorSingleFlag

Try to finish the transfer in a single ADMA descriptor. If transfer size is bigger than one ADMA descriptor's ability, new another descriptor for data transfer.

enumerator kUSDHC_AdmaDescriptorMultipleFlag

Create multiple ADMA descriptors within the ADMA table, this is used for mmc boot mode specifically, which need to modify the ADMA descriptor on the fly, so the flag should be used combining with stop at block gap feature.

enum _usdhc_burst_len

DMA transfer burst len config.

Values:

enumerator kUSDHC_EnBurstLenForINCR

Enable burst len for INCR.

enumerator kUSDHC_EnBurstLenForINCR4816

Enable burst len for INCR4/INCR8/INCR16.

enumerator kUSDHC_EnBurstLenForINCR4816WRAP

Enable burst len for INCR4/8/16 WRAP.

Enum _usdhc_transfer_data_type. Tansfer data type definition.

Values:

enumerator kUSDHC_TransferDataNormal

Transfer normal read/write data.

enumerator kUSDHC_TransferDataTuning

Transfer tuning data.

enumerator kUSDHC_TransferDataBoot

Transfer boot data.

enumerator kUSDHC_TransferDataBootcontinous

Transfer boot data continuously.

typedef enum *_usdhc_transfer_direction* usdhc_transfer_direction_t

Data transfer direction.

typedef enum *_usdhc_data_bus_width* usdhc_data_bus_width_t

Data transfer width.

typedef enum *_usdhc_endian_mode* usdhc_endian_mode_t

Endian mode.

typedef enum *_usdhc_dma_mode* usdhc_dma_mode_t

DMA mode.

typedef enum *_usdhc_boot_mode* usdhc_boot_mode_t

MMC card boot mode.

typedef enum *_usdhc_card_command_type* usdhc_card_command_type_t

The command type.

typedef enum *_usdhc_card_response_type* usdhc_card_response_type_t

The command response type.

Defines the command response type from card to host controller.

typedef enum *_usdhc_burst_len* usdhc_burst_len_t

DMA transfer burst len config.

typedef uint32_t usdhc_adma1_descriptor_t

Defines the ADMA1 descriptor structure.

typedef struct *_usdhc_adma2_descriptor* usdhc_adma2_descriptor_t

Defines the ADMA2 descriptor structure.

typedef struct *_usdhc_capability* usdhc_capability_t

USDHC capability information.

Defines a structure to save the capability information of USDHC.

typedef struct *_usdhc_boot_config* usdhc_boot_config_t

Data structure to configure the MMC boot feature.

typedef struct *_usdhc_config* usdhc_config_t

Data structure to initialize the USDHC.

typedef struct *_usdhc_command* usdhc_command_t

Card command descriptor.

Defines card command-related attribute.

typedef struct *_usdhc_adma_config* usdhc_adma_config_t

ADMA configuration.

typedef struct *_usdhc_scatter_gather_data_list* usdhc_scatter_gather_data_list_t

Card scatter gather data list.

Allow application register uncontinuous data buffer for data transfer.

typedef struct *_usdhc_scatter_gather_data* usdhc_scatter_gather_data_t

Card scatter gather data descriptor.

Defines a structure to contain data-related attribute. The 'enableIgnoreError' is used when upper card driver wants to ignore the error event to read/write all the data and not to stop read/write immediately when an error event happens. For example, bus testing procedure for MMC card.

typedef struct *_usdhc_scatter_gather_transfer* usdhc_scatter_gather_transfer_t

usdhc scatter gather transfer.

typedef struct *_usdhc_data* usdhc_data_t

Card data descriptor.

Defines a structure to contain data-related attribute. The 'enableIgnoreError' is used when upper card driver wants to ignore the error event to read/write all the data and not to stop read/write immediately when an error event happens. For example, bus testing procedure for MMC card.

typedef struct *_usdhc_transfer* usdhc_transfer_t

Transfer state.

typedef struct *_usdhc_handle* usdhc_handle_t

USDHC handle typedef.

typedef struct *_usdhc_transfer_callback* usdhc_transfer_callback_t

USDHC callback functions.

typedef *status_t* (*usdhc_transfer_function_t)(USDHC_Type *base, *usdhc_transfer_t* *content)

USDHC transfer function.

typedef struct *_usdhc_host* usdhc_host_t

USDHC host descriptor.

USDHC_MAX_BLOCK_COUNT

Maximum block count can be set one time.

FSL_USDHC_ENABLE_SCATTER_GATHER_TRANSFER

USDHC scatter gather feature control macro.

USDHC_ADMA1_ADDRESS_ALIGN

The alignment size for ADDRESS filed in ADMA1's descriptor.

USDHC_ADMA1_LENGTH_ALIGN

The alignment size for LENGTH field in ADMA1's descriptor.

USDHC_ADMA2_ADDRESS_ALIGN

The alignment size for ADDRESS field in ADMA2's descriptor.

USDHC_ADMA2_LENGTH_ALIGN

The alignment size for LENGTH filed in ADMA2's descriptor.

USDHC_ADMA1_DESCRIPTOR_ADDRESS_SHIFT

The bit shift for ADDRESS filed in ADMA1's descriptor.

Address/page field	Reserved	Attribute					
31 12	11 6	05	04	03	02	01	00
address or data length	000000	Act2	Act1	0	Int	End	Valid

Act2	Act1	Comment	31-28	27-12
0	0	No op	Don't care	
0	1	Set data length	0000	Data Length
1	0	Transfer data	Data address	
1	1	Link descriptor	Descriptor address	

USDHC_ADMA1_DESCRIPTOR_ADDRESS_MASK

The bit mask for ADDRESS field in ADMA1's descriptor.

USDHC_ADMA1_DESCRIPTOR_LENGTH_SHIFT

The bit shift for LENGTH filed in ADMA1's descriptor.

USDHC_ADMA1_DESCRIPTOR_LENGTH_MASK

The mask for LENGTH field in ADMA1's descriptor.

USDHC_ADMA1_DESCRIPTOR_MAX_LENGTH_PER_ENTRY

The maximum value of LENGTH filed in ADMA1's descriptor. Since the max transfer size ADMA1 support is 65535 which is indivisible by 4096, so to make sure a large data load transfer (>64KB) continuously (require the data address be always align with 4096), software will set the maximum data length for ADMA1 to (64 - 4)KB.

USDHC_ADMA2_DESCRIPTOR_LENGTH_SHIFT

The bit shift for LENGTH field in ADMA2's descriptor.

Address field	Length	Reserved	Attribute					
63 32	31 16	15 06	05	04	03	02	01	00
32-bit address	16-bit length	0000000000	Act2	Act1	0	Int	End	Valid

Act2	Act1	Comment	Operation
0	0	No op	Don't care
0	1	Reserved	Read this line and go to next one
1	0	Transfer data	Transfer data with address and length set in this descriptor line
1	1	Link descriptor	Link to another descriptor

USDHC_ADMA2_DESCRIPTOR_LENGTH_MASK
The bit mask for LENGTH field in ADMA2's descriptor.

USDHC_ADMA2_DESCRIPTOR_MAX_LENGTH_PER_ENTRY
The maximum value of LENGTH field in ADMA2's descriptor.

struct _usdhc_adma2_descriptor
#include <fsl_usdhc.h> Defines the ADMA2 descriptor structure.

Public Members

uint32_t attribute
The control and status field.

uint32_t address
The address field.

struct _usdhc_capability
#include <fsl_usdhc.h> USDHC capability information.
Defines a structure to save the capability information of USDHC.

Public Members

uint32_t sdVersion
Support SD card/sdio version.

uint32_t mmcVersion
Support EMMC card version.

uint32_t maxBlockLength
Maximum block length united as byte.

uint32_t maxBlockCount
Maximum block count can be set one time.

uint32_t flags
Capability flags to indicate the support information(*_usdhc_capability_flag*).

struct _usdhc_boot_config
#include <fsl_usdhc.h> Data structure to configure the MMC boot feature.

Public Members

uint32_t ackTimeoutCount
Timeout value for the boot ACK. The available range is 0 ~ 15.

usdhc_boot_mode_t bootMode
Boot mode selection.

uint32_t blockCount
Stop at block gap value of automatic mode. Available range is 0 ~ 65535.

size_t blockSize
Block size.

bool enableBootAck
Enable or disable boot ACK.

bool enableAutoStopAtBlockGap
Enable or disable auto stop at block gap function in boot period.

struct _usdhc_config
#include <fsl_usdhc.h> Data structure to initialize the USDHC.

Public Members

uint32_t dataTimeout
Data timeout value.

usdhc_endian_mode_t endianMode
Endian mode.

uint8_t readWatermarkLevel
Watermark level for DMA read operation. Available range is 1 ~ 128.

uint8_t writeWatermarkLevel
Watermark level for DMA write operation. Available range is 1 ~ 128.

struct _usdhc_command
#include <fsl_usdhc.h> Card command descriptor.
Defines card command-related attribute.

Public Members

uint32_t index
Command index.

uint32_t argument
Command argument.

usdhc_card_command_type_t type
Command type.

usdhc_card_response_type_t responseType
Command response type.

uint32_t response[4U]
Response for this command.

uint32_t responseErrorFlags
Response error flag, which need to check the command reponse.

uint32_t flags
Cmd flags.

struct _usdhc_adma_config
#include <fsl_usdhc.h> ADMA configuration.

Public Members

usdhc_dma_mode_t dmaMode
DMA mode.

uint32_t *admaTable
ADMA table address, can't be null if transfer way is ADMA1/ADMA2.

uint32_t admaTableWords
ADMA table length united as words, can't be 0 if transfer way is ADMA1/ADMA2.

struct _usdhc_scatter_gather_data_list
#include <fsl_usdhc.h> Card scatter gather data list.
Allow application register uncontinuous data buffer for data transfer.

struct _usdhc_scatter_gather_data
#include <fsl_usdhc.h> Card scatter gather data descriptor.
Defines a structure to contain data-related attribute. The 'enableIgnoreError' is used when upper card driver wants to ignore the error event to read/write all the data and not to stop read/write immediately when an error event happens. For example, bus testing procedure for MMC card.

Public Members

bool enableAutoCommand12
Enable auto CMD12.

bool enableAutoCommand23
Enable auto CMD23.

bool enableIgnoreError

Enable to ignore error event to read/write all the data.

usdhc_transfer_direction_t dataDirection

data direction

uint8_t dataType

this is used to distinguish the normal/tuning/boot data.

size_t blockSize

Block size.

usdhc_scatter_gather_data_list_t sgData

scatter gather data

struct __usdhc__scatter__gather__transfer

#include <fsl_usdhc.h> usdhc scatter gather transfer.

Public Members

usdhc_scatter_gather_data_t *data

Data to transfer.

usdhc_command_t *command

Command to send.

struct __usdhc__data

#include <fsl_usdhc.h> Card data descriptor.

Defines a structure to contain data-related attribute. The 'enableIgnoreError' is used when upper card driver wants to ignore the error event to read/write all the data and not to stop read/write immediately when an error event happens. For example, bus testing procedure for MMC card.

Public Members

bool enableAutoCommand12

Enable auto CMD12.

bool enableAutoCommand23

Enable auto CMD23.

bool enableIgnoreError

Enable to ignore error event to read/write all the data.

uint8_t dataType

this is used to distinguish the normal/tuning/boot data.

size_t blockSize
Block size.

uint32_t blockCount
Block count.

uint32_t *rxData
Buffer to save data read.

const uint32_t *txData
Data buffer to write.

struct _usdhc_transfer
#include <fsl_usdhc.h> Transfer state.

Public Members

usdhc_data_t *data
Data to transfer.

usdhc_command_t *command
Command to send.

struct _usdhc_transfer_callback
#include <fsl_usdhc.h> USDHC callback functions.

Public Members

void (*CardInserted)(USDHC_Type *base, void *userData)
Card inserted occurs when DAT3/CD pin is for card detect

void (*CardRemoved)(USDHC_Type *base, void *userData)
Card removed occurs

void (*SdioInterrupt)(USDHC_Type *base, void *userData)
SDIO card interrupt occurs

void (*BlockGap)(USDHC_Type *base, void *userData)
stopped at block gap event

void (*TransferComplete)(USDHC_Type *base, *usdhc_handle_t* *handle, *status_t* status, void *userData)
Transfer complete callback.

void (*ReTuning)(USDHC_Type *base, void *userData)
Handle the re-tuning.

struct __usdhc__handle

#include <fsl_usdhc.h> USDHC handle.

Defines the structure to save the USDHC state information and callback function.

Note: All the fields except interruptFlags and transferredWords must be allocated by the user.

Public Members

usdhc_data_t *volatile data

Transfer parameter. Data to transfer.

usdhc_command_t *volatile command

Transfer parameter. Command to send.

volatile uint32_t transferredWords

Transfer status. Words transferred by DATAPORT way.

usdhc_transfer_callback_t callback

Callback function.

void *userData

Parameter for transfer complete callback.

bool enDMA

Transfer parameter. Was DMA used?

struct __usdhc__host

#include <fsl_usdhc.h> USDHC host descriptor.

Public Members

USDHC_Type *base

USDHC peripheral base address.

uint32_t sourceClock__Hz

USDHC source clock frequency united in Hz.

usdhc_config_t config

USDHC configuration.

usdhc_capability_t capability

USDHC capability information.

usdhc_transfer_function_t transfer

USDHC transfer function.

2.69 WDOG: Watchdog Timer Driver

void WDOG_GetDefaultConfig(*wdog_config_t* *config)

Initializes the WDOG configuration structure.

This function initializes the WDOG configuration structure to default values. The default values are as follows.

```
wdogConfig->enableWdog = true;
wdogConfig->workMode.enableWait = true;
wdogConfig->workMode.enableStop = true;
wdogConfig->workMode.enableDebug = true;
wdogConfig->enableInterrupt = false;
wdogConfig->enablePowerdown = false;
wdogConfig->resetExtension = flase;
wdogConfig->timeoutValue = 0xFFU;
wdogConfig->interruptTimeValue = 0x04u;
wdogConfig->enableTimeOutAssert = false;
```

See also:

wdog_config_t

Note: On devices affected by errata ERR051322, enableTimeOutAssert defaults to true as a workaround. The ROM does not configure the WDOG_B pin to assert on timeout even when the WDOG_B_PIN_EN fuse is blown. Setting WCR[WDT] from application code compensates for this missing ROM behavior.

Parameters

- config – Pointer to the WDOG configuration structure.

void WDOG_Init(WDOG_Type *base, const *wdog_config_t* *config)

Initializes the WDOG.

This function initializes the WDOG. When called, the WDOG runs according to the configuration.

This is an example.

```
wdog_config_t config;
WDOG_GetDefaultConfig(&config);
config.timeoutValue = 0xffU;
config->interruptTimeValue = 0x04u;
WDOG_Init(wdog_base,&config);
```

Parameters

- base – WDOG peripheral base address
- config – The configuration of WDOG

void WDOG_Deinit(WDOG_Type *base)

Shuts down the WDOG.

This function shuts down the WDOG. Watchdog Enable bit is a write one once only bit. It is not possible to clear this bit by a software write, once the bit is set. This bit(WDE) can be set/reset only in debug mode(exception).

static inline void WDOG_Enable(WDOG_Type *base)

Enables the WDOG module.

This function writes a value into the WDOG_WCR register to enable the WDOG. This is a write one once only bit. It is not possible to clear this bit by a software write, once the bit is set. only debug mode exception.

Parameters

- base – WDOG peripheral base address

static inline void WDOG_Disable(WDOG_Type *base)

Disables the WDOG module.

This function writes a value into the WDOG_WCR register to disable the WDOG. This is a write one once only bit. It is not possible to clear this bit by a software write,once the bit is set. only debug mode exception

Parameters

- base – WDOG peripheral base address

static inline void WDOG_TriggerSystemSoftwareReset(WDOG_Type *base)

Trigger the system software reset.

This function will write to the WCR[SRS] bit to trigger a software system reset. This bit will automatically resets to "1" after it has been asserted to "0". Note: Calling this API will reset the system right now, please using it with more attention.

Parameters

- base – WDOG peripheral base address

static inline void WDOG_TriggerSoftwareSignal(WDOG_Type *base)

Trigger an output assertion.

This function will write to the WCR[WDA] bit to trigger WDOG_B signal assertion. The WDOG_B signal can be routed to external pin of the chip, the output pin will turn to assertion along with WDOG_B signal. Note: The WDOG_B signal will remain assert until a power on reset occurred, so, please take more attention while calling it.

Parameters

- base – WDOG peripheral base address

static inline void WDOG_EnableInterrupts(WDOG_Type *base, uint16_t mask)

Enables the WDOG interrupt.

This bit is a write once only bit. Once the software does a write access to this bit, it will get locked and cannot be reprogrammed until the next system reset assertion

Parameters

- base – WDOG peripheral base address
- mask – The interrupts to enable The parameter can be combination of the following source if defined.
  - kWDOG_InterruptEnable

uint16_t WDOG_GetStatusFlags(WDOG_Type *base)

Gets the WDOG all reset status flags.

This function gets all reset status flags.

```
uint16_t status;
status = WDOG_GetStatusFlags (wdog_base);
```

See also:

_wdog_status_flags

- true: a related status flag has been set.
- false: a related status flag is not set.

Parameters

- base – WDOG peripheral base address

Returns

State of the status flag: asserted (true) or not-asserted (false).

void WDOG_ClearInterruptStatus(WDOG_Type *base, uint16_t mask)

Clears the WDOG flag.

This function clears the WDOG status flag.

This is an example for clearing the interrupt flag.

```
WDOG_ClearStatusFlags(wdog_base,KWDOG_InterruptFlag);
```

Parameters

- base – WDOG peripheral base address
- mask – The status flags to clear. The parameter could be any combination of the following values. kWDOG_TimeoutFlag

static inline void WDOG_SetTimeoutValue(WDOG_Type *base, uint16_t timeoutCount)

Sets the WDOG timeout value.

This function sets the timeout value. This function writes a value into WCR registers. The time-out value can be written at any point of time but it is loaded to the counter at the time when WDOG is enabled or after the service routine has been performed.

Parameters

- base – WDOG peripheral base address
- timeoutCount – WDOG timeout value; count of WDOG clock tick.

static inline void WDOG_SetInterrputTimeoutValue(WDOG_Type *base, uint16_t timeoutCount)

Sets the WDOG interrupt count timeout value.

This function sets the interrupt count timeout value. This function writes a value into WIC registers which are wirte-once. This field is write once only. Once the software does a write access to this field, it will get locked and cannot be reprogrammed until the next system reset assertion.

Parameters

- base – WDOG peripheral base address
- timeoutCount – WDOG timeout value; count of WDOG clock tick.

static inline void WDOG_DisablePowerDownEnable(WDOG_Type *base)

Disable the WDOG power down enable bit.

This function disable the WDOG power down enable(PDE). This function writes a value into WMCR registers which are wirte-once. This field is write once only. Once software sets this bit it cannot be reset until the next system reset.

Parameters

- base – WDOG peripheral base address

void WDOG_Refresh(WDOG_Type *base)

Refreshes the WDOG timer.

This function feeds the WDOG. This function should be called before the WDOG timer is in timeout. Otherwise, a reset is asserted.

Parameters

- base – WDOG peripheral base address

FSL_WDOG_DRIVER_VERSION

Defines WDOG driver version.

WDOG_REFRESH_KEY

enum _wdog_interrupt_enable

WDOG interrupt configuration structure, default settings all disabled.

This structure contains the settings for all of the WDOG interrupt configurations.

Values:

enumerator kWDOG_InterruptEnable

WDOG timeout generates an interrupt before reset

enum _wdog_status_flags

WDOG status flags.

This structure contains the WDOG status flags for use in the WDOG functions.

Values:

enumerator kWDOG_RunningFlag

Running flag, set when WDOG is enabled

enumerator kWDOG_PowerOnResetFlag

Power On flag, set when reset is the result of a powerOnReset

enumerator kWDOG_TimeoutResetFlag

Timeout flag, set when reset is the result of a timeout

enumerator kWDOG_SoftwareResetFlag

Software flag, set when reset is the result of a software

enumerator kWDOG_InterruptFlag

interrupt flag,whether interrupt has occurred or not

typedef struct *_wdog_work_mode* wdog_work_mode_t

Defines WDOG work mode.

typedef struct *_wdog_config* wdog_config_t

Describes WDOG configuration structure.

struct _wdog_work_mode

#include <fsl_wdog.h> Defines WDOG work mode.

Public Members

bool enableWait
If set to true, WDOG continues in wait mode

bool enableStop
If set to true, WDOG continues in stop mode

bool enableDebug
If set to true, WDOG continues in debug mode

struct __wdog__config
#include <fsl_wdog.h> Describes WDOG configuration structure.

Public Members

bool enableWdog
Enables or disables WDOG

wdog_work_mode_t workMode
Configures WDOG work mode in debug stop and wait mode

bool enableInterrupt
Enables or disables WDOG interrupt

uint16_t timeoutValue
Timeout value

uint16_t interruptTimeValue
Interrupt count timeout value

bool softwareResetExtension
software reset extension

bool enablePowerDown
power down enable bit

bool enableTimeOutAssert
Enable WDOG_B timeout assertion.

2.70 XBARA: Inter-Peripheral Crossbar Switch

void XBARA__Init(XBARA_Type *base)
Initializes the XBARA module.

This function un-gates the XBARA clock.

Parameters

- base – XBARA peripheral address.

void XBARA_Deinit(XBARA_Type *base)

Shuts down the XBARA module.

This function disables XBARA clock.

Parameters

- base – XBARA peripheral address.

void XBARA_SetSignalsConnection(XBARA_Type *base, xbar_input_signal_t input, xbar_output_signal_t output)

Sets a connection between the selected XBARA_IN[*] input and the XBARA_OUT[*] output signal.

This function connects the XBARA input to the selected XBARA output. If more than one XBARA module is available, only the inputs and outputs from the same module can be connected.

Example:

```
XBARA_SetSignalsConnection(XBARA, kXBARA_InputPIT_TRG0, kXBARA_
↪OutputDMAMUX18);
```

Parameters

- base – XBARA peripheral address.
- input – XBARA input signal.
- output – XBARA output signal.

uint32_t XBARA_GetStatusFlags(XBARA_Type *base)

Gets the active edge detection status.

This function gets the active edge detect status of all XBARA_OUTs. If the active edge occurs, the return value is asserted. When the interrupt or the DMA functionality is enabled for the XBARA_OUTx, this field is 1 when the interrupt or DMA request is asserted and 0 when the interrupt or DMA request has been cleared.

Parameters

- base – XBARA peripheral address.

Returns

the mask of these status flag bits.

void XBARA_ClearStatusFlags(XBARA_Type *base, uint32_t mask)

Clears the edge detection status flags of relative mask.

Parameters

- base – XBARA peripheral address.
- mask – the status flags to clear.

void XBARA_SetOutputSignalConfig(XBARA_Type *base, xbar_output_signal_t output, const *xbara_control_config_t* *controlConfig)

Configures the XBARA control register.

This function configures an XBARA control register. The active edge detection and the DMA/IRQ function on the corresponding XBARA output can be set.

Example:

```
xbara_control_config_t userConfig;
userConfig.activeEdge = kXBARA_EdgeRising;
userConfig.requestType = kXBARA_RequestInterruptEnalbe;
XBARA_SetOutputSignalConfig(XBARA, kXBARA_OutputDMAMUX18, &userConfig);
```

Parameters

- base – XBARA peripheral address.
- output – XBARA output number.
- controlConfig – Pointer to structure that keeps configuration of control register.

enum _xbara_active_edge

XBARA active edge for detection.

Values:

enumerator kXBARA_EdgeNone

Edge detection status bit never asserts.

enumerator kXBARA_EdgeRising

Edge detection status bit asserts on rising edges.

enumerator kXBARA_EdgeFalling

Edge detection status bit asserts on falling edges.

enumerator kXBARA_EdgeRisingAndFalling

Edge detection status bit asserts on rising and falling edges.

enumerator kXBARA_EdgeMax

Max value.

enum _xbar_request

Defines the XBARA DMA and interrupt configurations.

Values:

enumerator kXBARA_RequestDisable

Interrupt and DMA are disabled.

enumerator kXBARA_RequestDMAEnable

DMA enabled, interrupt disabled.

enumerator kXBARA_RequestInterruptEnable

Interrupt enabled, DMA disabled.

enumerator kXBARA_RequestMax

Max value.

enum _xbara_status_flag_t

XBARA status flags.

This provides constants for the XBARA status flags for use in the XBARA functions.

Values:

enumerator kXBARA_EdgeDetectionOut0

XBAR_OUT0 active edge interrupt flag, sets when active edge detected.

enumerator kXBARA_EdgeDetectionOut1

XBAR_OUT1 active edge interrupt flag, sets when active edge detected.

enumerator kXBARA_EdgeDetectionOut2

XBAR_OUT2 active edge interrupt flag, sets when active edge detected.

enumerator kXBARA_EdgeDetectionOut3

XBAR_OUT3 active edge interrupt flag, sets when active edge detected.

typedef enum *_xbara_active_edge* xbara_active_edge_t

XBARA active edge for detection.

typedef enum *_xbar_request* xbara_request_t

Defines the XBARA DMA and interrupt configurations.

typedef enum *_xbara_status_flag_t* xbara_status_flag_t

XBARA status flags.

This provides constants for the XBARA status flags for use in the XBARA functions.

typedef struct *XBARAControlConfig* xbara_control_config_t

Defines the configuration structure of the XBARA control register.

This structure keeps the configuration of XBARA control register for one output. Control registers are available only for a few outputs. Not every XBARA module has control registers.

FSL_XBARA_DRIVER_VERSION

XBARA_SELx(base, output)

XBARA_WR_SELx_SELx(base, input, output)

kXBARA_RequestInterruptEnalbe

struct XBARAControlConfig

#include <fsl_xbara.h> Defines the configuration structure of the XBARA control register.

This structure keeps the configuration of XBARA control register for one output. Control registers are available only for a few outputs. Not every XBARA module has control registers.

Public Members

xbara_active_edge_t activeEdge
Active edge to be detected.

xbara_request_t requestType
Selects DMA/Interrupt request.

2.71 XBARB: Inter-Peripheral Crossbar Switch

void XBARB_Init(XBARB_Type *base)
Initializes the XBARB module.

This function un-gates the XBARB clock.

Parameters

- base – XBARB peripheral address.

void XBARB_Deinit(XBARB_Type *base)
Shuts down the XBARB module.

This function disables XBARB clock.

Parameters

- base – XBARB peripheral address.

void XBARB_SetSignalsConnection(XBARB_Type *base, xbar_input_signal_t input,
xbar_output_signal_t output)
Configures a connection between the selected XBARB_IN[*] input and the XBARB_OUT[*] output signal.

This function configures which XBARB input is connected to the selected XBARB output. If more than one XBARB module is available, only the inputs and outputs from the same module can be connected.

Parameters

- base – XBARB peripheral address.
- input – XBARB input signal.
- output – XBARB output signal.

FSL_XBARB_DRIVER_VERSION

XBARB_SELx(base, output)

XBARB_WR_SELx_SELx(base, input, output)

Chapter 3

Middleware

3.1 Connectivity

3.1.1 lwIP

This is the NXP fork of the lwIP networking stack.

- For details about changes and additions made by NXP, see *NXP lwIP Fork Changelog*.
- For details about the NXP porting layer, see *The NXP lwIP Port*.
- For usage and API of lwIP, use official documentation.

MCUXpresso SDK : mcuxsdk-middleware-lwip

Overview This repository is a fork of lwIP TCP/IP stack (http://git.savannah.gnu.org/cgit/lwip.git) for MCUXpresso SDK delivery and it contains the components officially provided in NXP MCUXpresso SDK. This repository is part of the MCUXpresso SDK overall delivery which is composed of several sub-repositories/projects. Navigate to the top/parent repository (mcuxsdk-manifests) for the complete delivery of MCUXpresso SDK.

Documentation Overall details can be reviewed here: MCUXpresso SDK Online Documentation

Visit lwIP - Documentation to review details on the contents in this sub-repo.

Setup Instructions on how to install the MCUXpresso SDK provided from GitHub via west manifest Getting Started with SDK - Detailed Installation Instructions

Contribution Contributions are not currently accepted. If the intended contribution is not related to NXP specific code, consider contributing directly to the upstream lwIP project. Once this lwIP fork is synchronized with the upstream project, such contributions will end up here as well. If the intended contribution is a bugfix or improvement for NXP porting layer or for code added or modified by NXP, please open an issue or contact NXP support.

NXP Fork This fork of lwIP contains specific modifications and enhancements for NXP MCUXpresso SDK integration, including:

- Adaptation layers for some of the network interfaces found on NXP MCU platforms.
- Adaptation layer supporting compilers used with MCUXpresso SDK.
- Operating system abstraction layers for FreeRTOS and bare-metal.
- NXP implementation of HTTP server.
- Bugfixes and modifications of upstream code, which could not be upstreamed directly.
- CMake and Kconfig files for integration with MCUXpresso SDK build system.

See *changelog* for details.

The NXP lwIP Port

Below is description of possible settings of the port layer and an overview of a few helper functions.

The best place for redefinition of any mentioned macro is lwipopts.h.

The declaration of every mentioned function is in ethernetif.h. Please check the doxygen comments of those functions before.

Link state Physical link state (up/down) and its speed and duplex must be read out from PHY over MDIO bus. Especially link information is useful for lwIP stack so it can for example send DHCP discovery immediately when a link becomes up.

To simplify this port layer offers a function ethernetif_probe_link() which reads those data from PHY and forwards them into lwIP stack.

In almost all examples this function is called every ETH_LINK_POLLING_INTERVAL_MS (1500ms) by a function probe_link_cyclic().

By setting ETH_LINK_POLLING_INTERVAL_MS to 0 polling will be disabled. On FreeRTOS, probe_link_cyclic() will be then called on an interrupt generated by PHY. GPIO port and pin for the interrupt line must be set in the ethernetifConfig struct passed to ethernetif_init(). On bare metal interrupts are not supported right now.

Rx task To improve the reaction time of the app, reception of packets is done in a dedicated task. The rx task stack size can be set by ETH_RX_TASK_STACK_SIZE macro, its priority by ETH_RX_TASK_PRIO.

If you want to save memory you can set reception to be done in an interrupt by setting ETH_DO_RX_IN_SEPARATE_TASK macro to 0.

Disabling Rx interrupt when out of buffers If ETH_DISABLE_RX_INT_WHEN_OUT_OF_BUFFERS is set to 1, then when the port gets out of Rx buffers, Rx enet interrupt will be disabled for a particular controller. Everytime Rx buffer is freed, Rx interrupt will be enabled.

This prevents your app from never getting out of Rx interrupt when the network is flooded with traffic.

ETH_DISABLE_RX_INT_WHEN_OUT_OF_BUFFERS is by default turned on, on FreeRTOS and off on bare metal.

Limit the number of packets read out from the driver at once on bare metal. You may define macro ETH_MAX_RX_PKTS_AT_ONCE to limit the number of received packets read out from the driver at once.

In case of heavy Rx traffic, lowering this number improves the realtime behaviour of an app. Increasing improves Rx throughput.

Setting it to value < 1 or not defining means “no limit”.

Helper functions If your application needs to wait for the link to become up you can use one of the following functions:

- ethernetif_wait_linkup()- Blocks until the link on the passed netif is not up.
- ethernetif_wait_linkup_array() - Blocks until the link on at least one netif from the passed list of netifs becomes up.

If your app needs to wait for the IPv4 address on a particular netif to become different than “ANY” address (255.255.255.255) function ethernetif_wait_ipv4_valid() does this.

NXP lwIP Fork Changelog

This file documents the changes and additions made in the NXP fork of lwIP. For the list of changes in the upstream project version which this fork is based on, see *CHANGELOG*.

Source code included in this SDK is currently based on development version 2.2.2 taken from 3rd party lwIP GIT repository. The webpage https://git.savannah.nongnu.org/cgit/lwip.git allows to browse that repository and also contains URLs for its cloning.

The development versions (X.Y.Z.dev) do not refer to a single source code snapshots. To avoid ambiguity, change log below contains SHA-1 hashes of GIT commits used when importing the upstream code into the SDK.

KSDK refers to Kinetis SDK, the predecessor of MCUXpresso SDK.

2.2.2_rev2 (newest) New features:

- Added support for LWIP_ENET_QOS_FLEXIBLE_CONFIGURATION in enet_ethernetif_qos to allow changing ‘enet_qos_config_t’ struct.

2.2.2_rev1 New features:

- Synced to upstream lwIP master (2026-06-16, SHA-1: 3d896ba0a37ff3ce73270ca5e230707fe47f60e3) to MCUXpresso SDK.

Bug fixes:

- snmpv3: fixed handling of packets with invalid msgAuthenticationParameters length in snmp_parse_inbound_frame (stack-based buffer overflow / DoS; BDSA, MCUX-89594 / upstream commit 0c957ec0).
- tcp_in: added bounds check for next TCP option byte.
- nd6: parse all IPv6 ND options instead of only the first.
- mdns: fixed copying of invalid search result domains.
- smtp: fixed server-driven AUTH line overflow into tx_buf.
- ip4_frag/ip6_frag: fixed potential NULL-pointer access on memory errors.

2.2.1_rev12 Bug fixes:

- Fixed uninitialized chg_report_tmr in mld6_get_or_alloc_data, this could cause the MLDv6 report to be sent after a long time.

2.2.1_rev11 Bug fixes:

- altcp_tls (mbedTLS): Fixed three related defects in altcp_mbedtls_handle_rx_appldata() that affected reception of large TLS records.
  - Permanent TCP receive window stall when a single TLS record is larger than the inner TCP receive window (TCP_WND) and therefore spans more than one TCP segment: on MBEDTLS_ERR_SSL_WANT_READ the handler now pre-acknowledges the encrypted bytes already consumed by bio_recv, so the inner TCP window reopens and the remaining segments of the record can arrive.
  - LWIP_ASSERT("bogus byte counts", ...) could fire on the record-completion path once the pre-ack above was in effect, because bio_bytes_read is no longer guaranteed to be greater than bio_bytes_appl; the assertion was replaced with a guarded subtraction that only acks the residual TLS overhead when bio_bytes_read > bio_bytes_appl.
  - Truncated/deferred application data when the decrypted plaintext of a single record exceeded PBUF_POOL_BUFSIZE (typically with default MBEDTLS_SSL_IN_CONTENT_LEN of 16 KB): the receive loop now keeps decrypting while mbedtls_ssl_read() returns plaintext, instead of exiting as soon as state->rx/state->rx_app are empty. The loop is bounded by mbedTLS's internal record buffer and the availability of encrypted input on state->rx.
- altcp_tls (mbedTLS): Removed per-connection auto-free of the client altcp_tls_config in altcp_mbedtls_dealloc(). The TLS config lifetime is owned by the application, which calls altcp_tls_free_config() once no remaining connections reference it. The previous behavior caused a use-after-free / double-free when a single client config was shared across multiple connections.

2.2.1_rev10 Bug fixes:

- Checking the value of USE_RTOS macro in porting layer, not only if the macro is defined.

New features:

- The function clock_gettime() renamed to lwiperf_clock_gettime() to avoid conflict with system libs in compilation.

2.2.1_rev9 New features:

- The feature "Iperf now tries to place buffers into fast memory" (introduced in 2.2.1_rev5) can be disabled by defining LWIPERF_QUICKACCESS_TX_BUFFER to 0.

2.2.1_rev8 New features:

- Applied patch #10465: Overall altcp_tls_mbedtls fixes and enhancements (https://savannah.nongnu.org/patch/?10465):
  - Copy received TCP flags
  - Free client's TLS config on dealloc

  - Fix abort management. Fix to return ERR_ABRT on altcp_abort(). Add check to altcp_mbedtls_abort().
  - Free SSL cache and session tickets on tls_free_config()
  - Mark freed members to prevent double free.
  - Correctly close and free LISTEN connections
  - Use handshake steps rather than a single blocking call
  - Add mbedtls return values.
  - free rx_app on dealloc
  - correct ALTCP_MBEDTLS_PLATFORM_ALLOC configuration
  - Manage write apiflags
  - Port to mbedtls v3
  - Fix the return value of altcp_mbdtls_sndbuf when the underlying tcp sndbuf is consumed by ssl expansion bytes.
- Switched from mbedTLS 2.x to mbedTLS 3.x:
  - Updated Kconfig dependencies.
  - Updated altcp_tls_mbedtls layer to use PSA crypto random number generator abstraction instead of ctr_drbg, which would be initialized twice.
  - Updated NXP web server (src/apps/httpsrv).

Bug fixes:

- Fixed sys_sem_new in FreeRTOS porting layer:
  - If the initial count argument was 1, the newly created semaphore was not signalled from the start.
  - The initial count should be just 0 or 1 according to lwIP documentation, but the implementation allowed it to be more than 1 (counting semaphore). Fixed to allow only binary semaphore behavior.
  - src/apps/httpsrv updated to use FreeRTOS API directly where it cannot use sys_sem_t as a counting semaphore anymore.

2.2.1_rev7 New features:

- Operating system abstraction layer: the default implementation of the sys_msleep function (using a temporary allocated semaphore) is overwritten by a new implementation in the adaptation layer (using vTaskDelay on FreeRTOS).

Bug fixes:

- ENET QoS adaptation layer: initialize the CSR (control and status register) clock field in the driver configuration. This fixes the bug with the number of clock ticks for one-microsecond reference timer being set incorrectly.
- Ethernet adaptation layers: Updated logic in ethernetif_pbuf_free_safe to allow the thread to sleep between attempts to schedule pbuf_free on tcpip_thread. Previously, when ETH_RX_TASK_PRIO was higher than tcpip_thread priority and scheduling failed due to resource constraints, ethernetif_pbuf_free_safe would loop indefinitely, preventing tcpip_thread from freeing those resources. Changes include:
  - Added thread sleep between scheduling attempts to prevent infinite loops

  - Removed looping on schedule failure in interrupt context (now asserts instead)
  - Added direct pbuf_free calls when LWIP_ALLOW_MEM_FREE_FROM_OTHER_CONTEXT is enabled

2.2.1_rev6 New features:

- Support for on-demand timers for some of the protocols. Reused from Espressif open source lwIP port. LwIP default protocol timer handlers are invoked periodically, even if there is nothing to do. This is not suitable for low-power applications entering into a sleep mode. The added feature calculates the time when the timer callback should be invoked, instead of waking-up periodically to only check if there is anything to do or not. The feature is turned off by default and can be enabled by the following options:
  - LWIP_IP4_REASSEMBLY_TIMERS_ONDEMAND
  - LWIP_IP6_REASSEMBLY_TIMERS_ONDEMAND
  - LWIP_DNS_TIMERS_ONDEMAND
  - LWIP_DHCP_FINE_TIMERS_ONDEMAND
  - LWIP_IGMP_TIMERS_ONDEMAND
  - LWIP_MLD6_TIMERS_ONDEMAND

  Note that other protocols don't have on-demand timers implemented, so for example if there are open TCP connections, its protocol timers will be still invoked periodically.
- NETC adaptation layer: Possibility to use switch port, see NETC_USE_SWT and related options.
- New option ETH_USE_GPIO_ADAPTER which can be used on platforms without GPIO adapter to bypass the code using it.
- New option ETH_ENET_QOS_MII_MODE to override default MII mode for ENET QoS.

2.2.1_rev5 New features:

- Iperf now tries to place buffers into fast memory.

2.2.1_rev4 New features:

- Enabled build of ALTCP_TLS:
  - This enables option to use ALTCP with mbedTLS for transparent TLS.
  - Added new MCUX_COMPONENT_middleware.lwip.apps.altcp_tls component to Kconfig.
  - Updated CMake to compile the source files if the component is enabled.
  - Added ALTCP_MBEDTLS configuration settings to Kconfig.
- Added support for NETC VSI pre-init.
- Removed unused files with LGPL license or unknown origin.

Bug fixes:

- src/apps/altcp_tls: fixed compilation when MBEDTLS_HAVE_TIME is not defined.

- src/core/ipv6/mld6.c: fixed the setting of MLDv6 general report delay. Especially when the maximum response delay was smaller than MLD6_TMR_INTERVAL, it could be assigned to a value out of the requested range, or end up not sending the report at all.
- Fixed ethernet transmission issue on USB (not handling chained pbufs).

2.2.1_rev3 New features:

- Added optional ability to generate lwIP options header file from Kconfig. The feature has been introduced to align with the changes in MCUXpresso SDK. It can be enabled when using Kconfig and lwIP is used directly from repository, for example when using west build command to generate or build projects. It cannot be used when working with SDK packages, but since lwIP examples (in lwip_examples folder) have been switched to this approach in repository, you can find pregenerated lwipopts_gen.h files in SDK packages (note that if you want to add/change some of the pregenerated settings, do it in lwipopts_gen.h, not lwipopts.h).

  The optional feature is disabled by default and can be enabled by setting CONFIG_MCUX_PRJSEG_middleware.lwip.generated_config=y in prj.conf. When enabled, lwIP options are configured using the Kconfig system, usually written in prj.conf file(s) or .config file with CONFIG_ prefix. Kconfig system uses this information to generate lwipopts_gen.h during build phase. It is included from lwipopts.h (from template/kconfig). The lwipopts.h file is not intended to be edited, it does preprocessing of the values from lwipopts_gen.h.

  The list of options, its types, default values and description is in the file named Kconfig, under the menu "Options (lwipopts_gen.h)".

  Note that there are some differences from using manually edited lwipopts.h:

  – Use CONFIG_ prefix in Kconfig, so for example CONFIG_LWIP_IPV6 instead of LWIP_IPV6. In the generated header, the prefix is removed.

  – For boolean options, use y or n instead of 0 and 1 in Kconfig. For example CONFIG_LWIP_IPV6=y instead of #define LWIP_IPV6 1.

  – Default values for some integer options are using formulas referencing other options. For example, TCP_WND default is (4 * TCP_MSS). Kconfig does not support this, so this option has string type instead of int. The consequence is that you must write quotes in Kconfig for such settings, even if entering integer value, for example CONFIG_TCP_WND="5840". There are more such options, please pay attention to the type definition in Kconfig and any Kconfig warnings in the output of processing.

  – Default values for some options are referencing other options, without using formulas. For example TCP_TTL defaults to IP_DEFAULT_TTL. But since Kconfig translates defaults to values, if you change IP_DEFAULT_TTL later, you may need to override the TCP_TTL to the same value as well (if you still want it the same as IP_DEFAULT_TTL).

  – Options which are function macros, taking no parameters, have a single space between the braces. For example the following written in manually edited lwipopts.h:

    #define LWIP_ASSERT_CORE_LOCKED() sys_check_core_locking()

    would be like this in Kconfig instead:

    CONFIG_LWIP_ASSERT_CORE_LOCKED( )="sys_check_core_locking()"

  – Some functions macros are used in the lwIP code if they are defined. Because Kconfig generates string options with empty value, it would result in a compilation error. Therefore, such options have _CUSTOM suffix in Kconfig and the final option is hidden, without a prompt, generated only when the one with the _CUSTOM suffix does not have empty string value. Example is LOCK_TCPIP_CORE option, where it could

be written as CONFIG_LOCK_TCPIP_CORE_CUSTOM( )="sys_lock_tcpip_core()" instead of CONFIG_LOCK_TCPIP_CORE( )="sys_lock_tcpip_core()".

- Function signatures cannot be written in the generated lwipopts_gen.h. But it can be written in CONFIG_LWIP_OPT_H_CODE_LINE_N options, where N is a line number. It will appear in template/kconfig/lwipopts.h before lwipopts_gen.h is included. That way the signatures will be visible for the code which uses the function macros. For example the options to bring some function signatures from NXP porting layer could be entered like this for FreeRTOS projects:

```
CONFIG_LWIP_OPT_H_CODE_LINE_1="void sys_lock_tcpip_core(void);"
CONFIG_LWIP_OPT_H_CODE_LINE_2="void sys_unlock_tcpip_core(void);"
CONFIG_LWIP_OPT_H_CODE_LINE_3="void sys_check_core_locking(void);"
CONFIG_LWIP_OPT_H_CODE_LINE_4="void sys_mark_tcpip_thread(void);"
```

  or for bare-metal projects:

  CONFIG_LWIP_OPT_H_CODE_LINE_1="void sys_check_core_locking(void);"

  Note that the "line numbers" must be used from 1 and numbers cannot be skipped. So it is not possible to comment or delete CONFIG_LWIP_OPT_H_CODE_LINE_2 and continue with CONFIG_LWIP_OPT_H_CODE_LINE_3, all must be renumbered. The function signatures must use only basic C data types, no data types or typedefs from stdint.h or lwIP etc. There might be a cast used in the function macro option if needed.

- Some boolean options, which are just defined without value in C code, must still be written with a value. Example is LWIP_DEBUG, it would be written as CONFIG_LWIP_DEBUG=y or CONFIG_LWIP_DEBUG=n. Other such options are LWIP_PROVIDE_ERRNO and LWIP_ENET_FLEXIBLE_CONFIGURATION.
- Note that Kconfig does not generate boolean values if they are set to "n". The file template/kconfig/lwipopts.h defines the non-generated options to 0, so that lwIP does not define them again to 1 (if it has such default value in its code).
- When using Kconfig, do not define LWIP_RAND. It's function macro is defined in the porting layer in cc.h. See LWIP_DEFINE_LWIP_RAND_IN_CC_H there and in Kconfig.
- All options from opt.h and some of the applications and porting layer options have been enabled Kconfig. If you find something missing, such macro can be added to a project command line or manually edited in lwipopts_gen.h (if it is used and will not be regenerated).
- Formatting macros like X8_F, U16_F, etc. are not present in Kconfig system and does not need to be put into lwipopts.h, see below.
- Default values for formatting macros like X8_F, U16_F, etc. are defined in cc.h, so it does not need to be entered in lwipopts.h.

Bug fixes:

- src/apps/mdns: mdns_resp_remove_netif stops all timeouts related to the netif now. This prevents the timeout handlers to send answers on netifs which already have mDNS responder disabled and prevents access to deallocated structures. It also prevents excessive number of timeouts being scheduled when mDNS responder is disabled and reenabled on netif quickly.

2.2.1_rev2 Bug fixes:

- src/apps/lwiperf: Fixed calculation of the number of UDP datagrams to transmit. When the desired rate was high and datagram size small, the ideal delay between each transmitted

datagram could be smaller than 1 microsecond. The value was truncated to zero and resulted in zero-division and/or no UDP transmitted at all.

- port/arch/cc.h: code which checks for the presence of Newlib standard C library adds import of a standard header first so the ___NEWLIB___ macro is always available when Newlib is used. This prevents LWIP_TIMEVAL_PRIVATE from having different values in one program.

2.2.1_rev1 New features:

- Ported lwIP 2.2.1.dev (2024-02-19, branch: master, SHA-1: d0efd9ef7ba08e54b46b1060e2b4629a4907391b) to MCUXpresso SDK.
- Added ETH_MAX_RX_PKTS_AT_ONCE macro. See *port/README.md* for details.
- In port/netc_ethernetif.c, added NETC_VSI_NUM_USED macro to support using VSI. A thread of SI message handling will be started to handle VSI-PSI messages.

Bug fixes:

- Added the missing implementation for IP_FORWARD_ALLOW_TX_ON_RX_NETIF option in the function ip6_forward. Therefore, IPv6 packets could be sent back out on the netif where they were originally received from.
- NETC adaptation layer: Do not call xEventGroupSetBits from ISR.
- Ethernet adaptation layers: Default value of priority of the receive task (ETH_RX_TASK_PRIO) is set lower than the priority of the FreeRTOS daemon task (timer task).

2.2.0_rev11 New features:

- NETC adaptation layer: Possible to disable IPv4/TCP/UDP checksum validation done in HW.
- EtherCAT EoE(Ethernet over EtherCAT) driver is added to lwip.

Bug fixes:

- src/apps/httpsrv/httpsrv_supp.c: Fixed performing of the HTTP server task priority limitation.

2.2.0_rev10 New features:

- Ported lwIP 2.2.0 (2023-09-25, branch: master, SHA-1: 0a0452b2c39bdd91e252aef045c115f88f6ca773, tag: STABLE-2_2_0_RELEASE) to MCUXpresso SDK.
- Enabled hardware-accelerated CRC computation and verification (MAC, IPv4, TCP, UDP, ICMPv4, ICMPv6) for ENET Kinetis, ENET QoS and ENET LPC.
- Enabled link state detection based on PHY interrupts. The ETH_LINK_POLLING_INTERVAL_MS macro controls this - setting it to 0 and specifying ethernetif_config_t->phyIntGpio enables it, setting it to a value greater than zero enables polling instead. Supported only under an RTOS (NO_SYS == 0). By default, the link state is polled.
- ND6: Implemented RFC 4191 type C host, which means default router list (learned from Router Advertisement messages) has been replaced with routing table, which contains default route records for each router and also routes learned from received Route Information Options. Changes partially based on patch 10114. The option LWIP_ND6_NUM_ROUTERS

has been removed, and the new option LWIP_ND6_NUM_ROUTES has been added to configure the size of the routing table.

- IPv6: Implemented a new hook - LWIP_HOOK_IP6_CANFORWARD. This hook can be used, for example, for multicast forwarding between netifs. Defining this hook enables multicast traffic forwarding, thus the hook is also invoked for multicast traffic.
- MLD6: Multicast Listener Discovery v1 replaced by v2 (RFC 3810) but without support of source specific multicast.
- port/enet_ethernetif_kinetis.c: Added check to generate/validate ICMPv6 checksum in SW as the Kinetis ENET peripheral does not do it.
- Added disabling of Rx interrupt when the port is out of Rx buffers. See *port/README.md* for more details.

Bug fixes:

- src/apps/lwiperf: Fixed access to invalid data when UDP report is to be sent from a timer but abort has been called before.
- src/apps/lwiperf: Fixed deallocation of TCP server started by client (in reverse or dual modes) which failed to connect.
- port/netc_ethernetif.c: Fixed cache control enablement macro (FSL_SDK_ENABLE_DRIVER_CACHE_CONTROL > FSL_ETH_ENABLE_CACHE_CONTROL).
- port/sys_arch.c: The function sys_assert does not call portENTER_CRITICAL when called from an interrupt.
- src/core/ipv4/ip4.c: Fixed checksum reset condition.
- ND6:
  - Lladdr length is now taken from netif->hwaddr_len so ND6 works properly regardless of NETIF_MAX_HWADDR_LEN.
  - Added check of sufficient length of lladdr options from incoming messages.
- src/apps/httpsrv/httpsrv.c: Fixed hangup in HTTPSRV_release if caller's task has higher priority than server task.
- port/arch/cc.h: LWIP_PLATFORM_DIAG is defined (and can be overridden) independently of the LWIP_DEBUG setting. Removed printing extra newline symbols from LWIP_PLATFORM_DIAG.
- src/apps/lwiperf: The "end of test" UDP datagram is resent more often. This increases the probability of the server to receive it and end the test when datagrams are getting lost.
- Added *port/README.md* describing possible settings and helper functions in the port layer.

2.2.0_rev9 New features:

- Ported lwIP 2.2.0.dev (2023-01-03, branch: master, SHA-1: 3fe8d2fc43a9b69f7ed28c63d44a7744f9c0def9) to MCUXpresso SDK.
- Applied patch to allow sending IPv6 router advertisement. Improved to allow selection of interface and router life time and to allow sending route information options.
- src/apps/lwiperf: Support for reverse test (client receives, server sends). Requires iperf version 2.1.0 or newer.

Bug fixes:

- src/apps/httpsrv: Fixed operation with LWIP_IPV6 enabled. Server can be also accessed using both IPv4 and IPv6 at the same time if compiled with both LWIP_IPV4=1 and LWIP_IPV6=1. Note the type of the field struct httpsrv_param_struct.address has changed from struct sockaddr to struct sockaddr_storage.

2.2.0_rev8 New features:

- src/apps/lwiperf: Added new parameter "buffer_len" to functions lwiperf_start_tcp_client() and lwiperf_start_udp_client() to configure TCP/UDP packet size.
- src/apps/lwiperf: Added new parameter "tos" to functions lwiperf_start_tcp_client() to configure TCP packet priority.
- NETC adaptation layer: Not forcing the RX/TX buffers placement in non-cacheable memory. Requires the symbol FSL_ETH_ENABLE_CACHE_CONTROL to be defined on project level if the memory region, where the buffers are placed by a linker, has cache enabled.

Bug fixes:

- src/apps/httpsrv: Added missing includes.
- src/apps/lwiperf: Fixed TCP client to send settings at the beginning of each 128 KB block like the PC iperf 2.0.x application does.
- src/apps/lwiperf: Fixed validation of TCP received data (with LWIPERF_CHECK_RX_DATA enabled, works with iperf 2.0.x).
- src/apps/lwiperf: Fixed lwiperf_list_remove() to clear references to the removed item.
- src/apps/lwiperf: Program does not assert when buffer cannot be cloned in UDP test, only "can't clone buffer" message is printed.

2.2.0_rev7 New features:

- Ported lwIP 2.2.0.dev (2022-05-09, branch: master, SHA-1: 239918ccc173cb2c2a62f41a40fd893f57faf1d6) to MCUXpresso SDK.
- Added function ethernetif_probe_link() which reads actual link, speed and duplex settings from phy and passes them to driver. Stack could be set to call this function periodically by setting ETH_LINK_POLLING_INTERVAL_MS to value higher than zero.
- Added helper functions ethernetif_wait_linkup() and ethernetif_wait_ipv4_valid() to allow blocking of RTOS task or bare metal application until link is up or IPv4 address becomes valid.
- Added NETC adaptation layer.
- Processing of rx packets under RTOS moved from ISR to a separate task to improve system reaction times. Switch back to old behavior can be done by setting ETH_DO_RX_IN_SEPARATE_TASK macro to 0.

Bug fixes:

- port: Fixed copying of pbuf contents. Previous code was using an incorrect end condition and could result in the overrun of the destination buffer if more packets were on the queue.
- port: Delegating pbuf_free calls to tcpip_thread via pbuf_free_callback where possible (RTOS), ensuring pbuf_free is not called from interrupt context when LWIP_ALLOW_MEM_FREE_FROM_OTHER_CONTEXT is not set (bare metal).
- port/enet_ethernetif_qos.c - Fixed ENET_RXBD_NUM which was used instead of ENET_TXBD_NUM.

- port/enet_ethernetif_qos.c - Fixed buffer alignment to be at least 64.
- src/apps/lwiperf: Fixed IPv6 TCP TX throughput lower than IPv4 by modifying maximum segment size to avoid sending two segments instead of one.
- src/apps/lwiperf: Out-of-order datagrams in UDP RX server mode are counted to the throughput.
- src/apps/httpsrv: Implemented receive timeouts on sockets.
- src/apps/httpsrv: Don't assert on HTTP session task creation failure.
- src/apps/httpsrv: Fixed build with IPv6 enabled.
- src/apps/httpsrv: Updated endianess macros required for websocket SHA generation.
- src/apps/httpsrv: Added missing includes.

2.2.0_rev6 New features:

- Ported lwIP 2.2.0.dev (2022-03-25, branch: master, SHA-1: 124dc0a64ef5d7c14a27e3115e5888df6559cb72) to MCUXpresso SDK.
- Implemented leaving of multicast groups on ENET and ENET QOS.

2.2.0_rev5 New features:

- Ported lwIP 2.2.0.dev (2021-05-11, branch: master, SHA-1: 7ec4e9be304e7f8953740f10b2c810a292e89449) to MCUXpresso SDK.
- LPC ENET adaptation layer allocates more buffers for frame reception now. Previously the number of receive buffers was determined by ENET_RXBD_NUM, which defaults to 5. It is determined by ENET_RXBUFF_NUM now, which is 2 * ENET_RXBD_NUM by default. Increase was needed because the actual version of LPC ENET driver always hold ENET_RXBD_NUM number of buffers and few additional buffers are needed for passing zero-copy frame data to lwIP. If this takes too much memory in your application, you can counteract by decreasing PBUF_POOL_SIZE, since PBUF_POOL is used only for transmission when LPC ENET, Kinetis ENET or ENET QOS is used.

2.2.0_rev4 New features:

- Ported lwIP 2.2.0.dev (2021-03-05, branch: master, SHA-1: 0056522cc974d2be2005c324f37187b5b3695765) to KSDK 2.0.0.
- LWIP_DHCP_DOES_ACD_CHECK option default changed to 0 (disabled):
  - Although the ACD check makes getting IP address from DHCP more robust, it added several seconds delay at startup of all applications which use DHCP.
  - This feature was not present in earlier versions of lwIP.
- ENET QOS adaptation layer - implemented zero-copy on receive.
- Kinetis ENET and ENET QOS adaptation layers allocate more buffers for frame reception now. Previously the number of receive buffers was determined by ENET_RXBD_NUM, which defaults to 5. It is determined by ENET_RXBUFF_NUM now, which is 2 * ENET_RXBD_NUM by default. Increase was needed because the actual version of Kinetis ENET and ENET QOS drivers always hold ENET_RXBD_NUM number of buffers and few additional buffers are needed for passing zero-copy frame data to lwIP. If this takes too much memory in your application, you can counteract by decreasing PBUF_POOL_SIZE, since PBUF_POOL is used only for transmission when Kinetis ENET or ENET QOS is used.

- Removed ethernetif_config_t.non_dma_memory field which was required to configure memory ranges unusable by ENET DMA on LPC devices. The setting has been replaced by BOARD_ENET_NON_DMA_MEMORY_ARRAY macro.

2.2.0_rev3 New features:

- Ported lwIP 2.2.0.dev (2020-07-07, branch: master, SHA-1: c385f31076b27efb8ee37f00cb5568783a58f299) to KSDK 2.0.0.

2.2.0_rev2 New features:

- Kinetis ENET adaptation layer - implemented zero-copy on receive.
- lwiperf - counter of transferred bytes extended from 32 to 64 bit

Bug fixes:

- Fixed restarting Auto IP from DHCP.

2.2.0_rev1 New features:

- Ported lwIP 2.2.0.dev (2019-12-12, branch: master, SHA-1: 555812dcec38c9a2ef1ef9b31816291549fbf9f8) to KSDK 2.0.0.
- Implemented LWIP_ASSERT_CORE_LOCKED related functions in sys_arch.c. It can be enabled in lwipopts.h:

```
#define LWIP_ASSERT_CORE_LOCKED() sys_check_core_locking()
#define LWIP_MARK_TCPIP_THREAD() sys_mark_tcpip_thread() // if NO_SYS == 0
#define LOCK_TCPIP_CORE() sys_lock_tcpip_core() // if NO_SYS == 0 and LWIP_TCPIP_
↪CORE_LOCKING == 1
#define UNLOCK_TCPIP_CORE() sys_unlock_tcpip_core() // if NO_SYS == 0 and LWIP_
↪TCPIP_CORE_LOCKING == 1
```

2.1.2_rev5 New features:

- Implemented TCP_USER_TIMEOUT socket option.
- Implemented SIOCOUTQ ioctl.

2.1.2_rev4 New features:

- Ported lwIP 2.1.3.dev (2019-02-27, branch: STABLE-2_1_x, SHA-1: 1bb6e7f52de1cd86be0eed31e348431edc2cd01e) to KSDK 2.0.0.
- Updated sys_thread_new implementation and comment.
- Kinetis ENET adaptation layer - reading frames into a pbuf chain is conditionally compiled only when a single pbuf from pool cannot hold maximum frame size (PBUF_POOL_BUFSIZE >= maximum frame size). Avoiding this code also reduces stack size requirements by about 1.5 kilobytes.

Bug fixes:

- Fixes in ethernetif_linkoutput() in enet_ethernetif_lpc.c:
  - Removed access to possibly freed pbuf.
  - Call pbuf_free() when transmit buffers not available.
  - When copying pbuf chain, updating the number of necessary transmit buffers to wait for, which can be often smaller in the copy.
- When CGI script is reading POST data by chunks, the loop in httpsrv_read() may cause blocking in receive function waiting for more data at the end of the stream
  - HTTPSRV_cgi_read() - added limiting of the last chunk length according to content length to avoid undesired blocking
- Applied AUTOIP patch 9847 - with modification to support multiple network interfaces.
- Fixed buffer overflow in httpsrv when application provided CGI script does not handle the whole content of POST request
  - Removed LwipMibCompiler contrib application as it contained LGPL licensed files in SharpSnmpLib.

2.1.2_rev3 New features:

- lwiperf updated with UDP client/server support from the patch 9751

2.1.2_rev2 Bug fixes:

- Fixed lwiperf_abort() in lwiperf.c to correctly close connections and free resources

2.1.2_rev1 New features:

- Ported lwIP 2.1.2 (2018-11-22, SHA-1: 159e31b689577dbf69cf0683bbaffbd71fa5ee10) to KSDK 2.0.0.
- Ported lwIP-contrib 2.1.0 (2018-09-24, SHA-1: 35b011d4cf4c4b480f8859c456587a884ec9d287) to KSDK 2.0.0.

2.0.3_rev1 New features:

- Ported lwIP 2.0.3 (2017-09-15, SHA-1: 92f23d6ca0971a32f2085b9480e738d34174417b) to KSDK 2.0.0.

2.0.2_rev1 New features:

- Ported lwIP 2.0.2 (2017-03-13, SHA-1: c0862d60746e2d1ceae69af4c6f24e469570ecef) to KSDK 2.0.0.

2.0.0_rev3 New features:

- Ported lwIP 2.0.0 (2016-11-10, SHA-1: 216bf89491815029aa15463a18744afa04df58fe) to KSDK 2.0.0.

2.0.0_rev2 New features:

- Ported lwIP 2.0.0 RC2 (2016-08-08, SHA-1: b1dfd00f9233d124514a36a8c8606990016f2ad4) to KSDK 2.0.0.

2.0.0_rev1 New features:

- Ported lwIP 2.0.0 RC0 (2016-05-26) to KSDK 2.0.0.
- Changed lwIP bare-metal examples to use poll-driven approach instead of interrupt-driven one.

1.4.1_rev2 New features:

- Enabled critical sections in lwIP.

Bug fixes:

- Fixed default lwIP packet-buffer size to be able to accept a maximum size frame from the ENET driver.
- Fixed possible drop of multi-frame packets during transmission.

1.4.1_rev1 New features:

- Ported lwIP 1.4.1 to KSDK 2.0.0.

3.2 File System

3.2.1 FatFs

MCUXpresso SDK : mcuxsdk-middleware-fatfs

Overview This repository is for FatFs middleware delivery and it contains the components officially provided in NXP MCUXpresso SDK. This repository is part of the MCUXpresso SDK overall delivery which is composed of several sub-repositories/projects. Navigate to the top/parent repository (mcuxsdk-manifests) for the complete delivery of MCUXpresso SDK.

Documentation Overall details can be reviewed here: MCUXpresso SDK Online Documentation

Visit FatFs - Documentation to review details on the contents in this sub-repo.

Setup Instructions on how to install the MCUXpresso SDK provided from GitHub via west manifest Getting Started with SDK - Detailed Installation Instructions

Contribution Contributions are not currently accepted. Guidelines to contribute will be posted in the future.

Repo Specific Content This is MCUXpresso SDK fork of FatFs (FAT file system created by ChaN). Official documentation is available at http://elm-chan.org/fsw/ff/

MCUXpresso version is extending original content by following hardware specific porting layers:

- mmc_disk
- nand_disk
- ram_disk
- sd_disk
- sdspi_disk
- usb_disk

Changelog FatFs

All notable changes to this project will be documented in this file.

The format is based on Keep a Changelog

[R0.16_rev0]

- Upgraded to version 0.16
- Applied patch https://elm-chan.org/fsw/ff/patch/ff16p1.diff
- Applied patch https://elm-chan.org/fsw/ff/patch/ffunicode.zip
- Make lb and hb variables conditional. Fix 'unused variable' issue:
  - ffunicode.c:10622:28: error: unused variable 'lb'
  - ffunicode.c:10622:7: error: unused variable 'hb'

[R0.15_rev0]

- Upgraded to version 0.15
- Applied patches from http://elm-chan.org/fsw/ff/patches.html

[R0.14b_rev1]

- Applied patches from http://elm-chan.org/fsw/ff/patches.html

[R0.14b_rev0]

- Upgraded to version 0.14b

[R0.14a_rev0]

- Upgraded to version 0.14a
- Applied patch ff14a_p1.diff and ff14a_p2.diff

[R0.14_rev0]

- Upgraded to version 0.14
- Applied patch ff14_p1.diff and ff14_p2.diff

[R0.13c_rev0]

- Upgraded to version 0.13c
- Applied patches ff_13c_p1.diff,ff_13c_p2.diff, ff_13c_p3.diff and ff_13c_p4.diff.

[R0.13b_rev0]

- Upgraded to version 0.13b

[R0.13a_rev0]

- Upgraded to version 0.13a. Added patch ff_13a_p1.diff.

[R0.12c_rev1]

- Add NAND disk support.

[R0.12c_rev0]

- Upgraded to version 0.12c and applied patches ff_12c_p1.diff and ff_12c_p2.diff.

[R0.12b_rev0]

- Upgraded to version 0.12b.

[R0.11a]

- Added glue functions for low-level drivers (SDHC, SDSPI, RAM, MMC). Modified diskio.c.
- Added RTOS wrappers to make FatFs thread safe. Modified syscall.c.
- Renamed ffconf.h to ffconf_template.h. Each application should contain its own ffconf.h.
- Included ffconf.h into diskio.c to enable the selection of physical disk from ffconf.h by macro definition.
- Conditional compilation of physical disk interfaces in diskio.c.

3.3 Motor Control

3.3.1 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_TII	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.

3.4 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.

3.4.1 Quick links

- *Release Notes*

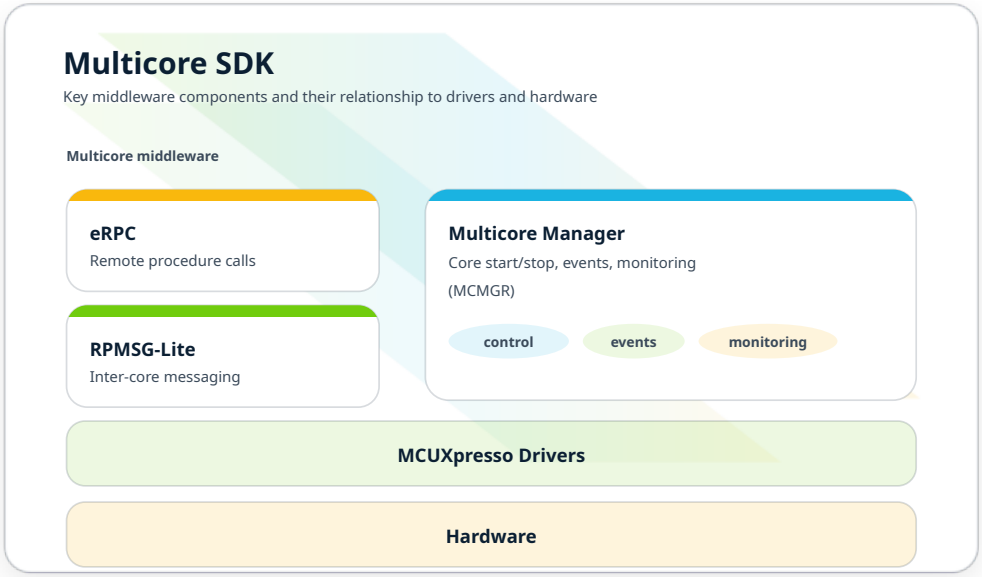


Fig. 1: Multicore SDK software stack (simplified).

- *Getting Started*
- Multicore SDK Changelog

3.4.2 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

MCSDK Getting Started

RPMSG-Lite

Quick links

- Changelog
- API documentation (Doxygen): https://nxp-mcuxpresso.github.io/rpmsg-lite/

Multicore Manager (MCMGR)

Quick links

- Changelog
- API documentation (Doxygen): https://nxp-mcuxpresso.github.io/mcux-mcmgr/

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

3.5 Wireless

3.5.1 NXP Wireless Framework and Stacks

Wi-Fi, Bluetooth, 802.15.4

Application notes

- Link AN12918-Wi-Fi-Tx-Power-Table-and-Channel-Scan-Management-for-i.MX-RT-SDK.pdf
- Link TN00066-WFA-Derivative-Certification-Process.pdf

User manuals

- Link UM11441-Getting-Started-with-NXP-based-Wireless-Modules-and-i.MX-RT-Platforms.pdf
- UM11442-NXP-Wi-Fi-and-Bluetooth-Demo-Applications-for-i.MX-RT-Platforms.pdf
- Link UM11443-NXP-Wi-Fi-and-Bluetooth-Debug-Feature-Configuration-Guide-for-i.MX-RT-Platforms.pdf
- Link UM11567-WFA-Certification-Guide-for-NXP-based-Wireless-Modules-on-i.MX-RT-Platform-Running-RTOS.pdf

Release notes Wireless SoC features and release notes for FreeRTOS

About this document

This document provides information about the supported features, release versions, fixed and/or known issues, performance of the Wi-Fi, Bluetooth/802.15.4 radios, including the coexistence.

The SDK release version 26.09.00 has been tested for the wireless SoCs listed in Supported products.

Supported products

- 88W8987
- IW416
- IW611

- IW612
- AW611
- RW610
- RW612

Parent topic: *About this document*

Features

Wi-Fi radio

Client mode

Features	Sub features
802.11n - High throughput	2.4 GHz band operation supported channel bandwidth: 20 M
802.11n - High throughput	2.4 GHz band supported channel bandwidth: 40 MHz
802.11n - High throughput	5 GHz band supported channel bandwidth: 20 MHz
802.11n - High throughput	5 GHz band supported channel bandwidth: 40 MHz
802.11n - High throughput	Short/long guard interval (400 ns/800 ns)
802.11n - High throughput	Data rates up to 72 Mbit/s (MCS 0 to MCS 7)
802.11n - High throughput	Data rates up to 150 Mbit/s (MCS 0 to MCS 7)
802.11n - High throughput	1 spatial stream (1x1)
802.11n - High throughput	HT protection mechanisms
802.11n - High throughput	Aggregated MAC protocol data unit (AMPDU) TX and RX sup
802.11n - High throughput	Aggregated MAC service data unit (AMSDU) 4k TX and RX su
802.11n - High throughput	TX MCS rate adaptation (BGN)
802.11n - High throughput	RX low density parity check (LDPC) 1x1 20 MHz and 40 MHz
802.11n - High throughput	HT Beamformee (explicit)
802.11ac - Very high throughput	2.4 GHz band supported channel bandwidth: 20MHz
802.11ac - Very high throughput	5 GHz band supported channel bandwidth: 20 MHz
802.11ac - Very high throughput	5 GHz band supported channel bandwidth: 40 MHz
802.11ac - Very high throughput	5 GHz band supported channel bandwidth: 80 MHz
802.11ac - Very high throughput	Data rates up to 86.7 Mbps (MCS0 to MCS 8)
802.11ac - Very high throughput	Data rates up to 433.3 Mbps (MCS 0 to MCS 9) - 1x1
802.11ac - Very high throughput	MU-MIMO Beamformee (Explicit and Implicit)
802.11ac - Very high throughput	RTS/CTS with BW signaling
802.11ac - Very high throughput	Operation mode notification
802.11ac - Very high throughput	Backward compatibility with non-VHT devices
802.11ac - Very high throughput	TX VHT MCS rate adaptation
802.11ac - Very high throughput	Low density parity check (LDPC)
802.11ax - High efficiency	2.4 GHz band supported channel bandwidth: 20MHz
802.11ax - High efficiency	5 GHz band supported channel bandwidth: 20 MHz
802.11ax - High efficiency	5 GHz band supported channel bandwidth: 40 MHz
802.11ax - High efficiency	5 GHz band supported channel bandwidths: 80 MHz
802.11ax - High efficiency	OFDMA (UL/DL, 106 RU)
802.11ax - High efficiency	OFDMA (UL/DL, 484 RU)
802.11ax - High efficiency	1024 QAM
802.11ax - High efficiency	Target wake time (TWT)
802.11ax - High efficiency	256 QAM modulation – MCS8 and MCS9
802.11ax - High efficiency	1024 QAM modulation – MCS10 and MCS11, 2.4 GHz
802.11ax - High efficiency	1024 QAM modulation – MCS10 and MCS11, 5 GHz
802.11ax - High efficiency	DCM
802.11ax - High efficiency	DCM
802.11ax - High efficiency	ER (extended range)

Table 1 – continued from p

Features	Sub features
802.11ax - High efficiency	SU Beamforming
802.11ax - High efficiency	OMI (operating mode indication)
802.11a/b/g features	802.11b/g data rates up to 54 Mbit/s
802.11a/b/g features	802.11a data rates up to 54 Mbit/s
802.11a/b/g features	TX rate adaptation (BG)
802.11a/b/g features	Fragmentation/defragmentation
802.11a/b/g features	ERP protection, slot time, preamble
802.11d	802.11d - Regulatory domain/operating class/country info
802.11e QoS	EDCA [enhanced distributed channel access] / WMM (wirele
802.11i security	Opensource WPA Supplicant Support
802.11i security	WPA2-PSK AES | WPA Supplicant
802.11i security	WPA3-SAE (Simultaneous Authentication of Equals) | WPA
802.11i security	WPA2+WPA3 PSK Mixed Mode (WPA3 Transition Mode) | W
802.11i security	Wi-Fi Enhanced Open - OWE (Opportunistic Wireless Encry
802.11i security	802.1x EAP Authentication Methods3 | WPA Supplicant
802.11i security	WPA2-Enterprise Mixed Mode | WPA Supplicant
802.11i security	WPA3-Enterprise3 (Suite-B) |National Security Algorithm (C
802.11i security	802.11w - PMF (Protected Management Frames) | WPA Supp
802.11i security	Embedded Supplicant Support
802.11i security	WPA2-PSK AES | Embedded Supplicant
802.11i security	WPA+WPA2 PSK Mixed Mode | Embedded Supplicant
802.11i security	WPA3-SAE (Simultaneous Authentication of Equals) | Embe
802.11i security	802.11w - PMF (Protected Management Frames) | Embedde
802.11i security	Wi-Fi Roaming
802.11i security	WPA3 Enterprise3
Power save mode	Deep sleep
Power save mode	IEEE power save
Power save mode	Host sleep/WoWLAN (inband)3
Power save mode	Host sleep/WoWLAN (outband)3
Power save mode	U-APSD
802.11w - PMF (protected management frames)	PMF require and capable
802.11w - PMF (protected management frames)	Unicast management frames - Encryption/decryption - using
802.11w - PMF (protected management frames)	Broadcast management frames - Encryption/decryption - us
802.11w - PMF (protected management frames)	SA query request/response
802.11w - PMF (protected management frames)	PMF support using embedded supplicant
DPP functionality	Wi-Fi easy connect3
General features	Embedded supplicant
General features	Host sleep packet filtering
General features	Host-based supplicant
General features	Embedded MLME
General features	EDMAC - EU adaptivity support (ETSI certification)
General features	External coexistence
General features	IPv6 NS offload
General features	FIPS
General features	TKIP1
General features	RF test mode
General features	802.11k
General features	802.11v
General features	DFS radar detection in peripheral mode (follow AP)5
General features	Embedded roaming based on RSSI threshold beacon loss
General features	ARP offload
General features	Cloud keep alive
General features	UNII-4 channel support
General features	ClockSync using TSF
General features	Auto reconnect

Table 1 – continued from p

Features	Sub features
General features	CSI (channel state information)
General features	Ambient Motion Index (AMI)3
General features	Independent reset (in-band)3
General features	Independent reset (out-band)3
General features	Wi-Fi agile multiband
General features	Network co-processor (NCP) mode
General features	802.11mc - WLS (Wi-Fi location service)3
General features	802.11az3
General features	WPA3 Transition Disable

Parent topic: Wi-Fi radio

[1] As per Wi-Fi specification, connecting in TKIP security in non 802.11n mode is allowed.

[2] Support available in host-base supplicant.

[3] Feature not enabled by default in the SDK. Refer to *Feature enable and memory impact* for the macro to enable the feature and the impact on the memory when enabling the feature.

[4] Read more about NCP feature in *References*. [5] To enable the feature, CONFIG_ECSA = 1 must be defined in wifi_config.h (does not apply to RW610 and RW612).

AP mode

Features	Sub features
802.11n - High throughput	2.4 GHz band operation supported channel bandwidth: 20 M
802.11n - High throughput	2.4 GHz band supported channel bandwidth: 40 MHz
802.11n - High throughput	5 GHz band supported channel bandwidth: 20 MHz
802.11n - High throughput	5 GHz band supported channel bandwidth: 40 MHz
802.11n - High throughput	Short/long guard interval (400 ns/800 ns)
802.11n - High throughput	Data rates up to 72 Mbit/s (MCS 0 to MCS 7)
802.11n - High throughput	Data rates up to 150 Mbit/s (MCS 0 to MCS 7)
802.11n - High throughput	1 spatial stream (1x1)
802.11n - High throughput	HT protection mechanisms
802.11n - High throughput	Aggregated MAC protocol data unit (AMPDU) Rx support
802.11n - High throughput	Aggregated MAC service data unit (AMSDU) -4k RX support
802.11n - High throughput	Max client support (up to 8 devices)
802.11n - High throughput	TX MCS rate adaptation (BGN)
802.11n - High throughput	RX low density parity check (LDPC)
802.11ac – Very high throughput	5 GHz band supported channel bandwidth: 20 MHz
802.11ac – Very high throughput	5 GHz band supported channel bandwidth: 40 MHz
802.11ac – Very high throughput	5 GHz band supported channel bandwidth: 80MHz
802.11ac – Very high throughput	Short/long guard interval (400ns/800ns)
802.11ac – Very high throughput	Data rates up to 86.7 Mbps (MCS0 to MCS 8)
802.11ac – Very high throughput	Data rates up to 433.3 Mbps (MCS 0 to MCS 9)
802.11ac – Very high throughput	Single user- Aggregated MAC protocol data unit (SU-AMPDU
802.11ac – Very high throughput	RTS/CTS with BW signaling
802.11ac – Very high throughput	Backward compatibility with non-VHT devices
802.11ac – Very high throughput	TX VHT MCS rate adaptation
802.11ac – Very high throughput	MU-MIMO Beamformee (explicit and implicit)
802.11ac – Very high throughput	Operation mode notification
802.11ax – High efficiency	2.4 GHz band operation (20 MHz channel bandwidth)
802.11ax – High efficiency	2.4 GHz band operation (40 MHz channel bandwidth)
802.11ax – High efficiency	5 GHz band operation (20MHz channel bandwidth)
802.11ax – High efficiency	5 GHz band operation (40MHz channel bandwidth)

Table 2 – continued from prev

Features	Sub features
802.11ax – High efficiency	5 GHz band operation (80 MHz channel bandwidth)
802.11d	802.11d - Regulatory domain/operating class/country info
802.11e -QoS	EDCA [enhanced distributed channel access] / WMM (wirele
802.11i security	Hostapd Support
802.11i security	WPA2-PSK AES | hostapd
802.11i security	WPA3-SAE (Simultaneous Authentication of Equals) | Hosta
802.11i security	WPA2+WPA3 PSK Mixed Mode (WPA3 Transition Mode) | H
802.11i security	Wi-Fi Enhanced Open - OWE (Opportunistic Wireless Encry
802.11i security	802.1x EAP Authentication Methods | Hostapd
802.11i security	WPA2-Enterprise Mixed Mode1 | Hostapd
802.11i security	WPA3-Enterprise (Suite-B)1 |National Security Algorithm (C
802.11i security	802.11w - PMF (Protected Management Frames) | Hostapd
802.11i security	Embedded Authenticator
802.11i security	WPA2-PSK AES | Embedded Supplicant
802.11i security	WPA+WPA2 PSK Mixed Mode | Embedded Supplicant
802.11i security	WPA3-SAE (Simultaneous Authentication of Equals) | Embe
802.11i security	802.11w - PMF (Protected Management Frames) | Embedde
802.11y	Extended channel switch announcement (ECSA)
802.11w - protected management frames (PMF)	PMF require and capable
802.11w - protected management frames (PMF)	Unicast management frames -Encryption/decryption - using
802.11w - protected management frames (PMF)	Broadcast management frames -encryption/decryption - usi
802.11w - protected management frames (PMF)	SA query request/response
General features	Embedded authenticator
General features	Embedded MLME
General features	EU adaptivity support
General features	Automatic channel selection (ACS)
General features	External coexistence (software interface)
General features	Independent reset (in-band)1
General features	Network co-processor (NCP) mode2
General features	Vendor specific IE (custom IE)
General features	Hidden SSID (broadcast SSID disabled)
General features	MAC address filter
General features	Multiple external STA support
General features	WPA3 Transition Disable

Parent topic: Wi-Fi radio

[1] Feature not enabled by default in the SDK. Refer to *Feature enable and memory impact* for the macro to enable the feature and the impact on the memory. [2] Read more about NCP feature in *References*.

AP-STA mode

Features	Sub features	88W8	IW41	IW611/IV	RW610/R\	IW61	AW611
Simultaneous AP-STA operation (same channel)	AP-STA functionality	Y	Y	Y	Y	Y	Y
SAD	Software antenna diversity1	Y	Y	Y	Y	Y	Y

Parent topic: Wi-Fi radio

[1] Feature not enabled by default in the SDK. Refer to *Feature enable and memory impact* for the macro to enable the feature and the impact on the memory when enabling the feature.

Parent topic: *Features*

Wi-Fi Generic features

Features	Sub features	88W898	IW416	IW611/IW6	RW610/RW6	IW610	AW611
Generic	Firmware download (parallel)1	Y	Y	Y	N	N	Y
Generic	Secure boot	N	N	Y	Y	Y	Y
Generic	Kconfig memory optimizer3	Y	Y	Y	Y	Y	Y
Generic	Firmware Compression2	N	Y	N	N	N	N
Generic	u-AP intra-BSS	Y	N	Y	Y	Y	Y
Generic	Net Monitor Mode	N	N	N	Y	Y	N
Generic	Net Monitor Mode with packet transmission	N	N	N	Y	Y	N
Generic	In-Channel Net Monitor mode	N	N	N	N	N	N

Parent topic: Wi-Fi radio

[1] Feature not enabled by default in the SDK. Refer to *Feature enable and memory impact* for the macro to enable the feature and the impact on the memory when enabling the feature. [2] The feature is used to compress the Wi-Fi Bluetooth firmware and optimize the flashing of the host [3] Refer to *10*.

Wi-Fi direct/P2P

Features	Sub features	88W89	IW41	IW611/IW	RW610/RW	IW61	AW6113
P2P basic functionality1	P2P Auto GO	Y	Y	Y	Y	Y	Y
P2P basic functionality1	P2P GO	Y	Y	Y	Y	Y	Y
P2P basic functionality1	P2P GC	Y	Y	Y	Y	Y	Y
P2P basic functionality1	P2P Persistent Group	Y	Y	Y	Y	Y	Y
P2P basic functionality1	P2P Invitation	Y	Y	Y	Y	Y	Y
P2P basic functionality1	P2P Device Discovery	Y	Y	Y	Y	Y	Y
P2P basic functionality1	P2P Provision Discovery	Y	Y	Y	Y	Y	Y
P2P basic functionality1	P2P simultaneous GO + STA	Y	Y	Y	Y	Y	Y
P2P basic functionality1	P2P simultaneous GC + uAP	Y	Y	Y	Y	Y	Y
P2P basic functionality1	P2P same channel simultaneous GO + uAP	Y	Y	Y	Y	Y	Y

Parent topic: Wi-Fi radio

[1] Feature not enabled by default in the SDK. Refer to *Feature enable and memory impact* for the macro to enable the feature and the impact on the memory when enabling the feature. [2] This is an experimental software release for this feature for IW416. [3] Contact your support representative to use this feature for.

Bluetooth radio

Bluetooth classic

Feature	Sub feature	88W8	IW41	IW611/	RW610/	IW61	AW611
General features	Bluetooth Class 1.5 and Class 2 support	Y	Y	Y	N	N	Y
General features	Scatternet support	Y	Y	Y	N	N	Y
General features	Maximum of seven simultaneous ACL connections – Central links	Y	Y	Y	N	N	Y
General features	Automatic packet type selection	Y	Y	Y	N	N	Y
General features	Bluetooth - 2.1 to 5.0 specification support	Y	Y	Y	N	N	Y
General features	Low power sniff	Y	Y	Y	N	N	Y
General features	Deep sleep using out-of-band	Y	Y	N	N	N	N
General features	Wake on Bluetooth (SoC to host)	Y	Y	Y	N	N	Y
General features	Independent reset (in-band)1	Y	Y	Y	Y	N	Y
General features	Independent reset (out-band)1	Y	Y	N	N	N	N
General features	Firmware download (parallel)1	Y	Y	N	N	N	N
General features	RF test mode	Y	Y	Y	N	N	Y
Bluetooth packet type supported	ACL (DM1, DH1, DM3, DH3, DM5, DH5, 2-DH1, 2-DH3, 2-DH5, 3-DH1, 3-DH3, 3-DH5)	Y	Y	Y	N	N	Y
Bluetooth packet type supported	SCO (HV1, HV3)	Y	Y	Y	N	N	Y
Bluetooth packet type supported	eSCO (EV3, EV4, EV5, 2EV3, 3EV3, 2EV5, 3EV5)	Y	Y	Y	N	N	Y
Bluetooth profiles supported	A2DP source/sink	Y	Y	Y	N	N	Y
Bluetooth profiles supported	AVRCP target/controller	Y	Y	Y	N	N	Y
Bluetooth profiles supported	HFP Dev/AG	Y	Y	Y	N	N	Y
Bluetooth profiles supported	OPP server/client	Y	Y	Y	N	N	Y
Bluetooth profiles supported	SPP server/client	Y	Y	Y	N	N	Y
Bluetooth profiles supported	HID target/device	Y	Y	Y	N	N	Y
Bluetooth audio features	PCM NBS central/peripheral	Y	Y	Y	N	N	Y
Bluetooth audio features	PCM WBS central/peripheral	Y	Y	Y	N	N	Y

Parent topic: Bluetooth radio

[1] Experimental feature intended for evaluation/early development only and not production. Incomplete mandatory certification.

Bluetooth LE

Features	Sub features
Generic features	Maximum 16 Bluetooth LE connections (central role)
Generic features	Deep sleep using out-of-band
Generic features	Wake on Bluetooth LE (SoC to Host)
Generic features	RF Test mode
Bluetooth profile support	Bluetooth LE GATT
Bluetooth profile support	Bluetooth LE HID over GATT
Bluetooth profile support	Bluetooth LE GAP
Bluetooth LE 4.0 support	Low Energy physical layer
Bluetooth LE 4.0 support	Low Energy link layer
Bluetooth LE 4.0 support	Enhancements to HCI for Low Energy
Bluetooth LE 4.0 support	Low energy direct test mode
Bluetooth 4.1 support	Low duty cycle directed advertising
Bluetooth 4.1 support	Bluetooth LE dual mode topology
Bluetooth 4.1 support	Bluetooth LE privacy v1.1
Bluetooth 4.1 support	Bluetooth LE link layer topology
Bluetooth 4.2 support	Bluetooth LE secure connection
Bluetooth 4.2 support	Bluetooth LE link layer privacy v1.2
Bluetooth 4.2 support	Bluetooth LE data length extension
Bluetooth 4.2 support	Link layer extended scanner filter policies
Bluetooth 5.0 support	Bluetooth LE 2 Mbps support
Bluetooth 5.0 support	High duty cycle directed advertising
Bluetooth 5.0 support	Low Energy advertising extension
Bluetooth 5.0 support	Low Energy long range
Bluetooth 5.0 support	Low Energy periodic advertisement
Bluetooth 5.2 support	Low Energy power control
Bluetooth LE audio support1 2	Isochronous channel
Bluetooth LE audio support1 2	Broadcast LE Audio BIS source
Bluetooth LE audio support1 2	Broadcast LE Audio BIS sink
Bluetooth LE audio support1 2	Broadcast LE Audio BIG Validation
Bluetooth LE audio support1 2	Broadcast LE Audio Phy: 1M/2M/ coded
Bluetooth LE audio support1 2	Broadcast LE Audio framed mode
Bluetooth LE audio support1 2	Broadcast LE Audio unframed mode
Bluetooth LE audio support1 2	Broadcast LE Audio sequential packing
Bluetooth LE audio support1 2	Broadcast LE Audio: Mono and Stereo
Bluetooth LE audio support1 2	Broadcast LE Audio BIS encrypted audio
Bluetooth LE audio support1 2	Broadcast LE Audio BIS unencrypted audio
Bluetooth LE audio support1 2	Unicast LE Audio CIS source
Bluetooth LE audio support1 2	Unicast LE Audio CIS sink
Bluetooth LE audio support1 2	Unicast LE Audio CIG validation
Bluetooth LE audio support1 2	Unicast LE Audio CIS synchronization
Bluetooth LE audio support1 2	Unicast LE Audio Phy: 1M/2M/ coded
Bluetooth LE audio support1 2	Unicast LE Audio framed mode
Bluetooth LE audio support1 2	Unicast LE Audio unframed mode
Bluetooth LE audio support1 2	Unicast LE Audio sequential packing
Bluetooth LE audio support1 2	Unicast LE Audio: mono and stereo
Bluetooth LE audio support1 2	Unicast LE Audio CIS encrypted audio
Bluetooth LE audio support1 2	Unicast LE Audio CIS unencrypted audio
Bluetooth LE audio support1 2	Unicast LE Audio TX/RX and bidirectional traffic

Table 3 – continued from prev

Features	Sub features
Bluetooth LE audio support1 2	ISO interval for LE Audio: 7.5ms 10ms 20ms 30ms
Bluetooth LE audio support1 2	Sampling frequency for LE Audio: 8kHz 16kHz 24kHz, 32kH
Bluetooth LE audio support1 2	LE Audio Auracast use cases: Auracast streaming 2 BISes
Bluetooth LE audio support1 2	LE Audio Unicast use cases: Unicast streaming 2 CISes
Bluetooth LE audio support1 2	LE Audio Unicast Use cases: Unicast streaming 4 CISes
Bluetooth LE audio support1 2	A2DP + Auracast/Unicast Bridge use cases – CIS/BIS
BCA TDM Coexistence mode (shared antenna)	STA + Bluetooth coexistence
BCA TDM Coexistence mode (shared antenna)	STA + Bluetooth LE coexistence
BCA TDM Coexistence mode (shared antenna)	STA + Bluetooth + Bluetooth LE coexistence
BCA TDM Coexistence mode (shared antenna)	AP + Bluetooth coexistence
BCA TDM Coexistence mode (shared antenna)	AP + Bluetooth LE coexistence
BCA TDM Coexistence mode (shared antenna)	AP + Bluetooth + Bluetooth LE coexistence
BCA TDM coexistence mode (separate antenna)	STA + Bluetooth coexistence
BCA TDM coexistence mode (separate antenna)	STA + Bluetooth LE coexistence
BCA TDM coexistence mode (separate antenna)	STA + Bluetooth + Bluetooth LE coexistence
BCA TDM coexistence mode (separate antenna)	AP + Bluetooth coexistence
BCA TDM coexistence mode (separate antenna)	AP + Bluetooth LE coexistence
BCA TDM coexistence mode (separate antenna)	AP + Bluetooth + Bluetooth LE coexistence

Note: Details of the tested Bluetooth LE Audio use cases:

- Number of streams:
  - 1-CIG | upto 4-CIS with 1 LE ACL (for 4-CIS: execute only mono UCs, SDU Int: 10ms)
  - 1-CIG | upto 4-CIS with 4 separate LE ACL (for 4-CIS: SDU Size= Max 100 Oct, PHY=2M, RTN=1, SDU Int: 10ms only) (execute only mono UCs for 4-CIS)
  - 1-BIG | upto 4-BIS (for 4-BIS: execute only mono UCs, SDU Int: 10ms only)
- PHY: 2M and 1M
- Audio mode: mono (for 1 to 4 streams) and stereo (for 1 stream)
- Packing: sequential and interleaved
- Bit rate: maximum 96kbps
  - For 1-CIG with upto 3-CIS: maximum bit rate 96kbps
  - For 1-CIG with 4-CIS: maximum bit rate 80kbps
  - For 1-BIG with 4-BIS: maximum bit rate 80kbps
  - For 2-CIG cases: maximum bit rate 80kbps
- Mode: unframed mode
- 48_5 and 48_6 mono and stereo configurations are not supported.

Details of the tested Bluetooth coexistence (Bluetooth + Bluetooth LE Audio) use cases:

- Bluetooth + Bluetooth LE Audio
- A2DP + Bluetooth LE Audio bridging support
- A2DP sink link (central) -> LEA 2-CIS (SDU Int: 10ms only | A2DP only with SBC Codec | PHY: 2M)

Parent topic: Bluetooth radio

[1] Experimental feature intended for evaluation/early development only and not production. Incomplete mandatory certification.

[2] LE audio feature is supported for standalone scenarios only and not for BR/EDR and Wi-Fi co-existence scenarios such as LE audio + BR/EDR link or LE audio + Wi-Fi link. From the perspective

of NXP Edgefast Bluetooth host stack, LE audio feature can be disabled by the CONFIG_BT_AUDIO macro without impact on any other features. LE audio feature can be tested by the user, using their own supported host stack.

Parent topic: *Features*

802.15.4 radio

Features	Sub features	IW612	IW610	RW612
General features	Spinel over SPI	Y	N	N
General features	OpenThread RCP Mode implementing Thread1.3	Y	N	N
General features	802.15.4-2015 MAC/PHY as required by Thread 1.3	Y	Y	Y
General features	OpenThread Border Router (OTBR) v1.1	Y	Y	Y
General features	Direct/indirect transmission with/without ACK	Y	Y	Y
General features	802.15.4 CSL parent feature implementation	Y	Y	Y
General features	Enhanced Frame Pending	Y	Y	Y
General features	Enhanced keep alive	Y	Y	Y
General features	Router	Y	Y	Y
General features	Leader	Y	Y	Y
General features	Router Eligible End Device (REED)	Y	Y	Y
General features	End Device (FED, MED)	Y	Y	Y
Zigbee features	Coordinator	N	N	Y
Zigbee features	Router	N	N	Y
Zigbee features	End Device (RX ON)	N	N	Y
Zigbee features	R23	N	N	Y
Zigbee features	OTA Client	N	N	Y
Zigbee features	OTA server	N	N	Y
Matter features	Matter over Wi-Fi	Y	N	N
Matter features	Matter over Thread	Y	N	Y

Parent topic: *Features*

Coexistence

Wi-Fi and Bluetooth/802.15.4 coexistence

Features	Sub features	IW6´	IW6´	RW612
BCA_TDM separate antenna1 (lower and higher isolation) 1x1 Wi-Fi, (Bluetooth and 802.15.4 shared)	STA + Bluetooth	Y	N	N
BCA_TDM separate antenna1 (lower and higher isolation) 1x1 Wi-Fi, (Bluetooth and 802.15.4 shared)	Mobile AP + Bluetooth	Y	N	N
BCA_TDM separate antenna1 (lower and higher isolation) 1x1 Wi-Fi, (Bluetooth and 802.15.4 shared)	Bluetooth LE + Wi-Fi	Y	Y	Y
BCA_TDM separate antenna1 (lower and higher isolation) 1x1 Wi-Fi, (Bluetooth and 802.15.4 shared)	Bluetooth + Bluetooth LE + Wi-Fi	Y	N	N
BCA_TDM separate antenna1 (lower and higher isolation) 1x1 Wi-Fi, (Bluetooth and 802.15.4 shared)	OpenThread + Bluetooth	Y	N	N
BCA_TDM separate antenna1 (lower and higher isolation) 1x1 Wi-Fi, (Bluetooth and 802.15.4 shared)	OpenThread + Bluetooth LE2	Y	Y	Y
BCA_TDM separate antenna1 (lower and higher isolation) 1x1 Wi-Fi, (Bluetooth and 802.15.4 shared)	OpenThread + Bluetooth + Bluetooth LE	Y	N	N
BCA_TDM separate antenna1 (lower and higher isolation) 1x1 Wi-Fi, (Bluetooth and 802.15.4 shared)	OpenThread + Wi-Fi	Y	Y	Y
BCA_TDM separate antenna1 (lower and higher isolation) 1x1 Wi-Fi, (Bluetooth and 802.15.4 shared)	Bluetooth + OpenThread + Wi-Fi	Y	N	N
BCA_TDM separate antenna1 (lower and higher isolation) 1x1 Wi-Fi, (Bluetooth and 802.15.4 shared)	Bluetooth LE + OpenThread + Wi-Fi	Y	Y	Y
BCA_TDM separate antenna1 (lower and higher isolation) 1x1 Wi-Fi, (Bluetooth and 802.15.4 shared)	Bluetooth + Bluetooth LE + OpenThread + Wi-Fi	Y	N	N
BCA_TDM separate antenna1 (lower and higher isolation) 1x1 Wi-Fi, (Bluetooth and 802.15.4 shared)	Single antenna configuration	Y	Y	Y
BCA_TDM separate antenna1 (lower and higher isolation) 1x1 Wi-Fi, (Bluetooth and 802.15.4 shared)	External Coexistence PTA	N	Y	Y

Parent topic: Coexistence

[1] Experimental feature intended for evaluation/early development only and not production. Incomplete mandatory certification.

[2] The narrow-band radio can be configured to support Bluetooth LE, 802.15.4, and to time-slice between Bluetooth LE and 802.15.4.

Parent topic: *Features*

Feature enable and memory impact

Features	Macros to enable the feature	Memory impact
CSI	CONFIG_CSI	Flash - 60K, RAM - 4K
AMI	CONFIG_CSI_AMI3	Flash - 2032K, RAM - 772K
DPP	CONFIG_WPA_SUPP_DPP	Flash - 240K, RAM - 12K
Independent reset	CONFIG_WIFI_IND_DNLDCONFIG_WIFI_IND_RESET	Minimal
Parallel firmware download Wi-Fi	CONFIG_WIFI_IND_DNLD	Minimal
Parallel firmware download Bluetooth	CONFIG_BT_IND_DNLD	Minimal
WPA3 enterprise	CONFIG_WPA_SUPP_CRYPTO_ENTERPRISE [Macros specific to EAP-methods included] CONFIG_EAP_TLS CONFIG_EAP_PEAP CONFIG_EAP_TTLS CONFIG_EAP_FAST CONFIG_EAP_SIM CONFIG_EAP_AKA CONFIG_EAP_AKA_PRIME	Flash - 165K, RAM - 18K
WPA2 enterprise	CONFIG_WPA_SUPP_CRYPTO_ENTERPRISE [Macros specific to EAP-methods included] CONFIG_EAP_TLS CONFIG_EAP_PEAP CONFIG_EAP_TTLS CONFIG_EAP_FAST CONFIG_EAP_SIM CONFIG_EAP_AKA CONFIG_EAP_AKA_PRIME	Flash - 165K, RAM - 18K
Host sleep	CONFIG_HOST_SLEEP	Minimal
WMM	CONFIG_WMM1	Flash - 10K, RAM - 57K
802.11mc	CONFIG_11MC CONFIG_CSI CONFIG_WLS_CSI_PROC2 CONFIG_11AZ	Flash: 52.78KB, RAM : 121.1KB
802.11az	CONFIG_11MC CONFIG_CSI[2] CONFIG_WLS_CSI_PROC2 CONFIG_11AZ	Flash: 52.78KB, RAM : 121.1KB
Non-blocking firmware download mechanism	CONFIG_FW_DNLD_ASYNC	—
Antenna diversity	CONFIG_WLAN_CALDATA_2ANT_DIVERSITY	-
P2P	CONFIG_WPA_SUPP_P2P	-

Note:

- For Wi-Fi, the macros are set with the value "**0**" by default in the file wifi_config_default.h

located in <SDK_PATH>/middleware/wifi_nxp/incl/ directory.

To enable the features, set the value of the macros to "***1****" in the file wifi_config.h located in*<SDK_Wi-Fi_Example_PATH>/ *directory****.***

- Bluetooth

  To enable the features, set the value of the macros to "**1**" in the file app_bluetooth_config.h located in <SDK_Bluetooth_Example_PATH>/ directory.

[1] The macro is not used for IW416.

[2] Prerequisite macros for 802.11mc and 802.11az features

[3] Enable PRINTF_FLOAT_ENABLE only for MCUXpresso IDE and specifically for the RT1060-EVKC and RT1170-EVKB platforms

- Go to project properties > C/C++ Build > Settings > Preprocessor.
- Add PRINTF_FLOAT_ENABLE=1

88W8987 release notes

Package information

- SDK version: 26.09.00

Parent topic: *88W8987 release notes*

Version information

- Wireless SoC: 88W8987
- Wi-Fi and Bluetooth/Bluetooth LE firmware version: 16.92.21.p156.4
  - 16 - Major revision
  - 92 - Feature pack
  - 21 - Release version
  - p156.4 - Patch number

Parent topic: *88W8987 release notes*

Host platform

- All i.MX RT platforms running FreeRTOS.
- Host interfaces
  - Wi-Fi over SDIO (SDIO 2.0 support, SDIO clock frequency: 50 MHz)
  - Bluetooth/Bluetooth LE over UART
- Test tools
  - iPerf (version 2.1.9)

Parent topic: *88W8987 release notes*

Wi-Fi and Bluetooth certification

The Wi-Fi and Bluetooth certification is obtained with the following combinations.

WFA certifications

- STA | 802.11n

- STA | 802.11ac
- STA | PMF
- STA | FFD
- STA | SVD
- STA | WPA2
- STA | WPA3
- STA | QTT

Refer to *6*.

Note: This release supports STAUT only certifications.

Parent topic: Wi-Fi and Bluetooth certification

Bluetooth controller certification

QDID: refer to *4*.

Parent topic: Wi-Fi and Bluetooth certification

Parent topic: *88W8987 release notes*

Wi-Fi throughput

Throughput test setup

- Environment: Shield Room - Over the Air
- External Access Point: ASUS AX88U
- DUT: W8987 Murata (Module: **1ZM M.2**) with EVK-MIMXRT1060 EVKC platform
- DUT Power Source: External power supply
- External Client: Apple MacBook Air
- Channel: 6 | 36
- Wi-Fi application: wifi_wpa_supplicant
- Compiler used to build application: armgcc
- Compiler Version: gcc-arm-none-eabi-13.2
- iPerf commands used in test:

TCP TX

```
iperf -c <remote_ip> -t 60
```

TCP RX

```
iperf -s
```

UDP TX

```
iperf -c <remote_ip> -t 60 -u -B <local_ip> -b 120
```

Note: The default rate is 100 Mbps.

UDP RX

```
iperf -s -u -B <local_ip>
```

Note: Read more about the throughput test setup and topology in *2*.

Parent topic: Wi-Fi throughput

STA throughput

External APs: ASUS AX88U

STA mode throughput - BGN Mode | 2.4 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	50	50	60	62
WPA2-AES	47	49	59	62
WPA3-SAE	49	48	59	62

STA mode throughput - BGN Mode | 2.4 GHz Band | 40 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	62	83	121	124
WPA2-AES	61	82	120	126
WPA3-SAE	60	82	120	126

STA mode throughput - AN Mode | 5 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	43	52	60	64
WPA2-AES	43	52	61	64
WPA3-SAE	43	52	60	65

STA mode throughput - AN Mode | 5 GHz Band | 40 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	64	87	126	125
WPA2-AES	63	85	125	120
WPA3-SAE	63	80	125	123

STA mode throughput - AC Mode | 5 GHz Band | 20 MHz (VHT)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	48	60	73	78
WPA2-AES	47	60	73	77
WPA3-SAE	47	60	73	77

STA mode throughput - AC Mode | 5 GHz Band | 40 MHz (VHT)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	68	96	161	157
WPA2-AES	69	92	160	155
WPA3-SAE	70	94	160	155

STA mode throughput - AC Mode | 5 GHz Band | 80 MHz (VHT)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	116	111	226	233
WPA2-AES	114	99	224	200
WPA3-SAE	114	99	226	200

Parent topic: Wi-Fi throughput

Mobile AP throughput

External client: Apple Macbook Air

Mobile AP Mode Throughput - BGN Mode | 2.4 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	47	51	58	62
WPA2-AES	46	50	58	62
WPA3-SAE	44	50	58	62

Mobile AP Mode Throughput - BGN Mode | 2.4 GHz Band | 40 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	66	81	107	121
WPA2-AES	65	80	107	120
WPA3-SAE	65	80	108	120

Mobile AP Mode Throughput - AN Mode | 5 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	44	52	60	61
WPA2-AES	44	51	60	61
WPA3-SAE	44	51	60	61

Mobile AP Mode Throughput - AN Mode | 5 GHz Band | 40 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	70	89	126	103
WPA2-AES	70	87	124	102
WPA3-SAE	70	88	125	103

Mobile AP Mode Throughput - AC Mode | 5 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	49	60	73	76
WPA2-AES	48	59	73	76
WPA3-SAE	48	60	73	76

Mobile AP Mode Throughput - AC Mode | 5 GHz Band | 40 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	77	106	161	102
WPA2-AES	77	104	160	102
WPA3-SAE	77	104	160	111

Mobile AP Mode Throughput - AC Mode | 5 GHz Band | 80 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	115	135	226	215
WPA2-AES	113	123	224	196
WPA3-SAE	113	123	225	172

Parent topic: Wi-Fi throughput

Parent topic: *88W8987 release notes*

EU conformance tests

- EU Adaptivity test - EN 300 328 v2.1.1 (for 2.4 GHz)
- EU Adaptivity test - EN 301 893 v2.1.1 (for 5 GHz)

Parent topic: *88W8987 release notes*

Bug fixes and/or feature enhancements

Firmware version: From 16.91.21.p64.1 to 16.91.21.p82

Component	Description
Wi-Fi	WPA3-R3 enabled APUT beacons does not have RSNXE when configured in H2E mode-Associated event is received even when connecting using wrong password WFA APUT Low iperf TCP/UDP Tx throughput with Realtek station

Parent topic: Bug fixes and/or feature enhancements

Firmware version: From 16.91.21.p82 to 16.91.21.p91.6

Component	Description
Wi-Fi	In wrong password scenario, After updating new password the phone is not able to connect with DUTAP

Parent topic: Bug fixes and/or feature enhancements

Firmware version: From 16.91.21.p91.6 to 16.91.21.p124

Component	Description
Wi-Fi	Cloud keep alive packets not seen after DUT enters host sleep. DUT is sending QOS null packets even in host sleep

Parent topic: Bug fixes and/or feature enhancements

Firmware version: From 16.91.21.p124 to 16.91.21.p133

Component	Description
Wi-Fi	Samsung S24 Ultra and Google Pixel 7 mobiles having Android 14 are not able connect to the DUTAP with WPA3 SAE security.

Parent topic: Bug fixes and/or feature enhancements

Firmware version: From 16.91.21.p133 to 16.91.21.p142.5

Component	Description
Wi-Fi	Fails to encrypt and decrypt data with ccmp 128 and 256 using CLI crypto commands.

Parent topic: Bug fixes and/or feature enhancements

Firmware version: From 16.91.21.p142.5 to 16.91.21.p149.2

Component	Description
Wi-Fi	DUTSTA does not associate to hidden SSID beaconing in DFS channel.

Parent topic: Bug fixes and/or feature enhancements

Firmware version: From 16.91.21.p149.2 to 16.92.21.p151.7

Component	Description
Wi-Fi	Getting low TCP/UDP TP in DUT-AP 11ac-vht80 mode after hard-reset or wlan-reset.

Parent topic: Bug fixes and/or feature enhancements

Firmware version: From 16.91.21.p149.2 to 16.92.21.p151.7

Component	Description
Wi-Fi	Getting low TCP/UDP TP in DUT-AP 11ac-vht80 mode after hard-reset or wlan-reset.

Parent topic: Bug fixes and/or feature enhancements

Firmware version: From 16.92.21.p151.7 to 16.92.21.p153.5

Component	Description
Wi-Fi	Added P2P Persistance and P2P Invitation

Parent topic: Bug fixes and/or feature enhancements

Firmware version: From 16.92.21.p153.5 to 16.92.21.p153.6

Component	Description
Wi-Fi	Enabled mbedtls 3.x

Parent topic: Bug fixes and/or feature enhancements

Firmware version: From 16.92.21.p153.6 to 16.92.21.p153.9

Component	Description
Wi-Fi	Enabled mbedtls 3.xThe HT/VHT Wi-Fi capabilities are not being properly configured when the Device Under Test (DUT) initiates operation as a P2P Group Owner (P2P-GO).Resolved RT685 Wi-Fi EAP-TLS crash caused by expired certificates

Parent topic: Bug fixes and/or feature enhancements

Firmware version: From 16.92.21.p153.9 to 16.92.21.p155.2

Component	Description
Wi-Fi	Enabled mbedtls 3.x

Parent topic: Bug fixes and/or feature enhancements

Firmware version: From 16.92.21.p155.2 to 16.92.21.p156.1

Component	Description
Wi-Fi	Fixed roaming failure (RRM-NEIGHBOR-REP-REQUEST-FAILED) when DUT roams between APs on different 5 GHz channels on i.MXRT685 platform.
Bluetooth	Fixed same EN_RAND value being generated on every AES-CCM encryption, causing predictable random number in BT pairing.

Parent topic: Bug fixes and/or feature enhancements

Firmware version: From 16.92.21.p156.1 to 16.92.21.p156.4

Component	Description
Wi-Fi	Fixed an issue where DHCP renewal was not triggered when roaming between independent APs sharing the same SSID. Fixed an issue where the DUT failed to fall back to legacy roaming after 802.11k/11v roaming returned no usable target AP

Parent topic: Bug fixes and/or feature enhancements

Parent topic: *88W8987 release notes*

Known issues

Component	Description
NA	

Parent topic: *88W8987 release notes*

IW416 release notes

Package information

- SDK version: 26.09.00

Parent topic: *IW416 release notes*

Version information

- Wireless SoC: IW416
- Wi-Fi and Bluetooth/Bluetooth LE firmware version: 16.92.21.p166
  - 16 - Major revision
  - 92 - Feature pack
  - 21 - Release version
  - p166 - Patch number

Parent topic: *IW416 release notes*

Host platform

- All i.MX RT platforms running FreeRTOS.
- Host interfaces
  - Wi-Fi over SDIO (SDIO 2.0 Support, SDIO clock frequency: 50 MHz)
  - Bluetooth/Bluetooth LE over UART
- Test tools
  - iPerf (version 2.1.9)

Parent topic: *IW416 release notes*

Wi-Fi and Bluetooth certification

The Wi-Fi and Bluetooth certification is obtained with the following combinations.

WFA certifications

- STA | 802.11n
- STA | PMF
- STA | FFD
- STA | SVD
- STA | WPA2
- STA | WPA3
- STA | QTT

Refer to *6*.

Note: This release supports STAUT only certifications.

Parent topic: Wi-Fi and Bluetooth certification

Bluetooth controller certification

QDID: refer to *4*.

Note: QDID upgrade to Bluetooth Core Specification Version 5.4 is in progress.

Parent topic: Wi-Fi and Bluetooth certification

Parent topic: *IW416 release notes*

Wi-Fi throughput

Throughput test setup

- Environment: Shield Room - Over the Air
- Access Point: Asus AX88u
- DUT: IW416 Murata (Module: 1XK M.2) with EVK-MIMXRT1060 EVKC platform
- DUT Power Source: External power supply
- Client: Apple MacBook Air
- Channel: 6 | 36
- Wi-Fi application: wifi_wpa_supplicant
- Compiler used to build application: armgcc
- Compiler Version: gcc-arm-none-eabi-13.2

- iPerf commands used in test:

TCP TX

```
iperf -c <remote_ip> -t 60
```

TCP RX

```
iperf -s
```

UDP TX

```
iperf -c <remote_ip> -t 60 -u -B <local_ip> -b 120
```

Note: The default rate is 100 Mbps.

UDP RX

```
iperf -s -u -B <local_ip>
```

Note: Read more about the throughput test setup and topology in *2*.

Parent topic: Wi-Fi throughput

STA throughput

External AP: Asus AX88u

STA mode throughput - BGN Mode | 2.4 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	43	45	53	54
WPA2-AES	42	45	51	53
WPA3-SAE	45	42	53	53

STA mode throughput - BGN Mode | 2.4 GHz Band | 40 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	41	59	115	123
WPA2-AES	50	58	113	93
WPA3-SAE	28	58	114	92

STA mode throughput - AN Mode | 5 GHz Band | 20 MHz (HT)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	45	48	61	56
WPA2-AES	42	48	61	44
WPA3-SAE	42	47	61	49

STA mode throughput - AN Mode | 5 GHz Band | 40 MHz (HT)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	43	64	118	132
WPA2-AES	50	59	118	131
WPA3-SAE	69	59	117	122

Parent topic: Wi-Fi throughput

Mobile AP throughput

External client: Apple MacBook Air

Mobile AP mode throughput - BGN Mode | 2.4 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	43	45	53	54
WPA2-AES	42	45	51	53
WPA3-SAE	45	42	53	53

Mobile AP mode throughput - BGN Mode | 2.4 GHz Band | 40 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	68	66	96	91
WPA2-AES	68	66	96	91
WPA3-SAE	68	66	96	91

Mobile AP mode throughput - AN Mode | 5 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	51	45	57	61
WPA2-AES	42	45	55	56
WPA3-SAE	43	45	57	56

Mobile AP mode throughput - AN Mode | 5 GHz Band | 40 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	83	68	118	101
WPA2-AES	77	68	118	100
WPA3-SAE	77	69	118	100

Parent topic: Wi-Fi throughput

Parent topic: *IW416 release notes*

EU conformance tests

- EU Adaptivity test - EN 300 328 v2.1.1 (for 2.4 GHz)

- EU Adaptivity test - EN 301 893 v2.1.1 (for 5 GHz)

Parent topic: *IW416 release notes*

Bug fixes and/or feature enhancements

Firmware version: From 16.91.21.p64.1 to 16.91.21.p82

Component	Description
Wi-Fi	WPA3-R3 enabled APUT beacons does not have RSNXE when configured in H2E mode

Parent topic: Bug fixes and/or feature enhancements

Firmware version: From 16.91.21.p82 to 16.91.21.p91.6

Component	Description
Wi-Fi	NA

Parent topic: Bug fixes and/or feature enhancements

Firmware version: From 16.91.21.p91.6 to 16.91.21.p124

Component	Description
Wi-Fi	Cloud keep alive packets not seen after DUT enters host sleep. DUT is sending QOS null packets even in host sleep

Parent topic: Bug fixes and/or feature enhancements

Firmware version: From 16.91.21.p124 to 16.91.21.p133

Component	Description
Wi-Fi	NA

Parent topic: Bug fixes and/or feature enhancements

Firmware version: From 16.91.21.p133 to 16.91.21.p133.2

Component	Description
Wi-Fi	DUT STA getting rebooted after 15~20 iterations of 11R-Command based roaming0xa4 command timeout after several hours of stress test

Parent topic: Bug fixes and/or feature enhancements

Firmware version: From 16.91.21.p133.2 to 16.91.21.p142.5

Component	Description
Wi-Fi	DUT fails to reconnect after the configured auto-reconnect time interval.
Coex	During HFP call, TX side noise is observed with coex CLI

Parent topic: Bug fixes and/or feature enhancements

Firmware version: From 16.91.21.p142.5 to 16.91.21.p149.4

Component	Description
-	NA

Parent topic: Bug fixes and/or feature enhancements

Firmware version: From 16.91.21.p149.4 to 16.92.21.p151.7

Component	Description
Wi-Fi	Samsung S24 Ultra and Google Pixel 7 mobiles having Android 14 are not able connect to the DUTAP with WPA3 SAE security.

Parent topic: Bug fixes and/or feature enhancements

Firmware version: From 16.92.21.p151.7 to 16.92.21.p153.5

Component	Description
Wi-Fi	The DUT encounters a command response timeout during the execution of the wlan-info command following UDP traffic tests.
Wi-Fi	Random hang issue seen when using wlan-p2p-find/stop in succession

Parent topic: Bug fixes and/or feature enhancements

Firmware version: From 16.92.21.p153.5 to 16.92.21.p153.6

Component	Description
Wi-Fi	Enabled mbedtls 3.x

Parent topic: Bug fixes and/or feature enhancements

Firmware version: From 16.92.21.p153.6 to 16.92.21.p153.9

Component	Description
Wi-Fi	Enabled mbedtls 3.x

Parent topic: Bug fixes and/or feature enhancements

Firmware version: From 16.92.21.p153.9 to 16.92.21.p155.2

Component	Description
Wi-Fi	Enabled mbedtls 3.xWi-Fi connection in 2.4GHz is not stable, observed deauthentication within 10sec.

Parent topic: Bug fixes and/or feature enhancements

Firmware version: From 16.92.21.p155.2 to 16.92.21.p161

Component	Description
Wi-Fi	Fixed STA unable to connect to AP in noisy conditions with antenna diversity enabled.
Bluetooth	Intermittent failure in LE connection establishment with reason code 0x3E

Parent topic: Bug fixes and/or feature enhancements

Firmware version: From 16.92.21.p161 to 16.92.21.p166

Component	Description
Wi-Fi	Fixed an issue where DHCP renewal was not triggered when roaming between independent APs sharing the same SSID. Fixed an issue where the DUT failed to fall back to legacy roaming after 802.11k/11v roaming returned no usable target AP

Parent topic: Bug fixes and/or feature enhancements

Parent topic: *IW416 release notes*

Known issues

Component	Description
NA	

Parent topic: *IW416 release notes*

IW611/IW612 release notes

Package information

- SDK version: 26.09.00

Parent topic: *IW611/IW612 release notes*

Version information

- Wireless SoC: IW611/IW612
- Wi-Fi and Bluetooth/Bluetooth LE firmware version: 18.99.8.p141
  - 18 - Major revision

  - 99 - Feature pack
  - 8 - Release version
  - p141 - Patch number

Parent topic: *IW611/IW612 release notes*

Host platform

- All i.MX RT platforms running FreeRTOS except i.MX RT1060 EVKA and i.MX RT1064 EVK
- Host interfaces
  - Wi-Fi over SDIO (SDIO 2.0 support, SDIO clock frequency: 50 MHz)
  - Bluetooth/Bluetooth LE over UART
  - 802.15.4 over SPI (IW612 only)
- Test tools
  - iPerf (version 2.1.9)

Parent topic: *IW611/IW612 release notes*

Wi-Fi and Bluetooth certification

The Wi-Fi and Bluetooth certification is obtained with the following combinations.

WFA certifications

- STA | 802.11n
- STA | PMF
- STA | FFD
- STA | SVD
- STA | WPA2
- STA | WPA3
- STA | 802.11ac
- STA | 802.11ax
- STA | QTT

Refer to *6*.

Note: This release supports STAUT only certifications.

Parent topic: Wi-Fi and Bluetooth certification

Bluetooth controller certification

QDID: refer to *4*.

Note: QDID upgrade to Bluetooth Core Specification Version 5.4 is in progress.

Parent topic: Wi-Fi and Bluetooth certification

Parent topic: *IW611/IW612 release notes*

Wi-Fi throughput

Throughput test setup

- Environment: Shield Room - Over the Air
- Access Point: Asus AX88u
- DUT: IW612 Murata (Module: 2EL M.2) with EVK-MIMXRT1060 EVKC platform
- DUT Power Source: External power supply
- Client: Apple MacBook Air
- Channel: 6 | 36
- Wi-Fi application: wifi_wpa_supplicant
- Compiler used to build application: armgcc
- Compiler Version gcc-arm-none-eabi-13.2
- iPerf commands used in test:

TCP TX

```
iperf -c <remote_ip> -t 60
```

TCP RX

```
iperf -s
```

UDP TX

```
iperf -c <remote_ip> -t 60 -u -B <local_ip> -b 120
```

Note: The default rate is 100 Mbps.

UDP RX

```
iperf -s -u -B <local_ip>
```

Note: Read more about the throughput test setup and topology in *2*

The throughput numbers are captured with default configurations using *wifi_wpa_supplicant* sample application.

Parent topic: Wi-Fi throughput

iPerf host configuration and impact on throughput

To get the highest throughput, the throughput values shown in STA throughput and Mobile AP throughput are measured with the maximum values of the default host configuration macros. STA and AP throughput captured with the minimum values of the host configuration macros shows the throughput numbers obtained when using the minimum values of the host configuration macros. The macro values are defined in *lwipopts.h* file.

The table below lists the minimum and maximum values of the host configuration macros.

Values of the host configuration macros

Parameter	Maximum value	Minimum value
TCPIP_MBOX_SIZE	96	32
DEFAULT_RAW_RECVMBOX_SIZE	32	12
DEFAULT_UDP_RECVMBOX_SIZE	64	12
DEFAULT_TCP_RECVMBOX_SIZE	64	12
TCP_MSS	1460	536
TCP_SND_BUF	24 * TCP_MSS	2 * TCP_MSS
MEM_SIZE	319160	41,080
TCP_WND	15 * TCP_MSS	10 * TCP_MSS
MEMP_NUM_PBUF	20	10
MEMP_NUM_TCP_SEG	96	12
MEMP_NUM_TCPIP_MSG_INPKT	80	16
MEMP_NUM_TCPIP_MSG_API	80	8
MEMP_NUM_NETBUF	32	16

STA and AP throughput captured with the minimum values of the host configuration macros

STA mode throughput - HE Mode | 5 GHz Band | 80 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	7	18	111	124
WPA2-AES	7	18	110	124
WPA3-SAE	6	18	110	124

Mobile AP mode throughput - HE Mode | 5 GHz Band | 80 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	2	19	93	127
WPA2-AES	2	19	105	126
WPA3-SAE	2	19	104	132

Parent topic: iPerf host configuration and impact on throughput

Parent topic: Wi-Fi throughput

STA throughput

External AP: Asus AX88u

STA mode throughput - BGN Mode | 2.4 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	46	46	53	62
WPA2-AES	54	55	70	72
WPA3-SAE	53	55	70	72

STA mode throughput - BGN Mode | 2.4 GHz Band | 40 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	77	77	130	131
WPA2-AES	80	79	130	129
WPA3-SAE	80	79	130	129

STA mode throughput - AN Mode | 5 GHz Band | 20 MHz (HT)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	41	48	63	64
WPA2-AES	56	58	71	75
WPA3-SAE	56	58	71	75

STA mode throughput - AN Mode | 5 GHz Band | 40 MHz (HT)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	78	81	129	134
WPA2-AES	78	80	129	133
WPA3-SAE	77	80	129	133

STA mode throughput - VHT Mode | 5 GHz Band | 20 MHz (VHT)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	45	54	71	76
WPA2-AES	56	58	71	75
WPA3-SAE	56	58	71	75

STA mode throughput - VHT Mode | 5 GHz Band | 40 MHz (VHT)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	92	86	153	157
WPA2-AES	89	91	153	157
WPA3-SAE	90	87	153	156

STA mode throughput - VHT Mode | 5 GHz Band | 80 MHz (VHT)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	120	93	233	226
WPA2-AES	114	104	233	226
WPA3-SAE	113	104	233	226

STA mode throughput - HE Mode | 2.4 GHz Band | 20 MHz (HE)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	69	53	95	102
WPA2-AES	63	65	93	95
WPA3-SAE	66	68	101	99

STA mode throughput - HE Mode | 2.4 GHz Band | 40 MHz (HE)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	97	88	214	210
WPA2-AES	99	88	204	209
WPA3-SAE	95	86	216	210

STA mode throughput - HE Mode | 5 GHz Band | 20 MHz (HE)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	59	51	93	127
WPA2-AES	60	51	91	126
WPA3-SAE	60	51	91	127

STA mode throughput - HE Mode | 5 GHz Band | 40 MHz (HE)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	103	90	185	228
WPA2-AES	102	91	180	228
WPA3-SAE	101	88	180	228

STA mode throughput - HE Mode | 5 GHz Band | 80 MHz (HE)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	123	97	233	226
WPA2-AES	123	70	233	225
WPA3-SAE	126	70	233	225

Parent topic: Wi-Fi throughput

Mobile AP throughput

External client: Apple MacBook Air

Mobile AP mode throughput - BGN Mode | 2.4 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	44	45	58	57
WPA2-AES	43	44	60	56
WPA3-SAE	43	44	59	56

Mobile AP mode throughput - BGN Mode | 2.4 GHz Band | 40 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	69	44	122	114
WPA2-AES	68	43	120	113
WPA3-SAE	68	43	120	112

Mobile AP mode throughput - AN Mode | 5 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	46	49	63	61
WPA2-AES	46	49	63	60
WPA3-SAE	46	49	63	60

Mobile AP mode throughput - AN Mode | 5 GHz Band | 40 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	75	78	127	113
WPA2-AES	74	78	127	112
WPA3-SAE	75	78	127	112

Mobile AP mode throughput - VHT Mode | 5 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	50	52	71	71
WPA2-AES	49	51	71	71
WPA3-SAE	50	51	71	71

Mobile AP mode throughput - VHT Mode | 5 GHz Band | 40 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	90	90	152	141
WPA2-AES	89	89	148	139
WPA3-SAE	89	89	144	140

Mobile AP mode throughput - VHT Mode | 5 GHz Band | 80 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	116	110	224	214
WPA2-AES	116	110	219	214
WPA3-SAE	115	110	219	214

Mobile AP mode throughput - HE Mode | 2.4 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	54	48	82	89
WPA2-AES	52	62	81	113
WPA3-SAE	51	62	81	112

Mobile AP mode throughput - HE Mode | 2.4 GHz Band | 40 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	83	76	133	180
WPA2-AES	62	74	132	132
WPA3-SAE	62	72	130	133

Mobile AP mode throughput - HE Mode | 5 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	60	52	81	89
WPA2-AES	60	51	80	90
WPA3-SAE	60	51	80	89

Mobile AP mode throughput - HE Mode | 5 GHz Band | 40 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	98	96	150	209
WPA2-AES	97	95	147	214
WPA3-SAE	97	95	147	214

Mobile AP mode throughput - HE Mode | 5 GHz Band | 80 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	116	107	222	216
WPA2-AES	115	106	217	216
WPA3-SAE	113	106	217	217

Parent topic: Wi-Fi throughput

Parent topic: *IW611/IW612 release notes*

EU conformance tests

- EU Adaptivity test - EN 300 328 v2.1.1 (for 2.4 GHz)
- EU Adaptivity test - EN 301 893 v2.1.1 (for 5 GHz)

Parent topic: *IW611/IW612 release notes*

Bug fixes and/or feature enhancements

Firmware version: 18.99.2.p7.19

Component	Description
-	NA

Parent topic: Bug fixes and/or feature enhancements

Firmware version: 18.99.2.p7.19 to 18.99.2.p49.9

Component	Description
-	NA

Parent topic: Bug fixes and/or feature enhancements

Firmware version: 18.99.2.p49.9 to 18.99.2.p155

Component	Description
Bluetooth	Audio lost occurs due to periodic adv sync lost, during 2 BIS 44.1kHz unencrypted streams with 1M PHY configuration.BIS sync loss may occur in long audio streaming sessions.

Parent topic: Bug fixes and/or feature enhancements

Firmware version: 18.99.2.p155 to 18.99.2.p66.30

Component	Description
Wi-Fi	802.11R Fast BSS roaming works only with hostapd and does not work with standard APs (supporting 11R)
Bluetooth	DUT is not able to sustain a connection with the remote device that does extended advertisement with coded PHY configuration. When 2 CIS streams are active, after the first device disconnects followed by the second device disconnecting, the second peripheral device hangs.Audio Play/Pause does not work in BIS case.

Parent topic: Bug fixes and/or feature enhancements

Firmware version: 18.99.2.p66.30 to 18.99.3.p10.5

Component	Description
Wi-Fi	STAUT not sending Neighbor Advertisement packet after receiving Neighbor Solicitation packet from Ex-AP.Antenna selection time exceeds configured evaluation time
Bluetooth	When DUT works as CIS source and CIS Offset is 612us, high packet drops observed which affects the audio streaming.For BIS Source Use Cases, Periodic Interval and ISO Interval should be multiple of each other value.In 1-CIS and 2-CIS, Continuous Audio Glitches are observed with 96 kbps bit rate.

Parent topic: Bug fixes and/or feature enhancements

Firmware version: 18.99.3.p10.5 to 18.99.3.p17.9

Component	Description
Wi-Fi	After performing independent reset (out-of-band mode), the STAUT fails to connect to the external AP via wlan-connect command, observed command timeout 0x107 error.
Bluetootl	Audio glitches observed with Google Pixel 7 Pro streaming audio after CIS is established with DUT.During Call Gateway (CG) / Call Terminal (CT) Use Case, the firmware periodically sends NULL PDU, which results in frequent Audio Glitch on both CG and CT sides.Heavy audio glitches observed with CIS SRC Google Pixel 7 ProContinuous audio glitches observed in 1 CIS and 2 CIS for 48_3 and 48_4 config.

Parent topic: Bug fixes and/or feature enhancements

Firmware version: 18.99.3.p17.9 to 18.99.3.p21.154

Component	Description
Wi-Fi	STAUT fail to ping AP backend machine when connected with DFS channel and DUTSTA went in bad state.
Bluetooth	CIS Sink frequently fails to acknowledge CIS Source TX PDU.

Parent topic: Bug fixes and/or feature enhancements

Firmware version: 18.99.3.p21.154 to 18.99.3.p23.16

Component	Description
-	NA

Parent topic: Bug fixes and/or feature enhancements

Firmware version: 18.99.3.p23.16 to 18.99.3.p25.11

Component	Description
Bluetooth	Packet lost observed in CIS case, which causes audio noise.

Parent topic: Bug fixes and/or feature enhancements

Firmware version: 18.99.3.p25.11 to 18.99.3.p26.10

Component	Description
Wi-Fi	During legacy roaming when the “Link Lost” observed the DUTSTA fails to roam
Wi-Fi	During the automated testing of the channel performance, a system hang can occur, with the error message “.sdio_drv_write failed”.

Parent topic: Bug fixes and/or feature enhancements

Firmware version: 18.99.3.p26.10 to 18.99.3.p27.1

Component	Description
Wi-Fi	Enabled mbedtls 3.x

Parent topic: Bug fixes and/or feature enhancements

Firmware version: 18.99.3.p27.10 to 18.99.8.p10

Component	Description
Wi-Fi	Fail to set band to 2.4G in wifi_test_mode application Enabled mbedtls 3.x

Parent topic: Bug fixes and/or feature enhancements

Firmware version: 18.99.8.p10 to 18.99.8.p68

Com po- nent	Description
Wi- Fi	Duplicate Packet Reception in JP Region due to Missing BA/ACK TransmissionFixed STA setting wrong primary channel when external AP has incorrect secondary channel offset field.Fixed WPA3 STA datapath ping stuck until next reassociation; fixed DUT TX throughput dropping to 0 or getting stuck in pre-sleep state under stress.Fixed scan results taking more than 5 seconds to appear in AP+STA mode.Fixed RSSI_INFO (0xa4) command timeout when DUT is left idle for extended periods in AP+STA mode.
Blue tootl	Sequential Removal of CIS Handles as per current Controller implementation i.e CIS Disconnection sequence should be in sequence => CIS - 4,3,2,1While 4-CIS streaming,audio glitches observed on all CIS SINK with Samsung Galaxy budsFixed missing Disconnect Complete event from controller causing host to get stuck after ACL link disconnection.Added mechanism to detect BT slave link FST on odd-numbered offsetFixed firmware crash observed after eSCO connection complete event.Fixed BTC ACL link disconnection due to CCM counter not being incremented during encryptionFixed BLE and BTC power backoff not working when BTU power control is disabled in MFG firmware.

Parent topic: Bug fixes and/or feature enhancements

Firmware version: 18.99.8.p68 to 18.99.8.p141

Component	Description
Wi-Fi	The device connects successfully but fails to roam when signal degrades.Fixed an issue where DHCP renewal was not triggered when roaming between independent APs sharing the same SSID Fixed an issue where ping responses failed on the first uAP BSS start in 2.4 GHz Fixed an issue where the DUT failed to fall back to legacy roaming after 802.11k/11v roaming returned no usable target AP
Bluetooth	Fixed an issue where the BR/EDR Coex shell app incorrectly initialized the Nighthawk (IW610) module and downloaded firmware instead of IW612, causing NB UART firmware download to fail with “HDR SIG not found” (-22) and subsequently WLAN initialization to fail.Fixed a sporadic LE RX degradation issue , where GATT indication delays exceeding 500 ms were observed due to excessive retransmissions in a concurrent BTC ACL + eSCO + BLE scatternet scenario. Fixed a issue where DUT sends HCI Hardware Error (0x22)Fixed a issue where DUT establishes LE connections with two devices using same connection handleFixed a issue where Controller will choose connection interval that won’t collide with eSCO link paremeters.

Parent topic: Bug fixes and/or feature enhancements

Parent topic: *IW611/IW612 release notes*

Known issues

Component	Description
NA	

Parent topic: *IW611/IW612 release notes*

AW611 release notes

Package information

- SDK version: 26.09.00

Parent topic: *IW611/IW612 release notes*

Version information

- Wireless SoC: AW611
- Wi-Fi and Bluetooth/Bluetooth LE firmware version: 18.99.8.p141
  - 18 - Major revision
  - 99 - Feature pack
  - 8 - Release version
  - p141 - Patch number

Parent topic: *AW611 release notes*

Host platform

- i.MX RT1170 EVK, i.MX RT1180 EVK and i.MX RT1180 EVKA Platforms running FreeRTOS
- Host interfaces

  - Wi-Fi over SDIO (SDIO 2.0 support, SDIO clock frequency: 50 MHz)
  - Bluetooth/Bluetooth LE over UART
- Test tools
  - iPerf (version 2.1.9)

Parent topic: *AW611 release notes*

Wi-Fi and Bluetooth certification

The Wi-Fi and Bluetooth certification is obtained with the following combinations.

WFA certifications

- STA | 802.11n
- STA | PMF
- STA | FFD
- STA | SVD
- STA | WPA2
- STA | WPA3
- STA | 802.11ac
- STA | 802.11ax
- STA | QTT

Refer to *6*.

Note: This release supports STAUT only certifications.

Parent topic: Wi-Fi and Bluetooth certification

Bluetooth controller certification

QDID: refer to *4*.

Note: QDID upgrade to Bluetooth Core Specification Version 5.4 is in progress.

Parent topic: Wi-Fi and Bluetooth certification

Parent topic: *IW611/IW612 release notes*

Wi-Fi throughput

Throughput test setup

- Environment: Shield Room - Over the Air
- Access Point: Asus AX88u
- DUT: IW612 Murata (Module: 2EL M.2) with EVK-MIMXRT1060 EVKC platform
- DUT Power Source: External power supply
- Client: Apple MacBook Air
- Channel: 6 | 36
- Wi-Fi application: wifi_wpa_supplicant
- Compiler used to build application: armgcc
- Compiler Version gcc-arm-none-eabi-13.2

- iPerf commands used in test:

TCP TX

```
iperf -c <remote_ip> -t 60
```

TCP RX

```
iperf -s
```

UDP TX

```
iperf -c <remote_ip> -t 60 -u -B <local_ip> -b 120
```

Note: The default rate is 100 Mbps.

UDP RX

```
iperf -s -u -B <local_ip>
```

Note: Read more about the throughput test setup and topology in *2*

The throughput numbers are captured with default configurations using *wifi_wpa_supplicant* sample application.

Parent topic: Wi-Fi throughput

iPerf host configuration and impact on throughput

To get the highest throughput, the throughput values shown in STA throughput and Mobile AP throughput are measured with the maximum values of the default host configuration macros. STA and AP throughput captured with the minimum values of the host configuration macros shows the throughput numbers obtained when using the minimum values of the host configuration macros. The macro values are defined in *lwipopts.h* file.

The table below lists the minimum and maximum values of the host configuration macros.

Values of the host configuration macros

Parameter	Maximum value	Minimum value
TCPIP_MBOX_SIZE	96	32
DEFAULT_RAW_RECVMBOX_SIZE	32	12
DEFAULT_UDP_RECVMBOX_SIZE	64	12
DEFAULT_TCP_RECVMBOX_SIZE	64	12
TCP_MSS	1460	536
TCP_SND_BUF	24 * TCP_MSS	2 * TCP_MSS
MEM_SIZE	319160	41,080
TCP_WND	15 * TCP_MSS	10 * TCP_MSS
MEMP_NUM_PBUF	20	10
MEMP_NUM_TCP_SEG	96	12
MEMP_NUM_TCPIP_MSG_INPKT	80	16
MEMP_NUM_TCPIP_MSG_API	80	8
MEMP_NUM_NETBUF	32	16

STA and AP throughput captured with the minimum values of the host configuration macros

STA mode throughput - HE Mode | 5 GHz Band | 80 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	7	18	111	124
WPA2-AES	7	18	110	124
WPA3-SAE	6	18	110	124

Mobile AP mode throughput - HE Mode | 5 GHz Band | 80 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	2	19	93	127
WPA2-AES	2	19	105	126
WPA3-SAE	2	19	104	132

Parent topic: iPerf host configuration and impact on throughput

Parent topic: Wi-Fi throughput

STA throughput

External AP: Asus AX88u

STA mode throughput - BGN Mode | 2.4 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	46	46	53	62
WPA2-AES	54	55	70	72
WPA3-SAE	53	55	70	72

STA mode throughput - BGN Mode | 2.4 GHz Band | 40 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	77	77	130	131
WPA2-AES	80	79	130	129
WPA3-SAE	80	79	130	129

STA mode throughput - AN Mode | 5 GHz Band | 20 MHz (HT)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	41	48	63	64
WPA2-AES	56	58	71	75
WPA3-SAE	56	58	71	75

STA mode throughput - AN Mode | 5 GHz Band | 40 MHz (HT)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	78	81	129	134
WPA2-AES	78	80	129	133
WPA3-SAE	77	80	129	133

STA mode throughput - VHT Mode | 5 GHz Band | 20 MHz (VHT)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	45	54	71	76
WPA2-AES	56	58	71	75
WPA3-SAE	56	58	71	75

STA mode throughput - VHT Mode | 5 GHz Band | 40 MHz (VHT)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	92	86	153	157
WPA2-AES	89	91	153	157
WPA3-SAE	90	87	153	156

STA mode throughput - VHT Mode | 5 GHz Band | 80 MHz (VHT)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	120	93	233	226
WPA2-AES	114	104	233	226
WPA3-SAE	113	104	233	226

STA mode throughput - HE Mode | 2.4 GHz Band | 20 MHz (HE)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	69	53	95	102
WPA2-AES	63	65	93	95
WPA3-SAE	66	68	101	99

STA mode throughput - HE Mode | 2.4 GHz Band | 40 MHz (HE)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	97	88	214	210
WPA2-AES	99	88	204	209
WPA3-SAE	95	86	216	210

STA mode throughput - HE Mode | 5 GHz Band | 20 MHz (HE)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	59	51	93	127
WPA2-AES	60	51	91	126
WPA3-SAE	60	51	91	127

STA mode throughput - HE Mode | 5 GHz Band | 40 MHz (HE)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	103	90	185	228
WPA2-AES	102	91	180	228
WPA3-SAE	101	88	180	228

STA mode throughput - HE Mode | 5 GHz Band | 80 MHz (HE)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	123	97	233	226
WPA2-AES	123	70	233	225
WPA3-SAE	126	70	233	225

Parent topic: Wi-Fi throughput

Mobile AP throughput

External client: Apple MacBook Air

Mobile AP mode throughput - BGN Mode | 2.4 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	44	45	58	57
WPA2-AES	43	44	60	56
WPA3-SAE	43	44	59	56

Mobile AP mode throughput - BGN Mode | 2.4 GHz Band | 40 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	69	44	122	114
WPA2-AES	68	43	120	113
WPA3-SAE	68	43	120	112

Mobile AP mode throughput - AN Mode | 5 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	46	49	63	61
WPA2-AES	46	49	63	60
WPA3-SAE	46	49	63	60

Mobile AP mode throughput - AN Mode | 5 GHz Band | 40 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	75	78	127	113
WPA2-AES	74	78	127	112
WPA3-SAE	75	78	127	112

Mobile AP mode throughput - VHT Mode | 5 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	50	52	71	71
WPA2-AES	49	51	71	71
WPA3-SAE	50	51	71	71

Mobile AP mode throughput - VHT Mode | 5 GHz Band | 40 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	90	90	152	141
WPA2-AES	89	89	148	139
WPA3-SAE	89	89	144	140

Mobile AP mode throughput - VHT Mode | 5 GHz Band | 80 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	116	110	224	214
WPA2-AES	116	110	219	214
WPA3-SAE	115	110	219	214

Mobile AP mode throughput - HE Mode | 2.4 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	54	48	82	89
WPA2-AES	52	62	81	113
WPA3-SAE	51	62	81	112

Mobile AP mode throughput - HE Mode | 2.4 GHz Band | 40 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	83	76	133	180
WPA2-AES	62	74	132	132
WPA3-SAE	62	72	130	133

Mobile AP mode throughput - HE Mode | 5 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	60	52	81	89
WPA2-AES	60	51	80	90
WPA3-SAE	60	51	80	89

Mobile AP mode throughput - HE Mode | 5 GHz Band | 40 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	98	96	150	209
WPA2-AES	97	95	147	214
WPA3-SAE	97	95	147	214

Mobile AP mode throughput - HE Mode | 5 GHz Band | 80 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	116	107	222	216
WPA2-AES	115	106	217	216
WPA3-SAE	113	106	217	217

Parent topic: Wi-Fi throughput

Parent topic: *IW611/IW612 release notes*

EU conformance tests

- EU Adaptivity test - EN 300 328 v2.1.1 (for 2.4 GHz)
- EU Adaptivity test - EN 301 893 v2.1.1 (for 5 GHz)

Parent topic: *IW611/IW612 release notes*

Bug fixes and/or feature enhancements

Firmware version: 18.99.2.p7.19

Component	Description
-	NA

Parent topic: Bug fixes and/or feature enhancements

Firmware version: 18.99.2.p7.19 to 18.99.2.p49.9

Component	Description
-	NA

Parent topic: Bug fixes and/or feature enhancements

Firmware version: 18.99.2.p49.9 to 18.99.2.p155

Component	Description
Bluetooth	Audio lost occurs due to periodic adv sync lost, during 2 BIS 44.1kHz unencrypted streams with 1M PHY configuration.BIS sync loss may occur in long audio streaming sessions.

Parent topic: Bug fixes and/or feature enhancements

Firmware version: 18.99.2.p155 to 18.99.2.p66.30

Component	Description
Wi-Fi	802.11R Fast BSS roaming works only with hostapd and does not work with standard APs (supporting 11R)
Bluetooth	DUT is not able to sustain a connection with the remote device that does extended advertisement with coded PHY configuration. When 2 CIS streams are active, after the first device disconnects followed by the second device disconnecting, the second peripheral device hangs.Audio Play/Pause does not work in BIS case.

Parent topic: Bug fixes and/or feature enhancements

Firmware version: 18.99.2.p66.30 to 18.99.3.p10.5

Component	Description
Wi-Fi	STAUT not sending Neighbor Advertisement packet after receiving Neighbor Solicitation packet from Ex-AP.Antenna selection time exceeds configured evaluation time
Bluetooth	When DUT works as CIS source and CIS Offset is 612us, high packet drops observed which affects the audio streaming.For BIS Source Use Cases, Periodic Interval and ISO Interval should be multiple of each other value.In 1-CIS and 2-CIS, Continuous Audio Glitches are observed with 96 kbps bit rate.

Parent topic: Bug fixes and/or feature enhancements

Firmware version: 18.99.3.p10.5 to 18.99.3.p17.9

Component	Description
Wi-Fi	After performing independent reset (out-of-band mode), the STAUT fails to connect to the external AP via wlan-connect command, observed command timeout 0x107 error.
Bluetooth	Audio glitches observed with Google Pixel 7 Pro streaming audio after CIS is established with DUT.During Call Gateway (CG) / Call Terminal (CT) Use Case, the firmware periodically sends NULL PDU, which results in frequent Audio Glitch on both CG and CT sides.Heavy audio glitches observed with CIS SRC Google Pixel 7 ProContinuous audio glitches observed in 1 CIS and 2 CIS for 48_3 and 48_4 config.

Parent topic: Bug fixes and/or feature enhancements

Firmware version: 18.99.3.p17.9 to 18.99.3.p21.154

Component	Description
Wi-Fi	STAUT fail to ping AP backend machine when connected with DFS channel and DUTSTA went in bad state.
Bluetooth	CIS Sink frequently fails to acknowledge CIS Source TX PDU.

Parent topic: Bug fixes and/or feature enhancements

Firmware version: 18.99.3.p21.154 to 18.99.3.p23.16

Component	Description
-	NA

Parent topic: Bug fixes and/or feature enhancements

Firmware version: 18.99.3.p23.16 to 18.99.3.p25.11

Component	Description
Bluetooth	Packet lost observed in CIS case, which causes audio noise.

Parent topic: Bug fixes and/or feature enhancements

Firmware version: 18.99.3.p25.11 to 18.99.3.p26.10

Component	Description
Wi-Fi	During legacy roaming when the “Link Lost” observed the DUTSTA fails to roam
Wi-Fi	During the automated testing of the channel performance, a system hang can occur, with the error message “.sdio_drv_write failed”.

Parent topic: Bug fixes and/or feature enhancements

Firmware version: 18.99.3.p26.10 to 18.99.3.p27.1

Component	Description
Wi-Fi	Enabled mbedtls 3.x

Parent topic: Bug fixes and/or feature enhancements

Firmware version: 18.99.3.p27.10 to 18.99.8.p10

Component	Description
Wi-Fi	Fail to set band to 2.4G in wifi_test_mode application Enabled mbedtls 3.x

Parent topic: Bug fixes and/or feature enhancements

Firmware version: 18.99.8.p10 to 18.99.8.p68

Component	Description
Wi-Fi	Duplicate Packet Reception in JP Region due to Missing BA/ACK TransmissionFixed STA setting wrong primary channel when external AP has incorrect secondary channel offset field.Fixed WPA3 STA datapath ping stuck until next reassociation; fixed DUT TX throughput dropping to 0 or getting stuck in pre-sleep state under stress.Fixed scan results taking more than 5 seconds to appear in AP+STA mode.Fixed RSSI_INFO (0xa4) command timeout when DUT is left idle for extended periods in AP+STA mode.
Bluetootl	Sequential Removal of CIS Handles as per current Controller implementation i.e CIS Disconnection sequence should be in sequence => CIS - 4,3,2,1While 4-CIS streaming,audio glitches observed on all CIS SINK with Samsung Galaxy budsFixed missing Disconnect Complete event from controller causing host to get stuck after ACL link disconnection.Added mechanism to detect BT slave link FST on odd-numbered offsetFixed firmware crash observed after eSCO connection complete event.Fixed BTC ACL link disconnection due to CCM counter not being incremented during encryptionFixed BLE and BTC power backoff not working when BTU power control is disabled in MFG firmware.

Parent topic: Bug fixes and/or feature enhancements

Firmware version: 18.99.8.p68 to 18.99.8.p141

Component	Description
Wi-Fi	The device connects successfully but fails to roam when signal degrades.Fixed an issue where DHCP renewal was not triggered when roaming between independent APs sharing the same SSID Fixed an issue where ping responses failed on the first uAP BSS start in 2.4 GHz Fixed an issue where the DUT failed to fall back to legacy roaming after 802.11k/11v roaming returned no usable target AP
Bluetootl	Fixed an issue where the BR/EDR Coex shell app incorrectly initialized the Nighthawk (IW610) module and downloaded firmware instead of IW612, causing NB UART firmware download to fail with “HDR SIG not found” (-22) and subsequently WLAN initialization to fail.Fixed a sporadic LE RX degradation issue , where GATT indication delays exceeding 500 ms were observed due to excessive retransmissions in a concurrent BTC ACL + eSCO + BLE scatternet scenario. Fixed a issue where DUT sends HCI Hardware Error (0x22)Fixed a issue where DUT establishes LE connections with two devices using same connection handleFixed a issue where Controller will choose connection interval that won’t collide with eSCO link paremeters.

Parent topic: Bug fixes and/or feature enhancements

Parent topic: *IW611/IW612 release notes*

Known issues

Component	Description
NA	

Parent topic: *IW611/IW612 release notes*

RW610/RW612 release notes

Package information

- SDK version: 26.09.00

Parent topic: *RW610/RW612 release notes*

Version information

- Wi-Fi firmware version: 18.99.8.p141
  - rw61x_sb_wifi_a2.bin for A2
  - 18 - Major revision
  - 99 - Feature pack
  - 8 - Release version
  - p141 - Patch number
- Bluetooth LE firmware version: 18.99.8.p141
  - rw61x_sb_ble_a2.bin for A2
  - 18 - Major revision
  - 99 - Feature pack
  - 8 - Release version
  - p141 - Patch number
- 802.15.4 and Bluetooth LE (up to core 4.1) firmware version: 18.99.8.p141
  - rw61x_sb_ble_15d4_combo_a2.bin for A2
  - 18 - Major revision
  - 99 - Feature pack
  - 8 - Release version
  - p141 - Patch number

Parent topic: *RW610/RW612 release notes*

Host platform

- RW610/RW612 platform running FreeRTOS
- Test tools
  - iPerf (version 2.1.9)

Parent topic: *RW610/RW612 release notes*

Wireless certification

The Wi-Fi and Bluetooth certification is obtained with the following combinations.

WFA certifications

- STA | 802.11n
- STA | PMF
- STA | FFD
- STA | SVD

- STA | WPA2
- STA | WPA3
- STA | 802.11ac
- STA | 802.11ax
- STA | QTT

Refer to *1*.

Note: This release supports STAUT only certifications.

Parent topic: Wireless certification

Bluetooth LE controller certification

QDID: Refer to *4*.

Parent topic: Wireless certification

Thread

Thread group: refer to *7*.

Product Name: NXP RW612 Wireless MCU with Integrated Tri-Radio

Thread version: V1.3.0

CID #: 13A109

Parent topic: Wireless certification

Matter

RW612 certification: refer to *8*.

Certificate ID: CSA23C36MAT41746-24

Device type: Root Node, Thermostat

Transport: Matter over Wi-Fi

RW610 certification: refer to *9*.

Certificate ID: CSA23C43MAT41753-50

Device type: Root Node, Thermostat

Transport: Matter over Wi-Fi and Matter over Thread

Parent topic: Wireless certification

Parent topic: *RW610/RW612 release notes*

Wi-Fi throughput

Throughput test setup

- Environment: Shield Room - Over the Air
- Access Point: Asus AX88u
- DUT: RW610/RW612
- External Client: Intel AX210
- Channel: 6 | 36
- Wi-Fi application: wifi_cli

- Compiler used to build application: armgcc
- Compiler version gcc-arm-none-eabi-13.2
- iPerf commands used in test:

TCP TX

```
iperf -c <remote_ip> -t 60
```

TCP RX

```
iperf -s
```

UDP TX

```
iperf -c <remote_ip> -t 60 -u -B <local_ip> -b 120
```

Note: The default rate is 100 Mbps.

UDP RX

```
iperf -s -u -B <local_ip>
```

Note: Read more about the throughput test setup and topology in *3*.

Parent topic: Wi-Fi throughput

STA throughput

External AP: Asus AX88u

STA mode throughput - BGN Mode | 2.4 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	35	39	62	63
WPA2-AES	34	39	60	63
WPA3-SAE	34	39	60	59

STA mode throughput - AN Mode | 5 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	35	40	64	65
WPA2-AES	35	40	62	64
WPA3-SAE	36	40	62	64

STA mode throughput - VHT Mode | 5 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	38	44	77	77
WPA2-AES	38	43	74	76
WPA3-SAE	38	43	74	76

STA mode throughput - HE Mode | 2.4 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	42	45	97	100
WPA2-AES	40	44	96	97
WPA3-SAE	40	44	96	97

STA mode throughput - HE Mode | 5 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	42	44	100	95
WPA2-AES	41	43	99	92
WPA3-SAE	41	44	99	98

Parent topic: Wi-Fi throughput

Mobile AP throughput

External client: Apple MacBook Air

Mobile AP throughput - BGN Mode | 2.4 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	36	39	61	61
WPA2-AES	36	38	59	59
WPA3-SAE	36	38	58	59

Mobile AP throughput - AN Mode | 5 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	38	41	64	64
WPA2-AES	37	40	62	62
WPA3-SAE	37	40	62	62

Mobile AP throughput - VHT Mode | 5 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	42	43	75	75
WPA2-AES	41	44	75	74
WPA3-SAE	41	44	75	74

Mobile AP throughput - HE Mode | 2.4 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	43	47	92	94
WPA2-AES	42	46	93	104
WPA3-SAE	42	46	93	104

Mobile AP throughput - HE Mode | 5 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	45	49	97	98
WPA2-AES	44	48	98	105
WPA3-SAE	44	48	98	105

Parent topic: Wi-Fi throughput

Parent topic: *RW610/RW612 release notes*

Bug fixes and/or feature enhancements

Firmware version: 18.99.6.p34 to 18.99.6.p40

Com-ponent	Description
Zigbee	Zigbee Coordinator and Router are disconnected during BLE connection pairing and bonding with a mobile app for the first time.

Parent topic: Bug fixes and/or feature enhancements

Firmware version: 18.99.6.p40 to 18.99.6.p46

Compo-nent	Description
Wi-Fi	Fails to establish a persistent connection when the device attempts to reinvoke the second stored Persistent Group
Blue-tooth	NCP cannot work after flash uart bins for both host and device side

Parent topic: Bug fixes and/or feature enhancements

Firmware version: 18.99.6.p46 to 18.99.6.p47

Component	Description
Wi-Fi	Enabled mbedtls 3.x

Parent topic: Bug fixes and/or feature enhancements

Firmware version: 18.99.6.p46 to 18.99.6.p47

Component	Description
Wi-Fi	Enabled mbedtls 3.xHost sleep with WoWLAN fails due to command response timeout (cmd 0x10d) during MCU sleep/wake transitions on WiFi NCP over UART, leading to system assert during sleep exitPing failures observed during concurrent network operations (fast ping from AP backend and DUT), with ICMP unreachable errors indicating routing/forwarding issues and performance degradation under specific network load conditions

Parent topic: Bug fixes and/or feature enhancements

Firmware version: 18.99.6.p50 to 18.99.8.p10

Component	Description
Wi-Fi	Enabled mbedtls 3.x Fail to set band to 2.4G in wifi_test_mode application

Parent topic: Bug fixes and/or feature enhancements

Firmware version: 18.99.8.p10 to 18.99.8.p68

Component	Description
Wi-Fi	Fixed WPA3 SAE security failure observed in WiFi CLI app during initiating connection with AP. Fixed TX power increase of ~3 dBm observed after connecting to a DFS channel and then reconnecting to a non-DFS channel. TX_POWER_LUT_EXTENSION_SUPPORT did not account for the one-channel DPD case, causing TX power to exceed the configured limit on subsequent non-DFS channel connections.
BLu tootl	Implementation of PaWR (Periodic Advertisement with response) feature
Co-ex-is-tanc	Fixed driver assert issue when wake up wifi cpu1Wi-Fi throughput drops to 0 when Wi-Fi and OT traffic run concurrently on non-overlapping channels at an RSSI of -80 dBm.Fixed high PING latency (up to ~1876 ms average ~397 ms) and packet loss observed with external FEM in Wi-Fi + Thread coexistence mode. RF control lines were switching excessively between Wi-Fi and Narrowband, disrupting Wi-Fi PSM operation and degrading throughput/latency.

Parent topic: Bug fixes and/or feature enhancements

Firmware version: 18.99.8.p68 to 18.99.8.p141

Component	Description
Wi-Fi	Fixed Wi-Fi RF Test Mode continuous-wave tone transmission stopping when run for a long duration (30 seconds to 15 minutes).Fixed an issue where DHCP renewal was not triggered when roaming between independent APs sharing the same SSID.Fixed an issue where EAP-TLS authentication failed with TLSv1.2 in the wpa_supplicant example.Fixed an issue where wlan-scan-opt parameters incorrectly applied to subsequent normal scans.Fixed an issue where the ip_addr_t type field was left uninitialized in net_configure_dns() when IPv6 was enabled.Fixed an issue where Wi-Fi ceased to function after receiving an 802.11k RRM beacon request for a 5 GHz channel when CONFIG_5GHz_SUPPORT=0.Fixed an issue where the DUT failed to fall back to legacy roaming after 802.11k/11v roaming returned no usable target AP.
Bluetooth LE	Fixed an issue where BLE TX power set above 10 dBm via vendor command 0xFC87 was reset to the OTP/calibration default (~8.3 dBm) upon BLE connection establishment.Fixed an issue where all Bluetooth commands became unresponsive while Wi-Fi continued to function normally after approximately 3–4 hours of Wi-Fi/BT coexistence testing.Fixed an issue where both Wi-Fi and the DUT hung up after approximately 50 iterations of a concurrent Wi-Fi connect/ping/disconnect and BLE init/advertise/disable loop stress test.Fixed an issue where BLE advertising and scanning failed after repeated bt init / bt disable stress cycles (~600 out of 1000 iterations).

Parent topic: Bug fixes and/or feature enhancements

Parent topic: *RW610/RW612 release notes*

Known issues

Component	Description
Wi-Fi	—
Bluetooth LE	—
Zigbee	-
Coex	-

Parent topic: *RW610/RW612 release notes*

IW610 release notes

Package information

- SDK version: 26.09.00

Parent topic: *IW610 release notes*

Version information

- Wireless SoC: IW610
- Wi-Fi and Bluetooth/Bluetooth LE firmware version: 18.99.8.p141
  - 18 - Major revision
  - 99 - Feature pack
  - 8 - Release version
  - p141 - Patch number

Parent topic: *IW610 release notes*

Host platform

- i.MX RT1170 EVKB, i.MX RT1160 EVK, i.MX RT1060 EVKC, i.MX RT595 EVK, i.MX RT685-AUD EVK, and i.MX RT700 EVK Platforms running FreeRTOS
- Host interfaces
  - Wi-Fi over SDIO (SDIO 2.0 support, SDIO clock frequency: 50 MHz)
  - Bluetooth/Bluetooth LE over UART
- Test tools
  - iPerf (version 2.1.9)

Parent topic: *IW610 release notes*

Wi-Fi and Bluetooth certification

The Wi-Fi and Bluetooth certification is obtained with the following combinations.

Bluetooth controller certification

QDID: Refer to *4*.

Note: QDID upgrade to Bluetooth Core Specification Version 5.4 is in progress.

Parent topic: Wi-Fi and Bluetooth certification

Parent topic: *IW610 release notes*

Wi-Fi throughput

Throughput test setup

- Environment: Shield Room - Over the Air
- Access Point: Asus AX88u
- DUT: IW610
- External Client: Intel AX210
- Channel: 6 | 36
- Wi-Fi application: wifi_cli
- Compiler used to build application: armgcc
- Compiler version gcc-arm-none-eabi-13.2
- iPerf commands used in test:

TCP TX

```
iperf -c <remote_ip> -t 60
```

TCP RX

```
iperf -s
```

UDP TX

```
iperf -c <remote_ip> -t 60 -u -B <local_ip> -b 120
```

Note: The default rate is 100 Mbps.

UDP RX

```
iperf -s -u -B <local_ip>
```

Note: Read more about the throughput test setup and topology in *3*.

Parent topic: Wi-Fi throughput

STA throughput

External AP: Asus AX88u

STA mode throughput - BGN Mode | 2.4 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	42	47	60	61
WPA2-AES	41	46	59	59
WPA3-SAE	42	46	58	60

STA mode throughput - AN Mode | 5 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	46	50	63	64
WPA2-AES	45	49	61	63
WPA3-SAE	45	50	61	62

STA mode throughput - VHT Mode | 2.4 GHz Band | 20 MHz (HT)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	41	40	72	72
WPA2-AES	40	40	72	72
WPA3-SAE	40	40	72	71

STA mode throughput - VHT Mode | 5 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	51	58	75	76
WPA2-AES	50	57	73	75
WPA3-SAE	50	57	73	75

STA mode throughput - HE Mode | 2.4 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	56	60	93	98
WPA2-AES	53	57	91	91
WPA3-SAE	54	54	94	91

STA mode throughput - HE Mode | 5 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	50	64	90	102
WPA2-AES	48	61	91	93
WPA3-SAE	48	62	90	93

Parent topic: Wi-Fi throughput

Mobile AP throughput

External client: Apple MacBook Air

Mobile AP mode throughput - BGN Mode | 2.4 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	47	44	63	61
WPA2-AES	46	43	61	60
WPA3-SAE	46	43	61	60

Mobile AP mode throughput - AN Mode | 5 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	49	46	63	63
WPA2-AES	48	45	61	61
WPA3-SAE	48	45	61	61

Mobile AP mode throughput - VHT Mode | 5 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	55	51	74	74
WPA2-AES	54	50	74	72
WPA3-SAE	54	50	74	73

Mobile AP mode throughput - HE Mode | 2.4 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	57	58	95	89
WPA2-AES	55	56	96	84
WPA3-SAE	55	56	96	84

Mobile AP mode throughput - HE Mode | 5 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	61	58	97	96
WPA2-AES	60	57	97	87
WPA3-SAE	60	57	97	88

Parent topic: Wi-Fi throughput

Parent topic: *IW610 release notes*

Bug fixes and/or feature enhancements

Firmware version: 18.99.5.p66 to 18.99.5.p76

Component	Description
Wi-Fi	The P2P client connection fails when an attempt is made to connect after the P2P Group Owner (P2P-GO) has been stopped.

Parent topic: Bug fixes and/or feature enhancements

Firmware version: 18.99.5.p76 to 18.99.5.p79

Component	Description
Wi-Fi	Enabled mbedtls 3.x

Parent topic: Bug fixes and/or feature enhancements

Firmware version: 18.99.5.p79 to 18.99.5.p86

Component	Description
Wi-Fi	Enabled mbedtls 3.xWi-Fi P2P client-GO connection failure due to channel coordination issues between STA and P2P operations; core problem lies in P2P behavior within general Wi-Fi functionality, not RTOS platform-specific features

Parent topic: Bug fixes and/or feature enhancements

Firmware version: 18.99.5.p86 to 18.99.8.p10

Component	Description
Wi-Fi	Enabled mbedtls 3.x Fail to set band to 2.4G in wifi_test_mode application

Parent topic: Bug fixes and/or feature enhancements

Firmware version: 18.99.8.p10 to 18.99.8.p68

Component	Description
Wi-Fi	Fixed Wi-Fi IOCTL CMD 0x256 response error.

Parent topic: Bug fixes and/or feature enhancements

Firmware version: 18.99.8.p68 to 18.99.8.p141

Component	Description
Wi-Fi	Fixed Wi-Fi RF Test Mode continuous-wave tone transmission stopping when run for a long duration (30 seconds to 15 minutes).Fixed an issue where DHCP renewal was not triggered when roaming between independent APs sharing the same SSID.Fixed an issue where Wi-Fi initialization failed after a JLink reset, requiring a power cycle to recover.Fixed an issue where the MCU RTOS hung permanently after executing wlan-reset.Fixed an issue where the DUT failed to fall back to legacy roaming after 802.11k/11v roaming returned no usable target AP.
Bluetooth LE	Fixed an issue where the DUT hung during BLE LE Scan when Wi-Fi was associated with a WPA2 security SSID concurrently.Fixed an issue where BLE scan failed to stop cleanly, causing an assertion/hang after issuing bt scan off.
Coex	Fixed an issue where adding a WPA3-SAE or Enterprise Wi-Fi profile caused a crash/hang, preventing Wi-Fi association during concurrent BLE L2CAP coexistence testing.Fixed an issue where BLE re-initialization failed during a BLE init/disable loop test while Wi-Fi was associated and running PING traffic.

Parent topic: Bug fixes and/or feature enhancements

Parent topic: *IW610 release notes*

Known issues

Component	Description
NA	

Parent topic: *IW610 release notes*

Abbreviations

Abbreviation	Definition
A2DP	Advanced audio distribution profile
AMPDU	Aggregated MAC protocol data unit
AMSDU	Aggregated MAC service data unit
AP	Access point
BW	Bandwidth
CCMP	Counter mode CBC-MAC protocol
CSI	Channel state information
CTS	Clear To Send
DL	Down link
EDCA	Enhanced distributed channel access
ER	Extended range

continues on next page

Table 4 – continued from previous page

Abbreviation	Definition
ERP	Extended rate physical
GATT	Generic attribute profile
HFP	Hands free profile
HID	Human interface device
HT	High throughput
LDPC	Low density parity check
MCS	Modulation and coding scheme
MLME	Mac layer management entity
OMI	Operating mode indication
PMF	Protected management frames
RTS	Request to send
SAE	Simultaneous authentication of equals
STA	Station
TWT	Target wake time
UL	Up link
VHT	Very high throughput
WEP	Wired equivalent private
WFD	Wi-Fi direct
WMM	Wireless multi-media
WPA	Wi-Fi protected access
WPS	Wi-Fi protected setup
WSC	Wi-Fi Simple Configuration

References

1. Application note - AN13681 – Wi-Fi Alliance (WFA) Derivative Certification Process (available in the SDK package)
2. User manual – UM11442 - NXP Wi-Fi and Bluetooth Demo Applications User Guide for i.MX RT Platforms (available in the SDK package)
3. User manual – UM11799 - NXP Wi-Fi and Bluetooth Demo Applications User Guide for RW61x (available in the SDK package)
4. Certification – Bluetooth controller - QDID (*link*)
5. User manual - UM12133 - NXP NCP Application Guide for RW612 with MCU Host
6. Technical note - TN00066 – Wi-Fi Alliance (WFA) Derivative Certification Process (available in the SDK package)
7. Web page – Thread certified products (link)
8. Web page – Connectivity standard alliance (csa) – NXP RW612 Tri-Radio Wireless MCU Development Platform (link)
9. Web page – Connectivity standard alliance (csa) – NXP RW610 Wireless MCU Development Platform (link)
10. Application note - AN14634 – Kconfig Memory Optimizer (link)

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.

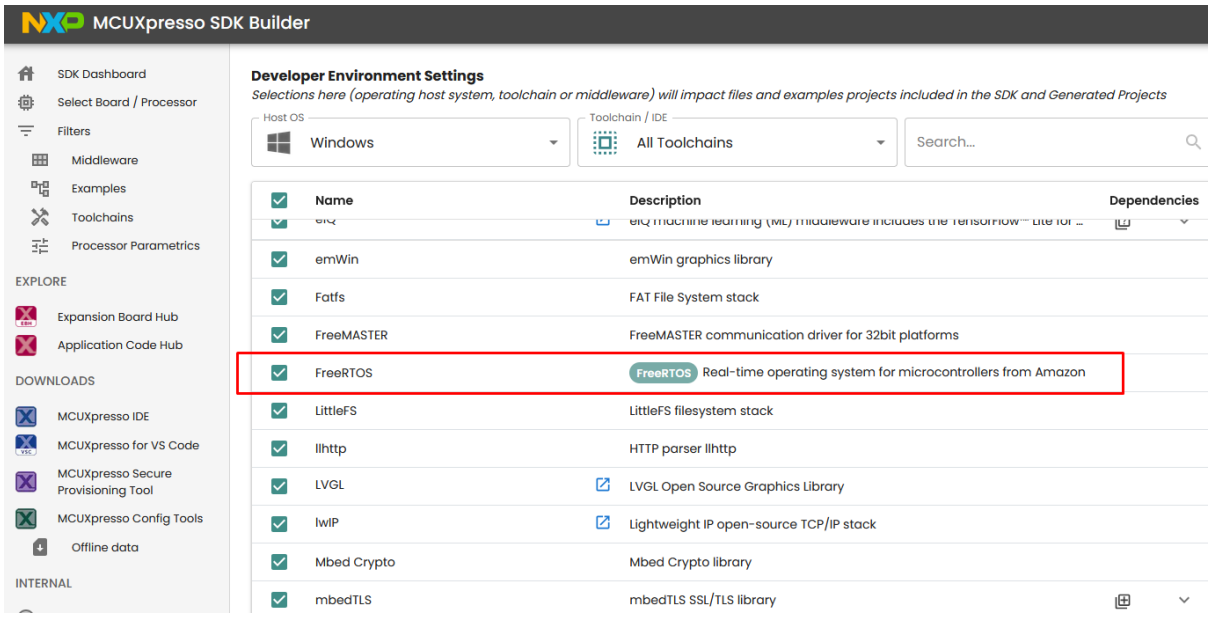
List of examples

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

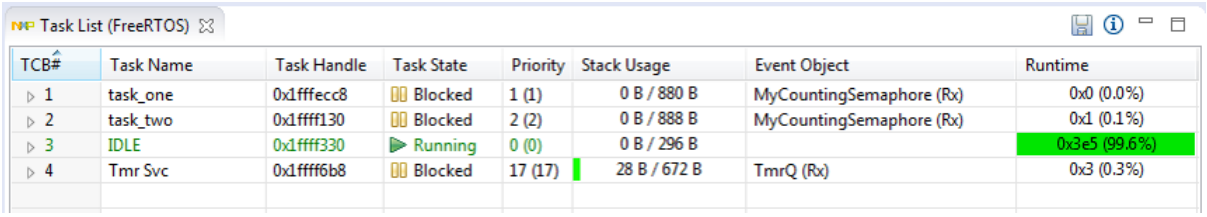
For information how to use the cmake and Kconfig based build and configuration system and how to build freertos_examples visit: MCUXpresso SDK documentation for Build And Configuration MCUXpresso SDK Getting Start Guide

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.



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 configFREERTOS_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-evkmimxrt1024.pdf — from *boards/RT/evkmimxrt1024/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

D

drivers_RT_RT1020_MIMXRT1024._adc_channel_config (*C struct*), 181
drivers_RT_RT1020_MIMXRT1024._adc_channel_config.channelNumber (*C var*), 181
drivers_RT_RT1020_MIMXRT1024._adc_channel_config.enableInterruptOnConversionCompleted (*C var*), 181
drivers_RT_RT1020_MIMXRT1024._adc_clock_drvier (*C enum*), 177
drivers_RT_RT1020_MIMXRT1024._adc_clock_drvier.kADC_ClockDriver1 (*C enumerator*), 177
drivers_RT_RT1020_MIMXRT1024._adc_clock_drvier.kADC_ClockDriver2 (*C enumerator*), 178
drivers_RT_RT1020_MIMXRT1024._adc_clock_drvier.kADC_ClockDriver4 (*C enumerator*), 178
drivers_RT_RT1020_MIMXRT1024._adc_clock_drvier.kADC_ClockDriver8 (*C enumerator*), 178
drivers_RT_RT1020_MIMXRT1024._adc_clock_source (*C enum*), 177
drivers_RT_RT1020_MIMXRT1024._adc_clock_source.kADC_ClockSourceAD (*C enumerator*), 177
drivers_RT_RT1020_MIMXRT1024._adc_clock_source.kADC_ClockSourceIPG (*C enumerator*), 177
drivers_RT_RT1020_MIMXRT1024._adc_clock_source.kADC_ClockSourceIPGDiv2 (*C enumerator*), 177
drivers_RT_RT1020_MIMXRT1024._adc_config (*C struct*), 180
drivers_RT_RT1020_MIMXRT1024._adc_config.clockDriver (*C var*), 180
drivers_RT_RT1020_MIMXRT1024._adc_config.clockSource (*C var*), 180
drivers_RT_RT1020_MIMXRT1024._adc_config.enableAsynchronousClockOutput (*C var*), 180
drivers_RT_RT1020_MIMXRT1024._adc_config.enableContinuousConversion (*C var*), 180
drivers_RT_RT1020_MIMXRT1024._adc_config.enableHighSpeed (*C var*), 180
drivers_RT_RT1020_MIMXRT1024._adc_config.enableLongSample (*C var*), 180
drivers_RT_RT1020_MIMXRT1024._adc_config.enableLowPower (*C var*), 180
drivers_RT_RT1020_MIMXRT1024._adc_config.enableOverWrite (*C var*), 180
drivers_RT_RT1020_MIMXRT1024._adc_config.referenceVoltageSource (*C var*), 180
drivers_RT_RT1020_MIMXRT1024._adc_config.resolution (*C var*), 180
drivers_RT_RT1020_MIMXRT1024._adc_config.samplePeriodMode (*C var*), 180
drivers_RT_RT1020_MIMXRT1024._adc_etc_config (*C struct*), 186
drivers_RT_RT1020_MIMXRT1024._adc_etc_dma_mode_selection (*C enum*), 185
drivers_RT_RT1020_MIMXRT1024._adc_etc_dma_mode_selection.kADC_ETC_TrigDMAWithLatchedSignal (*C enumerator*), 185
drivers_RT_RT1020_MIMXRT1024._adc_etc_dma_mode_selection.kADC_ETC_TrigDMAWithPulsedSignal (*C enumerator*), 185
drivers_RT_RT1020_MIMXRT1024._adc_etc_external_trigger_source (*C enum*), 185
drivers_RT_RT1020_MIMXRT1024._adc_etc_external_trigger_source.kADC_ETC_Trg0TriggerSource (*C enumerator*), 185
drivers_RT_RT1020_MIMXRT1024._adc_etc_external_trigger_source.kADC_ETC_Trg1TriggerSource (*C enumerator*), 185
drivers_RT_RT1020_MIMXRT1024._adc_etc_external_trigger_source.kADC_ETC_Trg2TriggerSource (*C enumerator*), 185
drivers_RT_RT1020_MIMXRT1024._adc_etc_external_trigger_source.kADC_ETC_Trg3TriggerSource (*C enumerator*), 185
drivers_RT_RT1020_MIMXRT1024._adc_etc_external_trigger_source.kADC_ETC_Trg4TriggerSource (*C enumerator*), 185
drivers_RT_RT1020_MIMXRT1024._adc_etc_external_trigger_source.kADC_ETC_Trg5TriggerSource

(*C enumerator*), 185
drivers_RT_RT1020_MIMXRT1024._adc_etc_external_trigger_source.kADC_ETC_Trg6TriggerSource (*C enumerator*), 185
drivers_RT_RT1020_MIMXRT1024._adc_etc_external_trigger_source.kADC_ETC_Trg7TriggerSource (*C enumerator*), 185
drivers_RT_RT1020_MIMXRT1024._adc_etc_external_trigger_source.kADC_ETC_TSC0TriggerSource (*C enumerator*), 185
drivers_RT_RT1020_MIMXRT1024._adc_etc_external_trigger_source.kADC_ETC_TSC1TriggerSource (*C enumerator*), 185
drivers_RT_RT1020_MIMXRT1024._adc_etc_interrupt_enable (*C enum*), 185
drivers_RT_RT1020_MIMXRT1024._adc_etc_interrupt_enable.kADC_ETC_Done0InterruptEnable (*C enumerator*), 185
drivers_RT_RT1020_MIMXRT1024._adc_etc_interrupt_enable.kADC_ETC_Done1InterruptEnable (*C enumerator*), 185
drivers_RT_RT1020_MIMXRT1024._adc_etc_interrupt_enable.kADC_ETC_Done2InterruptEnable (*C enumerator*), 185
drivers_RT_RT1020_MIMXRT1024._adc_etc_interrupt_enable.kADC_ETC_Done3InterruptEnable (*C enumerator*), 185
drivers_RT_RT1020_MIMXRT1024._adc_etc_status_flag_mask (*C enum*), 184
drivers_RT_RT1020_MIMXRT1024._adc_etc_status_flag_mask.kADC_ETC_Done0StatusFlagMask (*C enumerator*), 184
drivers_RT_RT1020_MIMXRT1024._adc_etc_status_flag_mask.kADC_ETC_Done1StatusFlagMask (*C enumerator*), 184
drivers_RT_RT1020_MIMXRT1024._adc_etc_status_flag_mask.kADC_ETC_Done2StatusFlagMask (*C enumerator*), 184
drivers_RT_RT1020_MIMXRT1024._adc_etc_status_flag_mask.kADC_ETC_Done3StatusFlagMask (*C enumerator*), 184
drivers_RT_RT1020_MIMXRT1024._adc_etc_status_flag_mask.kADC_ETC_ErrorStatusFlagMask (*C enumerator*), 185
drivers_RT_RT1020_MIMXRT1024._adc_etc_trigger_chain_config (*C struct*), 186
drivers_RT_RT1020_MIMXRT1024._adc_etc_trigger_config (*C struct*), 186
drivers_RT_RT1020_MIMXRT1024._adc_hardware_average_mode (*C enum*), 178
drivers_RT_RT1020_MIMXRT1024._adc_hardware_average_mode.kADC_HardwareAverageCount4 (*C enumerator*), 178
drivers_RT_RT1020_MIMXRT1024._adc_hardware_average_mode.kADC_HardwareAverageCount8 (*C enumerator*), 179
drivers_RT_RT1020_MIMXRT1024._adc_hardware_average_mode.kADC_HardwareAverageCount16 (*C enumerator*), 179
drivers_RT_RT1020_MIMXRT1024._adc_hardware_average_mode.kADC_HardwareAverageCount32 (*C enumerator*), 179
drivers_RT_RT1020_MIMXRT1024._adc_hardware_average_mode.kADC_HardwareAverageDiasable (*C enumerator*), 179
drivers_RT_RT1020_MIMXRT1024._adc_hardware_compare_config (*C struct*), 181
drivers_RT_RT1020_MIMXRT1024._adc_hardware_compare_config.hardwareCompareMode (*C var*), 181
drivers_RT_RT1020_MIMXRT1024._adc_hardware_compare_config.value1 (*C var*), 181
drivers_RT_RT1020_MIMXRT1024._adc_hardware_compare_config.value2 (*C var*), 181
drivers_RT_RT1020_MIMXRT1024._adc_hardware_compare_mode (*C enum*), 178
drivers_RT_RT1020_MIMXRT1024._adc_hardware_compare_mode.kADC_HardwareCompareMode0 (*C enumerator*), 178
drivers_RT_RT1020_MIMXRT1024._adc_hardware_compare_mode.kADC_HardwareCompareMode1 (*C enumerator*), 178
drivers_RT_RT1020_MIMXRT1024._adc_hardware_compare_mode.kADC_HardwareCompareMode2 (*C enumerator*), 178
drivers_RT_RT1020_MIMXRT1024._adc_hardware_compare_mode.kADC_HardwareCompareMode3 (*C enumerator*), 178
drivers_RT_RT1020_MIMXRT1024._adc_offest_config (*C struct*), 181
drivers_RT_RT1020_MIMXRT1024._adc_offest_config.enableSigned (*C var*), 181

drivers_RT_RT1020_MIMXRT1024._adc_offest_config.offsetValue (*C var*), 181
drivers_RT_RT1020_MIMXRT1024._adc_reference_voltage_source (*C enum*), 176
drivers_RT_RT1020_MIMXRT1024._adc_reference_voltage_source.kADC_ReferenceVoltageSourceAlt0 (*C enumerator*), 176
drivers_RT_RT1020_MIMXRT1024._adc_resolution (*C enum*), 178
drivers_RT_RT1020_MIMXRT1024._adc_resolution.kADC_Resolution8Bit (*C enumerator*), 178
drivers_RT_RT1020_MIMXRT1024._adc_resolution.kADC_Resolution10Bit (*C enumerator*), 178
drivers_RT_RT1020_MIMXRT1024._adc_resolution.kADC_Resolution12Bit (*C enumerator*), 178
drivers_RT_RT1020_MIMXRT1024._adc_sample_period_mode (*C enum*), 176
drivers_RT_RT1020_MIMXRT1024._adc_sample_period_mode.kADC_SamplePeriod2or12Clocks (*C enumerator*), 176
drivers_RT_RT1020_MIMXRT1024._adc_sample_period_mode.kADC_SamplePeriod4or16Clocks (*C enumerator*), 176
drivers_RT_RT1020_MIMXRT1024._adc_sample_period_mode.kADC_SamplePeriod6or20Clocks (*C enumerator*), 176
drivers_RT_RT1020_MIMXRT1024._adc_sample_period_mode.kADC_SamplePeriod8or24Clocks (*C enumerator*), 177
drivers_RT_RT1020_MIMXRT1024._adc_sample_period_mode.kADC_SamplePeriodLong12Clcoks (*C enumerator*), 177
drivers_RT_RT1020_MIMXRT1024._adc_sample_period_mode.kADC_SamplePeriodLong16Clcoks (*C enumerator*), 177
drivers_RT_RT1020_MIMXRT1024._adc_sample_period_mode.kADC_SamplePeriodLong20Clcoks (*C enumerator*), 177
drivers_RT_RT1020_MIMXRT1024._adc_sample_period_mode.kADC_SamplePeriodLong24Clcoks (*C enumerator*), 177
drivers_RT_RT1020_MIMXRT1024._adc_sample_period_mode.kADC_SamplePeriodShort2Clocks (*C enumerator*), 177
drivers_RT_RT1020_MIMXRT1024._adc_sample_period_mode.kADC_SamplePeriodShort4Clocks (*C enumerator*), 177
drivers_RT_RT1020_MIMXRT1024._adc_sample_period_mode.kADC_SamplePeriodShort6Clocks (*C enumerator*), 177
drivers_RT_RT1020_MIMXRT1024._adc_sample_period_mode.kADC_SamplePeriodShort8Clocks (*C enumerator*), 177
drivers_RT_RT1020_MIMXRT1024._adc_status_flags (*C enum*), 176
drivers_RT_RT1020_MIMXRT1024._adc_status_flags.kADC_AsynchronousWakeupInterruptFlag (*C enumerator*), 176
drivers_RT_RT1020_MIMXRT1024._adc_status_flags.kADC_CalibrationFailedFlag (*C enumerator*), 176
drivers_RT_RT1020_MIMXRT1024._adc_status_flags.kADC_ConversionActiveFlag (*C enumerator*), 176
drivers_RT_RT1020_MIMXRT1024._aipstz_master (*C enum*), 187
drivers_RT_RT1020_MIMXRT1024._aipstz_master_privilege_level (*C enum*), 187
drivers_RT_RT1020_MIMXRT1024._aipstz_master_privilege_level.kAIPSTZ_MasterBufferedWriteEnable (*C enumerator*), 187
drivers_RT_RT1020_MIMXRT1024._aipstz_master_privilege_level.kAIPSTZ_MasterForceUserModeEnable (*C enumerator*), 187
drivers_RT_RT1020_MIMXRT1024._aipstz_master_privilege_level.kAIPSTZ_MasterTrustedForReadEnable (*C enumerator*), 187
drivers_RT_RT1020_MIMXRT1024._aipstz_master_privilege_level.kAIPSTZ_MasterTrustedForWriteEnable (*C enumerator*), 187
drivers_RT_RT1020_MIMXRT1024._aipstz_master.kAIPSTZ_Master0 (*C enumerator*), 187
drivers_RT_RT1020_MIMXRT1024._aipstz_master.kAIPSTZ_Master1 (*C enumerator*), 187
drivers_RT_RT1020_MIMXRT1024._aipstz_master.kAIPSTZ_Master2 (*C enumerator*), 187
drivers_RT_RT1020_MIMXRT1024._aipstz_master.kAIPSTZ_Master3 (*C enumerator*), 187
drivers_RT_RT1020_MIMXRT1024._aipstz_master.kAIPSTZ_Master5 (*C enumerator*), 187
drivers_RT_RT1020_MIMXRT1024._aipstz_peripheral (*C enum*), 188
drivers_RT_RT1020_MIMXRT1024._aipstz_peripheral_access_control (*C enum*), 187
drivers_RT_RT1020_MIMXRT1024._aipstz_peripheral_access_control.kAIPSTZ_PeripheralAllowBufferedWrite

(*C enumerator*), 188
drivers_RT_RT1020_MIMXRT1024._aipstz_peripheral_access_control.kAIPSTZ_PeripheralAllowUntrustedMaster (*C enumerator*), 187
drivers_RT_RT1020_MIMXRT1024._aipstz_peripheral_access_control.kAIPSTZ_PeripheralRequireSupervisor (*C enumerator*), 188
drivers_RT_RT1020_MIMXRT1024._aipstz_peripheral_access_control.kAIPSTZ_PeripheralWriteProtected (*C enumerator*), 187
drivers_RT_RT1020_MIMXRT1024._aipstz_peripheral.kAIPSTZ_Peripheral0 (*C enumerator*), 188
drivers_RT_RT1020_MIMXRT1024._aipstz_peripheral.kAIPSTZ_Peripheral1 (*C enumerator*), 188
drivers_RT_RT1020_MIMXRT1024._aipstz_peripheral.kAIPSTZ_Peripheral2 (*C enumerator*), 188
drivers_RT_RT1020_MIMXRT1024._aipstz_peripheral.kAIPSTZ_Peripheral3 (*C enumerator*), 188
drivers_RT_RT1020_MIMXRT1024._aipstz_peripheral.kAIPSTZ_Peripheral4 (*C enumerator*), 188
drivers_RT_RT1020_MIMXRT1024._aipstz_peripheral.kAIPSTZ_Peripheral5 (*C enumerator*), 188
drivers_RT_RT1020_MIMXRT1024._aipstz_peripheral.kAIPSTZ_Peripheral6 (*C enumerator*), 188
drivers_RT_RT1020_MIMXRT1024._aipstz_peripheral.kAIPSTZ_Peripheral7 (*C enumerator*), 188
drivers_RT_RT1020_MIMXRT1024._aipstz_peripheral.kAIPSTZ_Peripheral8 (*C enumerator*), 188
drivers_RT_RT1020_MIMXRT1024._aipstz_peripheral.kAIPSTZ_Peripheral9 (*C enumerator*), 188
drivers_RT_RT1020_MIMXRT1024._aipstz_peripheral.kAIPSTZ_Peripheral10 (*C enumerator*), 188
drivers_RT_RT1020_MIMXRT1024._aipstz_peripheral.kAIPSTZ_Peripheral11 (*C enumerator*), 188
drivers_RT_RT1020_MIMXRT1024._aipstz_peripheral.kAIPSTZ_Peripheral12 (*C enumerator*), 188
drivers_RT_RT1020_MIMXRT1024._aipstz_peripheral.kAIPSTZ_Peripheral13 (*C enumerator*), 188
drivers_RT_RT1020_MIMXRT1024._aipstz_peripheral.kAIPSTZ_Peripheral14 (*C enumerator*), 188
drivers_RT_RT1020_MIMXRT1024._aipstz_peripheral.kAIPSTZ_Peripheral15 (*C enumerator*), 188
drivers_RT_RT1020_MIMXRT1024._aipstz_peripheral.kAIPSTZ_Peripheral16 (*C enumerator*), 188
drivers_RT_RT1020_MIMXRT1024._aipstz_peripheral.kAIPSTZ_Peripheral17 (*C enumerator*), 188
drivers_RT_RT1020_MIMXRT1024._aipstz_peripheral.kAIPSTZ_Peripheral18 (*C enumerator*), 189
drivers_RT_RT1020_MIMXRT1024._aipstz_peripheral.kAIPSTZ_Peripheral19 (*C enumerator*), 189
drivers_RT_RT1020_MIMXRT1024._aipstz_peripheral.kAIPSTZ_Peripheral20 (*C enumerator*), 189
drivers_RT_RT1020_MIMXRT1024._aipstz_peripheral.kAIPSTZ_Peripheral21 (*C enumerator*), 189
drivers_RT_RT1020_MIMXRT1024._aipstz_peripheral.kAIPSTZ_Peripheral22 (*C enumerator*), 189
drivers_RT_RT1020_MIMXRT1024._aipstz_peripheral.kAIPSTZ_Peripheral23 (*C enumerator*), 189
drivers_RT_RT1020_MIMXRT1024._aipstz_peripheral.kAIPSTZ_Peripheral24 (*C enumerator*), 189
drivers_RT_RT1020_MIMXRT1024._aipstz_peripheral.kAIPSTZ_Peripheral25 (*C enumerator*), 189
drivers_RT_RT1020_MIMXRT1024._aipstz_peripheral.kAIPSTZ_Peripheral26 (*C enumerator*), 189
drivers_RT_RT1020_MIMXRT1024._aipstz_peripheral.kAIPSTZ_Peripheral27 (*C enumerator*), 189
drivers_RT_RT1020_MIMXRT1024._aipstz_peripheral.kAIPSTZ_Peripheral28 (*C enumerator*), 189
drivers_RT_RT1020_MIMXRT1024._aipstz_peripheral.kAIPSTZ_Peripheral29 (*C enumerator*), 189
drivers_RT_RT1020_MIMXRT1024._aipstz_peripheral.kAIPSTZ_Peripheral30 (*C enumerator*), 189
drivers_RT_RT1020_MIMXRT1024._aipstz_peripheral.kAIPSTZ_Peripheral31 (*C enumerator*), 189
drivers_RT_RT1020_MIMXRT1024._aipstz_peripheral.kAIPSTZ_Peripheral32 (*C enumerator*), 189
drivers_RT_RT1020_MIMXRT1024._aipstz_peripheral.kAIPSTZ_Peripheral33 (*C enumerator*), 189
drivers_RT_RT1020_MIMXRT1024._aoi_event (*C enum*), 191
drivers_RT_RT1020_MIMXRT1024._aoi_event_config (*C struct*), 192
drivers_RT_RT1020_MIMXRT1024._aoi_event_config.PT0AC (*C var*), 192
drivers_RT_RT1020_MIMXRT1024._aoi_event_config.PT0BC (*C var*), 192
drivers_RT_RT1020_MIMXRT1024._aoi_event_config.PT0CC (*C var*), 192
drivers_RT_RT1020_MIMXRT1024._aoi_event_config.PT0DC (*C var*), 192
drivers_RT_RT1020_MIMXRT1024._aoi_event_config.PT1AC (*C var*), 192
drivers_RT_RT1020_MIMXRT1024._aoi_event_config.PT1BC (*C var*), 192
drivers_RT_RT1020_MIMXRT1024._aoi_event_config.PT1CC (*C var*), 192
drivers_RT_RT1020_MIMXRT1024._aoi_event_config.PT1DC (*C var*), 192
drivers_RT_RT1020_MIMXRT1024._aoi_event_config.PT2AC (*C var*), 193
drivers_RT_RT1020_MIMXRT1024._aoi_event_config.PT2BC (*C var*), 193
drivers_RT_RT1020_MIMXRT1024._aoi_event_config.PT2CC (*C var*), 193
drivers_RT_RT1020_MIMXRT1024._aoi_event_config.PT2DC (*C var*), 193
drivers_RT_RT1020_MIMXRT1024._aoi_event_config.PT3AC (*C var*), 193
drivers_RT_RT1020_MIMXRT1024._aoi_event_config.PT3BC (*C var*), 193
drivers_RT_RT1020_MIMXRT1024._aoi_event_config.PT3CC (*C var*), 193

drivers_RT_RT1020_MIMXRT1024._aoi_event_config.PT3DC (*C var*), 193
drivers_RT_RT1020_MIMXRT1024._aoi_event.kAOI_Event0 (*C enumerator*), 191
drivers_RT_RT1020_MIMXRT1024._aoi_event.kAOI_Event1 (*C enumerator*), 191
drivers_RT_RT1020_MIMXRT1024._aoi_event.kAOI_Event2 (*C enumerator*), 191
drivers_RT_RT1020_MIMXRT1024._aoi_event.kAOI_Event3 (*C enumerator*), 191
drivers_RT_RT1020_MIMXRT1024._aoi_input_config (*C enum*), 191
drivers_RT_RT1020_MIMXRT1024._aoi_input_config.kAOI_InputSignal (*C enumerator*), 191
drivers_RT_RT1020_MIMXRT1024._aoi_input_config.kAOI_InvInputSignal (*C enumerator*), 191
drivers_RT_RT1020_MIMXRT1024._aoi_input_config.kAOI_LogicOne (*C enumerator*), 191
drivers_RT_RT1020_MIMXRT1024._aoi_input_config.kAOI_LogicZero (*C enumerator*), 191
drivers_RT_RT1020_MIMXRT1024._bee_ac_prot_enable (*C enum*), 194
drivers_RT_RT1020_MIMXRT1024._bee_ac_prot_enable.kBEE_AccessProtDisabled (*C enumerator*), 194
drivers_RT_RT1020_MIMXRT1024._bee_ac_prot_enable.kBEE_AccessProtEnabled (*C enumerator*), 194
drivers_RT_RT1020_MIMXRT1024._bee_aes_mode (*C enum*), 194
drivers_RT_RT1020_MIMXRT1024._bee_aes_mode.kBEE_AesCtrMode (*C enumerator*), 194
drivers_RT_RT1020_MIMXRT1024._bee_aes_mode.kBEE_AesEcbMode (*C enumerator*), 194
drivers_RT_RT1020_MIMXRT1024._bee_endian_swap_enable (*C enum*), 194
drivers_RT_RT1020_MIMXRT1024._bee_endian_swap_enable.kBEE_EndianSwapDisabled (*C enumerator*), 194
drivers_RT_RT1020_MIMXRT1024._bee_endian_swap_enable.kBEE_EndianSwapEnabled (*C enumerator*), 195
drivers_RT_RT1020_MIMXRT1024._bee_region (*C enum*), 194
drivers_RT_RT1020_MIMXRT1024._bee_region_config (*C struct*), 198
drivers_RT_RT1020_MIMXRT1024._bee_region_config.accessPermission (*C var*), 198
drivers_RT_RT1020_MIMXRT1024._bee_region_config.endianSwapEn (*C var*), 199
drivers_RT_RT1020_MIMXRT1024._bee_region_config.region0AddrOffset (*C var*), 198
drivers_RT_RT1020_MIMXRT1024._bee_region_config.region0Mode (*C var*), 198
drivers_RT_RT1020_MIMXRT1024._bee_region_config.region0SecLevel (*C var*), 198
drivers_RT_RT1020_MIMXRT1024._bee_region_config.region1AddrOffset (*C var*), 198
drivers_RT_RT1020_MIMXRT1024._bee_region_config.region1Bot (*C var*), 198
drivers_RT_RT1020_MIMXRT1024._bee_region_config.region1Mode (*C var*), 198
drivers_RT_RT1020_MIMXRT1024._bee_region_config.region1SecLevel (*C var*), 198
drivers_RT_RT1020_MIMXRT1024._bee_region_config.region1Top (*C var*), 198
drivers_RT_RT1020_MIMXRT1024._bee_region.kBEE_Region0 (*C enumerator*), 194
drivers_RT_RT1020_MIMXRT1024._bee_region.kBEE_Region1 (*C enumerator*), 194
drivers_RT_RT1020_MIMXRT1024._bee_security_level (*C enum*), 195
drivers_RT_RT1020_MIMXRT1024._bee_security_level.kBEE_SecurityLevel0 (*C enumerator*), 195
drivers_RT_RT1020_MIMXRT1024._bee_security_level.kBEE_SecurityLevel1 (*C enumerator*), 195
drivers_RT_RT1020_MIMXRT1024._bee_security_level.kBEE_SecurityLevel2 (*C enumerator*), 195
drivers_RT_RT1020_MIMXRT1024._bee_security_level.kBEE_SecurityLevel3 (*C enumerator*), 195
drivers_RT_RT1020_MIMXRT1024._bee_status_flags (*C enum*), 195
drivers_RT_RT1020_MIMXRT1024._bee_status_flags.kBEE_DisableAbortFlag (*C enumerator*), 195
drivers_RT_RT1020_MIMXRT1024._bee_status_flags.kBEE_IdleFlag (*C enumerator*), 195
drivers_RT_RT1020_MIMXRT1024._bee_status_flags.kBEE_ReadIllegalAccess (*C enumerator*), 195
drivers_RT_RT1020_MIMXRT1024._bee_status_flags.kBEE_Reg0AccessViolation (*C enumerator*), 195
drivers_RT_RT1020_MIMXRT1024._bee_status_flags.kBEE_Reg0ReadSecViolation (*C enumerator*), 195
drivers_RT_RT1020_MIMXRT1024._bee_status_flags.kBEE_Reg1AccessViolation (*C enumerator*), 195
drivers_RT_RT1020_MIMXRT1024._bee_status_flags.kBEE_Reg1ReadSecViolation (*C enumerator*), 195
drivers_RT_RT1020_MIMXRT1024._clock_audio_pll_config (*C struct*), 244
drivers_RT_RT1020_MIMXRT1024._clock_audio_pll_config.denominator (*C var*), 244

drivers_RT_RT1020_MIMXRT1024._clock_audio_pll_config.loopDivider (*C var*), 244
drivers_RT_RT1020_MIMXRT1024._clock_audio_pll_config.numerator (*C var*), 244
drivers_RT_RT1020_MIMXRT1024._clock_audio_pll_config.postDivider (*C var*), 244
drivers_RT_RT1020_MIMXRT1024._clock_audio_pll_config.src (*C var*), 244
drivers_RT_RT1020_MIMXRT1024._clock_div (*C enum*), 211
drivers_RT_RT1020_MIMXRT1024._clock_div_value (*C enum*), 213
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_AhbDivBy1 (*C enumerator*), 215
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_AhbDivBy2 (*C enumerator*), 215
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_AhbDivBy3 (*C enumerator*), 215
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_AhbDivBy4 (*C enumerator*), 215
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_AhbDivBy5 (*C enumerator*), 215
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_AhbDivBy6 (*C enumerator*), 215
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_AhbDivBy7 (*C enumerator*), 215
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_AhbDivBy8 (*C enumerator*), 215
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_ArmDivBy1 (*C enumerator*), 213
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_ArmDivBy2 (*C enumerator*), 213
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_ArmDivBy3 (*C enumerator*), 213
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_ArmDivBy4 (*C enumerator*), 213
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_ArmDivBy5 (*C enumerator*), 213
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_ArmDivBy6 (*C enumerator*), 213
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_ArmDivBy7 (*C enumerator*), 213
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_ArmDivBy8 (*C enumerator*), 214
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_Flexio1DivBy1 (*C enumerator*), 218
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_Flexio1DivBy2 (*C enumerator*), 218
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_Flexio1DivBy3 (*C enumerator*), 218
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_Flexio1DivBy4 (*C enumerator*), 218
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_Flexio1DivBy5 (*C enumerator*), 218
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_Flexio1DivBy6 (*C enumerator*), 218
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_Flexio1DivBy7 (*C enumerator*), 218
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_Flexio1DivBy8 (*C enumerator*), 218
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_Flexio1PreDivBy1 (*C enumerator*), 219
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_Flexio1PreDivBy2 (*C enumerator*), 219
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_Flexio1PreDivBy3 (*C enumerator*), 219
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_Flexio1PreDivBy4 (*C enumerator*), 219
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_Flexio1PreDivBy5 (*C enumerator*), 219
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_Flexio1PreDivBy6 (*C enumerator*), 219
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_Flexio1PreDivBy7 (*C enumerator*), 219
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_Flexio1PreDivBy8 (*C enumerator*), 219
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_FlexspiDivBy1 (*C enumerator*), 216
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_FlexspiDivBy2 (*C enumerator*), 216

drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_FlexspiDivBy3 (*C enumerator*), 216
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_FlexspiDivBy4 (*C enumerator*), 216
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_FlexspiDivBy5 (*C enumerator*), 216
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_FlexspiDivBy6 (*C enumerator*), 216
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_FlexspiDivBy7 (*C enumerator*), 216
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_FlexspiDivBy8 (*C enumerator*), 216
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_IpgDivBy1 (*C enumerator*), 215
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_IpgDivBy2 (*C enumerator*), 215
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_IpgDivBy3 (*C enumerator*), 215
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_IpgDivBy4 (*C enumerator*), 215
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_LpspiDivBy1 (*C enumerator*), 215
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_LpspiDivBy2 (*C enumerator*), 215
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_LpspiDivBy3 (*C enumerator*), 215
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_LpspiDivBy4 (*C enumerator*), 216
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_LpspiDivBy5 (*C enumerator*), 216
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_LpspiDivBy6 (*C enumerator*), 216
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_LpspiDivBy7 (*C enumerator*), 216
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_LpspiDivBy8 (*C enumerator*), 216
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_MiscDivBy1 (*C enumerator*), 221
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_MiscDivBy2 (*C enumerator*), 221
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_MiscDivBy3 (*C enumerator*), 221
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_MiscDivBy4 (*C enumerator*), 221
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_MiscDivBy5 (*C enumerator*), 221
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_MiscDivBy6 (*C enumerator*), 221
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_MiscDivBy7 (*C enumerator*), 221
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_MiscDivBy8 (*C enumerator*), 222
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_MiscDivBy9 (*C enumerator*), 222
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_MiscDivBy10 (*C enumerator*), 222
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_MiscDivBy11 (*C enumerator*), 222
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_MiscDivBy12 (*C enumerator*), 222
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_MiscDivBy13 (*C enumerator*), 222
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_MiscDivBy14 (*C enumerator*), 222
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_MiscDivBy15 (*C enumerator*), 222
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_MiscDivBy16 (*C enumerator*), 222
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_MiscDivBy17 (*C enumerator*), 222
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_MiscDivBy18 (*C enumerator*), 222
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_MiscDivBy19 (*C enumerator*), 222
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_MiscDivBy20 (*C enumerator*), 222
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_MiscDivBy21 (*C enumerator*), 222
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_MiscDivBy22 (*C enumerator*), 222
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_MiscDivBy23 (*C enumerator*), 222
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_MiscDivBy24 (*C enumerator*), 223
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_MiscDivBy25 (*C enumerator*), 223
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_MiscDivBy26 (*C enumerator*), 223
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_MiscDivBy27 (*C enumerator*), 223
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_MiscDivBy28 (*C enumerator*), 223
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_MiscDivBy29 (*C enumerator*), 223
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_MiscDivBy30 (*C enumerator*), 223
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_MiscDivBy31 (*C enumerator*), 223
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_MiscDivBy32 (*C enumerator*), 223
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_MiscDivBy33 (*C enumerator*), 223
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_MiscDivBy34 (*C enumerator*), 223

drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_MiscDivBy35 (*C enumerator*), 223
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_MiscDivBy36 (*C enumerator*), 223
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_MiscDivBy37 (*C enumerator*), 223
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_MiscDivBy38 (*C enumerator*), 223
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_MiscDivBy39 (*C enumerator*), 223
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_MiscDivBy40 (*C enumerator*), 224
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_MiscDivBy41 (*C enumerator*), 224
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_MiscDivBy42 (*C enumerator*), 224
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_MiscDivBy43 (*C enumerator*), 224
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_MiscDivBy44 (*C enumerator*), 224
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_MiscDivBy45 (*C enumerator*), 224
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_MiscDivBy46 (*C enumerator*), 224
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_MiscDivBy47 (*C enumerator*), 224
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_MiscDivBy48 (*C enumerator*), 224
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_MiscDivBy49 (*C enumerator*), 224
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_MiscDivBy50 (*C enumerator*), 224
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_MiscDivBy51 (*C enumerator*), 224
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_MiscDivBy52 (*C enumerator*), 224
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_MiscDivBy53 (*C enumerator*), 224
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_MiscDivBy54 (*C enumerator*), 224
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_MiscDivBy55 (*C enumerator*), 224
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_MiscDivBy56 (*C enumerator*), 225
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_MiscDivBy57 (*C enumerator*), 225
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_MiscDivBy58 (*C enumerator*), 225
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_MiscDivBy59 (*C enumerator*), 225
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_MiscDivBy60 (*C enumerator*), 225
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_MiscDivBy61 (*C enumerator*), 225
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_MiscDivBy62 (*C enumerator*), 225
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_MiscDivBy63 (*C enumerator*), 225
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_MiscDivBy64 (*C enumerator*), 225
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_PeriphClk2DivBy1 (*C enumerator*), 214
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_PeriphClk2DivBy2 (*C enumerator*), 214
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_PeriphClk2DivBy3 (*C enumerator*), 214
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_PeriphClk2DivBy4 (*C enumerator*), 214
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_PeriphClk2DivBy5 (*C enumerator*), 214
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_PeriphClk2DivBy6 (*C enumerator*), 214
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_PeriphClk2DivBy7 (*C enumerator*), 214
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_PeriphClk2DivBy8 (*C enumerator*), 214
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_Sai1PreDivBy1 (*C enumerator*), 219
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_Sai1PreDivBy2 (*C enumerator*), 219
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_Sai1PreDivBy3 (*C enumerator*), 219
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_Sai1PreDivBy4 (*C enumerator*), 219
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_Sai1PreDivBy5 (*C enumerator*), 219
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_Sai1PreDivBy6 (*C enumerator*), 219

drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_Sai1PreDivBy7 (*C enumerator*), 219
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_Sai1PreDivBy8 (*C enumerator*), 220
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_Sai2PreDivBy1 (*C enumerator*), 220
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_Sai2PreDivBy2 (*C enumerator*), 220
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_Sai2PreDivBy3 (*C enumerator*), 220
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_Sai2PreDivBy4 (*C enumerator*), 220
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_Sai2PreDivBy5 (*C enumerator*), 220
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_Sai2PreDivBy6 (*C enumerator*), 220
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_Sai2PreDivBy7 (*C enumerator*), 220
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_Sai2PreDivBy8 (*C enumerator*), 220
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_Sai3PreDivBy1 (*C enumerator*), 218
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_Sai3PreDivBy2 (*C enumerator*), 218
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_Sai3PreDivBy3 (*C enumerator*), 218
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_Sai3PreDivBy4 (*C enumerator*), 218
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_Sai3PreDivBy5 (*C enumerator*), 218
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_Sai3PreDivBy6 (*C enumerator*), 218
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_Sai3PreDivBy7 (*C enumerator*), 218
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_Sai3PreDivBy8 (*C enumerator*), 219
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_SemcDivBy1 (*C enumerator*), 214
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_SemcDivBy2 (*C enumerator*), 214
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_SemcDivBy3 (*C enumerator*), 214
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_SemcDivBy4 (*C enumerator*), 214
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_SemcDivBy5 (*C enumerator*), 214
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_SemcDivBy6 (*C enumerator*), 214
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_SemcDivBy7 (*C enumerator*), 214
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_SemcDivBy8 (*C enumerator*), 215
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_Spdif0DivBy1 (*C enumerator*), 221
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_Spdif0DivBy2 (*C enumerator*), 221
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_Spdif0DivBy3 (*C enumerator*), 221
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_Spdif0DivBy4 (*C enumerator*), 221
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_Spdif0DivBy5 (*C enumerator*), 221
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_Spdif0DivBy6 (*C enumerator*), 221
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_Spdif0DivBy7 (*C enumerator*), 221
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_Spdif0DivBy8 (*C enumerator*), 221
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_Spdif0PreDivBy1 (*C enumerator*), 220
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_Spdif0PreDivBy2 (*C enumerator*), 220
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_Spdif0PreDivBy3 (*C enumerator*), 220

drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_Spdif0PreDivBy4 (*C enumerator*), 220
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_Spdif0PreDivBy5 (*C enumerator*), 220
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_Spdif0PreDivBy6 (*C enumerator*), 220
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_Spdif0PreDivBy7 (*C enumerator*), 220
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_Spdif0PreDivBy8 (*C enumerator*), 221
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_TraceDivBy1 (*C enumerator*), 216
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_TraceDivBy2 (*C enumerator*), 216
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_TraceDivBy3 (*C enumerator*), 216
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_TraceDivBy4 (*C enumerator*), 217
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_Usdhc1DivBy1 (*C enumerator*), 217
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_Usdhc1DivBy2 (*C enumerator*), 217
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_Usdhc1DivBy3 (*C enumerator*), 217
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_Usdhc1DivBy4 (*C enumerator*), 217
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_Usdhc1DivBy5 (*C enumerator*), 217
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_Usdhc1DivBy6 (*C enumerator*), 217
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_Usdhc1DivBy7 (*C enumerator*), 217
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_Usdhc1DivBy8 (*C enumerator*), 218
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_Usdhc2DivBy1 (*C enumerator*), 217
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_Usdhc2DivBy2 (*C enumerator*), 217
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_Usdhc2DivBy3 (*C enumerator*), 217
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_Usdhc2DivBy4 (*C enumerator*), 217
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_Usdhc2DivBy5 (*C enumerator*), 217
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_Usdhc2DivBy6 (*C enumerator*), 217
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_Usdhc2DivBy7 (*C enumerator*), 217
drivers_RT_RT1020_MIMXRT1024._clock_div_value.kCLOCK_Usdhc2DivBy8 (*C enumerator*), 217
drivers_RT_RT1020_MIMXRT1024._clock_div.kCLOCK_AhbDiv (*C enumerator*), 212
drivers_RT_RT1020_MIMXRT1024._clock_div.kCLOCK_ArmDiv (*C enumerator*), 211
drivers_RT_RT1020_MIMXRT1024._clock_div.kCLOCK_CanDiv (*C enumerator*), 212
drivers_RT_RT1020_MIMXRT1024._clock_div.kCLOCK_Flexio1Div (*C enumerator*), 212
drivers_RT_RT1020_MIMXRT1024._clock_div.kCLOCK_Flexio1PreDiv (*C enumerator*), 212
drivers_RT_RT1020_MIMXRT1024._clock_div.kCLOCK_FlexspiDiv (*C enumerator*), 212
drivers_RT_RT1020_MIMXRT1024._clock_div.kCLOCK_IpgDiv (*C enumerator*), 212
drivers_RT_RT1020_MIMXRT1024._clock_div.kCLOCK_Lpi2cDiv (*C enumerator*), 213
drivers_RT_RT1020_MIMXRT1024._clock_div.kCLOCK_LpspiDiv (*C enumerator*), 212
drivers_RT_RT1020_MIMXRT1024._clock_div.kCLOCK_NonePreDiv (*C enumerator*), 213
drivers_RT_RT1020_MIMXRT1024._clock_div.kCLOCK_PerclkDiv (*C enumerator*), 212
drivers_RT_RT1020_MIMXRT1024._clock_div.kCLOCK_PeriphClk2Div (*C enumerator*), 212

drivers_RT_RT1020_MIMXRT1024._clock_div.kCLOCK_Sai1Div (*C enumerator*), 213
drivers_RT_RT1020_MIMXRT1024._clock_div.kCLOCK_Sai1PreDiv (*C enumerator*), 213
drivers_RT_RT1020_MIMXRT1024._clock_div.kCLOCK_Sai2Div (*C enumerator*), 213
drivers_RT_RT1020_MIMXRT1024._clock_div.kCLOCK_Sai2PreDiv (*C enumerator*), 213
drivers_RT_RT1020_MIMXRT1024._clock_div.kCLOCK_Sai3Div (*C enumerator*), 212
drivers_RT_RT1020_MIMXRT1024._clock_div.kCLOCK_Sai3PreDiv (*C enumerator*), 212
drivers_RT_RT1020_MIMXRT1024._clock_div.kCLOCK_SemcDiv (*C enumerator*), 212
drivers_RT_RT1020_MIMXRT1024._clock_div.kCLOCK_Spdif0Div (*C enumerator*), 213
drivers_RT_RT1020_MIMXRT1024._clock_div.kCLOCK_Spdif0PreDiv (*C enumerator*), 213
drivers_RT_RT1020_MIMXRT1024._clock_div.kCLOCK_TraceDiv (*C enumerator*), 212
drivers_RT_RT1020_MIMXRT1024._clock_div.kCLOCK_UartDiv (*C enumerator*), 212
drivers_RT_RT1020_MIMXRT1024._clock_div.kCLOCK_Usdhc1Div (*C enumerator*), 212
drivers_RT_RT1020_MIMXRT1024._clock_div.kCLOCK_Usdhc2Div (*C enumerator*), 212
drivers_RT_RT1020_MIMXRT1024._clock_enet_pll_config (*C struct*), 244
drivers_RT_RT1020_MIMXRT1024._clock_enet_pll_config.enableClkOutput (*C var*), 245
drivers_RT_RT1020_MIMXRT1024._clock_enet_pll_config.enableClkOutput25M (*C var*), 245
drivers_RT_RT1020_MIMXRT1024._clock_enet_pll_config.enableClkOutput500M (*C var*), 245
drivers_RT_RT1020_MIMXRT1024._clock_enet_pll_config.loopDivider (*C var*), 245
drivers_RT_RT1020_MIMXRT1024._clock_enet_pll_config.src (*C var*), 245
drivers_RT_RT1020_MIMXRT1024._clock_gate_value (*C enum*), 209
drivers_RT_RT1020_MIMXRT1024._clock_gate_value.kCLOCK_ClockNeededRun (*C enumerator*), 209
drivers_RT_RT1020_MIMXRT1024._clock_gate_value.kCLOCK_ClockNeededRunWait (*C enumerator*), 210
drivers_RT_RT1020_MIMXRT1024._clock_gate_value.kCLOCK_ClockNotNeeded (*C enumerator*), 209
drivers_RT_RT1020_MIMXRT1024._clock_ip_name (*C enum*), 203
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_Acmp1 (*C enumerator*), 206
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_Acmp2 (*C enumerator*), 206
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_Acmp3 (*C enumerator*), 206
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_Acmp4 (*C enumerator*), 206
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_Adc1 (*C enumerator*), 205
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_Adc2 (*C enumerator*), 204
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_Aips_tz1 (*C enumerator*), 203
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_Aips_tz2 (*C enumerator*), 203
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_Aips_tz3 (*C enumerator*), 209
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_Aips_tz4 (*C enumerator*), 208
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_Anadig (*C enumerator*), 209
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_Aoi (*C enumerator*), 206
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_Bee (*C enumerator*), 207
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_Can1 (*C enumerator*), 204
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_Can1S (*C enumerator*), 204
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_Can2 (*C enumerator*), 204
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_Can2S (*C enumerator*), 204
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_Csu (*C enumerator*), 205
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_Dcdc (*C enumerator*), 208
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_Dcp (*C enumerator*), 204
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_Dma (*C enumerator*), 207
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_Enc1 (*C enumerator*), 207
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_Enc2 (*C enumerator*), 207
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_Enet (*C enumerator*), 205
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_Ewm0 (*C enumerator*), 206
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_Flexio1 (*C enumerator*), 207
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_FlexRam (*C enumerator*), 206
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_FlexSpi (*C enumerator*), 208
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_FlexSpiExsc (*C enumerator*), 203
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_Gpio1 (*C enumerator*), 205
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_Gpio2 (*C enumerator*), 204

drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_Gpio3 (*C enumerator*), 206
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_Gpio5 (*C enumerator*), 205
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_Gpt1 (*C enumerator*), 205
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_Gpt1S (*C enumerator*), 205
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_Gpt2 (*C enumerator*), 204
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_Gpt2S (*C enumerator*), 204
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_Iomuxc (*C enumerator*), 207
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_IomuxcGpr (*C enumerator*), 207
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_IomuxcSnvs (*C enumerator*), 205
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_IomuxcSnvsGpr (*C enumerator*), 206
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_IpInvalid (*C enumerator*), 203
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_Ipmux4 (*C enumerator*), 208
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_Kpp (*C enumerator*), 207
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_Lpi2c1 (*C enumerator*), 205
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_Lpi2c2 (*C enumerator*), 205
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_Lpi2c3 (*C enumerator*), 205
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_Lpi2c4 (*C enumerator*), 209
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_Lpspi1 (*C enumerator*), 204
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_Lpspi2 (*C enumerator*), 204
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_Lpspi3 (*C enumerator*), 204
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_Lpspi4 (*C enumerator*), 204
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_Lpuart1 (*C enumerator*), 208
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_Lpuart2 (*C enumerator*), 204
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_Lpuart3 (*C enumerator*), 204
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_Lpuart4 (*C enumerator*), 205
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_Lpuart5 (*C enumerator*), 206
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_Lpuart6 (*C enumerator*), 206
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_Lpuart7 (*C enumerator*), 208
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_Lpuart8 (*C enumerator*), 209
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_Mqs (*C enumerator*), 203
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_Ocotp (*C enumerator*), 205
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_OcramExsc (*C enumerator*), 205
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_Pit (*C enumerator*), 205
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_Pwm1 (*C enumerator*), 207
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_Pwm2 (*C enumerator*), 207
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_Rom (*C enumerator*), 207
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_Sai1 (*C enumerator*), 208
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_Sai2 (*C enumerator*), 208
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_Sai3 (*C enumerator*), 208
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_Semc (*C enumerator*), 206
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_SemcExsc (*C enumerator*), 205
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_Sim_m7_clk_r (*C enumerator*), 206
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_Sim_m_clk_r (*C enumerator*), 203
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_SimEms (*C enumerator*), 207
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_SimM (*C enumerator*), 207
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_SimM7 (*C enumerator*), 207
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_SimPer (*C enumerator*), 209
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_SnvsHp (*C enumerator*), 208
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_SnvsLp (*C enumerator*), 208
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_Spdif (*C enumerator*), 208
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_Timer1 (*C enumerator*), 209
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_Timer2 (*C enumerator*), 209
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_Timer4 (*C enumerator*), 209
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_Trace (*C enumerator*), 204
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_Trng (*C enumerator*), 209
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_Tsc (*C enumerator*), 207

drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_UsbOh3 (*C enumerator*), 208
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_Usdhc1 (*C enumerator*), 208
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_Usdhc2 (*C enumerator*), 208
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_Wdog1 (*C enumerator*), 206
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_Wdog2 (*C enumerator*), 208
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_Wdog3 (*C enumerator*), 207
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_Xbar1 (*C enumerator*), 206
drivers_RT_RT1020_MIMXRT1024._clock_ip_name.kCLOCK_Xbar2 (*C enumerator*), 206
drivers_RT_RT1020_MIMXRT1024._clock_mode_t (*C enum*), 210
drivers_RT_RT1020_MIMXRT1024._clock_mode_t.kCLOCK_ModeRun (*C enumerator*), 210
drivers_RT_RT1020_MIMXRT1024._clock_mode_t.kCLOCK_ModeStop (*C enumerator*), 210
drivers_RT_RT1020_MIMXRT1024._clock_mode_t.kCLOCK_ModeWait (*C enumerator*), 210
drivers_RT_RT1020_MIMXRT1024._clock_mux (*C enum*), 210
drivers_RT_RT1020_MIMXRT1024._clock_mux.kCLOCK_CanMux (*C enumerator*), 211
drivers_RT_RT1020_MIMXRT1024._clock_mux.kCLOCK_Flexio1Mux (*C enumerator*), 211
drivers_RT_RT1020_MIMXRT1024._clock_mux.kCLOCK_FlexspiMux (*C enumerator*), 211
drivers_RT_RT1020_MIMXRT1024._clock_mux.kCLOCK_Lpi2cMux (*C enumerator*), 211
drivers_RT_RT1020_MIMXRT1024._clock_mux.kCLOCK_LpspiMux (*C enumerator*), 210
drivers_RT_RT1020_MIMXRT1024._clock_mux.kCLOCK_PerclkMux (*C enumerator*), 211
drivers_RT_RT1020_MIMXRT1024._clock_mux.kCLOCK_PeriphClk2Mux (*C enumerator*), 210
drivers_RT_RT1020_MIMXRT1024._clock_mux.kCLOCK_PeriphMux (*C enumerator*), 210
drivers_RT_RT1020_MIMXRT1024._clock_mux.kCLOCK_Pll3SwMux (*C enumerator*), 210
drivers_RT_RT1020_MIMXRT1024._clock_mux.kCLOCK_PrePeriphMux (*C enumerator*), 210
drivers_RT_RT1020_MIMXRT1024._clock_mux.kCLOCK_Sai1Mux (*C enumerator*), 211
drivers_RT_RT1020_MIMXRT1024._clock_mux.kCLOCK_Sai2Mux (*C enumerator*), 211
drivers_RT_RT1020_MIMXRT1024._clock_mux.kCLOCK_Sai3Mux (*C enumerator*), 211
drivers_RT_RT1020_MIMXRT1024._clock_mux.kCLOCK_SemcAltMux (*C enumerator*), 210
drivers_RT_RT1020_MIMXRT1024._clock_mux.kCLOCK_SemcMux (*C enumerator*), 210
drivers_RT_RT1020_MIMXRT1024._clock_mux.kCLOCK_SpdifMux (*C enumerator*), 211
drivers_RT_RT1020_MIMXRT1024._clock_mux.kCLOCK_TraceMux (*C enumerator*), 210
drivers_RT_RT1020_MIMXRT1024._clock_mux.kCLOCK_UartMux (*C enumerator*), 211
drivers_RT_RT1020_MIMXRT1024._clock_mux.kCLOCK_Usdhc1Mux (*C enumerator*), 211
drivers_RT_RT1020_MIMXRT1024._clock_mux.kCLOCK_Usdhc2Mux (*C enumerator*), 211
drivers_RT_RT1020_MIMXRT1024._clock_name (*C enum*), 201
drivers_RT_RT1020_MIMXRT1024._clock_name.kCLOCK_AhbClk (*C enumerator*), 202
drivers_RT_RT1020_MIMXRT1024._clock_name.kCLOCK_AudioPllClk (*C enumerator*), 203
drivers_RT_RT1020_MIMXRT1024._clock_name.kCLOCK_CpuClk (*C enumerator*), 202
drivers_RT_RT1020_MIMXRT1024._clock_name.kCLOCK_EnetPll25MClk (*C enumerator*), 203
drivers_RT_RT1020_MIMXRT1024._clock_name.kCLOCK_EnetPll500MClk (*C enumerator*), 203
drivers_RT_RT1020_MIMXRT1024._clock_name.kCLOCK_EnetPllClk (*C enumerator*), 203
drivers_RT_RT1020_MIMXRT1024._clock_name.kCLOCK_IpgClk (*C enumerator*), 202
drivers_RT_RT1020_MIMXRT1024._clock_name.kCLOCK_NoneName (*C enumerator*), 203
drivers_RT_RT1020_MIMXRT1024._clock_name.kCLOCK_OscClk (*C enumerator*), 202
drivers_RT_RT1020_MIMXRT1024._clock_name.kCLOCK_PerClk (*C enumerator*), 202
drivers_RT_RT1020_MIMXRT1024._clock_name.kCLOCK_RtcClk (*C enumerator*), 202
drivers_RT_RT1020_MIMXRT1024._clock_name.kCLOCK_SemcClk (*C enumerator*), 202
drivers_RT_RT1020_MIMXRT1024._clock_name.kCLOCK_SysPllClk (*C enumerator*), 202
drivers_RT_RT1020_MIMXRT1024._clock_name.kCLOCK_SysPllPfd0Clk (*C enumerator*), 203
drivers_RT_RT1020_MIMXRT1024._clock_name.kCLOCK_SysPllPfd1Clk (*C enumerator*), 203
drivers_RT_RT1020_MIMXRT1024._clock_name.kCLOCK_SysPllPfd2Clk (*C enumerator*), 203
drivers_RT_RT1020_MIMXRT1024._clock_name.kCLOCK_SysPllPfd3Clk (*C enumerator*), 203
drivers_RT_RT1020_MIMXRT1024._clock_name.kCLOCK_Usb1PllClk (*C enumerator*), 202
drivers_RT_RT1020_MIMXRT1024._clock_name.kCLOCK_Usb1PllPfd0Clk (*C enumerator*), 202
drivers_RT_RT1020_MIMXRT1024._clock_name.kCLOCK_Usb1PllPfd1Clk (*C enumerator*), 202
drivers_RT_RT1020_MIMXRT1024._clock_name.kCLOCK_Usb1PllPfd2Clk (*C enumerator*), 202
drivers_RT_RT1020_MIMXRT1024._clock_name.kCLOCK_Usb1PllPfd3Clk (*C enumerator*), 202
drivers_RT_RT1020_MIMXRT1024._clock_name.kCLOCK_Usb1Sw60MClk (*C enumerator*), 202
drivers_RT_RT1020_MIMXRT1024._clock_name.kCLOCK_Usb1Sw80MClk (*C enumerator*), 202

drivers_RT_RT1020_MIMXRT1024._clock_name.kCLOCK_Usb1SwClk (*C enumerator*), 202
drivers_RT_RT1020_MIMXRT1024._clock_osc (*C enum*), 209
drivers_RT_RT1020_MIMXRT1024._clock_osc.kCLOCK_RcOsc (*C enumerator*), 209
drivers_RT_RT1020_MIMXRT1024._clock_osc.kCLOCK_XtalOsc (*C enumerator*), 209
drivers_RT_RT1020_MIMXRT1024._clock_output1_selection (*C enum*), 226
drivers_RT_RT1020_MIMXRT1024._clock_output1_selection.kCLOCK_DisableClockOutput1 (*C enumerator*), 227
drivers_RT_RT1020_MIMXRT1024._clock_output1_selection.kCLOCK_OutputAhbClk (*C enumerator*), 227
drivers_RT_RT1020_MIMXRT1024._clock_output1_selection.kCLOCK_OutputIpgClk (*C enumerator*), 227
drivers_RT_RT1020_MIMXRT1024._clock_output1_selection.kCLOCK_OutputPerClk (*C enumerator*), 227
drivers_RT_RT1020_MIMXRT1024._clock_output1_selection.kCLOCK_OutputPll4MainClk (*C enumerator*), 227
drivers_RT_RT1020_MIMXRT1024._clock_output1_selection.kCLOCK_OutputPllENET500M (*C enumerator*), 227
drivers_RT_RT1020_MIMXRT1024._clock_output1_selection.kCLOCK_OutputPllSys (*C enumerator*), 227
drivers_RT_RT1020_MIMXRT1024._clock_output1_selection.kCLOCK_OutputPllUsb1Sw (*C enumerator*), 227
drivers_RT_RT1020_MIMXRT1024._clock_output1_selection.kCLOCK_OutputSemcClk (*C enumerator*), 227
drivers_RT_RT1020_MIMXRT1024._clock_output2_selection (*C enum*), 227
drivers_RT_RT1020_MIMXRT1024._clock_output2_selection.kCLOCK_DisableClockOutput2 (*C enumerator*), 228
drivers_RT_RT1020_MIMXRT1024._clock_output2_selection.kCLOCK_OutputCanClk (*C enumerator*), 228
drivers_RT_RT1020_MIMXRT1024._clock_output2_selection.kCLOCK_OutputFlexspiClk (*C enumerator*), 228
drivers_RT_RT1020_MIMXRT1024._clock_output2_selection.kCLOCK_OutputLpi2cClk (*C enumerator*), 227
drivers_RT_RT1020_MIMXRT1024._clock_output2_selection.kCLOCK_OutputLpspiClk (*C enumerator*), 227
drivers_RT_RT1020_MIMXRT1024._clock_output2_selection.kCLOCK_OutputOscClk (*C enumerator*), 227
drivers_RT_RT1020_MIMXRT1024._clock_output2_selection.kCLOCK_OutputSai1Clk (*C enumerator*), 228
drivers_RT_RT1020_MIMXRT1024._clock_output2_selection.kCLOCK_OutputSai2Clk (*C enumerator*), 228
drivers_RT_RT1020_MIMXRT1024._clock_output2_selection.kCLOCK_OutputSai3Clk (*C enumerator*), 228
drivers_RT_RT1020_MIMXRT1024._clock_output2_selection.kCLOCK_OutputSpdif0Clk (*C enumerator*), 228
drivers_RT_RT1020_MIMXRT1024._clock_output2_selection.kCLOCK_OutputTraceClk (*C enumerator*), 228
drivers_RT_RT1020_MIMXRT1024._clock_output2_selection.kCLOCK_OutputUartClk (*C enumerator*), 228
drivers_RT_RT1020_MIMXRT1024._clock_output2_selection.kCLOCK_OutputUsdhc1Clk (*C enumerator*), 227
drivers_RT_RT1020_MIMXRT1024._clock_output2_selection.kCLOCK_OutputUsdhc2Clk (*C enumerator*), 227
drivers_RT_RT1020_MIMXRT1024._clock_output_divider (*C enum*), 228
drivers_RT_RT1020_MIMXRT1024._clock_output_divider.kCLOCK_DivideBy1 (*C enumerator*), 228
drivers_RT_RT1020_MIMXRT1024._clock_output_divider.kCLOCK_DivideBy2 (*C enumerator*), 228
drivers_RT_RT1020_MIMXRT1024._clock_output_divider.kCLOCK_DivideBy3 (*C enumerator*),

228
drivers_RT_RT1020_MIMXRT1024._clock_output_divider.kCLOCK_DivideBy4 (*C enumerator*), 228
drivers_RT_RT1020_MIMXRT1024._clock_output_divider.kCLOCK_DivideBy5 (*C enumerator*), 228
drivers_RT_RT1020_MIMXRT1024._clock_output_divider.kCLOCK_DivideBy6 (*C enumerator*), 228
drivers_RT_RT1020_MIMXRT1024._clock_output_divider.kCLOCK_DivideBy7 (*C enumerator*), 229
drivers_RT_RT1020_MIMXRT1024._clock_output_divider.kCLOCK_DivideBy8 (*C enumerator*), 229
drivers_RT_RT1020_MIMXRT1024._clock_pfd (*C enum*), 226
drivers_RT_RT1020_MIMXRT1024._clock_pfd.kCLOCK_Pfd0 (*C enumerator*), 226
drivers_RT_RT1020_MIMXRT1024._clock_pfd.kCLOCK_Pfd1 (*C enumerator*), 226
drivers_RT_RT1020_MIMXRT1024._clock_pfd.kCLOCK_Pfd2 (*C enumerator*), 226
drivers_RT_RT1020_MIMXRT1024._clock_pfd.kCLOCK_Pfd3 (*C enumerator*), 226
drivers_RT_RT1020_MIMXRT1024._clock_pll (*C enum*), 226
drivers_RT_RT1020_MIMXRT1024._clock_pll_clk_src (*C enum*), 225
drivers_RT_RT1020_MIMXRT1024._clock_pll_clk_src.kCLOCK_PllClkSrc24M (*C enumerator*), 226
drivers_RT_RT1020_MIMXRT1024._clock_pll_clk_src.kCLOCK_PllSrcClkPN (*C enumerator*), 226
drivers_RT_RT1020_MIMXRT1024._clock_pll.kCLOCK_PllAudio (*C enumerator*), 226
drivers_RT_RT1020_MIMXRT1024._clock_pll.kCLOCK_PllEnet (*C enumerator*), 226
drivers_RT_RT1020_MIMXRT1024._clock_pll.kCLOCK_PllEnet25M (*C enumerator*), 226
drivers_RT_RT1020_MIMXRT1024._clock_pll.kCLOCK_PllEnet500M (*C enumerator*), 226
drivers_RT_RT1020_MIMXRT1024._clock_pll.kCLOCK_PllSys (*C enumerator*), 226
drivers_RT_RT1020_MIMXRT1024._clock_pll.kCLOCK_PllUsb1 (*C enumerator*), 226
drivers_RT_RT1020_MIMXRT1024._clock_root (*C enum*), 229
drivers_RT_RT1020_MIMXRT1024._clock_root.kCLOCK_CanClkRoot (*C enumerator*), 229
drivers_RT_RT1020_MIMXRT1024._clock_root.kCLOCK_Flexio1ClkRoot (*C enumerator*), 229
drivers_RT_RT1020_MIMXRT1024._clock_root.kCLOCK_FlexspiClkRoot (*C enumerator*), 229
drivers_RT_RT1020_MIMXRT1024._clock_root.kCLOCK_Lpi2cClkRoot (*C enumerator*), 229
drivers_RT_RT1020_MIMXRT1024._clock_root.kCLOCK_LpspiClkRoot (*C enumerator*), 229
drivers_RT_RT1020_MIMXRT1024._clock_root.kCLOCK_Sai1ClkRoot (*C enumerator*), 229
drivers_RT_RT1020_MIMXRT1024._clock_root.kCLOCK_Sai2ClkRoot (*C enumerator*), 229
drivers_RT_RT1020_MIMXRT1024._clock_root.kCLOCK_Sai3ClkRoot (*C enumerator*), 229
drivers_RT_RT1020_MIMXRT1024._clock_root.kCLOCK_SpdifClkRoot (*C enumerator*), 229
drivers_RT_RT1020_MIMXRT1024._clock_root.kCLOCK_TraceClkRoot (*C enumerator*), 229
drivers_RT_RT1020_MIMXRT1024._clock_root.kCLOCK_UartClkRoot (*C enumerator*), 229
drivers_RT_RT1020_MIMXRT1024._clock_root.kCLOCK_Usdhc1ClkRoot (*C enumerator*), 229
drivers_RT_RT1020_MIMXRT1024._clock_root.kCLOCK_Usdhc2ClkRoot (*C enumerator*), 229
drivers_RT_RT1020_MIMXRT1024._clock_sys_pll_config (*C struct*), 243
drivers_RT_RT1020_MIMXRT1024._clock_sys_pll_config.denominator (*C var*), 244
drivers_RT_RT1020_MIMXRT1024._clock_sys_pll_config.loopDivider (*C var*), 244
drivers_RT_RT1020_MIMXRT1024._clock_sys_pll_config.numerator (*C var*), 244
drivers_RT_RT1020_MIMXRT1024._clock_sys_pll_config.src (*C var*), 244
drivers_RT_RT1020_MIMXRT1024._clock_sys_pll_config.ss_enable (*C var*), 244
drivers_RT_RT1020_MIMXRT1024._clock_sys_pll_config.ss_step (*C var*), 244
drivers_RT_RT1020_MIMXRT1024._clock_sys_pll_config.ss_stop (*C var*), 244
drivers_RT_RT1020_MIMXRT1024._clock_usb_phy_src (*C enum*), 225
drivers_RT_RT1020_MIMXRT1024._clock_usb_phy_src.kCLOCK_Usbphy480M (*C enumerator*), 225
drivers_RT_RT1020_MIMXRT1024._clock_usb_pll_config (*C struct*), 243
drivers_RT_RT1020_MIMXRT1024._clock_usb_pll_config.loopDivider (*C var*), 243
drivers_RT_RT1020_MIMXRT1024._clock_usb_pll_config.src (*C var*), 243
drivers_RT_RT1020_MIMXRT1024._clock_usb_src (*C enum*), 225
drivers_RT_RT1020_MIMXRT1024._clock_usb_src.kCLOCK_Usb480M (*C enumerator*), 225

drivers_RT_RT1020_MIMXRT1024._clock_usb_src.kCLOCK_UsbSrcUnused (*C enumerator*), 225
drivers_RT_RT1020_MIMXRT1024._cmp_config (*C struct*), 249
drivers_RT_RT1020_MIMXRT1024._cmp_config.enableCmp (*C var*), 249
drivers_RT_RT1020_MIMXRT1024._cmp_config.enableHighSpeed (*C var*), 249
drivers_RT_RT1020_MIMXRT1024._cmp_config.enableInvertOutput (*C var*), 249
drivers_RT_RT1020_MIMXRT1024._cmp_config.enablePinOut (*C var*), 249
drivers_RT_RT1020_MIMXRT1024._cmp_config.enableTriggerMode (*C var*), 249
drivers_RT_RT1020_MIMXRT1024._cmp_config.hysteresisMode (*C var*), 249
drivers_RT_RT1020_MIMXRT1024._cmp_config.useUnfilteredOutput (*C var*), 249
drivers_RT_RT1020_MIMXRT1024._cmp_dac_config (*C struct*), 250
drivers_RT_RT1020_MIMXRT1024._cmp_dac_config.DACValue (*C var*), 250
drivers_RT_RT1020_MIMXRT1024._cmp_dac_config.referenceVoltageSource (*C var*), 250
drivers_RT_RT1020_MIMXRT1024._cmp_filter_config (*C struct*), 249
drivers_RT_RT1020_MIMXRT1024._cmp_filter_config.enableSample (*C var*), 249
drivers_RT_RT1020_MIMXRT1024._cmp_filter_config.filterCount (*C var*), 250
drivers_RT_RT1020_MIMXRT1024._cmp_filter_config.filterPeriod (*C var*), 250
drivers_RT_RT1020_MIMXRT1024._cmp_hysteresis_mode (*C enum*), 248
drivers_RT_RT1020_MIMXRT1024._cmp_hysteresis_mode.kCMP_HysteresisLevel0 (*C enumerator*), 248
drivers_RT_RT1020_MIMXRT1024._cmp_hysteresis_mode.kCMP_HysteresisLevel1 (*C enumerator*), 248
drivers_RT_RT1020_MIMXRT1024._cmp_hysteresis_mode.kCMP_HysteresisLevel2 (*C enumerator*), 248
drivers_RT_RT1020_MIMXRT1024._cmp_hysteresis_mode.kCMP_HysteresisLevel3 (*C enumerator*), 248
drivers_RT_RT1020_MIMXRT1024._cmp_interrupt_enable (*C enum*), 247
drivers_RT_RT1020_MIMXRT1024._cmp_interrupt_enable.kCMP_OutputFallingInterruptEnable (*C enumerator*), 248
drivers_RT_RT1020_MIMXRT1024._cmp_interrupt_enable.kCMP_OutputRisingInterruptEnable (*C enumerator*), 248
drivers_RT_RT1020_MIMXRT1024._cmp_reference_voltage_source (*C enum*), 248
drivers_RT_RT1020_MIMXRT1024._cmp_reference_voltage_source.kCMP_VrefSourceVin1 (*C enumerator*), 248
drivers_RT_RT1020_MIMXRT1024._cmp_reference_voltage_source.kCMP_VrefSourceVin2 (*C enumerator*), 248
drivers_RT_RT1020_MIMXRT1024._cmp_status_flags (*C enum*), 248
drivers_RT_RT1020_MIMXRT1024._cmp_status_flags.kCMP_OutputAssertEventFlag (*C enumerator*), 248
drivers_RT_RT1020_MIMXRT1024._cmp_status_flags.kCMP_OutputFallingEventFlag (*C enumerator*), 248
drivers_RT_RT1020_MIMXRT1024._cmp_status_flags.kCMP_OutputRisingEventFlag (*C enumerator*), 248
drivers_RT_RT1020_MIMXRT1024._dcdc_clock_source (*C enum*), 256
drivers_RT_RT1020_MIMXRT1024._dcdc_clock_source.kDCDC_ClockAutoSwitch (*C enumerator*), 256
drivers_RT_RT1020_MIMXRT1024._dcdc_clock_source.kDCDC_ClockExternalOsc (*C enumerator*), 256
drivers_RT_RT1020_MIMXRT1024._dcdc_clock_source.kDCDC_ClockInternalOsc (*C enumerator*), 256
drivers_RT_RT1020_MIMXRT1024._dcdc_comparator_current_bias (*C enum*), 254
drivers_RT_RT1020_MIMXRT1024._dcdc_comparator_current_bias.kDCDC_ComparatorCurrentBias50nA (*C enumerator*), 255
drivers_RT_RT1020_MIMXRT1024._dcdc_comparator_current_bias.kDCDC_ComparatorCurrentBias100nA (*C enumerator*), 255
drivers_RT_RT1020_MIMXRT1024._dcdc_comparator_current_bias.kDCDC_ComparatorCurrentBias200nA (*C enumerator*), 255
drivers_RT_RT1020_MIMXRT1024._dcdc_comparator_current_bias.kDCDC_ComparatorCurrentBias400nA (*C enumerator*), 255

drivers_RT_RT1020_MIMXRT1024._dcdc_config (*C struct*), 258
drivers_RT_RT1020_MIMXRT1024._dcdc_config.controlMode (*C var*), 258
drivers_RT_RT1020_MIMXRT1024._dcdc_config.enableDcdcTimeout (*C var*), 258
drivers_RT_RT1020_MIMXRT1024._dcdc_config.enableSwitchingConverterOutput (*C var*), 258
drivers_RT_RT1020_MIMXRT1024._dcdc_config.trimInputMode (*C var*), 258
drivers_RT_RT1020_MIMXRT1024._dcdc_control_mode (*C enum*), 257
drivers_RT_RT1020_MIMXRT1024._dcdc_control_mode.kDCDC_SetPointControl (*C enumerator*), 257
drivers_RT_RT1020_MIMXRT1024._dcdc_control_mode.kDCDC_StaticControl (*C enumerator*), 257
drivers_RT_RT1020_MIMXRT1024._dcdc_count_charging_time_period (*C enum*), 256
drivers_RT_RT1020_MIMXRT1024._dcdc_count_charging_time_period.kDCDC_CountChargingTimePeriod8Cycle (*C enumerator*), 256
drivers_RT_RT1020_MIMXRT1024._dcdc_count_charging_time_period.kDCDC_CountChargingTimePeriod16Cycl (*C enumerator*), 256
drivers_RT_RT1020_MIMXRT1024._dcdc_count_charging_time_threshold (*C enum*), 256
drivers_RT_RT1020_MIMXRT1024._dcdc_count_charging_time_threshold.kDCDC_CountChargingTimeThreshold8 (*C enumerator*), 256
drivers_RT_RT1020_MIMXRT1024._dcdc_count_charging_time_threshold.kDCDC_CountChargingTimeThreshold1 (*C enumerator*), 256
drivers_RT_RT1020_MIMXRT1024._dcdc_count_charging_time_threshold.kDCDC_CountChargingTimeThreshold3 (*C enumerator*), 256
drivers_RT_RT1020_MIMXRT1024._dcdc_count_charging_time_threshold.kDCDC_CountChargingTimeThreshold6 (*C enumerator*), 256
drivers_RT_RT1020_MIMXRT1024._dcdc_detection_config (*C struct*), 258
drivers_RT_RT1020_MIMXRT1024._dcdc_detection_config.enableXtalokDetection (*C var*), 259
drivers_RT_RT1020_MIMXRT1024._dcdc_detection_config.OverCurrentThreshold (*C var*), 259
drivers_RT_RT1020_MIMXRT1024._dcdc_detection_config.PeakCurrentThreshold (*C var*), 259
drivers_RT_RT1020_MIMXRT1024._dcdc_detection_config.powerDownLowVlotageDetection (*C var*), 259
drivers_RT_RT1020_MIMXRT1024._dcdc_detection_config.powerDownOverCurrentDetection (*C var*), 259
drivers_RT_RT1020_MIMXRT1024._dcdc_detection_config.powerDownOverVoltageDetection (*C var*), 259
drivers_RT_RT1020_MIMXRT1024._dcdc_detection_config.powerDownPeakCurrentDetection (*C var*), 259
drivers_RT_RT1020_MIMXRT1024._dcdc_detection_config.powerDownZeroCrossDetection (*C var*), 259
drivers_RT_RT1020_MIMXRT1024._dcdc_internal_regulator_config (*C struct*), 260
drivers_RT_RT1020_MIMXRT1024._dcdc_internal_regulator_config.enableLoadResistor (*C var*), 260
drivers_RT_RT1020_MIMXRT1024._dcdc_internal_regulator_config.feedbackPoint (*C var*), 260
drivers_RT_RT1020_MIMXRT1024._dcdc_loop_control_config (*C struct*), 259
drivers_RT_RT1020_MIMXRT1024._dcdc_loop_control_config.complementFeedForwardStep (*C var*), 260
drivers_RT_RT1020_MIMXRT1024._dcdc_loop_control_config.enableCommonHysteresis (*C var*), 259
drivers_RT_RT1020_MIMXRT1024._dcdc_loop_control_config.enableCommonThresholdDetection (*C var*), 259
drivers_RT_RT1020_MIMXRT1024._dcdc_loop_control_config.enableDifferentialHysteresis (*C var*), 259
drivers_RT_RT1020_MIMXRT1024._dcdc_loop_control_config.enableDifferentialThresholdDetection (*C var*), 259
drivers_RT_RT1020_MIMXRT1024._dcdc_loop_control_config.enableInvertHysteresisSign (*C var*), 259
drivers_RT_RT1020_MIMXRT1024._dcdc_loop_control_config.enableRCScaleCircuit (*C var*), 260
drivers_RT_RT1020_MIMXRT1024._dcdc_loop_control_config.enableRCThresholdDetection (*C var*), 260

drivers_RT_RT1020_MIMXRT1024._dcdc_low_power_config (*C struct*), 260
drivers_RT_RT1020_MIMXRT1024._dcdc_low_power_config.countChargingTimePeriod (*C var*), 260
drivers_RT_RT1020_MIMXRT1024._dcdc_low_power_config.countChargingTimeThreshold (*C var*), 260
drivers_RT_RT1020_MIMXRT1024._dcdc_low_power_config.enableAdjustHystereticValue (*C var*), 260
drivers_RT_RT1020_MIMXRT1024._dcdc_low_power_mode (*C enum*), 256
drivers_RT_RT1020_MIMXRT1024._dcdc_low_power_mode.kDCDC_GpcStandbyLowPowerMode (*C enumerator*), 257
drivers_RT_RT1020_MIMXRT1024._dcdc_low_power_mode.kDCDC_LowPowerMode (*C enumerator*), 257
drivers_RT_RT1020_MIMXRT1024._dcdc_low_power_mode.kDCDC_StandbyMode (*C enumerator*), 256
drivers_RT_RT1020_MIMXRT1024._dcdc_min_power_config (*C struct*), 260
drivers_RT_RT1020_MIMXRT1024._dcdc_min_power_config.enableUseHalfFreqForContinuous (*C var*), 261
drivers_RT_RT1020_MIMXRT1024._dcdc_over_current_threshold (*C enum*), 255
drivers_RT_RT1020_MIMXRT1024._dcdc_over_current_threshold.kDCDC_OverCurrentThresholdAlt0 (*C enumerator*), 255
drivers_RT_RT1020_MIMXRT1024._dcdc_over_current_threshold.kDCDC_OverCurrentThresholdAlt1 (*C enumerator*), 255
drivers_RT_RT1020_MIMXRT1024._dcdc_over_current_threshold.kDCDC_OverCurrentThresholdAlt2 (*C enumerator*), 255
drivers_RT_RT1020_MIMXRT1024._dcdc_over_current_threshold.kDCDC_OverCurrentThresholdAlt3 (*C enumerator*), 255
drivers_RT_RT1020_MIMXRT1024._dcdc_peak_current_threshold (*C enum*), 255
drivers_RT_RT1020_MIMXRT1024._dcdc_peak_current_threshold.kDCDC_PeakCurrentThresholdAlt0 (*C enumerator*), 255
drivers_RT_RT1020_MIMXRT1024._dcdc_peak_current_threshold.kDCDC_PeakCurrentThresholdAlt1 (*C enumerator*), 255
drivers_RT_RT1020_MIMXRT1024._dcdc_peak_current_threshold.kDCDC_PeakCurrentThresholdAlt2 (*C enumerator*), 255
drivers_RT_RT1020_MIMXRT1024._dcdc_peak_current_threshold.kDCDC_PeakCurrentThresholdAlt3 (*C enumerator*), 255
drivers_RT_RT1020_MIMXRT1024._dcdc_peak_current_threshold.kDCDC_PeakCurrentThresholdAlt4 (*C enumerator*), 255
drivers_RT_RT1020_MIMXRT1024._dcdc_peak_current_threshold.kDCDC_PeakCurrentThresholdAlt5 (*C enumerator*), 256
drivers_RT_RT1020_MIMXRT1024._dcdc_status_flags_t (*C enum*), 254
drivers_RT_RT1020_MIMXRT1024._dcdc_status_flags_t.kDCDC_LockedOKStatus (*C enumerator*), 254
drivers_RT_RT1020_MIMXRT1024._dcdc_trim_input_mode (*C enum*), 257
drivers_RT_RT1020_MIMXRT1024._dcdc_trim_input_mode.kDCDC_HoldTrimInput (*C enumerator*), 257
drivers_RT_RT1020_MIMXRT1024._dcdc_trim_input_mode.kDCDC_SampleTrimInput (*C enumerator*), 257
drivers_RT_RT1020_MIMXRT1024._dcp_ch_enable (*C enum*), 262
drivers_RT_RT1020_MIMXRT1024._dcp_ch_enable_t (*C type*), 264
drivers_RT_RT1020_MIMXRT1024._dcp_ch_enable.kDCP_ch0Enable (*C enumerator*), 262
drivers_RT_RT1020_MIMXRT1024._dcp_ch_enable.kDCP_ch1Enable (*C enumerator*), 262
drivers_RT_RT1020_MIMXRT1024._dcp_ch_enable.kDCP_ch2Enable (*C enumerator*), 262
drivers_RT_RT1020_MIMXRT1024._dcp_ch_enable.kDCP_ch3Enable (*C enumerator*), 262
drivers_RT_RT1020_MIMXRT1024._dcp_ch_enable.kDCP_chDisable (*C enumerator*), 262
drivers_RT_RT1020_MIMXRT1024._dcp_ch_enable.kDCP_chEnableAll (*C enumerator*), 262
drivers_RT_RT1020_MIMXRT1024._dcp_ch_int_enable (*C enum*), 262
drivers_RT_RT1020_MIMXRT1024._dcp_ch_int_enable_t (*C type*), 264
drivers_RT_RT1020_MIMXRT1024._dcp_ch_int_enable.kDCP_ch0IntEnable (*C enumerator*), 262

drivers_RT_RT1020_MIMXRT1024._dcp_ch_int_enable.kDCP_ch1IntEnable (*C enumerator*), 262
drivers_RT_RT1020_MIMXRT1024._dcp_ch_int_enable.kDCP_ch2IntEnable (*C enumerator*), 262
drivers_RT_RT1020_MIMXRT1024._dcp_ch_int_enable.kDCP_ch3IntEnable (*C enumerator*), 262
drivers_RT_RT1020_MIMXRT1024._dcp_ch_int_enable.kDCP_chIntDisable (*C enumerator*), 262
drivers_RT_RT1020_MIMXRT1024._dcp_channel (*C enum*), 262
drivers_RT_RT1020_MIMXRT1024._dcp_channel.kDCP_Channel0 (*C enumerator*), 263
drivers_RT_RT1020_MIMXRT1024._dcp_channel.kDCP_Channel1 (*C enumerator*), 263
drivers_RT_RT1020_MIMXRT1024._dcp_channel.kDCP_Channel2 (*C enumerator*), 263
drivers_RT_RT1020_MIMXRT1024._dcp_channel.kDCP_Channel3 (*C enumerator*), 263
drivers_RT_RT1020_MIMXRT1024._dcp_config (*C struct*), 265
drivers_RT_RT1020_MIMXRT1024._dcp_config.enableChannel (*C var*), 266
drivers_RT_RT1020_MIMXRT1024._dcp_config.enableChannelInterrupt (*C var*), 266
drivers_RT_RT1020_MIMXRT1024._dcp_config.enableContextCaching (*C var*), 266
drivers_RT_RT1020_MIMXRT1024._dcp_config.enableContextSwitching (*C var*), 266
drivers_RT_RT1020_MIMXRT1024._dcp_config.gatherResidualWrites (*C var*), 266
drivers_RT_RT1020_MIMXRT1024._dcp_context (*C struct*), 265
drivers_RT_RT1020_MIMXRT1024._dcp_handle (*C struct*), 265
drivers_RT_RT1020_MIMXRT1024._dcp_handle.channel (*C var*), 265
drivers_RT_RT1020_MIMXRT1024._dcp_handle.keySlot (*C var*), 265
drivers_RT_RT1020_MIMXRT1024._dcp_handle.swapConfig (*C var*), 265
drivers_RT_RT1020_MIMXRT1024._dcp_hash_algo_t (*C enum*), 268
drivers_RT_RT1020_MIMXRT1024._dcp_hash_algo_t.kDCP_Crc32 (*C enumerator*), 268
drivers_RT_RT1020_MIMXRT1024._dcp_hash_algo_t.kDCP_Sha1 (*C enumerator*), 268
drivers_RT_RT1020_MIMXRT1024._dcp_hash_algo_t.kDCP_Sha256 (*C enumerator*), 268
drivers_RT_RT1020_MIMXRT1024._dcp_hash_ctx_t (*C struct*), 270
drivers_RT_RT1020_MIMXRT1024._dcp_key_slot (*C enum*), 263
drivers_RT_RT1020_MIMXRT1024._dcp_key_slot.kDCP_KeySlot0 (*C enumerator*), 263
drivers_RT_RT1020_MIMXRT1024._dcp_key_slot.kDCP_KeySlot1 (*C enumerator*), 263
drivers_RT_RT1020_MIMXRT1024._dcp_key_slot.kDCP_KeySlot2 (*C enumerator*), 263
drivers_RT_RT1020_MIMXRT1024._dcp_key_slot.kDCP_KeySlot3 (*C enumerator*), 263
drivers_RT_RT1020_MIMXRT1024._dcp_key_slot.kDCP_OtpKey (*C enumerator*), 263
drivers_RT_RT1020_MIMXRT1024._dcp_key_slot.kDCP_OtpUniqueKey (*C enumerator*), 263
drivers_RT_RT1020_MIMXRT1024._dcp_key_slot.kDCP_PayloadKey (*C enumerator*), 263
drivers_RT_RT1020_MIMXRT1024._dcp_status (*C enum*), 261
drivers_RT_RT1020_MIMXRT1024._dcp_status.kStatus_DCP_Again (*C enumerator*), 262
drivers_RT_RT1020_MIMXRT1024._dcp_swap (*C enum*), 263
drivers_RT_RT1020_MIMXRT1024._dcp_swap.kDCP_InputByteSwap (*C enumerator*), 264
drivers_RT_RT1020_MIMXRT1024._dcp_swap.kDCP_InputWordSwap (*C enumerator*), 264
drivers_RT_RT1020_MIMXRT1024._dcp_swap.kDCP_KeyByteSwap (*C enumerator*), 263
drivers_RT_RT1020_MIMXRT1024._dcp_swap.kDCP_KeyWordSwap (*C enumerator*), 264
drivers_RT_RT1020_MIMXRT1024._dcp_swap.kDCP_NoSwap (*C enumerator*), 263
drivers_RT_RT1020_MIMXRT1024._dcp_swap.kDCP_OutputByteSwap (*C enumerator*), 264
drivers_RT_RT1020_MIMXRT1024._dcp_swap.kDCP_OutputWordSwap (*C enumerator*), 264
drivers_RT_RT1020_MIMXRT1024._dcp_work_packet (*C struct*), 265
drivers_RT_RT1020_MIMXRT1024._edma_bandwidth (*C enum*), 288
drivers_RT_RT1020_MIMXRT1024._edma_bandwidth.kEDMA_BandwidthStall4Cycle (*C enumerator*), 288
drivers_RT_RT1020_MIMXRT1024._edma_bandwidth.kEDMA_BandwidthStall8Cycle (*C enumerator*), 288
drivers_RT_RT1020_MIMXRT1024._edma_bandwidth.kEDMA_BandwidthStallNone (*C enumerator*), 288
drivers_RT_RT1020_MIMXRT1024._edma_channel_link_type (*C enum*), 288
drivers_RT_RT1020_MIMXRT1024._edma_channel_link_type.kEDMA_LinkNone (*C enumerator*), 288
drivers_RT_RT1020_MIMXRT1024._edma_channel_link_type.kEDMA_MajorLink (*C enumerator*), 288
drivers_RT_RT1020_MIMXRT1024._edma_channel_link_type.kEDMA_MinorLink (*C enumerator*), 288

drivers_RT_RT1020_MIMXRT1024._edma_channel_Preemption_config (*C struct*), 292
drivers_RT_RT1020_MIMXRT1024._edma_channel_Preemption_config.channelPriority (*C var*), 293
drivers_RT_RT1020_MIMXRT1024._edma_channel_Preemption_config.enableChannelPreemption (*C var*), 293
drivers_RT_RT1020_MIMXRT1024._edma_channel_Preemption_config.enablePreemptAbility (*C var*), 293
drivers_RT_RT1020_MIMXRT1024._edma_config (*C struct*), 291
drivers_RT_RT1020_MIMXRT1024._edma_config.enableContinuousLinkMode (*C var*), 291
drivers_RT_RT1020_MIMXRT1024._edma_config.enableDebugMode (*C var*), 292
drivers_RT_RT1020_MIMXRT1024._edma_config.enableHaltOnError (*C var*), 291
drivers_RT_RT1020_MIMXRT1024._edma_config.enableRoundRobinArbitration (*C var*), 292
drivers_RT_RT1020_MIMXRT1024._edma_handle (*C struct*), 294
drivers_RT_RT1020_MIMXRT1024._edma_handle.base (*C var*), 294
drivers_RT_RT1020_MIMXRT1024._edma_handle.callback (*C var*), 294
drivers_RT_RT1020_MIMXRT1024._edma_handle.channel (*C var*), 294
drivers_RT_RT1020_MIMXRT1024._edma_handle.flags (*C var*), 295
drivers_RT_RT1020_MIMXRT1024._edma_handle.header (*C var*), 294
drivers_RT_RT1020_MIMXRT1024._edma_handle.tail (*C var*), 294
drivers_RT_RT1020_MIMXRT1024._edma_handle.tcdPool (*C var*), 294
drivers_RT_RT1020_MIMXRT1024._edma_handle.tcdSize (*C var*), 294
drivers_RT_RT1020_MIMXRT1024._edma_handle.tcdUsed (*C var*), 294
drivers_RT_RT1020_MIMXRT1024._edma_handle.userData (*C var*), 294
drivers_RT_RT1020_MIMXRT1024._edma_interrupt_enable (*C enum*), 289
drivers_RT_RT1020_MIMXRT1024._edma_interrupt_enable.kEDMA_ErrorInterruptEnable (*C enumerator*), 289
drivers_RT_RT1020_MIMXRT1024._edma_interrupt_enable.kEDMA_HalfInterruptEnable (*C enumerator*), 289
drivers_RT_RT1020_MIMXRT1024._edma_interrupt_enable.kEDMA_MajorInterruptEnable (*C enumerator*), 289
drivers_RT_RT1020_MIMXRT1024._edma_minor_offset_config (*C struct*), 293
drivers_RT_RT1020_MIMXRT1024._edma_minor_offset_config.enableDestMinorOffset (*C var*), 293
drivers_RT_RT1020_MIMXRT1024._edma_minor_offset_config.enableSrcMinorOffset (*C var*), 293
drivers_RT_RT1020_MIMXRT1024._edma_minor_offset_config.minorOffset (*C var*), 293
drivers_RT_RT1020_MIMXRT1024._edma_modulo (*C enum*), 285
drivers_RT_RT1020_MIMXRT1024._edma_modulo.kEDMA_Modulo1Gbytes (*C enumerator*), 287
drivers_RT_RT1020_MIMXRT1024._edma_modulo.kEDMA_Modulo1Kbytes (*C enumerator*), 286
drivers_RT_RT1020_MIMXRT1024._edma_modulo.kEDMA_Modulo1Mbytes (*C enumerator*), 287
drivers_RT_RT1020_MIMXRT1024._edma_modulo.kEDMA_Modulo2bytes (*C enumerator*), 286
drivers_RT_RT1020_MIMXRT1024._edma_modulo.kEDMA_Modulo2Gbytes (*C enumerator*), 287
drivers_RT_RT1020_MIMXRT1024._edma_modulo.kEDMA_Modulo2Kbytes (*C enumerator*), 286
drivers_RT_RT1020_MIMXRT1024._edma_modulo.kEDMA_Modulo2Mbytes (*C enumerator*), 287
drivers_RT_RT1020_MIMXRT1024._edma_modulo.kEDMA_Modulo4bytes (*C enumerator*), 286
drivers_RT_RT1020_MIMXRT1024._edma_modulo.kEDMA_Modulo4Kbytes (*C enumerator*), 286
drivers_RT_RT1020_MIMXRT1024._edma_modulo.kEDMA_Modulo4Mbytes (*C enumerator*), 287
drivers_RT_RT1020_MIMXRT1024._edma_modulo.kEDMA_Modulo8bytes (*C enumerator*), 286
drivers_RT_RT1020_MIMXRT1024._edma_modulo.kEDMA_Modulo8Kbytes (*C enumerator*), 286
drivers_RT_RT1020_MIMXRT1024._edma_modulo.kEDMA_Modulo8Mbytes (*C enumerator*), 287
drivers_RT_RT1020_MIMXRT1024._edma_modulo.kEDMA_Modulo16bytes (*C enumerator*), 286
drivers_RT_RT1020_MIMXRT1024._edma_modulo.kEDMA_Modulo16Kbytes (*C enumerator*), 286
drivers_RT_RT1020_MIMXRT1024._edma_modulo.kEDMA_Modulo16Mbytes (*C enumerator*), 287
drivers_RT_RT1020_MIMXRT1024._edma_modulo.kEDMA_Modulo32bytes (*C enumerator*), 286
drivers_RT_RT1020_MIMXRT1024._edma_modulo.kEDMA_Modulo32Kbytes (*C enumerator*), 286
drivers_RT_RT1020_MIMXRT1024._edma_modulo.kEDMA_Modulo32Mbytes (*C enumerator*), 287
drivers_RT_RT1020_MIMXRT1024._edma_modulo.kEDMA_Modulo64bytes (*C enumerator*), 286
drivers_RT_RT1020_MIMXRT1024._edma_modulo.kEDMA_Modulo64Kbytes (*C enumerator*), 287
drivers_RT_RT1020_MIMXRT1024._edma_modulo.kEDMA_Modulo64Mbytes (*C enumerator*), 287
drivers_RT_RT1020_MIMXRT1024._edma_modulo.kEDMA_Modulo128bytes (*C enumerator*), 286

drivers_RT_RT1020_MIMXRT1024._edma_modulo.kEDMA_Modulo128Kbytes (*C enumerator*), 287
drivers_RT_RT1020_MIMXRT1024._edma_modulo.kEDMA_Modulo128Mbytes (*C enumerator*), 287
drivers_RT_RT1020_MIMXRT1024._edma_modulo.kEDMA_Modulo256bytes (*C enumerator*), 286
drivers_RT_RT1020_MIMXRT1024._edma_modulo.kEDMA_Modulo256Kbytes (*C enumerator*), 287
drivers_RT_RT1020_MIMXRT1024._edma_modulo.kEDMA_Modulo256Mbytes (*C enumerator*), 287
drivers_RT_RT1020_MIMXRT1024._edma_modulo.kEDMA_Modulo512bytes (*C enumerator*), 286
drivers_RT_RT1020_MIMXRT1024._edma_modulo.kEDMA_Modulo512Kbytes (*C enumerator*), 287
drivers_RT_RT1020_MIMXRT1024._edma_modulo.kEDMA_Modulo512Mbytes (*C enumerator*), 287
drivers_RT_RT1020_MIMXRT1024._edma_modulo.kEDMA_ModuloDisable (*C enumerator*), 286
drivers_RT_RT1020_MIMXRT1024._edma_tcd (*C struct*), 293
drivers_RT_RT1020_MIMXRT1024._edma_transfer_config (*C struct*), 292
drivers_RT_RT1020_MIMXRT1024._edma_transfer_config.destAddr (*C var*), 292
drivers_RT_RT1020_MIMXRT1024._edma_transfer_config.destOffset (*C var*), 292
drivers_RT_RT1020_MIMXRT1024._edma_transfer_config.destTransferSize (*C var*), 292
drivers_RT_RT1020_MIMXRT1024._edma_transfer_config.majorLoopCounts (*C var*), 292
drivers_RT_RT1020_MIMXRT1024._edma_transfer_config.minorLoopBytes (*C var*), 292
drivers_RT_RT1020_MIMXRT1024._edma_transfer_config.srcAddr (*C var*), 292
drivers_RT_RT1020_MIMXRT1024._edma_transfer_config.srcOffset (*C var*), 292
drivers_RT_RT1020_MIMXRT1024._edma_transfer_config.srcTransferSize (*C var*), 292
drivers_RT_RT1020_MIMXRT1024._edma_transfer_size (*C enum*), 285
drivers_RT_RT1020_MIMXRT1024._edma_transfer_size.kEDMA_TransferSize1Bytes (*C enumerator*), 285
drivers_RT_RT1020_MIMXRT1024._edma_transfer_size.kEDMA_TransferSize2Bytes (*C enumerator*), 285
drivers_RT_RT1020_MIMXRT1024._edma_transfer_size.kEDMA_TransferSize4Bytes (*C enumerator*), 285
drivers_RT_RT1020_MIMXRT1024._edma_transfer_size.kEDMA_TransferSize8Bytes (*C enumerator*), 285
drivers_RT_RT1020_MIMXRT1024._edma_transfer_size.kEDMA_TransferSize16Bytes (*C enumerator*), 285
drivers_RT_RT1020_MIMXRT1024._edma_transfer_size.kEDMA_TransferSize32Bytes (*C enumerator*), 285
drivers_RT_RT1020_MIMXRT1024._edma_transfer_type (*C enum*), 289
drivers_RT_RT1020_MIMXRT1024._edma_transfer_type.kEDMA_MemoryToMemory (*C enumerator*), 289
drivers_RT_RT1020_MIMXRT1024._edma_transfer_type.kEDMA_MemoryToPeripheral (*C enumerator*), 290
drivers_RT_RT1020_MIMXRT1024._edma_transfer_type.kEDMA_PeripheralToMemory (*C enumerator*), 290
drivers_RT_RT1020_MIMXRT1024._edma_transfer_type.kEDMA_PeripheralToPeripheral (*C enumerator*), 290
drivers_RT_RT1020_MIMXRT1024._enc_config (*C struct*), 305
drivers_RT_RT1020_MIMXRT1024._enc_config.decoderWorkMode (*C var*), 305
drivers_RT_RT1020_MIMXRT1024._enc_config.enableModuloCountMode (*C var*), 306
drivers_RT_RT1020_MIMXRT1024._enc_config.enablePeriodMeasurementFunction (*C var*), 306
drivers_RT_RT1020_MIMXRT1024._enc_config.enableReverseDirection (*C var*), 305
drivers_RT_RT1020_MIMXRT1024._enc_config.enableTRIGGERClearHoldPositionCounter (*C var*), 305
drivers_RT_RT1020_MIMXRT1024._enc_config.enableTRIGGERClearPositionCounter (*C var*), 305
drivers_RT_RT1020_MIMXRT1024._enc_config.enableWatchdog (*C var*), 305
drivers_RT_RT1020_MIMXRT1024._enc_config.filterCount (*C var*), 305
drivers_RT_RT1020_MIMXRT1024._enc_config.filterPrescaler (*C var*), 305

drivers_RT_RT1020_MIMXRT1024._enc_config.filterSamplePeriod (*C var*), 305
drivers_RT_RT1020_MIMXRT1024._enc_config.HOMETriggerMode (*C var*), 305
drivers_RT_RT1020_MIMXRT1024._enc_config.INDEXTriggerMode (*C var*), 305
drivers_RT_RT1020_MIMXRT1024._enc_config.positionCompareValue (*C var*), 306
drivers_RT_RT1020_MIMXRT1024._enc_config.positionInitialValue (*C var*), 306
drivers_RT_RT1020_MIMXRT1024._enc_config.positionMatchMode (*C var*), 306
drivers_RT_RT1020_MIMXRT1024._enc_config.positionModulusValue (*C var*), 306
drivers_RT_RT1020_MIMXRT1024._enc_config.prescalerValue (*C var*), 306
drivers_RT_RT1020_MIMXRT1024._enc_config.revolutionCountCondition (*C var*), 306
drivers_RT_RT1020_MIMXRT1024._enc_config.watchdogTimeoutValue (*C var*), 305
drivers_RT_RT1020_MIMXRT1024._enc_decoder_work_mode (*C enum*), 301
drivers_RT_RT1020_MIMXRT1024._enc_decoder_work_mode.kENC_DecoderWorkAsNormalMode (*C enumerator*), 301
drivers_RT_RT1020_MIMXRT1024._enc_decoder_work_mode.kENC_DecoderWorkAsSignalPhaseCountMode (*C enumerator*), 301
drivers_RT_RT1020_MIMXRT1024._enc_filter_prescaler (*C enum*), 303
drivers_RT_RT1020_MIMXRT1024._enc_filter_prescaler.kENC_FilterPrescalerDiv1 (*C enumerator*), 303
drivers_RT_RT1020_MIMXRT1024._enc_filter_prescaler.kENC_FilterPrescalerDiv2 (*C enumerator*), 303
drivers_RT_RT1020_MIMXRT1024._enc_filter_prescaler.kENC_FilterPrescalerDiv4 (*C enumerator*), 303
drivers_RT_RT1020_MIMXRT1024._enc_filter_prescaler.kENC_FilterPrescalerDiv8 (*C enumerator*), 303
drivers_RT_RT1020_MIMXRT1024._enc_filter_prescaler.kENC_FilterPrescalerDiv16 (*C enumerator*), 303
drivers_RT_RT1020_MIMXRT1024._enc_filter_prescaler.kENC_FilterPrescalerDiv32 (*C enumerator*), 303
drivers_RT_RT1020_MIMXRT1024._enc_filter_prescaler.kENC_FilterPrescalerDiv64 (*C enumerator*), 303
drivers_RT_RT1020_MIMXRT1024._enc_filter_prescaler.kENC_FilterPrescalerDiv128 (*C enumerator*), 304
drivers_RT_RT1020_MIMXRT1024._enc_home_trigger_mode (*C enum*), 301
drivers_RT_RT1020_MIMXRT1024._enc_home_trigger_mode.kENC_HOMETriggerDisabled (*C enumerator*), 301
drivers_RT_RT1020_MIMXRT1024._enc_home_trigger_mode.kENC_HOMETriggerOnFallingEdge (*C enumerator*), 301
drivers_RT_RT1020_MIMXRT1024._enc_home_trigger_mode.kENC_HOMETriggerOnRisingEdge (*C enumerator*), 301
drivers_RT_RT1020_MIMXRT1024._enc_index_trigger_mode (*C enum*), 301
drivers_RT_RT1020_MIMXRT1024._enc_index_trigger_mode.kENC_INDEXTriggerDisabled (*C enumerator*), 301
drivers_RT_RT1020_MIMXRT1024._enc_index_trigger_mode.kENC_INDEXTriggerOnFallingEdge (*C enumerator*), 301
drivers_RT_RT1020_MIMXRT1024._enc_index_trigger_mode.kENC_INDEXTriggerOnRisingEdge (*C enumerator*), 301
drivers_RT_RT1020_MIMXRT1024._enc_interrupt_enable (*C enum*), 299
drivers_RT_RT1020_MIMXRT1024._enc_interrupt_enable.kENC_HOMETransitionInterruptEnable (*C enumerator*), 299
drivers_RT_RT1020_MIMXRT1024._enc_interrupt_enable.kENC_INDEXPulseInterruptEnable (*C enumerator*), 299
drivers_RT_RT1020_MIMXRT1024._enc_interrupt_enable.kENC_PositionCompareInerruptEnable (*C enumerator*), 299
drivers_RT_RT1020_MIMXRT1024._enc_interrupt_enable.kENC_PositionRollOverInterruptEnable (*C enumerator*), 299
drivers_RT_RT1020_MIMXRT1024._enc_interrupt_enable.kENC_PositionRollUnderInterruptEnable (*C enumerator*), 299
drivers_RT_RT1020_MIMXRT1024._enc_interrupt_enable.kENC_WatchdogTimeoutInterruptEnable

(*C enumerator*), 299
drivers_RT_RT1020_MIMXRT1024._enc_position_match_mode (*C enum*), 301
drivers_RT_RT1020_MIMXRT1024._enc_position_match_mode.kENC_POSMATCHOnPositionCounterEqualToCo
(*C enumerator*), 302
drivers_RT_RT1020_MIMXRT1024._enc_position_match_mode.kENC_POSMATCHOnReadingAnyPositionCounte
(*C enumerator*), 302
drivers_RT_RT1020_MIMXRT1024._enc_prescaler (*C enum*), 302
drivers_RT_RT1020_MIMXRT1024._enc_prescaler.kENC_ClockDiv1 (*C enumerator*), 302
drivers_RT_RT1020_MIMXRT1024._enc_prescaler.kENC_ClockDiv2 (*C enumerator*), 302
drivers_RT_RT1020_MIMXRT1024._enc_prescaler.kENC_ClockDiv4 (*C enumerator*), 302
drivers_RT_RT1020_MIMXRT1024._enc_prescaler.kENC_ClockDiv8 (*C enumerator*), 302
drivers_RT_RT1020_MIMXRT1024._enc_prescaler.kENC_ClockDiv16 (*C enumerator*), 302
drivers_RT_RT1020_MIMXRT1024._enc_prescaler.kENC_ClockDiv32 (*C enumerator*), 302
drivers_RT_RT1020_MIMXRT1024._enc_prescaler.kENC_ClockDiv64 (*C enumerator*), 303
drivers_RT_RT1020_MIMXRT1024._enc_prescaler.kENC_ClockDiv128 (*C enumerator*), 303
drivers_RT_RT1020_MIMXRT1024._enc_prescaler.kENC_ClockDiv256 (*C enumerator*), 303
drivers_RT_RT1020_MIMXRT1024._enc_prescaler.kENC_ClockDiv512 (*C enumerator*), 303
drivers_RT_RT1020_MIMXRT1024._enc_prescaler.kENC_ClockDiv1024 (*C enumerator*), 303
drivers_RT_RT1020_MIMXRT1024._enc_prescaler.kENC_ClockDiv2048 (*C enumerator*), 303
drivers_RT_RT1020_MIMXRT1024._enc_prescaler.kENC_ClockDiv4096 (*C enumerator*), 303
drivers_RT_RT1020_MIMXRT1024._enc_prescaler.kENC_ClockDiv8192 (*C enumerator*), 303
drivers_RT_RT1020_MIMXRT1024._enc_prescaler.kENC_ClockDiv16384 (*C enumerator*), 303
drivers_RT_RT1020_MIMXRT1024._enc_prescaler.kENC_ClockDiv32768 (*C enumerator*), 303
drivers_RT_RT1020_MIMXRT1024._enc_revolution_count_condition (*C enum*), 302
drivers_RT_RT1020_MIMXRT1024._enc_revolution_count_condition.kENC_RevolutionCountOnINDEXPulse
(*C enumerator*), 302
drivers_RT_RT1020_MIMXRT1024._enc_revolution_count_condition.kENC_RevolutionCountOnRollOverModulus
(*C enumerator*), 302
drivers_RT_RT1020_MIMXRT1024._enc_self_test_config (*C struct*), 306
drivers_RT_RT1020_MIMXRT1024._enc_self_test_config.signalCount (*C var*), 306
drivers_RT_RT1020_MIMXRT1024._enc_self_test_config.signalDirection (*C var*), 306
drivers_RT_RT1020_MIMXRT1024._enc_self_test_config.signalPeriod (*C var*), 306
drivers_RT_RT1020_MIMXRT1024._enc_self_test_direction (*C enum*), 302
drivers_RT_RT1020_MIMXRT1024._enc_self_test_direction.kENC_SelfTestDirectionNegative (*C enumerator*), 302
drivers_RT_RT1020_MIMXRT1024._enc_self_test_direction.kENC_SelfTestDirectionPositive (*C enumerator*), 302
drivers_RT_RT1020_MIMXRT1024._enc_signal_status_flags (*C enum*), 300
drivers_RT_RT1020_MIMXRT1024._enc_signal_status_flags.kENC_FilteredHOMEStatusFlag (*C enumerator*), 300
drivers_RT_RT1020_MIMXRT1024._enc_signal_status_flags.kENC_FilteredINDEXStatusFlag (*C enumerator*), 300
drivers_RT_RT1020_MIMXRT1024._enc_signal_status_flags.kENC_FilteredPHAStatusFlag (*C enumerator*), 300
drivers_RT_RT1020_MIMXRT1024._enc_signal_status_flags.kENC_FilteredPHBStatusFlag (*C enumerator*), 300
drivers_RT_RT1020_MIMXRT1024._enc_signal_status_flags.kENC_RawHOMEStatusFlag (*C enumerator*), 300
drivers_RT_RT1020_MIMXRT1024._enc_signal_status_flags.kENC_RawINDEXStatusFlag (*C enumerator*), 300
drivers_RT_RT1020_MIMXRT1024._enc_signal_status_flags.kENC_RawPHAEXStatusFlag (*C enumerator*), 300
drivers_RT_RT1020_MIMXRT1024._enc_signal_status_flags.kENC_RawPHBStatusFlag (*C enumerator*), 300
drivers_RT_RT1020_MIMXRT1024._enc_status_flags (*C enum*), 299
drivers_RT_RT1020_MIMXRT1024._enc_status_flags.kENC_HOMETransitionFlag (*C enumerator*), 299
drivers_RT_RT1020_MIMXRT1024._enc_status_flags.kENC_INDEXPulseFlag (*C enumerator*),

300
drivers_RT_RT1020_MIMXRT1024._enc_status_flags.kENC_LastCountDirectionFlag (*C enumerator*), 300
drivers_RT_RT1020_MIMXRT1024._enc_status_flags.kENC_PositionCompareFlag (*C enumerator*), 300
drivers_RT_RT1020_MIMXRT1024._enc_status_flags.kENC_PositionRollOverFlag (*C enumerator*), 300
drivers_RT_RT1020_MIMXRT1024._enc_status_flags.kENC_PositionRollUnderFlag (*C enumerator*), 300
drivers_RT_RT1020_MIMXRT1024._enc_status_flags.kENC_WatchdogTimeoutFlag (*C enumerator*), 300
drivers_RT_RT1020_MIMXRT1024._enet_avb_config (*C struct*), 336
drivers_RT_RT1020_MIMXRT1024._enet_avb_config.idleSlope (*C var*), 337
drivers_RT_RT1020_MIMXRT1024._enet_avb_config.rxClassifyMatch (*C var*), 337
drivers_RT_RT1020_MIMXRT1024._enet_buffer_config (*C struct*), 335
drivers_RT_RT1020_MIMXRT1024._enet_buffer_config.rxBdNumber (*C var*), 335
drivers_RT_RT1020_MIMXRT1024._enet_buffer_config.rxBdStartAddrAlign (*C var*), 336
drivers_RT_RT1020_MIMXRT1024._enet_buffer_config.rxBufferAlign (*C var*), 336
drivers_RT_RT1020_MIMXRT1024._enet_buffer_config.rxBuffSizeAlign (*C var*), 335
drivers_RT_RT1020_MIMXRT1024._enet_buffer_config.rxMaintainEnable (*C var*), 336
drivers_RT_RT1020_MIMXRT1024._enet_buffer_config.txBdNumber (*C var*), 335
drivers_RT_RT1020_MIMXRT1024._enet_buffer_config.txBdStartAddrAlign (*C var*), 336
drivers_RT_RT1020_MIMXRT1024._enet_buffer_config.txBufferAlign (*C var*), 336
drivers_RT_RT1020_MIMXRT1024._enet_buffer_config.txBuffSizeAlign (*C var*), 335
drivers_RT_RT1020_MIMXRT1024._enet_buffer_config.txFrameInfo (*C var*), 336
drivers_RT_RT1020_MIMXRT1024._enet_buffer_config.txMaintainEnable (*C var*), 336
drivers_RT_RT1020_MIMXRT1024._enet_buffer_struct (*C struct*), 340
drivers_RT_RT1020_MIMXRT1024._enet_buffer_struct.buffer (*C var*), 341
drivers_RT_RT1020_MIMXRT1024._enet_buffer_struct.length (*C var*), 341
drivers_RT_RT1020_MIMXRT1024._enet_config (*C struct*), 337
drivers_RT_RT1020_MIMXRT1024._enet_config.callback (*C var*), 339
drivers_RT_RT1020_MIMXRT1024._enet_config.intCoalesceCfg (*C var*), 338
drivers_RT_RT1020_MIMXRT1024._enet_config.interrupt (*C var*), 337
drivers_RT_RT1020_MIMXRT1024._enet_config.macSpecialConfig (*C var*), 337
drivers_RT_RT1020_MIMXRT1024._enet_config.miiDuplex (*C var*), 338
drivers_RT_RT1020_MIMXRT1024._enet_config.miiMode (*C var*), 338
drivers_RT_RT1020_MIMXRT1024._enet_config.miiSpeed (*C var*), 338
drivers_RT_RT1020_MIMXRT1024._enet_config.pauseDuration (*C var*), 338
drivers_RT_RT1020_MIMXRT1024._enet_config.ringNum (*C var*), 338
drivers_RT_RT1020_MIMXRT1024._enet_config.rxAccelerConfig (*C var*), 338
drivers_RT_RT1020_MIMXRT1024._enet_config.rxBuffAlloc (*C var*), 338
drivers_RT_RT1020_MIMXRT1024._enet_config.rxBuffFree (*C var*), 338
drivers_RT_RT1020_MIMXRT1024._enet_config.rxFifoEmptyThreshold (*C var*), 338
drivers_RT_RT1020_MIMXRT1024._enet_config.rxFifoFullThreshold (*C var*), 338
drivers_RT_RT1020_MIMXRT1024._enet_config.rxFifoStatEmptyThreshold (*C var*), 338
drivers_RT_RT1020_MIMXRT1024._enet_config.rxMaxFrameLen (*C var*), 338
drivers_RT_RT1020_MIMXRT1024._enet_config.txAccelerConfig (*C var*), 338
drivers_RT_RT1020_MIMXRT1024._enet_config.txFifoWatermark (*C var*), 338
drivers_RT_RT1020_MIMXRT1024._enet_config.userData (*C var*), 339
drivers_RT_RT1020_MIMXRT1024._enet_data_error_stats (*C struct*), 333
drivers_RT_RT1020_MIMXRT1024._enet_data_error_stats.statsRxAlignErr (*C var*), 333
drivers_RT_RT1020_MIMXRT1024._enet_data_error_stats.statsRxFcsErr (*C var*), 333
drivers_RT_RT1020_MIMXRT1024._enet_data_error_stats.statsRxLenGreaterErr (*C var*), 333
drivers_RT_RT1020_MIMXRT1024._enet_data_error_stats.statsRxOverRunErr (*C var*), 333
drivers_RT_RT1020_MIMXRT1024._enet_data_error_stats.statsRxTruncateErr (*C var*), 333
drivers_RT_RT1020_MIMXRT1024._enet_event (*C enum*), 326
drivers_RT_RT1020_MIMXRT1024._enet_event.kENET_ErrEvent (*C enumerator*), 326
drivers_RT_RT1020_MIMXRT1024._enet_event.kENET_RxEvent (*C enumerator*), 326

drivers_RT_RT1020_MIMXRT1024._enet_event.kENET_TimeStampAvailEvent (*C enumerator*), 326
drivers_RT_RT1020_MIMXRT1024._enet_event.kENET_TimeStampEvent (*C enumerator*), 326
drivers_RT_RT1020_MIMXRT1024._enet_event.kENET_TxEvent (*C enumerator*), 326
drivers_RT_RT1020_MIMXRT1024._enet_event.kENET_WakeUpEvent (*C enumerator*), 326
drivers_RT_RT1020_MIMXRT1024._enet_handle (*C struct*), 339
drivers_RT_RT1020_MIMXRT1024._enet_handle.callback (*C var*), 340
drivers_RT_RT1020_MIMXRT1024._enet_handle.enetClock (*C var*), 340
drivers_RT_RT1020_MIMXRT1024._enet_handle.multicastCount (*C var*), 340
drivers_RT_RT1020_MIMXRT1024._enet_handle.ringNum (*C var*), 340
drivers_RT_RT1020_MIMXRT1024._enet_handle.rxBdRing (*C var*), 340
drivers_RT_RT1020_MIMXRT1024._enet_handle.rxBuffAlloc (*C var*), 340
drivers_RT_RT1020_MIMXRT1024._enet_handle.rxBuffFree (*C var*), 340
drivers_RT_RT1020_MIMXRT1024._enet_handle.rxBuffSizeAlign (*C var*), 340
drivers_RT_RT1020_MIMXRT1024._enet_handle.rxMaintainEnable (*C var*), 340
drivers_RT_RT1020_MIMXRT1024._enet_handle.tsDelayCount (*C var*), 340
drivers_RT_RT1020_MIMXRT1024._enet_handle.txBdRing (*C var*), 340
drivers_RT_RT1020_MIMXRT1024._enet_handle.txBuffSizeAlign (*C var*), 340
drivers_RT_RT1020_MIMXRT1024._enet_handle.txDirtyRing (*C var*), 340
drivers_RT_RT1020_MIMXRT1024._enet_handle.txMaintainEnable (*C var*), 340
drivers_RT_RT1020_MIMXRT1024._enet_handle.txReclaimEnable (*C var*), 340
drivers_RT_RT1020_MIMXRT1024._enet_handle.userData (*C var*), 340
drivers_RT_RT1020_MIMXRT1024._enet_idle_slope (*C enum*), 326
drivers_RT_RT1020_MIMXRT1024._enet_idle_slope.kENET_IdleSlope1 (*C enumerator*), 327
drivers_RT_RT1020_MIMXRT1024._enet_idle_slope.kENET_IdleSlope2 (*C enumerator*), 327
drivers_RT_RT1020_MIMXRT1024._enet_idle_slope.kENET_IdleSlope4 (*C enumerator*), 327
drivers_RT_RT1020_MIMXRT1024._enet_idle_slope.kENET_IdleSlope8 (*C enumerator*), 327
drivers_RT_RT1020_MIMXRT1024._enet_idle_slope.kENET_IdleSlope16 (*C enumerator*), 327
drivers_RT_RT1020_MIMXRT1024._enet_idle_slope.kENET_IdleSlope32 (*C enumerator*), 327
drivers_RT_RT1020_MIMXRT1024._enet_idle_slope.kENET_IdleSlope64 (*C enumerator*), 327
drivers_RT_RT1020_MIMXRT1024._enet_idle_slope.kENET_IdleSlope128 (*C enumerator*), 327
drivers_RT_RT1020_MIMXRT1024._enet_idle_slope.kENET_IdleSlope256 (*C enumerator*), 327
drivers_RT_RT1020_MIMXRT1024._enet_idle_slope.kENET_IdleSlope384 (*C enumerator*), 327
drivers_RT_RT1020_MIMXRT1024._enet_idle_slope.kENET_IdleSlope512 (*C enumerator*), 327
drivers_RT_RT1020_MIMXRT1024._enet_idle_slope.kENET_IdleSlope640 (*C enumerator*), 327
drivers_RT_RT1020_MIMXRT1024._enet_idle_slope.kENET_IdleSlope768 (*C enumerator*), 327
drivers_RT_RT1020_MIMXRT1024._enet_idle_slope.kENET_IdleSlope896 (*C enumerator*), 327
drivers_RT_RT1020_MIMXRT1024._enet_idle_slope.kENET_IdleSlope1024 (*C enumerator*), 327
drivers_RT_RT1020_MIMXRT1024._enet_idle_slope.kENET_IdleSlope1152 (*C enumerator*), 327
drivers_RT_RT1020_MIMXRT1024._enet_idle_slope.kENET_IdleSlope1280 (*C enumerator*), 328
drivers_RT_RT1020_MIMXRT1024._enet_idle_slope.kENET_IdleSlope1408 (*C enumerator*), 328
drivers_RT_RT1020_MIMXRT1024._enet_idle_slope.kENET_IdleSlope1536 (*C enumerator*), 328
drivers_RT_RT1020_MIMXRT1024._enet_intcoalesce_config (*C struct*), 336
drivers_RT_RT1020_MIMXRT1024._enet_intcoalesce_config.rxCoalesceFrameCount (*C var*), 336
drivers_RT_RT1020_MIMXRT1024._enet_intcoalesce_config.rxCoalesceTimeCount (*C var*), 336
drivers_RT_RT1020_MIMXRT1024._enet_intcoalesce_config.txCoalesceFrameCount (*C var*), 336
drivers_RT_RT1020_MIMXRT1024._enet_intcoalesce_config.txCoalesceTimeCount (*C var*), 336
drivers_RT_RT1020_MIMXRT1024._enet_interrupt_enable (*C enum*), 325
drivers_RT_RT1020_MIMXRT1024._enet_interrupt_enable.kENET_BabrInterrupt (*C enumerator*), 325
drivers_RT_RT1020_MIMXRT1024._enet_interrupt_enable.kENET_BabtInterrupt (*C enumerator*), 325
drivers_RT_RT1020_MIMXRT1024._enet_interrupt_enable.kENET_EBusERInterrupt (*C enumerator*), 326
drivers_RT_RT1020_MIMXRT1024._enet_interrupt_enable.kENET_GraceStopInterrupt (*C enumerator*), 325
drivers_RT_RT1020_MIMXRT1024._enet_interrupt_enable.kENET_LateCollisionInterrupt (*C enumerator*), 326

drivers_RT_RT1020_MIMXRT1024._enet_interrupt_enable.kENET_MiiInterrupt (*C enumerator*), 325
drivers_RT_RT1020_MIMXRT1024._enet_interrupt_enable.kENET_PayloadRxInterrupt (*C enumerator*), 326
drivers_RT_RT1020_MIMXRT1024._enet_interrupt_enable.kENET_RetryLimitInterrupt (*C enumerator*), 326
drivers_RT_RT1020_MIMXRT1024._enet_interrupt_enable.kENET_RxBufferInterrupt (*C enumerator*), 325
drivers_RT_RT1020_MIMXRT1024._enet_interrupt_enable.kENET_RxFrameInterrupt (*C enumerator*), 325
drivers_RT_RT1020_MIMXRT1024._enet_interrupt_enable.kENET_TsAvailInterrupt (*C enumerator*), 326
drivers_RT_RT1020_MIMXRT1024._enet_interrupt_enable.kENET_TsTimerInterrupt (*C enumerator*), 326
drivers_RT_RT1020_MIMXRT1024._enet_interrupt_enable.kENET_TxBufferInterrupt (*C enumerator*), 325
drivers_RT_RT1020_MIMXRT1024._enet_interrupt_enable.kENET_TxFrameInterrupt (*C enumerator*), 325
drivers_RT_RT1020_MIMXRT1024._enet_interrupt_enable.kENET_UnderrunInterrupt (*C enumerator*), 326
drivers_RT_RT1020_MIMXRT1024._enet_interrupt_enable.kENET_WakeupInterrupt (*C enumerator*), 326
drivers_RT_RT1020_MIMXRT1024._enet_mii_duplex (*C enum*), 323
drivers_RT_RT1020_MIMXRT1024._enet_mii_duplex.kENET_MiiFullDuplex (*C enumerator*), 323
drivers_RT_RT1020_MIMXRT1024._enet_mii_duplex.kENET_MiiHalfDuplex (*C enumerator*), 323
drivers_RT_RT1020_MIMXRT1024._enet_mii_extend_opcode (*C enum*), 324
drivers_RT_RT1020_MIMXRT1024._enet_mii_extend_opcode.kENET_MiiAddrWrite_C45 (*C enumerator*), 324
drivers_RT_RT1020_MIMXRT1024._enet_mii_extend_opcode.kENET_MiiReadFrame_C45 (*C enumerator*), 324
drivers_RT_RT1020_MIMXRT1024._enet_mii_extend_opcode.kENET_MiiWriteFrame_C45 (*C enumerator*), 324
drivers_RT_RT1020_MIMXRT1024._enet_mii_mode (*C enum*), 322
drivers_RT_RT1020_MIMXRT1024._enet_mii_mode.kENET_MiiMode (*C enumerator*), 323
drivers_RT_RT1020_MIMXRT1024._enet_mii_mode.kENET_RgmiiMode (*C enumerator*), 323
drivers_RT_RT1020_MIMXRT1024._enet_mii_mode.kENET_RmiiMode (*C enumerator*), 323
drivers_RT_RT1020_MIMXRT1024._enet_mii_read (*C enum*), 323
drivers_RT_RT1020_MIMXRT1024._enet_mii_read.kENET_MiiReadNoCompliant (*C enumerator*), 324
drivers_RT_RT1020_MIMXRT1024._enet_mii_read.kENET_MiiReadValidFrame (*C enumerator*), 324
drivers_RT_RT1020_MIMXRT1024._enet_mii_speed (*C enum*), 323
drivers_RT_RT1020_MIMXRT1024._enet_mii_speed.kENET_MiiSpeed10M (*C enumerator*), 323
drivers_RT_RT1020_MIMXRT1024._enet_mii_speed.kENET_MiiSpeed100M (*C enumerator*), 323
drivers_RT_RT1020_MIMXRT1024._enet_mii_speed.kENET_MiiSpeed1000M (*C enumerator*), 323
drivers_RT_RT1020_MIMXRT1024._enet_mii_write (*C enum*), 323
drivers_RT_RT1020_MIMXRT1024._enet_mii_write.kENET_MiiWriteNoCompliant (*C enumerator*), 323
drivers_RT_RT1020_MIMXRT1024._enet_mii_write.kENET_MiiWriteValidFrame (*C enumerator*), 323
drivers_RT_RT1020_MIMXRT1024._enet_rx_accelerator (*C enum*), 328
drivers_RT_RT1020_MIMXRT1024._enet_rx_accelerator.kENET_RxAccelIpCheckEnabled (*C enumerator*), 328
drivers_RT_RT1020_MIMXRT1024._enet_rx_accelerator.kENET_RxAccelisShift16Enabled (*C enumerator*), 328
drivers_RT_RT1020_MIMXRT1024._enet_rx_accelerator.kENET_RxAccelMacCheckEnabled (*C enumerator*), 328
drivers_RT_RT1020_MIMXRT1024._enet_rx_accelerator.kENET_RxAccelPadRemoveEnabled (*C*

enumerator), 328
drivers_RT_RT1020_MIMXRT1024._enet_rx_accelerator.kENET_RxAccelProtoCheckEnabled (*C enumerator*), 328
drivers_RT_RT1020_MIMXRT1024._enet_rx_bd_ring (*C struct*), 339
drivers_RT_RT1020_MIMXRT1024._enet_rx_bd_ring.rxBdBase (*C var*), 339
drivers_RT_RT1020_MIMXRT1024._enet_rx_bd_ring.rxGenIdx (*C var*), 339
drivers_RT_RT1020_MIMXRT1024._enet_rx_bd_ring.rxRingLen (*C var*), 339
drivers_RT_RT1020_MIMXRT1024._enet_rx_bd_struct (*C struct*), 332
drivers_RT_RT1020_MIMXRT1024._enet_rx_bd_struct.buffer (*C var*), 332
drivers_RT_RT1020_MIMXRT1024._enet_rx_bd_struct.control (*C var*), 332
drivers_RT_RT1020_MIMXRT1024._enet_rx_bd_struct.length (*C var*), 332
drivers_RT_RT1020_MIMXRT1024._enet_rx_frame_attribute_struct (*C struct*), 341
drivers_RT_RT1020_MIMXRT1024._enet_rx_frame_attribute_struct.promiscuous (*C var*), 341
drivers_RT_RT1020_MIMXRT1024._enet_rx_frame_error (*C struct*), 333
drivers_RT_RT1020_MIMXRT1024._enet_rx_frame_error.statsRxAlignErr (*C var*), 333
drivers_RT_RT1020_MIMXRT1024._enet_rx_frame_error.statsRxFcsErr (*C var*), 333
drivers_RT_RT1020_MIMXRT1024._enet_rx_frame_error.statsRxLenGreaterErr (*C var*), 333
drivers_RT_RT1020_MIMXRT1024._enet_rx_frame_error.statsRxOverRunErr (*C var*), 333
drivers_RT_RT1020_MIMXRT1024._enet_rx_frame_error.statsRxTruncateErr (*C var*), 333
drivers_RT_RT1020_MIMXRT1024._enet_rx_frame_struct (*C struct*), 341
drivers_RT_RT1020_MIMXRT1024._enet_rx_frame_struct.rxAttribute (*C var*), 341
drivers_RT_RT1020_MIMXRT1024._enet_rx_frame_struct.rxBuffArray (*C var*), 341
drivers_RT_RT1020_MIMXRT1024._enet_rx_frame_struct.rxFrameError (*C var*), 341
drivers_RT_RT1020_MIMXRT1024._enet_rx_frame_struct.totLen (*C var*), 341
drivers_RT_RT1020_MIMXRT1024._enet_special_control_flag (*C enum*), 324
drivers_RT_RT1020_MIMXRT1024._enet_special_control_flag.kENET_ControlFlowControlEnable (*C enumerator*), 324
drivers_RT_RT1020_MIMXRT1024._enet_special_control_flag.kENET_ControlMacAddrInsert (*C enumerator*), 324
drivers_RT_RT1020_MIMXRT1024._enet_special_control_flag.kENET_ControlMIILoopEnable (*C enumerator*), 325
drivers_RT_RT1020_MIMXRT1024._enet_special_control_flag.kENET_ControlPromiscuousEnable (*C enumerator*), 325
drivers_RT_RT1020_MIMXRT1024._enet_special_control_flag.kENET_ControlRxBroadCastRejectEnable (*C enumerator*), 324
drivers_RT_RT1020_MIMXRT1024._enet_special_control_flag.kENET_ControlRxPadRemoveEnable (*C enumerator*), 324
drivers_RT_RT1020_MIMXRT1024._enet_special_control_flag.kENET_ControlRxPayloadCheckEnable (*C enumerator*), 324
drivers_RT_RT1020_MIMXRT1024._enet_special_control_flag.kENET_ControlSMIPreambleDisable (*C enumerator*), 325
drivers_RT_RT1020_MIMXRT1024._enet_special_control_flag.kENET_ControlStoreAndFwdDisable (*C enumerator*), 324
drivers_RT_RT1020_MIMXRT1024._enet_special_control_flag.kENET_ControlSVLANEnable (*C enumerator*), 325
drivers_RT_RT1020_MIMXRT1024._enet_special_control_flag.kENET_ControlVLANTagEnable (*C enumerator*), 325
drivers_RT_RT1020_MIMXRT1024._enet_special_control_flag.kENET_ControlVLANUseSecondTag (*C enumerator*), 325
drivers_RT_RT1020_MIMXRT1024._enet_transfer_stats (*C struct*), 334
drivers_RT_RT1020_MIMXRT1024._enet_transfer_stats.statsRxAlignErr (*C var*), 334
drivers_RT_RT1020_MIMXRT1024._enet_transfer_stats.statsRxCrcErr (*C var*), 334
drivers_RT_RT1020_MIMXRT1024._enet_transfer_stats.statsRxDropInvalidSFD (*C var*), 334
drivers_RT_RT1020_MIMXRT1024._enet_transfer_stats.statsRxFifoOverflowErr (*C var*), 334
drivers_RT_RT1020_MIMXRT1024._enet_transfer_stats.statsRxFrameCount (*C var*), 334
drivers_RT_RT1020_MIMXRT1024._enet_transfer_stats.statsRxFrameOk (*C var*), 334
drivers_RT_RT1020_MIMXRT1024._enet_transfer_stats.statsTxCrcAlignErr (*C var*), 334
drivers_RT_RT1020_MIMXRT1024._enet_transfer_stats.statsTxFifoUnderRunErr (*C var*), 334

drivers_RT_RT1020_MIMXRT1024._enet_transfer_stats.statsTxFrameCount (*C var*), 334
drivers_RT_RT1020_MIMXRT1024._enet_transfer_stats.statsTxFrameOk (*C var*), 334
drivers_RT_RT1020_MIMXRT1024._enet_tx_accelerator (*C enum*), 328
drivers_RT_RT1020_MIMXRT1024._enet_tx_accelerator.kENET_TxAccelIpCheckEnabled (*C enumerator*), 328
drivers_RT_RT1020_MIMXRT1024._enet_tx_accelerator.kENET_TxAccelIsShift16Enabled (*C enumerator*), 328
drivers_RT_RT1020_MIMXRT1024._enet_tx_accelerator.kENET_TxAccelProtoCheckEnabled (*C enumerator*), 328
drivers_RT_RT1020_MIMXRT1024._enet_tx_bd_ring (*C struct*), 339
drivers_RT_RT1020_MIMXRT1024._enet_tx_bd_ring.txBdBase (*C var*), 339
drivers_RT_RT1020_MIMXRT1024._enet_tx_bd_ring.txConsumIdx (*C var*), 339
drivers_RT_RT1020_MIMXRT1024._enet_tx_bd_ring.txDescUsed (*C var*), 339
drivers_RT_RT1020_MIMXRT1024._enet_tx_bd_ring.txGenIdx (*C var*), 339
drivers_RT_RT1020_MIMXRT1024._enet_tx_bd_ring.txRingLen (*C var*), 339
drivers_RT_RT1020_MIMXRT1024._enet_tx_bd_struct (*C struct*), 332
drivers_RT_RT1020_MIMXRT1024._enet_tx_bd_struct.buffer (*C var*), 333
drivers_RT_RT1020_MIMXRT1024._enet_tx_bd_struct.control (*C var*), 333
drivers_RT_RT1020_MIMXRT1024._enet_tx_bd_struct.length (*C var*), 333
drivers_RT_RT1020_MIMXRT1024._enet_tx_dirty_ring (*C struct*), 334
drivers_RT_RT1020_MIMXRT1024._enet_tx_dirty_ring.isFull (*C var*), 335
drivers_RT_RT1020_MIMXRT1024._enet_tx_dirty_ring.txConsumIdx (*C var*), 335
drivers_RT_RT1020_MIMXRT1024._enet_tx_dirty_ring.txDirtyBase (*C var*), 335
drivers_RT_RT1020_MIMXRT1024._enet_tx_dirty_ring.txGenIdx (*C var*), 335
drivers_RT_RT1020_MIMXRT1024._enet_tx_dirty_ring.txRingLen (*C var*), 335
drivers_RT_RT1020_MIMXRT1024._enet_tx_frame_struct (*C struct*), 341
drivers_RT_RT1020_MIMXRT1024._enet_tx_frame_struct.context (*C var*), 342
drivers_RT_RT1020_MIMXRT1024._enet_tx_frame_struct.txBuffArray (*C var*), 341
drivers_RT_RT1020_MIMXRT1024._enet_tx_frame_struct.txBuffNum (*C var*), 341
drivers_RT_RT1020_MIMXRT1024._ewm_config (*C struct*), 344
drivers_RT_RT1020_MIMXRT1024._ewm_config.clockSource (*C var*), 345
drivers_RT_RT1020_MIMXRT1024._ewm_config.compareHighValue (*C var*), 345
drivers_RT_RT1020_MIMXRT1024._ewm_config.compareLowValue (*C var*), 345
drivers_RT_RT1020_MIMXRT1024._ewm_config.enableEwm (*C var*), 344
drivers_RT_RT1020_MIMXRT1024._ewm_config.enableEwmInput (*C var*), 345
drivers_RT_RT1020_MIMXRT1024._ewm_config.enableInterrupt (*C var*), 345
drivers_RT_RT1020_MIMXRT1024._ewm_config.prescaler (*C var*), 345
drivers_RT_RT1020_MIMXRT1024._ewm_config.setInputAssertLogic (*C var*), 345
drivers_RT_RT1020_MIMXRT1024._ewm_interrupt_enable_t (*C enum*), 344
drivers_RT_RT1020_MIMXRT1024._ewm_interrupt_enable_t.kEWM_InterruptEnable (*C enumerator*), 344
drivers_RT_RT1020_MIMXRT1024._ewm_lpo_clock_source (*C enum*), 343
drivers_RT_RT1020_MIMXRT1024._ewm_lpo_clock_source.kEWM_LpoClockSource0 (*C enumerator*), 344
drivers_RT_RT1020_MIMXRT1024._ewm_lpo_clock_source.kEWM_LpoClockSource1 (*C enumerator*), 344
drivers_RT_RT1020_MIMXRT1024._ewm_lpo_clock_source.kEWM_LpoClockSource2 (*C enumerator*), 344
drivers_RT_RT1020_MIMXRT1024._ewm_lpo_clock_source.kEWM_LpoClockSource3 (*C enumerator*), 344
drivers_RT_RT1020_MIMXRT1024._ewm_status_flags_t (*C enum*), 344
drivers_RT_RT1020_MIMXRT1024._ewm_status_flags_t.kEWM_RunningFlag (*C enumerator*), 344
drivers_RT_RT1020_MIMXRT1024._flexcan_byte_error_syndrome (*C enum*), 374
drivers_RT_RT1020_MIMXRT1024._flexcan_byte_error_syndrome.kFLEXCAN_AllOneError (*C enumerator*), 375
drivers_RT_RT1020_MIMXRT1024._flexcan_byte_error_syndrome.kFLEXCAN_AllZeroError (*C enumerator*), 375

drivers_RT_RT1020_MIMXRT1024._flexcan_byte_error_syndrome.kFLEXCAN_DataBits0Error (*C enumerator*), 374
drivers_RT_RT1020_MIMXRT1024._flexcan_byte_error_syndrome.kFLEXCAN_DataBits1Error (*C enumerator*), 374
drivers_RT_RT1020_MIMXRT1024._flexcan_byte_error_syndrome.kFLEXCAN_DataBits2Error (*C enumerator*), 374
drivers_RT_RT1020_MIMXRT1024._flexcan_byte_error_syndrome.kFLEXCAN_DataBits3Error (*C enumerator*), 375
drivers_RT_RT1020_MIMXRT1024._flexcan_byte_error_syndrome.kFLEXCAN_DataBits4Error (*C enumerator*), 375
drivers_RT_RT1020_MIMXRT1024._flexcan_byte_error_syndrome.kFLEXCAN_DataBits5Error (*C enumerator*), 375
drivers_RT_RT1020_MIMXRT1024._flexcan_byte_error_syndrome.kFLEXCAN_DataBits6Error (*C enumerator*), 375
drivers_RT_RT1020_MIMXRT1024._flexcan_byte_error_syndrome.kFLEXCAN_DataBits7Error (*C enumerator*), 375
drivers_RT_RT1020_MIMXRT1024._flexcan_byte_error_syndrome.kFLEXCAN_NoError (*C enumerator*), 374
drivers_RT_RT1020_MIMXRT1024._flexcan_byte_error_syndrome.kFLEXCAN_NonCorrectableErrors (*C enumerator*), 375
drivers_RT_RT1020_MIMXRT1024._flexcan_byte_error_syndrome.kFLEXCAN_ParityBits0Error (*C enumerator*), 374
drivers_RT_RT1020_MIMXRT1024._flexcan_byte_error_syndrome.kFLEXCAN_ParityBits1Error (*C enumerator*), 374
drivers_RT_RT1020_MIMXRT1024._flexcan_byte_error_syndrome.kFLEXCAN_ParityBits2Error (*C enumerator*), 374
drivers_RT_RT1020_MIMXRT1024._flexcan_byte_error_syndrome.kFLEXCAN_ParityBits3Error (*C enumerator*), 374
drivers_RT_RT1020_MIMXRT1024._flexcan_byte_error_syndrome.kFLEXCAN_ParityBits4Error (*C enumerator*), 374
drivers_RT_RT1020_MIMXRT1024._flexcan_capture_point (*C enum*), 367
drivers_RT_RT1020_MIMXRT1024._flexcan_capture_point.kFLEXCAN_CANFDFrameRes (*C enumerator*), 367
drivers_RT_RT1020_MIMXRT1024._flexcan_capture_point.kFLEXCAN_CANFrameEnd (*C enumerator*), 367
drivers_RT_RT1020_MIMXRT1024._flexcan_capture_point.kFLEXCAN_CANFrameID2ndBit (*C enumerator*), 367
drivers_RT_RT1020_MIMXRT1024._flexcan_capture_point.kFLEXCAN_CANFrameStart (*C enumerator*), 367
drivers_RT_RT1020_MIMXRT1024._flexcan_clock_source (*C enum*), 366
drivers_RT_RT1020_MIMXRT1024._flexcan_clock_source.kFLEXCAN_ClkSrc0 (*C enumerator*), 366
drivers_RT_RT1020_MIMXRT1024._flexcan_clock_source.kFLEXCAN_ClkSrc1 (*C enumerator*), 366
drivers_RT_RT1020_MIMXRT1024._flexcan_clock_source.kFLEXCAN_ClkSrcOsc (*C enumerator*), 366
drivers_RT_RT1020_MIMXRT1024._flexcan_clock_source.kFLEXCAN_ClkSrcPeri (*C enumerator*), 366
drivers_RT_RT1020_MIMXRT1024._flexcan_config (*C struct*), 382
drivers_RT_RT1020_MIMXRT1024._flexcan_config.___unnamed29___ (*C union*), 391
drivers_RT_RT1020_MIMXRT1024._flexcan_config.___unnamed29___.___unnamed31___ (*C struct*), 391
drivers_RT_RT1020_MIMXRT1024._flexcan_config.___unnamed29___.___unnamed31___.baudRate (*C var*), 391
drivers_RT_RT1020_MIMXRT1024._flexcan_config.___unnamed29___.___unnamed31___.baudRateFD (*C var*), 391
drivers_RT_RT1020_MIMXRT1024._flexcan_config.___unnamed29___.___unnamed33___ (*C struct*), 391

drivers_RT_RT1020_MIMXRT1024._flexcan_config.___unnamed29___.___unnamed33___.bitRate (*C var*), 391
drivers_RT_RT1020_MIMXRT1024._flexcan_config.___unnamed29___.___unnamed33___.bitRateFD (*C var*), 391
drivers_RT_RT1020_MIMXRT1024._flexcan_config.capturePoint (*C var*), 384
drivers_RT_RT1020_MIMXRT1024._flexcan_config.captureTimeBase (*C var*), 384
drivers_RT_RT1020_MIMXRT1024._flexcan_config.clkSrc (*C var*), 382
drivers_RT_RT1020_MIMXRT1024._flexcan_config.disableBusOffAutoRecovery (*C var*), 383
drivers_RT_RT1020_MIMXRT1024._flexcan_config.disableSelfReception (*C var*), 383
drivers_RT_RT1020_MIMXRT1024._flexcan_config.enableDoze (*C var*), 383
drivers_RT_RT1020_MIMXRT1024._flexcan_config.enableEntireArbitrationCompare (*C var*), 383
drivers_RT_RT1020_MIMXRT1024._flexcan_config.enableExternalTimeTick (*C var*), 384
drivers_RT_RT1020_MIMXRT1024._flexcan_config.enableIndividMask (*C var*), 383
drivers_RT_RT1020_MIMXRT1024._flexcan_config.enableListenOnlyMode (*C var*), 383
drivers_RT_RT1020_MIMXRT1024._flexcan_config.enableLoopBack (*C var*), 382
drivers_RT_RT1020_MIMXRT1024._flexcan_config.enableMemoryErrorControl (*C var*), 383
drivers_RT_RT1020_MIMXRT1024._flexcan_config.enableNonCorrectableErrorEnterFreeze (*C var*), 383
drivers_RT_RT1020_MIMXRT1024._flexcan_config.enablePretendedeNetworking (*C var*), 383
drivers_RT_RT1020_MIMXRT1024._flexcan_config.enableRemoteRequestFrameStored (*C var*), 383
drivers_RT_RT1020_MIMXRT1024._flexcan_config.enableThreeSamplingMode (*C var*), 383
drivers_RT_RT1020_MIMXRT1024._flexcan_config.enableTimerSync (*C var*), 382
drivers_RT_RT1020_MIMXRT1024._flexcan_config.enableTransceiverDelayMeasure (*C var*), 383
drivers_RT_RT1020_MIMXRT1024._flexcan_config.maxMbNum (*C var*), 382
drivers_RT_RT1020_MIMXRT1024._flexcan_config.payloadEndianness (*C var*), 384
drivers_RT_RT1020_MIMXRT1024._flexcan_config.txArbitrationStartDelay (*C var*), 383
drivers_RT_RT1020_MIMXRT1024._flexcan_config.wakeupSrc (*C var*), 382
drivers_RT_RT1020_MIMXRT1024._flexcan_endianness (*C enum*), 367
drivers_RT_RT1020_MIMXRT1024._flexcan_endianness.kFLEXCAN_bigEndian (*C enumerator*), 367
drivers_RT_RT1020_MIMXRT1024._flexcan_endianness.kFLEXCAN_littleEndian (*C enumerator*), 367
drivers_RT_RT1020_MIMXRT1024._flexcan_error_flags (*C enum*), 372
drivers_RT_RT1020_MIMXRT1024._flexcan_error_flags.kFLEXCAN_AckError (*C enumerator*), 373
drivers_RT_RT1020_MIMXRT1024._flexcan_error_flags.kFLEXCAN_Bit0Error (*C enumerator*), 373
drivers_RT_RT1020_MIMXRT1024._flexcan_error_flags.kFLEXCAN_Bit1Error (*C enumerator*), 373
drivers_RT_RT1020_MIMXRT1024._flexcan_error_flags.kFLEXCAN_CrcError (*C enumerator*), 373
drivers_RT_RT1020_MIMXRT1024._flexcan_error_flags.kFLEXCAN_FDBit0Error (*C enumerator*), 372
drivers_RT_RT1020_MIMXRT1024._flexcan_error_flags.kFLEXCAN_FDBit1Error (*C enumerator*), 372
drivers_RT_RT1020_MIMXRT1024._flexcan_error_flags.kFLEXCAN_FDCrcError (*C enumerator*), 372
drivers_RT_RT1020_MIMXRT1024._flexcan_error_flags.kFLEXCAN_FDFormError (*C enumerator*), 372
drivers_RT_RT1020_MIMXRT1024._flexcan_error_flags.kFLEXCAN_FDStuffingError (*C enumerator*), 372
drivers_RT_RT1020_MIMXRT1024._flexcan_error_flags.kFLEXCAN_FormError (*C enumerator*), 373
drivers_RT_RT1020_MIMXRT1024._flexcan_error_flags.kFLEXCAN_RxErrorWarningFlag (*C enumerator*), 372
drivers_RT_RT1020_MIMXRT1024._flexcan_error_flags.kFLEXCAN_StuffingError (*C enumerator*), 373

drivers_RT_RT1020_MIMXRT1024._flexcan_error_flags.kFLEXCAN_TxErrorWarningFlag (*C enumerator*), 372
drivers_RT_RT1020_MIMXRT1024._flexcan_fd_frame (*C struct*), 381
drivers_RT_RT1020_MIMXRT1024._flexcan_fd_frame.___unnamed19___ (*C struct*), 389
drivers_RT_RT1020_MIMXRT1024._flexcan_fd_frame.___unnamed19___.brs (*C var*), 389
drivers_RT_RT1020_MIMXRT1024._flexcan_fd_frame.___unnamed19___.edl (*C var*), 389
drivers_RT_RT1020_MIMXRT1024._flexcan_fd_frame.___unnamed19___.esi (*C var*), 389
drivers_RT_RT1020_MIMXRT1024._flexcan_fd_frame.___unnamed19___.format (*C var*), 389
drivers_RT_RT1020_MIMXRT1024._flexcan_fd_frame.___unnamed19___.length (*C var*), 389
drivers_RT_RT1020_MIMXRT1024._flexcan_fd_frame.___unnamed19___.srr (*C var*), 389
drivers_RT_RT1020_MIMXRT1024._flexcan_fd_frame.___unnamed19___.timestamp (*C var*), 389
drivers_RT_RT1020_MIMXRT1024._flexcan_fd_frame.___unnamed19___.type (*C var*), 389
drivers_RT_RT1020_MIMXRT1024._flexcan_fd_frame.___unnamed21___ (*C struct*), 389
drivers_RT_RT1020_MIMXRT1024._flexcan_fd_frame.___unnamed21___.___pad0___ (*C var*), 390
drivers_RT_RT1020_MIMXRT1024._flexcan_fd_frame.___unnamed21___.id (*C var*), 390
drivers_RT_RT1020_MIMXRT1024._flexcan_fd_frame.___unnamed23___ (*C union*), 390
drivers_RT_RT1020_MIMXRT1024._flexcan_fd_frame.___unnamed23___.___unnamed25___ (*C struct*), 390
drivers_RT_RT1020_MIMXRT1024._flexcan_fd_frame.___unnamed23___.___unnamed25___.dataWord (*C var*), 390
drivers_RT_RT1020_MIMXRT1024._flexcan_fd_frame.___unnamed23___.___unnamed27___ (*C struct*), 390
drivers_RT_RT1020_MIMXRT1024._flexcan_fd_frame.___unnamed23___.___unnamed27___.dataByte0 (*C var*), 390
drivers_RT_RT1020_MIMXRT1024._flexcan_fd_frame.___unnamed23___.___unnamed27___.dataByte1 (*C var*), 390
drivers_RT_RT1020_MIMXRT1024._flexcan_fd_frame.___unnamed23___.___unnamed27___.dataByte2 (*C var*), 390
drivers_RT_RT1020_MIMXRT1024._flexcan_fd_frame.___unnamed23___.___unnamed27___.dataByte3 (*C var*), 390
drivers_RT_RT1020_MIMXRT1024._flexcan_fd_frame.___unnamed23___.___unnamed27___.dataByte4 (*C var*), 391
drivers_RT_RT1020_MIMXRT1024._flexcan_fd_frame.___unnamed23___.___unnamed27___.dataByte5 (*C var*), 390
drivers_RT_RT1020_MIMXRT1024._flexcan_fd_frame.___unnamed23___.___unnamed27___.dataByte6 (*C var*), 390
drivers_RT_RT1020_MIMXRT1024._flexcan_fd_frame.___unnamed23___.___unnamed27___.dataByte7 (*C var*), 390
drivers_RT_RT1020_MIMXRT1024._flexcan_fd_frame_length (*C enum*), 368
drivers_RT_RT1020_MIMXRT1024._flexcan_fd_frame_length.kFLEXCAN_0BperFrame (*C enumerator*), 368
drivers_RT_RT1020_MIMXRT1024._flexcan_fd_frame_length.kFLEXCAN_1BperFrame (*C enumerator*), 368
drivers_RT_RT1020_MIMXRT1024._flexcan_fd_frame_length.kFLEXCAN_2BperFrame (*C enumerator*), 368
drivers_RT_RT1020_MIMXRT1024._flexcan_fd_frame_length.kFLEXCAN_3BperFrame (*C enumerator*), 369
drivers_RT_RT1020_MIMXRT1024._flexcan_fd_frame_length.kFLEXCAN_4BperFrame (*C enumerator*), 369
drivers_RT_RT1020_MIMXRT1024._flexcan_fd_frame_length.kFLEXCAN_5BperFrame (*C enumerator*), 369
drivers_RT_RT1020_MIMXRT1024._flexcan_fd_frame_length.kFLEXCAN_6BperFrame (*C enumerator*), 369
drivers_RT_RT1020_MIMXRT1024._flexcan_fd_frame_length.kFLEXCAN_7BperFrame (*C enumerator*), 369
drivers_RT_RT1020_MIMXRT1024._flexcan_fd_frame_length.kFLEXCAN_8BperFrame (*C enumerator*), 369
drivers_RT_RT1020_MIMXRT1024._flexcan_fd_frame_length.kFLEXCAN_12BperFrame (*C enu-*

merator), 369
drivers_RT_RT1020_MIMXRT1024._flexcan_fd_frame_length.kFLEXCAN_16BperFrame (*C enumerator*), 369
drivers_RT_RT1020_MIMXRT1024._flexcan_fd_frame_length.kFLEXCAN_20BperFrame (*C enumerator*), 369
drivers_RT_RT1020_MIMXRT1024._flexcan_fd_frame_length.kFLEXCAN_24BperFrame (*C enumerator*), 369
drivers_RT_RT1020_MIMXRT1024._flexcan_fd_frame_length.kFLEXCAN_32BperFrame (*C enumerator*), 369
drivers_RT_RT1020_MIMXRT1024._flexcan_fd_frame_length.kFLEXCAN_48BperFrame (*C enumerator*), 369
drivers_RT_RT1020_MIMXRT1024._flexcan_fd_frame_length.kFLEXCAN_64BperFrame (*C enumerator*), 369
drivers_RT_RT1020_MIMXRT1024._flexcan_fd_frame.hrtimestamp (*C var*), 381
drivers_RT_RT1020_MIMXRT1024._flexcan_fifo_transfer (*C struct*), 386
drivers_RT_RT1020_MIMXRT1024._flexcan_fifo_transfer.frame (*C var*), 386
drivers_RT_RT1020_MIMXRT1024._flexcan_fifo_transfer.frameNum (*C var*), 386
drivers_RT_RT1020_MIMXRT1024._flexcan_flags (*C enum*), 370
drivers_RT_RT1020_MIMXRT1024._flexcan_flags.kFLEXCAN_AllMemoryErrorFlag (*C enumerator*), 372
drivers_RT_RT1020_MIMXRT1024._flexcan_flags.kFLEXCAN_AllMemoryErrorIntFlag (*C enumerator*), 372
drivers_RT_RT1020_MIMXRT1024._flexcan_flags.kFLEXCAN_BusoffDoneIntFlag (*C enumerator*), 371
drivers_RT_RT1020_MIMXRT1024._flexcan_flags.kFLEXCAN_BusOffIntFlag (*C enumerator*), 371
drivers_RT_RT1020_MIMXRT1024._flexcan_flags.kFLEXCAN_CorrectableErrorIntFlag (*C enumerator*), 372
drivers_RT_RT1020_MIMXRT1024._flexcan_flags.kFLEXCAN_CorrectableErrorOverrunFlag (*C enumerator*), 372
drivers_RT_RT1020_MIMXRT1024._flexcan_flags.kFLEXCAN_ErrorFlag (*C enumerator*), 371
drivers_RT_RT1020_MIMXRT1024._flexcan_flags.kFLEXCAN_ErrorIntFlag (*C enumerator*), 371
drivers_RT_RT1020_MIMXRT1024._flexcan_flags.kFLEXCAN_ErrorOverrunFlag (*C enumerator*), 371
drivers_RT_RT1020_MIMXRT1024._flexcan_flags.kFLEXCAN_FaultConfinementFlag (*C enumerator*), 371
drivers_RT_RT1020_MIMXRT1024._flexcan_flags.kFLEXCAN_FDErrorIntFlag (*C enumerator*), 371
drivers_RT_RT1020_MIMXRT1024._flexcan_flags.kFLEXCAN_FlexCanAccessNonCorrectableErrorIntFlag (*C enumerator*), 372
drivers_RT_RT1020_MIMXRT1024._flexcan_flags.kFLEXCAN_FlexCanAccessNonCorrectableErrorOverrunFlag (*C enumerator*), 372
drivers_RT_RT1020_MIMXRT1024._flexcan_flags.kFLEXCAN_HostAccessNonCorrectableErrorIntFlag (*C enumerator*), 371
drivers_RT_RT1020_MIMXRT1024._flexcan_flags.kFLEXCAN_HostAccessNonCorrectableErrorOverrunFlag (*C enumerator*), 372
drivers_RT_RT1020_MIMXRT1024._flexcan_flags.kFLEXCAN_IdleFlag (*C enumerator*), 371
drivers_RT_RT1020_MIMXRT1024._flexcan_flags.kFLEXCAN_PNMatchIntFlag (*C enumerator*), 371
drivers_RT_RT1020_MIMXRT1024._flexcan_flags.kFLEXCAN_PNTimeoutIntFlag (*C enumerator*), 371
drivers_RT_RT1020_MIMXRT1024._flexcan_flags.kFLEXCAN_ReceivingFlag (*C enumerator*), 371
drivers_RT_RT1020_MIMXRT1024._flexcan_flags.kFLEXCAN_RxWarningIntFlag (*C enumerator*), 371
drivers_RT_RT1020_MIMXRT1024._flexcan_flags.kFLEXCAN_SynchFlag (*C enumerator*), 371
drivers_RT_RT1020_MIMXRT1024._flexcan_flags.kFLEXCAN_TransmittingFlag (*C enumerator*), 371
drivers_RT_RT1020_MIMXRT1024._flexcan_flags.kFLEXCAN_TxWarningIntFlag (*C enumerator*), 371

drivers_RT_RT1020_MIMXRT1024._flexcan_frame (*C struct*), 381
drivers_RT_RT1020_MIMXRT1024._flexcan_frame.___unnamed9___ (*C struct*), 387
drivers_RT_RT1020_MIMXRT1024._flexcan_frame.___unnamed9___.___pad0___ (*C var*), 387
drivers_RT_RT1020_MIMXRT1024._flexcan_frame.___unnamed9___.format (*C var*), 387
drivers_RT_RT1020_MIMXRT1024._flexcan_frame.___unnamed9___.idhit (*C var*), 387
drivers_RT_RT1020_MIMXRT1024._flexcan_frame.___unnamed9___.length (*C var*), 387
drivers_RT_RT1020_MIMXRT1024._flexcan_frame.___unnamed9___.timestamp (*C var*), 387
drivers_RT_RT1020_MIMXRT1024._flexcan_frame.___unnamed9___.type (*C var*), 387
drivers_RT_RT1020_MIMXRT1024._flexcan_frame.___unnamed11___ (*C struct*), 388
drivers_RT_RT1020_MIMXRT1024._flexcan_frame.___unnamed11___.___pad0___ (*C var*), 388
drivers_RT_RT1020_MIMXRT1024._flexcan_frame.___unnamed11___.id (*C var*), 388
drivers_RT_RT1020_MIMXRT1024._flexcan_frame.___unnamed13___ (*C union*), 388
drivers_RT_RT1020_MIMXRT1024._flexcan_frame.___unnamed13___.___unnamed15___ (*C struct*), 388
drivers_RT_RT1020_MIMXRT1024._flexcan_frame.___unnamed13___.___unnamed15___.dataWord0 (*C var*), 388
drivers_RT_RT1020_MIMXRT1024._flexcan_frame.___unnamed13___.___unnamed15___.dataWord1 (*C var*), 388
drivers_RT_RT1020_MIMXRT1024._flexcan_frame.___unnamed13___.___unnamed17___ (*C struct*), 388
drivers_RT_RT1020_MIMXRT1024._flexcan_frame.___unnamed13___.___unnamed17___.dataByte0 (*C var*), 388
drivers_RT_RT1020_MIMXRT1024._flexcan_frame.___unnamed13___.___unnamed17___.dataByte1 (*C var*), 388
drivers_RT_RT1020_MIMXRT1024._flexcan_frame.___unnamed13___.___unnamed17___.dataByte2 (*C var*), 388
drivers_RT_RT1020_MIMXRT1024._flexcan_frame.___unnamed13___.___unnamed17___.dataByte3 (*C var*), 388
drivers_RT_RT1020_MIMXRT1024._flexcan_frame.___unnamed13___.___unnamed17___.dataByte4 (*C var*), 389
drivers_RT_RT1020_MIMXRT1024._flexcan_frame.___unnamed13___.___unnamed17___.dataByte5 (*C var*), 389
drivers_RT_RT1020_MIMXRT1024._flexcan_frame.___unnamed13___.___unnamed17___.dataByte6 (*C var*), 389
drivers_RT_RT1020_MIMXRT1024._flexcan_frame.___unnamed13___.___unnamed17___.dataByte7 (*C var*), 389
drivers_RT_RT1020_MIMXRT1024._flexcan_frame_format (*C enum*), 366
drivers_RT_RT1020_MIMXRT1024._flexcan_frame_format.kFLEXCAN_FrameFormatExtend (*C enumerator*), 366
drivers_RT_RT1020_MIMXRT1024._flexcan_frame_format.kFLEXCAN_FrameFormatStandard (*C enumerator*), 366
drivers_RT_RT1020_MIMXRT1024._flexcan_frame_type (*C enum*), 366
drivers_RT_RT1020_MIMXRT1024._flexcan_frame_type.kFLEXCAN_FrameTypeData (*C enumerator*), 366
drivers_RT_RT1020_MIMXRT1024._flexcan_frame_type.kFLEXCAN_FrameTypeRemote (*C enumerator*), 366
drivers_RT_RT1020_MIMXRT1024._flexcan_handle (*C struct*), 386
drivers_RT_RT1020_MIMXRT1024._flexcan_handle.callback (*C var*), 386
drivers_RT_RT1020_MIMXRT1024._flexcan_handle.mbFDFrameBuf (*C var*), 386
drivers_RT_RT1020_MIMXRT1024._flexcan_handle.mbFrameBuf (*C var*), 386
drivers_RT_RT1020_MIMXRT1024._flexcan_handle.mbState (*C var*), 387
drivers_RT_RT1020_MIMXRT1024._flexcan_handle.rxFifoFrameBuf (*C var*), 386
drivers_RT_RT1020_MIMXRT1024._flexcan_handle.rxFifoFrameNum (*C var*), 387
drivers_RT_RT1020_MIMXRT1024._flexcan_handle.rxFifoState (*C var*), 387
drivers_RT_RT1020_MIMXRT1024._flexcan_handle.rxFifoTransferTotalNum (*C var*), 387
drivers_RT_RT1020_MIMXRT1024._flexcan_handle.timestamp (*C var*), 387
drivers_RT_RT1020_MIMXRT1024._flexcan_handle.userData (*C var*), 386
drivers_RT_RT1020_MIMXRT1024._flexcan_interrupt_enable (*C enum*), 370

drivers_RT_RT1020_MIMXRT1024._flexcan_interrupt_enable.kFLEXCAN_BusOffInterruptEnable (*C enumerator*), 370
drivers_RT_RT1020_MIMXRT1024._flexcan_interrupt_enable.kFLEXCAN_ErrorInterruptEnable (*C enumerator*), 370
drivers_RT_RT1020_MIMXRT1024._flexcan_interrupt_enable.kFLEXCAN_FDErrorInterruptEnable (*C enumerator*), 370
drivers_RT_RT1020_MIMXRT1024._flexcan_interrupt_enable.kFLEXCAN_FlexCanAccessNCErrorInterruptEnable (*C enumerator*), 370
drivers_RT_RT1020_MIMXRT1024._flexcan_interrupt_enable.kFLEXCAN_HostAccessNCErrorInterruptEnable (*C enumerator*), 370
drivers_RT_RT1020_MIMXRT1024._flexcan_interrupt_enable.kFLEXCAN_HostOrFlexCanCErrorInterruptEnable (*C enumerator*), 370
drivers_RT_RT1020_MIMXRT1024._flexcan_interrupt_enable.kFLEXCAN_PNMatchWakeUpInterruptEnable (*C enumerator*), 370
drivers_RT_RT1020_MIMXRT1024._flexcan_interrupt_enable.kFLEXCAN_PNTimeoutWakeUpInterruptEnable (*C enumerator*), 370
drivers_RT_RT1020_MIMXRT1024._flexcan_interrupt_enable.kFLEXCAN_RxWarningInterruptEnable (*C enumerator*), 370
drivers_RT_RT1020_MIMXRT1024._flexcan_interrupt_enable.kFLEXCAN_TxWarningInterruptEnable (*C enumerator*), 370
drivers_RT_RT1020_MIMXRT1024._flexcan_mb_size (*C enum*), 368
drivers_RT_RT1020_MIMXRT1024._flexcan_mb_size.kFLEXCAN_8BperMB (*C enumerator*), 368
drivers_RT_RT1020_MIMXRT1024._flexcan_mb_size.kFLEXCAN_16BperMB (*C enumerator*), 368
drivers_RT_RT1020_MIMXRT1024._flexcan_mb_size.kFLEXCAN_32BperMB (*C enumerator*), 368
drivers_RT_RT1020_MIMXRT1024._flexcan_mb_size.kFLEXCAN_64BperMB (*C enumerator*), 368
drivers_RT_RT1020_MIMXRT1024._flexcan_MB_timestamp_base (*C enum*), 367
drivers_RT_RT1020_MIMXRT1024._flexcan_MB_timestamp_base.kFLEXCAN_CANTimer (*C enumerator*), 367
drivers_RT_RT1020_MIMXRT1024._flexcan_MB_timestamp_base.kFLEXCAN_Lower16bitsHRTimer (*C enumerator*), 367
drivers_RT_RT1020_MIMXRT1024._flexcan_MB_timestamp_base.kFLEXCAN_Upper16bitsHRTimer (*C enumerator*), 367
drivers_RT_RT1020_MIMXRT1024._flexcan_mb_transfer (*C struct*), 386
drivers_RT_RT1020_MIMXRT1024._flexcan_mb_transfer.frame (*C var*), 386
drivers_RT_RT1020_MIMXRT1024._flexcan_mb_transfer.mbIdx (*C var*), 386
drivers_RT_RT1020_MIMXRT1024._flexcan_memory_access_type (*C enum*), 373
drivers_RT_RT1020_MIMXRT1024._flexcan_memory_access_type.kFLEXCAN_MoveInAccess (*C enumerator*), 374
drivers_RT_RT1020_MIMXRT1024._flexcan_memory_access_type.kFLEXCAN_MoveOutFlexCanAccess (*C enumerator*), 374
drivers_RT_RT1020_MIMXRT1024._flexcan_memory_access_type.kFLEXCAN_MoveOutHostAccess (*C enumerator*), 374
drivers_RT_RT1020_MIMXRT1024._flexcan_memory_access_type.kFLEXCAN_RxMatchingAccess (*C enumerator*), 374
drivers_RT_RT1020_MIMXRT1024._flexcan_memory_access_type.kFLEXCAN_TxArbitrationAccess (*C enumerator*), 374
drivers_RT_RT1020_MIMXRT1024._flexcan_memory_error_report_status (*C struct*), 380
drivers_RT_RT1020_MIMXRT1024._flexcan_memory_error_report_status.accessAddress (*C var*), 381
drivers_RT_RT1020_MIMXRT1024._flexcan_memory_error_report_status.accessType (*C var*), 381
drivers_RT_RT1020_MIMXRT1024._flexcan_memory_error_report_status.byteStatus (*C struct*), 387
drivers_RT_RT1020_MIMXRT1024._flexcan_memory_error_report_status.byteStatus.byteIsRead (*C var*), 387
drivers_RT_RT1020_MIMXRT1024._flexcan_memory_error_report_status.errorData (*C var*), 381
drivers_RT_RT1020_MIMXRT1024._flexcan_memory_error_report_status.errorType (*C var*), 381
drivers_RT_RT1020_MIMXRT1024._flexcan_memory_error_type (*C enum*), 373
drivers_RT_RT1020_MIMXRT1024._flexcan_memory_error_type.kFLEXCAN_CorrectableError (*C*

enumerator), 373
drivers_RT_RT1020_MIMXRT1024._flexcan_memory_error_type.kFLEXCAN_NonCorrectableError (*C enumerator*), 373
drivers_RT_RT1020_MIMXRT1024._flexcan_pn_config (*C struct*), 384
drivers_RT_RT1020_MIMXRT1024._flexcan_pn_config.___unnamed35___ (*C union*), 391
drivers_RT_RT1020_MIMXRT1024._flexcan_pn_config.___unnamed35___.___unnamed39___ (*C struct*), 392
drivers_RT_RT1020_MIMXRT1024._flexcan_pn_config.___unnamed35___.___unnamed39___.lowerWord0 (*C var*), 392
drivers_RT_RT1020_MIMXRT1024._flexcan_pn_config.___unnamed35___.___unnamed39___.lowerWord1 (*C var*), 392
drivers_RT_RT1020_MIMXRT1024._flexcan_pn_config.___unnamed35___.___unnamed41___ (*C struct*), 392
drivers_RT_RT1020_MIMXRT1024._flexcan_pn_config.___unnamed35___.___unnamed41___.lowerByte0 (*C var*), 392
drivers_RT_RT1020_MIMXRT1024._flexcan_pn_config.___unnamed35___.___unnamed41___.lowerByte1 (*C var*), 392
drivers_RT_RT1020_MIMXRT1024._flexcan_pn_config.___unnamed35___.___unnamed41___.lowerByte2 (*C var*), 392
drivers_RT_RT1020_MIMXRT1024._flexcan_pn_config.___unnamed35___.___unnamed41___.lowerByte3 (*C var*), 392
drivers_RT_RT1020_MIMXRT1024._flexcan_pn_config.___unnamed35___.___unnamed41___.lowerByte4 (*C var*), 392
drivers_RT_RT1020_MIMXRT1024._flexcan_pn_config.___unnamed35___.___unnamed41___.lowerByte5 (*C var*), 392
drivers_RT_RT1020_MIMXRT1024._flexcan_pn_config.___unnamed35___.___unnamed41___.lowerByte6 (*C var*), 392
drivers_RT_RT1020_MIMXRT1024._flexcan_pn_config.___unnamed35___.___unnamed41___.lowerByte7 (*C var*), 392
drivers_RT_RT1020_MIMXRT1024._flexcan_pn_config.___unnamed37___ (*C union*), 392
drivers_RT_RT1020_MIMXRT1024._flexcan_pn_config.___unnamed37___.___unnamed43___ (*C struct*), 393
drivers_RT_RT1020_MIMXRT1024._flexcan_pn_config.___unnamed37___.___unnamed43___.upperWord0 (*C var*), 393
drivers_RT_RT1020_MIMXRT1024._flexcan_pn_config.___unnamed37___.___unnamed43___.upperWord1 (*C var*), 393
drivers_RT_RT1020_MIMXRT1024._flexcan_pn_config.___unnamed37___.___unnamed45___ (*C struct*), 393
drivers_RT_RT1020_MIMXRT1024._flexcan_pn_config.___unnamed37___.___unnamed45___.upperByte0 (*C var*), 393
drivers_RT_RT1020_MIMXRT1024._flexcan_pn_config.___unnamed37___.___unnamed45___.upperByte1 (*C var*), 393
drivers_RT_RT1020_MIMXRT1024._flexcan_pn_config.___unnamed37___.___unnamed45___.upperByte2 (*C var*), 393
drivers_RT_RT1020_MIMXRT1024._flexcan_pn_config.___unnamed37___.___unnamed45___.upperByte3 (*C var*), 393
drivers_RT_RT1020_MIMXRT1024._flexcan_pn_config.___unnamed37___.___unnamed45___.upperByte4 (*C var*), 393
drivers_RT_RT1020_MIMXRT1024._flexcan_pn_config.___unnamed37___.___unnamed45___.upperByte5 (*C var*), 393
drivers_RT_RT1020_MIMXRT1024._flexcan_pn_config.___unnamed37___.___unnamed45___.upperByte6 (*C var*), 393
drivers_RT_RT1020_MIMXRT1024._flexcan_pn_config.___unnamed37___.___unnamed45___.upperByte7 (*C var*), 393
drivers_RT_RT1020_MIMXRT1024._flexcan_pn_config.dataMatchMode (*C var*), 385
drivers_RT_RT1020_MIMXRT1024._flexcan_pn_config.enableMatch (*C var*), 385
drivers_RT_RT1020_MIMXRT1024._flexcan_pn_config.enableTimeout (*C var*), 384
drivers_RT_RT1020_MIMXRT1024._flexcan_pn_config.idLower (*C var*), 385

drivers_RT_RT1020_MIMXRT1024._flexcan_pn_config.idMatchMode (*C var*), 385
drivers_RT_RT1020_MIMXRT1024._flexcan_pn_config.idUpper (*C var*), 385
drivers_RT_RT1020_MIMXRT1024._flexcan_pn_config.lengthLower (*C var*), 385
drivers_RT_RT1020_MIMXRT1024._flexcan_pn_config.lengthUpper (*C var*), 385
drivers_RT_RT1020_MIMXRT1024._flexcan_pn_config.matchNum (*C var*), 385
drivers_RT_RT1020_MIMXRT1024._flexcan_pn_config.matchSrc (*C var*), 385
drivers_RT_RT1020_MIMXRT1024._flexcan_pn_config.timeoutValue (*C var*), 384
drivers_RT_RT1020_MIMXRT1024._flexcan_pn_match_mode (*C enum*), 375
drivers_RT_RT1020_MIMXRT1024._flexcan_pn_match_mode.kFLEXCAN_PNMatModeEqual (*C enumerator*), 375
drivers_RT_RT1020_MIMXRT1024._flexcan_pn_match_mode.kFLEXCAN_PNMatModeGreater (*C enumerator*), 375
drivers_RT_RT1020_MIMXRT1024._flexcan_pn_match_mode.kFLEXCAN_PNMatModeRange (*C enumerator*), 376
drivers_RT_RT1020_MIMXRT1024._flexcan_pn_match_mode.kFLEXCAN_PNMatModeSmaller (*C enumerator*), 375
drivers_RT_RT1020_MIMXRT1024._flexcan_pn_match_source (*C enum*), 375
drivers_RT_RT1020_MIMXRT1024._flexcan_pn_match_source.kFLEXCAN_PNMatSrcID (*C enumerator*), 375
drivers_RT_RT1020_MIMXRT1024._flexcan_pn_match_source.kFLEXCAN_PNMatSrcIDAndData (*C enumerator*), 375
drivers_RT_RT1020_MIMXRT1024._flexcan_rx_fifo_config (*C struct*), 385
drivers_RT_RT1020_MIMXRT1024._flexcan_rx_fifo_config.idFilterNum (*C var*), 385
drivers_RT_RT1020_MIMXRT1024._flexcan_rx_fifo_config.idFilterTable (*C var*), 385
drivers_RT_RT1020_MIMXRT1024._flexcan_rx_fifo_config.idFilterType (*C var*), 385
drivers_RT_RT1020_MIMXRT1024._flexcan_rx_fifo_config.priority (*C var*), 386
drivers_RT_RT1020_MIMXRT1024._flexcan_rx_fifo_filter_type (*C enum*), 367
drivers_RT_RT1020_MIMXRT1024._flexcan_rx_fifo_filter_type.kFLEXCAN_RxFifoFilterTypeA (*C enumerator*), 368
drivers_RT_RT1020_MIMXRT1024._flexcan_rx_fifo_filter_type.kFLEXCAN_RxFifoFilterTypeB (*C enumerator*), 368
drivers_RT_RT1020_MIMXRT1024._flexcan_rx_fifo_filter_type.kFLEXCAN_RxFifoFilterTypeC (*C enumerator*), 368
drivers_RT_RT1020_MIMXRT1024._flexcan_rx_fifo_filter_type.kFLEXCAN_RxFifoFilterTypeD (*C enumerator*), 368
drivers_RT_RT1020_MIMXRT1024._flexcan_rx_fifo_priority (*C enum*), 369
drivers_RT_RT1020_MIMXRT1024._flexcan_rx_fifo_priority.kFLEXCAN_RxFifoPrioHigh (*C enumerator*), 370
drivers_RT_RT1020_MIMXRT1024._flexcan_rx_fifo_priority.kFLEXCAN_RxFifoPrioLow (*C enumerator*), 369
drivers_RT_RT1020_MIMXRT1024._flexcan_rx_mb_config (*C struct*), 384
drivers_RT_RT1020_MIMXRT1024._flexcan_rx_mb_config.format (*C var*), 384
drivers_RT_RT1020_MIMXRT1024._flexcan_rx_mb_config.id (*C var*), 384
drivers_RT_RT1020_MIMXRT1024._flexcan_rx_mb_config.type (*C var*), 384
drivers_RT_RT1020_MIMXRT1024._flexcan_timing_config (*C struct*), 381
drivers_RT_RT1020_MIMXRT1024._flexcan_timing_config.fphaseSeg1 (*C var*), 382
drivers_RT_RT1020_MIMXRT1024._flexcan_timing_config.fphaseSeg2 (*C var*), 382
drivers_RT_RT1020_MIMXRT1024._flexcan_timing_config.fpreDivider (*C var*), 382
drivers_RT_RT1020_MIMXRT1024._flexcan_timing_config.fpropSeg (*C var*), 382
drivers_RT_RT1020_MIMXRT1024._flexcan_timing_config.frJumpwidth (*C var*), 382
drivers_RT_RT1020_MIMXRT1024._flexcan_timing_config.phaseSeg1 (*C var*), 382
drivers_RT_RT1020_MIMXRT1024._flexcan_timing_config.phaseSeg2 (*C var*), 382
drivers_RT_RT1020_MIMXRT1024._flexcan_timing_config.preDivider (*C var*), 381
drivers_RT_RT1020_MIMXRT1024._flexcan_timing_config.propSeg (*C var*), 382
drivers_RT_RT1020_MIMXRT1024._flexcan_timing_config.rJumpwidth (*C var*), 381
drivers_RT_RT1020_MIMXRT1024._flexcan_wake_up_source (*C enum*), 366
drivers_RT_RT1020_MIMXRT1024._flexcan_wake_up_source.kFLEXCAN_WakeupSrcFiltered (*C enumerator*), 367

drivers_RT_RT1020_MIMXRT1024._flexcan_wake_up_source.kFLEXCAN_WakeupSrcUnfiltered (*C enumerator*), 366
drivers_RT_RT1020_MIMXRT1024._flexio_config_ (*C struct*), 411
drivers_RT_RT1020_MIMXRT1024._flexio_config_.enableFastAccess (*C var*), 411
drivers_RT_RT1020_MIMXRT1024._flexio_config_.enableFlexio (*C var*), 411
drivers_RT_RT1020_MIMXRT1024._flexio_config_.enableInDebug (*C var*), 411
drivers_RT_RT1020_MIMXRT1024._flexio_config_.enableInDoze (*C var*), 411
drivers_RT_RT1020_MIMXRT1024._flexio_gpio_config (*C struct*), 413
drivers_RT_RT1020_MIMXRT1024._flexio_gpio_config.inputConfig (*C var*), 413
drivers_RT_RT1020_MIMXRT1024._flexio_gpio_config.outputLogic (*C var*), 413
drivers_RT_RT1020_MIMXRT1024._flexio_gpio_config.pinDirection (*C var*), 413
drivers_RT_RT1020_MIMXRT1024._flexio_gpio_direction (*C enum*), 408
drivers_RT_RT1020_MIMXRT1024._flexio_gpio_direction.kFLEXIO_DigitalInput (*C enumerator*), 408
drivers_RT_RT1020_MIMXRT1024._flexio_gpio_direction.kFLEXIO_DigitalOutput (*C enumerator*), 408
drivers_RT_RT1020_MIMXRT1024._flexio_i2c_direction (*C enum*), 431
drivers_RT_RT1020_MIMXRT1024._flexio_i2c_direction.kFLEXIO_I2C_Read (*C enumerator*), 431
drivers_RT_RT1020_MIMXRT1024._flexio_i2c_direction.kFLEXIO_I2C_Write (*C enumerator*), 431
drivers_RT_RT1020_MIMXRT1024._flexio_i2c_master_config (*C struct*), 432
drivers_RT_RT1020_MIMXRT1024._flexio_i2c_master_config.baudRate_Bps (*C var*), 432
drivers_RT_RT1020_MIMXRT1024._flexio_i2c_master_config.enableFastAccess (*C var*), 432
drivers_RT_RT1020_MIMXRT1024._flexio_i2c_master_config.enableInDebug (*C var*), 432
drivers_RT_RT1020_MIMXRT1024._flexio_i2c_master_config.enableInDoze (*C var*), 432
drivers_RT_RT1020_MIMXRT1024._flexio_i2c_master_config.enableMaster (*C var*), 432
drivers_RT_RT1020_MIMXRT1024._flexio_i2c_master_handle (*C struct*), 433
drivers_RT_RT1020_MIMXRT1024._flexio_i2c_master_handle.completionCallback (*C var*), 433
drivers_RT_RT1020_MIMXRT1024._flexio_i2c_master_handle.needRestart (*C var*), 433
drivers_RT_RT1020_MIMXRT1024._flexio_i2c_master_handle.state (*C var*), 433
drivers_RT_RT1020_MIMXRT1024._flexio_i2c_master_handle.transfer (*C var*), 433
drivers_RT_RT1020_MIMXRT1024._flexio_i2c_master_handle.transferSize (*C var*), 433
drivers_RT_RT1020_MIMXRT1024._flexio_i2c_master_handle.userData (*C var*), 433
drivers_RT_RT1020_MIMXRT1024._flexio_i2c_master_interrupt (*C enum*), 430
drivers_RT_RT1020_MIMXRT1024._flexio_i2c_master_interrupt.kFLEXIO_I2C_RxFullInterruptEnable (*C enumerator*), 430
drivers_RT_RT1020_MIMXRT1024._flexio_i2c_master_interrupt.kFLEXIO_I2C_TxEmptyInterruptEnable (*C enumerator*), 430
drivers_RT_RT1020_MIMXRT1024._flexio_i2c_master_status_flags (*C enum*), 430
drivers_RT_RT1020_MIMXRT1024._flexio_i2c_master_status_flags.kFLEXIO_I2C_ReceiveNakFlag (*C enumerator*), 431
drivers_RT_RT1020_MIMXRT1024._flexio_i2c_master_status_flags.kFLEXIO_I2C_RxFullFlag (*C enumerator*), 431
drivers_RT_RT1020_MIMXRT1024._flexio_i2c_master_status_flags.kFLEXIO_I2C_TxEmptyFlag (*C enumerator*), 431
drivers_RT_RT1020_MIMXRT1024._flexio_i2c_master_transfer (*C struct*), 432
drivers_RT_RT1020_MIMXRT1024._flexio_i2c_master_transfer.data (*C var*), 433
drivers_RT_RT1020_MIMXRT1024._flexio_i2c_master_transfer.dataSize (*C var*), 433
drivers_RT_RT1020_MIMXRT1024._flexio_i2c_master_transfer.direction (*C var*), 433
drivers_RT_RT1020_MIMXRT1024._flexio_i2c_master_transfer.flags (*C var*), 432
drivers_RT_RT1020_MIMXRT1024._flexio_i2c_master_transfer.slaveAddress (*C var*), 433
drivers_RT_RT1020_MIMXRT1024._flexio_i2c_master_transfer.subaddress (*C var*), 433
drivers_RT_RT1020_MIMXRT1024._flexio_i2c_master_transfer.subaddressSize (*C var*), 433
drivers_RT_RT1020_MIMXRT1024._flexio_i2c_type (*C struct*), 431
drivers_RT_RT1020_MIMXRT1024._flexio_i2c_type.baudrate (*C var*), 432
drivers_RT_RT1020_MIMXRT1024._flexio_i2c_type.flexioBase (*C var*), 432
drivers_RT_RT1020_MIMXRT1024._flexio_i2c_type.SCLPinIndex (*C var*), 432

drivers_RT_RT1020_MIMXRT1024._flexio_i2c_type.SDAPinIndex (*C var*), 432
drivers_RT_RT1020_MIMXRT1024._flexio_i2c_type.shifterIndex (*C var*), 432
drivers_RT_RT1020_MIMXRT1024._flexio_i2c_type.timerIndex (*C var*), 432
drivers_RT_RT1020_MIMXRT1024._flexio_i2s_config (*C struct*), 443
drivers_RT_RT1020_MIMXRT1024._flexio_i2s_config.bclkPinPolarity (*C var*), 444
drivers_RT_RT1020_MIMXRT1024._flexio_i2s_config.enableI2S (*C var*), 443
drivers_RT_RT1020_MIMXRT1024._flexio_i2s_config.fsPinPolarity (*C var*), 444
drivers_RT_RT1020_MIMXRT1024._flexio_i2s_config.masterSlave (*C var*), 443
drivers_RT_RT1020_MIMXRT1024._flexio_i2s_config.rxPinPolarity (*C var*), 444
drivers_RT_RT1020_MIMXRT1024._flexio_i2s_config.rxTimerPolarity (*C var*), 444
drivers_RT_RT1020_MIMXRT1024._flexio_i2s_config.txPinPolarity (*C var*), 443
drivers_RT_RT1020_MIMXRT1024._flexio_i2s_config.txTimerPolarity (*C var*), 444
drivers_RT_RT1020_MIMXRT1024._flexio_i2s_edma_handle (*C struct*), 416
drivers_RT_RT1020_MIMXRT1024._flexio_i2s_edma_handle.bytesPerFrame (*C var*), 416
drivers_RT_RT1020_MIMXRT1024._flexio_i2s_edma_handle.callback (*C var*), 417
drivers_RT_RT1020_MIMXRT1024._flexio_i2s_edma_handle.dmaHandle (*C var*), 416
drivers_RT_RT1020_MIMXRT1024._flexio_i2s_edma_handle.nbytes (*C var*), 417
drivers_RT_RT1020_MIMXRT1024._flexio_i2s_edma_handle.queue (*C var*), 417
drivers_RT_RT1020_MIMXRT1024._flexio_i2s_edma_handle.queueDriver (*C var*), 417
drivers_RT_RT1020_MIMXRT1024._flexio_i2s_edma_handle.queueUser (*C var*), 417
drivers_RT_RT1020_MIMXRT1024._flexio_i2s_edma_handle.state (*C var*), 417
drivers_RT_RT1020_MIMXRT1024._flexio_i2s_edma_handle.tcd (*C var*), 417
drivers_RT_RT1020_MIMXRT1024._flexio_i2s_edma_handle.transferSize (*C var*), 417
drivers_RT_RT1020_MIMXRT1024._flexio_i2s_edma_handle.userData (*C var*), 417
drivers_RT_RT1020_MIMXRT1024._flexio_i2s_format (*C struct*), 444
drivers_RT_RT1020_MIMXRT1024._flexio_i2s_format.bitWidth (*C var*), 444
drivers_RT_RT1020_MIMXRT1024._flexio_i2s_format.sampleRate_Hz (*C var*), 444
drivers_RT_RT1020_MIMXRT1024._flexio_i2s_handle (*C struct*), 444
drivers_RT_RT1020_MIMXRT1024._flexio_i2s_handle.bitWidth (*C var*), 445
drivers_RT_RT1020_MIMXRT1024._flexio_i2s_handle.callback (*C var*), 445
drivers_RT_RT1020_MIMXRT1024._flexio_i2s_handle.queue (*C var*), 445
drivers_RT_RT1020_MIMXRT1024._flexio_i2s_handle.queueDriver (*C var*), 445
drivers_RT_RT1020_MIMXRT1024._flexio_i2s_handle.queueUser (*C var*), 445
drivers_RT_RT1020_MIMXRT1024._flexio_i2s_handle.state (*C var*), 445
drivers_RT_RT1020_MIMXRT1024._flexio_i2s_handle.transferSize (*C var*), 445
drivers_RT_RT1020_MIMXRT1024._flexio_i2s_handle.userData (*C var*), 445
drivers_RT_RT1020_MIMXRT1024._flexio_i2s_master_slave (*C enum*), 440
drivers_RT_RT1020_MIMXRT1024._flexio_i2s_master_slave.kFLEXIO_I2S_Master (*C enumerator*), 440
drivers_RT_RT1020_MIMXRT1024._flexio_i2s_master_slave.kFLEXIO_I2S_Slave (*C enumerator*), 440
drivers_RT_RT1020_MIMXRT1024._flexio_i2s_sample_rate (*C enum*), 441
drivers_RT_RT1020_MIMXRT1024._flexio_i2s_sample_rate.kFLEXIO_I2S_SampleRate8KHz (*C enumerator*), 441
drivers_RT_RT1020_MIMXRT1024._flexio_i2s_sample_rate.kFLEXIO_I2S_SampleRate12KHz (*C enumerator*), 441
drivers_RT_RT1020_MIMXRT1024._flexio_i2s_sample_rate.kFLEXIO_I2S_SampleRate16KHz (*C enumerator*), 441
drivers_RT_RT1020_MIMXRT1024._flexio_i2s_sample_rate.kFLEXIO_I2S_SampleRate24KHz (*C enumerator*), 441
drivers_RT_RT1020_MIMXRT1024._flexio_i2s_sample_rate.kFLEXIO_I2S_SampleRate32KHz (*C enumerator*), 441
drivers_RT_RT1020_MIMXRT1024._flexio_i2s_sample_rate.kFLEXIO_I2S_SampleRate48KHz (*C enumerator*), 441
drivers_RT_RT1020_MIMXRT1024._flexio_i2s_sample_rate.kFLEXIO_I2S_SampleRate96KHz (*C enumerator*), 441
drivers_RT_RT1020_MIMXRT1024._flexio_i2s_sample_rate.kFLEXIO_I2S_SampleRate11025Hz (*C enumerator*), 441

drivers_RT_RT1020_MIMXRT1024._flexio_i2s_sample_rate.kFLEXIO_I2S_SampleRate22050Hz (*C enumerator*), 441
drivers_RT_RT1020_MIMXRT1024._flexio_i2s_sample_rate.kFLEXIO_I2S_SampleRate44100Hz (*C enumerator*), 441
drivers_RT_RT1020_MIMXRT1024._flexio_i2s_transfer (*C struct*), 444
drivers_RT_RT1020_MIMXRT1024._flexio_i2s_transfer.data (*C var*), 444
drivers_RT_RT1020_MIMXRT1024._flexio_i2s_transfer.dataSize (*C var*), 444
drivers_RT_RT1020_MIMXRT1024._flexio_i2s_type (*C struct*), 443
drivers_RT_RT1020_MIMXRT1024._flexio_i2s_type.bclkPinIndex (*C var*), 443
drivers_RT_RT1020_MIMXRT1024._flexio_i2s_type.bclkTimerIndex (*C var*), 443
drivers_RT_RT1020_MIMXRT1024._flexio_i2s_type.flexioBase (*C var*), 443
drivers_RT_RT1020_MIMXRT1024._flexio_i2s_type.fsPinIndex (*C var*), 443
drivers_RT_RT1020_MIMXRT1024._flexio_i2s_type.fsTimerIndex (*C var*), 443
drivers_RT_RT1020_MIMXRT1024._flexio_i2s_type.rxPinIndex (*C var*), 443
drivers_RT_RT1020_MIMXRT1024._flexio_i2s_type.rxShifterIndex (*C var*), 443
drivers_RT_RT1020_MIMXRT1024._flexio_i2s_type.txPinIndex (*C var*), 443
drivers_RT_RT1020_MIMXRT1024._flexio_i2s_type.txShifterIndex (*C var*), 443
drivers_RT_RT1020_MIMXRT1024._flexio_i2s_word_width (*C enum*), 442
drivers_RT_RT1020_MIMXRT1024._flexio_i2s_word_width.kFLEXIO_I2S_WordWidth8bits (*C enumerator*), 442
drivers_RT_RT1020_MIMXRT1024._flexio_i2s_word_width.kFLEXIO_I2S_WordWidth16bits (*C enumerator*), 442
drivers_RT_RT1020_MIMXRT1024._flexio_i2s_word_width.kFLEXIO_I2S_WordWidth24bits (*C enumerator*), 442
drivers_RT_RT1020_MIMXRT1024._flexio_i2s_word_width.kFLEXIO_I2S_WordWidth32bits (*C enumerator*), 442
drivers_RT_RT1020_MIMXRT1024._flexio_pin_config (*C enum*), 402
drivers_RT_RT1020_MIMXRT1024._flexio_pin_config.kFLEXIO_PinConfigBidirectionOutputData (*C enumerator*), 402
drivers_RT_RT1020_MIMXRT1024._flexio_pin_config.kFLEXIO_PinConfigOpenDrainOrBidirection (*C enumerator*), 402
drivers_RT_RT1020_MIMXRT1024._flexio_pin_config.kFLEXIO_PinConfigOutput (*C enumerator*), 402
drivers_RT_RT1020_MIMXRT1024._flexio_pin_config.kFLEXIO_PinConfigOutputDisabled (*C enumerator*), 402
drivers_RT_RT1020_MIMXRT1024._flexio_pin_input_config (*C enum*), 408
drivers_RT_RT1020_MIMXRT1024._flexio_pin_input_config.kFLEXIO_FlagFallingEdgeEnable (*C enumerator*), 409
drivers_RT_RT1020_MIMXRT1024._flexio_pin_input_config.kFLEXIO_FlagRisingEdgeEnable (*C enumerator*), 408
drivers_RT_RT1020_MIMXRT1024._flexio_pin_input_config.kFLEXIO_InputInterruptDisabled (*C enumerator*), 408
drivers_RT_RT1020_MIMXRT1024._flexio_pin_input_config.kFLEXIO_InputInterruptEnable (*C enumerator*), 408
drivers_RT_RT1020_MIMXRT1024._flexio_pin_polarity (*C enum*), 402
drivers_RT_RT1020_MIMXRT1024._flexio_pin_polarity.kFLEXIO_PinActiveHigh (*C enumerator*), 402
drivers_RT_RT1020_MIMXRT1024._flexio_pin_polarity.kFLEXIO_PinActiveLow (*C enumerator*), 403
drivers_RT_RT1020_MIMXRT1024._flexio_shifter_buffer_type (*C enum*), 408
drivers_RT_RT1020_MIMXRT1024._flexio_shifter_buffer_type.kFLEXIO_ShifterBuffer (*C enumerator*), 408
drivers_RT_RT1020_MIMXRT1024._flexio_shifter_buffer_type.kFLEXIO_ShifterBufferBitByteSwapped (*C enumerator*), 408
drivers_RT_RT1020_MIMXRT1024._flexio_shifter_buffer_type.kFLEXIO_ShifterBufferBitSwapped (*C enumerator*), 408
drivers_RT_RT1020_MIMXRT1024._flexio_shifter_buffer_type.kFLEXIO_ShifterBufferByteSwapped (*C enumerator*), 408

drivers_RT_RT1020_MIMXRT1024._flexio_shifter_buffer_type.kFLEXIO_ShifterBufferHalfWordSwapped (*C enumerator*), 408
drivers_RT_RT1020_MIMXRT1024._flexio_shifter_buffer_type.kFLEXIO_ShifterBufferNibbleByteSwapped (*C enumerator*), 408
drivers_RT_RT1020_MIMXRT1024._flexio_shifter_buffer_type.kFLEXIO_ShifterBufferNibbleSwapped (*C enumerator*), 408
drivers_RT_RT1020_MIMXRT1024._flexio_shifter_config (*C struct*), 412
drivers_RT_RT1020_MIMXRT1024._flexio_shifter_config.inputSource (*C var*), 413
drivers_RT_RT1020_MIMXRT1024._flexio_shifter_config.parallelWidth (*C var*), 413
drivers_RT_RT1020_MIMXRT1024._flexio_shifter_config.pinConfig (*C var*), 413
drivers_RT_RT1020_MIMXRT1024._flexio_shifter_config.pinPolarity (*C var*), 413
drivers_RT_RT1020_MIMXRT1024._flexio_shifter_config.pinSelect (*C var*), 413
drivers_RT_RT1020_MIMXRT1024._flexio_shifter_config.shifterMode (*C var*), 413
drivers_RT_RT1020_MIMXRT1024._flexio_shifter_config.shifterStart (*C var*), 413
drivers_RT_RT1020_MIMXRT1024._flexio_shifter_config.shifterStop (*C var*), 413
drivers_RT_RT1020_MIMXRT1024._flexio_shifter_config.timerPolarity (*C var*), 413
drivers_RT_RT1020_MIMXRT1024._flexio_shifter_config.timerSelect (*C var*), 413
drivers_RT_RT1020_MIMXRT1024._flexio_shifter_input_source (*C enum*), 407
drivers_RT_RT1020_MIMXRT1024._flexio_shifter_input_source.kFLEXIO_ShifterInputFromNextShifterOutput (*C enumerator*), 407
drivers_RT_RT1020_MIMXRT1024._flexio_shifter_input_source.kFLEXIO_ShifterInputFromPin (*C enumerator*), 407
drivers_RT_RT1020_MIMXRT1024._flexio_shifter_mode (*C enum*), 406
drivers_RT_RT1020_MIMXRT1024._flexio_shifter_mode.kFLEXIO_ShifterDisabled (*C enumerator*), 406
drivers_RT_RT1020_MIMXRT1024._flexio_shifter_mode.kFLEXIO_ShifterModeLogic (*C enumerator*), 407
drivers_RT_RT1020_MIMXRT1024._flexio_shifter_mode.kFLEXIO_ShifterModeMatchContinuous (*C enumerator*), 407
drivers_RT_RT1020_MIMXRT1024._flexio_shifter_mode.kFLEXIO_ShifterModeMatchStore (*C enumerator*), 406
drivers_RT_RT1020_MIMXRT1024._flexio_shifter_mode.kFLEXIO_ShifterModeReceive (*C enumerator*), 406
drivers_RT_RT1020_MIMXRT1024._flexio_shifter_mode.kFLEXIO_ShifterModeState (*C enumerator*), 407
drivers_RT_RT1020_MIMXRT1024._flexio_shifter_mode.kFLEXIO_ShifterModeTransmit (*C enumerator*), 406
drivers_RT_RT1020_MIMXRT1024._flexio_shifter_start_bit (*C enum*), 407
drivers_RT_RT1020_MIMXRT1024._flexio_shifter_start_bit.kFLEXIO_ShifterStartBitDisabledLoadDataOnEnable (*C enumerator*), 407
drivers_RT_RT1020_MIMXRT1024._flexio_shifter_start_bit.kFLEXIO_ShifterStartBitDisabledLoadDataOnShift (*C enumerator*), 407
drivers_RT_RT1020_MIMXRT1024._flexio_shifter_start_bit.kFLEXIO_ShifterStartBitHigh (*C enumerator*), 407
drivers_RT_RT1020_MIMXRT1024._flexio_shifter_start_bit.kFLEXIO_ShifterStartBitLow (*C enumerator*), 407
drivers_RT_RT1020_MIMXRT1024._flexio_shifter_stop_bit (*C enum*), 407
drivers_RT_RT1020_MIMXRT1024._flexio_shifter_stop_bit.kFLEXIO_ShifterStopBitDisable (*C enumerator*), 407
drivers_RT_RT1020_MIMXRT1024._flexio_shifter_stop_bit.kFLEXIO_ShifterStopBitHigh (*C enumerator*), 407
drivers_RT_RT1020_MIMXRT1024._flexio_shifter_stop_bit.kFLEXIO_ShifterStopBitLow (*C enumerator*), 407
drivers_RT_RT1020_MIMXRT1024._flexio_shifter_timer_polarity (*C enum*), 406
drivers_RT_RT1020_MIMXRT1024._flexio_shifter_timer_polarity.kFLEXIO_ShifterTimerPolarityOnNegitive (*C enumerator*), 406
drivers_RT_RT1020_MIMXRT1024._flexio_shifter_timer_polarity.kFLEXIO_ShifterTimerPolarityOnPositive (*C enumerator*), 406

drivers_RT_RT1020_MIMXRT1024._flexio_spi_clock_phase (*C enum*), 454
drivers_RT_RT1020_MIMXRT1024._flexio_spi_clock_phase.kFLEXIO_SPI_ClockPhaseFirstEdge (*C enumerator*), 454
drivers_RT_RT1020_MIMXRT1024._flexio_spi_clock_phase.kFLEXIO_SPI_ClockPhaseSecondEdge (*C enumerator*), 454
drivers_RT_RT1020_MIMXRT1024._flexio_spi_data_bitcount_mode (*C enum*), 454
drivers_RT_RT1020_MIMXRT1024._flexio_spi_data_bitcount_mode.kFLEXIO_SPI_8BitMode (*C enumerator*), 454
drivers_RT_RT1020_MIMXRT1024._flexio_spi_data_bitcount_mode.kFLEXIO_SPI_16BitMode (*C enumerator*), 454
drivers_RT_RT1020_MIMXRT1024._flexio_spi_data_bitcount_mode.kFLEXIO_SPI_32BitMode (*C enumerator*), 454
drivers_RT_RT1020_MIMXRT1024._flexio_spi_dma_enable (*C enum*), 455
drivers_RT_RT1020_MIMXRT1024._flexio_spi_dma_enable.kFLEXIO_SPI_DmaAllEnable (*C enumerator*), 455
drivers_RT_RT1020_MIMXRT1024._flexio_spi_dma_enable.kFLEXIO_SPI_RxDmaEnable (*C enumerator*), 455
drivers_RT_RT1020_MIMXRT1024._flexio_spi_dma_enable.kFLEXIO_SPI_TxDmaEnable (*C enumerator*), 455
drivers_RT_RT1020_MIMXRT1024._flexio_spi_interrupt_enable (*C enum*), 454
drivers_RT_RT1020_MIMXRT1024._flexio_spi_interrupt_enable.kFLEXIO_SPI_RxFullInterruptEnable (*C enumerator*), 454
drivers_RT_RT1020_MIMXRT1024._flexio_spi_interrupt_enable.kFLEXIO_SPI_TxEmptyInterruptEnable (*C enumerator*), 454
drivers_RT_RT1020_MIMXRT1024._flexio_spi_master_config (*C struct*), 457
drivers_RT_RT1020_MIMXRT1024._flexio_spi_master_config.baudRate_Bps (*C var*), 457
drivers_RT_RT1020_MIMXRT1024._flexio_spi_master_config.dataMode (*C var*), 457
drivers_RT_RT1020_MIMXRT1024._flexio_spi_master_config.enableFastAccess (*C var*), 457
drivers_RT_RT1020_MIMXRT1024._flexio_spi_master_config.enableInDebug (*C var*), 457
drivers_RT_RT1020_MIMXRT1024._flexio_spi_master_config.enableInDoze (*C var*), 457
drivers_RT_RT1020_MIMXRT1024._flexio_spi_master_config.enableMaster (*C var*), 457
drivers_RT_RT1020_MIMXRT1024._flexio_spi_master_config.phase (*C var*), 457
drivers_RT_RT1020_MIMXRT1024._flexio_spi_master_edma_handle (*C struct*), 420
drivers_RT_RT1020_MIMXRT1024._flexio_spi_master_edma_handle.callback (*C var*), 421
drivers_RT_RT1020_MIMXRT1024._flexio_spi_master_edma_handle.nbytes (*C var*), 420
drivers_RT_RT1020_MIMXRT1024._flexio_spi_master_edma_handle.rxHandle (*C var*), 420
drivers_RT_RT1020_MIMXRT1024._flexio_spi_master_edma_handle.rxInProgress (*C var*), 420
drivers_RT_RT1020_MIMXRT1024._flexio_spi_master_edma_handle.transferSize (*C var*), 420
drivers_RT_RT1020_MIMXRT1024._flexio_spi_master_edma_handle.txHandle (*C var*), 420
drivers_RT_RT1020_MIMXRT1024._flexio_spi_master_edma_handle.txInProgress (*C var*), 420
drivers_RT_RT1020_MIMXRT1024._flexio_spi_master_edma_handle.userData (*C var*), 421
drivers_RT_RT1020_MIMXRT1024._flexio_spi_master_handle (*C struct*), 458
drivers_RT_RT1020_MIMXRT1024._flexio_spi_master_handle.bytePerFrame (*C var*), 459
drivers_RT_RT1020_MIMXRT1024._flexio_spi_master_handle.callback (*C var*), 459
drivers_RT_RT1020_MIMXRT1024._flexio_spi_master_handle.direction (*C var*), 459
drivers_RT_RT1020_MIMXRT1024._flexio_spi_master_handle.isCsContinuous (*C var*), 459
drivers_RT_RT1020_MIMXRT1024._flexio_spi_master_handle.rxData (*C var*), 459
drivers_RT_RT1020_MIMXRT1024._flexio_spi_master_handle.rxRemainingBytes (*C var*), 459
drivers_RT_RT1020_MIMXRT1024._flexio_spi_master_handle.state (*C var*), 459
drivers_RT_RT1020_MIMXRT1024._flexio_spi_master_handle.timer1Cfg (*C var*), 459
drivers_RT_RT1020_MIMXRT1024._flexio_spi_master_handle.transferSize (*C var*), 459
drivers_RT_RT1020_MIMXRT1024._flexio_spi_master_handle.txData (*C var*), 459
drivers_RT_RT1020_MIMXRT1024._flexio_spi_master_handle.txRemainingBytes (*C var*), 459
drivers_RT_RT1020_MIMXRT1024._flexio_spi_master_handle.userData (*C var*), 459
drivers_RT_RT1020_MIMXRT1024._flexio_spi_shift_direction (*C enum*), 454
drivers_RT_RT1020_MIMXRT1024._flexio_spi_shift_direction.kFLEXIO_SPI_LsbFirst (*C enumerator*), 454

drivers_RT_RT1020_MIMXRT1024._flexio_spi_shift_direction.kFLEXIO_SPI_MsbFirst (*C enumerator*), 454
drivers_RT_RT1020_MIMXRT1024._flexio_spi_slave_config (*C struct*), 458
drivers_RT_RT1020_MIMXRT1024._flexio_spi_slave_config.dataMode (*C var*), 458
drivers_RT_RT1020_MIMXRT1024._flexio_spi_slave_config.enableFastAccess (*C var*), 458
drivers_RT_RT1020_MIMXRT1024._flexio_spi_slave_config.enableInDebug (*C var*), 458
drivers_RT_RT1020_MIMXRT1024._flexio_spi_slave_config.enableInDoze (*C var*), 458
drivers_RT_RT1020_MIMXRT1024._flexio_spi_slave_config.enableSlave (*C var*), 458
drivers_RT_RT1020_MIMXRT1024._flexio_spi_slave_config.phase (*C var*), 458
drivers_RT_RT1020_MIMXRT1024._flexio_spi_status_flags (*C enum*), 454
drivers_RT_RT1020_MIMXRT1024._flexio_spi_status_flags.kFLEXIO_SPI_RxBufferFullFlag (*C enumerator*), 455
drivers_RT_RT1020_MIMXRT1024._flexio_spi_status_flags.kFLEXIO_SPI_TxBufferEmptyFlag (*C enumerator*), 455
drivers_RT_RT1020_MIMXRT1024._flexio_spi_transfer (*C struct*), 458
drivers_RT_RT1020_MIMXRT1024._flexio_spi_transfer_flags (*C enum*), 455
drivers_RT_RT1020_MIMXRT1024._flexio_spi_transfer_flags.kFLEXIO_SPI_8bitLsb (*C enumerator*), 455
drivers_RT_RT1020_MIMXRT1024._flexio_spi_transfer_flags.kFLEXIO_SPI_8bitMsb (*C enumerator*), 455
drivers_RT_RT1020_MIMXRT1024._flexio_spi_transfer_flags.kFLEXIO_SPI_16bitLsb (*C enumerator*), 455
drivers_RT_RT1020_MIMXRT1024._flexio_spi_transfer_flags.kFLEXIO_SPI_16bitMsb (*C enumerator*), 455
drivers_RT_RT1020_MIMXRT1024._flexio_spi_transfer_flags.kFLEXIO_SPI_32bitLsb (*C enumerator*), 455
drivers_RT_RT1020_MIMXRT1024._flexio_spi_transfer_flags.kFLEXIO_SPI_32bitMsb (*C enumerator*), 455
drivers_RT_RT1020_MIMXRT1024._flexio_spi_transfer_flags.kFLEXIO_SPI_csContinuous (*C enumerator*), 455
drivers_RT_RT1020_MIMXRT1024._flexio_spi_transfer.dataSize (*C var*), 458
drivers_RT_RT1020_MIMXRT1024._flexio_spi_transfer.flags (*C var*), 458
drivers_RT_RT1020_MIMXRT1024._flexio_spi_transfer.rxData (*C var*), 458
drivers_RT_RT1020_MIMXRT1024._flexio_spi_transfer.txData (*C var*), 458
drivers_RT_RT1020_MIMXRT1024._flexio_spi_type (*C struct*), 456
drivers_RT_RT1020_MIMXRT1024._flexio_spi_type.CSnPinIndex (*C var*), 457
drivers_RT_RT1020_MIMXRT1024._flexio_spi_type.flexioBase (*C var*), 457
drivers_RT_RT1020_MIMXRT1024._flexio_spi_type.SCKPinIndex (*C var*), 457
drivers_RT_RT1020_MIMXRT1024._flexio_spi_type.SDIPinIndex (*C var*), 457
drivers_RT_RT1020_MIMXRT1024._flexio_spi_type.SDOPinIndex (*C var*), 457
drivers_RT_RT1020_MIMXRT1024._flexio_spi_type.shifterIndex (*C var*), 457
drivers_RT_RT1020_MIMXRT1024._flexio_spi_type.timerIndex (*C var*), 457
drivers_RT_RT1020_MIMXRT1024._flexio_timer_config (*C struct*), 411
drivers_RT_RT1020_MIMXRT1024._flexio_timer_config.pinConfig (*C var*), 412
drivers_RT_RT1020_MIMXRT1024._flexio_timer_config.pinPolarity (*C var*), 412
drivers_RT_RT1020_MIMXRT1024._flexio_timer_config.pinSelect (*C var*), 412
drivers_RT_RT1020_MIMXRT1024._flexio_timer_config.timerCompare (*C var*), 412
drivers_RT_RT1020_MIMXRT1024._flexio_timer_config.timerDecrement (*C var*), 412
drivers_RT_RT1020_MIMXRT1024._flexio_timer_config.timerDisable (*C var*), 412
drivers_RT_RT1020_MIMXRT1024._flexio_timer_config.timerEnable (*C var*), 412
drivers_RT_RT1020_MIMXRT1024._flexio_timer_config.timerMode (*C var*), 412
drivers_RT_RT1020_MIMXRT1024._flexio_timer_config.timerOutput (*C var*), 412
drivers_RT_RT1020_MIMXRT1024._flexio_timer_config.timerReset (*C var*), 412
drivers_RT_RT1020_MIMXRT1024._flexio_timer_config.timerStart (*C var*), 412
drivers_RT_RT1020_MIMXRT1024._flexio_timer_config.timerStop (*C var*), 412
drivers_RT_RT1020_MIMXRT1024._flexio_timer_config.triggerPolarity (*C var*), 412
drivers_RT_RT1020_MIMXRT1024._flexio_timer_config.triggerSelect (*C var*), 411
drivers_RT_RT1020_MIMXRT1024._flexio_timer_config.triggerSource (*C var*), 412

drivers_RT_RT1020_MIMXRT1024._flexio_timer_decrement_source (*C enum*), 403
drivers_RT_RT1020_MIMXRT1024._flexio_timer_decrement_source.kFLEXIO_TimerDecSrcOnFlexIOClockShiftTi (*C enumerator*), 403
drivers_RT_RT1020_MIMXRT1024._flexio_timer_decrement_source.kFLEXIO_TimerDecSrcOnPinInputShiftPinInp (*C enumerator*), 404
drivers_RT_RT1020_MIMXRT1024._flexio_timer_decrement_source.kFLEXIO_TimerDecSrcOnTriggerInputShiftTi (*C enumerator*), 403
drivers_RT_RT1020_MIMXRT1024._flexio_timer_decrement_source.kFLEXIO_TimerDecSrcOnTriggerInputShiftTr (*C enumerator*), 404
drivers_RT_RT1020_MIMXRT1024._flexio_timer_disable_condition (*C enum*), 404
drivers_RT_RT1020_MIMXRT1024._flexio_timer_disable_condition.kFLEXIO_TimerDisableNever (*C enumerator*), 404
drivers_RT_RT1020_MIMXRT1024._flexio_timer_disable_condition.kFLEXIO_TimerDisableOnPinBothEdge (*C enumerator*), 404
drivers_RT_RT1020_MIMXRT1024._flexio_timer_disable_condition.kFLEXIO_TimerDisableOnPinBothEdgeTrigge (*C enumerator*), 405
drivers_RT_RT1020_MIMXRT1024._flexio_timer_disable_condition.kFLEXIO_TimerDisableOnPreTimerDisable (*C enumerator*), 404
drivers_RT_RT1020_MIMXRT1024._flexio_timer_disable_condition.kFLEXIO_TimerDisableOnTimerCompare (*C enumerator*), 404
drivers_RT_RT1020_MIMXRT1024._flexio_timer_disable_condition.kFLEXIO_TimerDisableOnTimerCompareTrigg (*C enumerator*), 404
drivers_RT_RT1020_MIMXRT1024._flexio_timer_disable_condition.kFLEXIO_TimerDisableOnTriggerFallingEdge (*C enumerator*), 405
drivers_RT_RT1020_MIMXRT1024._flexio_timer_enable_condition (*C enum*), 405
drivers_RT_RT1020_MIMXRT1024._flexio_timer_enable_condition.kFLEXIO_TimerEnabledAlways (*C enumerator*), 405
drivers_RT_RT1020_MIMXRT1024._flexio_timer_enable_condition.kFLEXIO_TimerEnableOnPinRisingEdge (*C enumerator*), 405
drivers_RT_RT1020_MIMXRT1024._flexio_timer_enable_condition.kFLEXIO_TimerEnableOnPinRisingEdgeTrigge (*C enumerator*), 405
drivers_RT_RT1020_MIMXRT1024._flexio_timer_enable_condition.kFLEXIO_TimerEnableOnPrevTimerEnable (*C enumerator*), 405
drivers_RT_RT1020_MIMXRT1024._flexio_timer_enable_condition.kFLEXIO_TimerEnableOnTriggerBothEdge (*C enumerator*), 405
drivers_RT_RT1020_MIMXRT1024._flexio_timer_enable_condition.kFLEXIO_TimerEnableOnTriggerHigh (*C enumerator*), 405
drivers_RT_RT1020_MIMXRT1024._flexio_timer_enable_condition.kFLEXIO_TimerEnableOnTriggerHighPinHigh (*C enumerator*), 405
drivers_RT_RT1020_MIMXRT1024._flexio_timer_enable_condition.kFLEXIO_TimerEnableOnTriggerRisingEdge (*C enumerator*), 405
drivers_RT_RT1020_MIMXRT1024._flexio_timer_mode (*C enum*), 403
drivers_RT_RT1020_MIMXRT1024._flexio_timer_mode.kFLEXIO_TimerModeDisabled (*C enumerator*), 403
drivers_RT_RT1020_MIMXRT1024._flexio_timer_mode.kFLEXIO_TimerModeDual8BitBaudBit (*C enumerator*), 403
drivers_RT_RT1020_MIMXRT1024._flexio_timer_mode.kFLEXIO_TimerModeDual8BitPWM (*C enumerator*), 403
drivers_RT_RT1020_MIMXRT1024._flexio_timer_mode.kFLEXIO_TimerModeDual8BitPWMLow (*C enumerator*), 403
drivers_RT_RT1020_MIMXRT1024._flexio_timer_mode.kFLEXIO_TimerModeSingle16Bit (*C enumerator*), 403
drivers_RT_RT1020_MIMXRT1024._flexio_timer_output (*C enum*), 403
drivers_RT_RT1020_MIMXRT1024._flexio_timer_output_state (*C enum*), 406
drivers_RT_RT1020_MIMXRT1024._flexio_timer_output_state.kFLEXIO_PwmHigh (*C enumerator*), 406
drivers_RT_RT1020_MIMXRT1024._flexio_timer_output_state.kFLEXIO_PwmLow (*C enumerator*), 406

drivers_RT_RT1020_MIMXRT1024._flexio_timer_output.kFLEXIO_TimerOutputOneAffectedByReset (*C enumerator*), 403
drivers_RT_RT1020_MIMXRT1024._flexio_timer_output.kFLEXIO_TimerOutputOneNotAffectedByReset (*C enumerator*), 403
drivers_RT_RT1020_MIMXRT1024._flexio_timer_output.kFLEXIO_TimerOutputZeroAffectedByReset (*C enumerator*), 403
drivers_RT_RT1020_MIMXRT1024._flexio_timer_output.kFLEXIO_TimerOutputZeroNotAffectedByReset (*C enumerator*), 403
drivers_RT_RT1020_MIMXRT1024._flexio_timer_reset_condition (*C enum*), 404
drivers_RT_RT1020_MIMXRT1024._flexio_timer_reset_condition.kFLEXIO_TimerResetNever (*C enumerator*), 404
drivers_RT_RT1020_MIMXRT1024._flexio_timer_reset_condition.kFLEXIO_TimerResetOnTimerPinEqualToTimer (*C enumerator*), 404
drivers_RT_RT1020_MIMXRT1024._flexio_timer_reset_condition.kFLEXIO_TimerResetOnTimerPinRisingEdge (*C enumerator*), 404
drivers_RT_RT1020_MIMXRT1024._flexio_timer_reset_condition.kFLEXIO_TimerResetOnTimerTriggerBothEdge (*C enumerator*), 404
drivers_RT_RT1020_MIMXRT1024._flexio_timer_reset_condition.kFLEXIO_TimerResetOnTimerTriggerEqualToTi (*C enumerator*), 404
drivers_RT_RT1020_MIMXRT1024._flexio_timer_reset_condition.kFLEXIO_TimerResetOnTimerTriggerRisingEdg (*C enumerator*), 404
drivers_RT_RT1020_MIMXRT1024._flexio_timer_start_bit_condition (*C enum*), 406
drivers_RT_RT1020_MIMXRT1024._flexio_timer_start_bit_condition.kFLEXIO_TimerStartBitDisabled (*C enumerator*), 406
drivers_RT_RT1020_MIMXRT1024._flexio_timer_start_bit_condition.kFLEXIO_TimerStartBitEnabled (*C enumerator*), 406
drivers_RT_RT1020_MIMXRT1024._flexio_timer_stop_bit_condition (*C enum*), 405
drivers_RT_RT1020_MIMXRT1024._flexio_timer_stop_bit_condition.kFLEXIO_TimerStopBitDisabled (*C enumerator*), 405
drivers_RT_RT1020_MIMXRT1024._flexio_timer_stop_bit_condition.kFLEXIO_TimerStopBitEnableOnTimerCom (*C enumerator*), 405
drivers_RT_RT1020_MIMXRT1024._flexio_timer_stop_bit_condition.kFLEXIO_TimerStopBitEnableOnTimerCom (*C enumerator*), 406
drivers_RT_RT1020_MIMXRT1024._flexio_timer_stop_bit_condition.kFLEXIO_TimerStopBitEnableOnTimerDisa (*C enumerator*), 405
drivers_RT_RT1020_MIMXRT1024._flexio_timer_trigger_polarity (*C enum*), 402
drivers_RT_RT1020_MIMXRT1024._flexio_timer_trigger_polarity.kFLEXIO_TimerTriggerPolarityActiveHigh (*C enumerator*), 402
drivers_RT_RT1020_MIMXRT1024._flexio_timer_trigger_polarity.kFLEXIO_TimerTriggerPolarityActiveLow (*C enumerator*), 402
drivers_RT_RT1020_MIMXRT1024._flexio_timer_trigger_source (*C enum*), 402
drivers_RT_RT1020_MIMXRT1024._flexio_timer_trigger_source.kFLEXIO_TimerTriggerSourceExternal (*C enumerator*), 402
drivers_RT_RT1020_MIMXRT1024._flexio_timer_trigger_source.kFLEXIO_TimerTriggerSourceInternal (*C enumerator*), 402
drivers_RT_RT1020_MIMXRT1024._flexio_uart_bit_count_per_char (*C enum*), 467
drivers_RT_RT1020_MIMXRT1024._flexio_uart_bit_count_per_char.kFLEXIO_UART_7BitsPerChar (*C enumerator*), 467
drivers_RT_RT1020_MIMXRT1024._flexio_uart_bit_count_per_char.kFLEXIO_UART_8BitsPerChar (*C enumerator*), 467
drivers_RT_RT1020_MIMXRT1024._flexio_uart_bit_count_per_char.kFLEXIO_UART_9BitsPerChar (*C enumerator*), 467
drivers_RT_RT1020_MIMXRT1024._flexio_uart_config (*C struct*), 469
drivers_RT_RT1020_MIMXRT1024._flexio_uart_config.baudRate_Bps (*C var*), 469
drivers_RT_RT1020_MIMXRT1024._flexio_uart_config.bitCountPerChar (*C var*), 469
drivers_RT_RT1020_MIMXRT1024._flexio_uart_config.enableFastAccess (*C var*), 469
drivers_RT_RT1020_MIMXRT1024._flexio_uart_config.enableInDebug (*C var*), 469
drivers_RT_RT1020_MIMXRT1024._flexio_uart_config.enableInDoze (*C var*), 469

drivers_RT_RT1020_MIMXRT1024._flexio_uart_config.enableUart (*C var*), 469
drivers_RT_RT1020_MIMXRT1024._flexio_uart_edma_handle (*C struct*), 423
drivers_RT_RT1020_MIMXRT1024._flexio_uart_edma_handle.callback (*C var*), 423
drivers_RT_RT1020_MIMXRT1024._flexio_uart_edma_handle.nbytes (*C var*), 423
drivers_RT_RT1020_MIMXRT1024._flexio_uart_edma_handle.rxDataSizeAll (*C var*), 423
drivers_RT_RT1020_MIMXRT1024._flexio_uart_edma_handle.rxEdmaHandle (*C var*), 423
drivers_RT_RT1020_MIMXRT1024._flexio_uart_edma_handle.rxState (*C var*), 424
drivers_RT_RT1020_MIMXRT1024._flexio_uart_edma_handle.txDataSizeAll (*C var*), 423
drivers_RT_RT1020_MIMXRT1024._flexio_uart_edma_handle.txEdmaHandle (*C var*), 423
drivers_RT_RT1020_MIMXRT1024._flexio_uart_edma_handle.txState (*C var*), 424
drivers_RT_RT1020_MIMXRT1024._flexio_uart_edma_handle.userData (*C var*), 423
drivers_RT_RT1020_MIMXRT1024._flexio_uart_handle (*C struct*), 469
drivers_RT_RT1020_MIMXRT1024._flexio_uart_handle.callback (*C var*), 470
drivers_RT_RT1020_MIMXRT1024._flexio_uart_handle.rxData (*C var*), 470
drivers_RT_RT1020_MIMXRT1024._flexio_uart_handle.rxDataSize (*C var*), 470
drivers_RT_RT1020_MIMXRT1024._flexio_uart_handle.rxDataSizeAll (*C var*), 470
drivers_RT_RT1020_MIMXRT1024._flexio_uart_handle.rxRingBuffer (*C var*), 470
drivers_RT_RT1020_MIMXRT1024._flexio_uart_handle.rxRingBufferHead (*C var*), 470
drivers_RT_RT1020_MIMXRT1024._flexio_uart_handle.rxRingBufferSize (*C var*), 470
drivers_RT_RT1020_MIMXRT1024._flexio_uart_handle.rxRingBufferTail (*C var*), 470
drivers_RT_RT1020_MIMXRT1024._flexio_uart_handle.rxState (*C var*), 470
drivers_RT_RT1020_MIMXRT1024._flexio_uart_handle.txData (*C var*), 470
drivers_RT_RT1020_MIMXRT1024._flexio_uart_handle.txDataSize (*C var*), 470
drivers_RT_RT1020_MIMXRT1024._flexio_uart_handle.txDataSizeAll (*C var*), 470
drivers_RT_RT1020_MIMXRT1024._flexio_uart_handle.txState (*C var*), 470
drivers_RT_RT1020_MIMXRT1024._flexio_uart_handle.userData (*C var*), 470
drivers_RT_RT1020_MIMXRT1024._flexio_uart_interrupt_enable (*C enum*), 467
drivers_RT_RT1020_MIMXRT1024._flexio_uart_interrupt_enable.kFLEXIO_UART_RxDataRegFullInterruptEnab (*C enumerator*), 467
drivers_RT_RT1020_MIMXRT1024._flexio_uart_interrupt_enable.kFLEXIO_UART_TxDataRegEmptyInterruptEn (*C enumerator*), 467
drivers_RT_RT1020_MIMXRT1024._flexio_uart_status_flags (*C enum*), 468
drivers_RT_RT1020_MIMXRT1024._flexio_uart_status_flags.kFLEXIO_UART_RxDataRegFullFlag (*C enumerator*), 468
drivers_RT_RT1020_MIMXRT1024._flexio_uart_status_flags.kFLEXIO_UART_RxOverRunFlag (*C enumerator*), 468
drivers_RT_RT1020_MIMXRT1024._flexio_uart_status_flags.kFLEXIO_UART_TxDataRegEmptyFlag (*C enumerator*), 468
drivers_RT_RT1020_MIMXRT1024._flexio_uart_transfer (*C struct*), 469
drivers_RT_RT1020_MIMXRT1024._flexio_uart_transfer.___unnamed100___ (*C union*), 470
drivers_RT_RT1020_MIMXRT1024._flexio_uart_transfer.___unnamed100___.data (*C var*), 470
drivers_RT_RT1020_MIMXRT1024._flexio_uart_transfer.___unnamed100___.rxData (*C var*), 471
drivers_RT_RT1020_MIMXRT1024._flexio_uart_transfer.___unnamed100___.txData (*C var*), 471
drivers_RT_RT1020_MIMXRT1024._flexio_uart_transfer.dataSize (*C var*), 469
drivers_RT_RT1020_MIMXRT1024._flexio_uart_type (*C struct*), 468
drivers_RT_RT1020_MIMXRT1024._flexio_uart_type.flexioBase (*C var*), 468
drivers_RT_RT1020_MIMXRT1024._flexio_uart_type.RxPinIndex (*C var*), 468
drivers_RT_RT1020_MIMXRT1024._flexio_uart_type.shifterIndex (*C var*), 469
drivers_RT_RT1020_MIMXRT1024._flexio_uart_type.timerIndex (*C var*), 469
drivers_RT_RT1020_MIMXRT1024._flexio_uart_type.TxPinIndex (*C var*), 468
drivers_RT_RT1020_MIMXRT1024._flexpwm_capture_channel (*C enum*), 508
drivers_RT_RT1020_MIMXRT1024._flexpwm_capture_channel_config (*C struct*), 528
drivers_RT_RT1020_MIMXRT1024._flexpwm_capture_channel_config.edge0 (*C var*), 528
drivers_RT_RT1020_MIMXRT1024._flexpwm_capture_channel_config.edge1 (*C var*), 528
drivers_RT_RT1020_MIMXRT1024._flexpwm_capture_channel_config.edgeCompareValue (*C var*), 528
drivers_RT_RT1020_MIMXRT1024._flexpwm_capture_channel_config.enableEdgeCounter (*C var*), 528

drivers_RT_RT1020_MIMXRT1024._flexpwm_capture_channel_config.fifoWatermark (*C var*), 529
drivers_RT_RT1020_MIMXRT1024._flexpwm_capture_channel_config.inputSelect (*C var*), 528
drivers_RT_RT1020_MIMXRT1024._flexpwm_capture_channel_config.oneshot (*C var*), 528
drivers_RT_RT1020_MIMXRT1024._flexpwm_capture_channel.kFLEXPWM_Capture_A (*C enumerator*), 508
drivers_RT_RT1020_MIMXRT1024._flexpwm_capture_channel.kFLEXPWM_Capture_B (*C enumerator*), 508
drivers_RT_RT1020_MIMXRT1024._flexpwm_capture_channel.kFLEXPWM_Capture_X (*C enumerator*), 508
drivers_RT_RT1020_MIMXRT1024._flexpwm_capture_edge (*C enum*), 507
drivers_RT_RT1020_MIMXRT1024._flexpwm_capture_edge.kFLEXPWM_CaptureEdge_Both (*C enumerator*), 507
drivers_RT_RT1020_MIMXRT1024._flexpwm_capture_edge.kFLEXPWM_CaptureEdge_Disabled (*C enumerator*), 507
drivers_RT_RT1020_MIMXRT1024._flexpwm_capture_edge.kFLEXPWM_CaptureEdge_Falling (*C enumerator*), 507
drivers_RT_RT1020_MIMXRT1024._flexpwm_capture_edge.kFLEXPWM_CaptureEdge_Rising (*C enumerator*), 507
drivers_RT_RT1020_MIMXRT1024._flexpwm_capture_fifo_watermark (*C enum*), 507
drivers_RT_RT1020_MIMXRT1024._flexpwm_capture_fifo_watermark.kFLEXPWM_CaptureFifoWatermark_1 (*C enumerator*), 507
drivers_RT_RT1020_MIMXRT1024._flexpwm_capture_fifo_watermark.kFLEXPWM_CaptureFifoWatermark_2 (*C enumerator*), 507
drivers_RT_RT1020_MIMXRT1024._flexpwm_capture_fifo_watermark.kFLEXPWM_CaptureFifoWatermark_3 (*C enumerator*), 507
drivers_RT_RT1020_MIMXRT1024._flexpwm_capture_fifo_watermark.kFLEXPWM_CaptureFifoWatermark_4 (*C enumerator*), 507
drivers_RT_RT1020_MIMXRT1024._flexpwm_capture_filter_channel_config (*C struct*), 529
drivers_RT_RT1020_MIMXRT1024._flexpwm_capture_filter_channel_config.filterCount (*C var*), 529
drivers_RT_RT1020_MIMXRT1024._flexpwm_capture_filter_channel_config.filterPeriod (*C var*), 529
drivers_RT_RT1020_MIMXRT1024._flexpwm_capture_filter_config (*C struct*), 529
drivers_RT_RT1020_MIMXRT1024._flexpwm_capture_filter_config.captureA (*C var*), 530
drivers_RT_RT1020_MIMXRT1024._flexpwm_capture_filter_config.captureB (*C var*), 530
drivers_RT_RT1020_MIMXRT1024._flexpwm_capture_filter_config.captureX (*C var*), 530
drivers_RT_RT1020_MIMXRT1024._flexpwm_capture_index (*C enum*), 508
drivers_RT_RT1020_MIMXRT1024._flexpwm_capture_index.kFLEXPWM_Capture_A_Edge0 (*C enumerator*), 508
drivers_RT_RT1020_MIMXRT1024._flexpwm_capture_index.kFLEXPWM_Capture_A_Edge1 (*C enumerator*), 508
drivers_RT_RT1020_MIMXRT1024._flexpwm_capture_index.kFLEXPWM_Capture_B_Edge0 (*C enumerator*), 508
drivers_RT_RT1020_MIMXRT1024._flexpwm_capture_index.kFLEXPWM_Capture_B_Edge1 (*C enumerator*), 508
drivers_RT_RT1020_MIMXRT1024._flexpwm_capture_index.kFLEXPWM_Capture_X_Edge0 (*C enumerator*), 508
drivers_RT_RT1020_MIMXRT1024._flexpwm_capture_index.kFLEXPWM_Capture_X_Edge1 (*C enumerator*), 508
drivers_RT_RT1020_MIMXRT1024._flexpwm_capture_input_select (*C enum*), 507
drivers_RT_RT1020_MIMXRT1024._flexpwm_capture_input_select.kFLEXPWM_CaptureInput_EdgeCounter (*C enumerator*), 508
drivers_RT_RT1020_MIMXRT1024._flexpwm_capture_input_select.kFLEXPWM_CaptureInput_RawSignal (*C enumerator*), 508
drivers_RT_RT1020_MIMXRT1024._flexpwm_clock_source (*C enum*), 500
drivers_RT_RT1020_MIMXRT1024._flexpwm_clock_source.kFLEXPWM_ClockSource_ExtClock (*C enumerator*), 501
drivers_RT_RT1020_MIMXRT1024._flexpwm_clock_source.kFLEXPWM_ClockSource_IPBusClock

(*C enumerator*), 500
drivers_RT_RT1020_MIMXRT1024._flexpwm_clock_source.kFLEXPWM_ClockSource_Submodule0Clock (*C enumerator*), 501
drivers_RT_RT1020_MIMXRT1024._flexpwm_compare_mode (*C enum*), 502
drivers_RT_RT1020_MIMXRT1024._flexpwm_compare_mode.kFLEXPWM_CompareMode_Equal (*C enumerator*), 502
drivers_RT_RT1020_MIMXRT1024._flexpwm_compare_mode.kFLEXPWM_CompareMode_EqualGreater (*C enumerator*), 502
drivers_RT_RT1020_MIMXRT1024._flexpwm_counter_config (*C struct*), 520
drivers_RT_RT1020_MIMXRT1024._flexpwm_counter_config.initSource (*C var*), 520
drivers_RT_RT1020_MIMXRT1024._flexpwm_counter_config.initValue (*C var*), 520
drivers_RT_RT1020_MIMXRT1024._flexpwm_counter_config.modValue (*C var*), 520
drivers_RT_RT1020_MIMXRT1024._flexpwm_dma_capture_source (*C enum*), 513
drivers_RT_RT1020_MIMXRT1024._flexpwm_dma_capture_source.kFLEXPWM_DMADisable (*C enumerator*), 513
drivers_RT_RT1020_MIMXRT1024._flexpwm_dma_capture_source.kFLEXPWM_DMAFIFOWatermark (*C enumerator*), 513
drivers_RT_RT1020_MIMXRT1024._flexpwm_dma_capture_source.kFLEXPWM_DMALocalReload (*C enumerator*), 513
drivers_RT_RT1020_MIMXRT1024._flexpwm_dma_capture_source.kFLEXPWM_DMALocalSync (*C enumerator*), 513
drivers_RT_RT1020_MIMXRT1024._flexpwm_dma_config (*C struct*), 530
drivers_RT_RT1020_MIMXRT1024._flexpwm_dma_config.captureSource (*C var*), 530
drivers_RT_RT1020_MIMXRT1024._flexpwm_dma_config.fifoWatermarkAND (*C var*), 530
drivers_RT_RT1020_MIMXRT1024._flexpwm_dma_enable (*C enum*), 513
drivers_RT_RT1020_MIMXRT1024._flexpwm_dma_enable.kFLEXPWM_DMA_CaptureA0Enable (*C enumerator*), 514
drivers_RT_RT1020_MIMXRT1024._flexpwm_dma_enable.kFLEXPWM_DMA_CaptureA1Enable (*C enumerator*), 514
drivers_RT_RT1020_MIMXRT1024._flexpwm_dma_enable.kFLEXPWM_DMA_CaptureB0Enable (*C enumerator*), 513
drivers_RT_RT1020_MIMXRT1024._flexpwm_dma_enable.kFLEXPWM_DMA_CaptureB1Enable (*C enumerator*), 513
drivers_RT_RT1020_MIMXRT1024._flexpwm_dma_enable.kFLEXPWM_DMA_CaptureX0Enable (*C enumerator*), 513
drivers_RT_RT1020_MIMXRT1024._flexpwm_dma_enable.kFLEXPWM_DMA_CaptureX1Enable (*C enumerator*), 513
drivers_RT_RT1020_MIMXRT1024._flexpwm_dma_enable.kFLEXPWM_DMA_ValueWriteEnable (*C enumerator*), 514
drivers_RT_RT1020_MIMXRT1024._flexpwm_fault_clearing_mode (*C enum*), 505
drivers_RT_RT1020_MIMXRT1024._flexpwm_fault_clearing_mode.kFLEXPWM_FaultClearingMode_Automatic (*C enumerator*), 505
drivers_RT_RT1020_MIMXRT1024._flexpwm_fault_clearing_mode.kFLEXPWM_FaultClearingMode_Manual (*C enumerator*), 505
drivers_RT_RT1020_MIMXRT1024._flexpwm_fault_config (*C struct*), 525
drivers_RT_RT1020_MIMXRT1024._flexpwm_fault_config.enableCombinatorialPath (*C var*), 526
drivers_RT_RT1020_MIMXRT1024._flexpwm_fault_config.enableFullCycleRecovery (*C var*), 526
drivers_RT_RT1020_MIMXRT1024._flexpwm_fault_config.enableHalfCycleRecovery (*C var*), 526
drivers_RT_RT1020_MIMXRT1024._flexpwm_fault_config.enableSafetyMode (*C var*), 526
drivers_RT_RT1020_MIMXRT1024._flexpwm_fault_config.faultClearingMode (*C var*), 525
drivers_RT_RT1020_MIMXRT1024._flexpwm_fault_config.faultInputActiveLevel (*C var*), 525
drivers_RT_RT1020_MIMXRT1024._flexpwm_fault_filter_config (*C struct*), 526
drivers_RT_RT1020_MIMXRT1024._flexpwm_fault_filter_config.enableGlitchStretch (*C var*), 527
drivers_RT_RT1020_MIMXRT1024._flexpwm_fault_filter_config.filterCount (*C var*), 526
drivers_RT_RT1020_MIMXRT1024._flexpwm_fault_filter_config.filterPeriod (*C var*), 526
drivers_RT_RT1020_MIMXRT1024._flexpwm_fault_interrupt_enable (*C enum*), 511
drivers_RT_RT1020_MIMXRT1024._flexpwm_fault_interrupt_enable.kFLEXPWM_Fault0InterruptEnable (*C enumerator*), 511

drivers_RT_RT1020_MIMXRT1024._flexpwm_fault_interrupt_enable.kFLEXPWM_Fault1InterruptEnable (*C enumerator*), 511
drivers_RT_RT1020_MIMXRT1024._flexpwm_fault_interrupt_enable.kFLEXPWM_Fault2InterruptEnable (*C enumerator*), 511
drivers_RT_RT1020_MIMXRT1024._flexpwm_fault_interrupt_enable.kFLEXPWM_Fault3InterruptEnable (*C enumerator*), 511
drivers_RT_RT1020_MIMXRT1024._flexpwm_fault_mask (*C enum*), 504
drivers_RT_RT1020_MIMXRT1024._flexpwm_fault_mask.kFLEXPWM_FaultMask_0 (*C enumerator*), 504
drivers_RT_RT1020_MIMXRT1024._flexpwm_fault_mask.kFLEXPWM_FaultMask_1 (*C enumerator*), 505
drivers_RT_RT1020_MIMXRT1024._flexpwm_fault_mask.kFLEXPWM_FaultMask_2 (*C enumerator*), 505
drivers_RT_RT1020_MIMXRT1024._flexpwm_fault_mask.kFLEXPWM_FaultMask_3 (*C enumerator*), 505
drivers_RT_RT1020_MIMXRT1024._flexpwm_fault_output_behavior (*C enum*), 504
drivers_RT_RT1020_MIMXRT1024._flexpwm_fault_output_behavior.kFLEXPWM_FaultOutput_Force0 (*C enumerator*), 504
drivers_RT_RT1020_MIMXRT1024._flexpwm_fault_output_behavior.kFLEXPWM_FaultOutput_Force1 (*C enumerator*), 504
drivers_RT_RT1020_MIMXRT1024._flexpwm_fault_output_behavior.kFLEXPWM_FaultOutput_HighZ (*C enumerator*), 504
drivers_RT_RT1020_MIMXRT1024._flexpwm_fault_output_config (*C struct*), 524
drivers_RT_RT1020_MIMXRT1024._flexpwm_fault_output_config.disableMask_ch0 (*C var*), 525
drivers_RT_RT1020_MIMXRT1024._flexpwm_fault_output_config.outputBehavior (*C var*), 525
drivers_RT_RT1020_MIMXRT1024._flexpwm_fault_pin_status (*C enum*), 512
drivers_RT_RT1020_MIMXRT1024._flexpwm_fault_pin_status.kFLEXPWM_Fault0PinActive (*C enumerator*), 513
drivers_RT_RT1020_MIMXRT1024._flexpwm_fault_pin_status.kFLEXPWM_Fault1PinActive (*C enumerator*), 513
drivers_RT_RT1020_MIMXRT1024._flexpwm_fault_pin_status.kFLEXPWM_Fault2PinActive (*C enumerator*), 513
drivers_RT_RT1020_MIMXRT1024._flexpwm_fault_pin_status.kFLEXPWM_Fault3PinActive (*C enumerator*), 513
drivers_RT_RT1020_MIMXRT1024._flexpwm_fault_status_flag (*C enum*), 511
drivers_RT_RT1020_MIMXRT1024._flexpwm_fault_status_flag.kFLEXPWM_Fault0Flag (*C enumerator*), 512
drivers_RT_RT1020_MIMXRT1024._flexpwm_fault_status_flag.kFLEXPWM_Fault1Flag (*C enumerator*), 512
drivers_RT_RT1020_MIMXRT1024._flexpwm_fault_status_flag.kFLEXPWM_Fault2Flag (*C enumerator*), 512
drivers_RT_RT1020_MIMXRT1024._flexpwm_fault_status_flag.kFLEXPWM_Fault3Flag (*C enumerator*), 512
drivers_RT_RT1020_MIMXRT1024._flexpwm_fault_submodule_config (*C struct*), 525
drivers_RT_RT1020_MIMXRT1024._flexpwm_fault_submodule_config.pwma (*C var*), 525
drivers_RT_RT1020_MIMXRT1024._flexpwm_fault_submodule_config.pwmb (*C var*), 525
drivers_RT_RT1020_MIMXRT1024._flexpwm_fault_submodule_config.pwmx (*C var*), 525
drivers_RT_RT1020_MIMXRT1024._flexpwm_force_out_channel_config (*C struct*), 523
drivers_RT_RT1020_MIMXRT1024._flexpwm_force_out_channel_config.softwareValue (*C var*), 524
drivers_RT_RT1020_MIMXRT1024._flexpwm_force_out_channel_config.source (*C var*), 523
drivers_RT_RT1020_MIMXRT1024._flexpwm_force_out_config (*C struct*), 524
drivers_RT_RT1020_MIMXRT1024._flexpwm_force_out_config.forceSource (*C var*), 524
drivers_RT_RT1020_MIMXRT1024._flexpwm_force_out_config.pwma (*C var*), 524
drivers_RT_RT1020_MIMXRT1024._flexpwm_force_out_config.pwma_initialValue (*C var*), 524
drivers_RT_RT1020_MIMXRT1024._flexpwm_force_out_config.pwmb (*C var*), 524
drivers_RT_RT1020_MIMXRT1024._flexpwm_force_out_config.pwmb_initialValue (*C var*), 524
drivers_RT_RT1020_MIMXRT1024._flexpwm_force_out_config.pwmx_initialValue (*C var*), 524
drivers_RT_RT1020_MIMXRT1024._flexpwm_force_output_source (*C enum*), 503

drivers_RT_RT1020_MIMXRT1024._flexpwm_force_output_source.kFLEXPWM_ForceOutputSource_ExtForce (*C enumerator*), 503
drivers_RT_RT1020_MIMXRT1024._flexpwm_force_output_source.kFLEXPWM_ForceOutputSource_ExtSync (*C enumerator*), 503
drivers_RT_RT1020_MIMXRT1024._flexpwm_force_output_source.kFLEXPWM_ForceOutputSource_LocalForce (*C enumerator*), 503
drivers_RT_RT1020_MIMXRT1024._flexpwm_force_output_source.kFLEXPWM_ForceOutputSource_LocalReload (*C enumerator*), 503
drivers_RT_RT1020_MIMXRT1024._flexpwm_force_output_source.kFLEXPWM_ForceOutputSource_LocalSync (*C enumerator*), 503
drivers_RT_RT1020_MIMXRT1024._flexpwm_force_output_source.kFLEXPWM_ForceOutputSource_MasterForce (*C enumerator*), 503
drivers_RT_RT1020_MIMXRT1024._flexpwm_force_output_source.kFLEXPWM_ForceOutputSource_MasterReloa (*C enumerator*), 503
drivers_RT_RT1020_MIMXRT1024._flexpwm_force_output_source.kFLEXPWM_ForceOutputSource_MasterSync (*C enumerator*), 503
drivers_RT_RT1020_MIMXRT1024._flexpwm_force_pwm_source (*C enum*), 504
drivers_RT_RT1020_MIMXRT1024._flexpwm_force_pwm_source.kFLEXPWM_ForcePwmSource_External (*C enumerator*), 504
drivers_RT_RT1020_MIMXRT1024._flexpwm_force_pwm_source.kFLEXPWM_ForcePwmSource_Generated (*C enumerator*), 504
drivers_RT_RT1020_MIMXRT1024._flexpwm_force_pwm_source.kFLEXPWM_ForcePwmSource_Inverted (*C enumerator*), 504
drivers_RT_RT1020_MIMXRT1024._flexpwm_force_pwm_source.kFLEXPWM_ForcePwmSource_Software (*C enumerator*), 504
drivers_RT_RT1020_MIMXRT1024._flexpwm_init_source (*C enum*), 502
drivers_RT_RT1020_MIMXRT1024._flexpwm_init_source.kFLEXPWM_InitSource_ExtSync (*C enumerator*), 502
drivers_RT_RT1020_MIMXRT1024._flexpwm_init_source.kFLEXPWM_InitSource_LocalSync (*C enumerator*), 502
drivers_RT_RT1020_MIMXRT1024._flexpwm_init_source.kFLEXPWM_InitSource_MasterReload (*C enumerator*), 502
drivers_RT_RT1020_MIMXRT1024._flexpwm_init_source.kFLEXPWM_InitSource_MasterSync (*C enumerator*), 502
drivers_RT_RT1020_MIMXRT1024._flexpwm_input_capture_config (*C struct*), 529
drivers_RT_RT1020_MIMXRT1024._flexpwm_input_capture_config.captureA (*C var*), 529
drivers_RT_RT1020_MIMXRT1024._flexpwm_input_capture_config.captureB (*C var*), 529
drivers_RT_RT1020_MIMXRT1024._flexpwm_input_capture_config.captureX (*C var*), 529
drivers_RT_RT1020_MIMXRT1024._flexpwm_ipol_source (*C enum*), 503
drivers_RT_RT1020_MIMXRT1024._flexpwm_ipol_source.kFLEXPWM_IPOL_PWM23 (*C enumerator*), 503
drivers_RT_RT1020_MIMXRT1024._flexpwm_ipol_source.kFLEXPWM_IPOL_PWM45 (*C enumerator*), 503
drivers_RT_RT1020_MIMXRT1024._flexpwm_load_mode (*C enum*), 501
drivers_RT_RT1020_MIMXRT1024._flexpwm_load_mode.kFLEXPWM_LoadMode_Immediate (*C enumerator*), 501
drivers_RT_RT1020_MIMXRT1024._flexpwm_load_mode.kFLEXPWM_LoadMode_Opportunity (*C enumerator*), 501
drivers_RT_RT1020_MIMXRT1024._flexpwm_output_trigger_config (*C struct*), 527
drivers_RT_RT1020_MIMXRT1024._flexpwm_output_trigger_config.muxTrig0Source (*C var*), 527
drivers_RT_RT1020_MIMXRT1024._flexpwm_output_trigger_config.muxTrig1Source (*C var*), 527
drivers_RT_RT1020_MIMXRT1024._flexpwm_output_trigger_config.outTriggerEnable (*C var*), 527
drivers_RT_RT1020_MIMXRT1024._flexpwm_output_trigger_config.stretchPrescaler (*C var*), 528
drivers_RT_RT1020_MIMXRT1024._flexpwm_output_trigger_config.triggerFrequency (*C var*), 528
drivers_RT_RT1020_MIMXRT1024._flexpwm_output_trigger_mask (*C enum*), 506
drivers_RT_RT1020_MIMXRT1024._flexpwm_output_trigger_mask.kFLEXPWM_OutputTriggerMask_VAL0 (*C enumerator*), 506
drivers_RT_RT1020_MIMXRT1024._flexpwm_output_trigger_mask.kFLEXPWM_OutputTriggerMask_VAL1

(*C enumerator*), 506
drivers_RT_RT1020_MIMXRT1024._flexpwm_output_trigger_mask.kFLEXPWM_OutputTriggerMask_VAL2 (*C enumerator*), 506
drivers_RT_RT1020_MIMXRT1024._flexpwm_output_trigger_mask.kFLEXPWM_OutputTriggerMask_VAL3 (*C enumerator*), 506
drivers_RT_RT1020_MIMXRT1024._flexpwm_output_trigger_mask.kFLEXPWM_OutputTriggerMask_VAL4 (*C enumerator*), 507
drivers_RT_RT1020_MIMXRT1024._flexpwm_output_trigger_mask.kFLEXPWM_OutputTriggerMask_VAL5 (*C enumerator*), 507
drivers_RT_RT1020_MIMXRT1024._flexpwm_prescaler (*C enum*), 501
drivers_RT_RT1020_MIMXRT1024._flexpwm_prescaler.kFLEXPWM_Prescale_Divide_1 (*C enumerator*), 501
drivers_RT_RT1020_MIMXRT1024._flexpwm_prescaler.kFLEXPWM_Prescale_Divide_2 (*C enumerator*), 501
drivers_RT_RT1020_MIMXRT1024._flexpwm_prescaler.kFLEXPWM_Prescale_Divide_4 (*C enumerator*), 501
drivers_RT_RT1020_MIMXRT1024._flexpwm_prescaler.kFLEXPWM_Prescale_Divide_8 (*C enumerator*), 501
drivers_RT_RT1020_MIMXRT1024._flexpwm_prescaler.kFLEXPWM_Prescale_Divide_16 (*C enumerator*), 501
drivers_RT_RT1020_MIMXRT1024._flexpwm_prescaler.kFLEXPWM_Prescale_Divide_32 (*C enumerator*), 501
drivers_RT_RT1020_MIMXRT1024._flexpwm_prescaler.kFLEXPWM_Prescale_Divide_64 (*C enumerator*), 501
drivers_RT_RT1020_MIMXRT1024._flexpwm_prescaler.kFLEXPWM_Prescale_Divide_128 (*C enumerator*), 501
drivers_RT_RT1020_MIMXRT1024._flexpwm_pwm_channel_config (*C struct*), 522
drivers_RT_RT1020_MIMXRT1024._flexpwm_pwm_channel_config.compareValue_OFF (*C var*), 522
drivers_RT_RT1020_MIMXRT1024._flexpwm_pwm_channel_config.compareValue_ON (*C var*), 522
drivers_RT_RT1020_MIMXRT1024._flexpwm_pwm_channel_config.polarity (*C var*), 522
drivers_RT_RT1020_MIMXRT1024._flexpwm_pwm_config (*C struct*), 523
drivers_RT_RT1020_MIMXRT1024._flexpwm_pwm_config.complementary (*C var*), 523
drivers_RT_RT1020_MIMXRT1024._flexpwm_pwm_config.ipolSource (*C var*), 523
drivers_RT_RT1020_MIMXRT1024._flexpwm_pwm_config.pwma (*C var*), 523
drivers_RT_RT1020_MIMXRT1024._flexpwm_pwm_config.pwmb (*C var*), 523
drivers_RT_RT1020_MIMXRT1024._flexpwm_pwm_polarity (*C enum*), 503
drivers_RT_RT1020_MIMXRT1024._flexpwm_pwm_polarity.kFLEXPWM_Polarity_ActiveHigh (*C enumerator*), 503
drivers_RT_RT1020_MIMXRT1024._flexpwm_pwm_polarity.kFLEXPWM_Polarity_ActiveLow (*C enumerator*), 503
drivers_RT_RT1020_MIMXRT1024._flexpwm_reload_config (*C struct*), 520
drivers_RT_RT1020_MIMXRT1024._flexpwm_reload_config.enableFullCycleReload (*C var*), 521
drivers_RT_RT1020_MIMXRT1024._flexpwm_reload_config.enableHalfCycleReload (*C var*), 521
drivers_RT_RT1020_MIMXRT1024._flexpwm_reload_config.halfCycleValue (*C var*), 521
drivers_RT_RT1020_MIMXRT1024._flexpwm_reload_config.loadFrequency (*C var*), 521
drivers_RT_RT1020_MIMXRT1024._flexpwm_reload_config.loadMode (*C var*), 521
drivers_RT_RT1020_MIMXRT1024._flexpwm_reload_config.reloadSource (*C var*), 521
drivers_RT_RT1020_MIMXRT1024._flexpwm_reload_source (*C enum*), 501
drivers_RT_RT1020_MIMXRT1024._flexpwm_reload_source.kFLEXPWM_ReloadSource_LocalReload (*C enumerator*), 502
drivers_RT_RT1020_MIMXRT1024._flexpwm_reload_source.kFLEXPWM_ReloadSource_MasterReload (*C enumerator*), 502
drivers_RT_RT1020_MIMXRT1024._flexpwm_stretch_prescaler (*C enum*), 505
drivers_RT_RT1020_MIMXRT1024._flexpwm_stretch_prescaler.kFLEXPWM_StretchPrescaler_2Clocks (*C enumerator*), 505
drivers_RT_RT1020_MIMXRT1024._flexpwm_stretch_prescaler.kFLEXPWM_StretchPrescaler_4Clocks (*C enumerator*), 506

drivers_RT_RT1020_MIMXRT1024._flexpwm_stretch_prescaler.kFLEXPWM_StretchPrescaler_8Clocks (*C enumerator*), 506
drivers_RT_RT1020_MIMXRT1024._flexpwm_stretch_prescaler.kFLEXPWM_StretchPrescaler_None (*C enumerator*), 505
drivers_RT_RT1020_MIMXRT1024._flexpwm_submodule_config (*C struct*), 521
drivers_RT_RT1020_MIMXRT1024._flexpwm_submodule_config.clockSource (*C var*), 522
drivers_RT_RT1020_MIMXRT1024._flexpwm_submodule_config.counterConfig (*C var*), 522
drivers_RT_RT1020_MIMXRT1024._flexpwm_submodule_config.enableDebugMode (*C var*), 522
drivers_RT_RT1020_MIMXRT1024._flexpwm_submodule_config.prescaler (*C var*), 522
drivers_RT_RT1020_MIMXRT1024._flexpwm_submodule_config.reloadConfig (*C var*), 522
drivers_RT_RT1020_MIMXRT1024._flexpwm_submodule_interrupt_enable (*C enum*), 509
drivers_RT_RT1020_MIMXRT1024._flexpwm_submodule_interrupt_enable.kFLEXPWM_CaptureA0InterruptEnab (*C enumerator*), 509
drivers_RT_RT1020_MIMXRT1024._flexpwm_submodule_interrupt_enable.kFLEXPWM_CaptureA1InterruptEnab (*C enumerator*), 509
drivers_RT_RT1020_MIMXRT1024._flexpwm_submodule_interrupt_enable.kFLEXPWM_CaptureB0InterruptEnab (*C enumerator*), 509
drivers_RT_RT1020_MIMXRT1024._flexpwm_submodule_interrupt_enable.kFLEXPWM_CaptureB1InterruptEnab (*C enumerator*), 509
drivers_RT_RT1020_MIMXRT1024._flexpwm_submodule_interrupt_enable.kFLEXPWM_CaptureX0InterruptEnab (*C enumerator*), 509
drivers_RT_RT1020_MIMXRT1024._flexpwm_submodule_interrupt_enable.kFLEXPWM_CaptureX1InterruptEnab (*C enumerator*), 509
drivers_RT_RT1020_MIMXRT1024._flexpwm_submodule_interrupt_enable.kFLEXPWM_Compare0InterruptEnabl (*C enumerator*), 509
drivers_RT_RT1020_MIMXRT1024._flexpwm_submodule_interrupt_enable.kFLEXPWM_Compare1InterruptEnabl (*C enumerator*), 509
drivers_RT_RT1020_MIMXRT1024._flexpwm_submodule_interrupt_enable.kFLEXPWM_Compare2InterruptEnabl (*C enumerator*), 509
drivers_RT_RT1020_MIMXRT1024._flexpwm_submodule_interrupt_enable.kFLEXPWM_Compare3InterruptEnabl (*C enumerator*), 509
drivers_RT_RT1020_MIMXRT1024._flexpwm_submodule_interrupt_enable.kFLEXPWM_Compare4InterruptEnabl (*C enumerator*), 509
drivers_RT_RT1020_MIMXRT1024._flexpwm_submodule_interrupt_enable.kFLEXPWM_Compare5InterruptEnabl (*C enumerator*), 509
drivers_RT_RT1020_MIMXRT1024._flexpwm_submodule_interrupt_enable.kFLEXPWM_ReloadErrorInterruptEna (*C enumerator*), 509
drivers_RT_RT1020_MIMXRT1024._flexpwm_submodule_interrupt_enable.kFLEXPWM_ReloadInterruptEnable (*C enumerator*), 509
drivers_RT_RT1020_MIMXRT1024._flexpwm_submodule_mask (*C enum*), 502
drivers_RT_RT1020_MIMXRT1024._flexpwm_submodule_mask.kFLEXPWM_SubmoduleMask_0 (*C enumerator*), 502
drivers_RT_RT1020_MIMXRT1024._flexpwm_submodule_mask.kFLEXPWM_SubmoduleMask_1 (*C enumerator*), 502
drivers_RT_RT1020_MIMXRT1024._flexpwm_submodule_mask.kFLEXPWM_SubmoduleMask_2 (*C enumerator*), 502
drivers_RT_RT1020_MIMXRT1024._flexpwm_submodule_status_flag (*C enum*), 510
drivers_RT_RT1020_MIMXRT1024._flexpwm_submodule_status_flag.kFLEXPWM_CaptureA0Flag (*C enumerator*), 510
drivers_RT_RT1020_MIMXRT1024._flexpwm_submodule_status_flag.kFLEXPWM_CaptureA1Flag (*C enumerator*), 510
drivers_RT_RT1020_MIMXRT1024._flexpwm_submodule_status_flag.kFLEXPWM_CaptureB0Flag (*C enumerator*), 510
drivers_RT_RT1020_MIMXRT1024._flexpwm_submodule_status_flag.kFLEXPWM_CaptureB1Flag (*C enumerator*), 510
drivers_RT_RT1020_MIMXRT1024._flexpwm_submodule_status_flag.kFLEXPWM_CaptureX0Flag (*C enumerator*), 510
drivers_RT_RT1020_MIMXRT1024._flexpwm_submodule_status_flag.kFLEXPWM_CaptureX1Flag

(*C enumerator*), 510
drivers_RT_RT1020_MIMXRT1024._flexpwm_submodule_status_flag.kFLEXPWM_Compare0Flag (*C enumerator*), 510
drivers_RT_RT1020_MIMXRT1024._flexpwm_submodule_status_flag.kFLEXPWM_Compare1Flag (*C enumerator*), 510
drivers_RT_RT1020_MIMXRT1024._flexpwm_submodule_status_flag.kFLEXPWM_Compare2Flag (*C enumerator*), 510
drivers_RT_RT1020_MIMXRT1024._flexpwm_submodule_status_flag.kFLEXPWM_Compare3Flag (*C enumerator*), 510
drivers_RT_RT1020_MIMXRT1024._flexpwm_submodule_status_flag.kFLEXPWM_Compare4Flag (*C enumerator*), 510
drivers_RT_RT1020_MIMXRT1024._flexpwm_submodule_status_flag.kFLEXPWM_Compare5Flag (*C enumerator*), 510
drivers_RT_RT1020_MIMXRT1024._flexpwm_submodule_status_flag.kFLEXPWM_RegisterUpdatedFlag (*C enumerator*), 511
drivers_RT_RT1020_MIMXRT1024._flexpwm_submodule_status_flag.kFLEXPWM_ReloadErrorFlag (*C enumerator*), 511
drivers_RT_RT1020_MIMXRT1024._flexpwm_submodule_status_flag.kFLEXPWM_ReloadFlag (*C enumerator*), 510
drivers_RT_RT1020_MIMXRT1024._flexpwm_trigger_frequency (*C enum*), 506
drivers_RT_RT1020_MIMXRT1024._flexpwm_trigger_frequency.kFLEXPWM_TriggerFrequency_EveryCycle (*C enumerator*), 506
drivers_RT_RT1020_MIMXRT1024._flexpwm_trigger_frequency.kFLEXPWM_TriggerFrequency_OnReload (*C enumerator*), 506
drivers_RT_RT1020_MIMXRT1024._flexpwm_trigger_mux_source (*C enum*), 506
drivers_RT_RT1020_MIMXRT1024._flexpwm_trigger_mux_source.kFLEXPWM_TriggerMuxSource_GeneratedTrig (*C enumerator*), 506
drivers_RT_RT1020_MIMXRT1024._flexpwm_trigger_mux_source.kFLEXPWM_TriggerMuxSource_PWMOutput (*C enumerator*), 506
drivers_RT_RT1020_MIMXRT1024._flexpwm_write_protect (*C enum*), 505
drivers_RT_RT1020_MIMXRT1024._flexpwm_write_protect.kFLEXPWM_WriteProtect_Off (*C enumerator*), 505
drivers_RT_RT1020_MIMXRT1024._flexpwm_write_protect.kFLEXPWM_WriteProtect_OffLocked (*C enumerator*), 505
drivers_RT_RT1020_MIMXRT1024._flexpwm_write_protect.kFLEXPWM_WriteProtect_On (*C enumerator*), 505
drivers_RT_RT1020_MIMXRT1024._flexpwm_write_protect.kFLEXPWM_WriteProtect_OnLocked (*C enumerator*), 505
drivers_RT_RT1020_MIMXRT1024._flexram_dtcm_ecc_multi_error_info (*C struct*), 539
drivers_RT_RT1020_MIMXRT1024._flexram_dtcm_ecc_multi_error_info.DtcmMultiErrorAddr (*C var*), 539
drivers_RT_RT1020_MIMXRT1024._flexram_dtcm_ecc_multi_error_info.DtcmMultiErrorData (*C var*), 539
drivers_RT_RT1020_MIMXRT1024._flexram_dtcm_ecc_multi_error_info.DtcmMultiErrorInfo (*C var*), 539
drivers_RT_RT1020_MIMXRT1024._flexram_dtcm_ecc_single_error_info (*C struct*), 539
drivers_RT_RT1020_MIMXRT1024._flexram_dtcm_ecc_single_error_info.DtcmSingleErrorAddr (*C var*), 539
drivers_RT_RT1020_MIMXRT1024._flexram_dtcm_ecc_single_error_info.DtcmSingleErrorData (*C var*), 539
drivers_RT_RT1020_MIMXRT1024._flexram_dtcm_ecc_single_error_info.DtcmSingleErrorInfo (*C var*), 539
drivers_RT_RT1020_MIMXRT1024._flexram_ecc_error_type (*C struct*), 536
drivers_RT_RT1020_MIMXRT1024._flexram_ecc_error_type.Fource1BitDataInversion (*C var*), 537
drivers_RT_RT1020_MIMXRT1024._flexram_ecc_error_type.FourceConti1BitDataInversion (*C var*), 537
drivers_RT_RT1020_MIMXRT1024._flexram_ecc_error_type.FourceContiNCDataInversion (*C var*), 537

drivers_RT_RT1020_MIMXRT1024._flexram_ecc_error_type.FourceOneNCDataInversion (*C var*), 537
drivers_RT_RT1020_MIMXRT1024._flexram_ecc_error_type.SecondBitPos (*C var*), 537
drivers_RT_RT1020_MIMXRT1024._flexram_ecc_error_type.SingleBitPos (*C var*), 537
drivers_RT_RT1020_MIMXRT1024._flexram_itcm_ecc_multi_error_info (*C struct*), 538
drivers_RT_RT1020_MIMXRT1024._flexram_itcm_ecc_multi_error_info.ItcmMultiErrorAddr (*C var*), 538
drivers_RT_RT1020_MIMXRT1024._flexram_itcm_ecc_multi_error_info.ItcmMultiErrorDataLSB (*C var*), 538
drivers_RT_RT1020_MIMXRT1024._flexram_itcm_ecc_multi_error_info.ItcmMultiErrorDataMSB (*C var*), 538
drivers_RT_RT1020_MIMXRT1024._flexram_itcm_ecc_multi_error_info.ItcmMultiErrorInfo (*C var*), 538
drivers_RT_RT1020_MIMXRT1024._flexram_itcm_ecc_single_error_info (*C struct*), 538
drivers_RT_RT1020_MIMXRT1024._flexram_itcm_ecc_single_error_info.ItcmSingleErrorAddr (*C var*), 538
drivers_RT_RT1020_MIMXRT1024._flexram_itcm_ecc_single_error_info.ItcmSingleErrorDataLSB (*C var*), 538
drivers_RT_RT1020_MIMXRT1024._flexram_itcm_ecc_single_error_info.ItcmSingleErrorDataMSB (*C var*), 538
drivers_RT_RT1020_MIMXRT1024._flexram_itcm_ecc_single_error_info.ItcmSingleErrorInfo (*C var*), 538
drivers_RT_RT1020_MIMXRT1024._flexram_memory_type (*C enum*), 533
drivers_RT_RT1020_MIMXRT1024._flexram_memory_type.kFLEXRAM_D0TCM (*C enumerator*), 533
drivers_RT_RT1020_MIMXRT1024._flexram_memory_type.kFLEXRAM_D1TCM (*C enumerator*), 533
drivers_RT_RT1020_MIMXRT1024._flexram_memory_type.kFLEXRAM_ITCM (*C enumerator*), 533
drivers_RT_RT1020_MIMXRT1024._flexram_memory_type.kFLEXRAM_OCRAM (*C enumerator*), 533
drivers_RT_RT1020_MIMXRT1024._flexram_ocram_ecc_multi_error_info (*C struct*), 537
drivers_RT_RT1020_MIMXRT1024._flexram_ocram_ecc_multi_error_info.OcramMultiErrorAddr (*C var*), 538
drivers_RT_RT1020_MIMXRT1024._flexram_ocram_ecc_multi_error_info.OcramMultiErrorDataLSB (*C var*), 538
drivers_RT_RT1020_MIMXRT1024._flexram_ocram_ecc_multi_error_info.OcramMultiErrorDataMSB (*C var*), 538
drivers_RT_RT1020_MIMXRT1024._flexram_ocram_ecc_multi_error_info.OcramMultiErrorInfo (*C var*), 538
drivers_RT_RT1020_MIMXRT1024._flexram_ocram_ecc_single_error_info (*C struct*), 537
drivers_RT_RT1020_MIMXRT1024._flexram_ocram_ecc_single_error_info.OcramSingleErrorAddr (*C var*), 537
drivers_RT_RT1020_MIMXRT1024._flexram_ocram_ecc_single_error_info.OcramSingleErrorDataLSB (*C var*), 537
drivers_RT_RT1020_MIMXRT1024._flexram_ocram_ecc_single_error_info.OcramSingleErrorDataMSB (*C var*), 537
drivers_RT_RT1020_MIMXRT1024._flexram_ocram_ecc_single_error_info.OcramSingleErrorInfo (*C var*), 537
drivers_RT_RT1020_MIMXRT1024._flexram_tcm_access_mode (*C enum*), 532
drivers_RT_RT1020_MIMXRT1024._flexram_tcm_access_mode.kFLEXRAM_TCMAccessFastMode (*C enumerator*), 533
drivers_RT_RT1020_MIMXRT1024._flexram_tcm_access_mode.kFLEXRAM_TCMAccessWaitMode (*C enumerator*), 533
drivers_RT_RT1020_MIMXRT1024._flexspi_addr_map_config (*C struct*), 557
drivers_RT_RT1020_MIMXRT1024._flexspi_addr_map_config.addrEnd (*C var*), 558
drivers_RT_RT1020_MIMXRT1024._flexspi_addr_map_config.addrOffset (*C var*), 558
drivers_RT_RT1020_MIMXRT1024._flexspi_addr_map_config.addrStart (*C var*), 558

drivers_RT_RT1020_MIMXRT1024._flexspi_addr_map_config.remapEnable (*C var*), 558
drivers_RT_RT1020_MIMXRT1024._flexspi_ahb_error_code (*C enum*), 551
drivers_RT_RT1020_MIMXRT1024._flexspi_ahb_error_code.kFLEXSPI_AhbCmdErrorDdrDummyInSdrSequence (*C enumerator*), 551
drivers_RT_RT1020_MIMXRT1024._flexspi_ahb_error_code.kFLEXSPI_AhbCmdErrorJumpOnCsInWriteCmd (*C enumerator*), 551
drivers_RT_RT1020_MIMXRT1024._flexspi_ahb_error_code.kFLEXSPI_AhbCmdErrorNoError (*C enumerator*), 551
drivers_RT_RT1020_MIMXRT1024._flexspi_ahb_error_code.kFLEXSPI_AhbCmdErrorSdrDummyInDdrSequence (*C enumerator*), 551
drivers_RT_RT1020_MIMXRT1024._flexspi_ahb_error_code.kFLEXSPI_AhbCmdErrorUnknownOpCode (*C enumerator*), 551
drivers_RT_RT1020_MIMXRT1024._flexspi_ahb_error_code.kFLEXSPI_AhbCmdSequenceExecutionTimeout (*C enumerator*), 552
drivers_RT_RT1020_MIMXRT1024._flexspi_ahb_write_wait_unit (*C enum*), 550
drivers_RT_RT1020_MIMXRT1024._flexspi_ahb_write_wait_unit.kFLEXSPI_AhbWriteWaitUnit2AhbCycle (*C enumerator*), 550
drivers_RT_RT1020_MIMXRT1024._flexspi_ahb_write_wait_unit.kFLEXSPI_AhbWriteWaitUnit8AhbCycle (*C enumerator*), 550
drivers_RT_RT1020_MIMXRT1024._flexspi_ahb_write_wait_unit.kFLEXSPI_AhbWriteWaitUnit32AhbCycle (*C enumerator*), 550
drivers_RT_RT1020_MIMXRT1024._flexspi_ahb_write_wait_unit.kFLEXSPI_AhbWriteWaitUnit128AhbCycle (*C enumerator*), 550
drivers_RT_RT1020_MIMXRT1024._flexspi_ahb_write_wait_unit.kFLEXSPI_AhbWriteWaitUnit512AhbCycle (*C enumerator*), 550
drivers_RT_RT1020_MIMXRT1024._flexspi_ahb_write_wait_unit.kFLEXSPI_AhbWriteWaitUnit2048AhbCycle (*C enumerator*), 550
drivers_RT_RT1020_MIMXRT1024._flexspi_ahb_write_wait_unit.kFLEXSPI_AhbWriteWaitUnit8192AhbCycle (*C enumerator*), 550
drivers_RT_RT1020_MIMXRT1024._flexspi_ahb_write_wait_unit.kFLEXSPI_AhbWriteWaitUnit32768AhbCycle (*C enumerator*), 550
drivers_RT_RT1020_MIMXRT1024._flexspi_ahbBuffer_config (*C struct*), 554
drivers_RT_RT1020_MIMXRT1024._flexspi_ahbBuffer_config.bufferSize (*C var*), 554
drivers_RT_RT1020_MIMXRT1024._flexspi_ahbBuffer_config.enablePrefetch (*C var*), 554
drivers_RT_RT1020_MIMXRT1024._flexspi_ahbBuffer_config.masterIndex (*C var*), 554
drivers_RT_RT1020_MIMXRT1024._flexspi_ahbBuffer_config.priority (*C var*), 554
drivers_RT_RT1020_MIMXRT1024._flexspi_ahbBuffers_ctrl (*C struct*), 554
drivers_RT_RT1020_MIMXRT1024._flexspi_ahbBuffers_ctrl.buffer (*C var*), 554
drivers_RT_RT1020_MIMXRT1024._flexspi_arb_command_source (*C enum*), 552
drivers_RT_RT1020_MIMXRT1024._flexspi_arb_command_source.kFLEXSPI_AhbReadCommand (*C enumerator*), 552
drivers_RT_RT1020_MIMXRT1024._flexspi_arb_command_source.kFLEXSPI_AhbWriteCommand (*C enumerator*), 552
drivers_RT_RT1020_MIMXRT1024._flexspi_arb_command_source.kFLEXSPI_IpCommand (*C enumerator*), 552
drivers_RT_RT1020_MIMXRT1024._flexspi_arb_command_source.kFLEXSPI_SuspendedCommand (*C enumerator*), 552
drivers_RT_RT1020_MIMXRT1024._flexspi_command_type (*C enum*), 552
drivers_RT_RT1020_MIMXRT1024._flexspi_command_type.kFLEXSPI_Command (*C enumerator*), 552
drivers_RT_RT1020_MIMXRT1024._flexspi_command_type.kFLEXSPI_Config (*C enumerator*), 552
drivers_RT_RT1020_MIMXRT1024._flexspi_command_type.kFLEXSPI_Read (*C enumerator*), 552
drivers_RT_RT1020_MIMXRT1024._flexspi_command_type.kFLEXSPI_Write (*C enumerator*), 552
drivers_RT_RT1020_MIMXRT1024._flexspi_config (*C struct*), 554
drivers_RT_RT1020_MIMXRT1024._flexspi_config.ahbConfig (*C struct*), 558
drivers_RT_RT1020_MIMXRT1024._flexspi_config.ahbConfig.ahbBusTimeoutCycle (*C var*), 558

drivers_RT_RT1020_MIMXRT1024._flexspi_config.ahbConfig.ahbGrantTimeoutCycle (*C var*), 558
drivers_RT_RT1020_MIMXRT1024._flexspi_config.ahbConfig.buffer (*C var*), 558
drivers_RT_RT1020_MIMXRT1024._flexspi_config.ahbConfig.disableAhbReadResume (*C var*), 558
drivers_RT_RT1020_MIMXRT1024._flexspi_config.ahbConfig.enableAHBBufferable (*C var*), 558
drivers_RT_RT1020_MIMXRT1024._flexspi_config.ahbConfig.enableAHBCachable (*C var*), 559
drivers_RT_RT1020_MIMXRT1024._flexspi_config.ahbConfig.enableAHBPrefetch (*C var*), 558
drivers_RT_RT1020_MIMXRT1024._flexspi_config.ahbConfig.enableClearAHBBufferOpt (*C var*), 558
drivers_RT_RT1020_MIMXRT1024._flexspi_config.ahbConfig.enableReadAddressOpt (*C var*), 558
drivers_RT_RT1020_MIMXRT1024._flexspi_config.ahbConfig.resumeWaitCycle (*C var*), 558
drivers_RT_RT1020_MIMXRT1024._flexspi_config.clockDiv (*C var*), 554
drivers_RT_RT1020_MIMXRT1024._flexspi_config.enableDoze (*C var*), 554
drivers_RT_RT1020_MIMXRT1024._flexspi_config.enableHalfSpeedAccess (*C var*), 555
drivers_RT_RT1020_MIMXRT1024._flexspi_config.enableSameConfigForAll (*C var*), 555
drivers_RT_RT1020_MIMXRT1024._flexspi_config.enableSckFreeRunning (*C var*), 554
drivers_RT_RT1020_MIMXRT1024._flexspi_config.ipGrantTimeoutCycle (*C var*), 555
drivers_RT_RT1020_MIMXRT1024._flexspi_config.rxSampleClock (*C var*), 554
drivers_RT_RT1020_MIMXRT1024._flexspi_config.rxSampleClockDiff (*C var*), 555
drivers_RT_RT1020_MIMXRT1024._flexspi_config.rxSampleClockPortB (*C var*), 555
drivers_RT_RT1020_MIMXRT1024._flexspi_config.rxWatermark (*C var*), 555
drivers_RT_RT1020_MIMXRT1024._flexspi_config.seqTimeoutCycle (*C var*), 555
drivers_RT_RT1020_MIMXRT1024._flexspi_config.txWatermark (*C var*), 555
drivers_RT_RT1020_MIMXRT1024._flexspi_cs_interval_cycle_unit (*C enum*), 550
drivers_RT_RT1020_MIMXRT1024._flexspi_cs_interval_cycle_unit.kFLEXSPI_CsIntervalUnit1SckCycle (*C enumerator*), 550
drivers_RT_RT1020_MIMXRT1024._flexspi_cs_interval_cycle_unit.kFLEXSPI_CsIntervalUnit256SckCycle (*C enumerator*), 550
drivers_RT_RT1020_MIMXRT1024._flexspi_device_config (*C struct*), 555
drivers_RT_RT1020_MIMXRT1024._flexspi_device_config.addressShift (*C var*), 555
drivers_RT_RT1020_MIMXRT1024._flexspi_device_config.AHBWriteWaitInterval (*C var*), 556
drivers_RT_RT1020_MIMXRT1024._flexspi_device_config.AHBWriteWaitUnit (*C var*), 556
drivers_RT_RT1020_MIMXRT1024._flexspi_device_config.ARDSeqIndex (*C var*), 556
drivers_RT_RT1020_MIMXRT1024._flexspi_device_config.ARDSeqNumber (*C var*), 556
drivers_RT_RT1020_MIMXRT1024._flexspi_device_config.AWRSeqIndex (*C var*), 556
drivers_RT_RT1020_MIMXRT1024._flexspi_device_config.AWRSeqNumber (*C var*), 556
drivers_RT_RT1020_MIMXRT1024._flexspi_device_config.columnspace (*C var*), 556
drivers_RT_RT1020_MIMXRT1024._flexspi_device_config.CSHoldTime (*C var*), 556
drivers_RT_RT1020_MIMXRT1024._flexspi_device_config.CSInterval (*C var*), 556
drivers_RT_RT1020_MIMXRT1024._flexspi_device_config.CSIntervalUnit (*C var*), 555
drivers_RT_RT1020_MIMXRT1024._flexspi_device_config.CSSetupTime (*C var*), 556
drivers_RT_RT1020_MIMXRT1024._flexspi_device_config.dataValidTime (*C var*), 556
drivers_RT_RT1020_MIMXRT1024._flexspi_device_config.enableWordAddress (*C var*), 556
drivers_RT_RT1020_MIMXRT1024._flexspi_device_config.enableWriteMask (*C var*), 556
drivers_RT_RT1020_MIMXRT1024._flexspi_device_config.flashSize (*C var*), 555
drivers_RT_RT1020_MIMXRT1024._flexspi_device_config.flexspiRootClk (*C var*), 555
drivers_RT_RT1020_MIMXRT1024._flexspi_device_config.isFroClockSource (*C var*), 556
drivers_RT_RT1020_MIMXRT1024._flexspi_device_config.isSck2Enabled (*C var*), 555
drivers_RT_RT1020_MIMXRT1024._flexspi_edma_handle (*C struct*), 561
drivers_RT_RT1020_MIMXRT1024._flexspi_edma_handle.completionCallback (*C var*), 561
drivers_RT_RT1020_MIMXRT1024._flexspi_edma_handle.count (*C var*), 561
drivers_RT_RT1020_MIMXRT1024._flexspi_edma_handle.nbytes (*C var*), 561
drivers_RT_RT1020_MIMXRT1024._flexspi_edma_handle.nsize (*C var*), 561
drivers_RT_RT1020_MIMXRT1024._flexspi_edma_handle.rxDmaHandle (*C var*), 561
drivers_RT_RT1020_MIMXRT1024._flexspi_edma_handle.state (*C var*), 561
drivers_RT_RT1020_MIMXRT1024._flexspi_edma_handle.transferSize (*C var*), 561
drivers_RT_RT1020_MIMXRT1024._flexspi_edma_handle.txDmaHandle (*C var*), 561
drivers_RT_RT1020_MIMXRT1024._flexspi_edma_handle.userData (*C var*), 561
drivers_RT_RT1020_MIMXRT1024._flexspi_edma_ntransfer_size (*C enum*), 560

drivers_RT_RT1020_MIMXRT1024._flexspi_edma_ntransfer_size.kFLEXPSI_EDMAnSize1Bytes (*C enumerator*), 560
drivers_RT_RT1020_MIMXRT1024._flexspi_edma_ntransfer_size.kFLEXPSI_EDMAnSize2Bytes (*C enumerator*), 560
drivers_RT_RT1020_MIMXRT1024._flexspi_edma_ntransfer_size.kFLEXPSI_EDMAnSize4Bytes (*C enumerator*), 560
drivers_RT_RT1020_MIMXRT1024._flexspi_edma_ntransfer_size.kFLEXPSI_EDMAnSize8Bytes (*C enumerator*), 560
drivers_RT_RT1020_MIMXRT1024._flexspi_edma_ntransfer_size.kFLEXPSI_EDMAnSize32Bytes (*C enumerator*), 560
drivers_RT_RT1020_MIMXRT1024._flexspi_flags (*C enum*), 549
drivers_RT_RT1020_MIMXRT1024._flexspi_flags.kFLEXSPI_AhbBusErrorFlag (*C enumerator*), 549
drivers_RT_RT1020_MIMXRT1024._flexspi_flags.kFLEXSPI_AhbCommandGrantTimeoutFlag (*C enumerator*), 549
drivers_RT_RT1020_MIMXRT1024._flexspi_flags.kFLEXSPI_AhbCommandSequenceErrorFlag (*C enumerator*), 549
drivers_RT_RT1020_MIMXRT1024._flexspi_flags.kFLEXSPI_AllInterruptFlags (*C enumerator*), 549
drivers_RT_RT1020_MIMXRT1024._flexspi_flags.kFLEXSPI_IpCommandExecutionDoneFlag (*C enumerator*), 549
drivers_RT_RT1020_MIMXRT1024._flexspi_flags.kFLEXSPI_IpCommandGrantTimeoutFlag (*C enumerator*), 549
drivers_RT_RT1020_MIMXRT1024._flexspi_flags.kFLEXSPI_IpCommandSequenceErrorFlag (*C enumerator*), 549
drivers_RT_RT1020_MIMXRT1024._flexspi_flags.kFLEXSPI_IpRxFifoWatermarkAvailableFlag (*C enumerator*), 549
drivers_RT_RT1020_MIMXRT1024._flexspi_flags.kFLEXSPI_IpTxFifoWatermarkEmptyFlag (*C enumerator*), 549
drivers_RT_RT1020_MIMXRT1024._flexspi_flags.kFLEXSPI_SckStoppedBecauseRxFullFlag (*C enumerator*), 549
drivers_RT_RT1020_MIMXRT1024._flexspi_flags.kFLEXSPI_SckStoppedBecauseTxEmptyFlag (*C enumerator*), 549
drivers_RT_RT1020_MIMXRT1024._flexspi_flags.kFLEXSPI_SequenceExecutionTimeoutFlag (*C enumerator*), 549
drivers_RT_RT1020_MIMXRT1024._flexspi_handle (*C struct*), 557
drivers_RT_RT1020_MIMXRT1024._flexspi_handle.completionCallback (*C var*), 557
drivers_RT_RT1020_MIMXRT1024._flexspi_handle.data (*C var*), 557
drivers_RT_RT1020_MIMXRT1024._flexspi_handle.dataSize (*C var*), 557
drivers_RT_RT1020_MIMXRT1024._flexspi_handle.state (*C var*), 557
drivers_RT_RT1020_MIMXRT1024._flexspi_handle.transferTotalSize (*C var*), 557
drivers_RT_RT1020_MIMXRT1024._flexspi_handle.userData (*C var*), 557
drivers_RT_RT1020_MIMXRT1024._flexspi_ip_error_code (*C enum*), 551
drivers_RT_RT1020_MIMXRT1024._flexspi_ip_error_code.kFLEXSPI_IpCmdErrorDdrDummyInSdrSequence (*C enumerator*), 551
drivers_RT_RT1020_MIMXRT1024._flexspi_ip_error_code.kFLEXSPI_IpCmdErrorFlashBoundaryAcrosss (*C enumerator*), 551
drivers_RT_RT1020_MIMXRT1024._flexspi_ip_error_code.kFLEXSPI_IpCmdErrorInvalidAddress (*C enumerator*), 551
drivers_RT_RT1020_MIMXRT1024._flexspi_ip_error_code.kFLEXSPI_IpCmdErrorJumpOnCsInIpCmd (*C enumerator*), 551
drivers_RT_RT1020_MIMXRT1024._flexspi_ip_error_code.kFLEXSPI_IpCmdErrorNoError (*C enumerator*), 551
drivers_RT_RT1020_MIMXRT1024._flexspi_ip_error_code.kFLEXSPI_IpCmdErrorSdrDummyInDdrSequence (*C enumerator*), 551
drivers_RT_RT1020_MIMXRT1024._flexspi_ip_error_code.kFLEXSPI_IpCmdErrorSequenceExecutionTimeout (*C enumerator*), 551
drivers_RT_RT1020_MIMXRT1024._flexspi_ip_error_code.kFLEXSPI_IpCmdErrorUnknownOpCode

(*C enumerator*), 551
drivers_RT_RT1020_MIMXRT1024._flexspi_pad (*C enum*), 548
drivers_RT_RT1020_MIMXRT1024._flexspi_pad.kFLEXSPI_1PAD (*C enumerator*), 548
drivers_RT_RT1020_MIMXRT1024._flexspi_pad.kFLEXSPI_2PAD (*C enumerator*), 548
drivers_RT_RT1020_MIMXRT1024._flexspi_pad.kFLEXSPI_4PAD (*C enumerator*), 548
drivers_RT_RT1020_MIMXRT1024._flexspi_pad.kFLEXSPI_8PAD (*C enumerator*), 548
drivers_RT_RT1020_MIMXRT1024._flexspi_port (*C enum*), 552
drivers_RT_RT1020_MIMXRT1024._flexspi_port.kFLEXSPI_PortA1 (*C enumerator*), 552
drivers_RT_RT1020_MIMXRT1024._flexspi_port.kFLEXSPI_PortA2 (*C enumerator*), 552
drivers_RT_RT1020_MIMXRT1024._flexspi_port.kFLEXSPI_PortCount (*C enumerator*), 552
drivers_RT_RT1020_MIMXRT1024._flexspi_read_sample_clock (*C enum*), 549
drivers_RT_RT1020_MIMXRT1024._flexspi_read_sample_clock.kFLEXSPI_ReadSampleClkExternalInputFromDqsl
(*C enumerator*), 550
drivers_RT_RT1020_MIMXRT1024._flexspi_read_sample_clock.kFLEXSPI_ReadSampleClkLoopbackFromDqsPad
(*C enumerator*), 550
drivers_RT_RT1020_MIMXRT1024._flexspi_read_sample_clock.kFLEXSPI_ReadSampleClkLoopbackFromSckPad
(*C enumerator*), 550
drivers_RT_RT1020_MIMXRT1024._flexspi_read_sample_clock.kFLEXSPI_ReadSampleClkLoopbackInternally
(*C enumerator*), 549
drivers_RT_RT1020_MIMXRT1024._flexspi_transfer (*C struct*), 556
drivers_RT_RT1020_MIMXRT1024._flexspi_transfer.cmdType (*C var*), 557
drivers_RT_RT1020_MIMXRT1024._flexspi_transfer.data (*C var*), 557
drivers_RT_RT1020_MIMXRT1024._flexspi_transfer.dataSize (*C var*), 557
drivers_RT_RT1020_MIMXRT1024._flexspi_transfer.deviceAddress (*C var*), 557
drivers_RT_RT1020_MIMXRT1024._flexspi_transfer.port (*C var*), 557
drivers_RT_RT1020_MIMXRT1024._flexspi_transfer.seqIndex (*C var*), 557
drivers_RT_RT1020_MIMXRT1024._flexspi_transfer.SeqNumber (*C var*), 557
drivers_RT_RT1020_MIMXRT1024._gpio_interrupt_mode (*C enum*), 567
drivers_RT_RT1020_MIMXRT1024._gpio_interrupt_mode.kGPIO_IntFallingEdge (*C enumerator*),
567
drivers_RT_RT1020_MIMXRT1024._gpio_interrupt_mode.kGPIO_IntHighLevel (*C enumerator*),
567
drivers_RT_RT1020_MIMXRT1024._gpio_interrupt_mode.kGPIO_IntLowLevel (*C enumerator*),
567
drivers_RT_RT1020_MIMXRT1024._gpio_interrupt_mode.kGPIO_IntRisingEdge (*C enumerator*),
567
drivers_RT_RT1020_MIMXRT1024._gpio_interrupt_mode.kGPIO_IntRisingOrFallingEdge (*C enumerator*), 567
drivers_RT_RT1020_MIMXRT1024._gpio_interrupt_mode.kGPIO_NoIntmode (*C enumerator*), 567
drivers_RT_RT1020_MIMXRT1024._gpio_pin_config (*C struct*), 567
drivers_RT_RT1020_MIMXRT1024._gpio_pin_config.direction (*C var*), 568
drivers_RT_RT1020_MIMXRT1024._gpio_pin_config.interruptMode (*C var*), 568
drivers_RT_RT1020_MIMXRT1024._gpio_pin_config.outputLogic (*C var*), 568
drivers_RT_RT1020_MIMXRT1024._gpio_pin_direction (*C enum*), 567
drivers_RT_RT1020_MIMXRT1024._gpio_pin_direction.kGPIO_DigitalInput (*C enumerator*), 567
drivers_RT_RT1020_MIMXRT1024._gpio_pin_direction.kGPIO_DigitalOutput (*C enumerator*),
567
drivers_RT_RT1020_MIMXRT1024._gpt_clock_source (*C enum*), 572
drivers_RT_RT1020_MIMXRT1024._gpt_clock_source.kGPT_ClockSource_Ext (*C enumerator*),
572
drivers_RT_RT1020_MIMXRT1024._gpt_clock_source.kGPT_ClockSource_HighFreq (*C enumerator*), 572
drivers_RT_RT1020_MIMXRT1024._gpt_clock_source.kGPT_ClockSource_LowFreq (*C enumerator*), 572
drivers_RT_RT1020_MIMXRT1024._gpt_clock_source.kGPT_ClockSource_Off (*C enumerator*),
572
drivers_RT_RT1020_MIMXRT1024._gpt_clock_source.kGPT_ClockSource_Osc (*C enumerator*),
572

drivers_RT_RT1020_MIMXRT1024._gpt_clock_source.kGPT_ClockSource_Periph (*C enumerator*), 572
drivers_RT_RT1020_MIMXRT1024._gpt_init_config (*C struct*), 575
drivers_RT_RT1020_MIMXRT1024._gpt_init_config.clockSource (*C var*), 575
drivers_RT_RT1020_MIMXRT1024._gpt_init_config.divider (*C var*), 575
drivers_RT_RT1020_MIMXRT1024._gpt_init_config.enableFreeRun (*C var*), 575
drivers_RT_RT1020_MIMXRT1024._gpt_init_config.enableMode (*C var*), 576
drivers_RT_RT1020_MIMXRT1024._gpt_init_config.enableRunInDbg (*C var*), 576
drivers_RT_RT1020_MIMXRT1024._gpt_init_config.enableRunInDoze (*C var*), 576
drivers_RT_RT1020_MIMXRT1024._gpt_init_config.enableRunInStop (*C var*), 576
drivers_RT_RT1020_MIMXRT1024._gpt_init_config.enableRunInWait (*C var*), 575
drivers_RT_RT1020_MIMXRT1024._gpt_input_capture_channel (*C enum*), 572
drivers_RT_RT1020_MIMXRT1024._gpt_input_capture_channel.kGPT_InputCapture_Channel1 (*C enumerator*), 573
drivers_RT_RT1020_MIMXRT1024._gpt_input_capture_channel.kGPT_InputCapture_Channel2 (*C enumerator*), 573
drivers_RT_RT1020_MIMXRT1024._gpt_input_operation_mode (*C enum*), 573
drivers_RT_RT1020_MIMXRT1024._gpt_input_operation_mode.kGPT_InputOperation_BothEdge (*C enumerator*), 573
drivers_RT_RT1020_MIMXRT1024._gpt_input_operation_mode.kGPT_InputOperation_Disabled (*C enumerator*), 573
drivers_RT_RT1020_MIMXRT1024._gpt_input_operation_mode.kGPT_InputOperation_FallEdge (*C enumerator*), 573
drivers_RT_RT1020_MIMXRT1024._gpt_input_operation_mode.kGPT_InputOperation_RiseEdge (*C enumerator*), 573
drivers_RT_RT1020_MIMXRT1024._gpt_interrupt_enable (*C enum*), 574
drivers_RT_RT1020_MIMXRT1024._gpt_interrupt_enable.kGPT_InputCapture1InterruptEnable (*C enumerator*), 574
drivers_RT_RT1020_MIMXRT1024._gpt_interrupt_enable.kGPT_InputCapture2InterruptEnable (*C enumerator*), 574
drivers_RT_RT1020_MIMXRT1024._gpt_interrupt_enable.kGPT_OutputCompare1InterruptEnable (*C enumerator*), 574
drivers_RT_RT1020_MIMXRT1024._gpt_interrupt_enable.kGPT_OutputCompare2InterruptEnable (*C enumerator*), 574
drivers_RT_RT1020_MIMXRT1024._gpt_interrupt_enable.kGPT_OutputCompare3InterruptEnable (*C enumerator*), 574
drivers_RT_RT1020_MIMXRT1024._gpt_interrupt_enable.kGPT_RollOverFlagInterruptEnable (*C enumerator*), 574
drivers_RT_RT1020_MIMXRT1024._gpt_output_compare_channel (*C enum*), 573
drivers_RT_RT1020_MIMXRT1024._gpt_output_compare_channel.kGPT_OutputCompare_Channel1 (*C enumerator*), 573
drivers_RT_RT1020_MIMXRT1024._gpt_output_compare_channel.kGPT_OutputCompare_Channel2 (*C enumerator*), 573
drivers_RT_RT1020_MIMXRT1024._gpt_output_compare_channel.kGPT_OutputCompare_Channel3 (*C enumerator*), 573
drivers_RT_RT1020_MIMXRT1024._gpt_output_operation_mode (*C enum*), 573
drivers_RT_RT1020_MIMXRT1024._gpt_output_operation_mode.kGPT_OutputOperation_Activelow (*C enumerator*), 574
drivers_RT_RT1020_MIMXRT1024._gpt_output_operation_mode.kGPT_OutputOperation_Clear (*C enumerator*), 573
drivers_RT_RT1020_MIMXRT1024._gpt_output_operation_mode.kGPT_OutputOperation_Disconnected (*C enumerator*), 573
drivers_RT_RT1020_MIMXRT1024._gpt_output_operation_mode.kGPT_OutputOperation_Set (*C enumerator*), 574
drivers_RT_RT1020_MIMXRT1024._gpt_output_operation_mode.kGPT_OutputOperation_Toggle (*C enumerator*), 573
drivers_RT_RT1020_MIMXRT1024._gpt_status_flag (*C enum*), 574
drivers_RT_RT1020_MIMXRT1024._gpt_status_flag.kGPT_InputCapture1Flag (*C enumerator*),

574
drivers_RT_RT1020_MIMXRT1024._gpt_status_flag.kGPT_InputCapture2Flag (*C enumerator*), 574
drivers_RT_RT1020_MIMXRT1024._gpt_status_flag.kGPT_OutputCompare1Flag (*C enumerator*), 574
drivers_RT_RT1020_MIMXRT1024._gpt_status_flag.kGPT_OutputCompare2Flag (*C enumerator*), 574
drivers_RT_RT1020_MIMXRT1024._gpt_status_flag.kGPT_OutputCompare3Flag (*C enumerator*), 574
drivers_RT_RT1020_MIMXRT1024._gpt_status_flag.kGPT_RollOverFlag (*C enumerator*), 575
drivers_RT_RT1020_MIMXRT1024._iomuxc_gpr_mode (*C enum*), 576
drivers_RT_RT1020_MIMXRT1024._iomuxc_gpr_mode.kIOMUXC_GPR_AHBClockEnable (*C enumerator*), 576
drivers_RT_RT1020_MIMXRT1024._iomuxc_gpr_mode.kIOMUXC_GPR_ENET1RefClkMode (*C enumerator*), 576
drivers_RT_RT1020_MIMXRT1024._iomuxc_gpr_mode.kIOMUXC_GPR_ENET1TxClkOutputDir (*C enumerator*), 576
drivers_RT_RT1020_MIMXRT1024._iomuxc_gpr_mode.kIOMUXC_GPR_ExcMonitorSlavErrResponse (*C enumerator*), 576
drivers_RT_RT1020_MIMXRT1024._iomuxc_gpr_mode.kIOMUXC_GPR_GlobalInterruptRequest (*C enumerator*), 576
drivers_RT_RT1020_MIMXRT1024._iomuxc_gpr_mode.kIOMUXC_GPR_SAI1MClkOutputDir (*C enumerator*), 576
drivers_RT_RT1020_MIMXRT1024._iomuxc_gpr_mode.kIOMUXC_GPR_SAI2MClkOutputDir (*C enumerator*), 576
drivers_RT_RT1020_MIMXRT1024._iomuxc_gpr_mode.kIOMUXC_GPR_SAI3MClkOutputDir (*C enumerator*), 576
drivers_RT_RT1020_MIMXRT1024._iomuxc_gpr_saimclk (*C enum*), 576
drivers_RT_RT1020_MIMXRT1024._iomuxc_gpr_saimclk.kIOMUXC_GPR_SAI1MClk1Sel (*C enumerator*), 576
drivers_RT_RT1020_MIMXRT1024._iomuxc_gpr_saimclk.kIOMUXC_GPR_SAI1MClk2Sel (*C enumerator*), 576
drivers_RT_RT1020_MIMXRT1024._iomuxc_gpr_saimclk.kIOMUXC_GPR_SAI1MClk3Sel (*C enumerator*), 576
drivers_RT_RT1020_MIMXRT1024._iomuxc_gpr_saimclk.kIOMUXC_GPR_SAI2MClk3Sel (*C enumerator*), 576
drivers_RT_RT1020_MIMXRT1024._iomuxc_gpr_saimclk.kIOMUXC_GPR_SAI3MClk3Sel (*C enumerator*), 577
drivers_RT_RT1020_MIMXRT1024._iomuxc_mqs_pwm_oversample_rate (*C enum*), 577
drivers_RT_RT1020_MIMXRT1024._iomuxc_mqs_pwm_oversample_rate.kIOMUXC_MqsPwmOverSampleRate32 (*C enumerator*), 577
drivers_RT_RT1020_MIMXRT1024._iomuxc_mqs_pwm_oversample_rate.kIOMUXC_MqsPwmOverSampleRate64 (*C enumerator*), 577
drivers_RT_RT1020_MIMXRT1024._kpp_config (*C struct*), 608
drivers_RT_RT1020_MIMXRT1024._kpp_config.activeColumn (*C var*), 608
drivers_RT_RT1020_MIMXRT1024._kpp_config.activeRow (*C var*), 608
drivers_RT_RT1020_MIMXRT1024._kpp_config.interrupt (*C var*), 608
drivers_RT_RT1020_MIMXRT1024._kpp_interrupt_enable (*C enum*), 607
drivers_RT_RT1020_MIMXRT1024._kpp_interrupt_enable.kKPP_keyDepressInterrupt (*C enumerator*), 607
drivers_RT_RT1020_MIMXRT1024._kpp_interrupt_enable.kKPP_keyReleaseInterrupt (*C enumerator*), 607
drivers_RT_RT1020_MIMXRT1024._kpp_sync_operation (*C enum*), 607
drivers_RT_RT1020_MIMXRT1024._kpp_sync_operation.kKPP_ClearKeyDepressSyncChain (*C enumerator*), 607
drivers_RT_RT1020_MIMXRT1024._kpp_sync_operation.kKPP_SetKeyReleasesSyncChain (*C enumerator*), 607
drivers_RT_RT1020_MIMXRT1024._lpi2c_data_match_config_mode (*C enum*), 637

drivers_RT_RT1020_MIMXRT1024._lpi2c_data_match_config_mode.kLPI2C_1stWordAndM1EqualsM0AndM1 (*C enumerator*), 637
drivers_RT_RT1020_MIMXRT1024._lpi2c_data_match_config_mode.kLPI2C_1stWordEqualsM0And2ndWordEqua (*C enumerator*), 637
drivers_RT_RT1020_MIMXRT1024._lpi2c_data_match_config_mode.kLPI2C_1stWordEqualsM0OrM1 (*C enumerator*), 637
drivers_RT_RT1020_MIMXRT1024._lpi2c_data_match_config_mode.kLPI2C_AnyWordAndM1EqualsM0AndM1 (*C enumerator*), 637
drivers_RT_RT1020_MIMXRT1024._lpi2c_data_match_config_mode.kLPI2C_AnyWordEqualsM0AndNextWordEq (*C enumerator*), 637
drivers_RT_RT1020_MIMXRT1024._lpi2c_data_match_config_mode.kLPI2C_AnyWordEqualsM0OrM1 (*C enumerator*), 637
drivers_RT_RT1020_MIMXRT1024._lpi2c_data_match_config_mode.kLPI2C_MatchDisabled (*C enumerator*), 637
drivers_RT_RT1020_MIMXRT1024._lpi2c_direction (*C enum*), 636
drivers_RT_RT1020_MIMXRT1024._lpi2c_direction.kLPI2C_Read (*C enumerator*), 636
drivers_RT_RT1020_MIMXRT1024._lpi2c_direction.kLPI2C_Write (*C enumerator*), 636
drivers_RT_RT1020_MIMXRT1024._lpi2c_host_request_polarity (*C enum*), 637
drivers_RT_RT1020_MIMXRT1024._lpi2c_host_request_polarity.kLPI2C_HostRequestPinActiveHigh (*C enumerator*), 637
drivers_RT_RT1020_MIMXRT1024._lpi2c_host_request_polarity.kLPI2C_HostRequestPinActiveLow (*C enumerator*), 637
drivers_RT_RT1020_MIMXRT1024._lpi2c_host_request_source (*C enum*), 636
drivers_RT_RT1020_MIMXRT1024._lpi2c_host_request_source.kLPI2C_HostRequestExternalPin (*C enumerator*), 636
drivers_RT_RT1020_MIMXRT1024._lpi2c_host_request_source.kLPI2C_HostRequestInputTrigger (*C enumerator*), 636
drivers_RT_RT1020_MIMXRT1024._lpi2c_master_config (*C struct*), 639
drivers_RT_RT1020_MIMXRT1024._lpi2c_master_config.baudRate_Hz (*C var*), 639
drivers_RT_RT1020_MIMXRT1024._lpi2c_master_config.busIdleTimeout_ns (*C var*), 639
drivers_RT_RT1020_MIMXRT1024._lpi2c_master_config.debugEnable (*C var*), 639
drivers_RT_RT1020_MIMXRT1024._lpi2c_master_config.enableDoze (*C var*), 639
drivers_RT_RT1020_MIMXRT1024._lpi2c_master_config.enableMaster (*C var*), 639
drivers_RT_RT1020_MIMXRT1024._lpi2c_master_config.hostRequest (*C struct*), 641
drivers_RT_RT1020_MIMXRT1024._lpi2c_master_config.hostRequest (*C var*), 639
drivers_RT_RT1020_MIMXRT1024._lpi2c_master_config.hostRequest.enable (*C var*), 641
drivers_RT_RT1020_MIMXRT1024._lpi2c_master_config.hostRequest.polarity (*C var*), 641
drivers_RT_RT1020_MIMXRT1024._lpi2c_master_config.hostRequest.source (*C var*), 641
drivers_RT_RT1020_MIMXRT1024._lpi2c_master_config.ignoreAck (*C var*), 639
drivers_RT_RT1020_MIMXRT1024._lpi2c_master_config.pinConfig (*C var*), 639
drivers_RT_RT1020_MIMXRT1024._lpi2c_master_config.pinLowTimeout_ns (*C var*), 639
drivers_RT_RT1020_MIMXRT1024._lpi2c_master_config.sclGlitchFilterWidth_ns (*C var*), 639
drivers_RT_RT1020_MIMXRT1024._lpi2c_master_config.sdaGlitchFilterWidth_ns (*C var*), 639
drivers_RT_RT1020_MIMXRT1024._lpi2c_master_edma_handle (*C struct*), 643
drivers_RT_RT1020_MIMXRT1024._lpi2c_master_edma_handle.base (*C var*), 643
drivers_RT_RT1020_MIMXRT1024._lpi2c_master_edma_handle.commandBuffer (*C var*), 644
drivers_RT_RT1020_MIMXRT1024._lpi2c_master_edma_handle.completionCallback (*C var*), 644
drivers_RT_RT1020_MIMXRT1024._lpi2c_master_edma_handle.isBusy (*C var*), 643
drivers_RT_RT1020_MIMXRT1024._lpi2c_master_edma_handle.nbytes (*C var*), 644
drivers_RT_RT1020_MIMXRT1024._lpi2c_master_edma_handle.rx (*C var*), 644
drivers_RT_RT1020_MIMXRT1024._lpi2c_master_edma_handle.tcds (*C var*), 644
drivers_RT_RT1020_MIMXRT1024._lpi2c_master_edma_handle.transfer (*C var*), 644
drivers_RT_RT1020_MIMXRT1024._lpi2c_master_edma_handle.tx (*C var*), 644
drivers_RT_RT1020_MIMXRT1024._lpi2c_master_edma_handle.userData (*C var*), 644
drivers_RT_RT1020_MIMXRT1024._lpi2c_master_flags (*C enum*), 634
drivers_RT_RT1020_MIMXRT1024._lpi2c_master_flags.kLPI2C_MasterArbitrationLostFlag (*C enumerator*), 635

drivers_RT_RT1020_MIMXRT1024._lpi2c_master_flags.kLPI2C_MasterBusBusyFlag (*C enumerator*), 635
drivers_RT_RT1020_MIMXRT1024._lpi2c_master_flags.kLPI2C_MasterBusyFlag (*C enumerator*), 635
drivers_RT_RT1020_MIMXRT1024._lpi2c_master_flags.kLPI2C_MasterClearFlags (*C enumerator*), 635
drivers_RT_RT1020_MIMXRT1024._lpi2c_master_flags.kLPI2C_MasterDataMatchFlag (*C enumerator*), 635
drivers_RT_RT1020_MIMXRT1024._lpi2c_master_flags.kLPI2C_MasterEndOfPacketFlag (*C enumerator*), 635
drivers_RT_RT1020_MIMXRT1024._lpi2c_master_flags.kLPI2C_MasterErrorFlags (*C enumerator*), 635
drivers_RT_RT1020_MIMXRT1024._lpi2c_master_flags.kLPI2C_MasterFifoErrFlag (*C enumerator*), 635
drivers_RT_RT1020_MIMXRT1024._lpi2c_master_flags.kLPI2C_MasterIrqFlags (*C enumerator*), 635
drivers_RT_RT1020_MIMXRT1024._lpi2c_master_flags.kLPI2C_MasterNackDetectFlag (*C enumerator*), 635
drivers_RT_RT1020_MIMXRT1024._lpi2c_master_flags.kLPI2C_MasterPinLowTimeoutFlag (*C enumerator*), 635
drivers_RT_RT1020_MIMXRT1024._lpi2c_master_flags.kLPI2C_MasterRxReadyFlag (*C enumerator*), 635
drivers_RT_RT1020_MIMXRT1024._lpi2c_master_flags.kLPI2C_MasterStopDetectFlag (*C enumerator*), 635
drivers_RT_RT1020_MIMXRT1024._lpi2c_master_flags.kLPI2C_MasterTxReadyFlag (*C enumerator*), 635
drivers_RT_RT1020_MIMXRT1024._lpi2c_master_handle (*C struct*), 640
drivers_RT_RT1020_MIMXRT1024._lpi2c_master_handle.buf (*C var*), 641
drivers_RT_RT1020_MIMXRT1024._lpi2c_master_handle.chunkSize (*C var*), 641
drivers_RT_RT1020_MIMXRT1024._lpi2c_master_handle.commandBuffer (*C var*), 641
drivers_RT_RT1020_MIMXRT1024._lpi2c_master_handle.completionCallback (*C var*), 641
drivers_RT_RT1020_MIMXRT1024._lpi2c_master_handle.remainingBytes (*C var*), 641
drivers_RT_RT1020_MIMXRT1024._lpi2c_master_handle.state (*C var*), 641
drivers_RT_RT1020_MIMXRT1024._lpi2c_master_handle.transfer (*C var*), 641
drivers_RT_RT1020_MIMXRT1024._lpi2c_master_handle.userData (*C var*), 641
drivers_RT_RT1020_MIMXRT1024._lpi2c_master_pin_config (*C enum*), 636
drivers_RT_RT1020_MIMXRT1024._lpi2c_master_pin_config.kLPI2C_2PinOpenDrain (*C enumerator*), 636
drivers_RT_RT1020_MIMXRT1024._lpi2c_master_pin_config.kLPI2C_2PinOpenDrainWithSeparateSlave (*C enumerator*), 636
drivers_RT_RT1020_MIMXRT1024._lpi2c_master_pin_config.kLPI2C_2PinOutputOnly (*C enumerator*), 636
drivers_RT_RT1020_MIMXRT1024._lpi2c_master_pin_config.kLPI2C_2PinOutputOnlyWithSeparateSlave (*C enumerator*), 636
drivers_RT_RT1020_MIMXRT1024._lpi2c_master_pin_config.kLPI2C_2PinPushPull (*C enumerator*), 636
drivers_RT_RT1020_MIMXRT1024._lpi2c_master_pin_config.kLPI2C_2PinPushPullWithSeparateSlave (*C enumerator*), 636
drivers_RT_RT1020_MIMXRT1024._lpi2c_master_pin_config.kLPI2C_4PinPushPull (*C enumerator*), 636
drivers_RT_RT1020_MIMXRT1024._lpi2c_master_pin_config.kLPI2C_4PinPushPullWithInvertedOutput (*C enumerator*), 636
drivers_RT_RT1020_MIMXRT1024._lpi2c_master_transfer (*C struct*), 640
drivers_RT_RT1020_MIMXRT1024._lpi2c_master_transfer_flags (*C enum*), 637
drivers_RT_RT1020_MIMXRT1024._lpi2c_master_transfer_flags.kLPI2C_TransferDefaultFlag (*C enumerator*), 637
drivers_RT_RT1020_MIMXRT1024._lpi2c_master_transfer_flags.kLPI2C_TransferNoStartFlag (*C enumerator*), 637

drivers_RT_RT1020_MIMXRT1024._lpi2c_master_transfer_flags.kLPI2C_TransferNoStopFlag (*C enumerator*), 638
drivers_RT_RT1020_MIMXRT1024._lpi2c_master_transfer.data (*C var*), 640
drivers_RT_RT1020_MIMXRT1024._lpi2c_master_transfer.dataSize (*C var*), 640
drivers_RT_RT1020_MIMXRT1024._lpi2c_master_transfer.direction (*C var*), 640
drivers_RT_RT1020_MIMXRT1024._lpi2c_master_transfer.flags (*C var*), 640
drivers_RT_RT1020_MIMXRT1024._lpi2c_master_transfer.slaveAddress (*C var*), 640
drivers_RT_RT1020_MIMXRT1024._lpi2c_master_transfer.subaddress (*C var*), 640
drivers_RT_RT1020_MIMXRT1024._lpi2c_master_transfer.subaddressSize (*C var*), 640
drivers_RT_RT1020_MIMXRT1024._lpi2c_match_config (*C struct*), 639
drivers_RT_RT1020_MIMXRT1024._lpi2c_match_config.match0 (*C var*), 640
drivers_RT_RT1020_MIMXRT1024._lpi2c_match_config.match1 (*C var*), 640
drivers_RT_RT1020_MIMXRT1024._lpi2c_match_config.matchMode (*C var*), 640
drivers_RT_RT1020_MIMXRT1024._lpi2c_match_config.rxDataMatchOnly (*C var*), 640
drivers_RT_RT1020_MIMXRT1024._lpi2c_slave_address_match (*C enum*), 651
drivers_RT_RT1020_MIMXRT1024._lpi2c_slave_address_match.kLPI2C_MatchAddress0 (*C enumerator*), 651
drivers_RT_RT1020_MIMXRT1024._lpi2c_slave_address_match.kLPI2C_MatchAddress0OrAddress1 (*C enumerator*), 651
drivers_RT_RT1020_MIMXRT1024._lpi2c_slave_address_match.kLPI2C_MatchAddress0ThroughAddress1 (*C enumerator*), 651
drivers_RT_RT1020_MIMXRT1024._lpi2c_slave_config (*C struct*), 653
drivers_RT_RT1020_MIMXRT1024._lpi2c_slave_config.address0 (*C var*), 653
drivers_RT_RT1020_MIMXRT1024._lpi2c_slave_config.address1 (*C var*), 653
drivers_RT_RT1020_MIMXRT1024._lpi2c_slave_config.addressMatchMode (*C var*), 653
drivers_RT_RT1020_MIMXRT1024._lpi2c_slave_config.clockHoldTime_ns (*C var*), 654
drivers_RT_RT1020_MIMXRT1024._lpi2c_slave_config.dataValidDelay_ns (*C var*), 654
drivers_RT_RT1020_MIMXRT1024._lpi2c_slave_config.enableGeneralCall (*C var*), 653
drivers_RT_RT1020_MIMXRT1024._lpi2c_slave_config.enableReceivedAddressRead (*C var*), 654
drivers_RT_RT1020_MIMXRT1024._lpi2c_slave_config.enableSlave (*C var*), 653
drivers_RT_RT1020_MIMXRT1024._lpi2c_slave_config.filterDozeEnable (*C var*), 653
drivers_RT_RT1020_MIMXRT1024._lpi2c_slave_config.filterEnable (*C var*), 653
drivers_RT_RT1020_MIMXRT1024._lpi2c_slave_config.ignoreAck (*C var*), 653
drivers_RT_RT1020_MIMXRT1024._lpi2c_slave_config.sclGlitchFilterWidth_ns (*C var*), 654
drivers_RT_RT1020_MIMXRT1024._lpi2c_slave_config.sclStall (*C struct*), 655
drivers_RT_RT1020_MIMXRT1024._lpi2c_slave_config.sclStall (*C var*), 653
drivers_RT_RT1020_MIMXRT1024._lpi2c_slave_config.sclStall.enableAck (*C var*), 655
drivers_RT_RT1020_MIMXRT1024._lpi2c_slave_config.sclStall.enableAddress (*C var*), 655
drivers_RT_RT1020_MIMXRT1024._lpi2c_slave_config.sclStall.enableRx (*C var*), 655
drivers_RT_RT1020_MIMXRT1024._lpi2c_slave_config.sclStall.enableTx (*C var*), 655
drivers_RT_RT1020_MIMXRT1024._lpi2c_slave_config.sdaGlitchFilterWidth_ns (*C var*), 654
drivers_RT_RT1020_MIMXRT1024._lpi2c_slave_flags (*C enum*), 650
drivers_RT_RT1020_MIMXRT1024._lpi2c_slave_flags.kLPI2C_SlaveAddressMatch0Flag (*C enumerator*), 650
drivers_RT_RT1020_MIMXRT1024._lpi2c_slave_flags.kLPI2C_SlaveAddressMatch1Flag (*C enumerator*), 651
drivers_RT_RT1020_MIMXRT1024._lpi2c_slave_flags.kLPI2C_SlaveAddressValidFlag (*C enumerator*), 650
drivers_RT_RT1020_MIMXRT1024._lpi2c_slave_flags.kLPI2C_SlaveBitErrFlag (*C enumerator*), 650
drivers_RT_RT1020_MIMXRT1024._lpi2c_slave_flags.kLPI2C_SlaveBusBusyFlag (*C enumerator*), 651
drivers_RT_RT1020_MIMXRT1024._lpi2c_slave_flags.kLPI2C_SlaveBusyFlag (*C enumerator*), 651
drivers_RT_RT1020_MIMXRT1024._lpi2c_slave_flags.kLPI2C_SlaveClearFlags (*C enumerator*), 651
drivers_RT_RT1020_MIMXRT1024._lpi2c_slave_flags.kLPI2C_SlaveErrorFlags (*C enumerator*), 651

drivers_RT_RT1020_MIMXRT1024._lpi2c_slave_flags.kLPI2C_SlaveFifoErrFlag (*C enumerator*), 650
drivers_RT_RT1020_MIMXRT1024._lpi2c_slave_flags.kLPI2C_SlaveGeneralCallFlag (*C enumerator*), 651
drivers_RT_RT1020_MIMXRT1024._lpi2c_slave_flags.kLPI2C_SlaveIrqFlags (*C enumerator*), 651
drivers_RT_RT1020_MIMXRT1024._lpi2c_slave_flags.kLPI2C_SlaveRepeatedStartDetectFlag (*C enumerator*), 650
drivers_RT_RT1020_MIMXRT1024._lpi2c_slave_flags.kLPI2C_SlaveRxReadyFlag (*C enumerator*), 650
drivers_RT_RT1020_MIMXRT1024._lpi2c_slave_flags.kLPI2C_SlaveStopDetectFlag (*C enumerator*), 650
drivers_RT_RT1020_MIMXRT1024._lpi2c_slave_flags.kLPI2C_SlaveTransmitAckFlag (*C enumerator*), 650
drivers_RT_RT1020_MIMXRT1024._lpi2c_slave_flags.kLPI2C_SlaveTxReadyFlag (*C enumerator*), 650
drivers_RT_RT1020_MIMXRT1024._lpi2c_slave_handle (*C struct*), 654
drivers_RT_RT1020_MIMXRT1024._lpi2c_slave_handle.callback (*C var*), 655
drivers_RT_RT1020_MIMXRT1024._lpi2c_slave_handle.eventMask (*C var*), 655
drivers_RT_RT1020_MIMXRT1024._lpi2c_slave_handle.isBusy (*C var*), 655
drivers_RT_RT1020_MIMXRT1024._lpi2c_slave_handle.transfer (*C var*), 655
drivers_RT_RT1020_MIMXRT1024._lpi2c_slave_handle.transferredCount (*C var*), 655
drivers_RT_RT1020_MIMXRT1024._lpi2c_slave_handle.userData (*C var*), 655
drivers_RT_RT1020_MIMXRT1024._lpi2c_slave_handle.wasTransmit (*C var*), 655
drivers_RT_RT1020_MIMXRT1024._lpi2c_slave_transfer (*C struct*), 654
drivers_RT_RT1020_MIMXRT1024._lpi2c_slave_transfer_event (*C enum*), 651
drivers_RT_RT1020_MIMXRT1024._lpi2c_slave_transfer_event.kLPI2C_SlaveAddressMatchEvent (*C enumerator*), 651
drivers_RT_RT1020_MIMXRT1024._lpi2c_slave_transfer_event.kLPI2C_SlaveAllEvents (*C enumerator*), 652
drivers_RT_RT1020_MIMXRT1024._lpi2c_slave_transfer_event.kLPI2C_SlaveCompletionEvent (*C enumerator*), 652
drivers_RT_RT1020_MIMXRT1024._lpi2c_slave_transfer_event.kLPI2C_SlaveReceiveEvent (*C enumerator*), 652
drivers_RT_RT1020_MIMXRT1024._lpi2c_slave_transfer_event.kLPI2C_SlaveRepeatedStartEvent (*C enumerator*), 652
drivers_RT_RT1020_MIMXRT1024._lpi2c_slave_transfer_event.kLPI2C_SlaveTransmitAckEvent (*C enumerator*), 652
drivers_RT_RT1020_MIMXRT1024._lpi2c_slave_transfer_event.kLPI2C_SlaveTransmitEvent (*C enumerator*), 652
drivers_RT_RT1020_MIMXRT1024._lpi2c_slave_transfer.completionStatus (*C var*), 654
drivers_RT_RT1020_MIMXRT1024._lpi2c_slave_transfer.data (*C var*), 654
drivers_RT_RT1020_MIMXRT1024._lpi2c_slave_transfer.dataSize (*C var*), 654
drivers_RT_RT1020_MIMXRT1024._lpi2c_slave_transfer.event (*C var*), 654
drivers_RT_RT1020_MIMXRT1024._lpi2c_slave_transfer.receivedAddress (*C var*), 654
drivers_RT_RT1020_MIMXRT1024._lpi2c_slave_transfer.transferredCount (*C var*), 654
drivers_RT_RT1020_MIMXRT1024._lpspi_clock_phase (*C enum*), 669
drivers_RT_RT1020_MIMXRT1024._lpspi_clock_phase.kLPSPI_ClockPhaseFirstEdge (*C enumerator*), 669
drivers_RT_RT1020_MIMXRT1024._lpspi_clock_phase.kLPSPI_ClockPhaseSecondEdge (*C enumerator*), 669
drivers_RT_RT1020_MIMXRT1024._lpspi_clock_polarity (*C enum*), 669
drivers_RT_RT1020_MIMXRT1024._lpspi_clock_polarity.kLPSPI_ClockPolarityActiveHigh (*C enumerator*), 669
drivers_RT_RT1020_MIMXRT1024._lpspi_clock_polarity.kLPSPI_ClockPolarityActiveLow (*C enumerator*), 669
drivers_RT_RT1020_MIMXRT1024._lpspi_data_out_config (*C enum*), 671
drivers_RT_RT1020_MIMXRT1024._lpspi_data_out_config.kLpspiDataOutRetained (*C enumerator*), 671

drivers_RT_RT1020_MIMXRT1024._lpspi_data_out_config.kLpspiDataOutTristate (*C enumerator*), 671
drivers_RT_RT1020_MIMXRT1024._lpspi_delay_type (*C enum*), 671
drivers_RT_RT1020_MIMXRT1024._lpspi_delay_type.kLPSPI_BetweenTransfer (*C enumerator*), 671
drivers_RT_RT1020_MIMXRT1024._lpspi_delay_type.kLPSPI_LastSckToPcs (*C enumerator*), 671
drivers_RT_RT1020_MIMXRT1024._lpspi_delay_type.kLPSPI_PcsToSck (*C enumerator*), 671
drivers_RT_RT1020_MIMXRT1024._lpspi_dma_enable (*C enum*), 668
drivers_RT_RT1020_MIMXRT1024._lpspi_dma_enable.kLPSPI_RxDmaEnable (*C enumerator*), 668
drivers_RT_RT1020_MIMXRT1024._lpspi_dma_enable.kLPSPI_TxDmaEnable (*C enumerator*), 668
drivers_RT_RT1020_MIMXRT1024._lpspi_flags (*C enum*), 666
drivers_RT_RT1020_MIMXRT1024._lpspi_flags.kLPSPI_AllStatusFlag (*C enumerator*), 667
drivers_RT_RT1020_MIMXRT1024._lpspi_flags.kLPSPI_DataMatchFlag (*C enumerator*), 667
drivers_RT_RT1020_MIMXRT1024._lpspi_flags.kLPSPI_FrameCompleteFlag (*C enumerator*), 667
drivers_RT_RT1020_MIMXRT1024._lpspi_flags.kLPSPI_ModuleBusyFlag (*C enumerator*), 667
drivers_RT_RT1020_MIMXRT1024._lpspi_flags.kLPSPI_ReceiveErrorFlag (*C enumerator*), 667
drivers_RT_RT1020_MIMXRT1024._lpspi_flags.kLPSPI_RxDataReadyFlag (*C enumerator*), 667
drivers_RT_RT1020_MIMXRT1024._lpspi_flags.kLPSPI_TransferCompleteFlag (*C enumerator*), 667
drivers_RT_RT1020_MIMXRT1024._lpspi_flags.kLPSPI_TransmitErrorFlag (*C enumerator*), 667
drivers_RT_RT1020_MIMXRT1024._lpspi_flags.kLPSPI_TxDataRequestFlag (*C enumerator*), 666
drivers_RT_RT1020_MIMXRT1024._lpspi_flags.kLPSPI_WordCompleteFlag (*C enumerator*), 667
drivers_RT_RT1020_MIMXRT1024._lpspi_host_request_select (*C enum*), 670
drivers_RT_RT1020_MIMXRT1024._lpspi_host_request_select.kLPSPI_HostReqExtPin (*C enumerator*), 670
drivers_RT_RT1020_MIMXRT1024._lpspi_host_request_select.kLPSPI_HostReqInternalTrigger (*C enumerator*), 670
drivers_RT_RT1020_MIMXRT1024._lpspi_interrupt_enable (*C enum*), 667
drivers_RT_RT1020_MIMXRT1024._lpspi_interrupt_enable.kLPSPI_AllInterruptEnable (*C enumerator*), 668
drivers_RT_RT1020_MIMXRT1024._lpspi_interrupt_enable.kLPSPI_DataMatchInterruptEnable (*C enumerator*), 668
drivers_RT_RT1020_MIMXRT1024._lpspi_interrupt_enable.kLPSPI_FrameCompleteInterruptEnable (*C enumerator*), 667
drivers_RT_RT1020_MIMXRT1024._lpspi_interrupt_enable.kLPSPI_ReceiveErrorInterruptEnable (*C enumerator*), 668
drivers_RT_RT1020_MIMXRT1024._lpspi_interrupt_enable.kLPSPI_RxInterruptEnable (*C enumerator*), 667
drivers_RT_RT1020_MIMXRT1024._lpspi_interrupt_enable.kLPSPI_TransferCompleteInterruptEnable (*C enumerator*), 667
drivers_RT_RT1020_MIMXRT1024._lpspi_interrupt_enable.kLPSPI_TransmitErrorInterruptEnable (*C enumerator*), 667
drivers_RT_RT1020_MIMXRT1024._lpspi_interrupt_enable.kLPSPI_TxInterruptEnable (*C enumerator*), 667
drivers_RT_RT1020_MIMXRT1024._lpspi_interrupt_enable.kLPSPI_WordCompleteInterruptEnable (*C enumerator*), 667
drivers_RT_RT1020_MIMXRT1024._lpspi_master_config (*C struct*), 675
drivers_RT_RT1020_MIMXRT1024._lpspi_master_config.baudRate (*C var*), 675
drivers_RT_RT1020_MIMXRT1024._lpspi_master_config.betweenTransferDelayInNanoSec (*C var*), 676
drivers_RT_RT1020_MIMXRT1024._lpspi_master_config.bitsPerFrame (*C var*), 675
drivers_RT_RT1020_MIMXRT1024._lpspi_master_config.cpha (*C var*), 675
drivers_RT_RT1020_MIMXRT1024._lpspi_master_config.cpol (*C var*), 675
drivers_RT_RT1020_MIMXRT1024._lpspi_master_config.dataOutConfig (*C var*), 676
drivers_RT_RT1020_MIMXRT1024._lpspi_master_config.direction (*C var*), 675
drivers_RT_RT1020_MIMXRT1024._lpspi_master_config.enableInputDelay (*C var*), 676

drivers_RT_RT1020_MIMXRT1024._lpspi_master_config.lastSckToPcsDelayInNanoSec (*C var*), 676
drivers_RT_RT1020_MIMXRT1024._lpspi_master_config.pcsActiveHighOrLow (*C var*), 676
drivers_RT_RT1020_MIMXRT1024._lpspi_master_config.pcsToSckDelayInNanoSec (*C var*), 675
drivers_RT_RT1020_MIMXRT1024._lpspi_master_config.pinCfg (*C var*), 676
drivers_RT_RT1020_MIMXRT1024._lpspi_master_config.whichPcs (*C var*), 676
drivers_RT_RT1020_MIMXRT1024._lpspi_master_edma_handle (*C struct*), 684
drivers_RT_RT1020_MIMXRT1024._lpspi_master_edma_handle.bytesEachRead (*C var*), 684
drivers_RT_RT1020_MIMXRT1024._lpspi_master_edma_handle.bytesEachWrite (*C var*), 684
drivers_RT_RT1020_MIMXRT1024._lpspi_master_edma_handle.bytesLastRead (*C var*), 684
drivers_RT_RT1020_MIMXRT1024._lpspi_master_edma_handle.callback (*C var*), 686
drivers_RT_RT1020_MIMXRT1024._lpspi_master_edma_handle.dataBytesEveryTime (*C var*), 685
drivers_RT_RT1020_MIMXRT1024._lpspi_master_edma_handle.dmaTransmitTime (*C var*), 685
drivers_RT_RT1020_MIMXRT1024._lpspi_master_edma_handle.edmaRxRegToRxDataHandle (*C var*), 686
drivers_RT_RT1020_MIMXRT1024._lpspi_master_edma_handle.edmaTxDataToTxRegHandle (*C var*), 686
drivers_RT_RT1020_MIMXRT1024._lpspi_master_edma_handle.fifoSize (*C var*), 684
drivers_RT_RT1020_MIMXRT1024._lpspi_master_edma_handle.isByteSwap (*C var*), 684
drivers_RT_RT1020_MIMXRT1024._lpspi_master_edma_handle.isMultiDMATransmit (*C var*), 685
drivers_RT_RT1020_MIMXRT1024._lpspi_master_edma_handle.isPcsContinuous (*C var*), 684
drivers_RT_RT1020_MIMXRT1024._lpspi_master_edma_handle.isThereExtraRxBytes (*C var*), 684
drivers_RT_RT1020_MIMXRT1024._lpspi_master_edma_handle.lastTimeDataBytes (*C var*), 685
drivers_RT_RT1020_MIMXRT1024._lpspi_master_edma_handle.lastTimeTCD (*C var*), 685
drivers_RT_RT1020_MIMXRT1024._lpspi_master_edma_handle.lpspiSoftwareTCD (*C var*), 686
drivers_RT_RT1020_MIMXRT1024._lpspi_master_edma_handle.nbytes (*C var*), 685
drivers_RT_RT1020_MIMXRT1024._lpspi_master_edma_handle.readRegRemainingTimes (*C var*), 685
drivers_RT_RT1020_MIMXRT1024._lpspi_master_edma_handle.rxBuffIfNull (*C var*), 685
drivers_RT_RT1020_MIMXRT1024._lpspi_master_edma_handle.rxData (*C var*), 684
drivers_RT_RT1020_MIMXRT1024._lpspi_master_edma_handle.rxRemainingByteCount (*C var*), 685
drivers_RT_RT1020_MIMXRT1024._lpspi_master_edma_handle.rxWatermark (*C var*), 684
drivers_RT_RT1020_MIMXRT1024._lpspi_master_edma_handle.state (*C var*), 685
drivers_RT_RT1020_MIMXRT1024._lpspi_master_edma_handle.totalByteCount (*C var*), 685
drivers_RT_RT1020_MIMXRT1024._lpspi_master_edma_handle.transferConfigRx (*C var*), 685
drivers_RT_RT1020_MIMXRT1024._lpspi_master_edma_handle.transferConfigTx (*C var*), 685
drivers_RT_RT1020_MIMXRT1024._lpspi_master_edma_handle.transmitCommand (*C var*), 685
drivers_RT_RT1020_MIMXRT1024._lpspi_master_edma_handle.txBuffIfNull (*C var*), 685
drivers_RT_RT1020_MIMXRT1024._lpspi_master_edma_handle.txData (*C var*), 684
drivers_RT_RT1020_MIMXRT1024._lpspi_master_edma_handle.txRemainingByteCount (*C var*), 684
drivers_RT_RT1020_MIMXRT1024._lpspi_master_edma_handle.userData (*C var*), 686
drivers_RT_RT1020_MIMXRT1024._lpspi_master_edma_handle.writeRegRemainingTimes (*C var*), 685
drivers_RT_RT1020_MIMXRT1024._lpspi_master_handle (*C struct*), 677
drivers_RT_RT1020_MIMXRT1024._lpspi_master_handle.bytesEachRead (*C var*), 678
drivers_RT_RT1020_MIMXRT1024._lpspi_master_handle.bytesEachWrite (*C var*), 678
drivers_RT_RT1020_MIMXRT1024._lpspi_master_handle.bytesPerFrame (*C var*), 677
drivers_RT_RT1020_MIMXRT1024._lpspi_master_handle.callback (*C var*), 678
drivers_RT_RT1020_MIMXRT1024._lpspi_master_handle.fifoSize (*C var*), 678
drivers_RT_RT1020_MIMXRT1024._lpspi_master_handle.frameSize (*C var*), 677
drivers_RT_RT1020_MIMXRT1024._lpspi_master_handle.isByteSwap (*C var*), 677
drivers_RT_RT1020_MIMXRT1024._lpspi_master_handle.isPcsContinuous (*C var*), 677
drivers_RT_RT1020_MIMXRT1024._lpspi_master_handle.isTxMask (*C var*), 677
drivers_RT_RT1020_MIMXRT1024._lpspi_master_handle.readRegRemainingTimes (*C var*), 678
drivers_RT_RT1020_MIMXRT1024._lpspi_master_handle.rxData (*C var*), 678
drivers_RT_RT1020_MIMXRT1024._lpspi_master_handle.rxRemainingByteCount (*C var*), 678
drivers_RT_RT1020_MIMXRT1024._lpspi_master_handle.rxWatermark (*C var*), 678

drivers_RT_RT1020_MIMXRT1024._lpspi_master_handle.state (*C var*), 678
drivers_RT_RT1020_MIMXRT1024._lpspi_master_handle.totalByteCount (*C var*), 678
drivers_RT_RT1020_MIMXRT1024._lpspi_master_handle.txBuffIfNull (*C var*), 678
drivers_RT_RT1020_MIMXRT1024._lpspi_master_handle.txData (*C var*), 678
drivers_RT_RT1020_MIMXRT1024._lpspi_master_handle.txRemainingByteCount (*C var*), 678
drivers_RT_RT1020_MIMXRT1024._lpspi_master_handle.userData (*C var*), 678
drivers_RT_RT1020_MIMXRT1024._lpspi_master_handle.writeRegRemainingTimes (*C var*), 678
drivers_RT_RT1020_MIMXRT1024._lpspi_master_handle.writeTcrInIsr (*C var*), 677
drivers_RT_RT1020_MIMXRT1024._lpspi_master_slave_mode (*C enum*), 668
drivers_RT_RT1020_MIMXRT1024._lpspi_master_slave_mode.kLPSPI_Master (*C enumerator*), 668
drivers_RT_RT1020_MIMXRT1024._lpspi_master_slave_mode.kLPSPI_Slave (*C enumerator*), 668
drivers_RT_RT1020_MIMXRT1024._lpspi_match_config (*C enum*), 670
drivers_RT_RT1020_MIMXRT1024._lpspi_match_config.kLPSI_1stWordAndM1EqualsM0andM1 (*C enumerator*), 670
drivers_RT_RT1020_MIMXRT1024._lpspi_match_config.kLPSI_1stWordEqualsM0and2ndWordEqualsM1 (*C enumerator*), 670
drivers_RT_RT1020_MIMXRT1024._lpspi_match_config.kLPSI_1stWordEqualsM0orM1 (*C enumerator*), 670
drivers_RT_RT1020_MIMXRT1024._lpspi_match_config.kLPSI_AnyWordAndM1EqualsM0andM1 (*C enumerator*), 670
drivers_RT_RT1020_MIMXRT1024._lpspi_match_config.kLPSI_AnyWordEqualsM0andNxtWordEqualsM1 (*C enumerator*), 670
drivers_RT_RT1020_MIMXRT1024._lpspi_match_config.kLPSI_AnyWordEqualsM0orM1 (*C enumerator*), 670
drivers_RT_RT1020_MIMXRT1024._lpspi_match_config.kLPSI_MatchDisabled (*C enumerator*), 670
drivers_RT_RT1020_MIMXRT1024._lpspi_pcs_polarity (*C enum*), 669
drivers_RT_RT1020_MIMXRT1024._lpspi_pcs_polarity_config (*C enum*), 668
drivers_RT_RT1020_MIMXRT1024._lpspi_pcs_polarity_config.kLPSPI_PcsActiveHigh (*C enumerator*), 669
drivers_RT_RT1020_MIMXRT1024._lpspi_pcs_polarity_config.kLPSPI_PcsActiveLow (*C enumerator*), 669
drivers_RT_RT1020_MIMXRT1024._lpspi_pcs_polarity.kLPSPI_Pcs0ActiveLow (*C enumerator*), 669
drivers_RT_RT1020_MIMXRT1024._lpspi_pcs_polarity.kLPSPI_Pcs1ActiveLow (*C enumerator*), 669
drivers_RT_RT1020_MIMXRT1024._lpspi_pcs_polarity.kLPSPI_Pcs2ActiveLow (*C enumerator*), 669
drivers_RT_RT1020_MIMXRT1024._lpspi_pcs_polarity.kLPSPI_Pcs3ActiveLow (*C enumerator*), 669
drivers_RT_RT1020_MIMXRT1024._lpspi_pcs_polarity.kLPSPI_PcsAllActiveLow (*C enumerator*), 669
drivers_RT_RT1020_MIMXRT1024._lpspi_pin_config (*C enum*), 670
drivers_RT_RT1020_MIMXRT1024._lpspi_pin_config.kLPSPI_SdiInSdiOut (*C enumerator*), 671
drivers_RT_RT1020_MIMXRT1024._lpspi_pin_config.kLPSPI_SdiInSdoOut (*C enumerator*), 671
drivers_RT_RT1020_MIMXRT1024._lpspi_pin_config.kLPSPI_SdoInSdiOut (*C enumerator*), 671
drivers_RT_RT1020_MIMXRT1024._lpspi_pin_config.kLPSPI_SdoInSdoOut (*C enumerator*), 671
drivers_RT_RT1020_MIMXRT1024._lpspi_shift_direction (*C enum*), 670
drivers_RT_RT1020_MIMXRT1024._lpspi_shift_direction.kLPSPI_LsbFirst (*C enumerator*), 670
drivers_RT_RT1020_MIMXRT1024._lpspi_shift_direction.kLPSPI_MsbFirst (*C enumerator*), 670
drivers_RT_RT1020_MIMXRT1024._lpspi_slave_config (*C struct*), 676
drivers_RT_RT1020_MIMXRT1024._lpspi_slave_config.bitsPerFrame (*C var*), 676
drivers_RT_RT1020_MIMXRT1024._lpspi_slave_config.cpha (*C var*), 676
drivers_RT_RT1020_MIMXRT1024._lpspi_slave_config.cpol (*C var*), 676
drivers_RT_RT1020_MIMXRT1024._lpspi_slave_config.dataOutConfig (*C var*), 677
drivers_RT_RT1020_MIMXRT1024._lpspi_slave_config.direction (*C var*), 676
drivers_RT_RT1020_MIMXRT1024._lpspi_slave_config.pcsActiveHighOrLow (*C var*), 676

drivers_RT_RT1020_MIMXRT1024._lpspi_slave_config.pinCfg (*C var*), 677
drivers_RT_RT1020_MIMXRT1024._lpspi_slave_config.whichPcs (*C var*), 676
drivers_RT_RT1020_MIMXRT1024._lpspi_slave_edma_handle (*C struct*), 686
drivers_RT_RT1020_MIMXRT1024._lpspi_slave_edma_handle.bytesEachRead (*C var*), 686
drivers_RT_RT1020_MIMXRT1024._lpspi_slave_edma_handle.bytesEachWrite (*C var*), 686
drivers_RT_RT1020_MIMXRT1024._lpspi_slave_edma_handle.bytesLastRead (*C var*), 686
drivers_RT_RT1020_MIMXRT1024._lpspi_slave_edma_handle.callback (*C var*), 687
drivers_RT_RT1020_MIMXRT1024._lpspi_slave_edma_handle.edmaRxRegToRxDataHandle (*C var*), 687
drivers_RT_RT1020_MIMXRT1024._lpspi_slave_edma_handle.edmaTxDataToTxRegHandle (*C var*), 687
drivers_RT_RT1020_MIMXRT1024._lpspi_slave_edma_handle.errorCount (*C var*), 687
drivers_RT_RT1020_MIMXRT1024._lpspi_slave_edma_handle.fifoSize (*C var*), 686
drivers_RT_RT1020_MIMXRT1024._lpspi_slave_edma_handle.isByteSwap (*C var*), 686
drivers_RT_RT1020_MIMXRT1024._lpspi_slave_edma_handle.isThereExtraRxBytes (*C var*), 686
drivers_RT_RT1020_MIMXRT1024._lpspi_slave_edma_handle.lpspiSoftwareTCD (*C var*), 687
drivers_RT_RT1020_MIMXRT1024._lpspi_slave_edma_handle.nbytes (*C var*), 686
drivers_RT_RT1020_MIMXRT1024._lpspi_slave_edma_handle.readRegRemainingTimes (*C var*), 687
drivers_RT_RT1020_MIMXRT1024._lpspi_slave_edma_handle.rxBuffIfNull (*C var*), 687
drivers_RT_RT1020_MIMXRT1024._lpspi_slave_edma_handle.rxData (*C var*), 687
drivers_RT_RT1020_MIMXRT1024._lpspi_slave_edma_handle.rxRemainingByteCount (*C var*), 687
drivers_RT_RT1020_MIMXRT1024._lpspi_slave_edma_handle.rxWatermark (*C var*), 686
drivers_RT_RT1020_MIMXRT1024._lpspi_slave_edma_handle.state (*C var*), 687
drivers_RT_RT1020_MIMXRT1024._lpspi_slave_edma_handle.totalByteCount (*C var*), 687
drivers_RT_RT1020_MIMXRT1024._lpspi_slave_edma_handle.txBuffIfNull (*C var*), 687
drivers_RT_RT1020_MIMXRT1024._lpspi_slave_edma_handle.txData (*C var*), 686
drivers_RT_RT1020_MIMXRT1024._lpspi_slave_edma_handle.txRemainingByteCount (*C var*), 687
drivers_RT_RT1020_MIMXRT1024._lpspi_slave_edma_handle.userData (*C var*), 687
drivers_RT_RT1020_MIMXRT1024._lpspi_slave_edma_handle.writeRegRemainingTimes (*C var*), 687
drivers_RT_RT1020_MIMXRT1024._lpspi_slave_handle (*C struct*), 678
drivers_RT_RT1020_MIMXRT1024._lpspi_slave_handle.bytesEachRead (*C var*), 679
drivers_RT_RT1020_MIMXRT1024._lpspi_slave_handle.bytesEachWrite (*C var*), 679
drivers_RT_RT1020_MIMXRT1024._lpspi_slave_handle.callback (*C var*), 679
drivers_RT_RT1020_MIMXRT1024._lpspi_slave_handle.errorCount (*C var*), 679
drivers_RT_RT1020_MIMXRT1024._lpspi_slave_handle.fifoSize (*C var*), 679
drivers_RT_RT1020_MIMXRT1024._lpspi_slave_handle.isByteSwap (*C var*), 679
drivers_RT_RT1020_MIMXRT1024._lpspi_slave_handle.readRegRemainingTimes (*C var*), 679
drivers_RT_RT1020_MIMXRT1024._lpspi_slave_handle.rxData (*C var*), 679
drivers_RT_RT1020_MIMXRT1024._lpspi_slave_handle.rxRemainingByteCount (*C var*), 679
drivers_RT_RT1020_MIMXRT1024._lpspi_slave_handle.rxWatermark (*C var*), 679
drivers_RT_RT1020_MIMXRT1024._lpspi_slave_handle.state (*C var*), 679
drivers_RT_RT1020_MIMXRT1024._lpspi_slave_handle.totalByteCount (*C var*), 679
drivers_RT_RT1020_MIMXRT1024._lpspi_slave_handle.txData (*C var*), 679
drivers_RT_RT1020_MIMXRT1024._lpspi_slave_handle.txRemainingByteCount (*C var*), 679
drivers_RT_RT1020_MIMXRT1024._lpspi_slave_handle.userData (*C var*), 679
drivers_RT_RT1020_MIMXRT1024._lpspi_slave_handle.writeRegRemainingTimes (*C var*), 679
drivers_RT_RT1020_MIMXRT1024._lpspi_transfer (*C struct*), 677
drivers_RT_RT1020_MIMXRT1024._lpspi_transfer_config_flag_for_master (*C enum*), 672
drivers_RT_RT1020_MIMXRT1024._lpspi_transfer_config_flag_for_master.kLPSPI_MasterByteSwap (*C enumerator*), 672
drivers_RT_RT1020_MIMXRT1024._lpspi_transfer_config_flag_for_master.kLPSPI_MasterPcs0 (*C enumerator*), 672
drivers_RT_RT1020_MIMXRT1024._lpspi_transfer_config_flag_for_master.kLPSPI_MasterPcs1 (*C enumerator*), 672
drivers_RT_RT1020_MIMXRT1024._lpspi_transfer_config_flag_for_master.kLPSPI_MasterPcs2 (*C enumerator*), 672

drivers_RT_RT1020_MIMXRT1024._lpspi_transfer_config_flag_for_master.kLPSPI_MasterPcs3 (*C enumerator*), 672
drivers_RT_RT1020_MIMXRT1024._lpspi_transfer_config_flag_for_master.kLPSPI_MasterPcsContinuous (*C enumerator*), 672
drivers_RT_RT1020_MIMXRT1024._lpspi_transfer_config_flag_for_slave (*C enum*), 672
drivers_RT_RT1020_MIMXRT1024._lpspi_transfer_config_flag_for_slave.kLPSPI_SlaveByteSwap (*C enumerator*), 673
drivers_RT_RT1020_MIMXRT1024._lpspi_transfer_config_flag_for_slave.kLPSPI_SlavePcs0 (*C enumerator*), 672
drivers_RT_RT1020_MIMXRT1024._lpspi_transfer_config_flag_for_slave.kLPSPI_SlavePcs1 (*C enumerator*), 672
drivers_RT_RT1020_MIMXRT1024._lpspi_transfer_config_flag_for_slave.kLPSPI_SlavePcs2 (*C enumerator*), 672
drivers_RT_RT1020_MIMXRT1024._lpspi_transfer_config_flag_for_slave.kLPSPI_SlavePcs3 (*C enumerator*), 672
drivers_RT_RT1020_MIMXRT1024._lpspi_transfer_state (*C enum*), 673
drivers_RT_RT1020_MIMXRT1024._lpspi_transfer_state.kLPSPI_Busy (*C enumerator*), 673
drivers_RT_RT1020_MIMXRT1024._lpspi_transfer_state.kLPSPI_Error (*C enumerator*), 673
drivers_RT_RT1020_MIMXRT1024._lpspi_transfer_state.kLPSPI_Idle (*C enumerator*), 673
drivers_RT_RT1020_MIMXRT1024._lpspi_transfer_width (*C enum*), 671
drivers_RT_RT1020_MIMXRT1024._lpspi_transfer_width.kLPSPI_FourBitXfer (*C enumerator*), 671
drivers_RT_RT1020_MIMXRT1024._lpspi_transfer_width.kLPSPI_SingleBitXfer (*C enumerator*), 671
drivers_RT_RT1020_MIMXRT1024._lpspi_transfer_width.kLPSPI_TwoBitXfer (*C enumerator*), 671
drivers_RT_RT1020_MIMXRT1024._lpspi_transfer.configFlags (*C var*), 677
drivers_RT_RT1020_MIMXRT1024._lpspi_transfer.dataSize (*C var*), 677
drivers_RT_RT1020_MIMXRT1024._lpspi_transfer.rxData (*C var*), 677
drivers_RT_RT1020_MIMXRT1024._lpspi_transfer.txData (*C var*), 677
drivers_RT_RT1020_MIMXRT1024._lpspi_which_pcs_config (*C enum*), 668
drivers_RT_RT1020_MIMXRT1024._lpspi_which_pcs_config.kLPSPI_Pcs0 (*C enumerator*), 668
drivers_RT_RT1020_MIMXRT1024._lpspi_which_pcs_config.kLPSPI_Pcs1 (*C enumerator*), 668
drivers_RT_RT1020_MIMXRT1024._lpspi_which_pcs_config.kLPSPI_Pcs2 (*C enumerator*), 668
drivers_RT_RT1020_MIMXRT1024._lpspi_which_pcs_config.kLPSPI_Pcs3 (*C enumerator*), 668
drivers_RT_RT1020_MIMXRT1024._lpuart_config (*C struct*), 705
drivers_RT_RT1020_MIMXRT1024._lpuart_config.baudRate_Bps (*C var*), 705
drivers_RT_RT1020_MIMXRT1024._lpuart_config.dataBitsCount (*C var*), 705
drivers_RT_RT1020_MIMXRT1024._lpuart_config.enableRx (*C var*), 706
drivers_RT_RT1020_MIMXRT1024._lpuart_config.enableRxRTS (*C var*), 706
drivers_RT_RT1020_MIMXRT1024._lpuart_config.enableTx (*C var*), 706
drivers_RT_RT1020_MIMXRT1024._lpuart_config.enableTxCTS (*C var*), 706
drivers_RT_RT1020_MIMXRT1024._lpuart_config.enableTxRTS (*C var*), 706
drivers_RT_RT1020_MIMXRT1024._lpuart_config.inverseTxd (*C var*), 706
drivers_RT_RT1020_MIMXRT1024._lpuart_config.isMsb (*C var*), 705
drivers_RT_RT1020_MIMXRT1024._lpuart_config.parityMode (*C var*), 705
drivers_RT_RT1020_MIMXRT1024._lpuart_config.rtsWatermark (*C var*), 706
drivers_RT_RT1020_MIMXRT1024._lpuart_config.rxFifoWatermark (*C var*), 706
drivers_RT_RT1020_MIMXRT1024._lpuart_config.rxIdleConfig (*C var*), 706
drivers_RT_RT1020_MIMXRT1024._lpuart_config.rxIdleType (*C var*), 706
drivers_RT_RT1020_MIMXRT1024._lpuart_config.stopBitCount (*C var*), 706
drivers_RT_RT1020_MIMXRT1024._lpuart_config.swapTxdRxd (*C var*), 706
drivers_RT_RT1020_MIMXRT1024._lpuart_config.txCtsConfig (*C var*), 706
drivers_RT_RT1020_MIMXRT1024._lpuart_config.txCtsSource (*C var*), 706
drivers_RT_RT1020_MIMXRT1024._lpuart_config.txFifoWatermark (*C var*), 706
drivers_RT_RT1020_MIMXRT1024._lpuart_config.txRtsPolarity (*C var*), 706
drivers_RT_RT1020_MIMXRT1024._lpuart_data_bits (*C enum*), 700

drivers_RT_RT1020_MIMXRT1024._lpuart_data_bits.kLPUART_EightDataBits (*C enumerator*), 700
drivers_RT_RT1020_MIMXRT1024._lpuart_data_bits.kLPUART_SevenDataBits (*C enumerator*), 700
drivers_RT_RT1020_MIMXRT1024._lpuart_edma_handle (*C struct*), 711
drivers_RT_RT1020_MIMXRT1024._lpuart_edma_handle.callback (*C var*), 711
drivers_RT_RT1020_MIMXRT1024._lpuart_edma_handle.nbytes (*C var*), 712
drivers_RT_RT1020_MIMXRT1024._lpuart_edma_handle.rxDataSizeAll (*C var*), 711
drivers_RT_RT1020_MIMXRT1024._lpuart_edma_handle.rxEdmaHandle (*C var*), 712
drivers_RT_RT1020_MIMXRT1024._lpuart_edma_handle.rxState (*C var*), 712
drivers_RT_RT1020_MIMXRT1024._lpuart_edma_handle.txDataSizeAll (*C var*), 712
drivers_RT_RT1020_MIMXRT1024._lpuart_edma_handle.txEdmaHandle (*C var*), 712
drivers_RT_RT1020_MIMXRT1024._lpuart_edma_handle.txState (*C var*), 712
drivers_RT_RT1020_MIMXRT1024._lpuart_edma_handle.userData (*C var*), 711
drivers_RT_RT1020_MIMXRT1024._lpuart_flags (*C enum*), 703
drivers_RT_RT1020_MIMXRT1024._lpuart_flags.kLPUART_AllClearFlags (*C enumerator*), 704
drivers_RT_RT1020_MIMXRT1024._lpuart_flags.kLPUART_AllFlags (*C enumerator*), 704
drivers_RT_RT1020_MIMXRT1024._lpuart_flags.kLPUART_DataMatch1Flag (*C enumerator*), 704
drivers_RT_RT1020_MIMXRT1024._lpuart_flags.kLPUART_DataMatch2Flag (*C enumerator*), 704
drivers_RT_RT1020_MIMXRT1024._lpuart_flags.kLPUART_FramingErrorFlag (*C enumerator*), 703
drivers_RT_RT1020_MIMXRT1024._lpuart_flags.kLPUART_IdleLineFlag (*C enumerator*), 703
drivers_RT_RT1020_MIMXRT1024._lpuart_flags.kLPUART_LinBreakFlag (*C enumerator*), 704
drivers_RT_RT1020_MIMXRT1024._lpuart_flags.kLPUART_NoiseErrorFlag (*C enumerator*), 703
drivers_RT_RT1020_MIMXRT1024._lpuart_flags.kLPUART_ParityErrorFlag (*C enumerator*), 703
drivers_RT_RT1020_MIMXRT1024._lpuart_flags.kLPUART_RxActiveEdgeFlag (*C enumerator*), 704
drivers_RT_RT1020_MIMXRT1024._lpuart_flags.kLPUART_RxActiveFlag (*C enumerator*), 704
drivers_RT_RT1020_MIMXRT1024._lpuart_flags.kLPUART_RxDataRegFullFlag (*C enumerator*), 703
drivers_RT_RT1020_MIMXRT1024._lpuart_flags.kLPUART_RxFifoEmptyFlag (*C enumerator*), 704
drivers_RT_RT1020_MIMXRT1024._lpuart_flags.kLPUART_RxFifoUnderflowFlag (*C enumerator*), 704
drivers_RT_RT1020_MIMXRT1024._lpuart_flags.kLPUART_RxOverrunFlag (*C enumerator*), 703
drivers_RT_RT1020_MIMXRT1024._lpuart_flags.kLPUART_TransmissionCompleteFlag (*C enumerator*), 703
drivers_RT_RT1020_MIMXRT1024._lpuart_flags.kLPUART_TxDataRegEmptyFlag (*C enumerator*), 703
drivers_RT_RT1020_MIMXRT1024._lpuart_flags.kLPUART_TxFifoEmptyFlag (*C enumerator*), 704
drivers_RT_RT1020_MIMXRT1024._lpuart_flags.kLPUART_TxFifoOverflowFlag (*C enumerator*), 704
drivers_RT_RT1020_MIMXRT1024._lpuart_handle (*C struct*), 707
drivers_RT_RT1020_MIMXRT1024._lpuart_handle.___unnamed60___ (*C union*), 708
drivers_RT_RT1020_MIMXRT1024._lpuart_handle.___unnamed60___.txData (*C var*), 708
drivers_RT_RT1020_MIMXRT1024._lpuart_handle.___unnamed60___.txData16 (*C var*), 708
drivers_RT_RT1020_MIMXRT1024._lpuart_handle.___unnamed62___ (*C union*), 708
drivers_RT_RT1020_MIMXRT1024._lpuart_handle.___unnamed62___.rxData (*C var*), 708
drivers_RT_RT1020_MIMXRT1024._lpuart_handle.___unnamed62___.rxData16 (*C var*), 708
drivers_RT_RT1020_MIMXRT1024._lpuart_handle.___unnamed64___ (*C union*), 709
drivers_RT_RT1020_MIMXRT1024._lpuart_handle.___unnamed64___.rxRingBuffer (*C var*), 709
drivers_RT_RT1020_MIMXRT1024._lpuart_handle.___unnamed64___.rxRingBuffer16 (*C var*), 709
drivers_RT_RT1020_MIMXRT1024._lpuart_handle.callback (*C var*), 707
drivers_RT_RT1020_MIMXRT1024._lpuart_handle.is16bitData (*C var*), 708
drivers_RT_RT1020_MIMXRT1024._lpuart_handle.isSevenDataBits (*C var*), 708
drivers_RT_RT1020_MIMXRT1024._lpuart_handle.rxDataSize (*C var*), 707
drivers_RT_RT1020_MIMXRT1024._lpuart_handle.rxDataSizeAll (*C var*), 707

drivers_RT_RT1020_MIMXRT1024._lpuart_handle.rxRingBufferHead (*C var*), 707
drivers_RT_RT1020_MIMXRT1024._lpuart_handle.rxRingBufferSize (*C var*), 707
drivers_RT_RT1020_MIMXRT1024._lpuart_handle.rxRingBufferTail (*C var*), 707
drivers_RT_RT1020_MIMXRT1024._lpuart_handle.rxState (*C var*), 707
drivers_RT_RT1020_MIMXRT1024._lpuart_handle.txDataSize (*C var*), 707
drivers_RT_RT1020_MIMXRT1024._lpuart_handle.txDataSizeAll (*C var*), 707
drivers_RT_RT1020_MIMXRT1024._lpuart_handle.txState (*C var*), 707
drivers_RT_RT1020_MIMXRT1024._lpuart_handle.userData (*C var*), 707
drivers_RT_RT1020_MIMXRT1024._lpuart_idle_config (*C enum*), 701
drivers_RT_RT1020_MIMXRT1024._lpuart_idle_config.kLPUART_IdleCharacter1 (*C enumerator*), 701
drivers_RT_RT1020_MIMXRT1024._lpuart_idle_config.kLPUART_IdleCharacter2 (*C enumerator*), 701
drivers_RT_RT1020_MIMXRT1024._lpuart_idle_config.kLPUART_IdleCharacter4 (*C enumerator*), 701
drivers_RT_RT1020_MIMXRT1024._lpuart_idle_config.kLPUART_IdleCharacter8 (*C enumerator*), 702
drivers_RT_RT1020_MIMXRT1024._lpuart_idle_config.kLPUART_IdleCharacter16 (*C enumerator*), 702
drivers_RT_RT1020_MIMXRT1024._lpuart_idle_config.kLPUART_IdleCharacter32 (*C enumerator*), 702
drivers_RT_RT1020_MIMXRT1024._lpuart_idle_config.kLPUART_IdleCharacter64 (*C enumerator*), 702
drivers_RT_RT1020_MIMXRT1024._lpuart_idle_config.kLPUART_IdleCharacter128 (*C enumerator*), 702
drivers_RT_RT1020_MIMXRT1024._lpuart_idle_type_select (*C enum*), 701
drivers_RT_RT1020_MIMXRT1024._lpuart_idle_type_select.kLPUART_IdleTypeStartBit (*C enumerator*), 701
drivers_RT_RT1020_MIMXRT1024._lpuart_idle_type_select.kLPUART_IdleTypeStopBit (*C enumerator*), 701
drivers_RT_RT1020_MIMXRT1024._lpuart_interrupt_enable (*C enum*), 702
drivers_RT_RT1020_MIMXRT1024._lpuart_interrupt_enable.kLPUART_AllInterruptEnable (*C enumerator*), 703
drivers_RT_RT1020_MIMXRT1024._lpuart_interrupt_enable.kLPUART_FramingErrorInterruptEnable (*C enumerator*), 702
drivers_RT_RT1020_MIMXRT1024._lpuart_interrupt_enable.kLPUART_IdleLineInterruptEnable (*C enumerator*), 702
drivers_RT_RT1020_MIMXRT1024._lpuart_interrupt_enable.kLPUART_LinBreakInterruptEnable (*C enumerator*), 702
drivers_RT_RT1020_MIMXRT1024._lpuart_interrupt_enable.kLPUART_Match1InterruptEnable (*C enumerator*), 703
drivers_RT_RT1020_MIMXRT1024._lpuart_interrupt_enable.kLPUART_Match2InterruptEnable (*C enumerator*), 703
drivers_RT_RT1020_MIMXRT1024._lpuart_interrupt_enable.kLPUART_NoiseErrorInterruptEnable (*C enumerator*), 702
drivers_RT_RT1020_MIMXRT1024._lpuart_interrupt_enable.kLPUART_ParityErrorInterruptEnable (*C enumerator*), 703
drivers_RT_RT1020_MIMXRT1024._lpuart_interrupt_enable.kLPUART_RxActiveEdgeInterruptEnable (*C enumerator*), 702
drivers_RT_RT1020_MIMXRT1024._lpuart_interrupt_enable.kLPUART_RxDataRegFullInterruptEnable (*C enumerator*), 702
drivers_RT_RT1020_MIMXRT1024._lpuart_interrupt_enable.kLPUART_RxFifoUnderflowInterruptEnable (*C enumerator*), 703
drivers_RT_RT1020_MIMXRT1024._lpuart_interrupt_enable.kLPUART_RxOverrunInterruptEnable (*C enumerator*), 702
drivers_RT_RT1020_MIMXRT1024._lpuart_interrupt_enable.kLPUART_TransmissionCompleteInterruptEnable (*C enumerator*), 702
drivers_RT_RT1020_MIMXRT1024._lpuart_interrupt_enable.kLPUART_TxDataRegEmptyInterruptEnable

(*C enumerator*), 702
drivers_RT_RT1020_MIMXRT1024._lpuart_interrupt_enable.kLPUART_TxFifoOverflowInterruptEnable (*C enumerator*), 703
drivers_RT_RT1020_MIMXRT1024._lpuart_parity_mode (*C enum*), 700
drivers_RT_RT1020_MIMXRT1024._lpuart_parity_mode.kLPUART_ParityDisabled (*C enumerator*), 700
drivers_RT_RT1020_MIMXRT1024._lpuart_parity_mode.kLPUART_ParityEven (*C enumerator*), 700
drivers_RT_RT1020_MIMXRT1024._lpuart_parity_mode.kLPUART_ParityOdd (*C enumerator*), 700
drivers_RT_RT1020_MIMXRT1024._lpuart_stop_bit_count (*C enum*), 700
drivers_RT_RT1020_MIMXRT1024._lpuart_stop_bit_count.kLPUART_OneStopBit (*C enumerator*), 700
drivers_RT_RT1020_MIMXRT1024._lpuart_stop_bit_count.kLPUART_TwoStopBit (*C enumerator*), 700
drivers_RT_RT1020_MIMXRT1024._lpuart_transfer (*C struct*), 707
drivers_RT_RT1020_MIMXRT1024._lpuart_transfer.___unnamed58___ (*C union*), 708
drivers_RT_RT1020_MIMXRT1024._lpuart_transfer.___unnamed58___.data (*C var*), 708
drivers_RT_RT1020_MIMXRT1024._lpuart_transfer.___unnamed58___.rxData (*C var*), 708
drivers_RT_RT1020_MIMXRT1024._lpuart_transfer.___unnamed58___.rxData16 (*C var*), 708
drivers_RT_RT1020_MIMXRT1024._lpuart_transfer.___unnamed58___.txData (*C var*), 708
drivers_RT_RT1020_MIMXRT1024._lpuart_transfer.___unnamed58___.txData16 (*C var*), 708
drivers_RT_RT1020_MIMXRT1024._lpuart_transfer.dataSize (*C var*), 707
drivers_RT_RT1020_MIMXRT1024._lpuart_transmit_cts_config (*C enum*), 701
drivers_RT_RT1020_MIMXRT1024._lpuart_transmit_cts_config.kLPUART_CtsSampleAtIdle (*C enumerator*), 701
drivers_RT_RT1020_MIMXRT1024._lpuart_transmit_cts_config.kLPUART_CtsSampleAtStart (*C enumerator*), 701
drivers_RT_RT1020_MIMXRT1024._lpuart_transmit_cts_source (*C enum*), 700
drivers_RT_RT1020_MIMXRT1024._lpuart_transmit_cts_source.kLPUART_CtsSourceMatchResult (*C enumerator*), 701
drivers_RT_RT1020_MIMXRT1024._lpuart_transmit_cts_source.kLPUART_CtsSourcePin (*C enumerator*), 700
drivers_RT_RT1020_MIMXRT1024._lpuart_transmit_rts_polarity (*C enum*), 701
drivers_RT_RT1020_MIMXRT1024._lpuart_transmit_rts_polarity.kLPUART_RtsPolarityHigh (*C enumerator*), 701
drivers_RT_RT1020_MIMXRT1024._lpuart_transmit_rts_polarity.kLPUART_RtsPolarityLow (*C enumerator*), 701
drivers_RT_RT1020_MIMXRT1024._ocotp_timing (*C struct*), 715
drivers_RT_RT1020_MIMXRT1024._ocotp_timing.relax (*C var*), 716
drivers_RT_RT1020_MIMXRT1024._ocotp_timing.strobe_prog (*C var*), 716
drivers_RT_RT1020_MIMXRT1024._ocotp_timing.strobe_read (*C var*), 716
drivers_RT_RT1020_MIMXRT1024._ocotp_timing.wait (*C var*), 716
drivers_RT_RT1020_MIMXRT1024._pit_chnl (*C enum*), 719
drivers_RT_RT1020_MIMXRT1024._pit_chnl.kPIT_Chnl_0 (*C enumerator*), 719
drivers_RT_RT1020_MIMXRT1024._pit_chnl.kPIT_Chnl_1 (*C enumerator*), 719
drivers_RT_RT1020_MIMXRT1024._pit_chnl.kPIT_Chnl_2 (*C enumerator*), 719
drivers_RT_RT1020_MIMXRT1024._pit_chnl.kPIT_Chnl_3 (*C enumerator*), 719
drivers_RT_RT1020_MIMXRT1024._pit_config (*C struct*), 720
drivers_RT_RT1020_MIMXRT1024._pit_config.enableRunInDebug (*C var*), 720
drivers_RT_RT1020_MIMXRT1024._pit_interrupt_enable (*C enum*), 719
drivers_RT_RT1020_MIMXRT1024._pit_interrupt_enable.kPIT_TimerInterruptEnable (*C enumerator*), 719
drivers_RT_RT1020_MIMXRT1024._pit_status_flags (*C enum*), 719
drivers_RT_RT1020_MIMXRT1024._pit_status_flags.kPIT_TimerFlag (*C enumerator*), 719
drivers_RT_RT1020_MIMXRT1024._pmu_1p1_weak_reference_source (*C enum*), 726
drivers_RT_RT1020_MIMXRT1024._pmu_1p1_weak_reference_source.kPMU_1P1WeakReferenceSourceAlt0 (*C enumerator*), 726

drivers_RT_RT1020_MIMXRT1024._pmu_1p1_weak_reference_source.kPMU_1P1WeakReferenceSourceAlt1 (*C enumerator*), 726
drivers_RT_RT1020_MIMXRT1024._pmu_3p0_vbus_voltage_source (*C enum*), 726
drivers_RT_RT1020_MIMXRT1024._pmu_3p0_vbus_voltage_source.kPMU_3P0VBusVoltageSourceAlt0 (*C enumerator*), 726
drivers_RT_RT1020_MIMXRT1024._pmu_3p0_vbus_voltage_source.kPMU_3P0VBusVoltageSourceAlt1 (*C enumerator*), 726
drivers_RT_RT1020_MIMXRT1024._pmu_core_reg_voltage_ramp_rate (*C enum*), 726
drivers_RT_RT1020_MIMXRT1024._pmu_core_reg_voltage_ramp_rate.kPMU_CoreRegVoltageRampRateFast (*C enumerator*), 726
drivers_RT_RT1020_MIMXRT1024._pmu_core_reg_voltage_ramp_rate.kPMU_CoreRegVoltageRampRateMediuml (*C enumerator*), 727
drivers_RT_RT1020_MIMXRT1024._pmu_core_reg_voltage_ramp_rate.kPMU_CoreRegVoltageRampRateMediumS (*C enumerator*), 727
drivers_RT_RT1020_MIMXRT1024._pmu_core_reg_voltage_ramp_rate.kPMU_CoreRegVoltageRampRateSlow (*C enumerator*), 727
drivers_RT_RT1020_MIMXRT1024._pmu_power_bandgap (*C enum*), 727
drivers_RT_RT1020_MIMXRT1024._pmu_power_bandgap.kPMU_LowPowerBandgap (*C enumerator*), 727
drivers_RT_RT1020_MIMXRT1024._pmu_power_bandgap.kPMU_NormalPowerBandgap (*C enumerator*), 727
drivers_RT_RT1020_MIMXRT1024._pwm_channels (*C enum*), 733
drivers_RT_RT1020_MIMXRT1024._pwm_channels.kPWM_PwmA (*C enumerator*), 733
drivers_RT_RT1020_MIMXRT1024._pwm_channels.kPWM_PwmB (*C enumerator*), 733
drivers_RT_RT1020_MIMXRT1024._pwm_channels.kPWM_PwmX (*C enumerator*), 733
drivers_RT_RT1020_MIMXRT1024._pwm_chnl_pair_operation (*C enum*), 739
drivers_RT_RT1020_MIMXRT1024._pwm_chnl_pair_operation.kPWM_ComplementaryPwmA (*C enumerator*), 739
drivers_RT_RT1020_MIMXRT1024._pwm_chnl_pair_operation.kPWM_ComplementaryPwmB (*C enumerator*), 739
drivers_RT_RT1020_MIMXRT1024._pwm_chnl_pair_operation.kPWM_Independent (*C enumerator*), 739
drivers_RT_RT1020_MIMXRT1024._pwm_clock_prescale (*C enum*), 734
drivers_RT_RT1020_MIMXRT1024._pwm_clock_prescale.kPWM_Prescale_Divide_1 (*C enumerator*), 734
drivers_RT_RT1020_MIMXRT1024._pwm_clock_prescale.kPWM_Prescale_Divide_2 (*C enumerator*), 734
drivers_RT_RT1020_MIMXRT1024._pwm_clock_prescale.kPWM_Prescale_Divide_4 (*C enumerator*), 734
drivers_RT_RT1020_MIMXRT1024._pwm_clock_prescale.kPWM_Prescale_Divide_8 (*C enumerator*), 734
drivers_RT_RT1020_MIMXRT1024._pwm_clock_prescale.kPWM_Prescale_Divide_16 (*C enumerator*), 734
drivers_RT_RT1020_MIMXRT1024._pwm_clock_prescale.kPWM_Prescale_Divide_32 (*C enumerator*), 734
drivers_RT_RT1020_MIMXRT1024._pwm_clock_prescale.kPWM_Prescale_Divide_64 (*C enumerator*), 735
drivers_RT_RT1020_MIMXRT1024._pwm_clock_prescale.kPWM_Prescale_Divide_128 (*C enumerator*), 735
drivers_RT_RT1020_MIMXRT1024._pwm_clock_source (*C enum*), 734
drivers_RT_RT1020_MIMXRT1024._pwm_clock_source.kPWM_BusClock (*C enumerator*), 734
drivers_RT_RT1020_MIMXRT1024._pwm_clock_source.kPWM_ExternalClock (*C enumerator*), 734
drivers_RT_RT1020_MIMXRT1024._pwm_clock_source.kPWM_Submodule0Clock (*C enumerator*), 734
drivers_RT_RT1020_MIMXRT1024._pwm_config (*C struct*), 754
drivers_RT_RT1020_MIMXRT1024._pwm_config.clockSource (*C var*), 755
drivers_RT_RT1020_MIMXRT1024._pwm_config.enableDebugMode (*C var*), 755

drivers_RT_RT1020_MIMXRT1024._pwm_config.forceTrigger (*C var*), 755
drivers_RT_RT1020_MIMXRT1024._pwm_config.initializationControl (*C var*), 755
drivers_RT_RT1020_MIMXRT1024._pwm_config.pairOperation (*C var*), 755
drivers_RT_RT1020_MIMXRT1024._pwm_config.prescale (*C var*), 755
drivers_RT_RT1020_MIMXRT1024._pwm_config.reloadFrequency (*C var*), 755
drivers_RT_RT1020_MIMXRT1024._pwm_config.reloadLogic (*C var*), 755
drivers_RT_RT1020_MIMXRT1024._pwm_config.reloadSelect (*C var*), 755
drivers_RT_RT1020_MIMXRT1024._pwm_dma_enable (*C enum*), 742
drivers_RT_RT1020_MIMXRT1024._pwm_dma_enable.kPWM_CaptureA0DMAEnable (*C enumerator*), 742
drivers_RT_RT1020_MIMXRT1024._pwm_dma_enable.kPWM_CaptureA1DMAEnable (*C enumerator*), 742
drivers_RT_RT1020_MIMXRT1024._pwm_dma_enable.kPWM_CaptureB0DMAEnable (*C enumerator*), 742
drivers_RT_RT1020_MIMXRT1024._pwm_dma_enable.kPWM_CaptureB1DMAEnable (*C enumerator*), 742
drivers_RT_RT1020_MIMXRT1024._pwm_dma_enable.kPWM_CaptureX0DMAEnable (*C enumerator*), 742
drivers_RT_RT1020_MIMXRT1024._pwm_dma_enable.kPWM_CaptureX1DMAEnable (*C enumerator*), 742
drivers_RT_RT1020_MIMXRT1024._pwm_dma_source_select (*C enum*), 742
drivers_RT_RT1020_MIMXRT1024._pwm_dma_source_select.kPWM_DMALocalReload (*C enumerator*), 743
drivers_RT_RT1020_MIMXRT1024._pwm_dma_source_select.kPWM_DMALocalSync (*C enumerator*), 743
drivers_RT_RT1020_MIMXRT1024._pwm_dma_source_select.kPWM_DMARequestDisable (*C enumerator*), 742
drivers_RT_RT1020_MIMXRT1024._pwm_dma_source_select.kPWM_DMAWatermarksEnable (*C enumerator*), 743
drivers_RT_RT1020_MIMXRT1024._pwm_fault_channels (*C enum*), 738
drivers_RT_RT1020_MIMXRT1024._pwm_fault_channels.kPWM_faultchannel_0 (*C enumerator*), 738
drivers_RT_RT1020_MIMXRT1024._pwm_fault_clear (*C enum*), 744
drivers_RT_RT1020_MIMXRT1024._pwm_fault_clear.kPWM_Automatic (*C enumerator*), 744
drivers_RT_RT1020_MIMXRT1024._pwm_fault_clear.kPWM_ManualNormal (*C enumerator*), 744
drivers_RT_RT1020_MIMXRT1024._pwm_fault_clear.kPWM_ManualSafety (*C enumerator*), 744
drivers_RT_RT1020_MIMXRT1024._pwm_fault_disable (*C enum*), 737
drivers_RT_RT1020_MIMXRT1024._pwm_fault_disable.kPWM_FaultDisable_0 (*C enumerator*), 737
drivers_RT_RT1020_MIMXRT1024._pwm_fault_disable.kPWM_FaultDisable_1 (*C enumerator*), 738
drivers_RT_RT1020_MIMXRT1024._pwm_fault_disable.kPWM_FaultDisable_2 (*C enumerator*), 738
drivers_RT_RT1020_MIMXRT1024._pwm_fault_disable.kPWM_FaultDisable_3 (*C enumerator*), 738
drivers_RT_RT1020_MIMXRT1024._pwm_fault_input (*C enum*), 737
drivers_RT_RT1020_MIMXRT1024._pwm_fault_input_filter_param (*C struct*), 755
drivers_RT_RT1020_MIMXRT1024._pwm_fault_input_filter_param.faultFilterCount (*C var*), 755
drivers_RT_RT1020_MIMXRT1024._pwm_fault_input_filter_param.faultFilterPeriod (*C var*), 755
drivers_RT_RT1020_MIMXRT1024._pwm_fault_input_filter_param.faultGlitchStretch (*C var*), 755
drivers_RT_RT1020_MIMXRT1024._pwm_fault_input.kPWM_Fault_0 (*C enumerator*), 737
drivers_RT_RT1020_MIMXRT1024._pwm_fault_input.kPWM_Fault_1 (*C enumerator*), 737
drivers_RT_RT1020_MIMXRT1024._pwm_fault_input.kPWM_Fault_2 (*C enumerator*), 737
drivers_RT_RT1020_MIMXRT1024._pwm_fault_input.kPWM_Fault_3 (*C enumerator*), 737
drivers_RT_RT1020_MIMXRT1024._pwm_fault_param (*C struct*), 755
drivers_RT_RT1020_MIMXRT1024._pwm_fault_param.enableCombinationalPath (*C var*), 756
drivers_RT_RT1020_MIMXRT1024._pwm_fault_param.faultClearingMode (*C var*), 756
drivers_RT_RT1020_MIMXRT1024._pwm_fault_param.faultLevel (*C var*), 756

drivers_RT_RT1020_MIMXRT1024._pwm_fault_param.recoverMode (*C var*), 756
drivers_RT_RT1020_MIMXRT1024._pwm_fault_recovery_mode (*C enum*), 739
drivers_RT_RT1020_MIMXRT1024._pwm_fault_recovery_mode.kPWM_NoRecovery (*C enumerator*), 739
drivers_RT_RT1020_MIMXRT1024._pwm_fault_recovery_mode.kPWM_RecoverFullCycle (*C enumerator*), 739
drivers_RT_RT1020_MIMXRT1024._pwm_fault_recovery_mode.kPWM_RecoverHalfAndFullCycle (*C enumerator*), 739
drivers_RT_RT1020_MIMXRT1024._pwm_fault_recovery_mode.kPWM_RecoverHalfCycle (*C enumerator*), 739
drivers_RT_RT1020_MIMXRT1024._pwm_fault_state (*C enum*), 743
drivers_RT_RT1020_MIMXRT1024._pwm_fault_state.kPWM_PwmFaultState0 (*C enumerator*), 744
drivers_RT_RT1020_MIMXRT1024._pwm_fault_state.kPWM_PwmFaultState1 (*C enumerator*), 744
drivers_RT_RT1020_MIMXRT1024._pwm_fault_state.kPWM_PwmFaultState2 (*C enumerator*), 744
drivers_RT_RT1020_MIMXRT1024._pwm_fault_state.kPWM_PwmFaultState3 (*C enumerator*), 744
drivers_RT_RT1020_MIMXRT1024._pwm_force_output_trigger (*C enum*), 735
drivers_RT_RT1020_MIMXRT1024._pwm_force_output_trigger.kPWM_Force_External (*C enumerator*), 735
drivers_RT_RT1020_MIMXRT1024._pwm_force_output_trigger.kPWM_Force_ExternalSync (*C enumerator*), 735
drivers_RT_RT1020_MIMXRT1024._pwm_force_output_trigger.kPWM_Force_Local (*C enumerator*), 735
drivers_RT_RT1020_MIMXRT1024._pwm_force_output_trigger.kPWM_Force_LocalReload (*C enumerator*), 735
drivers_RT_RT1020_MIMXRT1024._pwm_force_output_trigger.kPWM_Force_LocalSync (*C enumerator*), 735
drivers_RT_RT1020_MIMXRT1024._pwm_force_output_trigger.kPWM_Force_Master (*C enumerator*), 735
drivers_RT_RT1020_MIMXRT1024._pwm_force_output_trigger.kPWM_Force_MasterReload (*C enumerator*), 735
drivers_RT_RT1020_MIMXRT1024._pwm_force_output_trigger.kPWM_Force_MasterSync (*C enumerator*), 735
drivers_RT_RT1020_MIMXRT1024._pwm_force_signal (*C enum*), 738
drivers_RT_RT1020_MIMXRT1024._pwm_force_signal.kPWM_InvertedPwm (*C enumerator*), 738
drivers_RT_RT1020_MIMXRT1024._pwm_force_signal.kPWM_SoftwareControl (*C enumerator*), 738
drivers_RT_RT1020_MIMXRT1024._pwm_force_signal.kPWM_UseExternal (*C enumerator*), 738
drivers_RT_RT1020_MIMXRT1024._pwm_force_signal.kPWM_UsePwm (*C enumerator*), 738
drivers_RT_RT1020_MIMXRT1024._pwm_init_source (*C enum*), 736
drivers_RT_RT1020_MIMXRT1024._pwm_init_source.kPWM_Initialize_ExtSync (*C enumerator*), 736
drivers_RT_RT1020_MIMXRT1024._pwm_init_source.kPWM_Initialize_LocalSync (*C enumerator*), 736
drivers_RT_RT1020_MIMXRT1024._pwm_init_source.kPWM_Initialize_MasterReload (*C enumerator*), 736
drivers_RT_RT1020_MIMXRT1024._pwm_init_source.kPWM_Initialize_MasterSync (*C enumerator*), 736
drivers_RT_RT1020_MIMXRT1024._pwm_input_capture_edge (*C enum*), 738
drivers_RT_RT1020_MIMXRT1024._pwm_input_capture_edge.kPWM_Disable (*C enumerator*), 738
drivers_RT_RT1020_MIMXRT1024._pwm_input_capture_edge.kPWM_FallingEdge (*C enumerator*), 738
drivers_RT_RT1020_MIMXRT1024._pwm_input_capture_edge.kPWM_RiseAndFallEdge (*C enumerator*), 738

drivers_RT_RT1020_MIMXRT1024._pwm_input_capture_edge.kPWM_RisingEdge (*C enumerator*), 738
drivers_RT_RT1020_MIMXRT1024._pwm_input_capture_param (*C struct*), 756
drivers_RT_RT1020_MIMXRT1024._pwm_input_capture_param.captureInputSel (*C var*), 756
drivers_RT_RT1020_MIMXRT1024._pwm_input_capture_param.edge0 (*C var*), 756
drivers_RT_RT1020_MIMXRT1024._pwm_input_capture_param.edge1 (*C var*), 756
drivers_RT_RT1020_MIMXRT1024._pwm_input_capture_param.edgeCompareValue (*C var*), 756
drivers_RT_RT1020_MIMXRT1024._pwm_input_capture_param.enableOneShotCapture (*C var*), 756
drivers_RT_RT1020_MIMXRT1024._pwm_input_capture_param.fifoWatermark (*C var*), 756
drivers_RT_RT1020_MIMXRT1024._pwm_interrupt_enable (*C enum*), 739
drivers_RT_RT1020_MIMXRT1024._pwm_interrupt_enable.kPWM_CaptureA0InterruptEnable (*C enumerator*), 740
drivers_RT_RT1020_MIMXRT1024._pwm_interrupt_enable.kPWM_CaptureA1InterruptEnable (*C enumerator*), 740
drivers_RT_RT1020_MIMXRT1024._pwm_interrupt_enable.kPWM_CaptureB0InterruptEnable (*C enumerator*), 740
drivers_RT_RT1020_MIMXRT1024._pwm_interrupt_enable.kPWM_CaptureB1InterruptEnable (*C enumerator*), 740
drivers_RT_RT1020_MIMXRT1024._pwm_interrupt_enable.kPWM_CaptureX0InterruptEnable (*C enumerator*), 740
drivers_RT_RT1020_MIMXRT1024._pwm_interrupt_enable.kPWM_CaptureX1InterruptEnable (*C enumerator*), 740
drivers_RT_RT1020_MIMXRT1024._pwm_interrupt_enable.kPWM_CompareVal0InterruptEnable (*C enumerator*), 740
drivers_RT_RT1020_MIMXRT1024._pwm_interrupt_enable.kPWM_CompareVal1InterruptEnable (*C enumerator*), 740
drivers_RT_RT1020_MIMXRT1024._pwm_interrupt_enable.kPWM_CompareVal2InterruptEnable (*C enumerator*), 740
drivers_RT_RT1020_MIMXRT1024._pwm_interrupt_enable.kPWM_CompareVal3InterruptEnable (*C enumerator*), 740
drivers_RT_RT1020_MIMXRT1024._pwm_interrupt_enable.kPWM_CompareVal4InterruptEnable (*C enumerator*), 740
drivers_RT_RT1020_MIMXRT1024._pwm_interrupt_enable.kPWM_CompareVal5InterruptEnable (*C enumerator*), 740
drivers_RT_RT1020_MIMXRT1024._pwm_interrupt_enable.kPWM_Fault0InterruptEnable (*C enumerator*), 740
drivers_RT_RT1020_MIMXRT1024._pwm_interrupt_enable.kPWM_Fault1InterruptEnable (*C enumerator*), 740
drivers_RT_RT1020_MIMXRT1024._pwm_interrupt_enable.kPWM_Fault2InterruptEnable (*C enumerator*), 741
drivers_RT_RT1020_MIMXRT1024._pwm_interrupt_enable.kPWM_Fault3InterruptEnable (*C enumerator*), 741
drivers_RT_RT1020_MIMXRT1024._pwm_interrupt_enable.kPWM_ReloadErrorInterruptEnable (*C enumerator*), 740
drivers_RT_RT1020_MIMXRT1024._pwm_interrupt_enable.kPWM_ReloadInterruptEnable (*C enumerator*), 740
drivers_RT_RT1020_MIMXRT1024._pwm_level_select (*C enum*), 743
drivers_RT_RT1020_MIMXRT1024._pwm_level_select.kPWM_HighTrue (*C enumerator*), 743
drivers_RT_RT1020_MIMXRT1024._pwm_level_select.kPWM_LowTrue (*C enumerator*), 743
drivers_RT_RT1020_MIMXRT1024._pwm_load_frequency (*C enum*), 736
drivers_RT_RT1020_MIMXRT1024._pwm_load_frequency.kPWM_LoadEvery2Oportunity (*C enumerator*), 736
drivers_RT_RT1020_MIMXRT1024._pwm_load_frequency.kPWM_LoadEvery3Oportunity (*C enumerator*), 736
drivers_RT_RT1020_MIMXRT1024._pwm_load_frequency.kPWM_LoadEvery4Oportunity (*C enumerator*), 736
drivers_RT_RT1020_MIMXRT1024._pwm_load_frequency.kPWM_LoadEvery5Oportunity (*C enu-*

merator), 736
drivers_RT_RT1020_MIMXRT1024.__pwm_load_frequency.kPWM_LoadEvery6Oportunity (*C enumerator*), 736
drivers_RT_RT1020_MIMXRT1024.__pwm_load_frequency.kPWM_LoadEvery7Oportunity (*C enumerator*), 736
drivers_RT_RT1020_MIMXRT1024.__pwm_load_frequency.kPWM_LoadEvery8Oportunity (*C enumerator*), 737
drivers_RT_RT1020_MIMXRT1024.__pwm_load_frequency.kPWM_LoadEvery9Oportunity (*C enumerator*), 737
drivers_RT_RT1020_MIMXRT1024.__pwm_load_frequency.kPWM_LoadEvery10Oportunity (*C enumerator*), 737
drivers_RT_RT1020_MIMXRT1024.__pwm_load_frequency.kPWM_LoadEvery11Oportunity (*C enumerator*), 737
drivers_RT_RT1020_MIMXRT1024.__pwm_load_frequency.kPWM_LoadEvery12Oportunity (*C enumerator*), 737
drivers_RT_RT1020_MIMXRT1024.__pwm_load_frequency.kPWM_LoadEvery13Oportunity (*C enumerator*), 737
drivers_RT_RT1020_MIMXRT1024.__pwm_load_frequency.kPWM_LoadEvery14Oportunity (*C enumerator*), 737
drivers_RT_RT1020_MIMXRT1024.__pwm_load_frequency.kPWM_LoadEvery15Oportunity (*C enumerator*), 737
drivers_RT_RT1020_MIMXRT1024.__pwm_load_frequency.kPWM_LoadEvery16Oportunity (*C enumerator*), 737
drivers_RT_RT1020_MIMXRT1024.__pwm_load_frequency.kPWM_LoadEveryOportunity (*C enumerator*), 736
drivers_RT_RT1020_MIMXRT1024.__pwm_mode (*C enum*), 743
drivers_RT_RT1020_MIMXRT1024.__pwm_mode.kPWM_CenterAligned (*C enumerator*), 743
drivers_RT_RT1020_MIMXRT1024.__pwm_mode.kPWM_EdgeAligned (*C enumerator*), 743
drivers_RT_RT1020_MIMXRT1024.__pwm_mode.kPWM_SignedCenterAligned (*C enumerator*), 743
drivers_RT_RT1020_MIMXRT1024.__pwm_mode.kPWM_SignedEdgeAligned (*C enumerator*), 743
drivers_RT_RT1020_MIMXRT1024.__pwm_module_control (*C enum*), 744
drivers_RT_RT1020_MIMXRT1024.__pwm_module_control.kPWM_Control_Module_0 (*C enumerator*), 744
drivers_RT_RT1020_MIMXRT1024.__pwm_module_control.kPWM_Control_Module_1 (*C enumerator*), 744
drivers_RT_RT1020_MIMXRT1024.__pwm_module_control.kPWM_Control_Module_2 (*C enumerator*), 744
drivers_RT_RT1020_MIMXRT1024.__pwm_module_control.kPWM_Control_Module_3 (*C enumerator*), 745
drivers_RT_RT1020_MIMXRT1024.__pwm_output_state (*C enum*), 735
drivers_RT_RT1020_MIMXRT1024.__pwm_output_state.kPWM_HighState (*C enumerator*), 735
drivers_RT_RT1020_MIMXRT1024.__pwm_output_state.kPWM_InvertState (*C enumerator*), 736
drivers_RT_RT1020_MIMXRT1024.__pwm_output_state.kPWM_LowState (*C enumerator*), 735
drivers_RT_RT1020_MIMXRT1024.__pwm_output_state.kPWM_MaskState (*C enumerator*), 736
drivers_RT_RT1020_MIMXRT1024.__pwm_output_state.kPWM_NormalState (*C enumerator*), 735
drivers_RT_RT1020_MIMXRT1024.__pwm_register_reload (*C enum*), 739
drivers_RT_RT1020_MIMXRT1024.__pwm_register_reload.kPWM_ReloadImmediate (*C enumerator*), 739
drivers_RT_RT1020_MIMXRT1024.__pwm_register_reload.kPWM_ReloadPwmFullCycle (*C enumerator*), 739
drivers_RT_RT1020_MIMXRT1024.__pwm_register_reload.kPWM_ReloadPwmHalfAndFullCycle (*C enumerator*), 739
drivers_RT_RT1020_MIMXRT1024.__pwm_register_reload.kPWM_ReloadPwmHalfCycle (*C enumerator*), 739
drivers_RT_RT1020_MIMXRT1024.__pwm_reload_source_select (*C enum*), 744
drivers_RT_RT1020_MIMXRT1024.__pwm_reload_source_select.kPWM_LocalReload (*C enumerator*), 744
drivers_RT_RT1020_MIMXRT1024.__pwm_reload_source_select.kPWM_MasterReload (*C enumer-*

ator), 744
drivers_RT_RT1020_MIMXRT1024._pwm_signal_param (*C struct*), 754
drivers_RT_RT1020_MIMXRT1024._pwm_signal_param.deadtimeValue (*C var*), 754
drivers_RT_RT1020_MIMXRT1024._pwm_signal_param.dutyCyclePercent (*C var*), 754
drivers_RT_RT1020_MIMXRT1024._pwm_signal_param.faultState (*C var*), 754
drivers_RT_RT1020_MIMXRT1024._pwm_signal_param.level (*C var*), 754
drivers_RT_RT1020_MIMXRT1024._pwm_signal_param.pwmChannel (*C var*), 754
drivers_RT_RT1020_MIMXRT1024._pwm_signal_param.pwmchannelenable (*C var*), 754
drivers_RT_RT1020_MIMXRT1024._pwm_status_flags (*C enum*), 741
drivers_RT_RT1020_MIMXRT1024._pwm_status_flags.kPWM_CaptureA0Flag (*C enumerator*), 741
drivers_RT_RT1020_MIMXRT1024._pwm_status_flags.kPWM_CaptureA1Flag (*C enumerator*), 741
drivers_RT_RT1020_MIMXRT1024._pwm_status_flags.kPWM_CaptureB0Flag (*C enumerator*), 741
drivers_RT_RT1020_MIMXRT1024._pwm_status_flags.kPWM_CaptureB1Flag (*C enumerator*), 741
drivers_RT_RT1020_MIMXRT1024._pwm_status_flags.kPWM_CaptureX0Flag (*C enumerator*), 741
drivers_RT_RT1020_MIMXRT1024._pwm_status_flags.kPWM_CaptureX1Flag (*C enumerator*), 741
drivers_RT_RT1020_MIMXRT1024._pwm_status_flags.kPWM_CompareVal0Flag (*C enumerator*), 741
drivers_RT_RT1020_MIMXRT1024._pwm_status_flags.kPWM_CompareVal1Flag (*C enumerator*), 741
drivers_RT_RT1020_MIMXRT1024._pwm_status_flags.kPWM_CompareVal2Flag (*C enumerator*), 741
drivers_RT_RT1020_MIMXRT1024._pwm_status_flags.kPWM_CompareVal3Flag (*C enumerator*), 741
drivers_RT_RT1020_MIMXRT1024._pwm_status_flags.kPWM_CompareVal4Flag (*C enumerator*), 741
drivers_RT_RT1020_MIMXRT1024._pwm_status_flags.kPWM_CompareVal5Flag (*C enumerator*), 741
drivers_RT_RT1020_MIMXRT1024._pwm_status_flags.kPWM_Fault0Flag (*C enumerator*), 742
drivers_RT_RT1020_MIMXRT1024._pwm_status_flags.kPWM_Fault1Flag (*C enumerator*), 742
drivers_RT_RT1020_MIMXRT1024._pwm_status_flags.kPWM_Fault2Flag (*C enumerator*), 742
drivers_RT_RT1020_MIMXRT1024._pwm_status_flags.kPWM_Fault3Flag (*C enumerator*), 742
drivers_RT_RT1020_MIMXRT1024._pwm_status_flags.kPWM_RegUpdatedFlag (*C enumerator*), 742
drivers_RT_RT1020_MIMXRT1024._pwm_status_flags.kPWM_ReloadErrorFlag (*C enumerator*), 742
drivers_RT_RT1020_MIMXRT1024._pwm_status_flags.kPWM_ReloadFlag (*C enumerator*), 741
drivers_RT_RT1020_MIMXRT1024._pwm_submodule (*C enum*), 732
drivers_RT_RT1020_MIMXRT1024._pwm_submodule.kPWM_Module_0 (*C enumerator*), 732
drivers_RT_RT1020_MIMXRT1024._pwm_submodule.kPWM_Module_1 (*C enumerator*), 733
drivers_RT_RT1020_MIMXRT1024._pwm_submodule.kPWM_Module_2 (*C enumerator*), 733
drivers_RT_RT1020_MIMXRT1024._pwm_value_register (*C enum*), 733
drivers_RT_RT1020_MIMXRT1024._pwm_value_register_mask (*C enum*), 733
drivers_RT_RT1020_MIMXRT1024._pwm_value_register_mask.kPWM_ValueRegisterMask_0 (*C enumerator*), 733
drivers_RT_RT1020_MIMXRT1024._pwm_value_register_mask.kPWM_ValueRegisterMask_1 (*C enumerator*), 733
drivers_RT_RT1020_MIMXRT1024._pwm_value_register_mask.kPWM_ValueRegisterMask_2 (*C enumerator*), 734
drivers_RT_RT1020_MIMXRT1024._pwm_value_register_mask.kPWM_ValueRegisterMask_3 (*C enumerator*), 734
drivers_RT_RT1020_MIMXRT1024._pwm_value_register_mask.kPWM_ValueRegisterMask_4 (*C enumerator*), 734

drivers_RT_RT1020_MIMXRT1024._pwm_value_register_mask.kPWM_ValueRegisterMask_5 (*C enumerator*), 734
drivers_RT_RT1020_MIMXRT1024._pwm_value_register.kPWM_ValueRegister_0 (*C enumerator*), 733
drivers_RT_RT1020_MIMXRT1024._pwm_value_register.kPWM_ValueRegister_1 (*C enumerator*), 733
drivers_RT_RT1020_MIMXRT1024._pwm_value_register.kPWM_ValueRegister_2 (*C enumerator*), 733
drivers_RT_RT1020_MIMXRT1024._pwm_value_register.kPWM_ValueRegister_3 (*C enumerator*), 733
drivers_RT_RT1020_MIMXRT1024._pwm_value_register.kPWM_ValueRegister_4 (*C enumerator*), 733
drivers_RT_RT1020_MIMXRT1024._pwm_value_register.kPWM_ValueRegister_5 (*C enumerator*), 733
drivers_RT_RT1020_MIMXRT1024._pwm_watermark_control (*C enum*), 743
drivers_RT_RT1020_MIMXRT1024._pwm_watermark_control.kPWM_FIFOWatermarksAND (*C enumerator*), 743
drivers_RT_RT1020_MIMXRT1024._pwm_watermark_control.kPWM_FIFOWatermarksOR (*C enumerator*), 743
drivers_RT_RT1020_MIMXRT1024._qtmr_channel_selection (*C enum*), 764
drivers_RT_RT1020_MIMXRT1024._qtmr_channel_selection.kQTMR_Channel_0 (*C enumerator*), 765
drivers_RT_RT1020_MIMXRT1024._qtmr_channel_selection.kQTMR_Channel_1 (*C enumerator*), 765
drivers_RT_RT1020_MIMXRT1024._qtmr_channel_selection.kQTMR_Channel_2 (*C enumerator*), 765
drivers_RT_RT1020_MIMXRT1024._qtmr_channel_selection.kQTMR_Channel_3 (*C enumerator*), 765
drivers_RT_RT1020_MIMXRT1024._qtmr_config (*C struct*), 768
drivers_RT_RT1020_MIMXRT1024._qtmr_config.debugMode (*C var*), 768
drivers_RT_RT1020_MIMXRT1024._qtmr_config.enableExternalForce (*C var*), 768
drivers_RT_RT1020_MIMXRT1024._qtmr_config.enableMasterMode (*C var*), 768
drivers_RT_RT1020_MIMXRT1024._qtmr_config.faultFilterCount (*C var*), 768
drivers_RT_RT1020_MIMXRT1024._qtmr_config.faultFilterPeriod (*C var*), 768
drivers_RT_RT1020_MIMXRT1024._qtmr_config.primarySource (*C var*), 768
drivers_RT_RT1020_MIMXRT1024._qtmr_config.secondarySource (*C var*), 768
drivers_RT_RT1020_MIMXRT1024._qtmr_counting_mode (*C enum*), 761
drivers_RT_RT1020_MIMXRT1024._qtmr_counting_mode.kQTMR_CascadeCount (*C enumerator*), 762
drivers_RT_RT1020_MIMXRT1024._qtmr_counting_mode.kQTMR_NoOperation (*C enumerator*), 761
drivers_RT_RT1020_MIMXRT1024._qtmr_counting_mode.kQTMR_PriSrcRiseAndFallEdge (*C enumerator*), 762
drivers_RT_RT1020_MIMXRT1024._qtmr_counting_mode.kQTMR_PriSrcRiseEdge (*C enumerator*), 761
drivers_RT_RT1020_MIMXRT1024._qtmr_counting_mode.kQTMR_PriSrcRiseEdgeSecDir (*C enumerator*), 762
drivers_RT_RT1020_MIMXRT1024._qtmr_counting_mode.kQTMR_PriSrcRiseEdgeSecInpHigh (*C enumerator*), 762
drivers_RT_RT1020_MIMXRT1024._qtmr_counting_mode.kQTMR_QuadCountMode (*C enumerator*), 762
drivers_RT_RT1020_MIMXRT1024._qtmr_counting_mode.kQTMR_SecSrcTrigPriCnt (*C enumerator*), 762
drivers_RT_RT1020_MIMXRT1024._qtmr_debug_action (*C enum*), 763
drivers_RT_RT1020_MIMXRT1024._qtmr_debug_action.kQTMR_ForceOutToZero (*C enumerator*), 764
drivers_RT_RT1020_MIMXRT1024._qtmr_debug_action.kQTMR_HaltCounter (*C enumerator*), 763

drivers_RT_RT1020_MIMXRT1024._qtmr_debug_action.kQTMR_HaltCountForceOutZero (*C enumerator*), 764
drivers_RT_RT1020_MIMXRT1024._qtmr_debug_action.kQTMR_RunNormalInDebug (*C enumerator*), 763
drivers_RT_RT1020_MIMXRT1024._qtmr_dma_enable (*C enum*), 765
drivers_RT_RT1020_MIMXRT1024._qtmr_dma_enable.kQTMR_ComparatorPreload1DmaEnable (*C enumerator*), 765
drivers_RT_RT1020_MIMXRT1024._qtmr_dma_enable.kQTMR_ComparatorPreload2DmaEnable (*C enumerator*), 765
drivers_RT_RT1020_MIMXRT1024._qtmr_dma_enable.kQTMR_InputEdgeFlagDmaEnable (*C enumerator*), 765
drivers_RT_RT1020_MIMXRT1024._qtmr_input_capture_edge (*C enum*), 763
drivers_RT_RT1020_MIMXRT1024._qtmr_input_capture_edge.kQTMR_FallingEdge (*C enumerator*), 763
drivers_RT_RT1020_MIMXRT1024._qtmr_input_capture_edge.kQTMR_NoCapture (*C enumerator*), 763
drivers_RT_RT1020_MIMXRT1024._qtmr_input_capture_edge.kQTMR_RisingAndFallingEdge (*C enumerator*), 763
drivers_RT_RT1020_MIMXRT1024._qtmr_input_capture_edge.kQTMR_RisingEdge (*C enumerator*), 763
drivers_RT_RT1020_MIMXRT1024._qtmr_input_source (*C enum*), 761
drivers_RT_RT1020_MIMXRT1024._qtmr_input_source.kQTMR_Counter0InputPin (*C enumerator*), 761
drivers_RT_RT1020_MIMXRT1024._qtmr_input_source.kQTMR_Counter1InputPin (*C enumerator*), 761
drivers_RT_RT1020_MIMXRT1024._qtmr_input_source.kQTMR_Counter2InputPin (*C enumerator*), 761
drivers_RT_RT1020_MIMXRT1024._qtmr_input_source.kQTMR_Counter3InputPin (*C enumerator*), 761
drivers_RT_RT1020_MIMXRT1024._qtmr_interrupt_enable (*C enum*), 764
drivers_RT_RT1020_MIMXRT1024._qtmr_interrupt_enable.kQTMR_Compare1InterruptEnable (*C enumerator*), 764
drivers_RT_RT1020_MIMXRT1024._qtmr_interrupt_enable.kQTMR_Compare2InterruptEnable (*C enumerator*), 764
drivers_RT_RT1020_MIMXRT1024._qtmr_interrupt_enable.kQTMR_CompareInterruptEnable (*C enumerator*), 764
drivers_RT_RT1020_MIMXRT1024._qtmr_interrupt_enable.kQTMR_EdgeInterruptEnable (*C enumerator*), 764
drivers_RT_RT1020_MIMXRT1024._qtmr_interrupt_enable.kQTMR_OverflowInterruptEnable (*C enumerator*), 764
drivers_RT_RT1020_MIMXRT1024._qtmr_output_mode (*C enum*), 762
drivers_RT_RT1020_MIMXRT1024._qtmr_output_mode.kQTMR_AssertWhenCountActive (*C enumerator*), 762
drivers_RT_RT1020_MIMXRT1024._qtmr_output_mode.kQTMR_ClearOnCompare (*C enumerator*), 762
drivers_RT_RT1020_MIMXRT1024._qtmr_output_mode.kQTMR_EnableGateClock (*C enumerator*), 763
drivers_RT_RT1020_MIMXRT1024._qtmr_output_mode.kQTMR_SetOnCompare (*C enumerator*), 762
drivers_RT_RT1020_MIMXRT1024._qtmr_output_mode.kQTMR_SetOnCompareClearOnCountRoll (*C enumerator*), 763
drivers_RT_RT1020_MIMXRT1024._qtmr_output_mode.kQTMR_SetOnCompareClearOnSecSrcInp (*C enumerator*), 763
drivers_RT_RT1020_MIMXRT1024._qtmr_output_mode.kQTMR_ToggleOnAltCompareReg (*C enumerator*), 762
drivers_RT_RT1020_MIMXRT1024._qtmr_output_mode.kQTMR_ToggleOnCompare (*C enumerator*), 762
drivers_RT_RT1020_MIMXRT1024._qtmr_preload_control (*C enum*), 763

drivers_RT_RT1020_MIMXRT1024._qtmr_preload_control.kQTMR_LoadOnComp1 (*C enumerator*), 763
drivers_RT_RT1020_MIMXRT1024._qtmr_preload_control.kQTMR_LoadOnComp2 (*C enumerator*), 763
drivers_RT_RT1020_MIMXRT1024._qtmr_preload_control.kQTMR_NoPreload (*C enumerator*), 763
drivers_RT_RT1020_MIMXRT1024._qtmr_primary_count_source (*C enum*), 760
drivers_RT_RT1020_MIMXRT1024._qtmr_primary_count_source.kQTMR_ClockCounter0InputPin (*C enumerator*), 760
drivers_RT_RT1020_MIMXRT1024._qtmr_primary_count_source.kQTMR_ClockCounter0Output (*C enumerator*), 760
drivers_RT_RT1020_MIMXRT1024._qtmr_primary_count_source.kQTMR_ClockCounter1InputPin (*C enumerator*), 760
drivers_RT_RT1020_MIMXRT1024._qtmr_primary_count_source.kQTMR_ClockCounter1Output (*C enumerator*), 760
drivers_RT_RT1020_MIMXRT1024._qtmr_primary_count_source.kQTMR_ClockCounter2InputPin (*C enumerator*), 760
drivers_RT_RT1020_MIMXRT1024._qtmr_primary_count_source.kQTMR_ClockCounter2Output (*C enumerator*), 760
drivers_RT_RT1020_MIMXRT1024._qtmr_primary_count_source.kQTMR_ClockCounter3InputPin (*C enumerator*), 760
drivers_RT_RT1020_MIMXRT1024._qtmr_primary_count_source.kQTMR_ClockCounter3Output (*C enumerator*), 760
drivers_RT_RT1020_MIMXRT1024._qtmr_primary_count_source.kQTMR_ClockDivide_1 (*C enumerator*), 760
drivers_RT_RT1020_MIMXRT1024._qtmr_primary_count_source.kQTMR_ClockDivide_2 (*C enumerator*), 761
drivers_RT_RT1020_MIMXRT1024._qtmr_primary_count_source.kQTMR_ClockDivide_4 (*C enumerator*), 761
drivers_RT_RT1020_MIMXRT1024._qtmr_primary_count_source.kQTMR_ClockDivide_8 (*C enumerator*), 761
drivers_RT_RT1020_MIMXRT1024._qtmr_primary_count_source.kQTMR_ClockDivide_16 (*C enumerator*), 761
drivers_RT_RT1020_MIMXRT1024._qtmr_primary_count_source.kQTMR_ClockDivide_32 (*C enumerator*), 761
drivers_RT_RT1020_MIMXRT1024._qtmr_primary_count_source.kQTMR_ClockDivide_64 (*C enumerator*), 761
drivers_RT_RT1020_MIMXRT1024._qtmr_primary_count_source.kQTMR_ClockDivide_128 (*C enumerator*), 761
drivers_RT_RT1020_MIMXRT1024._qtmr_pwm_out_state (*C enum*), 762
drivers_RT_RT1020_MIMXRT1024._qtmr_pwm_out_state.kQTMR_PwmHigh (*C enumerator*), 762
drivers_RT_RT1020_MIMXRT1024._qtmr_pwm_out_state.kQTMR_PwmLow (*C enumerator*), 762
drivers_RT_RT1020_MIMXRT1024._qtmr_status_flags (*C enum*), 764
drivers_RT_RT1020_MIMXRT1024._qtmr_status_flags.kQTMR_Compare1Flag (*C enumerator*), 764
drivers_RT_RT1020_MIMXRT1024._qtmr_status_flags.kQTMR_Compare2Flag (*C enumerator*), 764
drivers_RT_RT1020_MIMXRT1024._qtmr_status_flags.kQTMR_CompareFlag (*C enumerator*), 764
drivers_RT_RT1020_MIMXRT1024._qtmr_status_flags.kQTMR_EdgeFlag (*C enumerator*), 764
drivers_RT_RT1020_MIMXRT1024._qtmr_status_flags.kQTMR_OverflowFlag (*C enumerator*), 764
drivers_RT_RT1020_MIMXRT1024._rtwdog_clock_prescaler (*C enum*), 773
drivers_RT_RT1020_MIMXRT1024._rtwdog_clock_prescaler.kRTWDOG_ClockPrescalerDivide1 (*C enumerator*), 773
drivers_RT_RT1020_MIMXRT1024._rtwdog_clock_prescaler.kRTWDOG_ClockPrescalerDivide256

(*C enumerator*), 773
drivers_RT_RT1020_MIMXRT1024._rtwdog_clock_source (*C enum*), 772
drivers_RT_RT1020_MIMXRT1024._rtwdog_clock_source.kRTWDOG_ClockSource0 (*C enumerator*), 772
drivers_RT_RT1020_MIMXRT1024._rtwdog_clock_source.kRTWDOG_ClockSource1 (*C enumerator*), 772
drivers_RT_RT1020_MIMXRT1024._rtwdog_clock_source.kRTWDOG_ClockSource2 (*C enumerator*), 773
drivers_RT_RT1020_MIMXRT1024._rtwdog_clock_source.kRTWDOG_ClockSource3 (*C enumerator*), 773
drivers_RT_RT1020_MIMXRT1024._rtwdog_config (*C struct*), 774
drivers_RT_RT1020_MIMXRT1024._rtwdog_config.clockSource (*C var*), 774
drivers_RT_RT1020_MIMXRT1024._rtwdog_config.enableInterrupt (*C var*), 775
drivers_RT_RT1020_MIMXRT1024._rtwdog_config.enableRtwdog (*C var*), 774
drivers_RT_RT1020_MIMXRT1024._rtwdog_config.enableUpdate (*C var*), 775
drivers_RT_RT1020_MIMXRT1024._rtwdog_config.enableWindowMode (*C var*), 775
drivers_RT_RT1020_MIMXRT1024._rtwdog_config.prescaler (*C var*), 774
drivers_RT_RT1020_MIMXRT1024._rtwdog_config.testMode (*C var*), 775
drivers_RT_RT1020_MIMXRT1024._rtwdog_config.timeoutValue (*C var*), 775
drivers_RT_RT1020_MIMXRT1024._rtwdog_config.windowValue (*C var*), 775
drivers_RT_RT1020_MIMXRT1024._rtwdog_config.workMode (*C var*), 775
drivers_RT_RT1020_MIMXRT1024._rtwdog_interrupt_enable_t (*C enum*), 773
drivers_RT_RT1020_MIMXRT1024._rtwdog_interrupt_enable_t.kRTWDOG_InterruptEnable (*C enumerator*), 773
drivers_RT_RT1020_MIMXRT1024._rtwdog_status_flags_t (*C enum*), 773
drivers_RT_RT1020_MIMXRT1024._rtwdog_status_flags_t.kRTWDOG_InterruptFlag (*C enumerator*), 774
drivers_RT_RT1020_MIMXRT1024._rtwdog_status_flags_t.kRTWDOG_RunningFlag (*C enumerator*), 773
drivers_RT_RT1020_MIMXRT1024._rtwdog_test_mode (*C enum*), 773
drivers_RT_RT1020_MIMXRT1024._rtwdog_test_mode.kRTWDOG_HighByteTest (*C enumerator*), 773
drivers_RT_RT1020_MIMXRT1024._rtwdog_test_mode.kRTWDOG_LowByteTest (*C enumerator*), 773
drivers_RT_RT1020_MIMXRT1024._rtwdog_test_mode.kRTWDOG_TestModeDisabled (*C enumerator*), 773
drivers_RT_RT1020_MIMXRT1024._rtwdog_test_mode.kRTWDOG_UserModeEnabled (*C enumerator*), 773
drivers_RT_RT1020_MIMXRT1024._rtwdog_work_mode (*C struct*), 774
drivers_RT_RT1020_MIMXRT1024._rtwdog_work_mode.enableDebug (*C var*), 774
drivers_RT_RT1020_MIMXRT1024._rtwdog_work_mode.enableStop (*C var*), 774
drivers_RT_RT1020_MIMXRT1024._rtwdog_work_mode.enableWait (*C var*), 774
drivers_RT_RT1020_MIMXRT1024._sai_bclk_source (*C enum*), 793
drivers_RT_RT1020_MIMXRT1024._sai_bclk_source.kSAI_BclkSourceBusclk (*C enumerator*), 793
drivers_RT_RT1020_MIMXRT1024._sai_bclk_source.kSAI_BclkSourceMclkDiv (*C enumerator*), 793
drivers_RT_RT1020_MIMXRT1024._sai_bclk_source.kSAI_BclkSourceMclkOption1 (*C enumerator*), 793
drivers_RT_RT1020_MIMXRT1024._sai_bclk_source.kSAI_BclkSourceMclkOption2 (*C enumerator*), 793
drivers_RT_RT1020_MIMXRT1024._sai_bclk_source.kSAI_BclkSourceMclkOption3 (*C enumerator*), 793
drivers_RT_RT1020_MIMXRT1024._sai_bclk_source.kSAI_BclkSourceOtherSai0 (*C enumerator*), 793
drivers_RT_RT1020_MIMXRT1024._sai_bclk_source.kSAI_BclkSourceOtherSai1 (*C enumerator*), 793
drivers_RT_RT1020_MIMXRT1024._sai_bit_clock (*C struct*), 801
drivers_RT_RT1020_MIMXRT1024._sai_bit_clock.bclkInputDelay (*C var*), 801

drivers_RT_RT1020_MIMXRT1024._sai_bit_clock.bclkPolarity (*C var*), 801
drivers_RT_RT1020_MIMXRT1024._sai_bit_clock.bclkSource (*C var*), 801
drivers_RT_RT1020_MIMXRT1024._sai_bit_clock.bclkSrcSwap (*C var*), 801
drivers_RT_RT1020_MIMXRT1024._sai_clock_polarity (*C enum*), 792
drivers_RT_RT1020_MIMXRT1024._sai_clock_polarity.kSAI_PolarityActiveHigh (*C enumerator*), 792
drivers_RT_RT1020_MIMXRT1024._sai_clock_polarity.kSAI_PolarityActiveLow (*C enumerator*), 792
drivers_RT_RT1020_MIMXRT1024._sai_clock_polarity.kSAI_SampleOnFallingEdge (*C enumerator*), 792
drivers_RT_RT1020_MIMXRT1024._sai_clock_polarity.kSAI_SampleOnRisingEdge (*C enumerator*), 792
drivers_RT_RT1020_MIMXRT1024._sai_config (*C struct*), 799
drivers_RT_RT1020_MIMXRT1024._sai_config.bclkSource (*C var*), 799
drivers_RT_RT1020_MIMXRT1024._sai_config.masterSlave (*C var*), 799
drivers_RT_RT1020_MIMXRT1024._sai_config.mclkOutputEnable (*C var*), 799
drivers_RT_RT1020_MIMXRT1024._sai_config.protocol (*C var*), 799
drivers_RT_RT1020_MIMXRT1024._sai_config.syncMode (*C var*), 799
drivers_RT_RT1020_MIMXRT1024._sai_data_order (*C enum*), 792
drivers_RT_RT1020_MIMXRT1024._sai_data_order.kSAI_DataLSB (*C enumerator*), 792
drivers_RT_RT1020_MIMXRT1024._sai_data_order.kSAI_DataMSB (*C enumerator*), 792
drivers_RT_RT1020_MIMXRT1024._sai_data_pin_state (*C enum*), 796
drivers_RT_RT1020_MIMXRT1024._sai_data_pin_state.kSAI_DataPinStateOutputZero (*C enumerator*), 796
drivers_RT_RT1020_MIMXRT1024._sai_data_pin_state.kSAI_DataPinStateTriState (*C enumerator*), 796
drivers_RT_RT1020_MIMXRT1024._sai_edma_interleave (*C enum*), 810
drivers_RT_RT1020_MIMXRT1024._sai_edma_interleave.kSAI_EDMAInterleavePerChannelBlock (*C enumerator*), 810
drivers_RT_RT1020_MIMXRT1024._sai_edma_interleave.kSAI_EDMAInterleavePerChannelSample (*C enumerator*), 810
drivers_RT_RT1020_MIMXRT1024._sai_fifo (*C struct*), 801
drivers_RT_RT1020_MIMXRT1024._sai_fifo_combine (*C enum*), 796
drivers_RT_RT1020_MIMXRT1024._sai_fifo_combine.kSAI_FifoCombineDisabled (*C enumerator*), 796
drivers_RT_RT1020_MIMXRT1024._sai_fifo_combine.kSAI_FifoCombineModeEnabledOnRead (*C enumerator*), 796
drivers_RT_RT1020_MIMXRT1024._sai_fifo_combine.kSAI_FifoCombineModeEnabledOnReadWrite (*C enumerator*), 797
drivers_RT_RT1020_MIMXRT1024._sai_fifo_combine.kSAI_FifoCombineModeEnabledOnWrite (*C enumerator*), 796
drivers_RT_RT1020_MIMXRT1024._sai_fifo_combine.kSAI_RXFifoCombineModeEnabledOnRead (*C enumerator*), 796
drivers_RT_RT1020_MIMXRT1024._sai_fifo_combine.kSAI_RxFifoCombineModeEnabledOnWrite (*C enumerator*), 796
drivers_RT_RT1020_MIMXRT1024._sai_fifo_packing (*C enum*), 794
drivers_RT_RT1020_MIMXRT1024._sai_fifo_packing.kSAI_FifoPacking8bit (*C enumerator*), 795
drivers_RT_RT1020_MIMXRT1024._sai_fifo_packing.kSAI_FifoPacking16bit (*C enumerator*), 795
drivers_RT_RT1020_MIMXRT1024._sai_fifo_packing.kSAI_FifoPackingDisabled (*C enumerator*), 795
drivers_RT_RT1020_MIMXRT1024._sai_fifo.fifoCombine (*C var*), 801
drivers_RT_RT1020_MIMXRT1024._sai_fifo.fifoContinueOneError (*C var*), 801
drivers_RT_RT1020_MIMXRT1024._sai_fifo.fifoPacking (*C var*), 801
drivers_RT_RT1020_MIMXRT1024._sai_fifo.fifoWatermark (*C var*), 801
drivers_RT_RT1020_MIMXRT1024._sai_frame_sync (*C struct*), 801
drivers_RT_RT1020_MIMXRT1024._sai_frame_sync_len (*C enum*), 797
drivers_RT_RT1020_MIMXRT1024._sai_frame_sync_len.kSAI_FrameSyncLenOneBitClk (*C enumerator*), 797

drivers_RT_RT1020_MIMXRT1024._sai_frame_sync_len.kSAI_FrameSyncLenPerWordWidth (*C enumerator*), 797
drivers_RT_RT1020_MIMXRT1024._sai_frame_sync.frameSyncEarly (*C var*), 802
drivers_RT_RT1020_MIMXRT1024._sai_frame_sync.frameSyncGenerateOnDemand (*C var*), 802
drivers_RT_RT1020_MIMXRT1024._sai_frame_sync.frameSyncPolarity (*C var*), 802
drivers_RT_RT1020_MIMXRT1024._sai_frame_sync.frameSyncWidth (*C var*), 802
drivers_RT_RT1020_MIMXRT1024._sai_handle (*C struct*), 803
drivers_RT_RT1020_MIMXRT1024._sai_handle.base (*C var*), 804
drivers_RT_RT1020_MIMXRT1024._sai_handle.bitWidth (*C var*), 804
drivers_RT_RT1020_MIMXRT1024._sai_handle.callback (*C var*), 804
drivers_RT_RT1020_MIMXRT1024._sai_handle.channel (*C var*), 804
drivers_RT_RT1020_MIMXRT1024._sai_handle.channelMask (*C var*), 804
drivers_RT_RT1020_MIMXRT1024._sai_handle.channelNums (*C var*), 804
drivers_RT_RT1020_MIMXRT1024._sai_handle.endChannel (*C var*), 804
drivers_RT_RT1020_MIMXRT1024._sai_handle.queueDriver (*C var*), 804
drivers_RT_RT1020_MIMXRT1024._sai_handle.queueUser (*C var*), 804
drivers_RT_RT1020_MIMXRT1024._sai_handle.saiQueue (*C var*), 804
drivers_RT_RT1020_MIMXRT1024._sai_handle.state (*C var*), 804
drivers_RT_RT1020_MIMXRT1024._sai_handle.transferSize (*C var*), 804
drivers_RT_RT1020_MIMXRT1024._sai_handle.userData (*C var*), 804
drivers_RT_RT1020_MIMXRT1024._sai_handle.watermark (*C var*), 804
drivers_RT_RT1020_MIMXRT1024._sai_master_clock (*C struct*), 800
drivers_RT_RT1020_MIMXRT1024._sai_master_clock.mclkHz (*C var*), 800
drivers_RT_RT1020_MIMXRT1024._sai_master_clock.mclkOutputEnable (*C var*), 800
drivers_RT_RT1020_MIMXRT1024._sai_master_clock.mclkSourceClkHz (*C var*), 801
drivers_RT_RT1020_MIMXRT1024._sai_master_slave (*C enum*), 791
drivers_RT_RT1020_MIMXRT1024._sai_master_slave.kSAI_Bclk_Master_FrameSync_Slave (*C enumerator*), 791
drivers_RT_RT1020_MIMXRT1024._sai_master_slave.kSAI_Bclk_Slave_FrameSync_Master (*C enumerator*), 791
drivers_RT_RT1020_MIMXRT1024._sai_master_slave.kSAI_Master (*C enumerator*), 791
drivers_RT_RT1020_MIMXRT1024._sai_master_slave.kSAI_Slave (*C enumerator*), 791
drivers_RT_RT1020_MIMXRT1024._sai_mono_stereo (*C enum*), 792
drivers_RT_RT1020_MIMXRT1024._sai_mono_stereo.kSAI_MonoLeft (*C enumerator*), 792
drivers_RT_RT1020_MIMXRT1024._sai_mono_stereo.kSAI_MonoRight (*C enumerator*), 792
drivers_RT_RT1020_MIMXRT1024._sai_mono_stereo.kSAI_Stereo (*C enumerator*), 792
drivers_RT_RT1020_MIMXRT1024._sai_protocol (*C enum*), 791
drivers_RT_RT1020_MIMXRT1024._sai_protocol.kSAI_BusI2S (*C enumerator*), 791
drivers_RT_RT1020_MIMXRT1024._sai_protocol.kSAI_BusLeftJustified (*C enumerator*), 791
drivers_RT_RT1020_MIMXRT1024._sai_protocol.kSAI_BusPCMA (*C enumerator*), 791
drivers_RT_RT1020_MIMXRT1024._sai_protocol.kSAI_BusPCMB (*C enumerator*), 791
drivers_RT_RT1020_MIMXRT1024._sai_protocol.kSAI_BusRightJustified (*C enumerator*), 791
drivers_RT_RT1020_MIMXRT1024._sai_reset_type (*C enum*), 794
drivers_RT_RT1020_MIMXRT1024._sai_reset_type.kSAI_ResetAll (*C enumerator*), 794
drivers_RT_RT1020_MIMXRT1024._sai_reset_type.kSAI_ResetTypeFIFO (*C enumerator*), 794
drivers_RT_RT1020_MIMXRT1024._sai_reset_type.kSAI_ResetTypeSoftware (*C enumerator*), 794
drivers_RT_RT1020_MIMXRT1024._sai_sample_rate (*C enum*), 795
drivers_RT_RT1020_MIMXRT1024._sai_sample_rate.kSAI_SampleRate8KHz (*C enumerator*), 795
drivers_RT_RT1020_MIMXRT1024._sai_sample_rate.kSAI_SampleRate12KHz (*C enumerator*), 795
drivers_RT_RT1020_MIMXRT1024._sai_sample_rate.kSAI_SampleRate16KHz (*C enumerator*), 795
drivers_RT_RT1020_MIMXRT1024._sai_sample_rate.kSAI_SampleRate24KHz (*C enumerator*), 795
drivers_RT_RT1020_MIMXRT1024._sai_sample_rate.kSAI_SampleRate32KHz (*C enumerator*), 795
drivers_RT_RT1020_MIMXRT1024._sai_sample_rate.kSAI_SampleRate48KHz (*C enumerator*), 795

drivers_RT_RT1020_MIMXRT1024._sai_sample_rate.kSAI_SampleRate96KHz (*C enumerator*), 795
drivers_RT_RT1020_MIMXRT1024._sai_sample_rate.kSAI_SampleRate192KHz (*C enumerator*), 795
drivers_RT_RT1020_MIMXRT1024._sai_sample_rate.kSAI_SampleRate384KHz (*C enumerator*), 796
drivers_RT_RT1020_MIMXRT1024._sai_sample_rate.kSAI_SampleRate11025Hz (*C enumerator*), 795
drivers_RT_RT1020_MIMXRT1024._sai_sample_rate.kSAI_SampleRate22050Hz (*C enumerator*), 795
drivers_RT_RT1020_MIMXRT1024._sai_sample_rate.kSAI_SampleRate44100Hz (*C enumerator*), 795
drivers_RT_RT1020_MIMXRT1024._sai_serial_data (*C struct*), 802
drivers_RT_RT1020_MIMXRT1024._sai_serial_data.dataFirstBitShifted (*C var*), 802
drivers_RT_RT1020_MIMXRT1024._sai_serial_data.dataMaskedWord (*C var*), 802
drivers_RT_RT1020_MIMXRT1024._sai_serial_data.dataMode (*C var*), 802
drivers_RT_RT1020_MIMXRT1024._sai_serial_data.dataOrder (*C var*), 802
drivers_RT_RT1020_MIMXRT1024._sai_serial_data.dataWord0Length (*C var*), 802
drivers_RT_RT1020_MIMXRT1024._sai_serial_data.dataWordLength (*C var*), 802
drivers_RT_RT1020_MIMXRT1024._sai_serial_data.dataWordNLength (*C var*), 802
drivers_RT_RT1020_MIMXRT1024._sai_serial_data.dataWordNum (*C var*), 802
drivers_RT_RT1020_MIMXRT1024._sai_sync_mode (*C enum*), 792
drivers_RT_RT1020_MIMXRT1024._sai_sync_mode.kSAI_ModeAsync (*C enumerator*), 792
drivers_RT_RT1020_MIMXRT1024._sai_sync_mode.kSAI_ModeSync (*C enumerator*), 792
drivers_RT_RT1020_MIMXRT1024._sai_sync_mode.kSAI_ModeSyncWithOtherRx (*C enumerator*), 793
drivers_RT_RT1020_MIMXRT1024._sai_sync_mode.kSAI_ModeSyncWithOtherTx (*C enumerator*), 793
drivers_RT_RT1020_MIMXRT1024._sai_transceiver (*C struct*), 802
drivers_RT_RT1020_MIMXRT1024._sai_transceiver_type (*C enum*), 797
drivers_RT_RT1020_MIMXRT1024._sai_transceiver_type.kSAI_Receiver (*C enumerator*), 797
drivers_RT_RT1020_MIMXRT1024._sai_transceiver_type.kSAI_Transmitter (*C enumerator*), 797
drivers_RT_RT1020_MIMXRT1024._sai_transceiver.bitClock (*C var*), 803
drivers_RT_RT1020_MIMXRT1024._sai_transceiver.channelMask (*C var*), 803
drivers_RT_RT1020_MIMXRT1024._sai_transceiver.channelNums (*C var*), 803
drivers_RT_RT1020_MIMXRT1024._sai_transceiver.endChannel (*C var*), 803
drivers_RT_RT1020_MIMXRT1024._sai_transceiver.fifo (*C var*), 803
drivers_RT_RT1020_MIMXRT1024._sai_transceiver.frameSync (*C var*), 803
drivers_RT_RT1020_MIMXRT1024._sai_transceiver.masterSlave (*C var*), 803
drivers_RT_RT1020_MIMXRT1024._sai_transceiver.serialData (*C var*), 803
drivers_RT_RT1020_MIMXRT1024._sai_transceiver.startChannel (*C var*), 803
drivers_RT_RT1020_MIMXRT1024._sai_transceiver.syncMode (*C var*), 803
drivers_RT_RT1020_MIMXRT1024._sai_transfer (*C struct*), 803
drivers_RT_RT1020_MIMXRT1024._sai_transfer_format (*C struct*), 799
drivers_RT_RT1020_MIMXRT1024._sai_transfer_format.bitWidth (*C var*), 800
drivers_RT_RT1020_MIMXRT1024._sai_transfer_format.channel (*C var*), 800
drivers_RT_RT1020_MIMXRT1024._sai_transfer_format.channelMask (*C var*), 800
drivers_RT_RT1020_MIMXRT1024._sai_transfer_format.channelNums (*C var*), 800
drivers_RT_RT1020_MIMXRT1024._sai_transfer_format.endChannel (*C var*), 800
drivers_RT_RT1020_MIMXRT1024._sai_transfer_format.isFrameSyncCompact (*C var*), 800
drivers_RT_RT1020_MIMXRT1024._sai_transfer_format.masterClockHz (*C var*), 800
drivers_RT_RT1020_MIMXRT1024._sai_transfer_format.protocol (*C var*), 800
drivers_RT_RT1020_MIMXRT1024._sai_transfer_format.sampleRate_Hz (*C var*), 800
drivers_RT_RT1020_MIMXRT1024._sai_transfer_format.stereo (*C var*), 800
drivers_RT_RT1020_MIMXRT1024._sai_transfer_format.watermark (*C var*), 800
drivers_RT_RT1020_MIMXRT1024._sai_transfer.data (*C var*), 803
drivers_RT_RT1020_MIMXRT1024._sai_transfer.dataSize (*C var*), 803
drivers_RT_RT1020_MIMXRT1024._sai_word_width (*C enum*), 796

drivers_RT_RT1020_MIMXRT1024._sai_word_width.kSAI_WordWidth8bits (*C enumerator*), 796
drivers_RT_RT1020_MIMXRT1024._sai_word_width.kSAI_WordWidth16bits (*C enumerator*), 796
drivers_RT_RT1020_MIMXRT1024._sai_word_width.kSAI_WordWidth24bits (*C enumerator*), 796
drivers_RT_RT1020_MIMXRT1024._sai_word_width.kSAI_WordWidth32bits (*C enumerator*), 796
drivers_RT_RT1020_MIMXRT1024._SDK_AtomicLocalCompareAndSet (*C function*), 624
drivers_RT_RT1020_MIMXRT1024._SDK_AtomicTestAndSet (*C function*), 624
drivers_RT_RT1020_MIMXRT1024._semc_addr_mode (*C enum*), 824
drivers_RT_RT1020_MIMXRT1024._semc_addr_mode.kSEMC_AddrDataMux (*C enumerator*), 824
drivers_RT_RT1020_MIMXRT1024._semc_addr_mode.kSEMC_AddrDataNonMux (*C enumerator*), 825
drivers_RT_RT1020_MIMXRT1024._semc_addr_mode.kSEMC_AdvAddrdataMux (*C enumerator*), 825
drivers_RT_RT1020_MIMXRT1024._semc_adv_level_control (*C enum*), 825
drivers_RT_RT1020_MIMXRT1024._semc_adv_level_control.kSEMC_AdvHigh (*C enumerator*), 825
drivers_RT_RT1020_MIMXRT1024._semc_adv_level_control.kSEMC_AdvLow (*C enumerator*), 825
drivers_RT_RT1020_MIMXRT1024._semc_adv_polarity (*C enum*), 825
drivers_RT_RT1020_MIMXRT1024._semc_adv_polarity.kSEMC_AdvActiveHigh (*C enumerator*), 825
drivers_RT_RT1020_MIMXRT1024._semc_adv_polarity.kSEMC_AdvActiveLow (*C enumerator*), 825
drivers_RT_RT1020_MIMXRT1024._semc_axi_queueweight (*C struct*), 841
drivers_RT_RT1020_MIMXRT1024._semc_axi_queueweight.queueaEnable (*C var*), 841
drivers_RT_RT1020_MIMXRT1024._semc_axi_queueweight.queueaWeight (*C var*), 841
drivers_RT_RT1020_MIMXRT1024._semc_axi_queueweight.queuebEnable (*C var*), 841
drivers_RT_RT1020_MIMXRT1024._semc_axi_queueweight.queuebWeight (*C var*), 841
drivers_RT_RT1020_MIMXRT1024._semc_caslatency (*C enum*), 819
drivers_RT_RT1020_MIMXRT1024._semc_caslatency.kSEMC_LatencyOne (*C enumerator*), 819
drivers_RT_RT1020_MIMXRT1024._semc_caslatency.kSEMC_LatencyThree (*C enumerator*), 819
drivers_RT_RT1020_MIMXRT1024._semc_caslatency.kSEMC_LatencyTwo (*C enumerator*), 819
drivers_RT_RT1020_MIMXRT1024._semc_config_t (*C struct*), 841
drivers_RT_RT1020_MIMXRT1024._semc_config_t.busTimeoutCycles (*C var*), 842
drivers_RT_RT1020_MIMXRT1024._semc_config_t.cmdTimeoutCycles (*C var*), 842
drivers_RT_RT1020_MIMXRT1024._semc_config_t.dqsMode (*C var*), 842
drivers_RT_RT1020_MIMXRT1024._semc_config_t.queueWeight (*C var*), 842
drivers_RT_RT1020_MIMXRT1024._semc_dbi_burst_len (*C enum*), 823
drivers_RT_RT1020_MIMXRT1024._semc_dbi_burst_len.kSEMC_Dbi_BurstLen1 (*C enumerator*), 823
drivers_RT_RT1020_MIMXRT1024._semc_dbi_burst_len.kSEMC_Dbi_BurstLen2 (*C enumerator*), 823
drivers_RT_RT1020_MIMXRT1024._semc_dbi_burst_len.kSEMC_Dbi_BurstLen8 (*C enumerator*), 823
drivers_RT_RT1020_MIMXRT1024._semc_dbi_burst_len.kSEMC_Dbi_BurstLen16 (*C enumerator*), 823
drivers_RT_RT1020_MIMXRT1024._semc_dbi_burst_len.kSEMC_Dbi_BurstLen32 (*C enumerator*), 823
drivers_RT_RT1020_MIMXRT1024._semc_dbi_burst_len.kSEMC_Dbi_BurstLen64 (*C enumerator*), 823
drivers_RT_RT1020_MIMXRT1024._semc_dbi_burst_len.kSEMC_Dbi_Dbi_BurstLen4 (*C enumerator*), 823
drivers_RT_RT1020_MIMXRT1024._semc_dbi_column_bit_num (*C enum*), 822
drivers_RT_RT1020_MIMXRT1024._semc_dbi_column_bit_num.kSEMC_Dbi_Colum_2bit (*C enumerator*), 823
drivers_RT_RT1020_MIMXRT1024._semc_dbi_column_bit_num.kSEMC_Dbi_Colum_3bit (*C enumerator*), 823
drivers_RT_RT1020_MIMXRT1024._semc_dbi_column_bit_num.kSEMC_Dbi_Colum_4bit (*C*

enumerator), 823
drivers_RT_RT1020_MIMXRT1024._semc_dbi_column_bit_num.kSEMC_Dbi_Colum_5bit (*C enumerator*), 823
drivers_RT_RT1020_MIMXRT1024._semc_dbi_column_bit_num.kSEMC_Dbi_Colum_6bit (*C enumerator*), 823
drivers_RT_RT1020_MIMXRT1024._semc_dbi_column_bit_num.kSEMC_Dbi_Colum_7bit (*C enumerator*), 823
drivers_RT_RT1020_MIMXRT1024._semc_dbi_column_bit_num.kSEMC_Dbi_Colum_8bit (*C enumerator*), 823
drivers_RT_RT1020_MIMXRT1024._semc_dbi_column_bit_num.kSEMC_Dbi_Colum_9bit (*C enumerator*), 822
drivers_RT_RT1020_MIMXRT1024._semc_dbi_column_bit_num.kSEMC_Dbi_Colum_10bit (*C enumerator*), 822
drivers_RT_RT1020_MIMXRT1024._semc_dbi_column_bit_num.kSEMC_Dbi_Colum_11bit (*C enumerator*), 822
drivers_RT_RT1020_MIMXRT1024._semc_dbi_column_bit_num.kSEMC_Dbi_Colum_12bit (*C enumerator*), 822
drivers_RT_RT1020_MIMXRT1024._semc_dbi_config (*C struct*), 839
drivers_RT_RT1020_MIMXRT1024._semc_dbi_config.address (*C var*), 839
drivers_RT_RT1020_MIMXRT1024._semc_dbi_config.burstLen (*C var*), 839
drivers_RT_RT1020_MIMXRT1024._semc_dbi_config.columnAddrBitNum (*C var*), 839
drivers_RT_RT1020_MIMXRT1024._semc_dbi_config.csxPinMux (*C var*), 839
drivers_RT_RT1020_MIMXRT1024._semc_dbi_config.memsize_kbytes (*C var*), 839
drivers_RT_RT1020_MIMXRT1024._semc_dbi_config.portSize (*C var*), 839
drivers_RT_RT1020_MIMXRT1024._semc_dbi_config.tCsxHold_Ns (*C var*), 839
drivers_RT_RT1020_MIMXRT1024._semc_dbi_config.tCsxInterval_Ns (*C var*), 840
drivers_RT_RT1020_MIMXRT1024._semc_dbi_config.tCsxSetup_Ns (*C var*), 839
drivers_RT_RT1020_MIMXRT1024._semc_dbi_config.tRdxHigh_Ns (*C var*), 840
drivers_RT_RT1020_MIMXRT1024._semc_dbi_config.tRdxLow_Ns (*C var*), 840
drivers_RT_RT1020_MIMXRT1024._semc_dbi_config.tWexHigh_Ns (*C var*), 840
drivers_RT_RT1020_MIMXRT1024._semc_dbi_config.tWexLow_Ns (*C var*), 839
drivers_RT_RT1020_MIMXRT1024._semc_dqs_mode (*C enum*), 825
drivers_RT_RT1020_MIMXRT1024._semc_dqs_mode.kSEMC_Loopbackdqspad (*C enumerator*), 825
drivers_RT_RT1020_MIMXRT1024._semc_dqs_mode.kSEMC_Loopbackinternal (*C enumerator*), 825
drivers_RT_RT1020_MIMXRT1024._semc_interrupt_enable (*C enum*), 818
drivers_RT_RT1020_MIMXRT1024._semc_interrupt_enable.kSEMC_AXIBusErrInterrupt (*C enumerator*), 818
drivers_RT_RT1020_MIMXRT1024._semc_interrupt_enable.kSEMC_AXICmdErrInterrupt (*C enumerator*), 818
drivers_RT_RT1020_MIMXRT1024._semc_interrupt_enable.kSEMC_IPCmdDoneInterrupt (*C enumerator*), 818
drivers_RT_RT1020_MIMXRT1024._semc_interrupt_enable.kSEMC_IPCmdErrInterrupt (*C enumerator*), 818
drivers_RT_RT1020_MIMXRT1024._semc_iomux_nora27_pin (*C enum*), 824
drivers_RT_RT1020_MIMXRT1024._semc_iomux_nora27_pin.kSEMC_MORA27_NONE (*C enumerator*), 824
drivers_RT_RT1020_MIMXRT1024._semc_iomux_nora27_pin.kSEMC_NORA27_MUXCSX3 (*C enumerator*), 824
drivers_RT_RT1020_MIMXRT1024._semc_iomux_nora27_pin.kSEMC_NORA27_MUXRDY (*C enumerator*), 824
drivers_RT_RT1020_MIMXRT1024._semc_iomux_pin (*C enum*), 823
drivers_RT_RT1020_MIMXRT1024._semc_iomux_pin.kSEMC_MUXA8 (*C enumerator*), 824
drivers_RT_RT1020_MIMXRT1024._semc_iomux_pin.kSEMC_MUXCSX0 (*C enumerator*), 824
drivers_RT_RT1020_MIMXRT1024._semc_iomux_pin.kSEMC_MUXCSX1 (*C enumerator*), 824
drivers_RT_RT1020_MIMXRT1024._semc_iomux_pin.kSEMC_MUXCSX2 (*C enumerator*), 824
drivers_RT_RT1020_MIMXRT1024._semc_iomux_pin.kSEMC_MUXCSX3 (*C enumerator*), 824

drivers_RT_RT1020_MIMXRT1024._semc_iomux_pin.kSEMC_MUXRDY (*C enumerator*), 824
drivers_RT_RT1020_MIMXRT1024._semc_ipcmd_datasize (*C enum*), 818
drivers_RT_RT1020_MIMXRT1024._semc_ipcmd_datasize.kSEMC_IPcmdDataSize_1bytes (*C enumerator*), 818
drivers_RT_RT1020_MIMXRT1024._semc_ipcmd_datasize.kSEMC_IPcmdDataSize_2bytes (*C enumerator*), 818
drivers_RT_RT1020_MIMXRT1024._semc_ipcmd_datasize.kSEMC_IPcmdDataSize_3bytes (*C enumerator*), 819
drivers_RT_RT1020_MIMXRT1024._semc_ipcmd_datasize.kSEMC_IPcmdDataSize_4bytes (*C enumerator*), 819
drivers_RT_RT1020_MIMXRT1024._semc_ipcmd_nand_addrmode (*C enum*), 826
drivers_RT_RT1020_MIMXRT1024._semc_ipcmd_nand_addrmode.kSEMC_NANDAM_ColumnCA0 (*C enumerator*), 826
drivers_RT_RT1020_MIMXRT1024._semc_ipcmd_nand_addrmode.kSEMC_NANDAM_ColumnCA0CA1 (*C enumerator*), 826
drivers_RT_RT1020_MIMXRT1024._semc_ipcmd_nand_addrmode.kSEMC_NANDAM_ColumnRow (*C enumerator*), 826
drivers_RT_RT1020_MIMXRT1024._semc_ipcmd_nand_addrmode.kSEMC_NANDAM_RawRA0 (*C enumerator*), 826
drivers_RT_RT1020_MIMXRT1024._semc_ipcmd_nand_addrmode.kSEMC_NANDAM_RawRA0RA1 (*C enumerator*), 826
drivers_RT_RT1020_MIMXRT1024._semc_ipcmd_nand_addrmode.kSEMC_NANDAM_RawRA0RA1RA2 (*C enumerator*), 826
drivers_RT_RT1020_MIMXRT1024._semc_ipcmd_nand_cmdmode (*C enum*), 826
drivers_RT_RT1020_MIMXRT1024._semc_ipcmd_nand_cmdmode.kSEMC_NANDCM_Command (*C enumerator*), 826
drivers_RT_RT1020_MIMXRT1024._semc_ipcmd_nand_cmdmode.kSEMC_NANDCM_CommandAddress (*C enumerator*), 826
drivers_RT_RT1020_MIMXRT1024._semc_ipcmd_nand_cmdmode.kSEMC_NANDCM_CommandAddressHold (*C enumerator*), 826
drivers_RT_RT1020_MIMXRT1024._semc_ipcmd_nand_cmdmode.kSEMC_NANDCM_CommandAddressRead (*C enumerator*), 826
drivers_RT_RT1020_MIMXRT1024._semc_ipcmd_nand_cmdmode.kSEMC_NANDCM_CommandAddressWrite (*C enumerator*), 827
drivers_RT_RT1020_MIMXRT1024._semc_ipcmd_nand_cmdmode.kSEMC_NANDCM_CommandHold (*C enumerator*), 826
drivers_RT_RT1020_MIMXRT1024._semc_ipcmd_nand_cmdmode.kSEMC_NANDCM_CommandRead (*C enumerator*), 827
drivers_RT_RT1020_MIMXRT1024._semc_ipcmd_nand_cmdmode.kSEMC_NANDCM_CommandWrite (*C enumerator*), 827
drivers_RT_RT1020_MIMXRT1024._semc_ipcmd_nand_cmdmode.kSEMC_NANDCM_Read (*C enumerator*), 827
drivers_RT_RT1020_MIMXRT1024._semc_ipcmd_nand_cmdmode.kSEMC_NANDCM_Write (*C enumerator*), 827
drivers_RT_RT1020_MIMXRT1024._semc_ipcmd_nor_dbi (*C enum*), 827
drivers_RT_RT1020_MIMXRT1024._semc_ipcmd_nor_dbi.kSEMC_NORDBICM_Read (*C enumerator*), 827
drivers_RT_RT1020_MIMXRT1024._semc_ipcmd_nor_dbi.kSEMC_NORDBICM_Write (*C enumerator*), 827
drivers_RT_RT1020_MIMXRT1024._semc_ipcmd_sdram (*C enum*), 828
drivers_RT_RT1020_MIMXRT1024._semc_ipcmd_sdram.kSEMC_SDRAMCM_Active (*C enumerator*), 828
drivers_RT_RT1020_MIMXRT1024._semc_ipcmd_sdram.kSEMC_SDRAMCM_AutoRefresh (*C enumerator*), 828
drivers_RT_RT1020_MIMXRT1024._semc_ipcmd_sdram.kSEMC_SDRAMCM_Modeset (*C enumerator*), 828
drivers_RT_RT1020_MIMXRT1024._semc_ipcmd_sdram.kSEMC_SDRAMCM_Precharge (*C enumerator*), 828

drivers_RT_RT1020_MIMXRT1024._semc_ipcmd_sdram.kSEMC_SDRAMCM_Prechargeall (*C enumerator*), 828
drivers_RT_RT1020_MIMXRT1024._semc_ipcmd_sdram.kSEMC_SDRAMCM_Read (*C enumerator*), 828
drivers_RT_RT1020_MIMXRT1024._semc_ipcmd_sdram.kSEMC_SDRAMCM_SelfRefresh (*C enumerator*), 828
drivers_RT_RT1020_MIMXRT1024._semc_ipcmd_sdram.kSEMC_SDRAMCM_Write (*C enumerator*), 828
drivers_RT_RT1020_MIMXRT1024._semc_ipcmd_sram (*C enum*), 828
drivers_RT_RT1020_MIMXRT1024._semc_ipcmd_sram.kSEMC_SRAMCM_ArrayRead (*C enumerator*), 828
drivers_RT_RT1020_MIMXRT1024._semc_ipcmd_sram.kSEMC_SRAMCM_ArrayWrite (*C enumerator*), 828
drivers_RT_RT1020_MIMXRT1024._semc_ipcmd_sram.kSEMC_SRAMCM_RegRead (*C enumerator*), 828
drivers_RT_RT1020_MIMXRT1024._semc_ipcmd_sram.kSEMC_SRAMCM_RegWrite (*C enumerator*), 828
drivers_RT_RT1020_MIMXRT1024._semc_mem_type (*C enum*), 817
drivers_RT_RT1020_MIMXRT1024._semc_mem_type.kSEMC_MemType_8080 (*C enumerator*), 817
drivers_RT_RT1020_MIMXRT1024._semc_mem_type.kSEMC_MemType_NAND (*C enumerator*), 817
drivers_RT_RT1020_MIMXRT1024._semc_mem_type.kSEMC_MemType_NOR (*C enumerator*), 817
drivers_RT_RT1020_MIMXRT1024._semc_mem_type.kSEMC_MemType_SDRAM (*C enumerator*), 817
drivers_RT_RT1020_MIMXRT1024._semc_mem_type.kSEMC_MemType_SRAM (*C enumerator*), 817
drivers_RT_RT1020_MIMXRT1024._semc_nand_access_type (*C enum*), 818
drivers_RT_RT1020_MIMXRT1024._semc_nand_access_type.kSEMC_NAND_ACCESS_BY_AXI (*C enumerator*), 818
drivers_RT_RT1020_MIMXRT1024._semc_nand_access_type.kSEMC_NAND_ACCESS_BY_IPCMD (*C enumerator*), 818
drivers_RT_RT1020_MIMXRT1024._semc_nand_address_option (*C enum*), 827
drivers_RT_RT1020_MIMXRT1024._semc_nand_address_option.kSEMC_NandAddrOption_2byte_CA1RA1 (*C enumerator*), 827
drivers_RT_RT1020_MIMXRT1024._semc_nand_address_option.kSEMC_NandAddrOption_3byte_CA1RA2 (*C enumerator*), 827
drivers_RT_RT1020_MIMXRT1024._semc_nand_address_option.kSEMC_NandAddrOption_3byte_CA2RA1 (*C enumerator*), 827
drivers_RT_RT1020_MIMXRT1024._semc_nand_address_option.kSEMC_NandAddrOption_4byte_CA1RA3 (*C enumerator*), 827
drivers_RT_RT1020_MIMXRT1024._semc_nand_address_option.kSEMC_NandAddrOption_4byte_CA2RA2 (*C enumerator*), 827
drivers_RT_RT1020_MIMXRT1024._semc_nand_address_option.kSEMC_NandAddrOption_5byte_CA2RA3 (*C enumerator*), 827
drivers_RT_RT1020_MIMXRT1024._semc_nand_burst_len (*C enum*), 820
drivers_RT_RT1020_MIMXRT1024._semc_nand_burst_len.kSEMC_Nand_BurstLen1 (*C enumerator*), 820
drivers_RT_RT1020_MIMXRT1024._semc_nand_burst_len.kSEMC_Nand_BurstLen2 (*C enumerator*), 821
drivers_RT_RT1020_MIMXRT1024._semc_nand_burst_len.kSEMC_Nand_BurstLen4 (*C enumerator*), 821
drivers_RT_RT1020_MIMXRT1024._semc_nand_burst_len.kSEMC_Nand_BurstLen8 (*C enumerator*), 821
drivers_RT_RT1020_MIMXRT1024._semc_nand_burst_len.kSEMC_Nand_BurstLen16 (*C enumerator*), 821
drivers_RT_RT1020_MIMXRT1024._semc_nand_burst_len.kSEMC_Nand_BurstLen32 (*C enumer-*

ator), 821
drivers_RT_RT1020_MIMXRT1024._semc_nand_burst_len.kSEMC_Nand_BurstLen64 (*C enumerator*), 821
drivers_RT_RT1020_MIMXRT1024._semc_nand_column_bit_num (*C enum*), 820
drivers_RT_RT1020_MIMXRT1024._semc_nand_column_bit_num.kSEMC_NandColum_9bit (*C enumerator*), 820
drivers_RT_RT1020_MIMXRT1024._semc_nand_column_bit_num.kSEMC_NandColum_10bit (*C enumerator*), 820
drivers_RT_RT1020_MIMXRT1024._semc_nand_column_bit_num.kSEMC_NandColum_11bit (*C enumerator*), 820
drivers_RT_RT1020_MIMXRT1024._semc_nand_column_bit_num.kSEMC_NandColum_12bit (*C enumerator*), 820
drivers_RT_RT1020_MIMXRT1024._semc_nand_column_bit_num.kSEMC_NandColum_13bit (*C enumerator*), 820
drivers_RT_RT1020_MIMXRT1024._semc_nand_column_bit_num.kSEMC_NandColum_14bit (*C enumerator*), 820
drivers_RT_RT1020_MIMXRT1024._semc_nand_column_bit_num.kSEMC_NandColum_15bit (*C enumerator*), 820
drivers_RT_RT1020_MIMXRT1024._semc_nand_column_bit_num.kSEMC_NandColum_16bit (*C enumerator*), 820
drivers_RT_RT1020_MIMXRT1024._semc_nand_config (*C struct*), 834
drivers_RT_RT1020_MIMXRT1024._semc_nand_config.arrayAddrOption (*C var*), 835
drivers_RT_RT1020_MIMXRT1024._semc_nand_config.axiAddress (*C var*), 834
drivers_RT_RT1020_MIMXRT1024._semc_nand_config.axiMemsize_kbytes (*C var*), 834
drivers_RT_RT1020_MIMXRT1024._semc_nand_config.burstLen (*C var*), 835
drivers_RT_RT1020_MIMXRT1024._semc_nand_config.cePinMux (*C var*), 834
drivers_RT_RT1020_MIMXRT1024._semc_nand_config.columnAddrBitNum (*C var*), 835
drivers_RT_RT1020_MIMXRT1024._semc_nand_config.edoModeEnabled (*C var*), 835
drivers_RT_RT1020_MIMXRT1024._semc_nand_config.ipgAddress (*C var*), 834
drivers_RT_RT1020_MIMXRT1024._semc_nand_config.ipgMemsize_kbytes (*C var*), 835
drivers_RT_RT1020_MIMXRT1024._semc_nand_config.portSize (*C var*), 835
drivers_RT_RT1020_MIMXRT1024._semc_nand_config.rdyactivePolarity (*C var*), 835
drivers_RT_RT1020_MIMXRT1024._semc_nand_config.timingConfig (*C var*), 835
drivers_RT_RT1020_MIMXRT1024._semc_nand_timing_config (*C struct*), 833
drivers_RT_RT1020_MIMXRT1024._semc_nand_timing_config.tAle2WriteStart_Ns (*C var*), 834
drivers_RT_RT1020_MIMXRT1024._semc_nand_timing_config.tCeHold_Ns (*C var*), 833
drivers_RT_RT1020_MIMXRT1024._semc_nand_timing_config.tCeInterval_Ns (*C var*), 833
drivers_RT_RT1020_MIMXRT1024._semc_nand_timing_config.tCeSetup_Ns (*C var*), 833
drivers_RT_RT1020_MIMXRT1024._semc_nand_timing_config.tReady2Relow_Ns (*C var*), 834
drivers_RT_RT1020_MIMXRT1024._semc_nand_timing_config.tRehigh2Welow_Ns (*C var*), 834
drivers_RT_RT1020_MIMXRT1024._semc_nand_timing_config.tReHigh_Ns (*C var*), 834
drivers_RT_RT1020_MIMXRT1024._semc_nand_timing_config.tReLow_Ns (*C var*), 834
drivers_RT_RT1020_MIMXRT1024._semc_nand_timing_config.tTurnAround_Ns (*C var*), 834
drivers_RT_RT1020_MIMXRT1024._semc_nand_timing_config.tWehigh2Busy_Ns (*C var*), 834
drivers_RT_RT1020_MIMXRT1024._semc_nand_timing_config.tWehigh2Relow_Ns (*C var*), 834
drivers_RT_RT1020_MIMXRT1024._semc_nand_timing_config.tWeHigh_Ns (*C var*), 834
drivers_RT_RT1020_MIMXRT1024._semc_nand_timing_config.tWeLow_Ns (*C var*), 834
drivers_RT_RT1020_MIMXRT1024._semc_nor_config (*C struct*), 835
drivers_RT_RT1020_MIMXRT1024._semc_nor_config.addr27 (*C var*), 835
drivers_RT_RT1020_MIMXRT1024._semc_nor_config.address (*C var*), 835
drivers_RT_RT1020_MIMXRT1024._semc_nor_config.addrMode (*C var*), 836
drivers_RT_RT1020_MIMXRT1024._semc_nor_config.addrPortWidth (*C var*), 835
drivers_RT_RT1020_MIMXRT1024._semc_nor_config.advActivePolarity (*C var*), 836
drivers_RT_RT1020_MIMXRT1024._semc_nor_config.burstLen (*C var*), 836
drivers_RT_RT1020_MIMXRT1024._semc_nor_config.cePinMux (*C var*), 835
drivers_RT_RT1020_MIMXRT1024._semc_nor_config.columnAddrBitNum (*C var*), 836
drivers_RT_RT1020_MIMXRT1024._semc_nor_config.delayChain (*C var*), 837
drivers_RT_RT1020_MIMXRT1024._semc_nor_config.latencyCount (*C var*), 837

drivers_RT_RT1020_MIMXRT1024._semc_nor_config.memsize_kbytes (*C var*), 835
drivers_RT_RT1020_MIMXRT1024._semc_nor_config.portSize (*C var*), 836
drivers_RT_RT1020_MIMXRT1024._semc_nor_config.rdyactivePolarity (*C var*), 835
drivers_RT_RT1020_MIMXRT1024._semc_nor_config.readCycle (*C var*), 837
drivers_RT_RT1020_MIMXRT1024._semc_nor_config.tAddr2WriteHold_Ns (*C var*), 836
drivers_RT_RT1020_MIMXRT1024._semc_nor_config.tAddrHold_Ns (*C var*), 836
drivers_RT_RT1020_MIMXRT1024._semc_nor_config.tAddrSetup_Ns (*C var*), 836
drivers_RT_RT1020_MIMXRT1024._semc_nor_config.tCeHold_Ns (*C var*), 836
drivers_RT_RT1020_MIMXRT1024._semc_nor_config.tCeInterval_Ns (*C var*), 836
drivers_RT_RT1020_MIMXRT1024._semc_nor_config.tCeSetup_Ns (*C var*), 836
drivers_RT_RT1020_MIMXRT1024._semc_nor_config.tReHigh_Ns (*C var*), 836
drivers_RT_RT1020_MIMXRT1024._semc_nor_config.tReLow_Ns (*C var*), 836
drivers_RT_RT1020_MIMXRT1024._semc_nor_config.tTurnAround_Ns (*C var*), 836
drivers_RT_RT1020_MIMXRT1024._semc_nor_config.tWeHigh_Ns (*C var*), 836
drivers_RT_RT1020_MIMXRT1024._semc_nor_config.tWeLow_Ns (*C var*), 836
drivers_RT_RT1020_MIMXRT1024._semc_nor_config.tWriteHold_Ns (*C var*), 837
drivers_RT_RT1020_MIMXRT1024._semc_nor_config.tWriteSetup_Ns (*C var*), 837
drivers_RT_RT1020_MIMXRT1024._semc_norsram_burst_len (*C enum*), 822
drivers_RT_RT1020_MIMXRT1024._semc_norsram_burst_len.kSEMC_Nor_BurstLen1 (*C enumerator*), 822
drivers_RT_RT1020_MIMXRT1024._semc_norsram_burst_len.kSEMC_Nor_BurstLen2 (*C enumerator*), 822
drivers_RT_RT1020_MIMXRT1024._semc_norsram_burst_len.kSEMC_Nor_BurstLen4 (*C enumerator*), 822
drivers_RT_RT1020_MIMXRT1024._semc_norsram_burst_len.kSEMC_Nor_BurstLen8 (*C enumerator*), 822
drivers_RT_RT1020_MIMXRT1024._semc_norsram_burst_len.kSEMC_Nor_BurstLen16 (*C enumerator*), 822
drivers_RT_RT1020_MIMXRT1024._semc_norsram_burst_len.kSEMC_Nor_BurstLen32 (*C enumerator*), 822
drivers_RT_RT1020_MIMXRT1024._semc_norsram_burst_len.kSEMC_Nor_BurstLen64 (*C enumerator*), 822
drivers_RT_RT1020_MIMXRT1024._semc_norsram_column_bit_num (*C enum*), 821
drivers_RT_RT1020_MIMXRT1024._semc_norsram_column_bit_num.kSEMC_NorColum_2bit (*C enumerator*), 822
drivers_RT_RT1020_MIMXRT1024._semc_norsram_column_bit_num.kSEMC_NorColum_3bit (*C enumerator*), 822
drivers_RT_RT1020_MIMXRT1024._semc_norsram_column_bit_num.kSEMC_NorColum_4bit (*C enumerator*), 821
drivers_RT_RT1020_MIMXRT1024._semc_norsram_column_bit_num.kSEMC_NorColum_5bit (*C enumerator*), 821
drivers_RT_RT1020_MIMXRT1024._semc_norsram_column_bit_num.kSEMC_NorColum_6bit (*C enumerator*), 821
drivers_RT_RT1020_MIMXRT1024._semc_norsram_column_bit_num.kSEMC_NorColum_7bit (*C enumerator*), 821
drivers_RT_RT1020_MIMXRT1024._semc_norsram_column_bit_num.kSEMC_NorColum_8bit (*C enumerator*), 821
drivers_RT_RT1020_MIMXRT1024._semc_norsram_column_bit_num.kSEMC_NorColum_9bit (*C enumerator*), 821
drivers_RT_RT1020_MIMXRT1024._semc_norsram_column_bit_num.kSEMC_NorColum_10bit (*C enumerator*), 821
drivers_RT_RT1020_MIMXRT1024._semc_norsram_column_bit_num.kSEMC_NorColum_11bit (*C enumerator*), 821
drivers_RT_RT1020_MIMXRT1024._semc_norsram_column_bit_num.kSEMC_NorColum_12bit (*C enumerator*), 821
drivers_RT_RT1020_MIMXRT1024._semc_port_size (*C enum*), 824
drivers_RT_RT1020_MIMXRT1024._semc_port_size.kSEMC_PortSize8Bit (*C enumerator*), 824
drivers_RT_RT1020_MIMXRT1024._semc_port_size.kSEMC_PortSize16Bit (*C enumerator*), 824

drivers_RT_RT1020_MIMXRT1024._semc_queuea_weight (*C union*), 840
drivers_RT_RT1020_MIMXRT1024._semc_queuea_weight_struct (*C struct*), 840
drivers_RT_RT1020_MIMXRT1024._semc_queuea_weight_struct.aging (*C var*), 840
drivers_RT_RT1020_MIMXRT1024._semc_queuea_weight_struct.qos (*C var*), 840
drivers_RT_RT1020_MIMXRT1024._semc_queuea_weight_struct.slaveHitNoswitch (*C var*), 840
drivers_RT_RT1020_MIMXRT1024._semc_queuea_weight_struct.slaveHitSwitch (*C var*), 840
drivers_RT_RT1020_MIMXRT1024._semc_queuea_weight.queueaConfig (*C var*), 840
drivers_RT_RT1020_MIMXRT1024._semc_queuea_weight.queueaValue (*C var*), 840
drivers_RT_RT1020_MIMXRT1024._semc_queueb_weight (*C union*), 841
drivers_RT_RT1020_MIMXRT1024._semc_queueb_weight_struct (*C struct*), 840
drivers_RT_RT1020_MIMXRT1024._semc_queueb_weight_struct.aging (*C var*), 841
drivers_RT_RT1020_MIMXRT1024._semc_queueb_weight_struct.bankRotation (*C var*), 841
drivers_RT_RT1020_MIMXRT1024._semc_queueb_weight_struct.qos (*C var*), 841
drivers_RT_RT1020_MIMXRT1024._semc_queueb_weight_struct.slaveHitNoswitch (*C var*), 841
drivers_RT_RT1020_MIMXRT1024._semc_queueb_weight_struct.weightPagehit (*C var*), 841
drivers_RT_RT1020_MIMXRT1024._semc_queueb_weight.queuebConfig (*C var*), 841
drivers_RT_RT1020_MIMXRT1024._semc_queueb_weight.queuebValue (*C var*), 841
drivers_RT_RT1020_MIMXRT1024._semc_rdy_polarity (*C enum*), 825
drivers_RT_RT1020_MIMXRT1024._semc_rdy_polarity.kSEMC_RdyActivehigh (*C enumerator*), 826
drivers_RT_RT1020_MIMXRT1024._semc_rdy_polarity.kSEMC_RdyActiveLow (*C enumerator*), 826
drivers_RT_RT1020_MIMXRT1024._semc_refresh_time (*C enum*), 819
drivers_RT_RT1020_MIMXRT1024._semc_refresh_time.kSEMC_RefreshNineClocks (*C enumerator*), 819
drivers_RT_RT1020_MIMXRT1024._semc_refresh_time.kSEMC_RefreshSixClocks (*C enumerator*), 819
drivers_RT_RT1020_MIMXRT1024._semc_refresh_time.kSEMC_RefreshThreeClocks (*C enumerator*), 819
drivers_RT_RT1020_MIMXRT1024._semc_sdram_burst_len (*C enum*), 820
drivers_RT_RT1020_MIMXRT1024._semc_sdram_burst_len.kSEMC_Sdram_BurstLen1 (*C enumerator*), 820
drivers_RT_RT1020_MIMXRT1024._semc_sdram_column_bit_num (*C enum*), 819
drivers_RT_RT1020_MIMXRT1024._semc_sdram_column_bit_num.kSEMC_SdramColunm_8bit (*C enumerator*), 820
drivers_RT_RT1020_MIMXRT1024._semc_sdram_column_bit_num.kSEMC_SdramColunm_9bit (*C enumerator*), 819
drivers_RT_RT1020_MIMXRT1024._semc_sdram_column_bit_num.kSEMC_SdramColunm_10bit (*C enumerator*), 819
drivers_RT_RT1020_MIMXRT1024._semc_sdram_column_bit_num.kSEMC_SdramColunm_11bit (*C enumerator*), 819
drivers_RT_RT1020_MIMXRT1024._semc_sdram_column_bit_num.kSEMC_SdramColunm_12bit (*C enumerator*), 819
drivers_RT_RT1020_MIMXRT1024._semc_sdram_config (*C struct*), 831
drivers_RT_RT1020_MIMXRT1024._semc_sdram_config.address (*C var*), 832
drivers_RT_RT1020_MIMXRT1024._semc_sdram_config.autofreshTimes (*C var*), 833
drivers_RT_RT1020_MIMXRT1024._semc_sdram_config.burstLen (*C var*), 832
drivers_RT_RT1020_MIMXRT1024._semc_sdram_config.casLatency (*C var*), 832
drivers_RT_RT1020_MIMXRT1024._semc_sdram_config.columnAddrBitNum (*C var*), 832
drivers_RT_RT1020_MIMXRT1024._semc_sdram_config.csxPinMux (*C var*), 832
drivers_RT_RT1020_MIMXRT1024._semc_sdram_config.delayChain (*C var*), 833
drivers_RT_RT1020_MIMXRT1024._semc_sdram_config.memsize_kbytes (*C var*), 832
drivers_RT_RT1020_MIMXRT1024._semc_sdram_config.portSize (*C var*), 832
drivers_RT_RT1020_MIMXRT1024._semc_sdram_config.refreshBurstLen (*C var*), 833
drivers_RT_RT1020_MIMXRT1024._semc_sdram_config.refreshPeriod_nsPerRow (*C var*), 833
drivers_RT_RT1020_MIMXRT1024._semc_sdram_config.refreshUrgThreshold (*C var*), 833
drivers_RT_RT1020_MIMXRT1024._semc_sdram_config.tAct2Act_Ns (*C var*), 833
drivers_RT_RT1020_MIMXRT1024._semc_sdram_config.tAct2Prechage_Ns (*C var*), 833

drivers_RT_RT1020_MIMXRT1024._semc_sdram_config.tAct2ReadWrite_Ns (*C var*), 832
drivers_RT_RT1020_MIMXRT1024._semc_sdram_config.tCkeOff_Ns (*C var*), 832
drivers_RT_RT1020_MIMXRT1024._semc_sdram_config.tIdleTimeout_Ns (*C var*), 833
drivers_RT_RT1020_MIMXRT1024._semc_sdram_config.tPrecharge2Act_Ns (*C var*), 832
drivers_RT_RT1020_MIMXRT1024._semc_sdram_config.tPrescalePeriod_Ns (*C var*), 833
drivers_RT_RT1020_MIMXRT1024._semc_sdram_config.tRefresh2Refresh_Ns (*C var*), 833
drivers_RT_RT1020_MIMXRT1024._semc_sdram_config.tRefreshRecovery_Ns (*C var*), 832
drivers_RT_RT1020_MIMXRT1024._semc_sdram_config.tSelfRefRecovery_Ns (*C var*), 833
drivers_RT_RT1020_MIMXRT1024._semc_sdram_config.tWriteRecovery_Ns (*C var*), 832
drivers_RT_RT1020_MIMXRT1024._semc_sdram_cs (*C enum*), 817
drivers_RT_RT1020_MIMXRT1024._semc_sdram_cs.kSEMC_SDRAM_CS0 (*C enumerator*), 817
drivers_RT_RT1020_MIMXRT1024._semc_sdram_cs.kSEMC_SDRAM_CS1 (*C enumerator*), 817
drivers_RT_RT1020_MIMXRT1024._semc_sdram_cs.kSEMC_SDRAM_CS2 (*C enumerator*), 818
drivers_RT_RT1020_MIMXRT1024._semc_sdram_cs.kSEMC_SDRAM_CS3 (*C enumerator*), 818
drivers_RT_RT1020_MIMXRT1024._semc_sram_config (*C struct*), 837
drivers_RT_RT1020_MIMXRT1024._semc_sram_config.addr27 (*C var*), 837
drivers_RT_RT1020_MIMXRT1024._semc_sram_config.address (*C var*), 837
drivers_RT_RT1020_MIMXRT1024._semc_sram_config.addrMode (*C var*), 837
drivers_RT_RT1020_MIMXRT1024._semc_sram_config.addrPortWidth (*C var*), 837
drivers_RT_RT1020_MIMXRT1024._semc_sram_config.advActivePolarity (*C var*), 837
drivers_RT_RT1020_MIMXRT1024._semc_sram_config.advLevelCtrl (*C var*), 838
drivers_RT_RT1020_MIMXRT1024._semc_sram_config.burstLen (*C var*), 837
drivers_RT_RT1020_MIMXRT1024._semc_sram_config.cePinMux (*C var*), 837
drivers_RT_RT1020_MIMXRT1024._semc_sram_config.delayChain (*C var*), 839
drivers_RT_RT1020_MIMXRT1024._semc_sram_config.latencyCount (*C var*), 839
drivers_RT_RT1020_MIMXRT1024._semc_sram_config.memsize_kbytes (*C var*), 837
drivers_RT_RT1020_MIMXRT1024._semc_sram_config.portSize (*C var*), 837
drivers_RT_RT1020_MIMXRT1024._semc_sram_config.readCycle (*C var*), 839
drivers_RT_RT1020_MIMXRT1024._semc_sram_config.readHoldTime_Ns (*C var*), 838
drivers_RT_RT1020_MIMXRT1024._semc_sram_config.syncMode (*C var*), 838
drivers_RT_RT1020_MIMXRT1024._semc_sram_config.tAddr2WriteHold_Ns (*C var*), 838
drivers_RT_RT1020_MIMXRT1024._semc_sram_config.tAddrHold_Ns (*C var*), 838
drivers_RT_RT1020_MIMXRT1024._semc_sram_config.tAddrSetup_Ns (*C var*), 838
drivers_RT_RT1020_MIMXRT1024._semc_sram_config.tCeHold_Ns (*C var*), 838
drivers_RT_RT1020_MIMXRT1024._semc_sram_config.tCeInterval_Ns (*C var*), 838
drivers_RT_RT1020_MIMXRT1024._semc_sram_config.tCeSetup_Ns (*C var*), 838
drivers_RT_RT1020_MIMXRT1024._semc_sram_config.tReHigh_Ns (*C var*), 838
drivers_RT_RT1020_MIMXRT1024._semc_sram_config.tReLow_Ns (*C var*), 838
drivers_RT_RT1020_MIMXRT1024._semc_sram_config.tTurnAround_Ns (*C var*), 838
drivers_RT_RT1020_MIMXRT1024._semc_sram_config.tWeHigh_Ns (*C var*), 838
drivers_RT_RT1020_MIMXRT1024._semc_sram_config.tWeLow_Ns (*C var*), 838
drivers_RT_RT1020_MIMXRT1024._semc_sram_config.tWriteHold_Ns (*C var*), 839
drivers_RT_RT1020_MIMXRT1024._semc_sram_config.tWriteSetup_Ns (*C var*), 839
drivers_RT_RT1020_MIMXRT1024._semc_sram_config.waitEnable (*C var*), 838
drivers_RT_RT1020_MIMXRT1024._semc_sram_config.waitSample (*C var*), 838
drivers_RT_RT1020_MIMXRT1024._semc_sram_cs (*C enum*), 818
drivers_RT_RT1020_MIMXRT1024._semc_sram_cs.kSEMC_SRAM_CS0 (*C enumerator*), 818
drivers_RT_RT1020_MIMXRT1024._semc_sync_mode (*C enum*), 825
drivers_RT_RT1020_MIMXRT1024._semc_sync_mode.kSEMC_AsyncMode (*C enumerator*), 825
drivers_RT_RT1020_MIMXRT1024._semc_sync_mode.kSEMC_SyncMode (*C enumerator*), 825
drivers_RT_RT1020_MIMXRT1024._semc_waitready_polarity (*C enum*), 817
drivers_RT_RT1020_MIMXRT1024._semc_waitready_polarity.kSEMC_HighActive (*C enumerator*), 817
drivers_RT_RT1020_MIMXRT1024._semc_waitready_polarity.kSEMC_LowActive (*C enumerator*), 817
drivers_RT_RT1020_MIMXRT1024._snvs_hp_interrupts (*C enum*), 846
drivers_RT_RT1020_MIMXRT1024._snvs_hp_interrupts.kSNVS_RTC_AlarmInterrupt (*C enumerator*), 846

drivers_RT_RT1020_MIMXRT1024._snvs_hp_interrupts.kSNVS_RTC_PeriodicInterrupt (*C enumerator*), 846
drivers_RT_RT1020_MIMXRT1024._snvs_hp_rtc_config (*C struct*), 851
drivers_RT_RT1020_MIMXRT1024._snvs_hp_rtc_config.periodicInterruptFreq (*C var*), 851
drivers_RT_RT1020_MIMXRT1024._snvs_hp_rtc_config.rtcCalEnable (*C var*), 851
drivers_RT_RT1020_MIMXRT1024._snvs_hp_rtc_config.rtcCalValue (*C var*), 851
drivers_RT_RT1020_MIMXRT1024._snvs_hp_rtc_datetime (*C struct*), 850
drivers_RT_RT1020_MIMXRT1024._snvs_hp_rtc_datetime.day (*C var*), 851
drivers_RT_RT1020_MIMXRT1024._snvs_hp_rtc_datetime.hour (*C var*), 851
drivers_RT_RT1020_MIMXRT1024._snvs_hp_rtc_datetime.minute (*C var*), 851
drivers_RT_RT1020_MIMXRT1024._snvs_hp_rtc_datetime.month (*C var*), 851
drivers_RT_RT1020_MIMXRT1024._snvs_hp_rtc_datetime.second (*C var*), 851
drivers_RT_RT1020_MIMXRT1024._snvs_hp_rtc_datetime.year (*C var*), 851
drivers_RT_RT1020_MIMXRT1024._snvs_hp_ssm_state (*C enum*), 847
drivers_RT_RT1020_MIMXRT1024._snvs_hp_ssm_state.kSNVS_SSMCheck (*C enumerator*), 847
drivers_RT_RT1020_MIMXRT1024._snvs_hp_ssm_state.kSNVS_SSMHardFail (*C enumerator*), 847
drivers_RT_RT1020_MIMXRT1024._snvs_hp_ssm_state.kSNVS_SSMInit (*C enumerator*), 847
drivers_RT_RT1020_MIMXRT1024._snvs_hp_ssm_state.kSNVS_SSMInitInter (*C enumerator*), 847
drivers_RT_RT1020_MIMXRT1024._snvs_hp_ssm_state.kSNVS_SSMNonSecure (*C enumerator*), 847
drivers_RT_RT1020_MIMXRT1024._snvs_hp_ssm_state.kSNVS_SSMSecure (*C enumerator*), 847
drivers_RT_RT1020_MIMXRT1024._snvs_hp_ssm_state.kSNVS_SSMSoftFail (*C enumerator*), 847
drivers_RT_RT1020_MIMXRT1024._snvs_hp_ssm_state.kSNVS_SSMTrusted (*C enumerator*), 847
drivers_RT_RT1020_MIMXRT1024._snvs_hp_status_flags (*C enum*), 846
drivers_RT_RT1020_MIMXRT1024._snvs_hp_status_flags.kSNVS_OTPMK_ZeroFlag (*C enumerator*), 846
drivers_RT_RT1020_MIMXRT1024._snvs_hp_status_flags.kSNVS_RTC_AlarmInterruptFlag (*C enumerator*), 846
drivers_RT_RT1020_MIMXRT1024._snvs_hp_status_flags.kSNVS_RTC_PeriodicInterruptFlag (*C enumerator*), 846
drivers_RT_RT1020_MIMXRT1024._snvs_hp_status_flags.kSNVS_ZMK_ZeroFlag (*C enumerator*), 846
drivers_RT_RT1020_MIMXRT1024._snvs_hp_sv_status_flags (*C enum*), 846
drivers_RT_RT1020_MIMXRT1024._snvs_hp_sv_status_flags.kSNVS_FatalSoftwareViolationFlag (*C enumerator*), 847
drivers_RT_RT1020_MIMXRT1024._snvs_hp_sv_status_flags.kSNVS_LP_SoftwareViolationFlag (*C enumerator*), 846
drivers_RT_RT1020_MIMXRT1024._snvs_hp_sv_status_flags.kSNVS_LP_ViolationFlag (*C enumerator*), 846
drivers_RT_RT1020_MIMXRT1024._snvs_hp_sv_status_flags.kSNVS_SoftwareViolationFlag (*C enumerator*), 847
drivers_RT_RT1020_MIMXRT1024._snvs_hp_sv_status_flags.kSNVS_Violation0Flag (*C enumerator*), 847
drivers_RT_RT1020_MIMXRT1024._snvs_hp_sv_status_flags.kSNVS_Violation1Flag (*C enumerator*), 847
drivers_RT_RT1020_MIMXRT1024._snvs_hp_sv_status_flags.kSNVS_Violation2Flag (*C enumerator*), 847
drivers_RT_RT1020_MIMXRT1024._snvs_hp_sv_status_flags.kSNVS_Violation4Flag (*C enumerator*), 847
drivers_RT_RT1020_MIMXRT1024._snvs_hp_sv_status_flags.kSNVS_Violation5Flag (*C enumerator*), 847
drivers_RT_RT1020_MIMXRT1024._snvs_hp_sv_status_flags.kSNVS_ZMK_EccFailFlag (*C enumerator*), 846
drivers_RT_RT1020_MIMXRT1024._snvs_lp_active_clock (*C enum*), 858
drivers_RT_RT1020_MIMXRT1024._snvs_lp_active_clock.kSNVS_ActiveTamper2HZ (*C enumerator*), 858

drivers_RT_RT1020_MIMXRT1024._snvs_lp_active_clock.kSNVS_ActiveTamper4HZ (*C enumerator*), 858
drivers_RT_RT1020_MIMXRT1024._snvs_lp_active_clock.kSNVS_ActiveTamper8HZ (*C enumerator*), 858
drivers_RT_RT1020_MIMXRT1024._snvs_lp_active_clock.kSNVS_ActiveTamper16HZ (*C enumerator*), 858
drivers_RT_RT1020_MIMXRT1024._snvs_lp_active_tamper (*C enum*), 858
drivers_RT_RT1020_MIMXRT1024._snvs_lp_active_tamper.kSNVS_ActiveTamper1 (*C enumerator*), 858
drivers_RT_RT1020_MIMXRT1024._snvs_lp_active_tamper.kSNVS_ActiveTamper2 (*C enumerator*), 858
drivers_RT_RT1020_MIMXRT1024._snvs_lp_active_tamper.kSNVS_ActiveTamper3 (*C enumerator*), 858
drivers_RT_RT1020_MIMXRT1024._snvs_lp_active_tamper.kSNVS_ActiveTamper4 (*C enumerator*), 858
drivers_RT_RT1020_MIMXRT1024._snvs_lp_active_tamper.kSNVS_ActiveTamper5 (*C enumerator*), 858
drivers_RT_RT1020_MIMXRT1024._snvs_lp_external_tamper (*C enum*), 858
drivers_RT_RT1020_MIMXRT1024._snvs_lp_external_tamper_polarity (*C enum*), 858
drivers_RT_RT1020_MIMXRT1024._snvs_lp_external_tamper_polarity.kSNVS_ExternalTamperActiveHigh (*C enumerator*), 859
drivers_RT_RT1020_MIMXRT1024._snvs_lp_external_tamper_polarity.kSNVS_ExternalTamperActiveLow (*C enumerator*), 858
drivers_RT_RT1020_MIMXRT1024._snvs_lp_external_tamper_status (*C enum*), 858
drivers_RT_RT1020_MIMXRT1024._snvs_lp_external_tamper_status.kSNVS_TamperDetected (*C enumerator*), 858
drivers_RT_RT1020_MIMXRT1024._snvs_lp_external_tamper_status.kSNVS_TamperNotDetected (*C enumerator*), 858
drivers_RT_RT1020_MIMXRT1024._snvs_lp_external_tamper.kSNVS_ExternalTamper1 (*C enumerator*), 858
drivers_RT_RT1020_MIMXRT1024._snvs_lp_master_key_mode (*C enum*), 859
drivers_RT_RT1020_MIMXRT1024._snvs_lp_master_key_mode.kSNVS_CMK (*C enumerator*), 859
drivers_RT_RT1020_MIMXRT1024._snvs_lp_master_key_mode.kSNVS_OTPMK (*C enumerator*), 859
drivers_RT_RT1020_MIMXRT1024._snvs_lp_master_key_mode.kSNVS_ZMK (*C enumerator*), 859
drivers_RT_RT1020_MIMXRT1024._snvs_lp_srtc_config (*C struct*), 861
drivers_RT_RT1020_MIMXRT1024._snvs_lp_srtc_config.srtcCalEnable (*C var*), 861
drivers_RT_RT1020_MIMXRT1024._snvs_lp_srtc_config.srtcCalValue (*C var*), 861
drivers_RT_RT1020_MIMXRT1024._snvs_lp_srtc_datetime (*C struct*), 861
drivers_RT_RT1020_MIMXRT1024._snvs_lp_srtc_datetime.day (*C var*), 861
drivers_RT_RT1020_MIMXRT1024._snvs_lp_srtc_datetime.hour (*C var*), 861
drivers_RT_RT1020_MIMXRT1024._snvs_lp_srtc_datetime.minute (*C var*), 861
drivers_RT_RT1020_MIMXRT1024._snvs_lp_srtc_datetime.month (*C var*), 861
drivers_RT_RT1020_MIMXRT1024._snvs_lp_srtc_datetime.second (*C var*), 861
drivers_RT_RT1020_MIMXRT1024._snvs_lp_srtc_datetime.year (*C var*), 861
drivers_RT_RT1020_MIMXRT1024._snvs_lp_srtc_interrupts (*C enum*), 857
drivers_RT_RT1020_MIMXRT1024._snvs_lp_srtc_interrupts.kSNVS_SRTC_AlarmInterrupt (*C enumerator*), 857
drivers_RT_RT1020_MIMXRT1024._snvs_lp_srtc_status_flags (*C enum*), 857
drivers_RT_RT1020_MIMXRT1024._snvs_lp_srtc_status_flags.kSNVS_SRTC_AlarmInterruptFlag (*C enumerator*), 857
drivers_RT_RT1020_MIMXRT1024._snvs_lp_zmk_program_mode (*C enum*), 859
drivers_RT_RT1020_MIMXRT1024._snvs_lp_zmk_program_mode.kSNVS_ZMKHardwareProgram (*C enumerator*), 859
drivers_RT_RT1020_MIMXRT1024._snvs_lp_zmk_program_mode.kSNVS_ZMKSoftwareProgram (*C enumerator*), 859

drivers_RT_RT1020_MIMXRT1024._spdif_config (*C struct*), 875
drivers_RT_RT1020_MIMXRT1024._spdif_config.DPLLClkSource (*C var*), 875
drivers_RT_RT1020_MIMXRT1024._spdif_config.gain (*C var*), 876
drivers_RT_RT1020_MIMXRT1024._spdif_config.isRxAutoSync (*C var*), 875
drivers_RT_RT1020_MIMXRT1024._spdif_config.isTxAutoSync (*C var*), 875
drivers_RT_RT1020_MIMXRT1024._spdif_config.rxFullSelect (*C var*), 875
drivers_RT_RT1020_MIMXRT1024._spdif_config.txClkSource (*C var*), 875
drivers_RT_RT1020_MIMXRT1024._spdif_config.txFullSelect (*C var*), 875
drivers_RT_RT1020_MIMXRT1024._spdif_config.txSource (*C var*), 875
drivers_RT_RT1020_MIMXRT1024._spdif_config.uChannelSrc (*C var*), 875
drivers_RT_RT1020_MIMXRT1024._spdif_config.validityConfig (*C var*), 875
drivers_RT_RT1020_MIMXRT1024._spdif_edma_handle (*C struct*), 880
drivers_RT_RT1020_MIMXRT1024._spdif_edma_handle.callback (*C var*), 880
drivers_RT_RT1020_MIMXRT1024._spdif_edma_handle.count (*C var*), 880
drivers_RT_RT1020_MIMXRT1024._spdif_edma_handle.dmaLeftHandle (*C var*), 880
drivers_RT_RT1020_MIMXRT1024._spdif_edma_handle.dmaRightHandle (*C var*), 880
drivers_RT_RT1020_MIMXRT1024._spdif_edma_handle.leftTcd (*C var*), 880
drivers_RT_RT1020_MIMXRT1024._spdif_edma_handle.nbytes (*C var*), 880
drivers_RT_RT1020_MIMXRT1024._spdif_edma_handle.queueDriver (*C var*), 880
drivers_RT_RT1020_MIMXRT1024._spdif_edma_handle.queueUser (*C var*), 880
drivers_RT_RT1020_MIMXRT1024._spdif_edma_handle.rightTcd (*C var*), 880
drivers_RT_RT1020_MIMXRT1024._spdif_edma_handle.spdifQueue (*C var*), 880
drivers_RT_RT1020_MIMXRT1024._spdif_edma_handle.state (*C var*), 880
drivers_RT_RT1020_MIMXRT1024._spdif_edma_handle.transferSize (*C var*), 880
drivers_RT_RT1020_MIMXRT1024._spdif_edma_handle.userData (*C var*), 880
drivers_RT_RT1020_MIMXRT1024._spdif_edma_transfer (*C struct*), 879
drivers_RT_RT1020_MIMXRT1024._spdif_edma_transfer.dataSize (*C var*), 880
drivers_RT_RT1020_MIMXRT1024._spdif_edma_transfer.leftData (*C var*), 879
drivers_RT_RT1020_MIMXRT1024._spdif_edma_transfer.rightData (*C var*), 879
drivers_RT_RT1020_MIMXRT1024._spdif_gain_select (*C enum*), 872
drivers_RT_RT1020_MIMXRT1024._spdif_gain_select.kSPDIF_GAIN_3 (*C enumerator*), 872
drivers_RT_RT1020_MIMXRT1024._spdif_gain_select.kSPDIF_GAIN_4 (*C enumerator*), 872
drivers_RT_RT1020_MIMXRT1024._spdif_gain_select.kSPDIF_GAIN_6 (*C enumerator*), 872
drivers_RT_RT1020_MIMXRT1024._spdif_gain_select.kSPDIF_GAIN_8 (*C enumerator*), 872
drivers_RT_RT1020_MIMXRT1024._spdif_gain_select.kSPDIF_GAIN_12 (*C enumerator*), 872
drivers_RT_RT1020_MIMXRT1024._spdif_gain_select.kSPDIF_GAIN_16 (*C enumerator*), 872
drivers_RT_RT1020_MIMXRT1024._spdif_gain_select.kSPDIF_GAIN_24 (*C enumerator*), 872
drivers_RT_RT1020_MIMXRT1024._spdif_handle (*C struct*), 876
drivers_RT_RT1020_MIMXRT1024._spdif_handle.callback (*C var*), 876
drivers_RT_RT1020_MIMXRT1024._spdif_handle.queueDriver (*C var*), 876
drivers_RT_RT1020_MIMXRT1024._spdif_handle.queueUser (*C var*), 876
drivers_RT_RT1020_MIMXRT1024._spdif_handle.spdifQueue (*C var*), 876
drivers_RT_RT1020_MIMXRT1024._spdif_handle.state (*C var*), 876
drivers_RT_RT1020_MIMXRT1024._spdif_handle.transferSize (*C var*), 876
drivers_RT_RT1020_MIMXRT1024._spdif_handle.userData (*C var*), 876
drivers_RT_RT1020_MIMXRT1024._spdif_handle.watermark (*C var*), 877
drivers_RT_RT1020_MIMXRT1024._spdif_rxfull_select (*C enum*), 871
drivers_RT_RT1020_MIMXRT1024._spdif_rxfull_select.kSPDIF_RxFull1Sample (*C enumerator*), 871
drivers_RT_RT1020_MIMXRT1024._spdif_rxfull_select.kSPDIF_RxFull4Samples (*C enumerator*), 871
drivers_RT_RT1020_MIMXRT1024._spdif_rxfull_select.kSPDIF_RxFull8Samples (*C enumerator*), 871
drivers_RT_RT1020_MIMXRT1024._spdif_rxfull_select.kSPDIF_RxFull16Samples (*C enumerator*), 871
drivers_RT_RT1020_MIMXRT1024._spdif_transfer (*C struct*), 876
drivers_RT_RT1020_MIMXRT1024._spdif_transfer.data (*C var*), 876
drivers_RT_RT1020_MIMXRT1024._spdif_transfer.dataSize (*C var*), 876

drivers_RT_RT1020_MIMXRT1024._spdif_transfer.qdata (*C var*), 876
drivers_RT_RT1020_MIMXRT1024._spdif_transfer.udata (*C var*), 876
drivers_RT_RT1020_MIMXRT1024._spdif_tx_source (*C enum*), 872
drivers_RT_RT1020_MIMXRT1024._spdif_tx_source.kSPDIF_txFromReceiver (*C enumerator*), 872
drivers_RT_RT1020_MIMXRT1024._spdif_tx_source.kSPDIF_txNormal (*C enumerator*), 872
drivers_RT_RT1020_MIMXRT1024._spdif_txempty_select (*C enum*), 871
drivers_RT_RT1020_MIMXRT1024._spdif_txempty_select.kSPDIF_TxEmpty0Sample (*C enumerator*), 871
drivers_RT_RT1020_MIMXRT1024._spdif_txempty_select.kSPDIF_TxEmpty4Samples (*C enumerator*), 871
drivers_RT_RT1020_MIMXRT1024._spdif_txempty_select.kSPDIF_TxEmpty8Samples (*C enumerator*), 871
drivers_RT_RT1020_MIMXRT1024._spdif_txempty_select.kSPDIF_TxEmpty12Samples (*C enumerator*), 871
drivers_RT_RT1020_MIMXRT1024._spdif_uchannel_source (*C enum*), 871
drivers_RT_RT1020_MIMXRT1024._spdif_uchannel_source.kSPDIF_NoUChannel (*C enumerator*), 872
drivers_RT_RT1020_MIMXRT1024._spdif_uchannel_source.kSPDIF_UChannelFromRx (*C enumerator*), 872
drivers_RT_RT1020_MIMXRT1024._spdif_uchannel_source.kSPDIF_UChannelFromTx (*C enumerator*), 872
drivers_RT_RT1020_MIMXRT1024._spdif_validity_config (*C enum*), 872
drivers_RT_RT1020_MIMXRT1024._spdif_validity_config.kSPDIF_validityFlagAlwaysClear (*C enumerator*), 873
drivers_RT_RT1020_MIMXRT1024._spdif_validity_config.kSPDIF_validityFlagAlwaysSet (*C enumerator*), 873
drivers_RT_RT1020_MIMXRT1024._src_mix_reset_stretch_cycles (*C enum*), 882
drivers_RT_RT1020_MIMXRT1024._src_mix_reset_stretch_cycles.kSRC_MixResetStretchCycleAlt0 (*C enumerator*), 882
drivers_RT_RT1020_MIMXRT1024._src_mix_reset_stretch_cycles.kSRC_MixResetStretchCycleAlt1 (*C enumerator*), 882
drivers_RT_RT1020_MIMXRT1024._src_mix_reset_stretch_cycles.kSRC_MixResetStretchCycleAlt2 (*C enumerator*), 882
drivers_RT_RT1020_MIMXRT1024._src_mix_reset_stretch_cycles.kSRC_MixResetStretchCycleAlt3 (*C enumerator*), 882
drivers_RT_RT1020_MIMXRT1024._src_reset_status_flags (*C enum*), 881
drivers_RT_RT1020_MIMXRT1024._src_reset_status_flags.kSRC_CoreLockupResetFlag (*C enumerator*), 882
drivers_RT_RT1020_MIMXRT1024._src_reset_status_flags.kSRC_CsuResetFlag (*C enumerator*), 881
drivers_RT_RT1020_MIMXRT1024._src_reset_status_flags.kSRC_IppResetPinFlag (*C enumerator*), 882
drivers_RT_RT1020_MIMXRT1024._src_reset_status_flags.kSRC_IppUserResetFlag (*C enumerator*), 881
drivers_RT_RT1020_MIMXRT1024._src_reset_status_flags.kSRC_JTAGGeneratedResetFlag (*C enumerator*), 881
drivers_RT_RT1020_MIMXRT1024._src_reset_status_flags.kSRC_JTAGSoftwareResetFlag (*C enumerator*), 881
drivers_RT_RT1020_MIMXRT1024._src_reset_status_flags.kSRC_JTAGSystemResetFlag (*C enumerator*), 881
drivers_RT_RT1020_MIMXRT1024._src_reset_status_flags.kSRC_LockupSysResetFlag (*C enumerator*), 882
drivers_RT_RT1020_MIMXRT1024._src_reset_status_flags.kSRC_PowerOnResetFlag (*C enumerator*), 882
drivers_RT_RT1020_MIMXRT1024._src_reset_status_flags.kSRC_ResetOutputEnableFlag (*C enumerator*), 881
drivers_RT_RT1020_MIMXRT1024._src_reset_status_flags.kSRC_SNVSFailResetFlag (*C enumera-*

tor), 881
drivers_RT_RT1020_MIMXRT1024._src_reset_status_flags.kSRC_SoftwareResetFlag (*C enumerator*), 881
drivers_RT_RT1020_MIMXRT1024._src_reset_status_flags.kSRC_TemperatureSensorResetFlag (*C enumerator*), 881
drivers_RT_RT1020_MIMXRT1024._src_reset_status_flags.kSRC_WatchdogResetFlag (*C enumerator*), 881
drivers_RT_RT1020_MIMXRT1024._src_reset_status_flags.kSRC_Wdog3ResetFlag (*C enumerator*), 881
drivers_RT_RT1020_MIMXRT1024._src_status_flags (*C enum*), 882
drivers_RT_RT1020_MIMXRT1024._src_status_flags.kSRC_Core0WdogResetReqFlag (*C enumerator*), 882
drivers_RT_RT1020_MIMXRT1024._src_warm_reset_bypass_count (*C enum*), 883
drivers_RT_RT1020_MIMXRT1024._src_warm_reset_bypass_count.kSRC_WarmResetWaitAlways (*C enumerator*), 883
drivers_RT_RT1020_MIMXRT1024._src_warm_reset_bypass_count.kSRC_WarmResetWaitClk16 (*C enumerator*), 883
drivers_RT_RT1020_MIMXRT1024._src_warm_reset_bypass_count.kSRC_WarmResetWaitClk32 (*C enumerator*), 883
drivers_RT_RT1020_MIMXRT1024._src_warm_reset_bypass_count.kSRC_WarmResetWaitClk64 (*C enumerator*), 883
drivers_RT_RT1020_MIMXRT1024._src_wdog3_reset_option (*C enum*), 882
drivers_RT_RT1020_MIMXRT1024._src_wdog3_reset_option.kSRC_Wdog3ResetOptionAlt0 (*C enumerator*), 882
drivers_RT_RT1020_MIMXRT1024._src_wdog3_reset_option.kSRC_Wdog3ResetOptionAlt1 (*C enumerator*), 882
drivers_RT_RT1020_MIMXRT1024._status_groups (*C enum*), 612
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_AON_COMMON (*C enumerator*), 620
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_AON_POWER (*C enumerator*), 620
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_ApplicationRangeStart (*C enumerator*), 617
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_ASRC (*C enumerator*), 619
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_AUDMIX (*C enumerator*), 617
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_BMA (*C enumerator*), 620
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_BUTTON (*C enumerator*), 618
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_CAAM (*C enumerator*), 615
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_CLIF (*C enumerator*), 620
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_CODEC (*C enumerator*), 619
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_COMMON_TASK (*C enumerator*), 619
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_CSI (*C enumerator*), 614
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_CSS_PKC (*C enumerator*), 620
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_DCP (*C enumerator*), 616
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_DebugConsole (*C enumerator*), 617
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_DMA (*C enumerator*), 615
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_DMAMGR (*C enumerator*), 615
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_DMIC (*C enumerator*), 615
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_DSPI (*C enumerator*), 612
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_ECSPI (*C enumerator*), 615
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_EDMA (*C enumerator*), 615
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_ELA_CSEC (*C enumerator*), 620
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_ELE (*C enumerator*), 620
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_ELEMU (*C enumerator*), 619
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_ELS_PKC (*C enumerator*), 620

drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_ENDAT3 (*C enumerator*), 620
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_ENET (*C enumerator*), 614
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_ENET_QOS (*C enumerator*), 619
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_ESAI (*C enumerator*), 616
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_EVENT_CTRL (*C enumerator*), 621
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_EXTERN_EEPROM (*C enumerator*), 618
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_FLASH (*C enumerator*), 612
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_FLASHIAP (*C enumerator*), 614
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_FLEXCAN (*C enumerator*), 615
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_FLEXCOMM_I2C (*C enumerator*), 614
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_FLEXIO_A_FORMAT (*C enumerator*), 621
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_FLEXIO_CAMERA (*C enumerator*), 615
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_FLEXIO_I2C (*C enumerator*), 613
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_FLEXIO_I2S (*C enumerator*), 614
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_FLEXIO_MCULCD (*C enumerator*), 614
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_FLEXIO_SPI (*C enumerator*), 612
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_FLEXIO_T_FORMAT (*C enumerator*), 620
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_FLEXIO_UART (*C enumerator*), 613
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_FLEXSPI (*C enumerator*), 616
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_Generic (*C enumerator*), 612
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_GLIKEY (*C enumerator*), 620
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_HAL_ADC_SENSOR (*C enumerator*), 618
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_HAL_FLASH (*C enumerator*), 618
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_HAL_GPIO (*C enumerator*), 617
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_HAL_I2C (*C enumerator*), 618
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_HAL_I2S (*C enumerator*), 618
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_HAL_PWM (*C enumerator*), 618
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_HAL_RNG (*C enumerator*), 618
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_HAL_SPI (*C enumerator*), 618
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_HAL_TIMER (*C enumerator*), 617
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_HAL_UART (*C enumerator*), 617
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_HASHCRYPT (*C enumerator*), 616
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_HIPERFACE (*C enumerator*), 620
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_HOSTIF (*C enumerator*), 620
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_I2C (*C enumerator*), 613
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_I2S (*C enumerator*), 614
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_I3C (*C enumerator*), 616
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_I3CBUS (*C enumerator*), 619

drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_IAP (*C enumerator*), 617
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_ICS (*C enumerator*), 616
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_IPED (*C enumerator*), 620
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_IUART (*C enumerator*), 614
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_JPEGDEC (*C enumerator*), 617
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_LED (*C enumerator*), 618
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_LIST (*C enumerator*), 618
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_LMEM (*C enumerator*), 614
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_LOG (*C enumerator*), 619
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_LPC_I2C (*C enumerator*), 616
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_LPC_I2C_1 (*C enumerator*), 616
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_LPC_MINISPI (*C enumerator*), 616
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_LPC_QSPI (*C enumerator*), 621
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_LPC_SPI (*C enumerator*), 615
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_LPC_SPI_SSP (*C enumerator*), 616
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_LPC_USART (*C enumerator*), 615
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_LPI2C (*C enumerator*), 613
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_LPSCI (*C enumerator*), 613
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_LPSPI (*C enumerator*), 612
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_LPUART (*C enumerator*), 613
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_LTC (*C enumerator*), 615
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_MCAN (*C enumerator*), 615
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_MCG (*C enumerator*), 613
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_MECC (*C enumerator*), 619
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_MEM_MANAGER (*C enumerator*), 618
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_MIPI_DSI (*C enumerator*), 614
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_MMDC (*C enumerator*), 616
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_MSCAN (*C enumerator*), 616
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_MSG (*C enumerator*), 619
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_NETC (*C enumerator*), 620
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_NOTIFIER (*C enumerator*), 616
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_NPX (*C enumerator*), 620
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_OSA (*C enumerator*), 618
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_OTFAD (*C enumerator*), 619
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_OTP (*C enumerator*), 615
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_PDM (*C enumerator*), 616
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_PHY (*C enumerator*), 614
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_PNGDEC (*C enumerator*), 617
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_POWER (*C enumerator*), 614
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_POWER_MANAGER (*C enumerator*), 619
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_PUF (*C enumerator*), 617
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_QSCI (*C enumerator*), 619
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_QSPI (*C enumerator*), 614
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_QUEUEDSPI (*C enumerator*), 619
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_SAI (*C enumerator*), 613
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_SCG (*C enumerator*), 613
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_SDHC (*C enumerator*), 613
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_SDIF (*C enumerator*), 615
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_SDIOSLV (*C enumerator*), 619
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_SDK_FLEXSPINOR (*C enumerator*), 619

drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_SDK_OCOTP (*C enumerator*), 619
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_SDMA (*C enumerator*), 616
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_SDMMC (*C enumerator*), 613
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_SDRAMC (*C enumerator*), 614
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_SDSPI (*C enumerator*), 613
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_SEMA42 (*C enumerator*), 613
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_SEMC (*C enumerator*), 617
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_SERIALMANAGER (*C enumerator*), 618
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_SFA (*C enumerator*), 617
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_SHELL (*C enumerator*), 618
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_SMARTCARD (*C enumerator*), 614
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_SPC (*C enumerator*), 617
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_SPDIF (*C enumerator*), 616
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_SPI (*C enumerator*), 613
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_SPIFI (*C enumerator*), 615
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_TIMERMANAGER (*C enumerator*), 618
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_TOUCH_PANEL (*C enumerator*), 617
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_TRGMUX (*C enumerator*), 614
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_UART (*C enumerator*), 613
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_USDHC (*C enumerator*), 615
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_VBAT (*C enumerator*), 617
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_XRDC (*C enumerator*), 613
drivers_RT_RT1020_MIMXRT1024._status_groups.kStatusGroup_XSPI (*C enumerator*), 617
drivers_RT_RT1020_MIMXRT1024._tempmon_alarm_mode (*C enum*), 887
drivers_RT_RT1020_MIMXRT1024._tempmon_alarm_mode.kTEMPMON_HighAlarmMode (*C enumerator*), 887
drivers_RT_RT1020_MIMXRT1024._tempmon_alarm_mode.kTEMPMON_LowAlarmMode (*C enumerator*), 887
drivers_RT_RT1020_MIMXRT1024._tempmon_alarm_mode.kTEMPMON_PanicAlarmMode (*C enumerator*), 887
drivers_RT_RT1020_MIMXRT1024._tempmon_config (*C struct*), 888
drivers_RT_RT1020_MIMXRT1024._tempmon_config.frequency (*C var*), 888
drivers_RT_RT1020_MIMXRT1024._tempmon_config.highAlarmTemp (*C var*), 888
drivers_RT_RT1020_MIMXRT1024._tempmon_config.lowAlarmTemp (*C var*), 889
drivers_RT_RT1020_MIMXRT1024._tempmon_config.panicAlarmTemp (*C var*), 889
drivers_RT_RT1020_MIMXRT1024._trng_clock_mode (*C enum*), 891
drivers_RT_RT1020_MIMXRT1024._trng_clock_mode.kTRNG_ClockModeRingOscillator (*C enumerator*), 891
drivers_RT_RT1020_MIMXRT1024._trng_clock_mode.kTRNG_ClockModeSystem (*C enumerator*), 891
drivers_RT_RT1020_MIMXRT1024._trng_ring_osc_div (*C enum*), 891
drivers_RT_RT1020_MIMXRT1024._trng_ring_osc_div.kTRNG_RingOscDiv0 (*C enumerator*), 891
drivers_RT_RT1020_MIMXRT1024._trng_ring_osc_div.kTRNG_RingOscDiv2 (*C enumerator*), 891
drivers_RT_RT1020_MIMXRT1024._trng_ring_osc_div.kTRNG_RingOscDiv4 (*C enumerator*), 891
drivers_RT_RT1020_MIMXRT1024._trng_ring_osc_div.kTRNG_RingOscDiv8 (*C enumerator*), 891
drivers_RT_RT1020_MIMXRT1024._trng_sample_mode (*C enum*), 890
drivers_RT_RT1020_MIMXRT1024._trng_sample_mode.kTRNG_SampleModeRaw (*C enumerator*), 890
drivers_RT_RT1020_MIMXRT1024._trng_sample_mode.kTRNG_SampleModeVonNeumann (*C*

enumerator), 890
drivers_RT_RT1020_MIMXRT1024._trng_sample_mode.kTRNG_SampleModeVonNeumannRaw (*C enumerator*), 890
drivers_RT_RT1020_MIMXRT1024._trng_statistical_check_limit (*C struct*), 892
drivers_RT_RT1020_MIMXRT1024._trng_statistical_check_limit.maximum (*C var*), 892
drivers_RT_RT1020_MIMXRT1024._trng_statistical_check_limit.minimum (*C var*), 892
drivers_RT_RT1020_MIMXRT1024._trng_user_config (*C struct*), 893
drivers_RT_RT1020_MIMXRT1024._trng_user_config.clockMode (*C var*), 893
drivers_RT_RT1020_MIMXRT1024._trng_user_config.entropyDelay (*C var*), 893
drivers_RT_RT1020_MIMXRT1024._trng_user_config.frequencyCountLimit (*C var*), 894
drivers_RT_RT1020_MIMXRT1024._trng_user_config.lock (*C var*), 893
drivers_RT_RT1020_MIMXRT1024._trng_user_config.longRunMaxLimit (*C var*), 893
drivers_RT_RT1020_MIMXRT1024._trng_user_config.monobitLimit (*C var*), 893
drivers_RT_RT1020_MIMXRT1024._trng_user_config.pokerLimit (*C var*), 894
drivers_RT_RT1020_MIMXRT1024._trng_user_config.retryCount (*C var*), 893
drivers_RT_RT1020_MIMXRT1024._trng_user_config.ringOscDiv (*C var*), 893
drivers_RT_RT1020_MIMXRT1024._trng_user_config.runBit1Limit (*C var*), 893
drivers_RT_RT1020_MIMXRT1024._trng_user_config.runBit2Limit (*C var*), 894
drivers_RT_RT1020_MIMXRT1024._trng_user_config.runBit3Limit (*C var*), 894
drivers_RT_RT1020_MIMXRT1024._trng_user_config.runBit4Limit (*C var*), 894
drivers_RT_RT1020_MIMXRT1024._trng_user_config.runBit5Limit (*C var*), 894
drivers_RT_RT1020_MIMXRT1024._trng_user_config.runBit6PlusLimit (*C var*), 894
drivers_RT_RT1020_MIMXRT1024._trng_user_config.sampleMode (*C var*), 893
drivers_RT_RT1020_MIMXRT1024._trng_user_config.sampleSize (*C var*), 893
drivers_RT_RT1020_MIMXRT1024._trng_user_config.sparseBitLimit (*C var*), 893
drivers_RT_RT1020_MIMXRT1024._usdhc_adma2_descriptor (*C struct*), 923
drivers_RT_RT1020_MIMXRT1024._usdhc_adma2_descriptor.address (*C var*), 923
drivers_RT_RT1020_MIMXRT1024._usdhc_adma2_descriptor.attribute (*C var*), 923
drivers_RT_RT1020_MIMXRT1024._usdhc_adma_config (*C struct*), 925
drivers_RT_RT1020_MIMXRT1024._usdhc_adma_config.admaTable (*C var*), 925
drivers_RT_RT1020_MIMXRT1024._usdhc_adma_config.admaTableWords (*C var*), 925
drivers_RT_RT1020_MIMXRT1024._usdhc_adma_config.dmaMode (*C var*), 925
drivers_RT_RT1020_MIMXRT1024._usdhc_boot_config (*C struct*), 923
drivers_RT_RT1020_MIMXRT1024._usdhc_boot_config.ackTimeoutCount (*C var*), 923
drivers_RT_RT1020_MIMXRT1024._usdhc_boot_config.blockCount (*C var*), 924
drivers_RT_RT1020_MIMXRT1024._usdhc_boot_config.blockSize (*C var*), 924
drivers_RT_RT1020_MIMXRT1024._usdhc_boot_config.bootMode (*C var*), 923
drivers_RT_RT1020_MIMXRT1024._usdhc_boot_config.enableAutoStopAtBlockGap (*C var*), 924
drivers_RT_RT1020_MIMXRT1024._usdhc_boot_config.enableBootAck (*C var*), 924
drivers_RT_RT1020_MIMXRT1024._usdhc_boot_mode (*C enum*), 916
drivers_RT_RT1020_MIMXRT1024._usdhc_boot_mode.kUSDHC_BootModeAlternative (*C enumerator*), 916
drivers_RT_RT1020_MIMXRT1024._usdhc_boot_mode.kUSDHC_BootModeNormal (*C enumerator*), 916
drivers_RT_RT1020_MIMXRT1024._usdhc_burst_len (*C enum*), 919
drivers_RT_RT1020_MIMXRT1024._usdhc_burst_len.kUSDHC_EnBurstLenForINCR (*C enumerator*), 919
drivers_RT_RT1020_MIMXRT1024._usdhc_burst_len.kUSDHC_EnBurstLenForINCR4816 (*C enumerator*), 919
drivers_RT_RT1020_MIMXRT1024._usdhc_burst_len.kUSDHC_EnBurstLenForINCR4816WRAP (*C enumerator*), 919
drivers_RT_RT1020_MIMXRT1024._usdhc_capability (*C struct*), 923
drivers_RT_RT1020_MIMXRT1024._usdhc_capability.flags (*C var*), 923
drivers_RT_RT1020_MIMXRT1024._usdhc_capability.maxBlockCount (*C var*), 923
drivers_RT_RT1020_MIMXRT1024._usdhc_capability.maxBlockLength (*C var*), 923
drivers_RT_RT1020_MIMXRT1024._usdhc_capability.mmcVersion (*C var*), 923
drivers_RT_RT1020_MIMXRT1024._usdhc_capability.sdVersion (*C var*), 923
drivers_RT_RT1020_MIMXRT1024._usdhc_card_command_type (*C enum*), 916

drivers_RT_RT1020_MIMXRT1024._usdhc_card_command_type.kCARD_CommandTypeAbort (*C enumerator*), 916
drivers_RT_RT1020_MIMXRT1024._usdhc_card_command_type.kCARD_CommandTypeEmpty (*C enumerator*), 916
drivers_RT_RT1020_MIMXRT1024._usdhc_card_command_type.kCARD_CommandTypeNormal (*C enumerator*), 916
drivers_RT_RT1020_MIMXRT1024._usdhc_card_command_type.kCARD_CommandTypeResume (*C enumerator*), 916
drivers_RT_RT1020_MIMXRT1024._usdhc_card_command_type.kCARD_CommandTypeSuspend (*C enumerator*), 916
drivers_RT_RT1020_MIMXRT1024._usdhc_card_response_type (*C enum*), 916
drivers_RT_RT1020_MIMXRT1024._usdhc_card_response_type.kCARD_ResponseTypeNone (*C enumerator*), 917
drivers_RT_RT1020_MIMXRT1024._usdhc_card_response_type.kCARD_ResponseTypeR1 (*C enumerator*), 917
drivers_RT_RT1020_MIMXRT1024._usdhc_card_response_type.kCARD_ResponseTypeR1b (*C enumerator*), 917
drivers_RT_RT1020_MIMXRT1024._usdhc_card_response_type.kCARD_ResponseTypeR2 (*C enumerator*), 917
drivers_RT_RT1020_MIMXRT1024._usdhc_card_response_type.kCARD_ResponseTypeR3 (*C enumerator*), 917
drivers_RT_RT1020_MIMXRT1024._usdhc_card_response_type.kCARD_ResponseTypeR4 (*C enumerator*), 917
drivers_RT_RT1020_MIMXRT1024._usdhc_card_response_type.kCARD_ResponseTypeR5 (*C enumerator*), 917
drivers_RT_RT1020_MIMXRT1024._usdhc_card_response_type.kCARD_ResponseTypeR5b (*C enumerator*), 917
drivers_RT_RT1020_MIMXRT1024._usdhc_card_response_type.kCARD_ResponseTypeR6 (*C enumerator*), 917
drivers_RT_RT1020_MIMXRT1024._usdhc_card_response_type.kCARD_ResponseTypeR7 (*C enumerator*), 917
drivers_RT_RT1020_MIMXRT1024._usdhc_command (*C struct*), 924
drivers_RT_RT1020_MIMXRT1024._usdhc_command.argument (*C var*), 924
drivers_RT_RT1020_MIMXRT1024._usdhc_command.flags (*C var*), 925
drivers_RT_RT1020_MIMXRT1024._usdhc_command.index (*C var*), 924
drivers_RT_RT1020_MIMXRT1024._usdhc_command.response (*C var*), 925
drivers_RT_RT1020_MIMXRT1024._usdhc_command.responseErrorFlags (*C var*), 925
drivers_RT_RT1020_MIMXRT1024._usdhc_command.responseType (*C var*), 924
drivers_RT_RT1020_MIMXRT1024._usdhc_command.type (*C var*), 924
drivers_RT_RT1020_MIMXRT1024._usdhc_config (*C struct*), 924
drivers_RT_RT1020_MIMXRT1024._usdhc_config.dataTimeout (*C var*), 924
drivers_RT_RT1020_MIMXRT1024._usdhc_config.endianMode (*C var*), 924
drivers_RT_RT1020_MIMXRT1024._usdhc_config.readWatermarkLevel (*C var*), 924
drivers_RT_RT1020_MIMXRT1024._usdhc_config.writeWatermarkLevel (*C var*), 924
drivers_RT_RT1020_MIMXRT1024._usdhc_data (*C struct*), 926
drivers_RT_RT1020_MIMXRT1024._usdhc_data_bus_width (*C enum*), 915
drivers_RT_RT1020_MIMXRT1024._usdhc_data_bus_width.kUSDHC_DataBusWidth1Bit (*C enumerator*), 915
drivers_RT_RT1020_MIMXRT1024._usdhc_data_bus_width.kUSDHC_DataBusWidth4Bit (*C enumerator*), 915
drivers_RT_RT1020_MIMXRT1024._usdhc_data_bus_width.kUSDHC_DataBusWidth8Bit (*C enumerator*), 915
drivers_RT_RT1020_MIMXRT1024._usdhc_data.blockCount (*C var*), 927
drivers_RT_RT1020_MIMXRT1024._usdhc_data.blockSize (*C var*), 926
drivers_RT_RT1020_MIMXRT1024._usdhc_data.dataType (*C var*), 926
drivers_RT_RT1020_MIMXRT1024._usdhc_data.enableAutoCommand12 (*C var*), 926
drivers_RT_RT1020_MIMXRT1024._usdhc_data.enableAutoCommand23 (*C var*), 926
drivers_RT_RT1020_MIMXRT1024._usdhc_data.enableIgnoreError (*C var*), 926

drivers_RT_RT1020_MIMXRT1024._usdhc_data.rxData (*C var*), 927
drivers_RT_RT1020_MIMXRT1024._usdhc_data.txData (*C var*), 927
drivers_RT_RT1020_MIMXRT1024._usdhc_dma_mode (*C enum*), 915
drivers_RT_RT1020_MIMXRT1024._usdhc_dma_mode.kUSDHC_DmaModeAdma1 (*C enumerator*), 915
drivers_RT_RT1020_MIMXRT1024._usdhc_dma_mode.kUSDHC_DmaModeAdma2 (*C enumerator*), 915
drivers_RT_RT1020_MIMXRT1024._usdhc_dma_mode.kUSDHC_DmaModeSimple (*C enumerator*), 915
drivers_RT_RT1020_MIMXRT1024._usdhc_dma_mode.kUSDHC_ExternalDMA (*C enumerator*), 915
drivers_RT_RT1020_MIMXRT1024._usdhc_endian_mode (*C enum*), 915
drivers_RT_RT1020_MIMXRT1024._usdhc_endian_mode.kUSDHC_EndianModeBig (*C enumerator*), 915
drivers_RT_RT1020_MIMXRT1024._usdhc_endian_mode.kUSDHC_EndianModeHalfWordBig (*C enumerator*), 915
drivers_RT_RT1020_MIMXRT1024._usdhc_endian_mode.kUSDHC_EndianModeLittle (*C enumerator*), 915
drivers_RT_RT1020_MIMXRT1024._usdhc_handle (*C struct*), 927
drivers_RT_RT1020_MIMXRT1024._usdhc_handle.callback (*C var*), 928
drivers_RT_RT1020_MIMXRT1024._usdhc_handle.command (*C var*), 928
drivers_RT_RT1020_MIMXRT1024._usdhc_handle.data (*C var*), 928
drivers_RT_RT1020_MIMXRT1024._usdhc_handle.enDMA (*C var*), 928
drivers_RT_RT1020_MIMXRT1024._usdhc_handle.transferredWords (*C var*), 928
drivers_RT_RT1020_MIMXRT1024._usdhc_handle.userData (*C var*), 928
drivers_RT_RT1020_MIMXRT1024._usdhc_host (*C struct*), 928
drivers_RT_RT1020_MIMXRT1024._usdhc_host.base (*C var*), 928
drivers_RT_RT1020_MIMXRT1024._usdhc_host.capability (*C var*), 928
drivers_RT_RT1020_MIMXRT1024._usdhc_host.config (*C var*), 928
drivers_RT_RT1020_MIMXRT1024._usdhc_host.sourceClock_Hz (*C var*), 928
drivers_RT_RT1020_MIMXRT1024._usdhc_host.transfer (*C var*), 928
drivers_RT_RT1020_MIMXRT1024._usdhc_scatter_gather_data (*C struct*), 925
drivers_RT_RT1020_MIMXRT1024._usdhc_scatter_gather_data_list (*C struct*), 925
drivers_RT_RT1020_MIMXRT1024._usdhc_scatter_gather_data.blockSize (*C var*), 926
drivers_RT_RT1020_MIMXRT1024._usdhc_scatter_gather_data.dataDirection (*C var*), 926
drivers_RT_RT1020_MIMXRT1024._usdhc_scatter_gather_data.dataType (*C var*), 926
drivers_RT_RT1020_MIMXRT1024._usdhc_scatter_gather_data.enableAutoCommand12 (*C var*), 925
drivers_RT_RT1020_MIMXRT1024._usdhc_scatter_gather_data.enableAutoCommand23 (*C var*), 925
drivers_RT_RT1020_MIMXRT1024._usdhc_scatter_gather_data.enableIgnoreError (*C var*), 925
drivers_RT_RT1020_MIMXRT1024._usdhc_scatter_gather_data.sgData (*C var*), 926
drivers_RT_RT1020_MIMXRT1024._usdhc_scatter_gather_transfer (*C struct*), 926
drivers_RT_RT1020_MIMXRT1024._usdhc_scatter_gather_transfer.command (*C var*), 926
drivers_RT_RT1020_MIMXRT1024._usdhc_scatter_gather_transfer.data (*C var*), 926
drivers_RT_RT1020_MIMXRT1024._usdhc_transfer (*C struct*), 927
drivers_RT_RT1020_MIMXRT1024._usdhc_transfer_callback (*C struct*), 927
drivers_RT_RT1020_MIMXRT1024._usdhc_transfer_callback.BlockGap (*C var*), 927
drivers_RT_RT1020_MIMXRT1024._usdhc_transfer_callback.CardInserted (*C var*), 927
drivers_RT_RT1020_MIMXRT1024._usdhc_transfer_callback.CardRemoved (*C var*), 927
drivers_RT_RT1020_MIMXRT1024._usdhc_transfer_callback.ReTuning (*C var*), 927
drivers_RT_RT1020_MIMXRT1024._usdhc_transfer_callback.SdioInterrupt (*C var*), 927
drivers_RT_RT1020_MIMXRT1024._usdhc_transfer_callback.TransferComplete (*C var*), 927
drivers_RT_RT1020_MIMXRT1024._usdhc_transfer_direction (*C enum*), 914
drivers_RT_RT1020_MIMXRT1024._usdhc_transfer_direction.kUSDHC_TransferDirectionReceive (*C enumerator*), 914
drivers_RT_RT1020_MIMXRT1024._usdhc_transfer_direction.kUSDHC_TransferDirectionSend (*C enumerator*), 915

drivers_RT_RT1020_MIMXRT1024._usdhc_transfer.command (*C var*), 927
drivers_RT_RT1020_MIMXRT1024._usdhc_transfer.data (*C var*), 927
drivers_RT_RT1020_MIMXRT1024._wdog_config (*C struct*), 933
drivers_RT_RT1020_MIMXRT1024._wdog_config.enableInterrupt (*C var*), 933
drivers_RT_RT1020_MIMXRT1024._wdog_config.enablePowerDown (*C var*), 933
drivers_RT_RT1020_MIMXRT1024._wdog_config.enableTimeOutAssert (*C var*), 933
drivers_RT_RT1020_MIMXRT1024._wdog_config.enableWdog (*C var*), 933
drivers_RT_RT1020_MIMXRT1024._wdog_config.interruptTimeValue (*C var*), 933
drivers_RT_RT1020_MIMXRT1024._wdog_config.softwareResetExtension (*C var*), 933
drivers_RT_RT1020_MIMXRT1024._wdog_config.timeoutValue (*C var*), 933
drivers_RT_RT1020_MIMXRT1024._wdog_config.workMode (*C var*), 933
drivers_RT_RT1020_MIMXRT1024._wdog_interrupt_enable (*C enum*), 932
drivers_RT_RT1020_MIMXRT1024._wdog_interrupt_enable.kWDOG_InterruptEnable (*C enumerator*), 932
drivers_RT_RT1020_MIMXRT1024._wdog_status_flags (*C enum*), 932
drivers_RT_RT1020_MIMXRT1024._wdog_status_flags.kWDOG_InterruptFlag (*C enumerator*), 932
drivers_RT_RT1020_MIMXRT1024._wdog_status_flags.kWDOG_PowerOnResetFlag (*C enumerator*), 932
drivers_RT_RT1020_MIMXRT1024._wdog_status_flags.kWDOG_RunningFlag (*C enumerator*), 932
drivers_RT_RT1020_MIMXRT1024._wdog_status_flags.kWDOG_SoftwareResetFlag (*C enumerator*), 932
drivers_RT_RT1020_MIMXRT1024._wdog_status_flags.kWDOG_TimeoutResetFlag (*C enumerator*), 932
drivers_RT_RT1020_MIMXRT1024._wdog_work_mode (*C struct*), 932
drivers_RT_RT1020_MIMXRT1024._wdog_work_mode.enableDebug (*C var*), 933
drivers_RT_RT1020_MIMXRT1024._wdog_work_mode.enableStop (*C var*), 933
drivers_RT_RT1020_MIMXRT1024._wdog_work_mode.enableWait (*C var*), 933
drivers_RT_RT1020_MIMXRT1024._xbar_request (*C enum*), 935
drivers_RT_RT1020_MIMXRT1024._xbar_request.kXBARA_RequestDisable (*C enumerator*), 935
drivers_RT_RT1020_MIMXRT1024._xbar_request.kXBARA_RequestDMAEnable (*C enumerator*), 935
drivers_RT_RT1020_MIMXRT1024._xbar_request.kXBARA_RequestInterruptEnable (*C enumerator*), 935
drivers_RT_RT1020_MIMXRT1024._xbar_request.kXBARA_RequestMax (*C enumerator*), 935
drivers_RT_RT1020_MIMXRT1024._xbara_active_edge (*C enum*), 935
drivers_RT_RT1020_MIMXRT1024._xbara_active_edge.kXBARA_EdgeFalling (*C enumerator*), 935
drivers_RT_RT1020_MIMXRT1024._xbara_active_edge.kXBARA_EdgeMax (*C enumerator*), 935
drivers_RT_RT1020_MIMXRT1024._xbara_active_edge.kXBARA_EdgeNone (*C enumerator*), 935
drivers_RT_RT1020_MIMXRT1024._xbara_active_edge.kXBARA_EdgeRising (*C enumerator*), 935
drivers_RT_RT1020_MIMXRT1024._xbara_active_edge.kXBARA_EdgeRisingAndFalling (*C enumerator*), 935
drivers_RT_RT1020_MIMXRT1024._xbara_status_flag_t (*C enum*), 935
drivers_RT_RT1020_MIMXRT1024._xbara_status_flag_t.kXBARA_EdgeDetectionOut0 (*C enumerator*), 936
drivers_RT_RT1020_MIMXRT1024._xbara_status_flag_t.kXBARA_EdgeDetectionOut1 (*C enumerator*), 936
drivers_RT_RT1020_MIMXRT1024._xbara_status_flag_t.kXBARA_EdgeDetectionOut2 (*C enumerator*), 936
drivers_RT_RT1020_MIMXRT1024._xbara_status_flag_t.kXBARA_EdgeDetectionOut3 (*C enumerator*), 936
drivers_RT_RT1020_MIMXRT1024.adc_channel_config_t (*C type*), 180
drivers_RT_RT1020_MIMXRT1024.ADC_ClearStatusFlags (*C function*), 176
drivers_RT_RT1020_MIMXRT1024.adc_clock_driver_t (*C type*), 179
drivers_RT_RT1020_MIMXRT1024.adc_clock_source_t (*C type*), 179
drivers_RT_RT1020_MIMXRT1024.ADC_CLOCKS (*C macro*), 235

drivers_RT_RT1020_MIMXRT1024.adc_config_t (*C type*), 179
drivers_RT_RT1020_MIMXRT1024.ADC_Deinit (*C function*), 173
drivers_RT_RT1020_MIMXRT1024.ADC_DoAutoCalibration (*C function*), 174
drivers_RT_RT1020_MIMXRT1024.ADC_EnableDMA (*C function*), 175
drivers_RT_RT1020_MIMXRT1024.ADC_ETC_ClearDMAStatusFlags (*C function*), 183
drivers_RT_RT1020_MIMXRT1024.ADC_ETC_ClearInterruptStatusFlags (*C function*), 183
drivers_RT_RT1020_MIMXRT1024.adc_etc_config_t (*C type*), 186
drivers_RT_RT1020_MIMXRT1024.ADC_ETC_Deinit (*C function*), 182
drivers_RT_RT1020_MIMXRT1024.ADC_ETC_DisableDMA (*C function*), 183
drivers_RT_RT1020_MIMXRT1024.ADC_ETC_DMA_CTRL_TRGn_REQ_MASK (*C macro*), 186
drivers_RT_RT1020_MIMXRT1024.adc_etc_dma_mode_selection_t (*C type*), 186
drivers_RT_RT1020_MIMXRT1024.ADC_ETC_DoSoftwareReset (*C function*), 183
drivers_RT_RT1020_MIMXRT1024.ADC_ETC_DoSoftwareTrigger (*C function*), 184
drivers_RT_RT1020_MIMXRT1024.ADC_ETC_DoSoftwareTriggerBlocking (*C function*), 184
drivers_RT_RT1020_MIMXRT1024.ADC_ETC_EnableDMA (*C function*), 183
drivers_RT_RT1020_MIMXRT1024.adc_etc_external_trigger_source_t (*C type*), 186
drivers_RT_RT1020_MIMXRT1024.ADC_ETC_GetADCConversionValue (*C function*), 184
drivers_RT_RT1020_MIMXRT1024.ADC_ETC_GetDefaultConfig (*C function*), 182
drivers_RT_RT1020_MIMXRT1024.ADC_ETC_GetDMAStatusFlags (*C function*), 183
drivers_RT_RT1020_MIMXRT1024.ADC_ETC_GetInterruptStatusFlags (*C function*), 183
drivers_RT_RT1020_MIMXRT1024.ADC_ETC_Init (*C function*), 182
drivers_RT_RT1020_MIMXRT1024.adc_etc_interrupt_enable_t (*C type*), 186
drivers_RT_RT1020_MIMXRT1024.ADC_ETC_SetTriggerChainConfig (*C function*), 182
drivers_RT_RT1020_MIMXRT1024.ADC_ETC_SetTriggerConfig (*C function*), 182
drivers_RT_RT1020_MIMXRT1024.adc_etc_trigger_chain_config_t (*C type*), 186
drivers_RT_RT1020_MIMXRT1024.adc_etc_trigger_config_t (*C type*), 186
drivers_RT_RT1020_MIMXRT1024.ADC_GetChannelConversionValue (*C function*), 174
drivers_RT_RT1020_MIMXRT1024.ADC_GetChannelStatusFlags (*C function*), 174
drivers_RT_RT1020_MIMXRT1024.ADC_GetDefaultConfig (*C function*), 173
drivers_RT_RT1020_MIMXRT1024.ADC_GetStatusFlags (*C function*), 176
drivers_RT_RT1020_MIMXRT1024.adc_hardware_average_mode_t (*C type*), 179
drivers_RT_RT1020_MIMXRT1024.adc_hardware_compare_config_t (*C type*), 179
drivers_RT_RT1020_MIMXRT1024.adc_hardware_compare_mode_t (*C type*), 179
drivers_RT_RT1020_MIMXRT1024.ADC_Init (*C function*), 173
drivers_RT_RT1020_MIMXRT1024.adc_offest_config_t (*C type*), 179
drivers_RT_RT1020_MIMXRT1024.adc_reference_voltage_source_t (*C type*), 179
drivers_RT_RT1020_MIMXRT1024.adc_resolution_t (*C type*), 179
drivers_RT_RT1020_MIMXRT1024.adc_sample_period_mode_t (*C type*), 179
drivers_RT_RT1020_MIMXRT1024.ADC_SetChannelConfig (*C function*), 173
drivers_RT_RT1020_MIMXRT1024.ADC_SetHardwareAverageConfig (*C function*), 175
drivers_RT_RT1020_MIMXRT1024.ADC_SetHardwareCompareConfig (*C function*), 175
drivers_RT_RT1020_MIMXRT1024.ADC_SetOffsetConfig (*C function*), 175
drivers_RT_RT1020_MIMXRT1024.adc_status_flags_t (*C type*), 179
drivers_RT_RT1020_MIMXRT1024.aipstz_master_privilege_level_t (*C type*), 189
drivers_RT_RT1020_MIMXRT1024.aipstz_master_t (*C type*), 189
drivers_RT_RT1020_MIMXRT1024.aipstz_peripheral_access_control_t (*C type*), 189
drivers_RT_RT1020_MIMXRT1024.aipstz_peripheral_t (*C type*), 189
drivers_RT_RT1020_MIMXRT1024.AIPSTZ_SetMasterPriviledgeLevel (*C function*), 186
drivers_RT_RT1020_MIMXRT1024.AIPSTZ_SetPeripheralAccessControl (*C function*), 186
drivers_RT_RT1020_MIMXRT1024.AOI (*C macro*), 192
drivers_RT_RT1020_MIMXRT1024.AOI_CLOCKS (*C macro*), 235
drivers_RT_RT1020_MIMXRT1024.AOI_Deinit (*C function*), 190
drivers_RT_RT1020_MIMXRT1024.aoi_event_config_t (*C type*), 192
drivers_RT_RT1020_MIMXRT1024.aoi_event_t (*C type*), 192
drivers_RT_RT1020_MIMXRT1024.AOI_GetEventLogicConfig (*C function*), 190
drivers_RT_RT1020_MIMXRT1024.AOI_Init (*C function*), 190
drivers_RT_RT1020_MIMXRT1024.aoi_input_config_t (*C type*), 192

drivers_RT_RT1020_MIMXRT1024.AOI_SetEventLogicConfig (*C function*), 190
drivers_RT_RT1020_MIMXRT1024.ARRAY_SIZE (*C macro*), 624
drivers_RT_RT1020_MIMXRT1024.AT_CACHE_LINE_SECTION (*C macro*), 611
drivers_RT_RT1020_MIMXRT1024.AT_CACHE_LINE_SECTION_INIT (*C macro*), 611
drivers_RT_RT1020_MIMXRT1024.AT_NONCACHEABLE_SECTION (*C macro*), 611
drivers_RT_RT1020_MIMXRT1024.AT_NONCACHEABLE_SECTION_ALIGN (*C macro*), 611
drivers_RT_RT1020_MIMXRT1024.AT_NONCACHEABLE_SECTION_ALIGN_INIT (*C macro*), 611
drivers_RT_RT1020_MIMXRT1024.AT_NONCACHEABLE_SECTION_INIT (*C macro*), 611
drivers_RT_RT1020_MIMXRT1024.AT_QUICKACCESS_SECTION_CODE (*C macro*), 612
drivers_RT_RT1020_MIMXRT1024.AT_QUICKACCESS_SECTION_DATA (*C macro*), 612
drivers_RT_RT1020_MIMXRT1024.AT_QUICKACCESS_SECTION_DATA_ALIGN (*C macro*), 612
drivers_RT_RT1020_MIMXRT1024.bee_ac_prot_enable (*C type*), 196
drivers_RT_RT1020_MIMXRT1024.bee_aes_mode_t (*C type*), 195
drivers_RT_RT1020_MIMXRT1024.BEE_ClearStatusFlags (*C function*), 198
drivers_RT_RT1020_MIMXRT1024.BEE_CLOCKS (*C macro*), 235
drivers_RT_RT1020_MIMXRT1024.BEE_Deinit (*C function*), 196
drivers_RT_RT1020_MIMXRT1024.BEE_Disable (*C function*), 196
drivers_RT_RT1020_MIMXRT1024.BEE_Enable (*C function*), 196
drivers_RT_RT1020_MIMXRT1024.bee_endian_swap_enable (*C type*), 196
drivers_RT_RT1020_MIMXRT1024.BEE_GetDefaultConfig (*C function*), 196
drivers_RT_RT1020_MIMXRT1024.BEE_GetStatusFlags (*C function*), 198
drivers_RT_RT1020_MIMXRT1024.BEE_Init (*C function*), 196
drivers_RT_RT1020_MIMXRT1024.bee_region_config_t (*C type*), 196
drivers_RT_RT1020_MIMXRT1024.bee_region_t (*C type*), 196
drivers_RT_RT1020_MIMXRT1024.bee_security_level (*C type*), 196
drivers_RT_RT1020_MIMXRT1024.BEE_SetConfig (*C function*), 197
drivers_RT_RT1020_MIMXRT1024.BEE_SetRegionKey (*C function*), 197
drivers_RT_RT1020_MIMXRT1024.BEE_SetRegionNonce (*C function*), 197
drivers_RT_RT1020_MIMXRT1024.bee_status_flags_t (*C type*), 196
drivers_RT_RT1020_MIMXRT1024.CACHE_LINE_DATA (*C macro*), 612
drivers_RT_RT1020_MIMXRT1024.CACRR_OFFSET (*C macro*), 234
drivers_RT_RT1020_MIMXRT1024.CAN_CLOCK_CHECK_NO_AFFECTS (*C macro*), 235
drivers_RT_RT1020_MIMXRT1024.CBCDR_OFFSET (*C macro*), 234
drivers_RT_RT1020_MIMXRT1024.CBCMR_OFFSET (*C macro*), 234
drivers_RT_RT1020_MIMXRT1024.CCM_ANALOG_PLL_BYPASS_CLK_SRC_MASK (*C macro*), 234
drivers_RT_RT1020_MIMXRT1024.CCM_ANALOG_PLL_BYPASS_CLK_SRC_SHIFT (*C macro*), 234
drivers_RT_RT1020_MIMXRT1024.CCM_ANALOG_PLL_BYPASS_SHIFT (*C macro*), 233
drivers_RT_RT1020_MIMXRT1024.CCM_ANALOG_TUPLE (*C macro*), 235
drivers_RT_RT1020_MIMXRT1024.CCM_ANALOG_TUPLE_REG (*C macro*), 235
drivers_RT_RT1020_MIMXRT1024.CCM_ANALOG_TUPLE_REG_OFF (*C macro*), 235
drivers_RT_RT1020_MIMXRT1024.CCM_ANALOG_TUPLE_SHIFT (*C macro*), 235
drivers_RT_RT1020_MIMXRT1024.CCM_NO_BUSY_WAIT (*C macro*), 235
drivers_RT_RT1020_MIMXRT1024.CCM_TUPLE (*C macro*), 234
drivers_RT_RT1020_MIMXRT1024.CCM_TUPLE_BUSY_SHIFT (*C macro*), 235
drivers_RT_RT1020_MIMXRT1024.CCM_TUPLE_MASK (*C macro*), 234
drivers_RT_RT1020_MIMXRT1024.CCM_TUPLE_REG (*C macro*), 234
drivers_RT_RT1020_MIMXRT1024.CCM_TUPLE_SHIFT (*C macro*), 234
drivers_RT_RT1020_MIMXRT1024.CCSR_OFFSET (*C macro*), 234
drivers_RT_RT1020_MIMXRT1024.CDCDR_OFFSET (*C macro*), 234
drivers_RT_RT1020_MIMXRT1024.CLKPN_FREQ (*C macro*), 235
drivers_RT_RT1020_MIMXRT1024.clock_audio_pll_config_t (*C type*), 230
drivers_RT_RT1020_MIMXRT1024.CLOCK_ControlGate (*C function*), 232
drivers_RT_RT1020_MIMXRT1024.CLOCK_DeinitAudioPll (*C function*), 240
drivers_RT_RT1020_MIMXRT1024.CLOCK_DeinitEnetPll (*C function*), 240

drivers_RT_RT1020_MIMXRT1024.CLOCK_DeinitExternalClk (*C function*), 238
drivers_RT_RT1020_MIMXRT1024.CLOCK_DeinitRcOsc24M (*C function*), 238
drivers_RT_RT1020_MIMXRT1024.CLOCK_DeinitSysPfd (*C function*), 241
drivers_RT_RT1020_MIMXRT1024.CLOCK_DeinitSysPll (*C function*), 240
drivers_RT_RT1020_MIMXRT1024.CLOCK_DeinitUsb1Pfd (*C function*), 242
drivers_RT_RT1020_MIMXRT1024.CLOCK_DeinitUsb1Pll (*C function*), 240
drivers_RT_RT1020_MIMXRT1024.CLOCK_DisableClock (*C function*), 232
drivers_RT_RT1020_MIMXRT1024.CLOCK_DisableUsbhs0PhyPllClock (*C function*), 242
drivers_RT_RT1020_MIMXRT1024.clock_div_t (*C type*), 230
drivers_RT_RT1020_MIMXRT1024.clock_div_value_t (*C type*), 230
drivers_RT_RT1020_MIMXRT1024.CLOCK_EnableClock (*C function*), 232
drivers_RT_RT1020_MIMXRT1024.CLOCK_EnableUsbhs0Clock (*C function*), 238
drivers_RT_RT1020_MIMXRT1024.CLOCK_EnableUsbhs0PhyPllClock (*C function*), 242
drivers_RT_RT1020_MIMXRT1024.clock_enet_pll_config_t (*C type*), 231
drivers_RT_RT1020_MIMXRT1024.clock_gate_value_t (*C type*), 230
drivers_RT_RT1020_MIMXRT1024.CLOCK_GetAhbFreq (*C function*), 233
drivers_RT_RT1020_MIMXRT1024.CLOCK_GetClockOutCLKO1Freq (*C function*), 243
drivers_RT_RT1020_MIMXRT1024.CLOCK_GetClockOutClkO2Freq (*C function*), 243
drivers_RT_RT1020_MIMXRT1024.CLOCK_GetClockRootFreq (*C function*), 233
drivers_RT_RT1020_MIMXRT1024.CLOCK_GetCoreSysClkFreq (*C macro*), 237
drivers_RT_RT1020_MIMXRT1024.CLOCK_GetCpuClkFreq (*C function*), 233
drivers_RT_RT1020_MIMXRT1024.CLOCK_GetDiv (*C function*), 232
drivers_RT_RT1020_MIMXRT1024.CLOCK_GetFreq (*C function*), 233
drivers_RT_RT1020_MIMXRT1024.CLOCK_GetIpgFreq (*C function*), 233
drivers_RT_RT1020_MIMXRT1024.CLOCK_GetMux (*C function*), 231
drivers_RT_RT1020_MIMXRT1024.CLOCK_GetOscFreq (*C function*), 232
drivers_RT_RT1020_MIMXRT1024.CLOCK_GetPerClkFreq (*C function*), 233
drivers_RT_RT1020_MIMXRT1024.CLOCK_GetPllBypassRefClk (*C function*), 240
drivers_RT_RT1020_MIMXRT1024.CLOCK_GetPllFreq (*C function*), 241
drivers_RT_RT1020_MIMXRT1024.CLOCK_GetRtcFreq (*C function*), 238
drivers_RT_RT1020_MIMXRT1024.CLOCK_GetSemcFreq (*C function*), 233
drivers_RT_RT1020_MIMXRT1024.CLOCK_GetSysPfdFreq (*C function*), 242
drivers_RT_RT1020_MIMXRT1024.CLOCK_GetUsb1PfdFreq (*C function*), 242
drivers_RT_RT1020_MIMXRT1024.CLOCK_InitAudioPll (*C function*), 240
drivers_RT_RT1020_MIMXRT1024.CLOCK_InitEnetPll (*C function*), 240
drivers_RT_RT1020_MIMXRT1024.CLOCK_InitExternalClk (*C function*), 238
drivers_RT_RT1020_MIMXRT1024.CLOCK_InitRcOsc24M (*C function*), 238
drivers_RT_RT1020_MIMXRT1024.CLOCK_InitSysPfd (*C function*), 241
drivers_RT_RT1020_MIMXRT1024.CLOCK_InitSysPll (*C function*), 240
drivers_RT_RT1020_MIMXRT1024.CLOCK_InitUsb1Pfd (*C function*), 241
drivers_RT_RT1020_MIMXRT1024.CLOCK_InitUsb1Pll (*C function*), 240
drivers_RT_RT1020_MIMXRT1024.clock_ip_name_t (*C type*), 230
drivers_RT_RT1020_MIMXRT1024.CLOCK_IsPllBypassed (*C function*), 239
drivers_RT_RT1020_MIMXRT1024.CLOCK_IsPllEnabled (*C function*), 239
drivers_RT_RT1020_MIMXRT1024.CLOCK_IsSysPfdEnabled (*C function*), 241
drivers_RT_RT1020_MIMXRT1024.CLOCK_IsUsb1PfdEnabled (*C function*), 242
drivers_RT_RT1020_MIMXRT1024.clock_mode_t (*C type*), 230
drivers_RT_RT1020_MIMXRT1024.clock_mux_t (*C type*), 230
drivers_RT_RT1020_MIMXRT1024.clock_name_t (*C type*), 230
drivers_RT_RT1020_MIMXRT1024.clock_osc_t (*C type*), 230
drivers_RT_RT1020_MIMXRT1024.clock_output1_selection_t (*C type*), 231
drivers_RT_RT1020_MIMXRT1024.clock_output2_selection_t (*C type*), 231
drivers_RT_RT1020_MIMXRT1024.clock_output_divider_t (*C type*), 231
drivers_RT_RT1020_MIMXRT1024.clock_pfd_t (*C type*), 231
drivers_RT_RT1020_MIMXRT1024.clock_pll_t (*C type*), 231
drivers_RT_RT1020_MIMXRT1024.CLOCK_ROOT_DIV_TUPLE (*C macro*), 237
drivers_RT_RT1020_MIMXRT1024.CLOCK_ROOT_MUX_TUPLE (*C macro*), 237
drivers_RT_RT1020_MIMXRT1024.CLOCK_ROOT_NONE_PRE_DIV (*C macro*), 237

drivers_RT_RT1020_MIMXRT1024.CLOCK_ROOT_SOUCE (*C macro*), 237
drivers_RT_RT1020_MIMXRT1024.clock_root_t (*C type*), 231
drivers_RT_RT1020_MIMXRT1024.CLOCK_SetClockOutput1 (*C function*), 242
drivers_RT_RT1020_MIMXRT1024.CLOCK_SetClockOutput2 (*C function*), 243
drivers_RT_RT1020_MIMXRT1024.CLOCK_SetDiv (*C function*), 232
drivers_RT_RT1020_MIMXRT1024.CLOCK_SetMode (*C function*), 232
drivers_RT_RT1020_MIMXRT1024.CLOCK_SetMux (*C function*), 231
drivers_RT_RT1020_MIMXRT1024.CLOCK_SetPllBypass (*C function*), 239
drivers_RT_RT1020_MIMXRT1024.CLOCK_SetPllBypassRefClkSrc (*C function*), 239
drivers_RT_RT1020_MIMXRT1024.CLOCK_SetRtcXtalFreq (*C function*), 238
drivers_RT_RT1020_MIMXRT1024.CLOCK_SetXtal0Freq (*C macro*), 235
drivers_RT_RT1020_MIMXRT1024.CLOCK_SetXtal32Freq (*C macro*), 235
drivers_RT_RT1020_MIMXRT1024.CLOCK_SetXtalFreq (*C function*), 238
drivers_RT_RT1020_MIMXRT1024.CLOCK_SOURCE_NONE (*C macro*), 237
drivers_RT_RT1020_MIMXRT1024.CLOCK_SwitchOsc (*C function*), 238
drivers_RT_RT1020_MIMXRT1024.clock_sys_pll_config_t (*C type*), 230
drivers_RT_RT1020_MIMXRT1024.clock_usb_phy_src_t (*C type*), 230
drivers_RT_RT1020_MIMXRT1024.clock_usb_pll_config_t (*C type*), 230
drivers_RT_RT1020_MIMXRT1024.clock_usb_src_t (*C type*), 230
drivers_RT_RT1020_MIMXRT1024.CMP_ClearStatusFlags (*C function*), 247
drivers_RT_RT1020_MIMXRT1024.CMP_CLOCKS (*C macro*), 235
drivers_RT_RT1020_MIMXRT1024.cmp_config_t (*C type*), 249
drivers_RT_RT1020_MIMXRT1024.cmp_dac_config_t (*C type*), 249
drivers_RT_RT1020_MIMXRT1024.CMP_Deinit (*C function*), 245
drivers_RT_RT1020_MIMXRT1024.CMP_DisableInterrupts (*C function*), 247
drivers_RT_RT1020_MIMXRT1024.CMP_Enable (*C function*), 245
drivers_RT_RT1020_MIMXRT1024.CMP_EnableDMA (*C function*), 246
drivers_RT_RT1020_MIMXRT1024.CMP_EnableInterrupts (*C function*), 247
drivers_RT_RT1020_MIMXRT1024.CMP_EnablePassThroughMode (*C function*), 246
drivers_RT_RT1020_MIMXRT1024.CMP_EnableWindowMode (*C function*), 246
drivers_RT_RT1020_MIMXRT1024.cmp_filter_config_t (*C type*), 249
drivers_RT_RT1020_MIMXRT1024.CMP_GetDefaultConfig (*C function*), 246
drivers_RT_RT1020_MIMXRT1024.CMP_GetStatusFlags (*C function*), 247
drivers_RT_RT1020_MIMXRT1024.cmp_hysteresis_mode_t (*C type*), 248
drivers_RT_RT1020_MIMXRT1024.CMP_Init (*C function*), 245
drivers_RT_RT1020_MIMXRT1024.cmp_reference_voltage_source_t (*C type*), 249
drivers_RT_RT1020_MIMXRT1024.CMP_SetDACConfig (*C function*), 247
drivers_RT_RT1020_MIMXRT1024.CMP_SetFilterConfig (*C function*), 247
drivers_RT_RT1020_MIMXRT1024.CMP_SetInputChannels (*C function*), 246
drivers_RT_RT1020_MIMXRT1024.COUNT_TO_MSEC (*C macro*), 611
drivers_RT_RT1020_MIMXRT1024.COUNT_TO_USEC (*C macro*), 611
drivers_RT_RT1020_MIMXRT1024.CS1CDR_OFFSET (*C macro*), 234
drivers_RT_RT1020_MIMXRT1024.CS2CDR_OFFSET (*C macro*), 234
drivers_RT_RT1020_MIMXRT1024.CSCDR1_OFFSET (*C macro*), 234
drivers_RT_RT1020_MIMXRT1024.CSCDR2_OFFSET (*C macro*), 234
drivers_RT_RT1020_MIMXRT1024.CSCMR1_OFFSET (*C macro*), 234
drivers_RT_RT1020_MIMXRT1024.CSCMR2_OFFSET (*C macro*), 234
drivers_RT_RT1020_MIMXRT1024.DCACHE_CleanByRange (*C function*), 201
drivers_RT_RT1020_MIMXRT1024.DCACHE_CleanInvalidateByRange (*C function*), 201
drivers_RT_RT1020_MIMXRT1024.DCACHE_InvalidateByRange (*C function*), 200
drivers_RT_RT1020_MIMXRT1024.DCDC_AdjustLowPowerTargetVoltage (*C function*), 253
drivers_RT_RT1020_MIMXRT1024.DCDC_AdjustRunTargetVoltage (*C function*), 253
drivers_RT_RT1020_MIMXRT1024.DCDC_AdjustTargetVoltage (*C function*), 253
drivers_RT_RT1020_MIMXRT1024.DCDC_BootIntoCCM (*C function*), 254
drivers_RT_RT1020_MIMXRT1024.DCDC_BootIntoDCM (*C function*), 254
drivers_RT_RT1020_MIMXRT1024.dcdc_clock_source_t (*C type*), 257
drivers_RT_RT1020_MIMXRT1024.DCDC_CLOCKS (*C macro*), 235
drivers_RT_RT1020_MIMXRT1024.dcdc_comparator_current_bias_t (*C type*), 257

drivers_RT_RT1020_MIMXRT1024.dcdc_config_t (*C type*), 258
drivers_RT_RT1020_MIMXRT1024.dcdc_control_mode_t (*C type*), 258
drivers_RT_RT1020_MIMXRT1024.dcdc_count_charging_time_period_t (*C type*), 257
drivers_RT_RT1020_MIMXRT1024.dcdc_count_charging_time_threshold_t (*C type*), 257
drivers_RT_RT1020_MIMXRT1024.DCDC_Deinit (*C function*), 250
drivers_RT_RT1020_MIMXRT1024.dcdc_detection_config_t (*C type*), 258
drivers_RT_RT1020_MIMXRT1024.DCDC_EnableImproveTransition (*C function*), 254
drivers_RT_RT1020_MIMXRT1024.DCDC_EnableOutputRangeComparator (*C function*), 251
drivers_RT_RT1020_MIMXRT1024.DCDC_EnterLowPowerMode (*C function*), 251
drivers_RT_RT1020_MIMXRT1024.DCDC_GetDefaultConfig (*C function*), 250
drivers_RT_RT1020_MIMXRT1024.DCDC_GetDefaultDetectionConfig (*C function*), 251
drivers_RT_RT1020_MIMXRT1024.DCDC_GetDefaultLoopControlConfig (*C function*), 252
drivers_RT_RT1020_MIMXRT1024.DCDC_GetDefaultLowPowerConfig (*C function*), 251
drivers_RT_RT1020_MIMXRT1024.DCDC_GetstatusFlags (*C function*), 250
drivers_RT_RT1020_MIMXRT1024.DCDC_Init (*C function*), 250
drivers_RT_RT1020_MIMXRT1024.dcdc_internal_regulator_config_t (*C type*), 258
drivers_RT_RT1020_MIMXRT1024.DCDC_LockTargetVoltage (*C function*), 253
drivers_RT_RT1020_MIMXRT1024.dcdc_loop_control_config_t (*C type*), 258
drivers_RT_RT1020_MIMXRT1024.dcdc_low_power_config_t (*C type*), 258
drivers_RT_RT1020_MIMXRT1024.dcdc_low_power_mode_t (*C type*), 257
drivers_RT_RT1020_MIMXRT1024.dcdc_min_power_config_t (*C type*), 258
drivers_RT_RT1020_MIMXRT1024.dcdc_over_current_threshold_t (*C type*), 257
drivers_RT_RT1020_MIMXRT1024.dcdc_peak_current_threshold_t (*C type*), 257
drivers_RT_RT1020_MIMXRT1024.DCDC_ResetCurrentAlertSignal (*C function*), 252
drivers_RT_RT1020_MIMXRT1024.DCDC_SetBandgapVoltageTrimValue (*C function*), 252
drivers_RT_RT1020_MIMXRT1024.DCDC_SetClockSource (*C function*), 251
drivers_RT_RT1020_MIMXRT1024.DCDC_SetDetectionConfig (*C function*), 251
drivers_RT_RT1020_MIMXRT1024.DCDC_SetInternalRegulatorConfig (*C function*), 254
drivers_RT_RT1020_MIMXRT1024.DCDC_SetLoopControlConfig (*C function*), 252
drivers_RT_RT1020_MIMXRT1024.DCDC_SetLowPowerConfig (*C function*), 252
drivers_RT_RT1020_MIMXRT1024.DCDC_SetLPComparatorBiasValue (*C function*), 253
drivers_RT_RT1020_MIMXRT1024.DCDC_SetMinPowerConfig (*C function*), 253
drivers_RT_RT1020_MIMXRT1024.dcdc_trim_input_mode_t (*C type*), 258
drivers_RT_RT1020_MIMXRT1024.DCP_AES_BLOCK_SIZE (*C macro*), 268
drivers_RT_RT1020_MIMXRT1024.DCP_AES_DecryptCbc (*C function*), 267
drivers_RT_RT1020_MIMXRT1024.DCP_AES_DecryptCbcNonBlocking (*C function*), 271
drivers_RT_RT1020_MIMXRT1024.DCP_AES_DecryptEcb (*C function*), 267
drivers_RT_RT1020_MIMXRT1024.DCP_AES_DecryptEcbNonBlocking (*C function*), 270
drivers_RT_RT1020_MIMXRT1024.DCP_AES_EncryptCbc (*C function*), 267
drivers_RT_RT1020_MIMXRT1024.DCP_AES_EncryptCbcNonBlocking (*C function*), 271
drivers_RT_RT1020_MIMXRT1024.DCP_AES_EncryptEcb (*C function*), 266
drivers_RT_RT1020_MIMXRT1024.DCP_AES_EncryptEcbNonBlocking (*C function*), 270
drivers_RT_RT1020_MIMXRT1024.DCP_AES_SetKey (*C function*), 266
drivers_RT_RT1020_MIMXRT1024.dcp_channel_t (*C type*), 264
drivers_RT_RT1020_MIMXRT1024.DCP_CLOCKS (*C macro*), 235
drivers_RT_RT1020_MIMXRT1024.dcp_config_t (*C type*), 264
drivers_RT_RT1020_MIMXRT1024.dcp_context_t (*C type*), 264
drivers_RT_RT1020_MIMXRT1024.DCP_Deinit (*C function*), 264
drivers_RT_RT1020_MIMXRT1024.DCP_GetDefaultConfig (*C function*), 265
drivers_RT_RT1020_MIMXRT1024.dcp_handle_t (*C type*), 264
drivers_RT_RT1020_MIMXRT1024.DCP_HASH (*C function*), 269
drivers_RT_RT1020_MIMXRT1024.dcp_hash_algo_t (*C type*), 268
drivers_RT_RT1020_MIMXRT1024.DCP_HASH_BLOCK_SIZE (*C macro*), 270
drivers_RT_RT1020_MIMXRT1024.DCP_HASH_CAVP_COMPATIBLE (*C macro*), 269
drivers_RT_RT1020_MIMXRT1024.DCP_HASH_CTX_SIZE (*C macro*), 270
drivers_RT_RT1020_MIMXRT1024.dcp_hash_ctx_t (*C type*), 268
drivers_RT_RT1020_MIMXRT1024.DCP_HASH_Finish (*C function*), 269
drivers_RT_RT1020_MIMXRT1024.DCP_HASH_Init (*C function*), 268

drivers_RT_RT1020_MIMXRT1024.DCP_HASH_Update (*C function*), 268
drivers_RT_RT1020_MIMXRT1024.DCP_Init (*C function*), 264
drivers_RT_RT1020_MIMXRT1024.dcp_key_slot_t (*C type*), 264
drivers_RT_RT1020_MIMXRT1024.DCP_SHA_BLOCK_SIZE (*C macro*), 270
drivers_RT_RT1020_MIMXRT1024.dcp_swap_t (*C type*), 264
drivers_RT_RT1020_MIMXRT1024.DCP_WaitForChannelComplete (*C function*), 265
drivers_RT_RT1020_MIMXRT1024.dcp_work_packet_t (*C type*), 264
drivers_RT_RT1020_MIMXRT1024.DEBUG_CONSOLE_DEVICE_TYPE_FLEXCOMM (*C macro*), 608
drivers_RT_RT1020_MIMXRT1024.DEBUG_CONSOLE_DEVICE_TYPE_IUART (*C macro*), 609
drivers_RT_RT1020_MIMXRT1024.DEBUG_CONSOLE_DEVICE_TYPE_LPSCI (*C macro*), 608
drivers_RT_RT1020_MIMXRT1024.DEBUG_CONSOLE_DEVICE_TYPE_LPUART (*C macro*), 608
drivers_RT_RT1020_MIMXRT1024.DEBUG_CONSOLE_DEVICE_TYPE_MINI_USART (*C macro*), 609
drivers_RT_RT1020_MIMXRT1024.DEBUG_CONSOLE_DEVICE_TYPE_NONE (*C macro*), 608
drivers_RT_RT1020_MIMXRT1024.DEBUG_CONSOLE_DEVICE_TYPE_QSCI (*C macro*), 609
drivers_RT_RT1020_MIMXRT1024.DEBUG_CONSOLE_DEVICE_TYPE_SWO (*C macro*), 609
drivers_RT_RT1020_MIMXRT1024.DEBUG_CONSOLE_DEVICE_TYPE_UART (*C macro*), 608
drivers_RT_RT1020_MIMXRT1024.DEBUG_CONSOLE_DEVICE_TYPE_USBCDC (*C macro*), 608
drivers_RT_RT1020_MIMXRT1024.DEBUG_CONSOLE_DEVICE_TYPE_VUSART (*C macro*), 609
drivers_RT_RT1020_MIMXRT1024.DisableGlobalIRQ (*C function*), 623
drivers_RT_RT1020_MIMXRT1024.DisableIRQ (*C function*), 622
drivers_RT_RT1020_MIMXRT1024.DLC_LENGTH_DECODE (*C macro*), 378
drivers_RT_RT1020_MIMXRT1024.DMA_DCHPRI_INDEX (*C macro*), 291
drivers_RT_RT1020_MIMXRT1024.DMA_MAX_TRANSFER_COUNT (*C macro*), 680
drivers_RT_RT1020_MIMXRT1024.DMAMUX_CHANNEL_ENDIAN_CONVERTn (*C macro*), 273
drivers_RT_RT1020_MIMXRT1024.DMAMUX_CLOCKS (*C macro*), 235
drivers_RT_RT1020_MIMXRT1024.DMAMUX_Deinit (*C function*), 272
drivers_RT_RT1020_MIMXRT1024.DMAMUX_DisableChannel (*C function*), 272
drivers_RT_RT1020_MIMXRT1024.DMAMUX_DisablePeriodTrigger (*C function*), 272
drivers_RT_RT1020_MIMXRT1024.DMAMUX_EnableAlwaysOn (*C function*), 273
drivers_RT_RT1020_MIMXRT1024.DMAMUX_EnableChannel (*C function*), 272
drivers_RT_RT1020_MIMXRT1024.DMAMUX_EnablePeriodTrigger (*C function*), 272
drivers_RT_RT1020_MIMXRT1024.DMAMUX_Init (*C function*), 272
drivers_RT_RT1020_MIMXRT1024.DMAMUX_SetSource (*C function*), 272
drivers_RT_RT1020_MIMXRT1024.EDMA_AbortTransfer (*C function*), 284
drivers_RT_RT1020_MIMXRT1024.edma_bandwidth_t (*C type*), 290
drivers_RT_RT1020_MIMXRT1024.edma_callback (*C type*), 291
drivers_RT_RT1020_MIMXRT1024.edma_channel_link_type_t (*C type*), 290
drivers_RT_RT1020_MIMXRT1024.edma_channel_Preemption_config_t (*C type*), 290
drivers_RT_RT1020_MIMXRT1024.EDMA_ClearChannelStatusFlags (*C function*), 281
drivers_RT_RT1020_MIMXRT1024.EDMA_CLOCKS (*C macro*), 235
drivers_RT_RT1020_MIMXRT1024.edma_config_t (*C type*), 290
drivers_RT_RT1020_MIMXRT1024.EDMA_CreateHandle (*C function*), 281
drivers_RT_RT1020_MIMXRT1024.EDMA_Deinit (*C function*), 273
drivers_RT_RT1020_MIMXRT1024.EDMA_DisableChannelInterrupts (*C function*), 277
drivers_RT_RT1020_MIMXRT1024.EDMA_DisableChannelRequest (*C function*), 280
drivers_RT_RT1020_MIMXRT1024.EDMA_DriverIRQHandler (*C function*), 285
drivers_RT_RT1020_MIMXRT1024.EDMA_EnableAsyncRequest (*C function*), 277
drivers_RT_RT1020_MIMXRT1024.EDMA_EnableAutoStopRequest (*C function*), 277
drivers_RT_RT1020_MIMXRT1024.EDMA_EnableChannelInterrupts (*C function*), 277
drivers_RT_RT1020_MIMXRT1024.EDMA_EnableChannelRequest (*C function*), 280
drivers_RT_RT1020_MIMXRT1024.EDMA_EnableContinuousChannelLinkMode (*C function*), 274
drivers_RT_RT1020_MIMXRT1024.EDMA_EnableMinorLoopMapping (*C function*), 274
drivers_RT_RT1020_MIMXRT1024.EDMA_GetChannelStatusFlags (*C function*), 281

drivers_RT_RT1020_MIMXRT1024.EDMA_GetDefaultConfig (*C function*), 274
drivers_RT_RT1020_MIMXRT1024.EDMA_GetErrorStatusFlags (*C function*), 281
drivers_RT_RT1020_MIMXRT1024.EDMA_GetNextTCDAddress (*C function*), 284
drivers_RT_RT1020_MIMXRT1024.EDMA_GetRemainingMajorLoopCount (*C function*), 280
drivers_RT_RT1020_MIMXRT1024.EDMA_GetUnusedTCDNumber (*C function*), 284
drivers_RT_RT1020_MIMXRT1024.edma_handle_t (*C type*), 291
drivers_RT_RT1020_MIMXRT1024.EDMA_HandleIRQ (*C function*), 284
drivers_RT_RT1020_MIMXRT1024.EDMA_Init (*C function*), 273
drivers_RT_RT1020_MIMXRT1024.EDMA_InstallTCD (*C function*), 273
drivers_RT_RT1020_MIMXRT1024.EDMA_InstallTCDMemory (*C function*), 282
drivers_RT_RT1020_MIMXRT1024.edma_interrupt_enable_t (*C type*), 290
drivers_RT_RT1020_MIMXRT1024.edma_minor_offset_config_t (*C type*), 291
drivers_RT_RT1020_MIMXRT1024.edma_modulo_t (*C type*), 290
drivers_RT_RT1020_MIMXRT1024.EDMA_PrepareTransfer (*C function*), 283
drivers_RT_RT1020_MIMXRT1024.EDMA_PrepareTransferConfig (*C function*), 282
drivers_RT_RT1020_MIMXRT1024.EDMA_ResetChannel (*C function*), 274
drivers_RT_RT1020_MIMXRT1024.EDMA_SetBandWidth (*C function*), 276
drivers_RT_RT1020_MIMXRT1024.EDMA_SetCallback (*C function*), 282
drivers_RT_RT1020_MIMXRT1024.EDMA_SetChannelLink (*C function*), 276
drivers_RT_RT1020_MIMXRT1024.EDMA_SetChannelPreemptionConfig (*C function*), 275
drivers_RT_RT1020_MIMXRT1024.EDMA_SetMajorOffsetConfig (*C function*), 277
drivers_RT_RT1020_MIMXRT1024.EDMA_SetMinorOffsetConfig (*C function*), 275
drivers_RT_RT1020_MIMXRT1024.EDMA_SetModulo (*C function*), 276
drivers_RT_RT1020_MIMXRT1024.EDMA_SetTransferConfig (*C function*), 275
drivers_RT_RT1020_MIMXRT1024.EDMA_StartTransfer (*C function*), 283
drivers_RT_RT1020_MIMXRT1024.EDMA_StopTransfer (*C function*), 284
drivers_RT_RT1020_MIMXRT1024.EDMA_SubmitTransfer (*C function*), 283
drivers_RT_RT1020_MIMXRT1024.edma_tcd_t (*C type*), 291
drivers_RT_RT1020_MIMXRT1024.EDMA_TcdDisableInterrupts (*C function*), 280
drivers_RT_RT1020_MIMXRT1024.EDMA_TcdEnableAutoStopRequest (*C function*), 279
drivers_RT_RT1020_MIMXRT1024.EDMA_TcdEnableInterrupts (*C function*), 280
drivers_RT_RT1020_MIMXRT1024.EDMA_TcdReset (*C function*), 277
drivers_RT_RT1020_MIMXRT1024.EDMA_TcdSetBandWidth (*C function*), 279
drivers_RT_RT1020_MIMXRT1024.EDMA_TcdSetChannelLink (*C function*), 278
drivers_RT_RT1020_MIMXRT1024.EDMA_TcdSetMajorOffsetConfig (*C function*), 280
drivers_RT_RT1020_MIMXRT1024.EDMA_TcdSetMinorOffsetConfig (*C function*), 278
drivers_RT_RT1020_MIMXRT1024.EDMA_TcdSetModulo (*C function*), 279
drivers_RT_RT1020_MIMXRT1024.EDMA_TcdSetTransferConfig (*C function*), 278
drivers_RT_RT1020_MIMXRT1024.edma_transfer_config_t (*C type*), 290
drivers_RT_RT1020_MIMXRT1024.edma_transfer_size_t (*C type*), 290
drivers_RT_RT1020_MIMXRT1024.edma_transfer_type_t (*C type*), 290
drivers_RT_RT1020_MIMXRT1024.EDMA_TriggerChannelStart (*C function*), 280
drivers_RT_RT1020_MIMXRT1024.EnableGlobalIRQ (*C function*), 624
drivers_RT_RT1020_MIMXRT1024.EnableIRQ (*C function*), 622
drivers_RT_RT1020_MIMXRT1024.EnableIRQWithPriority (*C function*), 622
drivers_RT_RT1020_MIMXRT1024.ENC_ClearStatusFlags (*C function*), 296
drivers_RT_RT1020_MIMXRT1024.ENC_CLOCKS (*C macro*), 235
drivers_RT_RT1020_MIMXRT1024.enc_config_t (*C type*), 304
drivers_RT_RT1020_MIMXRT1024.enc_decoder_work_mode_t (*C type*), 304
drivers_RT_RT1020_MIMXRT1024.ENC_Deinit (*C function*), 295
drivers_RT_RT1020_MIMXRT1024.ENC_DisableInterrupts (*C function*), 297
drivers_RT_RT1020_MIMXRT1024.ENC_DoSoftwareLoadInitialPositionValue (*C function*), 296
drivers_RT_RT1020_MIMXRT1024.ENC_EnableInterrupts (*C function*), 297
drivers_RT_RT1020_MIMXRT1024.ENC_EnableWatchdog (*C function*), 296
drivers_RT_RT1020_MIMXRT1024.enc_filter_prescaler_t (*C type*), 304
drivers_RT_RT1020_MIMXRT1024.ENC_GetDefaultConfig (*C function*), 295
drivers_RT_RT1020_MIMXRT1024.ENC_GetEnabledInterrupts (*C function*), 297
drivers_RT_RT1020_MIMXRT1024.ENC_GetHoldLastEdgeTimeValue (*C function*), 298

drivers_RT_RT1020_MIMXRT1024.ENC_GetHoldPositionDifferencePeriodValue (*C function*), 299
drivers_RT_RT1020_MIMXRT1024.ENC_GetHoldPositionDifferenceValue (*C function*), 298
drivers_RT_RT1020_MIMXRT1024.ENC_GetHoldPositionValue (*C function*), 297
drivers_RT_RT1020_MIMXRT1024.ENC_GetHoldRevolutionValue (*C function*), 298
drivers_RT_RT1020_MIMXRT1024.ENC_GetLastEdgeTimeValue (*C function*), 298
drivers_RT_RT1020_MIMXRT1024.ENC_GetPositionDifferencePeriodBufferValue (*C function*), 299
drivers_RT_RT1020_MIMXRT1024.ENC_GetPositionDifferencePeriodValue (*C function*), 298
drivers_RT_RT1020_MIMXRT1024.ENC_GetPositionDifferenceValue (*C function*), 297
drivers_RT_RT1020_MIMXRT1024.ENC_GetPositionValue (*C function*), 297
drivers_RT_RT1020_MIMXRT1024.ENC_GetRevolutionValue (*C function*), 298
drivers_RT_RT1020_MIMXRT1024.ENC_GetSignalStatusFlags (*C function*), 297
drivers_RT_RT1020_MIMXRT1024.ENC_GetStatusFlags (*C function*), 296
drivers_RT_RT1020_MIMXRT1024.enc_home_trigger_mode_t (*C type*), 304
drivers_RT_RT1020_MIMXRT1024.enc_index_trigger_mode_t (*C type*), 304
drivers_RT_RT1020_MIMXRT1024.ENC_Init (*C function*), 295
drivers_RT_RT1020_MIMXRT1024.enc_position_match_mode_t (*C type*), 304
drivers_RT_RT1020_MIMXRT1024.enc_prescaler_t (*C type*), 304
drivers_RT_RT1020_MIMXRT1024.enc_revolution_count_condition_t (*C type*), 304
drivers_RT_RT1020_MIMXRT1024.enc_self_test_config_t (*C type*), 304
drivers_RT_RT1020_MIMXRT1024.enc_self_test_direction_t (*C type*), 304
drivers_RT_RT1020_MIMXRT1024.ENC_SetInitialPositionValue (*C function*), 296
drivers_RT_RT1020_MIMXRT1024.ENC_SetSelfTestConfig (*C function*), 296
drivers_RT_RT1020_MIMXRT1024.ENET_ActiveRead (*C function*), 313
drivers_RT_RT1020_MIMXRT1024.ENET_AddMulticastGroup (*C function*), 312
drivers_RT_RT1020_MIMXRT1024.enet_avb_config_t (*C type*), 330
drivers_RT_RT1020_MIMXRT1024.ENET_BUFFDESCRIPTOR_RX_BROADCAST_MASK (*C macro*), 320
drivers_RT_RT1020_MIMXRT1024.ENET_BUFFDESCRIPTOR_RX_CRC_MASK (*C macro*), 321
drivers_RT_RT1020_MIMXRT1024.ENET_BUFFDESCRIPTOR_RX_EMPTY_MASK (*C macro*), 320
drivers_RT_RT1020_MIMXRT1024.ENET_BUFFDESCRIPTOR_RX_ERR_MASK (*C macro*), 332
drivers_RT_RT1020_MIMXRT1024.ENET_BUFFDESCRIPTOR_RX_LAST_MASK (*C macro*), 320
drivers_RT_RT1020_MIMXRT1024.ENET_BUFFDESCRIPTOR_RX_LENVLIOLATE_MASK (*C macro*), 321
drivers_RT_RT1020_MIMXRT1024.ENET_BUFFDESCRIPTOR_RX_MISS_MASK (*C macro*), 320
drivers_RT_RT1020_MIMXRT1024.ENET_BUFFDESCRIPTOR_RX_MULTICAST_MASK (*C macro*), 321
drivers_RT_RT1020_MIMXRT1024.ENET_BUFFDESCRIPTOR_RX_NOOCTET_MASK (*C macro*), 321
drivers_RT_RT1020_MIMXRT1024.ENET_BUFFDESCRIPTOR_RX_OVERRUN_MASK (*C macro*), 321
drivers_RT_RT1020_MIMXRT1024.ENET_BUFFDESCRIPTOR_RX_SOFTOWNER1_MASK (*C macro*), 320
drivers_RT_RT1020_MIMXRT1024.ENET_BUFFDESCRIPTOR_RX_SOFTOWNER2_Mask (*C macro*), 320
drivers_RT_RT1020_MIMXRT1024.ENET_BUFFDESCRIPTOR_RX_TRUNC_MASK (*C macro*), 321
drivers_RT_RT1020_MIMXRT1024.ENET_BUFFDESCRIPTOR_RX_WRAP_MASK (*C macro*), 320
drivers_RT_RT1020_MIMXRT1024.ENET_BUFFDESCRIPTOR_TX_LAST_MASK (*C macro*), 321
drivers_RT_RT1020_MIMXRT1024.ENET_BUFFDESCRIPTOR_TX_READY_MASK (*C macro*), 321
drivers_RT_RT1020_MIMXRT1024.ENET_BUFFDESCRIPTOR_TX_SOFTOWENER1_MASK (*C macro*), 321
drivers_RT_RT1020_MIMXRT1024.ENET_BUFFDESCRIPTOR_TX_SOFTOWENER2_MASK

(*C macro*), 321
drivers_RT_RT1020_MIMXRT1024.ENET_BUFFDESCRIPTOR_TX_TRANMITCRC_MASK (*C macro*), 321
drivers_RT_RT1020_MIMXRT1024.ENET_BUFFDESCRIPTOR_TX_WRAP_MASK (*C macro*), 321
drivers_RT_RT1020_MIMXRT1024.enet_buffer_config_t (*C type*), 330
drivers_RT_RT1020_MIMXRT1024.enet_buffer_struct_t (*C type*), 332
drivers_RT_RT1020_MIMXRT1024.enet_callback_t (*C type*), 331
drivers_RT_RT1020_MIMXRT1024.ENET_ClearInterruptStatus (*C function*), 314
drivers_RT_RT1020_MIMXRT1024.ENET_CLOCKS (*C macro*), 235
drivers_RT_RT1020_MIMXRT1024.ENET_CommonFrame0DriverIRQHandler (*C function*), 320
drivers_RT_RT1020_MIMXRT1024.ENET_CommonFrame0IRQHandler (*C function*), 320
drivers_RT_RT1020_MIMXRT1024.enet_config_t (*C type*), 331
drivers_RT_RT1020_MIMXRT1024.enet_data_error_stats_t (*C type*), 330
drivers_RT_RT1020_MIMXRT1024.ENET_Deinit (*C function*), 308
drivers_RT_RT1020_MIMXRT1024.ENET_DisableInterrupts (*C function*), 314
drivers_RT_RT1020_MIMXRT1024.ENET_Down (*C function*), 308
drivers_RT_RT1020_MIMXRT1024.ENET_EnableInterrupts (*C function*), 313
drivers_RT_RT1020_MIMXRT1024.ENET_EnableSleepMode (*C function*), 313
drivers_RT_RT1020_MIMXRT1024.ENET_EnableStatistics (*C function*), 315
drivers_RT_RT1020_MIMXRT1024.ENET_ERR_INTERRUPT (*C macro*), 322
drivers_RT_RT1020_MIMXRT1024.ENET_ErrorIRQHandler (*C function*), 319
drivers_RT_RT1020_MIMXRT1024.enet_event_t (*C type*), 329
drivers_RT_RT1020_MIMXRT1024.ENET_FIFO_MIN_RX_FULL (*C macro*), 321
drivers_RT_RT1020_MIMXRT1024.ENET_FRAME_CRC_LEN (*C macro*), 321
drivers_RT_RT1020_MIMXRT1024.enet_frame_info (*C struct*), 334
drivers_RT_RT1020_MIMXRT1024.enet_frame_info_t (*C type*), 330
drivers_RT_RT1020_MIMXRT1024.enet_frame_info.context (*C var*), 334
drivers_RT_RT1020_MIMXRT1024.ENET_FRAME_MAX_FRAMELEN (*C macro*), 321
drivers_RT_RT1020_MIMXRT1024.ENET_FRAME_TX_LEN_LIMITATION (*C macro*), 321
drivers_RT_RT1020_MIMXRT1024.ENET_FRAME_VLAN_TAGLEN (*C macro*), 321
drivers_RT_RT1020_MIMXRT1024.ENET_GetAccelFunction (*C function*), 313
drivers_RT_RT1020_MIMXRT1024.ENET_GetDefaultConfig (*C function*), 307
drivers_RT_RT1020_MIMXRT1024.ENET_GetInstance (*C function*), 332
drivers_RT_RT1020_MIMXRT1024.ENET_GetInterruptStatus (*C function*), 314
drivers_RT_RT1020_MIMXRT1024.ENET_GetMacAddr (*C function*), 312
drivers_RT_RT1020_MIMXRT1024.ENET_GetRxErrBeforeReadFrame (*C function*), 315
drivers_RT_RT1020_MIMXRT1024.ENET_GetRxFrame (*C function*), 318
drivers_RT_RT1020_MIMXRT1024.ENET_GetRxFrameSize (*C function*), 315
drivers_RT_RT1020_MIMXRT1024.ENET_GetSMI (*C function*), 309
drivers_RT_RT1020_MIMXRT1024.ENET_GetStatistics (*C function*), 315
drivers_RT_RT1020_MIMXRT1024.enet_handle_t (*C type*), 331
drivers_RT_RT1020_MIMXRT1024.enet_idle_slope_t (*C type*), 329
drivers_RT_RT1020_MIMXRT1024.ENET_Init (*C function*), 307
drivers_RT_RT1020_MIMXRT1024.enet_intcoalesce_config_t (*C type*), 330
drivers_RT_RT1020_MIMXRT1024.enet_interrupt_enable_t (*C type*), 329
drivers_RT_RT1020_MIMXRT1024.enet_isr_t (*C type*), 332
drivers_RT_RT1020_MIMXRT1024.ENET_LeaveMulticastGroup (*C function*), 312
drivers_RT_RT1020_MIMXRT1024.ENET_MDIOC45Read (*C function*), 312
drivers_RT_RT1020_MIMXRT1024.ENET_MDIOC45Write (*C function*), 311
drivers_RT_RT1020_MIMXRT1024.ENET_MDIORead (*C function*), 310
drivers_RT_RT1020_MIMXRT1024.ENET_MDIOWrite (*C function*), 310
drivers_RT_RT1020_MIMXRT1024.enet_mii_duplex_t (*C type*), 329
drivers_RT_RT1020_MIMXRT1024.enet_mii_extend_opcode (*C type*), 329
drivers_RT_RT1020_MIMXRT1024.enet_mii_mode_t (*C type*), 328
drivers_RT_RT1020_MIMXRT1024.enet_mii_read_t (*C type*), 329
drivers_RT_RT1020_MIMXRT1024.enet_mii_speed_t (*C type*), 329
drivers_RT_RT1020_MIMXRT1024.enet_mii_write_t (*C type*), 329

drivers_RT_RT1020_MIMXRT1024.ENET_PHY_MAXADDRESS (*C macro*), 322
drivers_RT_RT1020_MIMXRT1024.ENET_Ptp1588DriverIRQHandler (*C function*), 320
drivers_RT_RT1020_MIMXRT1024.ENET_Ptp1588IRQHandler (*C function*), 320
drivers_RT_RT1020_MIMXRT1024.ENET_ReadFrame (*C function*), 316
drivers_RT_RT1020_MIMXRT1024.ENET_ReadSMIData (*C function*), 309
drivers_RT_RT1020_MIMXRT1024.ENET_ReceiveIRQHandler (*C function*), 319
drivers_RT_RT1020_MIMXRT1024.ENET_ReclaimTxDescriptor (*C function*), 318
drivers_RT_RT1020_MIMXRT1024.ENET_Reset (*C function*), 308
drivers_RT_RT1020_MIMXRT1024.ENET_ResetStatistics (*C function*), 315
drivers_RT_RT1020_MIMXRT1024.enet_rx_accelerator_t (*C type*), 329
drivers_RT_RT1020_MIMXRT1024.enet_rx_alloc_callback_t (*C type*), 330
drivers_RT_RT1020_MIMXRT1024.enet_rx_bd_ring_t (*C type*), 331
drivers_RT_RT1020_MIMXRT1024.enet_rx_bd_struct_t (*C type*), 329
drivers_RT_RT1020_MIMXRT1024.enet_rx_frame_attribute_t (*C type*), 332
drivers_RT_RT1020_MIMXRT1024.enet_rx_frame_error_t (*C type*), 330
drivers_RT_RT1020_MIMXRT1024.enet_rx_frame_struct_t (*C type*), 332
drivers_RT_RT1020_MIMXRT1024.enet_rx_free_callback_t (*C type*), 330
drivers_RT_RT1020_MIMXRT1024.ENET_RX_INTERRUPT (*C macro*), 322
drivers_RT_RT1020_MIMXRT1024.ENET_RX_MIN_BUFFERSIZE (*C macro*), 322
drivers_RT_RT1020_MIMXRT1024.ENET_SendFrame (*C function*), 317
drivers_RT_RT1020_MIMXRT1024.ENET_SetErrISRHandler (*C function*), 314
drivers_RT_RT1020_MIMXRT1024.ENET_SetMacAddr (*C function*), 312
drivers_RT_RT1020_MIMXRT1024.ENET_SetMII (*C function*), 309
drivers_RT_RT1020_MIMXRT1024.ENET_SetRGMIIClockDelay (*C function*), 312
drivers_RT_RT1020_MIMXRT1024.ENET_SetRxISRHandler (*C function*), 314
drivers_RT_RT1020_MIMXRT1024.ENET_SetSMI (*C function*), 309
drivers_RT_RT1020_MIMXRT1024.ENET_SetTxISRHandler (*C function*), 314
drivers_RT_RT1020_MIMXRT1024.ENET_SetTxReclaim (*C function*), 317
drivers_RT_RT1020_MIMXRT1024.enet_special_control_flag_t (*C type*), 329
drivers_RT_RT1020_MIMXRT1024.ENET_StartExtC45SMIReadData (*C function*), 311
drivers_RT_RT1020_MIMXRT1024.ENET_StartExtC45SMIWriteData (*C function*), 311
drivers_RT_RT1020_MIMXRT1024.ENET_StartExtC45SMIWriteReg (*C function*), 310
drivers_RT_RT1020_MIMXRT1024.ENET_StartSMIRead (*C function*), 310
drivers_RT_RT1020_MIMXRT1024.ENET_StartSMIWrite (*C function*), 309
drivers_RT_RT1020_MIMXRT1024.ENET_StartTxFrame (*C function*), 319
drivers_RT_RT1020_MIMXRT1024.enet_transfer_stats_t (*C type*), 330
drivers_RT_RT1020_MIMXRT1024.ENET_TransmitIRQHandler (*C function*), 319
drivers_RT_RT1020_MIMXRT1024.ENET_TS_INTERRUPT (*C macro*), 322
drivers_RT_RT1020_MIMXRT1024.enet_tx_accelerator_t (*C type*), 329
drivers_RT_RT1020_MIMXRT1024.enet_tx_bd_ring_t (*C type*), 331
drivers_RT_RT1020_MIMXRT1024.enet_tx_bd_struct_t (*C type*), 330
drivers_RT_RT1020_MIMXRT1024.enet_tx_dirty_ring_t (*C type*), 330
drivers_RT_RT1020_MIMXRT1024.enet_tx_frame_struct_t (*C type*), 332
drivers_RT_RT1020_MIMXRT1024.ENET_TX_INTERRUPT (*C macro*), 322
drivers_RT_RT1020_MIMXRT1024.ENET_Up (*C function*), 307
drivers_RT_RT1020_MIMXRT1024.EWM_CLOCKS (*C macro*), 236
drivers_RT_RT1020_MIMXRT1024.ewm_config_t (*C type*), 344
drivers_RT_RT1020_MIMXRT1024.EWM_Deinit (*C function*), 342
drivers_RT_RT1020_MIMXRT1024.EWM_DisableInterrupts (*C function*), 343
drivers_RT_RT1020_MIMXRT1024.EWM_EnableInterrupts (*C function*), 343
drivers_RT_RT1020_MIMXRT1024.EWM_GetDefaultConfig (*C function*), 342
drivers_RT_RT1020_MIMXRT1024.EWM_GetStatusFlags (*C function*), 343
drivers_RT_RT1020_MIMXRT1024.EWM_Init (*C function*), 342
drivers_RT_RT1020_MIMXRT1024.ewm_lpo_clock_source_t (*C type*), 344
drivers_RT_RT1020_MIMXRT1024.EWM_Refresh (*C function*), 343
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_BusoffErrorHandleIRQ (*C function*), 364
drivers_RT_RT1020_MIMXRT1024.flexcan_byte_error_syndrome_t (*C type*), 377
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_CalculateImprovedTimingValues (*C function*), 346

drivers_RT_RT1020_MIMXRT1024.FLEXCAN_CALLBACK (*C macro*), 380
drivers_RT_RT1020_MIMXRT1024.flexcan_capture_point_t (*C type*), 376
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_ClearHigh64MbStatusFlags (*C function*), 354
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_ClearMbStatusFlags (*C function*), 354
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_ClearStatusFlags (*C function*), 353
drivers_RT_RT1020_MIMXRT1024.flexcan_clock_source_t (*C type*), 376
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_CLOCKS (*C macro*), 236
drivers_RT_RT1020_MIMXRT1024.flexcan_config_t (*C type*), 377
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_Deinit (*C function*), 348
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_DisableHigh64MbInterrupts (*C function*), 355
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_DisableInterrupts (*C function*), 354
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_DisableMbInterrupts (*C function*), 355
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_Enable (*C function*), 355
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_EnableHigh64MbInterrupts (*C function*), 355
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_EnableInterrupts (*C function*), 354
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_EnableMbInterrupts (*C function*), 355
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_EnableRxFifoDMA (*C function*), 355
drivers_RT_RT1020_MIMXRT1024.flexcan_endianness_t (*C type*), 376
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_ENHANCED_RX_FIFO_INT_FLAG (*C macro*), 380
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_EnterFreezeMode (*C function*), 346
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_ERROR_AND_STATUS_INT_FLAG (*C macro*), 380
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_ExitFreezeMode (*C function*), 346
drivers_RT_RT1020_MIMXRT1024.flexcan_fd_frame_t (*C type*), 377
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_FDCalculateImprovedTimingValues (*C function*), 347
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_FDInit (*C function*), 347
drivers_RT_RT1020_MIMXRT1024.flexcan_fifo_transfer_t (*C type*), 378
drivers_RT_RT1020_MIMXRT1024.flexcan_frame_format_t (*C type*), 376
drivers_RT_RT1020_MIMXRT1024.flexcan_frame_t (*C type*), 377
drivers_RT_RT1020_MIMXRT1024.flexcan_frame_type_t (*C type*), 376
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_GetBusErrCount (*C function*), 353
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_GetDefaultConfig (*C function*), 348
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_GetFDMailboxOffset (*C function*), 345
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_GetHigh64MbStatusFlags (*C function*), 353
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_GetHighResolutionTimeStamp (*C function*), 363
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_GetInstance (*C function*), 346
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_GetMbStatusFlags (*C function*), 353
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_GetMemoryErrorReportStatus (*C function*), 354
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_GetPNMatchCount (*C function*), 354
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_GetRxFifoHeadAddr (*C function*), 355
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_GetStatusFlags (*C function*), 353
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_GetTimeStamp (*C function*), 363
drivers_RT_RT1020_MIMXRT1024.flexcan_handle_t (*C type*), 378
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_ID_EXT (*C macro*), 378
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_ID_STD (*C macro*), 378
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_Init (*C function*), 347
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_IsInstanceHasFDMode (*C function*), 345
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_MB_BUSY_TIMEOUT (*C macro*), 378
drivers_RT_RT1020_MIMXRT1024.flexcan_mb_size_t (*C type*), 376
drivers_RT_RT1020_MIMXRT1024.flexcan_MB_timestamp_base_t (*C type*), 376
drivers_RT_RT1020_MIMXRT1024.flexcan_mb_transfer_t (*C type*), 378
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_MbHandleIRQ (*C function*), 364
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_MECR_INT_MASK (*C macro*), 380
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_MECR_INT_UNMASK (*C macro*), 380
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_MECR_STATUS_MASK (*C macro*), 380
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_MECR_STATUS_UNMASK (*C macro*), 380

drivers_RT_RT1020_MIMXRT1024.flexcan_memory_access_type_t (*C type*), 376
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_MEMORY_ERROR_INT_FLAG (*C macro*), 380
drivers_RT_RT1020_MIMXRT1024.flexcan_memory_error_report_status_t (*C type*), 377
drivers_RT_RT1020_MIMXRT1024.flexcan_memory_error_type_t (*C type*), 376
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_MemoryErrorHandleIRQ (*C function*), 364
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_MODULE_TIMEOUT (*C macro*), 378
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_PERIPH_CLOCKS (*C macro*), 236
drivers_RT_RT1020_MIMXRT1024.flexcan_pn_config_t (*C type*), 377
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_PN_EXT_MASK (*C macro*), 380
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_PN_INT_MASK (*C macro*), 380
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_PN_INT_UNMASK (*C macro*), 380
drivers_RT_RT1020_MIMXRT1024.flexcan_pn_match_mode_t (*C type*), 377
drivers_RT_RT1020_MIMXRT1024.flexcan_pn_match_source_t (*C type*), 377
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_PN_STATUS_MASK (*C macro*), 380
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_PN_STATUS_UNMASK (*C macro*), 380
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_PN_STD_MASK (*C macro*), 380
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_PNWAKE_UP_FLAG (*C macro*), 380
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_PNWakeUpHandleIRQ (*C function*), 364
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_POLLING_TIMEOUT (*C macro*), 378
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_ReadFDRxMb (*C function*), 357
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_ReadPNWakeUpMB (*C function*), 357
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_ReadRxFifo (*C function*), 357
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_ReadRxMb (*C function*), 356
drivers_RT_RT1020_MIMXRT1024.flexcan_rx_fifo_config_t (*C type*), 378
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_A (*C macro*), 379
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_B_HIGH (*C macro*), 379
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_B_LOW (*C macro*), 379
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_HIGH (*C macro*), 380
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_LOW (*C macro*), 380
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_MID_HIGH (*C macro*), 380
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_MID_LOW (*C macro*), 380
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_RX_FIFO_EXT_MASK_TYPE_A (*C macro*), 379
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_RX_FIFO_EXT_MASK_TYPE_B_HIGH (*C macro*), 379
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_RX_FIFO_EXT_MASK_TYPE_B_LOW (*C macro*), 379
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_HIGH (*C macro*), 379
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_LOW (*C macro*), 379
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_MID_HIGH (*C macro*), 379
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_MID_LOW (*C macro*), 379
drivers_RT_RT1020_MIMXRT1024.flexcan_rx_fifo_filter_type_t (*C type*), 376
drivers_RT_RT1020_MIMXRT1024.flexcan_rx_fifo_priority_t (*C type*), 376
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_RX_FIFO_STD_FILTER_TYPE_A (*C macro*), 379
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_RX_FIFO_STD_FILTER_TYPE_B_HIGH (*C macro*), 379

drivers_RT_RT1020_MIMXRT1024.FLEXCAN_RX_FIFO_STD_FILTER_TYPE_B_LOW (*C macro*), 379
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_RX_FIFO_STD_FILTER_TYPE_C_HIGH (*C macro*), 379
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_RX_FIFO_STD_FILTER_TYPE_C_LOW (*C macro*), 379
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_RX_FIFO_STD_FILTER_TYPE_C_MID_HIGH (*C macro*), 379
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_RX_FIFO_STD_FILTER_TYPE_C_MID_LOW (*C macro*), 379
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_RX_FIFO_STD_MASK_TYPE_A (*C macro*), 378
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_RX_FIFO_STD_MASK_TYPE_B_HIGH (*C macro*), 378
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_RX_FIFO_STD_MASK_TYPE_B_LOW (*C macro*), 378
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_HIGH (*C macro*), 378
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_LOW (*C macro*), 379
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_MID_HIGH (*C macro*), 379
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_MID_LOW (*C macro*), 379
drivers_RT_RT1020_MIMXRT1024.flexcan_rx_mb_config_t (*C type*), 377
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_RX_MB_EXT_MASK (*C macro*), 378
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_RX_MB_STD_MASK (*C macro*), 378
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_SetBitRate (*C function*), 349
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_SetFDBitRate (*C function*), 349
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_SetFDMbID (*C function*), 352
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_SetFDRxMbConfig (*C function*), 351
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_SetFDTimingConfig (*C function*), 349
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_SetFDTxMbConfig (*C function*), 351
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_SetMbID (*C function*), 351
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_SetPNConfig (*C function*), 352
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_SetRemoteResponseMbConfig (*C function*), 352
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_SetRxFifoConfig (*C function*), 352
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_SetRxFifoGlobalMask (*C function*), 350
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_SetRxIndividualMask (*C function*), 350
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_SetRxMbConfig (*C function*), 350
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_SetRxMbGlobalMask (*C function*), 350
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_SetTimingConfig (*C function*), 348
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_SetTxMbConfig (*C function*), 350
drivers_RT_RT1020_MIMXRT1024.flexcan_timing_config_t (*C type*), 377
drivers_RT_RT1020_MIMXRT1024.flexcan_transfer_callback_t (*C type*), 378
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_TransferAbortReceive (*C function*), 363
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_TransferAbortReceiveFifo (*C function*), 364
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_TransferAbortSend (*C function*), 363
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_TransferCreateHandle (*C function*), 360
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_TransferFDAbortReceive (*C function*), 359
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_TransferFDAbortSend (*C function*), 359
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_TransferFDReceiveBlocking (*C function*), 358
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_TransferFDReceiveNonBlocking (*C function*), 359
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_TransferFDSendBlocking (*C function*), 358
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_TransferFDSendNonBlocking (*C function*), 358
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_TransferGetReceiveFifoCount (*C function*), 363
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_TransferHandleIRQ (*C function*), 364
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_TransferReceiveBlocking (*C function*), 360
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_TransferReceiveFifoBlocking (*C function*), 360

drivers_RT_RT1020_MIMXRT1024.FLEXCAN_TransferReceiveFifoNonBlocking (*C function*), 362
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_TransferReceiveNonBlocking (*C function*), 361
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_TransferRemoteRequestNonBlocking (*C function*), 361
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_TransferRemoteResponseNonBlocking (*C function*), 362
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_TransferSendBlocking (*C function*), 359
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_TransferSendNonBlocking (*C function*), 361
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_WAIT_TIMEOUT (*C macro*), 378
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_WAKE_UP_FLAG (*C macro*), 380
drivers_RT_RT1020_MIMXRT1024.flexcan_wake_up_source_t (*C type*), 376
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_WriteFDTxMb (*C function*), 356
drivers_RT_RT1020_MIMXRT1024.FLEXCAN_WriteTxMb (*C function*), 356
drivers_RT_RT1020_MIMXRT1024.FLEXIO_ClearPortOutput (*C function*), 400
drivers_RT_RT1020_MIMXRT1024.FLEXIO_ClearPortStatus (*C function*), 401
drivers_RT_RT1020_MIMXRT1024.FLEXIO_ClearShifterErrorFlags (*C function*), 398
drivers_RT_RT1020_MIMXRT1024.FLEXIO_ClearShifterStatusFlags (*C function*), 398
drivers_RT_RT1020_MIMXRT1024.FLEXIO_ClearTimerStatusFlags (*C function*), 399
drivers_RT_RT1020_MIMXRT1024.FLEXIO_CLOCKS (*C macro*), 236
drivers_RT_RT1020_MIMXRT1024.flexio_config_t (*C type*), 410
drivers_RT_RT1020_MIMXRT1024.FLEXIO_ConfigPinOverride (*C function*), 401
drivers_RT_RT1020_MIMXRT1024.FLEXIO_Deinit (*C function*), 394
drivers_RT_RT1020_MIMXRT1024.FLEXIO_DisableShifterErrorInterrupts (*C function*), 397
drivers_RT_RT1020_MIMXRT1024.FLEXIO_DisableShifterStatusInterrupts (*C function*), 396
drivers_RT_RT1020_MIMXRT1024.FLEXIO_DisableTimerStatusInterrupts (*C function*), 397
drivers_RT_RT1020_MIMXRT1024.FLEXIO_Enable (*C function*), 395
drivers_RT_RT1020_MIMXRT1024.FLEXIO_EnablePinOutput (*C function*), 400
drivers_RT_RT1020_MIMXRT1024.FLEXIO_EnableShifterErrorInterrupts (*C function*), 397
drivers_RT_RT1020_MIMXRT1024.FLEXIO_EnableShifterStatusDMA (*C function*), 399
drivers_RT_RT1020_MIMXRT1024.FLEXIO_EnableShifterStatusInterrupts (*C function*), 396
drivers_RT_RT1020_MIMXRT1024.FLEXIO_EnableTimerStatusInterrupts (*C function*), 397
drivers_RT_RT1020_MIMXRT1024.FLEXIO_GetDefaultConfig (*C function*), 394
drivers_RT_RT1020_MIMXRT1024.FLEXIO_GetInstance (*C function*), 394
drivers_RT_RT1020_MIMXRT1024.FLEXIO_GetPinOverride (*C function*), 401
drivers_RT_RT1020_MIMXRT1024.FLEXIO_GetPinStatus (*C function*), 401
drivers_RT_RT1020_MIMXRT1024.FLEXIO_GetShifterBufferAddress (*C function*), 399
drivers_RT_RT1020_MIMXRT1024.FLEXIO_GetShifterErrorFlags (*C function*), 398
drivers_RT_RT1020_MIMXRT1024.FLEXIO_GetShifterState (*C function*), 395
drivers_RT_RT1020_MIMXRT1024.FLEXIO_GetShifterStatusFlags (*C function*), 398
drivers_RT_RT1020_MIMXRT1024.FLEXIO_GetTimerStatusFlags (*C function*), 398
drivers_RT_RT1020_MIMXRT1024.flexio_gpio_config_t (*C type*), 410
drivers_RT_RT1020_MIMXRT1024.flexio_gpio_direction_t (*C type*), 410
drivers_RT_RT1020_MIMXRT1024.FLEXIO_I2C_CheckForBusyBus (*C function*), 424
drivers_RT_RT1020_MIMXRT1024.flexio_i2c_direction_t (*C type*), 431
drivers_RT_RT1020_MIMXRT1024.flexio_i2c_master_config_t (*C type*), 431
drivers_RT_RT1020_MIMXRT1024.flexio_i2c_master_handle_t (*C type*), 431
drivers_RT_RT1020_MIMXRT1024.flexio_i2c_master_transfer_callback_t (*C type*), 431
drivers_RT_RT1020_MIMXRT1024.flexio_i2c_master_transfer_t (*C type*), 431
drivers_RT_RT1020_MIMXRT1024.FLEXIO_I2C_MasterAbortStop (*C function*), 426
drivers_RT_RT1020_MIMXRT1024.FLEXIO_I2C_MasterClearStatusFlags (*C function*), 425
drivers_RT_RT1020_MIMXRT1024.FLEXIO_I2C_MasterDeinit (*C function*), 424
drivers_RT_RT1020_MIMXRT1024.FLEXIO_I2C_MasterDisableInterrupts (*C function*), 425
drivers_RT_RT1020_MIMXRT1024.FLEXIO_I2C_MasterEnable (*C function*), 425
drivers_RT_RT1020_MIMXRT1024.FLEXIO_I2C_MasterEnableAck (*C function*), 426
drivers_RT_RT1020_MIMXRT1024.FLEXIO_I2C_MasterEnableInterrupts (*C function*), 425
drivers_RT_RT1020_MIMXRT1024.FLEXIO_I2C_MasterGetDefaultConfig (*C function*), 425
drivers_RT_RT1020_MIMXRT1024.FLEXIO_I2C_MasterGetStatusFlags (*C function*), 425
drivers_RT_RT1020_MIMXRT1024.FLEXIO_I2C_MasterInit (*C function*), 424

drivers_RT_RT1020_MIMXRT1024.FLEXIO_I2C_MasterReadBlocking (*C function*), 428
drivers_RT_RT1020_MIMXRT1024.FLEXIO_I2C_MasterReadByte (*C function*), 427
drivers_RT_RT1020_MIMXRT1024.FLEXIO_I2C_MasterRepeatedStart (*C function*), 426
drivers_RT_RT1020_MIMXRT1024.FLEXIO_I2C_MasterSetBaudRate (*C function*), 426
drivers_RT_RT1020_MIMXRT1024.FLEXIO_I2C_MasterSetTransferCount (*C function*), 427
drivers_RT_RT1020_MIMXRT1024.FLEXIO_I2C_MasterStart (*C function*), 426
drivers_RT_RT1020_MIMXRT1024.FLEXIO_I2C_MasterStop (*C function*), 426
drivers_RT_RT1020_MIMXRT1024.FLEXIO_I2C_MasterTransferAbort (*C function*), 429
drivers_RT_RT1020_MIMXRT1024.FLEXIO_I2C_MasterTransferBlocking (*C function*), 428
drivers_RT_RT1020_MIMXRT1024.FLEXIO_I2C_MasterTransferCreateHandle (*C function*), 428
drivers_RT_RT1020_MIMXRT1024.FLEXIO_I2C_MasterTransferGetCount (*C function*), 429
drivers_RT_RT1020_MIMXRT1024.FLEXIO_I2C_MasterTransferHandleIRQ (*C function*), 430
drivers_RT_RT1020_MIMXRT1024.FLEXIO_I2C_MasterTransferNonBlocking (*C function*), 429
drivers_RT_RT1020_MIMXRT1024.FLEXIO_I2C_MasterWriteBlocking (*C function*), 427
drivers_RT_RT1020_MIMXRT1024.FLEXIO_I2C_MasterWriteByte (*C function*), 427
drivers_RT_RT1020_MIMXRT1024.FLEXIO_I2C_Type (*C type*), 431
drivers_RT_RT1020_MIMXRT1024.flexio_i2s_callback_t (*C type*), 442
drivers_RT_RT1020_MIMXRT1024.flexio_i2s_config_t (*C type*), 442
drivers_RT_RT1020_MIMXRT1024.FLEXIO_I2S_Deinit (*C function*), 434
drivers_RT_RT1020_MIMXRT1024.FLEXIO_I2S_DisableInterrupts (*C function*), 435
drivers_RT_RT1020_MIMXRT1024.flexio_i2s_edma_callback_t (*C type*), 416
drivers_RT_RT1020_MIMXRT1024.flexio_i2s_edma_handle_t (*C type*), 416
drivers_RT_RT1020_MIMXRT1024.FLEXIO_I2S_Enable (*C function*), 434
drivers_RT_RT1020_MIMXRT1024.FLEXIO_I2S_EnableInterrupts (*C function*), 434
drivers_RT_RT1020_MIMXRT1024.flexio_i2s_format_t (*C type*), 442
drivers_RT_RT1020_MIMXRT1024.FLEXIO_I2S_GetDefaultConfig (*C function*), 434
drivers_RT_RT1020_MIMXRT1024.FLEXIO_I2S_GetStatusFlags (*C function*), 434
drivers_RT_RT1020_MIMXRT1024.flexio_i2s_handle_t (*C type*), 442
drivers_RT_RT1020_MIMXRT1024.FLEXIO_I2S_Init (*C function*), 433
drivers_RT_RT1020_MIMXRT1024.flexio_i2s_master_slave_t (*C type*), 442
drivers_RT_RT1020_MIMXRT1024.FLEXIO_I2S_MasterSetFormat (*C function*), 435
drivers_RT_RT1020_MIMXRT1024.FLEXIO_I2S_ReadBlocking (*C function*), 436
drivers_RT_RT1020_MIMXRT1024.FLEXIO_I2S_ReadData (*C function*), 437
drivers_RT_RT1020_MIMXRT1024.FLEXIO_I2S_RxEnableDMA (*C function*), 435
drivers_RT_RT1020_MIMXRT1024.FLEXIO_I2S_RxGetDataRegisterAddress (*C function*), 435
drivers_RT_RT1020_MIMXRT1024.flexio_i2s_sample_rate_t (*C type*), 442
drivers_RT_RT1020_MIMXRT1024.FLEXIO_I2S_SlaveSetFormat (*C function*), 435
drivers_RT_RT1020_MIMXRT1024.flexio_i2s_transfer_t (*C type*), 442
drivers_RT_RT1020_MIMXRT1024.FLEXIO_I2S_TransferAbortReceive (*C function*), 439
drivers_RT_RT1020_MIMXRT1024.FLEXIO_I2S_TransferAbortReceiveEDMA (*C function*), 415
drivers_RT_RT1020_MIMXRT1024.FLEXIO_I2S_TransferAbortSend (*C function*), 438
drivers_RT_RT1020_MIMXRT1024.FLEXIO_I2S_TransferAbortSendEDMA (*C function*), 415
drivers_RT_RT1020_MIMXRT1024.FLEXIO_I2S_TransferGetReceiveCount (*C function*), 439
drivers_RT_RT1020_MIMXRT1024.FLEXIO_I2S_TransferGetReceiveCountEDMA (*C function*), 416
drivers_RT_RT1020_MIMXRT1024.FLEXIO_I2S_TransferGetSendCount (*C function*), 439
drivers_RT_RT1020_MIMXRT1024.FLEXIO_I2S_TransferGetSendCountEDMA (*C function*), 416
drivers_RT_RT1020_MIMXRT1024.FLEXIO_I2S_TransferReceiveEDMA (*C function*), 415
drivers_RT_RT1020_MIMXRT1024.FLEXIO_I2S_TransferReceiveNonBlocking (*C function*), 438
drivers_RT_RT1020_MIMXRT1024.FLEXIO_I2S_TransferRxCreateHandle (*C function*), 437
drivers_RT_RT1020_MIMXRT1024.FLEXIO_I2S_TransferRxCreateHandleEDMA (*C function*), 414
drivers_RT_RT1020_MIMXRT1024.FLEXIO_I2S_TransferRxHandleIRQ (*C function*), 440
drivers_RT_RT1020_MIMXRT1024.FLEXIO_I2S_TransferSendEDMA (*C function*), 414
drivers_RT_RT1020_MIMXRT1024.FLEXIO_I2S_TransferSendNonBlocking (*C function*), 437
drivers_RT_RT1020_MIMXRT1024.FLEXIO_I2S_TransferSetFormat (*C function*), 437
drivers_RT_RT1020_MIMXRT1024.FLEXIO_I2S_TransferSetFormatEDMA (*C function*), 414
drivers_RT_RT1020_MIMXRT1024.FLEXIO_I2S_TransferTxCreateHandle (*C function*), 437
drivers_RT_RT1020_MIMXRT1024.FLEXIO_I2S_TransferTxCreateHandleEDMA (*C function*), 414
drivers_RT_RT1020_MIMXRT1024.FLEXIO_I2S_TransferTxHandleIRQ (*C function*), 439

drivers_RT_RT1020_MIMXRT1024.FLEXIO_I2S_TxEnableDMA (*C function*), 435
drivers_RT_RT1020_MIMXRT1024.FLEXIO_I2S_TxGetDataRegisterAddress (*C function*), 435
drivers_RT_RT1020_MIMXRT1024.FLEXIO_I2S_Type (*C type*), 442
drivers_RT_RT1020_MIMXRT1024.flexio_i2s_word_width_t (*C type*), 442
drivers_RT_RT1020_MIMXRT1024.FLEXIO_I2S_WriteBlocking (*C function*), 436
drivers_RT_RT1020_MIMXRT1024.FLEXIO_I2S_WriteData (*C function*), 436
drivers_RT_RT1020_MIMXRT1024.FLEXIO_I2S_XFER_QUEUE_SIZE (*C macro*), 442
drivers_RT_RT1020_MIMXRT1024.FLEXIO_Init (*C function*), 394
drivers_RT_RT1020_MIMXRT1024.flexio_isr_t (*C type*), 410
drivers_RT_RT1020_MIMXRT1024.flexio_pin_config_t (*C type*), 409
drivers_RT_RT1020_MIMXRT1024.flexio_pin_input_config_t (*C type*), 410
drivers_RT_RT1020_MIMXRT1024.flexio_pin_polarity_t (*C type*), 409
drivers_RT_RT1020_MIMXRT1024.FLEXIO_PinRead (*C function*), 400
drivers_RT_RT1020_MIMXRT1024.FLEXIO_PinWrite (*C function*), 400
drivers_RT_RT1020_MIMXRT1024.FLEXIO_ReadPinInput (*C function*), 395
drivers_RT_RT1020_MIMXRT1024.FLEXIO_RegisterHandleIRQ (*C function*), 399
drivers_RT_RT1020_MIMXRT1024.FLEXIO_Reset (*C function*), 394
drivers_RT_RT1020_MIMXRT1024.FLEXIO_SetClockMode (*C function*), 396
drivers_RT_RT1020_MIMXRT1024.FLEXIO_SetPinConfig (*C function*), 410
drivers_RT_RT1020_MIMXRT1024.FLEXIO_SetPinLevel (*C function*), 401
drivers_RT_RT1020_MIMXRT1024.FLEXIO_SetPortOutput (*C function*), 400
drivers_RT_RT1020_MIMXRT1024.FLEXIO_SetShifterConfig (*C function*), 395
drivers_RT_RT1020_MIMXRT1024.FLEXIO_SetTimerConfig (*C function*), 395
drivers_RT_RT1020_MIMXRT1024.flexio_shifter_buffer_type_t (*C type*), 410
drivers_RT_RT1020_MIMXRT1024.flexio_shifter_config_t (*C type*), 410
drivers_RT_RT1020_MIMXRT1024.flexio_shifter_input_source_t (*C type*), 410
drivers_RT_RT1020_MIMXRT1024.flexio_shifter_mode_t (*C type*), 409
drivers_RT_RT1020_MIMXRT1024.flexio_shifter_start_bit_t (*C type*), 410
drivers_RT_RT1020_MIMXRT1024.flexio_shifter_stop_bit_t (*C type*), 410
drivers_RT_RT1020_MIMXRT1024.flexio_shifter_timer_polarity_t (*C type*), 409
drivers_RT_RT1020_MIMXRT1024.FLEXIO_SPI_ClearStatusFlags (*C function*), 447
drivers_RT_RT1020_MIMXRT1024.flexio_spi_clock_phase_t (*C type*), 456
drivers_RT_RT1020_MIMXRT1024.flexio_spi_data_bitcount_mode_t (*C type*), 456
drivers_RT_RT1020_MIMXRT1024.FLEXIO_SPI_DisableInterrupts (*C function*), 448
drivers_RT_RT1020_MIMXRT1024.FLEXIO_SPI_DUMMYDATA (*C macro*), 456
drivers_RT_RT1020_MIMXRT1024.FLEXIO_SPI_Enable (*C function*), 449
drivers_RT_RT1020_MIMXRT1024.FLEXIO_SPI_EnableDMA (*C function*), 448
drivers_RT_RT1020_MIMXRT1024.FLEXIO_SPI_EnableInterrupts (*C function*), 448
drivers_RT_RT1020_MIMXRT1024.FLEXIO_SPI_FlushShifters (*C function*), 451
drivers_RT_RT1020_MIMXRT1024.FLEXIO_SPI_GetRxDataRegisterAddress (*C function*), 449
drivers_RT_RT1020_MIMXRT1024.FLEXIO_SPI_GetStatusFlags (*C function*), 447
drivers_RT_RT1020_MIMXRT1024.FLEXIO_SPI_GetTxDataRegisterAddress (*C function*), 448
drivers_RT_RT1020_MIMXRT1024.flexio_spi_master_config_t (*C type*), 456
drivers_RT_RT1020_MIMXRT1024.flexio_spi_master_edma_handle_t (*C type*), 420
drivers_RT_RT1020_MIMXRT1024.flexio_spi_master_edma_transfer_callback_t (*C type*), 420
drivers_RT_RT1020_MIMXRT1024.flexio_spi_master_handle_t (*C type*), 456
drivers_RT_RT1020_MIMXRT1024.flexio_spi_master_transfer_callback_t (*C type*), 456
drivers_RT_RT1020_MIMXRT1024.FLEXIO_SPI_MasterDeinit (*C function*), 446
drivers_RT_RT1020_MIMXRT1024.FLEXIO_SPI_MasterGetDefaultConfig (*C function*), 446
drivers_RT_RT1020_MIMXRT1024.FLEXIO_SPI_MasterInit (*C function*), 445
drivers_RT_RT1020_MIMXRT1024.FLEXIO_SPI_MasterSetBaudRate (*C function*), 449
drivers_RT_RT1020_MIMXRT1024.FLEXIO_SPI_MasterTransferAbort (*C function*), 451
drivers_RT_RT1020_MIMXRT1024.FLEXIO_SPI_MasterTransferAbortEDMA (*C function*), 418
drivers_RT_RT1020_MIMXRT1024.FLEXIO_SPI_MasterTransferBlocking (*C function*), 450
drivers_RT_RT1020_MIMXRT1024.FLEXIO_SPI_MasterTransferCreateHandle (*C function*), 451
drivers_RT_RT1020_MIMXRT1024.FLEXIO_SPI_MasterTransferCreateHandleEDMA (*C function*), 417
drivers_RT_RT1020_MIMXRT1024.FLEXIO_SPI_MasterTransferEDMA (*C function*), 418

drivers_RT_RT1020_MIMXRT1024.FLEXIO_SPI_MasterTransferGetCount (*C function*), 452
drivers_RT_RT1020_MIMXRT1024.FLEXIO_SPI_MasterTransferGetCountEDMA (*C function*), 418
drivers_RT_RT1020_MIMXRT1024.FLEXIO_SPI_MasterTransferHandleIRQ (*C function*), 452
drivers_RT_RT1020_MIMXRT1024.FLEXIO_SPI_MasterTransferNonBlocking (*C function*), 451
drivers_RT_RT1020_MIMXRT1024.FLEXIO_SPI_ReadBlocking (*C function*), 450
drivers_RT_RT1020_MIMXRT1024.FLEXIO_SPI_ReadData (*C function*), 449
drivers_RT_RT1020_MIMXRT1024.flexio_spi_shift_direction_t (*C type*), 456
drivers_RT_RT1020_MIMXRT1024.flexio_spi_slave_config_t (*C type*), 456
drivers_RT_RT1020_MIMXRT1024.flexio_spi_slave_edma_handle_t (*C type*), 420
drivers_RT_RT1020_MIMXRT1024.flexio_spi_slave_edma_transfer_callback_t (*C type*), 420
drivers_RT_RT1020_MIMXRT1024.flexio_spi_slave_handle_t (*C type*), 456
drivers_RT_RT1020_MIMXRT1024.flexio_spi_slave_transfer_callback_t (*C type*), 456
drivers_RT_RT1020_MIMXRT1024.FLEXIO_SPI_SlaveDeinit (*C function*), 447
drivers_RT_RT1020_MIMXRT1024.FLEXIO_SPI_SlaveGetDefaultConfig (*C function*), 447
drivers_RT_RT1020_MIMXRT1024.FLEXIO_SPI_SlaveInit (*C function*), 446
drivers_RT_RT1020_MIMXRT1024.FLEXIO_SPI_SlaveTransferAbort (*C function*), 453
drivers_RT_RT1020_MIMXRT1024.FLEXIO_SPI_SlaveTransferAbortEDMA (*C function*), 419
drivers_RT_RT1020_MIMXRT1024.FLEXIO_SPI_SlaveTransferCreateHandle (*C function*), 452
drivers_RT_RT1020_MIMXRT1024.FLEXIO_SPI_SlaveTransferCreateHandleEDMA (*C function*), 418
drivers_RT_RT1020_MIMXRT1024.FLEXIO_SPI_SlaveTransferEDMA (*C function*), 419
drivers_RT_RT1020_MIMXRT1024.FLEXIO_SPI_SlaveTransferGetCount (*C function*), 453
drivers_RT_RT1020_MIMXRT1024.FLEXIO_SPI_SlaveTransferGetCountEDMA (*C function*), 419
drivers_RT_RT1020_MIMXRT1024.FLEXIO_SPI_SlaveTransferHandleIRQ (*C function*), 453
drivers_RT_RT1020_MIMXRT1024.FLEXIO_SPI_SlaveTransferNonBlocking (*C function*), 452
drivers_RT_RT1020_MIMXRT1024.flexio_spi_transfer_t (*C type*), 456
drivers_RT_RT1020_MIMXRT1024.FLEXIO_SPI_Type (*C type*), 456
drivers_RT_RT1020_MIMXRT1024.FLEXIO_SPI_WriteBlocking (*C function*), 450
drivers_RT_RT1020_MIMXRT1024.FLEXIO_SPI_WriteData (*C function*), 449
drivers_RT_RT1020_MIMXRT1024.FLEXIO_SPI_XFER_DATA_FORMAT (*C macro*), 456
drivers_RT_RT1020_MIMXRT1024.flexio_timer_config_t (*C type*), 410
drivers_RT_RT1020_MIMXRT1024.flexio_timer_decrement_source_t (*C type*), 409
drivers_RT_RT1020_MIMXRT1024.flexio_timer_disable_condition_t (*C type*), 409
drivers_RT_RT1020_MIMXRT1024.flexio_timer_enable_condition_t (*C type*), 409
drivers_RT_RT1020_MIMXRT1024.flexio_timer_mode_t (*C type*), 409
drivers_RT_RT1020_MIMXRT1024.flexio_timer_output_state_t (*C type*), 409
drivers_RT_RT1020_MIMXRT1024.flexio_timer_output_t (*C type*), 409
drivers_RT_RT1020_MIMXRT1024.flexio_timer_reset_condition_t (*C type*), 409
drivers_RT_RT1020_MIMXRT1024.flexio_timer_start_bit_condition_t (*C type*), 409
drivers_RT_RT1020_MIMXRT1024.flexio_timer_stop_bit_condition_t (*C type*), 409
drivers_RT_RT1020_MIMXRT1024.flexio_timer_trigger_polarity_t (*C type*), 409
drivers_RT_RT1020_MIMXRT1024.FLEXIO_TIMER_TRIGGER_SEL_PININPUT (*C macro*), 411
drivers_RT_RT1020_MIMXRT1024.FLEXIO_TIMER_TRIGGER_SEL_SHIFTnSTAT (*C macro*), 411
drivers_RT_RT1020_MIMXRT1024.FLEXIO_TIMER_TRIGGER_SEL_TIMn (*C macro*), 411
drivers_RT_RT1020_MIMXRT1024.flexio_timer_trigger_source_t (*C type*), 409
drivers_RT_RT1020_MIMXRT1024.FLEXIO_TogglePortOutput (*C function*), 400
drivers_RT_RT1020_MIMXRT1024.flexio_uart_bit_count_per_char_t (*C type*), 468
drivers_RT_RT1020_MIMXRT1024.FLEXIO_UART_ClearStatusFlags (*C function*), 460
drivers_RT_RT1020_MIMXRT1024.flexio_uart_config_t (*C type*), 468
drivers_RT_RT1020_MIMXRT1024.FLEXIO_UART_Deinit (*C function*), 460
drivers_RT_RT1020_MIMXRT1024.FLEXIO_UART_DisableInterrupts (*C function*), 461
drivers_RT_RT1020_MIMXRT1024.flexio_uart_edma_handle_t (*C type*), 423
drivers_RT_RT1020_MIMXRT1024.flexio_uart_edma_transfer_callback_t (*C type*), 423
drivers_RT_RT1020_MIMXRT1024.FLEXIO_UART_Enable (*C function*), 462
drivers_RT_RT1020_MIMXRT1024.FLEXIO_UART_EnableInterrupts (*C function*), 461
drivers_RT_RT1020_MIMXRT1024.FLEXIO_UART_EnableRxDMA (*C function*), 462

drivers_RT_RT1020_MIMXRT1024.FLEXIO_UART_EnableTxDMA (*C function*), 461
drivers_RT_RT1020_MIMXRT1024.FLEXIO_UART_FlushShifters (*C function*), 466
drivers_RT_RT1020_MIMXRT1024.FLEXIO_UART_GetDefaultConfig (*C function*), 460
drivers_RT_RT1020_MIMXRT1024.FLEXIO_UART_GetRxDataRegisterAddress (*C function*), 461
drivers_RT_RT1020_MIMXRT1024.FLEXIO_UART_GetStatusFlags (*C function*), 460
drivers_RT_RT1020_MIMXRT1024.FLEXIO_UART_GetTxDataRegisterAddress (*C function*), 461
drivers_RT_RT1020_MIMXRT1024.flexio_uart_handle_t (*C type*), 468
drivers_RT_RT1020_MIMXRT1024.FLEXIO_UART_Init (*C function*), 459
drivers_RT_RT1020_MIMXRT1024.FLEXIO_UART_ReadBlocking (*C function*), 463
drivers_RT_RT1020_MIMXRT1024.FLEXIO_UART_ReadByte (*C function*), 462
drivers_RT_RT1020_MIMXRT1024.flexio_uart_transfer_callback_t (*C type*), 468
drivers_RT_RT1020_MIMXRT1024.flexio_uart_transfer_t (*C type*), 468
drivers_RT_RT1020_MIMXRT1024.FLEXIO_UART_TransferAbortReceive (*C function*), 466
drivers_RT_RT1020_MIMXRT1024.FLEXIO_UART_TransferAbortReceiveEDMA (*C function*), 422
drivers_RT_RT1020_MIMXRT1024.FLEXIO_UART_TransferAbortSend (*C function*), 464
drivers_RT_RT1020_MIMXRT1024.FLEXIO_UART_TransferAbortSendEDMA (*C function*), 422
drivers_RT_RT1020_MIMXRT1024.FLEXIO_UART_TransferCreateHandle (*C function*), 463
drivers_RT_RT1020_MIMXRT1024.FLEXIO_UART_TransferCreateHandleEDMA (*C function*), 421
drivers_RT_RT1020_MIMXRT1024.FLEXIO_UART_TransferGetReceiveCount (*C function*), 466
drivers_RT_RT1020_MIMXRT1024.FLEXIO_UART_TransferGetReceiveCountEDMA (*C function*), 422
drivers_RT_RT1020_MIMXRT1024.FLEXIO_UART_TransferGetSendCount (*C function*), 465
drivers_RT_RT1020_MIMXRT1024.FLEXIO_UART_TransferGetSendCountEDMA (*C function*), 422
drivers_RT_RT1020_MIMXRT1024.FLEXIO_UART_TransferHandleIRQ (*C function*), 466
drivers_RT_RT1020_MIMXRT1024.FLEXIO_UART_TransferReceiveEDMA (*C function*), 421
drivers_RT_RT1020_MIMXRT1024.FLEXIO_UART_TransferReceiveNonBlocking (*C function*), 465
drivers_RT_RT1020_MIMXRT1024.FLEXIO_UART_TransferSendEDMA (*C function*), 421
drivers_RT_RT1020_MIMXRT1024.FLEXIO_UART_TransferSendNonBlocking (*C function*), 464
drivers_RT_RT1020_MIMXRT1024.FLEXIO_UART_TransferStartRingBuffer (*C function*), 463
drivers_RT_RT1020_MIMXRT1024.FLEXIO_UART_TransferStopRingBuffer (*C function*), 464
drivers_RT_RT1020_MIMXRT1024.FLEXIO_UART_Type (*C type*), 468
drivers_RT_RT1020_MIMXRT1024.FLEXIO_UART_WriteBlocking (*C function*), 462
drivers_RT_RT1020_MIMXRT1024.FLEXIO_UART_WriteByte (*C function*), 462
drivers_RT_RT1020_MIMXRT1024.FLEXIO_UnregisterHandleIRQ (*C function*), 400
drivers_RT_RT1020_MIMXRT1024.flexpwm_capture_channel_config_t (*C type*), 518
drivers_RT_RT1020_MIMXRT1024.flexpwm_capture_channel_t (*C type*), 518
drivers_RT_RT1020_MIMXRT1024.flexpwm_capture_edge_t (*C type*), 517
drivers_RT_RT1020_MIMXRT1024.flexpwm_capture_fifo_watermark_t (*C type*), 518
drivers_RT_RT1020_MIMXRT1024.flexpwm_capture_filter_channel_config_t (*C type*), 518
drivers_RT_RT1020_MIMXRT1024.flexpwm_capture_filter_config_t (*C type*), 519
drivers_RT_RT1020_MIMXRT1024.flexpwm_capture_index_t (*C type*), 518
drivers_RT_RT1020_MIMXRT1024.flexpwm_capture_input_select_t (*C type*), 518
drivers_RT_RT1020_MIMXRT1024.FLEXPWM_ClearFaultStatusFlags (*C function*), 498
drivers_RT_RT1020_MIMXRT1024.FLEXPWM_ClearLoadOkay (*C function*), 492
drivers_RT_RT1020_MIMXRT1024.FLEXPWM_ClearSubmoduleStatusFlags (*C function*), 497
drivers_RT_RT1020_MIMXRT1024.flexpwm_clock_source_t (*C type*), 514
drivers_RT_RT1020_MIMXRT1024.flexpwm_compare_mode_t (*C type*), 514
drivers_RT_RT1020_MIMXRT1024.FLEXPWM_ConfigDMA (*C function*), 499
drivers_RT_RT1020_MIMXRT1024.FLEXPWM_ConfigFaultFilter (*C function*), 477
drivers_RT_RT1020_MIMXRT1024.FLEXPWM_ConfigFaultProtection (*C function*), 476
drivers_RT_RT1020_MIMXRT1024.FLEXPWM_ConfigFaultSubmodule (*C function*), 477
drivers_RT_RT1020_MIMXRT1024.FLEXPWM_ConfigForceOut (*C function*), 474
drivers_RT_RT1020_MIMXRT1024.FLEXPWM_ConfigInputCapture (*C function*), 480
drivers_RT_RT1020_MIMXRT1024.FLEXPWM_ConfigInputCaptureFilter (*C function*), 481
drivers_RT_RT1020_MIMXRT1024.FLEXPWM_ConfigOutputTrigger (*C function*), 478
drivers_RT_RT1020_MIMXRT1024.FLEXPWM_ConfigPWM (*C function*), 472
drivers_RT_RT1020_MIMXRT1024.FLEXPWM_ConfigPWMChannelX (*C function*), 473

drivers_RT_RT1020_MIMXRT1024.FLEXPWM_ConfigSubmodule (*C function*), 471
drivers_RT_RT1020_MIMXRT1024.flexpwm_counter_config_t (*C type*), 514
drivers_RT_RT1020_MIMXRT1024.FLEXPWM_DeInit (*C function*), 520
drivers_RT_RT1020_MIMXRT1024.FLEXPWM_DisableDMA (*C function*), 500
drivers_RT_RT1020_MIMXRT1024.FLEXPWM_DisableFaultInterrupts (*C function*), 498
drivers_RT_RT1020_MIMXRT1024.FLEXPWM_DisableInputCapture (*C function*), 483
drivers_RT_RT1020_MIMXRT1024.FLEXPWM_DisablePWMOutput (*C function*), 495
drivers_RT_RT1020_MIMXRT1024.FLEXPWM_DisableSubmoduleCounter (*C function*), 494
drivers_RT_RT1020_MIMXRT1024.FLEXPWM_DisableSubmoduleInterrupts (*C function*), 497
drivers_RT_RT1020_MIMXRT1024.flexpwm_dma_capture_source_t (*C type*), 519
drivers_RT_RT1020_MIMXRT1024.flexpwm_dma_config_t (*C type*), 519
drivers_RT_RT1020_MIMXRT1024.FLEXPWM_EnableDMA (*C function*), 500
drivers_RT_RT1020_MIMXRT1024.FLEXPWM_EnableFaultInterrupts (*C function*), 497
drivers_RT_RT1020_MIMXRT1024.FLEXPWM_EnableInputCapture (*C function*), 483
drivers_RT_RT1020_MIMXRT1024.FLEXPWM_EnablePWMOutput (*C function*), 495
drivers_RT_RT1020_MIMXRT1024.FLEXPWM_EnableSubmoduleCounter (*C function*), 494
drivers_RT_RT1020_MIMXRT1024.FLEXPWM_EnableSubmoduleInterrupts (*C function*), 496
drivers_RT_RT1020_MIMXRT1024.FLEXPWM_EnableUpdateMaskImmediately (*C function*), 488
drivers_RT_RT1020_MIMXRT1024.flexpwm_fault_clearing_mode_t (*C type*), 516
drivers_RT_RT1020_MIMXRT1024.flexpwm_fault_config_t (*C type*), 516
drivers_RT_RT1020_MIMXRT1024.flexpwm_fault_filter_config_t (*C type*), 516
drivers_RT_RT1020_MIMXRT1024.flexpwm_fault_mask_t (*C type*), 516
drivers_RT_RT1020_MIMXRT1024.flexpwm_fault_output_behavior_t (*C type*), 515
drivers_RT_RT1020_MIMXRT1024.flexpwm_fault_output_config_t (*C type*), 516
drivers_RT_RT1020_MIMXRT1024.flexpwm_fault_submodule_config_t (*C type*), 516
drivers_RT_RT1020_MIMXRT1024.flexpwm_force_out_channel_config_t (*C type*), 515
drivers_RT_RT1020_MIMXRT1024.flexpwm_force_out_config_t (*C type*), 515
drivers_RT_RT1020_MIMXRT1024.flexpwm_force_output_source_t (*C type*), 515
drivers_RT_RT1020_MIMXRT1024.flexpwm_force_pwm_source_t (*C type*), 515
drivers_RT_RT1020_MIMXRT1024.FLEXPWM_GET_FCTRL2_REG (*C macro*), 520
drivers_RT_RT1020_MIMXRT1024.FLEXPWM_GET_FCTRL_REG (*C macro*), 520
drivers_RT_RT1020_MIMXRT1024.FLEXPWM_GET_FFILT_REG (*C macro*), 520
drivers_RT_RT1020_MIMXRT1024.FLEXPWM_GET_FSTS_REG (*C macro*), 520
drivers_RT_RT1020_MIMXRT1024.FLEXPWM_GetDefaultCaptureFilterConfig (*C function*), 479
drivers_RT_RT1020_MIMXRT1024.FLEXPWM_GetDefaultFaultConfig (*C function*), 475
drivers_RT_RT1020_MIMXRT1024.FLEXPWM_GetDefaultFaultFilterConfig (*C function*), 475
drivers_RT_RT1020_MIMXRT1024.FLEXPWM_GetDefaultFaultSubmoduleConfig (*C function*), 475
drivers_RT_RT1020_MIMXRT1024.FLEXPWM_GetDefaultForceOutConfig (*C function*), 474
drivers_RT_RT1020_MIMXRT1024.FLEXPWM_GetDefaultInputCaptureConfig (*C function*), 479
drivers_RT_RT1020_MIMXRT1024.FLEXPWM_GetDefaultOutputTriggerConfig (*C function*), 478
drivers_RT_RT1020_MIMXRT1024.FLEXPWM_GetDefaultSubmoduleConfig (*C function*), 471
drivers_RT_RT1020_MIMXRT1024.FLEXPWM_GetFaultFilteredPinStatusFlags (*C function*), 499
drivers_RT_RT1020_MIMXRT1024.FLEXPWM_GetFaultStatusFlags (*C function*), 498
drivers_RT_RT1020_MIMXRT1024.FLEXPWM_GetInputCaptureCycle (*C function*), 482
drivers_RT_RT1020_MIMXRT1024.FLEXPWM_GetInputCaptureValue (*C function*), 481
drivers_RT_RT1020_MIMXRT1024.FLEXPWM_GetInstance (*C function*), 519
drivers_RT_RT1020_MIMXRT1024.FLEXPWM_GetLoadOkay (*C function*), 492
drivers_RT_RT1020_MIMXRT1024.FLEXPWM_GetSubmoduleStatusFlags (*C function*), 497
drivers_RT_RT1020_MIMXRT1024.FLEXPWM_Init (*C function*), 519
drivers_RT_RT1020_MIMXRT1024.flexpwm_init_source_t (*C type*), 514
drivers_RT_RT1020_MIMXRT1024.flexpwm_input_capture_config_t (*C type*), 518
drivers_RT_RT1020_MIMXRT1024.flexpwm_ipol_source_t (*C type*), 515
drivers_RT_RT1020_MIMXRT1024.flexpwm_load_mode_t (*C type*), 514
drivers_RT_RT1020_MIMXRT1024.FLEXPWM_MaskPWMOutput (*C function*), 487
drivers_RT_RT1020_MIMXRT1024.flexpwm_output_trigger_config_t (*C type*), 517
drivers_RT_RT1020_MIMXRT1024.flexpwm_output_trigger_mask_t (*C type*), 517
drivers_RT_RT1020_MIMXRT1024.flexpwm_prescaler_t (*C type*), 514
drivers_RT_RT1020_MIMXRT1024.flexpwm_pwm_channel_config_t (*C type*), 515

drivers_RT_RT1020_MIMXRT1024.flexpwm_pwm_config_t (*C type*), 515
drivers_RT_RT1020_MIMXRT1024.flexpwm_pwm_polarity_t (*C type*), 515
drivers_RT_RT1020_MIMXRT1024.flexpwm_reload_config_t (*C type*), 514
drivers_RT_RT1020_MIMXRT1024.flexpwm_reload_source_t (*C type*), 514
drivers_RT_RT1020_MIMXRT1024.FLEXPWM_SetCompareMode (*C function*), 487
drivers_RT_RT1020_MIMXRT1024.FLEXPWM_SetComplementaryPWMSource23 (*C function*), 490
drivers_RT_RT1020_MIMXRT1024.FLEXPWM_SetComplementaryPWMSource45 (*C function*), 491
drivers_RT_RT1020_MIMXRT1024.FLEXPWM_SetCounterForceInitialization (*C function*), 493
drivers_RT_RT1020_MIMXRT1024.FLEXPWM_SetDTCNT0 (*C function*), 486
drivers_RT_RT1020_MIMXRT1024.FLEXPWM_SetDTCNT1 (*C function*), 486
drivers_RT_RT1020_MIMXRT1024.FLEXPWM_SetINIT (*C function*), 483
drivers_RT_RT1020_MIMXRT1024.FLEXPWM_SetLoadOkay (*C function*), 491
drivers_RT_RT1020_MIMXRT1024.FLEXPWM_SetLocalForceOut (*C function*), 493
drivers_RT_RT1020_MIMXRT1024.FLEXPWM_SetPhaseDelay (*C function*), 486
drivers_RT_RT1020_MIMXRT1024.FLEXPWM_SetPWMSource23 (*C function*), 488
drivers_RT_RT1020_MIMXRT1024.FLEXPWM_SetPWMSource45 (*C function*), 489
drivers_RT_RT1020_MIMXRT1024.FLEXPWM_SetSoftwareControlOutput23 (*C function*), 489
drivers_RT_RT1020_MIMXRT1024.FLEXPWM_SetSoftwareControlOutput45 (*C function*), 490
drivers_RT_RT1020_MIMXRT1024.FLEXPWM_SetVAL0 (*C function*), 484
drivers_RT_RT1020_MIMXRT1024.FLEXPWM_SetVAL1 (*C function*), 484
drivers_RT_RT1020_MIMXRT1024.FLEXPWM_SetVAL2 (*C function*), 484
drivers_RT_RT1020_MIMXRT1024.FLEXPWM_SetVAL3 (*C function*), 485
drivers_RT_RT1020_MIMXRT1024.FLEXPWM_SetVAL4 (*C function*), 485
drivers_RT_RT1020_MIMXRT1024.FLEXPWM_SetVAL5 (*C function*), 485
drivers_RT_RT1020_MIMXRT1024.FLEXPWM_SetWriteProtection (*C function*), 496
drivers_RT_RT1020_MIMXRT1024.flexpwm_stretch_prescaler_t (*C type*), 517
drivers_RT_RT1020_MIMXRT1024.flexpwm_submodule_config_t (*C type*), 514
drivers_RT_RT1020_MIMXRT1024.flexpwm_submodule_mask_t (*C type*), 514
drivers_RT_RT1020_MIMXRT1024.flexpwm_trigger_frequency_t (*C type*), 517
drivers_RT_RT1020_MIMXRT1024.flexpwm_trigger_mux_source_t (*C type*), 517
drivers_RT_RT1020_MIMXRT1024.FLEXPWM_UnMaskPWMOutput (*C function*), 488
drivers_RT_RT1020_MIMXRT1024.flexpwm_write_protect_t (*C type*), 516
drivers_RT_RT1020_MIMXRT1024.FLEXRAM_ClearInterruptStatus (*C function*), 531
drivers_RT_RT1020_MIMXRT1024.FLEXRAM_CLOCKS (*C macro*), 236
drivers_RT_RT1020_MIMXRT1024.FLEXRAM_Deinit (*C function*), 530
drivers_RT_RT1020_MIMXRT1024.FLEXRAM_DisableInterruptSignal (*C function*), 531
drivers_RT_RT1020_MIMXRT1024.FLEXRAM_DisableInterruptStatus (*C function*), 531
drivers_RT_RT1020_MIMXRT1024.flexram_dtcm_ecc_multi_error_info_t (*C type*), 534
drivers_RT_RT1020_MIMXRT1024.flexram_dtcm_ecc_single_error_info_t (*C type*), 534
drivers_RT_RT1020_MIMXRT1024.FLEXRAM_ECC_ERROR_DETAILED_INFO (*C macro*), 536
drivers_RT_RT1020_MIMXRT1024.flexram_ecc_error_type_t (*C type*), 534
drivers_RT_RT1020_MIMXRT1024.FLEXRAM_EnableECC (*C function*), 535
drivers_RT_RT1020_MIMXRT1024.FLEXRAM_EnableForceRamClockOn (*C function*), 534
drivers_RT_RT1020_MIMXRT1024.FLEXRAM_EnableInterruptSignal (*C function*), 531
drivers_RT_RT1020_MIMXRT1024.FLEXRAM_EnableInterruptStatus (*C function*), 531
drivers_RT_RT1020_MIMXRT1024.FLEXRAM_ErrorInjection (*C function*), 535
drivers_RT_RT1020_MIMXRT1024.FLEXRAM_GetDtcmMultiErroInfo (*C function*), 536
drivers_RT_RT1020_MIMXRT1024.FLEXRAM_GetDtcmSingleErroInfo (*C function*), 536
drivers_RT_RT1020_MIMXRT1024.FLEXRAM_GetInterruptStatus (*C function*), 531
drivers_RT_RT1020_MIMXRT1024.FLEXRAM_GetItcmMultiErroInfo (*C function*), 536
drivers_RT_RT1020_MIMXRT1024.FLEXRAM_GetItcmSingleErroInfo (*C function*), 536
drivers_RT_RT1020_MIMXRT1024.FLEXRAM_GetOcramMultiErroInfo (*C function*), 536
drivers_RT_RT1020_MIMXRT1024.FLEXRAM_GetOcramSingleErroInfo (*C function*), 535
drivers_RT_RT1020_MIMXRT1024.FLEXRAM_Init (*C function*), 530
drivers_RT_RT1020_MIMXRT1024.flexram_itcm_ecc_multi_error_info_t (*C type*), 534
drivers_RT_RT1020_MIMXRT1024.flexram_itcm_ecc_single_error_info_t (*C type*), 534
drivers_RT_RT1020_MIMXRT1024.flexram_memory_type_t (*C type*), 533
drivers_RT_RT1020_MIMXRT1024.flexram_ocram_ecc_multi_error_info_t (*C type*), 534

drivers_RT_RT1020_MIMXRT1024.flexram_ocram_ecc_single_error_info_t (*C type*), 534
drivers_RT_RT1020_MIMXRT1024.FLEXRAM_SetDTCMMagicAddr (*C function*), 535
drivers_RT_RT1020_MIMXRT1024.FLEXRAM_SetITCMMagicAddr (*C function*), 535
drivers_RT_RT1020_MIMXRT1024.FLEXRAM_SetOCRAMMagicAddr (*C function*), 534
drivers_RT_RT1020_MIMXRT1024.FLEXRAM_SetTCMReadAccessMode (*C function*), 534
drivers_RT_RT1020_MIMXRT1024.FLEXRAM_SetTCMWriteAccessMode (*C function*), 534
drivers_RT_RT1020_MIMXRT1024.flexram_tcm_access_mode_t (*C type*), 533
drivers_RT_RT1020_MIMXRT1024.flexspi_addr_map_config_t (*C type*), 553
drivers_RT_RT1020_MIMXRT1024.flexspi_ahb_error_code_t (*C type*), 553
drivers_RT_RT1020_MIMXRT1024.flexspi_ahb_write_wait_unit_t (*C type*), 553
drivers_RT_RT1020_MIMXRT1024.flexspi_ahbBuffer_config_t (*C type*), 553
drivers_RT_RT1020_MIMXRT1024.flexspi_ahbBuffers_ctrl_t (*C type*), 553
drivers_RT_RT1020_MIMXRT1024.flexspi_arb_command_source_t (*C type*), 553
drivers_RT_RT1020_MIMXRT1024.FLEXSPI_CheckAndClearError (*C function*), 539
drivers_RT_RT1020_MIMXRT1024.FLEXSPI_ClearInterruptStatusFlags (*C function*), 542
drivers_RT_RT1020_MIMXRT1024.FLEXSPI_CLOCKS (*C macro*), 236
drivers_RT_RT1020_MIMXRT1024.flexspi_command_type_t (*C type*), 553
drivers_RT_RT1020_MIMXRT1024.flexspi_config_t (*C type*), 553
drivers_RT_RT1020_MIMXRT1024.flexspi_cs_interval_cycle_unit_t (*C type*), 553
drivers_RT_RT1020_MIMXRT1024.FLEXSPI_Deinit (*C function*), 540
drivers_RT_RT1020_MIMXRT1024.flexspi_device_config_t (*C type*), 553
drivers_RT_RT1020_MIMXRT1024.FLEXSPI_DisableInterrupts (*C function*), 541
drivers_RT_RT1020_MIMXRT1024.flexspi_edma_callback_t (*C type*), 561
drivers_RT_RT1020_MIMXRT1024.flexspi_edma_handle_t (*C type*), 560
drivers_RT_RT1020_MIMXRT1024.flexspi_edma_transfer_nsize_t (*C type*), 561
drivers_RT_RT1020_MIMXRT1024.FLEXSPI_Enable (*C function*), 540
drivers_RT_RT1020_MIMXRT1024.FLEXSPI_EnableInterrupts (*C function*), 541
drivers_RT_RT1020_MIMXRT1024.FLEXSPI_EnableRxDMA (*C function*), 541
drivers_RT_RT1020_MIMXRT1024.FLEXSPI_EnableTxDMA (*C function*), 541
drivers_RT_RT1020_MIMXRT1024.FLEXSPI_EXSC_CLOCKS (*C macro*), 236
drivers_RT_RT1020_MIMXRT1024.flexspi_flags_t (*C type*), 552
drivers_RT_RT1020_MIMXRT1024.FLEXSPI_GetAHBCommandErrorCode (*C function*), 543
drivers_RT_RT1020_MIMXRT1024.FLEXSPI_GetArbitratorCommandSource (*C function*), 542
drivers_RT_RT1020_MIMXRT1024.FLEXSPI_GetBusIdleStatus (*C function*), 543
drivers_RT_RT1020_MIMXRT1024.FLEXSPI_GetDefaultConfig (*C function*), 540
drivers_RT_RT1020_MIMXRT1024.FLEXSPI_GetFifoCounts (*C function*), 542
drivers_RT_RT1020_MIMXRT1024.FLEXSPI_GetInstance (*C function*), 539
drivers_RT_RT1020_MIMXRT1024.FLEXSPI_GetInterruptStatusFlags (*C function*), 542
drivers_RT_RT1020_MIMXRT1024.FLEXSPI_GetIPCommandErrorCode (*C function*), 542
drivers_RT_RT1020_MIMXRT1024.FLEXSPI_GetRxFifoAddress (*C function*), 541
drivers_RT_RT1020_MIMXRT1024.FLEXSPI_GetTxFifoAddress (*C function*), 541
drivers_RT_RT1020_MIMXRT1024.flexspi_handle_t (*C type*), 553
drivers_RT_RT1020_MIMXRT1024.FLEXSPI_Init (*C function*), 539
drivers_RT_RT1020_MIMXRT1024.flexspi_ip_error_code_t (*C type*), 553
drivers_RT_RT1020_MIMXRT1024.FLEXSPI_LUT_SEQ (*C macro*), 554
drivers_RT_RT1020_MIMXRT1024.flexspi_pad_t (*C type*), 552
drivers_RT_RT1020_MIMXRT1024.flexspi_port_t (*C type*), 553
drivers_RT_RT1020_MIMXRT1024.flexspi_read_sample_clock_t (*C type*), 553
drivers_RT_RT1020_MIMXRT1024.FLEXSPI_ReadBlocking (*C function*), 544
drivers_RT_RT1020_MIMXRT1024.FLEXSPI_ReadData (*C function*), 543
drivers_RT_RT1020_MIMXRT1024.FLEXSPI_ResetFifos (*C function*), 542
drivers_RT_RT1020_MIMXRT1024.FLEXSPI_SetFlashConfig (*C function*), 540
drivers_RT_RT1020_MIMXRT1024.FLEXSPI_SoftwareReset (*C function*), 540
drivers_RT_RT1020_MIMXRT1024.flexspi_transfer_callback_t (*C type*), 553
drivers_RT_RT1020_MIMXRT1024.flexspi_transfer_t (*C type*), 553
drivers_RT_RT1020_MIMXRT1024.FLEXSPI_TransferAbort (*C function*), 546
drivers_RT_RT1020_MIMXRT1024.FLEXSPI_TransferAbortEDMA (*C function*), 560
drivers_RT_RT1020_MIMXRT1024.FLEXSPI_TransferBlocking (*C function*), 544

drivers_RT_RT1020_MIMXRT1024.FLEXSPI_TransferCreateHandle (*C function*), 545
drivers_RT_RT1020_MIMXRT1024.FLEXSPI_TransferCreateHandleEDMA (*C function*), 559
drivers_RT_RT1020_MIMXRT1024.FLEXSPI_TransferEDMA (*C function*), 559
drivers_RT_RT1020_MIMXRT1024.FLEXSPI_TransferGetCount (*C function*), 545
drivers_RT_RT1020_MIMXRT1024.FLEXSPI_TransferGetTransferCountEDMA (*C function*), 560
drivers_RT_RT1020_MIMXRT1024.FLEXSPI_TransferHandleIRQ (*C function*), 546
drivers_RT_RT1020_MIMXRT1024.FLEXSPI_TransferNonBlocking (*C function*), 545
drivers_RT_RT1020_MIMXRT1024.FLEXSPI_TransferUpdateSizeEDMA (*C function*), 559
drivers_RT_RT1020_MIMXRT1024.FLEXSPI_UpdateAhbBuffersSettings (*C function*), 541
drivers_RT_RT1020_MIMXRT1024.FLEXSPI_UpdateDllValue (*C function*), 540
drivers_RT_RT1020_MIMXRT1024.FLEXSPI_UpdateLUT (*C function*), 543
drivers_RT_RT1020_MIMXRT1024.FLEXSPI_UpdateRxSampleClock (*C function*), 543
drivers_RT_RT1020_MIMXRT1024.FLEXSPI_WriteBlocking (*C function*), 544
drivers_RT_RT1020_MIMXRT1024.FLEXSPI_WriteData (*C function*), 543
drivers_RT_RT1020_MIMXRT1024.FSL_ADC_DRIVER_VERSION (*C macro*), 180
drivers_RT_RT1020_MIMXRT1024.FSL_ADC_ETC_DRIVER_VERSION (*C macro*), 186
drivers_RT_RT1020_MIMXRT1024.FSL_AIPSTZ_DRIVER_VERSION (*C macro*), 187
drivers_RT_RT1020_MIMXRT1024.FSL_AOI_DRIVER_VERSION (*C macro*), 191
drivers_RT_RT1020_MIMXRT1024.FSL_BEE_DRIVER_VERSION (*C macro*), 193
drivers_RT_RT1020_MIMXRT1024.FSL_CACHE_DRIVER_VERSION (*C macro*), 201
drivers_RT_RT1020_MIMXRT1024.FSL_CLOCK_DRIVER_VERSION (*C macro*), 233
drivers_RT_RT1020_MIMXRT1024.FSL_CMP_DRIVER_VERSION (*C macro*), 247
drivers_RT_RT1020_MIMXRT1024.FSL_COMMON_DRIVER_VERSION (*C macro*), 608
drivers_RT_RT1020_MIMXRT1024.FSL_COMPONENT_ID (*C macro*), 605
drivers_RT_RT1020_MIMXRT1024.FSL_DCDC_DRIVER_VERSION (*C macro*), 258
drivers_RT_RT1020_MIMXRT1024.FSL_DCP_DRIVER_VERSION (*C macro*), 261
drivers_RT_RT1020_MIMXRT1024.FSL_DMAMUX_DRIVER_VERSION (*C macro*), 273
drivers_RT_RT1020_MIMXRT1024.FSL_DRIVER_TRANSFER_DOUBLE_WEAK_IRQ (*C macro*), 624
drivers_RT_RT1020_MIMXRT1024.FSL_EDMA_DRIVER_VERSION (*C macro*), 285
drivers_RT_RT1020_MIMXRT1024.FSL_ENC_DRIVER_VERSION (*C macro*), 305
drivers_RT_RT1020_MIMXRT1024.FSL_ENET_DRIVER_VERSION (*C macro*), 320
drivers_RT_RT1020_MIMXRT1024.FSL_EWM_DRIVER_VERSION (*C macro*), 343
drivers_RT_RT1020_MIMXRT1024.FSL_FEATURE_FLEXSPI_AHB_BUFFER_COUNT (*C macro*), 553
drivers_RT_RT1020_MIMXRT1024.FSL_FLEXCAN_DRIVER_VERSION (*C macro*), 364
drivers_RT_RT1020_MIMXRT1024.FSL_FLEXIO_DRIVER_VERSION (*C macro*), 402
drivers_RT_RT1020_MIMXRT1024.FSL_FLEXIO_I2C_MASTER_DRIVER_VERSION (*C macro*), 430
drivers_RT_RT1020_MIMXRT1024.FSL_FLEXIO_I2S_DRIVER_VERSION (*C macro*), 440
drivers_RT_RT1020_MIMXRT1024.FSL_FLEXIO_I2S_EDMA_DRIVER_VERSION (*C macro*), 416
drivers_RT_RT1020_MIMXRT1024.FSL_FLEXIO_SPI_DRIVER_VERSION (*C macro*), 453
drivers_RT_RT1020_MIMXRT1024.FSL_FLEXIO_SPI_EDMA_DRIVER_VERSION (*C macro*), 420
drivers_RT_RT1020_MIMXRT1024.FSL_FLEXIO_UART_DRIVER_VERSION (*C macro*), 466
drivers_RT_RT1020_MIMXRT1024.FSL_FLEXIO_UART_EDMA_DRIVER_VERSION (*C macro*), 423
drivers_RT_RT1020_MIMXRT1024.FSL_FLEXPWM_DRIVER_VERSION (*C macro*), 500
drivers_RT_RT1020_MIMXRT1024.FSL_FLEXRAM_DRIVER_VERSION (*C macro*), 531
drivers_RT_RT1020_MIMXRT1024.FSL_FLEXSPI_DRIVER_VERSION (*C macro*), 546
drivers_RT_RT1020_MIMXRT1024.FSL_FLEXSPI_EDMA_DRIVER_VERSION (*C macro*), 560
drivers_RT_RT1020_MIMXRT1024.FSL_GPC_DRIVER_VERSION (*C macro*), 561
drivers_RT_RT1020_MIMXRT1024.FSL_GPIO_DRIVER_VERSION (*C macro*), 566
drivers_RT_RT1020_MIMXRT1024.FSL_GPT_DRIVER_VERSION (*C macro*), 572
drivers_RT_RT1020_MIMXRT1024.FSL_IOMUXC_DRIVER_VERSION (*C macro*), 605
drivers_RT_RT1020_MIMXRT1024.FSL_KPP_DRIVER_VERSION (*C macro*), 607
drivers_RT_RT1020_MIMXRT1024.FSL_LPI2C_DRIVER_VERSION (*C macro*), 625

drivers_RT_RT1020_MIMXRT1024.FSL_LPSPI_DRIVER_VERSION (*C macro*), 666
drivers_RT_RT1020_MIMXRT1024.FSL_LPSPI_EDMA_DRIVER_VERSION (*C macro*), 680
drivers_RT_RT1020_MIMXRT1024.FSL_LPUART_DRIVER_VERSION (*C macro*), 699
drivers_RT_RT1020_MIMXRT1024.FSL_LPUART_EDMA_DRIVER_VERSION (*C macro*), 711
drivers_RT_RT1020_MIMXRT1024.FSL_OCOTP_DRIVER_VERSION (*C macro*), 712
drivers_RT_RT1020_MIMXRT1024.FSL_PIT_DRIVER_VERSION (*C macro*), 719
drivers_RT_RT1020_MIMXRT1024.FSL_PMU_DRIVER_VERSION (*C macro*), 725
drivers_RT_RT1020_MIMXRT1024.FSL_PWM_DRIVER_VERSION (*C macro*), 732
drivers_RT_RT1020_MIMXRT1024.FSL_QTMR_DRIVER_VERSION (*C macro*), 760
drivers_RT_RT1020_MIMXRT1024.FSL_RTWDOG_DRIVER_VERSION (*C macro*), 772
drivers_RT_RT1020_MIMXRT1024.FSL_SAI_DRIVER_VERSION (*C macro*), 790
drivers_RT_RT1020_MIMXRT1024.FSL_SAI_EDMA_DRIVER_VERSION (*C macro*), 810
drivers_RT_RT1020_MIMXRT1024.FSL_SAI_HAS_FIFO_EXTEND_FEATURE (*C macro*), 799
drivers_RT_RT1020_MIMXRT1024.FSL_SDK_DISABLE_DRIVER_CLOCK_CONTROL (*C macro*), 243
drivers_RT_RT1020_MIMXRT1024.FSL_SEMC_DRIVER_VERSION (*C macro*), 816
drivers_RT_RT1020_MIMXRT1024.FSL_SNVS_HP_DRIVER_VERSION (*C macro*), 846
drivers_RT_RT1020_MIMXRT1024.FSL_SNVS_LP_DRIVER_VERSION (*C macro*), 857
drivers_RT_RT1020_MIMXRT1024.FSL_SPDIF_DRIVER_VERSION (*C macro*), 870
drivers_RT_RT1020_MIMXRT1024.FSL_SPDIF_EDMA_DRIVER_VERSION (*C macro*), 879
drivers_RT_RT1020_MIMXRT1024.FSL_SRC_DRIVER_VERSION (*C macro*), 881
drivers_RT_RT1020_MIMXRT1024.FSL_TEMPMON_DRIVER_VERSION (*C macro*), 887
drivers_RT_RT1020_MIMXRT1024.FSL_TRNG_DRIVER_VERSION (*C macro*), 889
drivers_RT_RT1020_MIMXRT1024.FSL_USDHC_DRIVER_VERSION (*C macro*), 905
drivers_RT_RT1020_MIMXRT1024.FSL_USDHC_ENABLE_SCATTER_GATHER_TRANSFER (*C macro*), 921
drivers_RT_RT1020_MIMXRT1024.FSL_WDOG_DRIVER_VERSION (*C macro*), 932
drivers_RT_RT1020_MIMXRT1024.FSL_XBARA_DRIVER_VERSION (*C macro*), 936
drivers_RT_RT1020_MIMXRT1024.FSL_XBARB_DRIVER_VERSION (*C macro*), 937
drivers_RT_RT1020_MIMXRT1024.g_lpspiDummyData (*C var*), 675
drivers_RT_RT1020_MIMXRT1024.g_rtcXtalFreq (*C var*), 231
drivers_RT_RT1020_MIMXRT1024.g_xtalFreq (*C var*), 231
drivers_RT_RT1020_MIMXRT1024.GPC_AllowIRQs (*C function*), 561
drivers_RT_RT1020_MIMXRT1024.GPC_DisableIRQ (*C function*), 562
drivers_RT_RT1020_MIMXRT1024.GPC_DisallowIRQs (*C function*), 562
drivers_RT_RT1020_MIMXRT1024.GPC_EnableIRQ (*C function*), 562
drivers_RT_RT1020_MIMXRT1024.GPC_GetIRQStatusFlag (*C function*), 562
drivers_RT_RT1020_MIMXRT1024.GPC_GetVADCPowerDownFlag (*C function*), 563
drivers_RT_RT1020_MIMXRT1024.GPC_HasDVFS0ChangeRequest (*C function*), 563
drivers_RT_RT1020_MIMXRT1024.GPC_RequestDisplayPowerOn (*C function*), 563
drivers_RT_RT1020_MIMXRT1024.GPC_RequestL2CachePowerDown (*C function*), 562
drivers_RT_RT1020_MIMXRT1024.GPC_RequestMEGAPowerOn (*C function*), 563
drivers_RT_RT1020_MIMXRT1024.GPC_RequestPdram0PowerDown (*C function*), 562
drivers_RT_RT1020_MIMXRT1024.GPC_RequestVADCPowerDown (*C function*), 563
drivers_RT_RT1020_MIMXRT1024.GPIO_ClearPinsInterruptFlags (*C function*), 566
drivers_RT_RT1020_MIMXRT1024.GPIO_ClearPinsOutput (*C function*), 564
drivers_RT_RT1020_MIMXRT1024.GPIO_CLOCKS (*C macro*), 236
drivers_RT_RT1020_MIMXRT1024.GPIO_DisableInterrupts (*C function*), 566
drivers_RT_RT1020_MIMXRT1024.GPIO_EnableInterrupts (*C function*), 566
drivers_RT_RT1020_MIMXRT1024.GPIO_GetPinsInterruptFlags (*C function*), 566
drivers_RT_RT1020_MIMXRT1024.gpio_interrupt_mode_t (*C type*), 567
drivers_RT_RT1020_MIMXRT1024.gpio_pin_config_t (*C type*), 567
drivers_RT_RT1020_MIMXRT1024.gpio_pin_direction_t (*C type*), 567
drivers_RT_RT1020_MIMXRT1024.GPIO_PinInit (*C function*), 563
drivers_RT_RT1020_MIMXRT1024.GPIO_PinRead (*C function*), 564
drivers_RT_RT1020_MIMXRT1024.GPIO_PinReadPadStatus (*C function*), 565
drivers_RT_RT1020_MIMXRT1024.GPIO_PinSetInterruptConfig (*C function*), 565
drivers_RT_RT1020_MIMXRT1024.GPIO_PinWrite (*C function*), 563

drivers_RT_RT1020_MIMXRT1024.GPIO_PortClear (*C function*), 564
drivers_RT_RT1020_MIMXRT1024.GPIO_PortClearInterruptFlags (*C function*), 566
drivers_RT_RT1020_MIMXRT1024.GPIO_PortDisableInterrupts (*C function*), 566
drivers_RT_RT1020_MIMXRT1024.GPIO_PortEnableInterrupts (*C function*), 565
drivers_RT_RT1020_MIMXRT1024.GPIO_PortGetInterruptFlags (*C function*), 566
drivers_RT_RT1020_MIMXRT1024.GPIO_PortSet (*C function*), 564
drivers_RT_RT1020_MIMXRT1024.GPIO_PortToggle (*C function*), 564
drivers_RT_RT1020_MIMXRT1024.GPIO_ReadPadStatus (*C function*), 565
drivers_RT_RT1020_MIMXRT1024.GPIO_ReadPinInput (*C function*), 565
drivers_RT_RT1020_MIMXRT1024.GPIO_SetPinInterruptConfig (*C function*), 565
drivers_RT_RT1020_MIMXRT1024.GPIO_SetPinsOutput (*C function*), 564
drivers_RT_RT1020_MIMXRT1024.GPIO_WritePinOutput (*C function*), 564
drivers_RT_RT1020_MIMXRT1024.GPT_ClearStatusFlags (*C function*), 572
drivers_RT_RT1020_MIMXRT1024.gpt_clock_source_t (*C type*), 575
drivers_RT_RT1020_MIMXRT1024.GPT_CLOCKS (*C macro*), 236
drivers_RT_RT1020_MIMXRT1024.gpt_config_t (*C type*), 575
drivers_RT_RT1020_MIMXRT1024.GPT_Deinit (*C function*), 568
drivers_RT_RT1020_MIMXRT1024.GPT_DisableInterrupts (*C function*), 571
drivers_RT_RT1020_MIMXRT1024.GPT_EnableInterrupts (*C function*), 571
drivers_RT_RT1020_MIMXRT1024.GPT_ForceOutput (*C function*), 571
drivers_RT_RT1020_MIMXRT1024.GPT_GetClockDivider (*C function*), 569
drivers_RT_RT1020_MIMXRT1024.GPT_GetClockSource (*C function*), 569
drivers_RT_RT1020_MIMXRT1024.GPT_GetCurrentTimerCount (*C function*), 569
drivers_RT_RT1020_MIMXRT1024.GPT_GetDefaultConfig (*C function*), 568
drivers_RT_RT1020_MIMXRT1024.GPT_GetEnabledInterrupts (*C function*), 571
drivers_RT_RT1020_MIMXRT1024.GPT_GetInputCaptureValue (*C function*), 570
drivers_RT_RT1020_MIMXRT1024.GPT_GetInputOperationMode (*C function*), 570
drivers_RT_RT1020_MIMXRT1024.GPT_GetOscClockDivider (*C function*), 569
drivers_RT_RT1020_MIMXRT1024.GPT_GetOutputCompareValue (*C function*), 571
drivers_RT_RT1020_MIMXRT1024.GPT_GetOutputOperationMode (*C function*), 570
drivers_RT_RT1020_MIMXRT1024.GPT_GetStatusFlags (*C function*), 572
drivers_RT_RT1020_MIMXRT1024.GPT_Init (*C function*), 568
drivers_RT_RT1020_MIMXRT1024.gpt_input_capture_channel_t (*C type*), 575
drivers_RT_RT1020_MIMXRT1024.gpt_input_operation_mode_t (*C type*), 575
drivers_RT_RT1020_MIMXRT1024.gpt_interrupt_enable_t (*C type*), 575
drivers_RT_RT1020_MIMXRT1024.gpt_output_compare_channel_t (*C type*), 575
drivers_RT_RT1020_MIMXRT1024.gpt_output_operation_mode_t (*C type*), 575
drivers_RT_RT1020_MIMXRT1024.GPT_SetClockDivider (*C function*), 569
drivers_RT_RT1020_MIMXRT1024.GPT_SetClockSource (*C function*), 568
drivers_RT_RT1020_MIMXRT1024.GPT_SetInputOperationMode (*C function*), 570
drivers_RT_RT1020_MIMXRT1024.GPT_SetOscClockDivider (*C function*), 569
drivers_RT_RT1020_MIMXRT1024.GPT_SetOutputCompareValue (*C function*), 571
drivers_RT_RT1020_MIMXRT1024.GPT_SetOutputOperationMode (*C function*), 570
drivers_RT_RT1020_MIMXRT1024.GPT_SoftwareReset (*C function*), 568
drivers_RT_RT1020_MIMXRT1024.GPT_StartTimer (*C function*), 569
drivers_RT_RT1020_MIMXRT1024.gpt_status_flag_t (*C type*), 575
drivers_RT_RT1020_MIMXRT1024.GPT_StopTimer (*C function*), 569
drivers_RT_RT1020_MIMXRT1024.I2C_RETRY_TIMES (*C macro*), 431, 626
drivers_RT_RT1020_MIMXRT1024.I2S_RETRY_TIMES (*C macro*), 442
drivers_RT_RT1020_MIMXRT1024.ICACHE_InvalidateByRange (*C function*), 200
drivers_RT_RT1020_MIMXRT1024.IOMUXC_EnableMode (*C function*), 604
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_00_GPIO1_IO00 (*C macro*), 590
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_00_GPT1_COMPARE1 (*C macro*), 590
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_00_JTAG_TMS (*C macro*), 590
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_01_GPIO1_IO01 (*C macro*), 590
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_01_GPT1_CAPTURE2 (*C macro*), 590

drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_01_JTAG_TCK (*C macro*), 590
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_02_GPIO1_IO02 (*C macro*), 590
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_02_GPT1_CAPTURE1 (*C macro*), 590
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_02_JTAG_MOD (*C macro*), 590
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_03_CCM_PMIC_RDY (*C macro*), 590
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_03_GPIO1_IO03 (*C macro*), 590
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_03_JTAG_TDI (*C macro*), 590
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_03_SAI1_MCLK (*C macro*), 590
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_03_USB_OTG1_OC (*C macro*), 590
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_03_USDHC1_WP (*C macro*), 590
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_03_USDHC2_CD_B (*C macro*), 590
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_03_WDOG1_B (*C macro*), 590
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_04_ENET_MDIO (*C macro*), 591
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_04_EWM_OUT_B (*C macro*), 591
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_04_FLEXCAN1_TX (*C macro*), 591
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_04_GPIO1_IO04 (*C macro*), 591
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_04_JTAG_TDO (*C macro*), 590
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_04_QTIMER2_TIMER0 (*C macro*), 591
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_04_USB_OTG1_PWR (*C macro*), 591
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_04_USDHC1_WP (*C macro*), 591
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_05_ENET_MDC (*C macro*), 591
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_05_FLEXCAN1_RX (*C macro*), 591
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_05_GPIO1_IO05 (*C macro*), 591
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_05_JTAG_TRSTB (*C macro*), 591
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_05_NMI_GLUE_NMI (*C macro*), 591
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_05_QTIMER2_TIMER1 (*C macro*), 591
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_05_USB_OTG1_ID (*C macro*), 591
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_05_USDHC1_CD_B (*C macro*), 591
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_06_FLEXPWM2_PWMA03 (*C macro*), 591
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_06_GPIO1_IO06 (*C macro*), 591
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_06_LPUART1_TX (*C macro*), 591
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_06_MQS_RIGHT (*C macro*), 591
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_06_PIT_TRIGGER00 (*C macro*), 591
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_06_QTIMER2_TIMER2 (*C macro*), 591
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_06_REF_32K_OUT (*C macro*), 591
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_07_FLEXPWM2_PWMB03 (*C macro*), 592
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_07_GPIO1_IO07 (*C macro*), 592
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_07_LPUART1_RX (*C macro*), 592
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_07_MQS_LEFT (*C macro*), 592
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_07_PIT_TRIGGER01 (*C macro*), 591
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_07_QTIMER2_TIMER3 (*C macro*), 592
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_07_REF_24M_OUT (*C macro*), 592

drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_08_ARM_CM7_TXEV (*C macro*), 592
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_08_ENET_REF_CLK1 (*C macro*), 592
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_08_ENET_TX_CLK (*C macro*), 592
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_08_GPIO1_IO08 (*C macro*), 592
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_08_KPP_COL00 (*C macro*), 592
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_08_LPI2C3_SCL (*C macro*), 592
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_08_LPUART1_CTS_B (*C macro*), 592
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_09_ARM_CM7_RXEV (*C macro*), 592
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_09_ENET_RDATA01 (*C macro*), 592
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_09_GPIO1_IO09 (*C macro*), 592
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_09_KPP_ROW00 (*C macro*), 592
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_09_LPI2C3_SDA (*C macro*), 592
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_09_LPUART1_RTS_B (*C macro*), 592
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_10_ARM_CM7_TRACE_CLK (*C macro*), 593
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_10_ENET_RDATA00 (*C macro*), 592
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_10_FLEXPWM2_PWMA02 (*C macro*), 593
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_10_GPIO1_IO10 (*C macro*), 593
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_10_KPP_COL01 (*C macro*), 592
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_10_LPSPI1_SCK (*C macro*), 592
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_10_LPUART5_TX (*C macro*), 592
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_11_ARM_CM7_TRACE_SWO (*C macro*), 593
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_11_ENET_RX_EN (*C macro*), 593
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_11_FLEXPWM2_PWMB02 (*C macro*), 593
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_11_GPIO1_IO11 (*C macro*), 593
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_11_KPP_ROW01 (*C macro*), 593
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_11_LPSPI1_PCS0 (*C macro*), 593
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_11_LPUART5_RX (*C macro*), 593
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_12_ARM_CM7_TRACE00 (*C macro*), 593
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_12_ENET_RX_ER (*C macro*), 593
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_12_FLEXPWM2_PWMA01 (*C macro*), 593
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_12_GPIO1_IO12 (*C macro*), 593
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_12_KPP_COL02 (*C macro*), 593
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_12_LPSPI1_SDO (*C macro*), 593
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_12_LPUART3_CTS_B (*C macro*), 593
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_12_SNVS_HP_VIO_5_CTL (*C macro*), 593
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_13_ARM_CM7_TRACE01 (*C macro*), 594
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_13_ENET_TX_EN (*C macro*), 593
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_13_FLEXPWM2_PWMB01 (*C macro*), 593
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_13_GPIO1_IO13 (*C macro*), 594
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_13_KPP_ROW02 (*C macro*), 593

drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_13_LPSPI1_SDI (*C macro*), 593
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_13_LPUART3_RTS_B (*C macro*), 593
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_13_SNVS_HP_VIO_5_B (*C macro*), 594
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_14_ARM_CM7_TRACE02 (*C macro*), 594
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_14_ENET_TDATA00 (*C macro*), 594
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_14_FLEXCAN2_TX (*C macro*), 594
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_14_FLEXPWM2_PWMA00 (*C macro*), 594
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_14_GPIO1_IO14 (*C macro*), 594
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_14_KPP_COL03 (*C macro*), 594
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_14_LPUART3_TX (*C macro*), 594
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_14_WDOG1_ANY (*C macro*), 594
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_15_ARM_CM7_TRACE03 (*C macro*), 594
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_15_ENET_TDATA01 (*C macro*), 594
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_15_FLEXCAN2_RX (*C macro*), 594
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_15_FLEXPWM2_PWMB00 (*C macro*), 594
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_15_GPIO1_IO15 (*C macro*), 594
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_15_KPP_ROW03 (*C macro*), 594
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B0_15_LPUART3_RX (*C macro*), 594
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B1_06_FLEXIO1_FLEXIO09 (*C macro*), 594
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B1_06_FLEXPWM1_PWMA00 (*C macro*), 594
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B1_06_GPIO1_IO22 (*C macro*), 595
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B1_06_KPP_COL07 (*C macro*), 595
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B1_06_LPSPI1_PCS3 (*C macro*), 595
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B1_06_LPUART2_CTS_B (*C macro*), 594
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B1_06_SAI1_RX_BCLK (*C macro*), 594
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B1_06_USDHC1_RESET_B (*C macro*), 594
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B1_07_FLEXIO1_FLEXIO08 (*C macro*), 595
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B1_07_FLEXPWM1_PWMB00 (*C macro*), 595
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B1_07_GPIO1_IO23 (*C macro*), 595
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B1_07_KPP_ROW07 (*C macro*), 595
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B1_07_LPSPI3_PCS3 (*C macro*), 595
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B1_07_LPUART2_RTS_B (*C macro*), 595
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B1_07_SAI1_TX_DATA01 (*C macro*), 595
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B1_07_USDHC1_VSELECT (*C macro*), 595
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B1_08_FLEXIO1_FLEXIO07 (*C macro*), 595
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B1_08_FLEXPWM1_PWMA01 (*C macro*), 595
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B1_08_GPIO1_IO24 (*C macro*), 595
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B1_08_LPI2C2_SCL (*C macro*), 595

drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B1_08_LPSPI3_PCS2 (*C macro*), 595
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B1_08_LPUART2_TX (*C macro*), 595
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B1_08_SAI1_TX_DATA02 (*C macro*), 595
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B1_08_XBAR1_INOUT12 (*C macro*), 595
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B1_09_FLEXIO1_FLEXIO06 (*C macro*), 596
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B1_09_FLEXPWM1_PWMB01 (*C macro*), 595
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B1_09_GPIO1_IO25 (*C macro*), 596
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B1_09_LPI2C2_SDA (*C macro*), 595
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B1_09_LPSPI3_PCS1 (*C macro*), 596
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B1_09_LPUART2_RX (*C macro*), 595
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B1_09_SAI1_TX_DATA03 (*C macro*), 595
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B1_09_XBAR1_INOUT13 (*C macro*), 596
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B1_10_FLEXIO1_FLEXIO05 (*C macro*), 596
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B1_10_FLEXPWM1_PWMA02 (*C macro*), 596
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B1_10_GPIO1_IO26 (*C macro*), 596
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B1_10_GPT2_CAPTURE1 (*C macro*), 596
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B1_10_LPUART4_TX (*C macro*), 596
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B1_10_USB_OTG1_PWR (*C macro*), 596
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B1_10_USDHC1_CD_B (*C macro*), 596
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B1_11_FLEXIO1_FLEXIO04 (*C macro*), 596
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B1_11_FLEXPWM1_PWMB02 (*C macro*), 596
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B1_11_GPIO1_IO27 (*C macro*), 596
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B1_11_GPT2_COMPARE1 (*C macro*), 596
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B1_11_LPUART4_RX (*C macro*), 596
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B1_11_USB_OTG1_ID (*C macro*), 596
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B1_11_USDHC1_WP (*C macro*), 596
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B1_12_ACMP1_OUT (*C macro*), 596
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B1_12_FLEXIO1_FLEXIO03 (*C macro*), 596
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B1_12_FLEXPWM1_PWMA03 (*C macro*), 597
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B1_12_GPIO1_IO28 (*C macro*), 597
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B1_12_LPSPI3_SCK (*C macro*), 596
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B1_12_USB_OTG1_OC (*C macro*), 596
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B1_12_USDHC2_CD_B (*C macro*), 596
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B1_13_ACMP2_OUT (*C macro*), 597
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B1_13_FLEXIO1_FLEXIO02 (*C macro*), 597
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B1_13_FLEXPWM1_PWMB03 (*C macro*), 597
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B1_13_GPIO1_IO29 (*C macro*), 597
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B1_13_LPI2C1_HREQ (*C macro*), 597

drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B1_13_LPSPI3_PCS0 (*C macro*), 597
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B1_13_USDHC2_WP (*C macro*), 597
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B1_14_ACMP3_OUT (*C macro*), 597
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B1_14_ENET_1588_EVENT0_OUT (*C macro*), 597
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B1_14_FLEXIO1_FLEXIO01 (*C macro*), 597
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B1_14_GPIO1_IO30 (*C macro*), 597
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B1_14_LPI2C1_SCL (*C macro*), 597
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B1_14_LPSPI3_SDO (*C macro*), 597
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B1_15_ACMP4_OUT (*C macro*), 597
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B1_15_ENET_1588_EVENT0_IN (*C macro*), 597
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B1_15_FLEXIO1_FLEXIO00 (*C macro*), 597
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B1_15_GPIO1_IO31 (*C macro*), 597
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B1_15_LPI2C1_SDA (*C macro*), 597
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_AD_B1_15_LPSPI3_SDI (*C macro*), 597
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_00_FLEXCAN1_TX (*C macro*), 577
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_00_GPIO2_IO00 (*C macro*), 577
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_00_LPSPI2_SCK (*C macro*), 577
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_00_LPUART4_CTS_B (*C macro*), 577
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_00_PIT_TRIGGER02 (*C macro*), 577
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_00_QTIMER2_TIMER0 (*C macro*), 577
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_00_SEMC_DATA00 (*C macro*), 577
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_00_SPDIF_SR_CLK (*C macro*), 577
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_01_FLEXCAN1_RX (*C macro*), 577
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_01_GPIO2_IO01 (*C macro*), 577
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_01_LPSPI2_PCS0 (*C macro*), 577
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_01_LPUART4_RTS_B (*C macro*), 577
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_01_PIT_TRIGGER03 (*C macro*), 578
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_01_QTIMER2_TIMER1 (*C macro*), 577
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_01_SEMC_DATA01 (*C macro*), 577
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_01_SPDIF_OUT (*C macro*), 577
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_02_GPIO2_IO02 (*C macro*), 578
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_02_LPI2C1_SCL (*C macro*), 578
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_02_LPSPI2_SDO (*C macro*), 578
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_02_LPUART4_TX (*C macro*), 578
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_02_QTIMER2_TIMER2 (*C macro*), 578
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_02_SEMC_DATA02 (*C macro*), 578
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_02_SPDIF_LOCK (*C macro*), 578
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_03_GPIO2_IO03 (*C macro*), 578
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_03_LPI2C1_SDA (*C macro*), 578
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_03_LPSPI2_SDI (*C macro*), 578
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_03_LPUART4_RX (*C macro*), 578
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_03_QTIMER2_TIMER3 (*C macro*), 578
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_03_SEMC_DATA03 (*C macro*), 578
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_03_SPDIF_EXT_CLK (*C macro*), 578
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_04_FLEXIO1_FLEXIO16 (*C macro*), 578
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_04_GPIO2_IO04 (*C macro*), 578
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_04_SAI2_TX_BCLK (*C macro*), 578

drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_04_SEMC_DATA04 (*C macro*), 578
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_04_SPDIF_OUT (*C macro*), 578
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_04_XBAR1_INOUT04 (*C macro*), 578
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_05_FLEXIO1_FLEXIO17 (*C macro*), 579
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_05_GPIO2_IO05 (*C macro*), 579
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_05_SAI2_TX_SYNC (*C macro*), 579
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_05_SEMC_DATA05 (*C macro*), 578
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_05_SPDIF_IN (*C macro*), 579
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_05_XBAR1_INOUT05 (*C macro*), 578
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_06_FLEXIO1_FLEXIO18 (*C macro*), 579
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_06_GPIO2_IO06 (*C macro*), 579
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_06_LPUART3_TX (*C macro*), 579
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_06_SAI2_TX_DATA (*C macro*), 579
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_06_SEMC_DATA06 (*C macro*), 579
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_06_XBAR1_INOUT06 (*C macro*), 579
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_07_FLEXIO1_FLEXIO19 (*C macro*), 579
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_07_GPIO2_IO07 (*C macro*), 579
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_07_LPUART3_RX (*C macro*), 579
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_07_SAI2_RX_SYNC (*C macro*), 579
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_07_SEMC_DATA07 (*C macro*), 579
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_07_XBAR1_INOUT07 (*C macro*), 579
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_08_FLEXCAN2_TX (*C macro*), 579
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_08_FLEXIO1_FLEXIO20 (*C macro*), 579
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_08_GPIO2_IO08 (*C macro*), 579
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_08_SAI2_RX_DATA (*C macro*), 579
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_08_SEMC_DM00 (*C macro*), 579
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_08_XBAR1_INOUT08 (*C macro*), 579
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_09_FLEXCAN2_RX (*C macro*), 580
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_09_FLEXIO1_FLEXIO21 (*C macro*), 580
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_09_GPIO2_IO09 (*C macro*), 580
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_09_SAI2_RX_BCLK (*C macro*), 580
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_09_SEMC_WE (*C macro*), 579
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_09_XBAR1_INOUT09 (*C macro*), 580
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_10_FLEXPWM2_PWMX00 (*C macro*), 580
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_10_GPIO2_IO10 (*C macro*), 580
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_10_LPI2C4_SDA (*C macro*), 580
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_10_LPSPI2_SCK (*C macro*), 580
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_10_SAI1_TX_SYNC (*C macro*), 580
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_10_SEMC_CAS (*C macro*), 580
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_10_XBAR1_INOUT10 (*C macro*), 580
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_11_FLEXPWM2_PWMX01 (*C macro*), 580
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_11_GPIO2_IO11 (*C macro*), 580
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_11_LPI2C4_SCL (*C macro*), 580
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_11_LPSPI2_PCS0 (*C macro*), 580
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_11_SAI1_TX_BCLK (*C macro*), 580
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_11_SEMC_RAS (*C macro*), 580
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_11_XBAR1_INOUT11 (*C macro*), 580
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_12_FLEXPWM2_PWMX02 (*C macro*), 581
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_12_GPIO2_IO12 (*C macro*), 581
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_12_LPSPI2_SDO (*C macro*), 581

drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_12_LPUART6_TX (*C macro*), 580
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_12_SAI1_TX_DATA00 (*C macro*), 580
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_12_SEMC_CS0 (*C macro*), 580
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_12_XBAR1_INOUT12 (*C macro*), 580
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_13_CCM_PMIC_RDY (*C macro*), 581
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_13_FLEXPWM2_PWMX03 (*C macro*), 581
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_13_GPIO2_IO13 (*C macro*), 581
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_13_LPSPI2_SDI (*C macro*), 581
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_13_LPUART6_RX (*C macro*), 581
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_13_SAI1_RX_DATA00 (*C macro*), 581
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_13_SEMC_BA0 (*C macro*), 581
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_13_XBAR1_INOUT13 (*C macro*), 581
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_14_FLEXCAN1_TX (*C macro*), 581
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_14_GPIO2_IO14 (*C macro*), 581
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_14_LPSPI2_PCS1 (*C macro*), 581
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_14_LPUART6_CTS_B (*C macro*), 581
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_14_SAI1_RX_BCLK (*C macro*), 581
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_14_SEMC_BA1 (*C macro*), 581
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_14_XBAR1_INOUT14 (*C macro*), 581
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_15_FLEXCAN1_RX (*C macro*), 582
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_15_GPIO2_IO15 (*C macro*), 582
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_15_LPUART6_RTS_B (*C macro*), 581
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_15_SAI1_RX_SYNC (*C macro*), 581
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_15_SEMC_ADDR10 (*C macro*), 581
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_15_WDOG1_B (*C macro*), 581
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_15_XBAR1_INOUT15 (*C macro*), 581
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_16_GPIO2_IO16 (*C macro*), 582
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_16_MQS_RIGHT (*C macro*), 582
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_16_SAI2_MCLK (*C macro*), 582
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_16_SEMC_ADDR00 (*C macro*), 582
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_16_SRC_BOOT_MODE00 (*C macro*), 582
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_17_GPIO2_IO17 (*C macro*), 582
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_17_MQS_LEFT (*C macro*), 582
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_17_SAI3_MCLK (*C macro*), 582
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_17_SEMC_ADDR01 (*C macro*), 582
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_17_SRC_BOOT_MODE01 (*C macro*), 582
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_18_FLEXIO1_FLEXIO22 (*C macro*), 582
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_18_GPIO2_IO18 (*C macro*), 582
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_18_LPI2C2_SDA (*C macro*), 582
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_18_SAI1_RX_SYNC (*C macro*), 582
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_18_SEMC_ADDR02 (*C macro*), 582
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_18_SRC_BT_CFG00 (*C macro*), 582
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_18_XBAR1_INOUT16 (*C macro*), 582
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_19_FLEXIO1_FLEXIO23 (*C macro*), 583
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_19_GPIO2_IO19 (*C macro*), 583
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_19_LPI2C2_SCL (*C macro*), 582
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_19_SAI1_RX_BCLK (*C macro*), 582
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_19_SEMC_ADDR03 (*C macro*), 582
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_19_SRC_BT_CFG01 (*C macro*), 583

drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_19_XBAR1_INOUT17 (*C macro*), 582
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_20_FLEXIO1_FLEXIO24 (*C macro*), 583
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_20_FLEXPWM1_PWMA03 (*C macro*), 583
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_20_GPIO2_IO20 (*C macro*), 583
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_20_LPUART2_CTS_B (*C macro*), 583
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_20_SAI1_MCLK (*C macro*), 583
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_20_SEMC_ADDR04 (*C macro*), 583
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_20_SRC_BT_CFG02 (*C macro*), 583
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_21_FLEXIO1_FLEXIO25 (*C macro*), 583
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_21_FLEXPWM1_PWMB03 (*C macro*), 583
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_21_GPIO2_IO21 (*C macro*), 583
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_21_LPUART2_RTS_B (*C macro*), 583
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_21_SAI1_RX_DATA00 (*C macro*), 583
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_21_SEMC_ADDR05 (*C macro*), 583
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_21_SRC_BT_CFG03 (*C macro*), 583
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_22_FLEXIO1_FLEXIO26 (*C macro*), 583
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_22_FLEXPWM1_PWMA02 (*C macro*), 583
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_22_GPIO2_IO22 (*C macro*), 583
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_22_LPUART2_TX (*C macro*), 583
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_22_SAI1_TX_DATA03 (*C macro*), 583
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_22_SEMC_ADDR06 (*C macro*), 583
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_22_SRC_BT_CFG04 (*C macro*), 584
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_23_FLEXIO1_FLEXIO27 (*C macro*), 584
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_23_FLEXPWM1_PWMB02 (*C macro*), 584
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_23_GPIO2_IO23 (*C macro*), 584
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_23_LPUART2_RX (*C macro*), 584
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_23_SAI1_TX_DATA02 (*C macro*), 584
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_23_SEMC_ADDR07 (*C macro*), 584
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_23_SRC_BT_CFG05 (*C macro*), 584
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_24_FLEXIO1_FLEXIO28 (*C macro*), 584
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_24_FLEXPWM1_PWMA01 (*C macro*), 584
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_24_GPIO2_IO24 (*C macro*), 584
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_24_LPUART8_CTS_B (*C macro*), 584
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_24_SAI1_TX_DATA01 (*C macro*), 584
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_24_SEMC_ADDR08 (*C macro*), 584
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_24_SRC_BT_CFG06 (*C macro*), 584
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_25_FLEXIO1_FLEXIO29 (*C macro*), 584
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_25_FLEXPWM1_PWMB01 (*C macro*), 584
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_25_GPIO2_IO25 (*C macro*), 584

drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_25_LPUART8_RTS_B (*C macro*), 584
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_25_SAI1_TX_DATA00 (*C macro*), 584
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_25_SEMC_ADDR09 (*C macro*), 584
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_25_SRC_BT_CFG07 (*C macro*), 584
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_26_FLEXIO1_FLEXIO30 (*C macro*), 585
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_26_FLEXPWM1_PWMA00 (*C macro*), 585
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_26_GPIO2_IO26 (*C macro*), 585
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_26_LPUART8_TX (*C macro*), 585
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_26_SAI1_TX_BCLK (*C macro*), 585
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_26_SEMC_ADDR11 (*C macro*), 584
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_26_SRC_BT_CFG08 (*C macro*), 585
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_27_FLEXIO1_FLEXIO31 (*C macro*), 585
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_27_FLEXPWM1_PWMB00 (*C macro*), 585
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_27_GPIO2_IO27 (*C macro*), 585
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_27_LPUART8_RX (*C macro*), 585
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_27_SAI1_TX_SYNC (*C macro*), 585
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_27_SEMC_ADDR12 (*C macro*), 585
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_27_SRC_BT_CFG09 (*C macro*), 585
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_28_EWM_OUT_B (*C macro*), 585
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_28_FLEXPWM1_PWMX00 (*C macro*), 585
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_28_FLEXPWM2_PWMA03 (*C macro*), 585
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_28_GPIO2_IO28 (*C macro*), 585
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_28_GPT2_CAPTURE2 (*C macro*), 585
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_28_SAI3_MCLK (*C macro*), 585
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_28_SEMC_DQS (*C macro*), 585
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_28_XBAR1_INOUT18 (*C macro*), 585
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_29_FLEXPWM1_PWMX01 (*C macro*), 586
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_29_FLEXPWM2_PWMB03 (*C macro*), 585
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_29_GPIO2_IO29 (*C macro*), 586
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_29_GPT2_COMPARE2 (*C macro*), 586
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_29_SAI3_RX_BCLK (*C macro*), 586
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_29_SEMC_CKE (*C macro*), 585
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_29_WDOG2_RST_B_DEB (*C macro*), 586
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_29_XBAR1_INOUT19 (*C macro*), 586
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_30_FLEXPWM1_PWMX02 (*C macro*), 586
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_30_FLEXPWM2_PWMA02 (*C macro*), 586
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_30_GPIO2_IO30 (*C macro*), 586
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_30_GPT2_COMPARE3 (*C macro*), 586
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_30_LPUART4_CTS_B (*C macro*), 586
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_30_SAI3_RX_SYNC (*C macro*), 586
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_30_SEMC_CLK (*C macro*), 586

drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_30_WDOG1_RST_B_DEB (*C macro*), 586
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_31_FLEXPWM1_PWMX03 (*C macro*), 586
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_31_FLEXPWM2_PWMB02 (*C macro*), 586
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_31_GPIO2_IO31 (*C macro*), 586
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_31_GPT2_CLK (*C macro*), 586
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_31_LPUART4_RTS_B (*C macro*), 586
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_31_SAI3_RX_DATA (*C macro*), 586
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_31_SEMC_DM01 (*C macro*), 586
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_31_WDOG2_B (*C macro*), 586
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_32_GPIO3_IO00 (*C macro*), 587
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_32_LPSPI4_SCK (*C macro*), 587
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_32_LPUART4_TX (*C macro*), 587
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_32_QTIMER1_TIMER0 (*C macro*), 587
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_32_REF_24M_OUT (*C macro*), 587
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_32_SAI3_TX_DATA (*C macro*), 587
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_32_SEMC_DATA08 (*C macro*), 586
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_33_GPIO3_IO01 (*C macro*), 587
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_33_LPSPI4_PCS0 (*C macro*), 587
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_33_LPUART4_RX (*C macro*), 587
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_33_QTIMER1_TIMER1 (*C macro*), 587
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_33_SAI3_TX_BCLK (*C macro*), 587
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_33_SEMC_DATA09 (*C macro*), 587
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_34_ENET_CRS (*C macro*), 587
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_34_GPIO3_IO02 (*C macro*), 587
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_34_LPSPI4_SDO (*C macro*), 587
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_34_LPUART7_TX (*C macro*), 587
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_34_QTIMER1_TIMER2 (*C macro*), 587
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_34_SAI3_TX_SYNC (*C macro*), 587
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_34_SEMC_DATA10 (*C macro*), 587
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_35_ENET_COL (*C macro*), 588
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_35_GPIO3_IO03 (*C macro*), 588
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_35_LPSPI4_SDI (*C macro*), 588
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_35_LPUART7_RX (*C macro*), 587
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_35_QTIMER1_TIMER3 (*C macro*), 587
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_35_SEMC_DATA11 (*C macro*), 587
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_35_USDHC2_WP (*C macro*), 587
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_36_CCM_PMIC_RDY (*C macro*), 588
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_36_ENET_RX_CLK (*C macro*), 588
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_36_FLEXPWM2_PWMA01 (*C macro*), 588
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_36_GPIO3_IO04 (*C macro*), 588
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_36_LPSPI4_PCS1 (*C macro*), 588
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_36_LPUART5_CTS_B (*C macro*), 588
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_36_SEMC_DATA12 (*C macro*), 588
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_36_USDHC1_WP (*C macro*), 588
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_37_ENET_RDATA03 (*C macro*), 588
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_37_FLEXPWM2_PWMB01 (*C macro*), 588
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_37_GPIO3_IO05 (*C macro*), 588

drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_37_LPSPI4_PCS2 (*C macro*), 588
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_37_LPUART5_RTS_B (*C macro*), 588
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_37_MQS_RIGHT (*C macro*), 588
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_37_SEMC_DATA13 (*C macro*), 588
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_37_USDHC1_VSELECT (*C macro*), 588
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_38_ENET_RDATA02 (*C macro*), 589
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_38_FLEXPWM2_PWMA00 (*C macro*), 588
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_38_GPIO3_IO06 (*C macro*), 589
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_38_LPSPI4_PCS3 (*C macro*), 589
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_38_LPUART5_TX (*C macro*), 588
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_38_MQS_LEFT (*C macro*), 588
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_38_SEMC_DATA14 (*C macro*), 588
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_38_USDHC1_CD_B (*C macro*), 589
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_39_ENET_TX_ER (*C macro*), 589
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_39_FLEXPWM2_PWMB00 (*C macro*), 589
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_39_GPIO3_IO07 (*C macro*), 589
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_39_GPT1_CLK (*C macro*), 589
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_39_LPUART5_RX (*C macro*), 589
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_39_SEMC_DATA15 (*C macro*), 589
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_39_USB_OTG1_OC (*C macro*), 589
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_39_WDOG1_B (*C macro*), 589
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_40_ENET_MDIO (*C macro*), 589
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_40_ENET_TDATA03 (*C macro*), 589
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_40_GPIO3_IO08 (*C macro*), 589
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_40_GPT1_COMPARE3 (*C macro*), 589
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_40_SEMC_CSX00 (*C macro*), 589
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_40_SPDIF_OUT (*C macro*), 589
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_40_USB_OTG1_ID (*C macro*), 589
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_40_XBAR1_INOUT18 (*C macro*), 589
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_41_ENET_MDC (*C macro*), 590
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_41_ENET_TDATA02 (*C macro*), 590
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_41_GPIO3_IO09 (*C macro*), 590
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_41_GPT1_COMPARE2 (*C macro*), 590
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_41_SEMC_READY (*C macro*), 589
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_41_SPDIF_IN (*C macro*), 589
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_41_USB_OTG1_PWR (*C macro*), 590
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_EMC_41_XBAR1_INOUT19 (*C macro*), 589
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B0_00_FLEXSPI_A_SS1_B (*C macro*), 598
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B0_00_GPIO3_IO13 (*C macro*), 598
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B0_00_LPI2C3_SCL (*C macro*), 598
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B0_00_QTIMER1_TIMER0 (*C macro*), 597
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B0_00_SAI1_MCLK (*C macro*), 598
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B0_00_SAI2_MCLK (*C macro*), 598
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B0_00_USDHC1_DATA2 (*C macro*), 597
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B0_00_XBAR1_INOUT14 (*C macro*), 598
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B0_01_FLEXSPI_B_SS1_B (*C macro*), 598
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B0_01_GPIO3_IO14 (*C macro*), 598

drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B0_01_LPI2C3_SDA (*C macro*), 598
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B0_01_QTIMER1_TIMER1 (*C macro*), 598
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B0_01_REF_24M_OUT (*C macro*), 598
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B0_01_SAI2_RX_SYNC (*C macro*), 598
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B0_01_USDHC1_DATA3 (*C macro*), 598
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B0_01_XBAR1_INOUT15 (*C macro*), 598
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B0_02_ENET_MDIO (*C macro*), 598
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B0_02_GPIO3_IO15 (*C macro*), 598
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B0_02_LPSPI1_SCK (*C macro*), 598
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B0_02_LPUART7_CTS_B (*C macro*), 598
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B0_02_QTIMER1_TIMER2 (*C macro*), 598
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B0_02_SAI2_RX_BCLK (*C macro*), 598
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B0_02_USDHC1_CMD (*C macro*), 598
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B0_02_XBAR1_INOUT16 (*C macro*), 598
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B0_03_ENET_MDC (*C macro*), 599
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B0_03_GPIO3_IO16 (*C macro*), 599
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B0_03_LPSPI1_PCS0 (*C macro*), 599
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B0_03_LPUART7_RTS_B (*C macro*), 599
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B0_03_QTIMER1_TIMER3 (*C macro*), 599
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B0_03_SAI2_RX_DATA (*C macro*), 599
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B0_03_USDHC1_CLK (*C macro*), 598
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B0_04_FLEXCAN2_TX (*C macro*), 599
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B0_04_FLEXSPI_B_SS0_B (*C macro*), 599
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B0_04_GPIO3_IO17 (*C macro*), 599
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B0_04_LPSPI1_SDO (*C macro*), 599
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B0_04_LPUART7_TX (*C macro*), 599
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B0_04_SAI2_TX_DATA (*C macro*), 599
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B0_04_USDHC1_DATA0 (*C macro*), 599
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B0_05_FLEXCAN2_RX (*C macro*), 599
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B0_05_FLEXSPI_B_DQS (*C macro*), 599
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B0_05_GPIO3_IO18 (*C macro*), 599
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B0_05_LPSPI1_SDI (*C macro*), 599
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B0_05_LPUART7_RX (*C macro*), 599
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B0_05_SAI2_TX_BCLK (*C macro*), 599
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B0_05_USDHC1_DATA1 (*C macro*), 599
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B0_06_GPIO3_IO19 (*C macro*), 600
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B0_06_REF_32K_OUT (*C macro*), 599
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B0_06_SAI2_TX_SYNC (*C macro*), 600
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B0_06_USDHC1_CD_B (*C macro*), 599
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B0_06_USDHC1_RESET_B (*C macro*),

599
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B0_06_WDOG1_B (*C macro*), 600
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B0_06_XBAR1_INOUT17 (*C macro*), 600
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B1_00_FLEXCAN1_TX (*C macro*), 600
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B1_00_FLEXSPI_B_DATA03 (*C macro*), 600
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B1_00_GPIO3_IO20 (*C macro*), 600
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B1_00_LPUART6_TX (*C macro*), 600
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B1_00_USDHC2_DATA2 (*C macro*), 600
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B1_00_XBAR1_INOUT10 (*C macro*), 600
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B1_01_FLEXCAN1_RX (*C macro*), 600
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B1_01_FLEXSPI_A_SS1_B (*C macro*), 600
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B1_01_FLEXSPI_B_SCLK (*C macro*), 600
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B1_01_GPIO3_IO21 (*C macro*), 600
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B1_01_LPUART6_RX (*C macro*), 600
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B1_01_USDHC2_DATA3 (*C macro*), 600
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B1_02_CCM_CLKO1 (*C macro*), 600
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B1_02_ENET_1588_EVENT1_OUT (*C macro*), 600
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B1_02_FLEXSPI_B_DATA00 (*C macro*), 600
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B1_02_GPIO3_IO22 (*C macro*), 600
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B1_02_LPI2C4_SCL (*C macro*), 600
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B1_02_LPUART8_TX (*C macro*), 600
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B1_02_USDHC2_CMD (*C macro*), 600
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B1_03_CCM_CLKO2 (*C macro*), 601
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B1_03_ENET_1588_EVENT1_IN (*C macro*), 601
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B1_03_FLEXSPI_B_DATA02 (*C macro*), 601
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B1_03_GPIO3_IO23 (*C macro*), 601
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B1_03_LPI2C4_SDA (*C macro*), 601
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B1_03_LPUART8_RX (*C macro*), 601
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B1_03_USDHC2_CLK (*C macro*), 601
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B1_04_CCM_WAIT (*C macro*), 601
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B1_04_ENET_REF_CLK1 (*C macro*), 601
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B1_04_ENET_TX_CLK (*C macro*), 601
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B1_04_EWM_OUT_B (*C macro*), 601
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B1_04_FLEXSPI_B_DATA01 (*C macro*), 601
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B1_04_GPIO3_IO24 (*C macro*), 601
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B1_04_USDHC2_DATA0 (*C macro*), 601
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B1_05_CCM_PMIC_RDY (*C macro*), 601
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B1_05_ENET_RDATA01 (*C macro*), 601
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B1_05_FLEXSPI_A_DQS (*C macro*), 601
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B1_05_FLEXSPI_B_SS0_B (*C macro*),

601
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B1_05_GPIO3_IO25 (C macro), 601
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B1_05_SAI3_MCLK (C macro), 601
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B1_05_USDHC2_DATA1 (C macro), 601
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B1_06_CCM_STOP (C macro), 602
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B1_06_ENET_RDATA00 (C macro), 602
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B1_06_FLEXSPI_A_DATA03 (C macro), 601
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B1_06_GPIO3_IO26 (C macro), 602
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B1_06_LPSPI2_PCS0 (C macro), 602
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B1_06_SAI3_TX_BCLK (C macro), 602
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B1_06_USDHC2_CD_B (C macro), 601
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B1_07_ENET_RX_EN (C macro), 602
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B1_07_FLEXSPI_A_SCLK (C macro), 602
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B1_07_GPIO3_IO27 (C macro), 602
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B1_07_LPSPI2_SCK (C macro), 602
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B1_07_SAI3_TX_SYNC (C macro), 602
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B1_07_USDHC2_RESET_B (C macro), 602
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B1_08_ENET_RX_ER (C macro), 602
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B1_08_FLEXSPI_A_DATA00 (C macro), 602
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B1_08_GPIO3_IO28 (C macro), 602
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B1_08_LPSPI2_SDO (C macro), 602
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B1_08_SAI3_TX_DATA (C macro), 602
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B1_08_USDHC2_DATA4 (C macro), 602
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B1_09_ENET_TX_EN (C macro), 602
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B1_09_FLEXSPI_A_DATA02 (C macro), 602
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B1_09_GPIO3_IO29 (C macro), 602
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B1_09_LPSPI2_SDI (C macro), 602
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B1_09_SAI3_RX_BCLK (C macro), 602
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B1_09_USDHC2_DATA5 (C macro), 602
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B1_10_ENET_TDATA00 (C macro), 603
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B1_10_FLEXSPI_A_DATA01 (C macro), 603
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B1_10_GPIO3_IO30 (C macro), 603
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B1_10_LPSPI2_PCS2 (C macro), 603
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B1_10_SAI3_RX_SYNC (C macro), 603
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B1_10_USDHC2_DATA6 (C macro), 603
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B1_11_ENET_TDATA01 (C macro), 603
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B1_11_FLEXSPI_A_SS0_B (C macro), 603
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B1_11_GPIO3_IO31 (C macro), 603
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B1_11_LPSPI2_PCS3 (C macro), 603

drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B1_11_SAI3_RX_DATA (*C macro*), 603
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPIO_SD_B1_11_USDHC2_DATA7 (*C macro*), 603
drivers_RT_RT1020_MIMXRT1024.iomuxc_gpr_mode_t (*C type*), 577
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPR_SAIMCLK_HIGHBITMASK (*C macro*), 603
drivers_RT_RT1020_MIMXRT1024.IOMUXC_GPR_SAIMCLK_LOWBITMASK (*C macro*), 603
drivers_RT_RT1020_MIMXRT1024.iomuxc_gpr_saimclk_t (*C type*), 577
drivers_RT_RT1020_MIMXRT1024.iomuxc_mqs_pwm_oversample_rate_t (*C type*), 577
drivers_RT_RT1020_MIMXRT1024.IOMUXC_MQSConfig (*C function*), 605
drivers_RT_RT1020_MIMXRT1024.IOMUXC_MQSEnable (*C function*), 605
drivers_RT_RT1020_MIMXRT1024.IOMUXC_MQSEnterSoftwareReset (*C function*), 605
drivers_RT_RT1020_MIMXRT1024.IOMUXC_SetPinConfig (*C function*), 604
drivers_RT_RT1020_MIMXRT1024.IOMUXC_SetPinMux (*C function*), 604
drivers_RT_RT1020_MIMXRT1024.IOMUXC_SetSaiMClkClockSource (*C function*), 605
drivers_RT_RT1020_MIMXRT1024.IOMUXC_SNVS_ONOFF (*C macro*), 603
drivers_RT_RT1020_MIMXRT1024.IOMUXC_SNVS_PMIC_ON_REQ_GPIO5_IO01 (*C macro*), 603
drivers_RT_RT1020_MIMXRT1024.IOMUXC_SNVS_PMIC_ON_REQ_SNVS_LP_PMIC_ON_REQ (*C macro*), 603
drivers_RT_RT1020_MIMXRT1024.IOMUXC_SNVS_PMIC_STBY_REQ_CCM_PMIC_VSTBY_REQ (*C macro*), 603
drivers_RT_RT1020_MIMXRT1024.IOMUXC_SNVS_PMIC_STBY_REQ_GPIO5_IO02 (*C macro*), 603
drivers_RT_RT1020_MIMXRT1024.IOMUXC_SNVS_POR_B (*C macro*), 603
drivers_RT_RT1020_MIMXRT1024.IOMUXC_SNVS_TEST_MODE (*C macro*), 603
drivers_RT_RT1020_MIMXRT1024.IOMUXC_SNVS_WAKEUP_GPIO5_IO00 (*C macro*), 603
drivers_RT_RT1020_MIMXRT1024.IOMUXC_SNVS_WAKEUP_NMI_GLUE_NMI (*C macro*), 603
drivers_RT_RT1020_MIMXRT1024.IRQ_ClearPendingIRQ (*C function*), 623
drivers_RT_RT1020_MIMXRT1024.IRQ_SetPriority (*C function*), 623
drivers_RT_RT1020_MIMXRT1024.kCLOCK_CoreSysClk (*C macro*), 237
drivers_RT_RT1020_MIMXRT1024.kLpi2cMasterIrqs (*C var*), 625
drivers_RT_RT1020_MIMXRT1024.kLpi2cSlaveIrqs (*C var*), 625
drivers_RT_RT1020_MIMXRT1024.KPP_ClearStatusFlag (*C function*), 606
drivers_RT_RT1020_MIMXRT1024.KPP_CLOCKS (*C macro*), 236
drivers_RT_RT1020_MIMXRT1024.kpp_config_t (*C type*), 608
drivers_RT_RT1020_MIMXRT1024.KPP_Deinit (*C function*), 606
drivers_RT_RT1020_MIMXRT1024.KPP_DisableInterrupts (*C function*), 606
drivers_RT_RT1020_MIMXRT1024.KPP_EnableInterrupts (*C function*), 606
drivers_RT_RT1020_MIMXRT1024.KPP_GetStatusFlag (*C function*), 606
drivers_RT_RT1020_MIMXRT1024.KPP_Init (*C function*), 606
drivers_RT_RT1020_MIMXRT1024.kpp_interrupt_enable_t (*C type*), 607
drivers_RT_RT1020_MIMXRT1024.KPP_KEYPAD_COLUMNNUM_MAX (*C macro*), 608
drivers_RT_RT1020_MIMXRT1024.KPP_KEYPAD_ROWNUM_MAX (*C macro*), 608
drivers_RT_RT1020_MIMXRT1024.KPP_keyPressScanning (*C function*), 607
drivers_RT_RT1020_MIMXRT1024.KPP_SetSynchronizeChain (*C function*), 606
drivers_RT_RT1020_MIMXRT1024.kpp_sync_operation_t (*C type*), 607
drivers_RT_RT1020_MIMXRT1024.kXBARA_RequestInterruptEnalbe (*C macro*), 936
drivers_RT_RT1020_MIMXRT1024.L1CACHE_CleanDCache (*C function*), 199
drivers_RT_RT1020_MIMXRT1024.L1CACHE_CleanDCacheByRange (*C function*), 200
drivers_RT_RT1020_MIMXRT1024.L1CACHE_CleanInvalidateDCache (*C function*), 199
drivers_RT_RT1020_MIMXRT1024.L1CACHE_CleanInvalidateDCacheByRange (*C function*), 200
drivers_RT_RT1020_MIMXRT1024.L1CACHE_DisableDCache (*C function*), 199
drivers_RT_RT1020_MIMXRT1024.L1CACHE_DisableICache (*C function*), 199
drivers_RT_RT1020_MIMXRT1024.L1CACHE_EnableDCache (*C function*), 199
drivers_RT_RT1020_MIMXRT1024.L1CACHE_EnableICache (*C function*), 199
drivers_RT_RT1020_MIMXRT1024.L1CACHE_InvalidateDCache (*C function*), 199

drivers_RT_RT1020_MIMXRT1024.L1CACHE_InvalidateDCacheByRange (*C function*), 199
drivers_RT_RT1020_MIMXRT1024.L1CACHE_InvalidateICache (*C function*), 199
drivers_RT_RT1020_MIMXRT1024.L1CACHE_InvalidateICacheByRange (*C function*), 199
drivers_RT_RT1020_MIMXRT1024.LPI2C_CheckForBusyBus (*C function*), 627
drivers_RT_RT1020_MIMXRT1024.LPI2C_CLOCKS (*C macro*), 236
drivers_RT_RT1020_MIMXRT1024.lpi2c_data_match_config_mode_t (*C type*), 638
drivers_RT_RT1020_MIMXRT1024.lpi2c_data_match_config_t (*C type*), 638
drivers_RT_RT1020_MIMXRT1024.lpi2c_direction_t (*C type*), 638
drivers_RT_RT1020_MIMXRT1024.LPI2C_DriverIRQHandler (*C function*), 625
drivers_RT_RT1020_MIMXRT1024.LPI2C_GetInstance (*C function*), 626
drivers_RT_RT1020_MIMXRT1024.lpi2c_host_request_polarity_t (*C type*), 638
drivers_RT_RT1020_MIMXRT1024.lpi2c_host_request_source_t (*C type*), 638
drivers_RT_RT1020_MIMXRT1024.lpi2c_master_config_t (*C type*), 638
drivers_RT_RT1020_MIMXRT1024.lpi2c_master_edma_handle_t (*C type*), 643
drivers_RT_RT1020_MIMXRT1024.lpi2c_master_edma_transfer_callback_t (*C type*), 643
drivers_RT_RT1020_MIMXRT1024.lpi2c_master_handle_t (*C type*), 638
drivers_RT_RT1020_MIMXRT1024.lpi2c_master_isr_t (*C type*), 639
drivers_RT_RT1020_MIMXRT1024.lpi2c_master_pin_config_t (*C type*), 638
drivers_RT_RT1020_MIMXRT1024.lpi2c_master_transfer_callback_t (*C type*), 638
drivers_RT_RT1020_MIMXRT1024.lpi2c_master_transfer_t (*C type*), 638
drivers_RT_RT1020_MIMXRT1024.LPI2C_MasterCheckAndClearError (*C function*), 627
drivers_RT_RT1020_MIMXRT1024.LPI2C_MasterClearStatusFlags (*C function*), 628
drivers_RT_RT1020_MIMXRT1024.LPI2C_MasterConfigureDataMatch (*C function*), 627
drivers_RT_RT1020_MIMXRT1024.LPI2C_MasterCreateEDMAHandle (*C function*), 642
drivers_RT_RT1020_MIMXRT1024.LPI2C_MasterDeinit (*C function*), 627
drivers_RT_RT1020_MIMXRT1024.LPI2C_MasterDisableInterrupts (*C function*), 629
drivers_RT_RT1020_MIMXRT1024.LPI2C_MasterEnable (*C function*), 628
drivers_RT_RT1020_MIMXRT1024.LPI2C_MasterEnableDMA (*C function*), 629
drivers_RT_RT1020_MIMXRT1024.LPI2C_MasterEnableInterrupts (*C function*), 629
drivers_RT_RT1020_MIMXRT1024.LPI2C_MasterGetBusIdleState (*C function*), 630
drivers_RT_RT1020_MIMXRT1024.LPI2C_MasterGetDefaultConfig (*C function*), 626
drivers_RT_RT1020_MIMXRT1024.LPI2C_MasterGetEnabledInterrupts (*C function*), 629
drivers_RT_RT1020_MIMXRT1024.LPI2C_MasterGetFifoCounts (*C function*), 630
drivers_RT_RT1020_MIMXRT1024.LPI2C_MasterGetRxFifoAddress (*C function*), 629
drivers_RT_RT1020_MIMXRT1024.LPI2C_MasterGetStatusFlags (*C function*), 628
drivers_RT_RT1020_MIMXRT1024.LPI2C_MasterGetTxFifoAddress (*C function*), 629
drivers_RT_RT1020_MIMXRT1024.LPI2C_MasterInit (*C function*), 626
drivers_RT_RT1020_MIMXRT1024.LPI2C_MasterReceive (*C function*), 632
drivers_RT_RT1020_MIMXRT1024.LPI2C_MasterRepeatedStart (*C function*), 631
drivers_RT_RT1020_MIMXRT1024.LPI2C_MasterReset (*C function*), 627
drivers_RT_RT1020_MIMXRT1024.LPI2C_MasterSend (*C function*), 631
drivers_RT_RT1020_MIMXRT1024.LPI2C_MasterSetBaudRate (*C function*), 630
drivers_RT_RT1020_MIMXRT1024.LPI2C_MasterSetWatermarks (*C function*), 630
drivers_RT_RT1020_MIMXRT1024.LPI2C_MasterStart (*C function*), 631
drivers_RT_RT1020_MIMXRT1024.LPI2C_MasterStop (*C function*), 632
drivers_RT_RT1020_MIMXRT1024.LPI2C_MasterTransferAbort (*C function*), 634
drivers_RT_RT1020_MIMXRT1024.LPI2C_MasterTransferAbortEDMA (*C function*), 643
drivers_RT_RT1020_MIMXRT1024.LPI2C_MasterTransferBlocking (*C function*), 632
drivers_RT_RT1020_MIMXRT1024.LPI2C_MasterTransferCreateHandle (*C function*), 633
drivers_RT_RT1020_MIMXRT1024.LPI2C_MasterTransferEDMA (*C function*), 642
drivers_RT_RT1020_MIMXRT1024.LPI2C_MasterTransferGetCount (*C function*), 634
drivers_RT_RT1020_MIMXRT1024.LPI2C_MasterTransferGetCountEDMA (*C function*), 642
drivers_RT_RT1020_MIMXRT1024.LPI2C_MasterTransferHandleIRQ (*C function*), 634
drivers_RT_RT1020_MIMXRT1024.LPI2C_MasterTransferNonBlocking (*C function*), 633
drivers_RT_RT1020_MIMXRT1024.lpi2c_slave_address_match_t (*C type*), 652
drivers_RT_RT1020_MIMXRT1024.lpi2c_slave_config_t (*C type*), 652
drivers_RT_RT1020_MIMXRT1024.lpi2c_slave_handle_t (*C type*), 652
drivers_RT_RT1020_MIMXRT1024.lpi2c_slave_transfer_callback_t (*C type*), 652

drivers_RT_RT1020_MIMXRT1024.lpi2c_slave_transfer_event_t (*C type*), 652
drivers_RT_RT1020_MIMXRT1024.lpi2c_slave_transfer_t (*C type*), 652
drivers_RT_RT1020_MIMXRT1024.LPI2C_SlaveClearStatusFlags (*C function*), 646
drivers_RT_RT1020_MIMXRT1024.LPI2C_SlaveDeinit (*C function*), 645
drivers_RT_RT1020_MIMXRT1024.LPI2C_SlaveDisableInterrupts (*C function*), 646
drivers_RT_RT1020_MIMXRT1024.LPI2C_SlaveEnable (*C function*), 645
drivers_RT_RT1020_MIMXRT1024.LPI2C_SlaveEnableAckStall (*C function*), 647
drivers_RT_RT1020_MIMXRT1024.LPI2C_SlaveEnableDMA (*C function*), 647
drivers_RT_RT1020_MIMXRT1024.LPI2C_SlaveEnableInterrupts (*C function*), 646
drivers_RT_RT1020_MIMXRT1024.LPI2C_SlaveGetBusIdleState (*C function*), 647
drivers_RT_RT1020_MIMXRT1024.LPI2C_SlaveGetDefaultConfig (*C function*), 644
drivers_RT_RT1020_MIMXRT1024.LPI2C_SlaveGetEnabledInterrupts (*C function*), 646
drivers_RT_RT1020_MIMXRT1024.LPI2C_SlaveGetReceivedAddress (*C function*), 647
drivers_RT_RT1020_MIMXRT1024.LPI2C_SlaveGetStatusFlags (*C function*), 645
drivers_RT_RT1020_MIMXRT1024.LPI2C_SlaveInit (*C function*), 645
drivers_RT_RT1020_MIMXRT1024.LPI2C_SlaveReceive (*C function*), 648
drivers_RT_RT1020_MIMXRT1024.LPI2C_SlaveReset (*C function*), 645
drivers_RT_RT1020_MIMXRT1024.LPI2C_SlaveSend (*C function*), 648
drivers_RT_RT1020_MIMXRT1024.LPI2C_SlaveTransferAbort (*C function*), 649
drivers_RT_RT1020_MIMXRT1024.LPI2C_SlaveTransferCreateHandle (*C function*), 648
drivers_RT_RT1020_MIMXRT1024.LPI2C_SlaveTransferGetCount (*C function*), 649
drivers_RT_RT1020_MIMXRT1024.LPI2C_SlaveTransferHandleIRQ (*C function*), 649
drivers_RT_RT1020_MIMXRT1024.LPI2C_SlaveTransferNonBlocking (*C function*), 648
drivers_RT_RT1020_MIMXRT1024.LPI2C_SlaveTransmitAck (*C function*), 647
drivers_RT_RT1020_MIMXRT1024.LPSPI_CheckTransferArgument (*C function*), 659
drivers_RT_RT1020_MIMXRT1024.LPSPI_ClearStatusFlags (*C function*), 657
drivers_RT_RT1020_MIMXRT1024.lpspi_clock_phase_t (*C type*), 673
drivers_RT_RT1020_MIMXRT1024.lpspi_clock_polarity_t (*C type*), 673
drivers_RT_RT1020_MIMXRT1024.LPSPI_CLOCKS (*C macro*), 236
drivers_RT_RT1020_MIMXRT1024.lpspi_data_out_config_t (*C type*), 674
drivers_RT_RT1020_MIMXRT1024.LPSPI_Deinit (*C function*), 656
drivers_RT_RT1020_MIMXRT1024.lpspi_delay_type_t (*C type*), 674
drivers_RT_RT1020_MIMXRT1024.LPSPI_DisableDMA (*C function*), 658
drivers_RT_RT1020_MIMXRT1024.LPSPI_DisableInterrupts (*C function*), 658
drivers_RT_RT1020_MIMXRT1024.LPSPI_DriverIRQHandler (*C function*), 666
drivers_RT_RT1020_MIMXRT1024.LPSPI_DUMMY_DATA (*C macro*), 675
drivers_RT_RT1020_MIMXRT1024.LPSPI_Enable (*C function*), 657
drivers_RT_RT1020_MIMXRT1024.LPSPI_EnableDMA (*C function*), 658
drivers_RT_RT1020_MIMXRT1024.LPSPI_EnableInterrupts (*C function*), 658
drivers_RT_RT1020_MIMXRT1024.LPSPI_FlushFifo (*C function*), 660
drivers_RT_RT1020_MIMXRT1024.LPSPI_GetInstance (*C function*), 656
drivers_RT_RT1020_MIMXRT1024.LPSPI_GetRxFifoCount (*C function*), 657
drivers_RT_RT1020_MIMXRT1024.LPSPI_GetRxFifoSize (*C function*), 657
drivers_RT_RT1020_MIMXRT1024.LPSPI_GetRxRegisterAddress (*C function*), 659
drivers_RT_RT1020_MIMXRT1024.LPSPI_GetStatusFlags (*C function*), 657
drivers_RT_RT1020_MIMXRT1024.LPSPI_GetTcr (*C function*), 658
drivers_RT_RT1020_MIMXRT1024.LPSPI_GetTxFifoCount (*C function*), 657
drivers_RT_RT1020_MIMXRT1024.LPSPI_GetTxFifoSize (*C function*), 657
drivers_RT_RT1020_MIMXRT1024.LPSPI_GetTxRegisterAddress (*C function*), 659
drivers_RT_RT1020_MIMXRT1024.lpspi_host_request_select_t (*C type*), 673
drivers_RT_RT1020_MIMXRT1024.LPSPI_IsMaster (*C function*), 660
drivers_RT_RT1020_MIMXRT1024.lpspi_master_config_t (*C type*), 674
drivers_RT_RT1020_MIMXRT1024.lpspi_master_edma_handle_t (*C type*), 680
drivers_RT_RT1020_MIMXRT1024.lpspi_master_edma_transfer_callback_t (*C type*), 680
drivers_RT_RT1020_MIMXRT1024.lpspi_master_handle_t (*C type*), 674
drivers_RT_RT1020_MIMXRT1024.LPSPI_MASTER_PCS_MASK (*C macro*), 675
drivers_RT_RT1020_MIMXRT1024.LPSPI_MASTER_PCS_SHIFT (*C macro*), 675
drivers_RT_RT1020_MIMXRT1024.lpspi_master_slave_mode_t (*C type*), 673

drivers_RT_RT1020_MIMXRT1024.lpspi_master_transfer_callback_t (*C type*), 674
drivers_RT_RT1020_MIMXRT1024.LPSPI_MasterGetDefaultConfig (*C function*), 656
drivers_RT_RT1020_MIMXRT1024.LPSPI_MasterInit (*C function*), 656
drivers_RT_RT1020_MIMXRT1024.LPSPI_MasterSetBaudRate (*C function*), 661
drivers_RT_RT1020_MIMXRT1024.LPSPI_MasterSetDelayScaler (*C function*), 661
drivers_RT_RT1020_MIMXRT1024.LPSPI_MasterSetDelayTimes (*C function*), 662
drivers_RT_RT1020_MIMXRT1024.LPSPI_MasterTransferAbort (*C function*), 664
drivers_RT_RT1020_MIMXRT1024.LPSPI_MasterTransferAbortEDMA (*C function*), 682
drivers_RT_RT1020_MIMXRT1024.LPSPI_MasterTransferBlocking (*C function*), 663
drivers_RT_RT1020_MIMXRT1024.LPSPI_MasterTransferCreateHandle (*C function*), 663
drivers_RT_RT1020_MIMXRT1024.LPSPI_MasterTransferCreateHandleEDMA (*C function*), 680
drivers_RT_RT1020_MIMXRT1024.LPSPI_MasterTransferEDMA (*C function*), 681
drivers_RT_RT1020_MIMXRT1024.LPSPI_MasterTransferEDMALite (*C function*), 681
drivers_RT_RT1020_MIMXRT1024.LPSPI_MasterTransferGetCount (*C function*), 664
drivers_RT_RT1020_MIMXRT1024.LPSPI_MasterTransferGetCountEDMA (*C function*), 682
drivers_RT_RT1020_MIMXRT1024.LPSPI_MasterTransferHandleIRQ (*C function*), 664
drivers_RT_RT1020_MIMXRT1024.LPSPI_MasterTransferNonBlocking (*C function*), 663
drivers_RT_RT1020_MIMXRT1024.LPSPI_MasterTransferPrepareEDMALite (*C function*), 681
drivers_RT_RT1020_MIMXRT1024.lpspi_match_config_t (*C type*), 673
drivers_RT_RT1020_MIMXRT1024.lpspi_pcs_polarity_config_t (*C type*), 673
drivers_RT_RT1020_MIMXRT1024.lpspi_pin_config_t (*C type*), 674
drivers_RT_RT1020_MIMXRT1024.LPSPI_ReadData (*C function*), 663
drivers_RT_RT1020_MIMXRT1024.LPSPI_Reset (*C function*), 656
drivers_RT_RT1020_MIMXRT1024.LPSPI_SelectTransferPCS (*C function*), 659
drivers_RT_RT1020_MIMXRT1024.LPSPI_SetAllPcsPolarity (*C function*), 660
drivers_RT_RT1020_MIMXRT1024.LPSPI_SetDummyData (*C function*), 663
drivers_RT_RT1020_MIMXRT1024.LPSPI_SetFifoWatermarks (*C function*), 660
drivers_RT_RT1020_MIMXRT1024.LPSPI_SetFrameSize (*C function*), 661
drivers_RT_RT1020_MIMXRT1024.LPSPI_SetMasterSlaveMode (*C function*), 659
drivers_RT_RT1020_MIMXRT1024.LPSPI_SetPCSContinous (*C function*), 660
drivers_RT_RT1020_MIMXRT1024.lpspi_shift_direction_t (*C type*), 673
drivers_RT_RT1020_MIMXRT1024.lpspi_slave_config_t (*C type*), 674
drivers_RT_RT1020_MIMXRT1024.lpspi_slave_edma_handle_t (*C type*), 680
drivers_RT_RT1020_MIMXRT1024.lpspi_slave_edma_transfer_callback_t (*C type*), 680
drivers_RT_RT1020_MIMXRT1024.lpspi_slave_handle_t (*C type*), 674
drivers_RT_RT1020_MIMXRT1024.LPSPI_SLAVE_PCS_MASK (*C macro*), 675
drivers_RT_RT1020_MIMXRT1024.LPSPI_SLAVE_PCS_SHIFT (*C macro*), 675
drivers_RT_RT1020_MIMXRT1024.lpspi_slave_transfer_callback_t (*C type*), 674
drivers_RT_RT1020_MIMXRT1024.LPSPI_SlaveGetDefaultConfig (*C function*), 656
drivers_RT_RT1020_MIMXRT1024.LPSPI_SlaveInit (*C function*), 656
drivers_RT_RT1020_MIMXRT1024.LPSPI_SlaveTransferAbort (*C function*), 665
drivers_RT_RT1020_MIMXRT1024.LPSPI_SlaveTransferAbortEDMA (*C function*), 683
drivers_RT_RT1020_MIMXRT1024.LPSPI_SlaveTransferCreateHandle (*C function*), 664
drivers_RT_RT1020_MIMXRT1024.LPSPI_SlaveTransferCreateHandleEDMA (*C function*), 682
drivers_RT_RT1020_MIMXRT1024.LPSPI_SlaveTransferEDMA (*C function*), 683
drivers_RT_RT1020_MIMXRT1024.LPSPI_SlaveTransferGetCount (*C function*), 665
drivers_RT_RT1020_MIMXRT1024.LPSPI_SlaveTransferGetCountEDMA (*C function*), 683
drivers_RT_RT1020_MIMXRT1024.LPSPI_SlaveTransferHandleIRQ (*C function*), 665
drivers_RT_RT1020_MIMXRT1024.LPSPI_SlaveTransferNonBlocking (*C function*), 665
drivers_RT_RT1020_MIMXRT1024.lpspi_transfer_t (*C type*), 675
drivers_RT_RT1020_MIMXRT1024.lpspi_transfer_width_t (*C type*), 674
drivers_RT_RT1020_MIMXRT1024.LPSPI_WaitTxFifoEmpty (*C function*), 666
drivers_RT_RT1020_MIMXRT1024.lpspi_which_pcs_t (*C type*), 673
drivers_RT_RT1020_MIMXRT1024.LPSPI_WriteData (*C function*), 662
drivers_RT_RT1020_MIMXRT1024.LPUART_ClearStatusFlags (*C function*), 691
drivers_RT_RT1020_MIMXRT1024.LPUART_CLOCKS (*C macro*), 236
drivers_RT_RT1020_MIMXRT1024.lpuart_config_t (*C type*), 705
drivers_RT_RT1020_MIMXRT1024.lpuart_data_bits_t (*C type*), 704

drivers_RT_RT1020_MIMXRT1024.LPUART_Deinit (*C function*), 688
drivers_RT_RT1020_MIMXRT1024.LPUART_DisableInterrupts (*C function*), 691
drivers_RT_RT1020_MIMXRT1024.LPUART_DriverIRQHandler (*C function*), 698
drivers_RT_RT1020_MIMXRT1024.lpuart_edma_handle_t (*C type*), 711
drivers_RT_RT1020_MIMXRT1024.lpuart_edma_transfer_callback_t (*C type*), 711
drivers_RT_RT1020_MIMXRT1024.LPUART_Enable9bitMode (*C function*), 689
drivers_RT_RT1020_MIMXRT1024.LPUART_EnableInterrupts (*C function*), 691
drivers_RT_RT1020_MIMXRT1024.LPUART_EnableMatchAddress (*C function*), 690
drivers_RT_RT1020_MIMXRT1024.LPUART_EnableRx (*C function*), 693
drivers_RT_RT1020_MIMXRT1024.LPUART_EnableRxDMA (*C function*), 692
drivers_RT_RT1020_MIMXRT1024.LPUART_EnableTx (*C function*), 693
drivers_RT_RT1020_MIMXRT1024.LPUART_EnableTxDMA (*C function*), 692
drivers_RT_RT1020_MIMXRT1024.LPUART_GetDataRegisterAddress (*C function*), 692
drivers_RT_RT1020_MIMXRT1024.LPUART_GetDefaultConfig (*C function*), 688
drivers_RT_RT1020_MIMXRT1024.LPUART_GetEnabledInterrupts (*C function*), 691
drivers_RT_RT1020_MIMXRT1024.LPUART_GetInstance (*C function*), 692
drivers_RT_RT1020_MIMXRT1024.LPUART_GetRxFifoCount (*C function*), 693
drivers_RT_RT1020_MIMXRT1024.LPUART_GetStatusFlags (*C function*), 690
drivers_RT_RT1020_MIMXRT1024.LPUART_GetTxFifoCount (*C function*), 693
drivers_RT_RT1020_MIMXRT1024.lpuart_handle_t (*C type*), 705
drivers_RT_RT1020_MIMXRT1024.lpuart_idle_config_t (*C type*), 705
drivers_RT_RT1020_MIMXRT1024.lpuart_idle_type_select_t (*C type*), 705
drivers_RT_RT1020_MIMXRT1024.LPUART_Init (*C function*), 688
drivers_RT_RT1020_MIMXRT1024.lpuart_isr_t (*C type*), 705
drivers_RT_RT1020_MIMXRT1024.lpuart_parity_mode_t (*C type*), 704
drivers_RT_RT1020_MIMXRT1024.LPUART_ReadBlocking (*C function*), 694
drivers_RT_RT1020_MIMXRT1024.LPUART_ReadBlocking16bit (*C function*), 695
drivers_RT_RT1020_MIMXRT1024.LPUART_ReadByte (*C function*), 693
drivers_RT_RT1020_MIMXRT1024.LPUART_ReceiveEDMA (*C function*), 710
drivers_RT_RT1020_MIMXRT1024.LPUART_SendAddress (*C function*), 693
drivers_RT_RT1020_MIMXRT1024.LPUART_SendEDMA (*C function*), 709
drivers_RT_RT1020_MIMXRT1024.LPUART_SetBaudRate (*C function*), 689
drivers_RT_RT1020_MIMXRT1024.LPUART_SetMatchAddress (*C function*), 689
drivers_RT_RT1020_MIMXRT1024.LPUART_SetRxFifoWatermark (*C function*), 690
drivers_RT_RT1020_MIMXRT1024.LPUART_SetTxFifoWatermark (*C function*), 690
drivers_RT_RT1020_MIMXRT1024.LPUART_SoftwareReset (*C function*), 688
drivers_RT_RT1020_MIMXRT1024.lpuart_stop_bit_count_t (*C type*), 704
drivers_RT_RT1020_MIMXRT1024.lpuart_transfer_callback_t (*C type*), 705
drivers_RT_RT1020_MIMXRT1024.lpuart_transfer_t (*C type*), 705
drivers_RT_RT1020_MIMXRT1024.LPUART_TransferAbortReceive (*C function*), 698
drivers_RT_RT1020_MIMXRT1024.LPUART_TransferAbortReceiveEDMA (*C function*), 710
drivers_RT_RT1020_MIMXRT1024.LPUART_TransferAbortSend (*C function*), 697
drivers_RT_RT1020_MIMXRT1024.LPUART_TransferAbortSendEDMA (*C function*), 710
drivers_RT_RT1020_MIMXRT1024.LPUART_TransferCreateHandle (*C function*), 695
drivers_RT_RT1020_MIMXRT1024.LPUART_TransferCreateHandleEDMA (*C function*), 709
drivers_RT_RT1020_MIMXRT1024.LPUART_TransferEdmaHandleIRQ (*C function*), 711
drivers_RT_RT1020_MIMXRT1024.LPUART_TransferEnable16Bit (*C function*), 690
drivers_RT_RT1020_MIMXRT1024.LPUART_TransferGetReceiveCount (*C function*), 698
drivers_RT_RT1020_MIMXRT1024.LPUART_TransferGetReceiveCountEDMA (*C function*), 710
drivers_RT_RT1020_MIMXRT1024.LPUART_TransferGetRxRingBufferLength (*C function*), 697
drivers_RT_RT1020_MIMXRT1024.LPUART_TransferGetSendCount (*C function*), 697
drivers_RT_RT1020_MIMXRT1024.LPUART_TransferGetSendCountEDMA (*C function*), 710
drivers_RT_RT1020_MIMXRT1024.LPUART_TransferHandleErrorIRQ (*C function*), 698
drivers_RT_RT1020_MIMXRT1024.LPUART_TransferHandleIRQ (*C function*), 698
drivers_RT_RT1020_MIMXRT1024.LPUART_TransferReceiveNonBlocking (*C function*), 697
drivers_RT_RT1020_MIMXRT1024.LPUART_TransferSendNonBlocking (*C function*), 695
drivers_RT_RT1020_MIMXRT1024.LPUART_TransferStartRingBuffer (*C function*), 696
drivers_RT_RT1020_MIMXRT1024.LPUART_TransferStopRingBuffer (*C function*), 696

drivers_RT_RT1020_MIMXRT1024.lpuart_transmit_cts_config_t (*C type*), 704
drivers_RT_RT1020_MIMXRT1024.lpuart_transmit_cts_source_t (*C type*), 704
drivers_RT_RT1020_MIMXRT1024.lpuart_transmit_rts_polarity_t (*C type*), 704
drivers_RT_RT1020_MIMXRT1024.LPUART_WriteBlocking (*C function*), 694
drivers_RT_RT1020_MIMXRT1024.LPUART_WriteBlocking16bit (*C function*), 694
drivers_RT_RT1020_MIMXRT1024.LPUART_WriteByte (*C function*), 693
drivers_RT_RT1020_MIMXRT1024.MAKE_STATUS (*C macro*), 624
drivers_RT_RT1020_MIMXRT1024.MAKE_VERSION (*C macro*), 624
drivers_RT_RT1020_MIMXRT1024.MAX (*C macro*), 609
drivers_RT_RT1020_MIMXRT1024.MCUX_CS (*C macro*), 611
drivers_RT_RT1020_MIMXRT1024.MCUX_DEPRECATED (*C macro*), 612
drivers_RT_RT1020_MIMXRT1024.MCUX_DEPRECATED_MACRO (*C macro*), 612
drivers_RT_RT1020_MIMXRT1024.MCUX_EXPERIMENTAL (*C macro*), 612
drivers_RT_RT1020_MIMXRT1024.MCUX_EXPERIMENTAL_MACRO (*C macro*), 612
drivers_RT_RT1020_MIMXRT1024.MCUX_MASK_INVERT_8 (*C macro*), 609
drivers_RT_RT1020_MIMXRT1024.MCUX_MASK_INVERT_16 (*C macro*), 609
drivers_RT_RT1020_MIMXRT1024.MCUX_MASK_INVERT_32 (*C macro*), 609
drivers_RT_RT1020_MIMXRT1024.MCUX_RAMFUNC (*C macro*), 612
drivers_RT_RT1020_MIMXRT1024.MCUX_REG_BIT_CLEAR8 (*C macro*), 610
drivers_RT_RT1020_MIMXRT1024.MCUX_REG_BIT_CLEAR16 (*C macro*), 610
drivers_RT_RT1020_MIMXRT1024.MCUX_REG_BIT_CLEAR32 (*C macro*), 610
drivers_RT_RT1020_MIMXRT1024.MCUX_REG_BIT_GET8 (*C macro*), 610
drivers_RT_RT1020_MIMXRT1024.MCUX_REG_BIT_GET16 (*C macro*), 610
drivers_RT_RT1020_MIMXRT1024.MCUX_REG_BIT_GET32 (*C macro*), 610
drivers_RT_RT1020_MIMXRT1024.MCUX_REG_BIT_SET8 (*C macro*), 610
drivers_RT_RT1020_MIMXRT1024.MCUX_REG_BIT_SET16 (*C macro*), 610
drivers_RT_RT1020_MIMXRT1024.MCUX_REG_BIT_SET32 (*C macro*), 610
drivers_RT_RT1020_MIMXRT1024.MCUX_REG_MODIFY8 (*C macro*), 610
drivers_RT_RT1020_MIMXRT1024.MCUX_REG_MODIFY16 (*C macro*), 610
drivers_RT_RT1020_MIMXRT1024.MCUX_REG_MODIFY32 (*C macro*), 610
drivers_RT_RT1020_MIMXRT1024.MCUX_REG_READ8 (*C macro*), 609
drivers_RT_RT1020_MIMXRT1024.MCUX_REG_READ16 (*C macro*), 610
drivers_RT_RT1020_MIMXRT1024.MCUX_REG_READ32 (*C macro*), 610
drivers_RT_RT1020_MIMXRT1024.MCUX_REG_WRITE8 (*C macro*), 609
drivers_RT_RT1020_MIMXRT1024.MCUX_REG_WRITE16 (*C macro*), 609
drivers_RT_RT1020_MIMXRT1024.MCUX_REG_WRITE32 (*C macro*), 609
drivers_RT_RT1020_MIMXRT1024.MCUX_SDK_SAI_ALLOW_NULL_FIFO_WATERMARK (*C macro*), 799
drivers_RT_RT1020_MIMXRT1024.MCUX_SDK_SAI_DISABLE_IMPLICIT_CHAN_CONFIG (*C macro*), 799
drivers_RT_RT1020_MIMXRT1024.MCUX_SDK_SAI_EDMA_RX_ENABLE_INTERNAL (*C macro*), 810
drivers_RT_RT1020_MIMXRT1024.MCUX_SDK_SAI_EDMA_TX_ENABLE_INTERNAL (*C macro*), 810
drivers_RT_RT1020_MIMXRT1024.MIN (*C macro*), 609
drivers_RT_RT1020_MIMXRT1024.MSDK_HAS_DWT_CYCCNT (*C macro*), 624
drivers_RT_RT1020_MIMXRT1024.MSDK_INVALID_IRQ_HANDLER (*C macro*), 624
drivers_RT_RT1020_MIMXRT1024.MSDK_REG_NONSECURE_ADDR (*C macro*), 624
drivers_RT_RT1020_MIMXRT1024.MSDK_REG_SECURE_ADDR (*C macro*), 624
drivers_RT_RT1020_MIMXRT1024.MSEC_TO_COUNT (*C macro*), 611
drivers_RT_RT1020_MIMXRT1024.OCOTP_CheckBusyStatus (*C function*), 713
drivers_RT_RT1020_MIMXRT1024.OCOTP_CheckErrorStatus (*C function*), 713
drivers_RT_RT1020_MIMXRT1024.OCOTP_ClearErrorStatus (*C function*), 713
drivers_RT_RT1020_MIMXRT1024.OCOTP_Deinit (*C function*), 713
drivers_RT_RT1020_MIMXRT1024.OCOTP_GetVersion (*C function*), 715
drivers_RT_RT1020_MIMXRT1024.OCOTP_Init (*C function*), 713
drivers_RT_RT1020_MIMXRT1024.OCOTP_READ_FUSE_DATA_COUNT (*C macro*), 715
drivers_RT_RT1020_MIMXRT1024.OCOTP_ReadFuse (*C function*), 714

drivers_RT_RT1020_MIMXRT1024.OCOTP_ReadFuseExt (*C function*), 714
drivers_RT_RT1020_MIMXRT1024.OCOTP_ReadFuseShadowRegister (*C macro*), 715
drivers_RT_RT1020_MIMXRT1024.OCOTP_ReadFuseShadowRegisterExt (*C macro*), 715
drivers_RT_RT1020_MIMXRT1024.OCOTP_ReloadShadowRegister (*C function*), 713
drivers_RT_RT1020_MIMXRT1024.ocotp_timing_t (*C type*), 712
drivers_RT_RT1020_MIMXRT1024.OCOTP_WriteFuse (*C function*), 714
drivers_RT_RT1020_MIMXRT1024.OCOTP_WriteFuseShadowRegister (*C macro*), 715
drivers_RT_RT1020_MIMXRT1024.OCOTP_WriteFuseShadowRegisterWithLock (*C macro*), 715
drivers_RT_RT1020_MIMXRT1024.OCOTP_WriteFuseWithLock (*C function*), 714
drivers_RT_RT1020_MIMXRT1024.OCRAM_EXSC_CLOCKS (*C macro*), 236
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kEDMA_ChannelPriorityErrorFlag (*C enumerator*), 289
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kEDMA_DestinationAddressErrorFlag (*C enumerator*), 289
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kEDMA_DestinationBusErrorFlag (*C enumerator*), 288
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kEDMA_DestinationOffsetErrorFlag (*C enumerator*), 289
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kEDMA_DoneFlag (*C enumerator*), 288
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kEDMA_ErrorChannelFlag (*C enumerator*), 289
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kEDMA_ErrorFlag (*C enumerator*), 288
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kEDMA_InterruptFlag (*C enumerator*), 288
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kEDMA_NbytesErrorFlag (*C enumerator*), 289
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kEDMA_ScatterGatherErrorFlag (*C enumerator*), 289
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kEDMA_SourceAddressErrorFlag (*C enumerator*), 289
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kEDMA_SourceBusErrorFlag (*C enumerator*), 288
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kEDMA_SourceOffsetErrorFlag (*C enumerator*), 289
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kEDMA_TransferCanceledFlag (*C enumerator*), 289
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kEDMA_ValidFlag (*C enumerator*), 289
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kFLEXCAN_RxFifoFrameAvlFlag (*C enumerator*), 373
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kFLEXCAN_RxFifoOverflowFlag (*C enumerator*), 373
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kFLEXCAN_RxFifoWarningFlag (*C enumerator*), 373
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kFLEXIO_I2S_RxDataRegFullFlag (*C enumerator*), 441
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kFLEXIO_I2S_RxDataRegFullInterruptEnable (*C enumerator*), 441
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kFLEXIO_I2S_TxDataRegEmptyFlag (*C enumerator*), 441
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kFLEXIO_I2S_TxDataRegEmptyInterruptEnable (*C enumerator*), 440
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kFLEXRAM_D0TCMECCMultiError (*C enumerator*), 532
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kFLEXRAM_D0TCMECCSingleError (*C enumerator*), 532

drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kFLEXRAM_D1TCMECCMultiError (*C enumerator*), 532
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kFLEXRAM_D1TCMECCSingleError (*C enumerator*), 532
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kFLEXRAM_DTCMAccessError (*C enumerator*), 532
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kFLEXRAM_DTCMMagicAddrMatch (*C enumerator*), 532
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kFLEXRAM_InterruptStatusAll (*C enumerator*), 532
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kFLEXRAM_ITCMAccessError (*C enumerator*), 532
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kFLEXRAM_ITCMECCMultiError (*C enumerator*), 532
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kFLEXRAM_ITCMECCSingleError (*C enumerator*), 532
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kFLEXRAM_ITCMMagicAddrMatch (*C enumerator*), 532
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kFLEXRAM_OCRAMAccessError (*C enumerator*), 532
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kFLEXRAM_OCRAMECCMultiError (*C enumerator*), 532
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kFLEXRAM_OCRAMECCSingleError (*C enumerator*), 532
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kFLEXRAM_OCRAMMagicAddrMatch (*C enumerator*), 532
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kFLEXRAM_Read (*C enumerator*), 531
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kFLEXRAM_TCMSize32KB (*C enumerator*), 533
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kFLEXRAM_TCMSize64KB (*C enumerator*), 533
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kFLEXRAM_TCMSize128KB (*C enumerator*), 533
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kFLEXRAM_TCMSize256KB (*C enumerator*), 533
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kFLEXRAM_TCMSize512KB (*C enumerator*), 533
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kFLEXRAM_Write (*C enumerator*), 531
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kFLEXSPI_Command_CADDR_DDR (*C enumerator*), 547
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kFLEXSPI_Command_CADDR_SDR (*C enumerator*), 547
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kFLEXSPI_Command_DATSZ_DDR (*C enumerator*), 548
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kFLEXSPI_Command_DATSZ_SDR (*C enumerator*), 547
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kFLEXSPI_Command_DDR (*C enumerator*), 547
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kFLEXSPI_Command_DUMMY_DDR (*C enumerator*), 548
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kFLEXSPI_Command_DUMMY_RWDS_DDR (*C enumerator*), 548
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kFLEXSPI_Command_DUMMY_RWDS_SDR (*C enumerator*), 547
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kFLEXSPI_Command_DUMMY_SDR (*C enumerator*), 547

drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kFLEXSPI_Command_JUMP_ON_CS (*C enumerator*), 548
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kFLEXSPI_Command_LEARN_DDR (*C enumerator*), 548
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kFLEXSPI_Command_LEARN_SDR (*C enumerator*), 547
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kFLEXSPI_Command_MODE1_DDR (*C enumerator*), 548
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kFLEXSPI_Command_MODE1_SDR (*C enumerator*), 547
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kFLEXSPI_Command_MODE2_DDR (*C enumerator*), 548
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kFLEXSPI_Command_MODE2_SDR (*C enumerator*), 547
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kFLEXSPI_Command_MODE4_DDR (*C enumerator*), 548
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kFLEXSPI_Command_MODE4_SDR (*C enumerator*), 547
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kFLEXSPI_Command_MODE8_DDR (*C enumerator*), 548
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kFLEXSPI_Command_MODE8_SDR (*C enumerator*), 547
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kFLEXSPI_Command_RADDR_DDR (*C enumerator*), 547
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kFLEXSPI_Command_RADDR_SDR (*C enumerator*), 547
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kFLEXSPI_Command_READ_DDR (*C enumerator*), 548
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kFLEXSPI_Command_READ_SDR (*C enumerator*), 547
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kFLEXSPI_Command_SDR (*C enumerator*), 547
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kFLEXSPI_Command_STOP (*C enumerator*), 546
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kFLEXSPI_Command_WRITE_DDR (*C enumerator*), 548
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kFLEXSPI_Command_WRITE_SDR (*C enumerator*), 547
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kPMU_1P1BrownoutOnOutput (*C enumerator*), 726
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kPMU_1P1RegulatorOutputOK (*C enumerator*), 726
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kPMU_2P5BrownoutOnOutput (*C enumerator*), 726
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kPMU_2P5RegulatorOutputOK (*C enumerator*), 726
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kPMU_3P0BrownoutOnOutput (*C enumerator*), 726
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kPMU_3P0RegulatorOutputOK (*C enumerator*), 726
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kSAI_Channel0Mask (*C enumerator*), 790
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kSAI_Channel1Mask (*C enumerator*), 790
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kSAI_Channel2Mask (*C enumerator*), 790
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kSAI_Channel3Mask (*C enumerator*), 790

drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kSAI_Channel4Mask (*C enumerator*), 791
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kSAI_Channel5Mask (*C enumerator*), 791
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kSAI_Channel6Mask (*C enumerator*), 791
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kSAI_Channel7Mask (*C enumerator*), 791
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kSAI_FIFOErrorFlag (*C enumerator*), 794
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kSAI_FIFOErrorInterruptEnable (*C enumerator*), 793
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kSAI_FIFORequestDMAEnable (*C enumerator*), 794
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kSAI_FIFORequestFlag (*C enumerator*), 794
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kSAI_FIFORequestInterruptEnable (*C enumerator*), 794
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kSAI_FIFOWarningDMAEnable (*C enumerator*), 794
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kSAI_FIFOWarningFlag (*C enumerator*), 794
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kSAI_FIFOWarningInterruptEnable (*C enumerator*), 793
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kSAI_SyncErrorFlag (*C enumerator*), 794
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kSAI_SyncErrorInterruptEnable (*C enumerator*), 793
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kSAI_WordStartFlag (*C enumerator*), 794
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kSAI_WordStartInterruptEnable (*C enumerator*), 793
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kSPDIF_AllInterrupt (*C enumerator*), 874
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kSPDIF_LockLoss (*C enumerator*), 874
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kSPDIF_QChannelReceiveRegisterFull (*C enumerator*), 873
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kSPDIF_QChannelReceiveRegisterOverrun (*C enumerator*), 873
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kSPDIF_RxControlChannelChange (*C enumerator*), 873
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kSPDIF_RxDMAEnable (*C enumerator*), 874
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kSPDIF_RxDPLLLocked (*C enumerator*), 873
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kSPDIF_RxFIFOError (*C enumerator*), 874
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kSPDIF_RxFIFOFull (*C enumerator*), 874
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kSPDIF_RxFIFOResync (*C enumerator*), 874
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kSPDIF_RxIllegalSymbol (*C enumerator*), 873
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kSPDIF_RxParityBitError (*C enumerator*), 873
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kSPDIF_TxDMAEnable (*C enumerator*), 874
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kSPDIF_TxFIFOEmpty (*C enumera-*

tor), 874
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kSPDIF_TxFIFOError (*C enumerator*), 873
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kSPDIF_TxFIFOResync (*C enumerator*), 873
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kSPDIF_UChannelReceiveRegisterFull (*C enumerator*), 873
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kSPDIF_UChannelReceiveRegisterOverrun (*C enumerator*), 873
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kSPDIF_UQChannelFrameError (*C enumerator*), 874
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kSPDIF_UQChannelSync (*C enumerator*), 873
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kSPDIF_ValidityFlagNoGood (*C enumerator*), 873
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_Busy (*C enumerator*), 621
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_EDMA_Busy (*C enumerator*), 290
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_EDMA_QueueFull (*C enumerator*), 290
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_ENET_InitMemoryFail (*C enumerator*), 322
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_ENET_RxFrameDrop (*C enumerator*), 322
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_ENET_RxFrameEmpty (*C enumerator*), 322
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_ENET_RxFrameError (*C enumerator*), 322
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_ENET_RxFrameFail (*C enumerator*), 322
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_ENET_TxFrameBusy (*C enumerator*), 322
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_ENET_TxFrameFail (*C enumerator*), 322
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_ENET_TxFrameOverLen (*C enumerator*), 322
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_Fail (*C enumerator*), 621
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_FLEXCAN_ErrorStatus (*C enumerator*), 365
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_FLEXCAN_MemoryError (*C enumerator*), 365
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_FLEXCAN_RxBusy (*C enumerator*), 365
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_FLEXCAN_RxFifoBusy (*C enumerator*), 365
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_FLEXCAN_RxFifoDisabled (*C enumerator*), 365
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_FLEXCAN_RxFifoIdle (*C enumerator*), 365
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_FLEXCAN_RxFifoOverflow (*C enumerator*), 365
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_FLEXCAN_RxFifoWarning (*C enumerator*), 365
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_FLEXCAN_RxIdle (*C enumerator*), 365
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_FLEXCAN_RxOverflow (*C enumerator*), 365
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_FLEXCAN_RxRemote (*C enu-*

merator), 365
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_FLEXCAN_TxBusy (*C enumerator*), 365
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_FLEXCAN_TxIdle (*C enumerator*), 365
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_FLEXCAN_TxSwitchToRx (*C enumerator*), 365
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_FLEXCAN_UnHandled (*C enumerator*), 365
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_FLEXCAN_WakeUp (*C enumerator*), 365
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_FLEXIO_I2C_Busy (*C enumerator*), 430
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_FLEXIO_I2C_Idle (*C enumerator*), 430
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_FLEXIO_I2C_Nak (*C enumerator*), 430
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_FLEXIO_I2C_Timeout (*C enumerator*), 430
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_FLEXIO_I2S_Error (*C enumerator*), 440
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_FLEXIO_I2S_Idle (*C enumerator*), 440
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_FLEXIO_I2S_QueueFull (*C enumerator*), 440
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_FLEXIO_I2S_RxBusy (*C enumerator*), 440
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_FLEXIO_I2S_Timeout (*C enumerator*), 440
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_FLEXIO_I2S_TxBusy (*C enumerator*), 440
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_FLEXIO_SPI_Busy (*C enumerator*), 453
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_FLEXIO_SPI_Error (*C enumerator*), 453
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_FLEXIO_SPI_Idle (*C enumerator*), 453
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_FLEXIO_SPI_Timeout (*C enumerator*), 454
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_FLEXIO_UART_BaudrateNotSupport (*C enumerator*), 467
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_FLEXIO_UART_ERROR (*C enumerator*), 467
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_FLEXIO_UART_RxBusy (*C enumerator*), 467
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_FLEXIO_UART_RxHardwareOverrun (*C enumerator*), 467
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_FLEXIO_UART_RxIdle (*C enumerator*), 467
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_FLEXIO_UART_RxRingBufferOverrun (*C enumerator*), 467
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_FLEXIO_UART_Timeout (*C enumerator*), 467
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_FLEXIO_UART_TxBusy (*C enumerator*), 466
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_FLEXIO_UART_TxIdle (*C enumerator*), 467
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_FLEXSPI_Busy (*C enumera-*

tor), 546
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_FLEXSPI_IpCommandGrantTimeout (*C enumerator*), 546
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_FLEXSPI_IpCommandSequenceError (*C enumerator*), 546
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_FLEXSPI_SequenceExecutionTimeout (*C enumerator*), 546
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_InvalidArgument (*C enumerator*), 621
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_LPI2C_ArbitrationLost (*C enumerator*), 625
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_LPI2C_BitError (*C enumerator*), 625
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_LPI2C_Busy (*C enumerator*), 625
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_LPI2C_DmaRequestFail (*C enumerator*), 625
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_LPI2C_FifoError (*C enumerator*), 625
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_LPI2C_Idle (*C enumerator*), 625
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_LPI2C_Nak (*C enumerator*), 625
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_LPI2C_NoTransferInProgress (*C enumerator*), 625
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_LPI2C_PinLowTimeout (*C enumerator*), 625
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_LPI2C_Timeout (*C enumerator*), 625
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_LPSPI_Busy (*C enumerator*), 666
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_LPSPI_Error (*C enumerator*), 666
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_LPSPI_Idle (*C enumerator*), 666
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_LPSPI_OutOfRange (*C enumerator*), 666
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_LPSPI_Timeout (*C enumerator*), 666
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_LPUART_BaudrateNotSupport (*C enumerator*), 700
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_LPUART_Error (*C enumerator*), 699
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_LPUART_FlagCannotClearManually (*C enumerator*), 699
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_LPUART_FramingError (*C enumerator*), 699
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_LPUART_IdleLineDetected (*C enumerator*), 700
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_LPUART_NoiseError (*C enumerator*), 699
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_LPUART_ParityError (*C enumerator*), 699
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_LPUART_RxBusy (*C enumerator*), 699
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_LPUART_RxHardwareOverrun (*C enumerator*), 699
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_LPUART_RxIdle (*C enumera-*

tor), 699
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_LPUART_RxRingBufferOverrun (*C enumerator*), 699
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_LPUART_RxWatermarkTooLarge (*C enumerator*), 699
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_LPUART_Timeout (*C enumerator*), 700
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_LPUART_TxBusy (*C enumerator*), 699
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_LPUART_TxIdle (*C enumerator*), 699
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_LPUART_TxWatermarkTooLarge (*C enumerator*), 699
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_NoData (*C enumerator*), 621
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_NoTransferInProgress (*C enumerator*), 621
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_OCOTP_AccessError (*C enumerator*), 712
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_OCOTP_CrcFail (*C enumerator*), 712
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_OCOTP_Locked (*C enumerator*), 712
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_OCOTP_ProgramFail (*C enumerator*), 712
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_OCOTP_ReloadError (*C enumerator*), 712
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_OutOfRange (*C enumerator*), 621
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_ReadOnly (*C enumerator*), 621
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_SAI_QueueFull (*C enumerator*), 790
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_SAI_RxBusy (*C enumerator*), 790
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_SAI_RxError (*C enumerator*), 790
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_SAI_RxIdle (*C enumerator*), 790
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_SAI_TxBusy (*C enumerator*), 790
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_SAI_TxError (*C enumerator*), 790
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_SAI_TxIdle (*C enumerator*), 790
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_SEMC_AxiCommandExecutionError (*C enumerator*), 816
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_SEMC_InvalidAddressPortWidth (*C enumerator*), 816
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_SEMC_InvalidBaseAddress (*C enumerator*), 817
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_SEMC_InvalidBurstLength (*C enumerator*), 816
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_SEMC_InvalidColumnAddressBitWidth (*C enumerator*), 817
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_SEMC_InvalidDataPortWidth (*C enumerator*), 816
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_SEMC_InvalidDeviceType (*C enumerator*), 816
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_SEMC_InvalidIpcmdDataSize

(*C enumerator*), 816
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_SEMC_InvalidMemorySize (*C enumerator*), 816
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_SEMC_InvalidSwPinmuxSelection (*C enumerator*), 816
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_SEMC_InvalidTimerSetting (*C enumerator*), 817
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_SEMC_IpCommandExecutionError (*C enumerator*), 816
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_SPDIF_LockLoss (*C enumerator*), 871
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_SPDIF_QChannelOverrun (*C enumerator*), 870
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_SPDIF_QueueFull (*C enumerator*), 871
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_SPDIF_RxCnew (*C enumerator*), 870
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_SPDIF_RxDPLLLocked (*C enumerator*), 870
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_SPDIF_RxFIFOError (*C enumerator*), 870
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_SPDIF_RxFIFOResync (*C enumerator*), 870
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_SPDIF_RxIdle (*C enumerator*), 871
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_SPDIF_RxIllegalSymbol (*C enumerator*), 870
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_SPDIF_RxParityBitError (*C enumerator*), 870
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_SPDIF_TxFIFOError (*C enumerator*), 870
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_SPDIF_TxFIFOResync (*C enumerator*), 870
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_SPDIF_TxIdle (*C enumerator*), 871
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_SPDIF_UChannelOverrun (*C enumerator*), 870
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_SPDIF_UQChannelFrameError (*C enumerator*), 870
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_SPDIF_UQChannelSync (*C enumerator*), 870
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_SPDIF_ValidatyNoGood (*C enumerator*), 870
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_Success (*C enumerator*), 621
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_Timeout (*C enumerator*), 621
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_USDHC_BusyTransferring (*C enumerator*), 905
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_USDHC_DMADataAddrNotAlign (*C enumerator*), 906
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_USDHC_NotSupport (*C enumerator*), 906
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_USDHC_PrepareAdmaDescriptorFailed (*C enumerator*), 905
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_USDHC_ReTuningRequest (*C enumerator*), 906
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_USDHC_SendCommandFailed (*C enumerator*), 905
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_USDHC_SendCommandSuccess

(*C enumerator*), 906
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_USDHC_TransferDataComplete (*C enumerator*), 906
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_USDHC_TransferDataFailed (*C enumerator*), 906
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_USDHC_TransferDMAComplete (*C enumerator*), 906
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kStatus_USDHC_TuningError (*C enumerator*), 906
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_Adma1DescriptorActivity1Flag (*C enumerator*), 917
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_Adma1DescriptorActivity2Flag (*C enumerator*), 918
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_Adma1DescriptorEndFlag (*C enumerator*), 917
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_Adma1DescriptorInterrupFlag (*C enumerator*), 917
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_Adma1DescriptorTypeLink (*C enumerator*), 918
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_Adma1DescriptorTypeNop (*C enumerator*), 918
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_Adma1DescriptorTypeSetLength (*C enumerator*), 918
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_Adma1DescriptorTypeTransfer (*C enumerator*), 918
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_Adma1DescriptorValidFlag (*C enumerator*), 917
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_Adma2DescriptorActivity1Flag (*C enumerator*), 918
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_Adma2DescriptorActivity2Flag (*C enumerator*), 918
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_Adma2DescriptorEndFlag (*C enumerator*), 918
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_Adma2DescriptorInterruptFlag (*C enumerator*), 918
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_Adma2DescriptorTypeLink (*C enumerator*), 918
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_Adma2DescriptorTypeNop (*C enumerator*), 918
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_Adma2DescriptorTypeReserved (*C enumerator*), 918
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_Adma2DescriptorTypeTransfer (*C enumerator*), 918
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_Adma2DescriptorValidFlag (*C enumerator*), 918
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_AdmaDescriptorErrorFlag (*C enumerator*), 913
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_AdmaDescriptorMultipleFlag (*C enumerator*), 919
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_AdmaDescriptorSingleFlag (*C enumerator*), 919
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_AdmaErrorState (*C enumerator*), 913
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_AdmaErrorStateChangeAddress (*C enumerator*), 913
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_AdmaErrorStateFetchDescriptor (*C enumerator*), 913
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_AdmaErrorStateInvalidDescriptor

(*C enumerator*), 913
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_AdmaErrorStateInvalidLength (*C enumerator*), 913
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_AdmaErrorStateStopDma (*C enumerator*), 913
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_AdmaErrorStateTransferData (*C enumerator*), 913
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_AdmaLenghMismatchFlag (*C enumerator*), 913
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_AllInterruptFlags (*C enumerator*), 912
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_AutoCommand12CrcErrorFlag (*C enumerator*), 912
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_AutoCommand12EndBitErrorFlag (*C enumerator*), 912
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_AutoCommand12ErrorFlag (*C enumerator*), 911
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_AutoCommand12IndexErrorFlag (*C enumerator*), 912
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_AutoCommand12NotExecutedFlag (*C enumerator*), 912
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_AutoCommand12NotIssuedFlag (*C enumerator*), 912
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_AutoCommand12TimeoutFlag (*C enumerator*), 912
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_BlockGapEventFlag (*C enumerator*), 910
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_BufferReadEnableFlag (*C enumerator*), 909
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_BufferReadReadyFlag (*C enumerator*), 910
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_BufferWriteEnableFlag (*C enumerator*), 909
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_BufferWriteReadyFlag (*C enumerator*), 910
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_CardDetectFlag (*C enumerator*), 912
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_CardInsertedFlag (*C enumerator*), 909
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_CardInsertionFlag (*C enumerator*), 910
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_CardInterruptFlag (*C enumerator*), 910
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_CardRemovalFlag (*C enumerator*), 910
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_CommandCompleteFlag (*C enumerator*), 910
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_CommandCrcErrorFlag (*C enumerator*), 911
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_CommandEndBitErrorFlag (*C enumerator*), 911
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_CommandErrorFlag (*C enumerator*), 911
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_CommandFlag (*C enumerator*), 912
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_CommandIndexErrorFlag (*C enumerator*), 911
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_CommandInhibitFlag (*C enu-*

merator), 909
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_CommandLineLevelFlag (*C enumerator*), 909
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_CommandTimeoutFlag (*C enumerator*), 911
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_CommandTypeAbortFlag (*C enumerator*), 908
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_CommandTypeResumeFlag (*C enumerator*), 908
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_CommandTypeSuspendFlag (*C enumerator*), 908
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_Data0LineLevelFlag (*C enumerator*), 909
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_Data1LineLevelFlag (*C enumerator*), 909
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_Data2LineLevelFlag (*C enumerator*), 909
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_Data3LineLevelFlag (*C enumerator*), 910
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_Data4LineLevelFlag (*C enumerator*), 910
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_Data5LineLevelFlag (*C enumerator*), 910
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_Data6LineLevelFlag (*C enumerator*), 910
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_Data7LineLevelFlag (*C enumerator*), 910
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_DataCompleteFlag (*C enumerator*), 910
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_DataCrcErrorFlag (*C enumerator*), 911
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_DataDMAFlag (*C enumerator*), 911
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_DataEndBitErrorFlag (*C enumerator*), 911
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_DataErrorFlag (*C enumerator*), 911
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_DataFlag (*C enumerator*), 911
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_DataInhibitFlag (*C enumerator*), 909
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_DataLineActiveFlag (*C enumerator*), 909
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_DataPresentFlag (*C enumerator*), 908
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_DataReadFlag (*C enumerator*), 908
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_DataTimeoutFlag (*C enumerator*), 911
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_DelaySettingFinishedFlag (*C enumerator*), 909
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_DmaCompleteFlag (*C enumerator*), 910
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_DmaErrorFlag (*C enumerator*), 911
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_EnableAutoCommand12Flag (*C enumerator*), 908
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_EnableAutoCommand23Flag

(*C enumerator*), 908
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_EnableBlockCountFlag (*C enumerator*), 908
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_EnableCrcCheckFlag (*C enumerator*), 908
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_EnableDmaFlag (*C enumerator*), 908
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_EnableIndexCheckFlag (*C enumerator*), 908
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_ErrorFlag (*C enumerator*), 911
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_ExactBlockNumberReadFlag (*C enumerator*), 916
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_ExecuteTuning (*C enumerator*), 912
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_ForceEventAutoCommand12CrcError (*C enumerator*), 913
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_ForceEventAutoCommand12Error (*C enumerator*), 914
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_ForceEventAutoCommand12IndexError (*C enumerator*), 914
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_ForceEventAutoCommand12NotExecuted (*C enumerator*), 913
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_ForceEventAutoCommand12NotIssued (*C enumerator*), 914
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_ForceEventAutoCommand12Timeout (*C enumerator*), 913
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_ForceEventCardInt (*C enumerator*), 914
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_ForceEventCommandCrcError (*C enumerator*), 914
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_ForceEventCommandEndBitError (*C enumerator*), 914
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_ForceEventCommandIndexError (*C enumerator*), 914
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_ForceEventCommandTimeout (*C enumerator*), 914
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_ForceEventDataCrcError (*C enumerator*), 914
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_ForceEventDataEndBitError (*C enumerator*), 914
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_ForceEventDataTimeout (*C enumerator*), 914
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_ForceEventDmaError (*C enumerator*), 914
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_ForceEventEndBitError (*C enumerator*), 913
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_ForceEventsAll (*C enumerator*), 914
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_ForceEventTuningError (*C enumerator*), 914
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_InterruptAtBlockGapFlag (*C enumerator*), 916
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_MultipleBlockFlag (*C enumerator*), 908
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_ReadDoneNo8CLK (*C enumerator*), 916
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_ReadTransferActiveFlag (*C*

enumerator), 909
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_ReadWaitControlFlag (*C enumerator*), 916
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_ResetAll (*C enumerator*), 907
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_ResetCommand (*C enumerator*), 907
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_ResetData (*C enumerator*), 907
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_ResetsAll (*C enumerator*), 907
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_ResetTuning (*C enumerator*), 907
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_ResponseLength48BusyFlag (*C enumerator*), 908
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_ResponseLength48Flag (*C enumerator*), 908
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_ResponseLength136Flag (*C enumerator*), 908
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_ReTuningEventFlag (*C enumerator*), 910
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_ReTuningRequestFlag (*C enumerator*), 909
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_SdClockStableFlag (*C enumerator*), 909
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_SDR104TuningFlag (*C enumerator*), 912
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_StopAtBlockGapFlag (*C enumerator*), 916
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_Support4BitFlag (*C enumerator*), 906
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_Support8BitFlag (*C enumerator*), 907
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_SupportAdmaFlag (*C enumerator*), 906
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_SupportDDR50Flag (*C enumerator*), 907
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_SupportDmaFlag (*C enumerator*), 906
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_SupportHighSpeedFlag (*C enumerator*), 906
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_SupportSDR50Flag (*C enumerator*), 907
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_SupportSDR104Flag (*C enumerator*), 907
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_SupportSuspendResumeFlag (*C enumerator*), 906
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_SupportV300Flag (*C enumerator*), 906
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_SupportV330Flag (*C enumerator*), 906
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_TransferDataBoot (*C enumerator*), 919
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_TransferDataBootcontinous (*C enumerator*), 919
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_TransferDataNormal (*C enumerator*), 919
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_TransferDataTuning (*C enumerator*), 919

drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_TuningErrorFlag (*C enumerator*), 911
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_TuningPassFlag (*C enumerator*), 911
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_TuningSampleClockSel (*C enumerator*), 912
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_WakeupEventOnCardInsert (*C enumerator*), 907
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_WakeupEventOnCardInt (*C enumerator*), 907
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_WakeupEventOnCardRemove (*C enumerator*), 907
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_WakeupEventsAll (*C enumerator*), 907
drivers_RT_RT1020_MIMXRT1024.PhonyNameDueToError.kUSDHC_WriteTransferActiveFlag (*C enumerator*), 909
drivers_RT_RT1020_MIMXRT1024.pit_chnl_t (*C type*), 719
drivers_RT_RT1020_MIMXRT1024.PIT_ClearStatusFlags (*C function*), 717
drivers_RT_RT1020_MIMXRT1024.PIT_CLOCKS (*C macro*), 236
drivers_RT_RT1020_MIMXRT1024.pit_config_t (*C type*), 720
drivers_RT_RT1020_MIMXRT1024.PIT_Deinit (*C function*), 716
drivers_RT_RT1020_MIMXRT1024.PIT_DisableInterrupts (*C function*), 717
drivers_RT_RT1020_MIMXRT1024.PIT_EnableInterrupts (*C function*), 717
drivers_RT_RT1020_MIMXRT1024.PIT_GetCurrentTimerCount (*C function*), 718
drivers_RT_RT1020_MIMXRT1024.PIT_GetDefaultConfig (*C function*), 716
drivers_RT_RT1020_MIMXRT1024.PIT_GetEnabledInterrupts (*C function*), 717
drivers_RT_RT1020_MIMXRT1024.PIT_GetLifetimeTimerCount (*C function*), 720
drivers_RT_RT1020_MIMXRT1024.PIT_GetStatusFlags (*C function*), 717
drivers_RT_RT1020_MIMXRT1024.PIT_Init (*C function*), 716
drivers_RT_RT1020_MIMXRT1024.pit_interrupt_enable_t (*C type*), 719
drivers_RT_RT1020_MIMXRT1024.PIT_SetTimerChainMode (*C function*), 716
drivers_RT_RT1020_MIMXRT1024.PIT_SetTimerPeriod (*C function*), 718
drivers_RT_RT1020_MIMXRT1024.PIT_StartTimer (*C function*), 718
drivers_RT_RT1020_MIMXRT1024.pit_status_flags_t (*C type*), 719
drivers_RT_RT1020_MIMXRT1024.PIT_StopTimer (*C function*), 718
drivers_RT_RT1020_MIMXRT1024.PLL_AUDIO_OFFSET (*C macro*), 234
drivers_RT_RT1020_MIMXRT1024.PLL_ENET_OFFSET (*C macro*), 234
drivers_RT_RT1020_MIMXRT1024.PLL_SYS_OFFSET (*C macro*), 234
drivers_RT_RT1020_MIMXRT1024.PLL_USB1_OFFSET (*C macro*), 234
drivers_RT_RT1020_MIMXRT1024.pmu_1p1_weak_reference_source_t (*C type*), 727
drivers_RT_RT1020_MIMXRT1024.PMU_1P1EnableBrownout (*C function*), 722
drivers_RT_RT1020_MIMXRT1024.PMU_1P1EnableCurrentLimit (*C function*), 721
drivers_RT_RT1020_MIMXRT1024.PMU_1P1EnableOutput (*C function*), 722
drivers_RT_RT1020_MIMXRT1024.PMU_1P1EnablePullDown (*C function*), 721
drivers_RT_RT1020_MIMXRT1024.PMU_1P1EnableWeakRegulator (*C function*), 721
drivers_RT_RT1020_MIMXRT1024.PMU_1P1SetBrownoutOffsetVoltage (*C function*), 721
drivers_RT_RT1020_MIMXRT1024.PMU_1P1SetRegulatorOutputVoltage (*C function*), 721
drivers_RT_RT1020_MIMXRT1024.PMU_1P1SetWeakReferenceSource (*C function*), 720
drivers_RT_RT1020_MIMXRT1024.PMU_2P1EnablePullDown (*C function*), 724
drivers_RT_RT1020_MIMXRT1024.PMU_2P5EnableCurrentLimit (*C function*), 724
drivers_RT_RT1020_MIMXRT1024.PMU_2P5EnableOutput (*C function*), 724
drivers_RT_RT1020_MIMXRT1024.PMU_2P5EnablePullDown (*C function*), 724
drivers_RT_RT1020_MIMXRT1024.PMU_2P5EnableWeakRegulator (*C function*), 723
drivers_RT_RT1020_MIMXRT1024.PMU_2P5nableBrownout (*C function*), 724
drivers_RT_RT1020_MIMXRT1024.PMU_2P5SetBrownoutOffsetVoltage (*C function*), 723
drivers_RT_RT1020_MIMXRT1024.PMU_2P5SetRegulatorOutputVoltage (*C function*), 723
drivers_RT_RT1020_MIMXRT1024.pmu_3p0_vbus_voltage_source_t (*C type*), 727
drivers_RT_RT1020_MIMXRT1024.PMU_3P0EnableBrownout (*C function*), 723

drivers_RT_RT1020_MIMXRT1024.PMU_3P0EnableCurrentLimit (*C function*), 722
drivers_RT_RT1020_MIMXRT1024.PMU_3P0EnableOutput (*C function*), 723
drivers_RT_RT1020_MIMXRT1024.PMU_3P0SetBrownoutOffsetVoltage (*C function*), 722
drivers_RT_RT1020_MIMXRT1024.PMU_3P0SetRegulatorOutputVoltage (*C function*), 722
drivers_RT_RT1020_MIMXRT1024.PMU_3P0SetVBusVoltageSource (*C function*), 722
drivers_RT_RT1020_MIMXRT1024.pmu_core_reg_voltage_ramp_rate_t (*C type*), 727
drivers_RT_RT1020_MIMXRT1024.PMU_CoreEnableIncreaseGateDrive (*C function*), 724
drivers_RT_RT1020_MIMXRT1024.PMU_CoreSetARMCoreDomainVoltage (*C function*), 725
drivers_RT_RT1020_MIMXRT1024.PMU_CoreSetRegulatorVoltageRampRate (*C function*), 724
drivers_RT_RT1020_MIMXRT1024.PMU_CoreSetSOCDomainVoltage (*C function*), 725
drivers_RT_RT1020_MIMXRT1024.PMU_GetStatusFlags (*C function*), 720
drivers_RT_RT1020_MIMXRT1024.pmu_power_bandgap_t (*C type*), 727
drivers_RT_RT1020_MIMXRT1024.PWM_ActivateOutputTrigger (*C function*), 750
drivers_RT_RT1020_MIMXRT1024.pwm_channels_t (*C type*), 745
drivers_RT_RT1020_MIMXRT1024.pwm_chnl_pair_operation_t (*C type*), 745
drivers_RT_RT1020_MIMXRT1024.PWM_ClearStatusFlags (*C function*), 732
drivers_RT_RT1020_MIMXRT1024.pwm_clock_prescale_t (*C type*), 745
drivers_RT_RT1020_MIMXRT1024.pwm_clock_source_t (*C type*), 745
drivers_RT_RT1020_MIMXRT1024.PWM_CLOCKS (*C macro*), 236
drivers_RT_RT1020_MIMXRT1024.pwm_config_t (*C type*), 746
drivers_RT_RT1020_MIMXRT1024.PWM_DeactivateOutputTrigger (*C function*), 750
drivers_RT_RT1020_MIMXRT1024.PWM_Deinit (*C function*), 728
drivers_RT_RT1020_MIMXRT1024.PWM_DisableInterrupts (*C function*), 730
drivers_RT_RT1020_MIMXRT1024.pwm_dma_enable_t (*C type*), 746
drivers_RT_RT1020_MIMXRT1024.pwm_dma_source_select_t (*C type*), 746
drivers_RT_RT1020_MIMXRT1024.PWM_DMACaptureSourceSelect (*C function*), 731
drivers_RT_RT1020_MIMXRT1024.PWM_DMAFIFOWatermarkControl (*C function*), 731
drivers_RT_RT1020_MIMXRT1024.PWM_EnableDMACapture (*C function*), 731
drivers_RT_RT1020_MIMXRT1024.PWM_EnableDMAWrite (*C function*), 731
drivers_RT_RT1020_MIMXRT1024.PWM_EnableInterrupts (*C function*), 730
drivers_RT_RT1020_MIMXRT1024.PWM_EnableLocalForce (*C function*), 749
drivers_RT_RT1020_MIMXRT1024.pwm_fault_channels_t (*C type*), 745
drivers_RT_RT1020_MIMXRT1024.pwm_fault_clear_t (*C type*), 746
drivers_RT_RT1020_MIMXRT1024.pwm_fault_disable_t (*C type*), 745
drivers_RT_RT1020_MIMXRT1024.pwm_fault_input_filter_param_t (*C type*), 747
drivers_RT_RT1020_MIMXRT1024.pwm_fault_input_t (*C type*), 745
drivers_RT_RT1020_MIMXRT1024.pwm_fault_param_t (*C type*), 747
drivers_RT_RT1020_MIMXRT1024.pwm_fault_recovery_mode_t (*C type*), 746
drivers_RT_RT1020_MIMXRT1024.pwm_fault_state_t (*C type*), 746
drivers_RT_RT1020_MIMXRT1024.PWM_FaultDefaultConfig (*C function*), 748
drivers_RT_RT1020_MIMXRT1024.pwm_force_output_trigger_t (*C type*), 745
drivers_RT_RT1020_MIMXRT1024.pwm_force_signal_t (*C type*), 745
drivers_RT_RT1020_MIMXRT1024.PWM_GetDefaultConfig (*C function*), 728
drivers_RT_RT1020_MIMXRT1024.PWM_GetEnabledInterrupts (*C function*), 731
drivers_RT_RT1020_MIMXRT1024.PWM_GetInputCaptureValue (*C function*), 747
drivers_RT_RT1020_MIMXRT1024.PWM_GetPwmChannelState (*C function*), 752
drivers_RT_RT1020_MIMXRT1024.PWM_GetStatusFlags (*C function*), 732
drivers_RT_RT1020_MIMXRT1024.PWM_GetVALxValue (*C function*), 749
drivers_RT_RT1020_MIMXRT1024.PWM_Init (*C function*), 727
drivers_RT_RT1020_MIMXRT1024.pwm_init_source_t (*C type*), 745
drivers_RT_RT1020_MIMXRT1024.pwm_input_capture_edge_t (*C type*), 745
drivers_RT_RT1020_MIMXRT1024.pwm_input_capture_param_t (*C type*), 747
drivers_RT_RT1020_MIMXRT1024.pwm_interrupt_enable_t (*C type*), 746
drivers_RT_RT1020_MIMXRT1024.pwm_level_select_t (*C type*), 746
drivers_RT_RT1020_MIMXRT1024.pwm_load_frequency_t (*C type*), 745
drivers_RT_RT1020_MIMXRT1024.pwm_mode_t (*C type*), 746
drivers_RT_RT1020_MIMXRT1024.pwm_module_control_t (*C type*), 746
drivers_RT_RT1020_MIMXRT1024.pwm_output_state_t (*C type*), 745

drivers_RT_RT1020_MIMXRT1024.PWM_OutputDisable (*C function*), 752
drivers_RT_RT1020_MIMXRT1024.PWM_OutputEnable (*C function*), 751
drivers_RT_RT1020_MIMXRT1024.PWM_OutputTriggerEnable (*C function*), 750
drivers_RT_RT1020_MIMXRT1024.pwm_register_reload_t (*C type*), 746
drivers_RT_RT1020_MIMXRT1024.pwm_reload_source_select_t (*C type*), 746
drivers_RT_RT1020_MIMXRT1024.PWM_SetChannelOutput (*C function*), 753
drivers_RT_RT1020_MIMXRT1024.PWM_SetClockMode (*C function*), 752
drivers_RT_RT1020_MIMXRT1024.PWM_SetFilterSampleCount (*C function*), 753
drivers_RT_RT1020_MIMXRT1024.PWM_SetFilterSamplePeriod (*C function*), 753
drivers_RT_RT1020_MIMXRT1024.PWM_SetOutputToIdle (*C function*), 752
drivers_RT_RT1020_MIMXRT1024.PWM_SetPhaseDelay (*C function*), 753
drivers_RT_RT1020_MIMXRT1024.PWM_SetPwmFaultState (*C function*), 751
drivers_RT_RT1020_MIMXRT1024.PWM_SetPwmForceOutputToZero (*C function*), 753
drivers_RT_RT1020_MIMXRT1024.PWM_SetPwmLdok (*C function*), 750
drivers_RT_RT1020_MIMXRT1024.PWM_SetupFaultDisableMap (*C function*), 751
drivers_RT_RT1020_MIMXRT1024.PWM_SetupFaultInputFilter (*C function*), 747
drivers_RT_RT1020_MIMXRT1024.PWM_SetupFaultInputFilterExt (*C function*), 748
drivers_RT_RT1020_MIMXRT1024.PWM_SetupFaults (*C function*), 748
drivers_RT_RT1020_MIMXRT1024.PWM_SetupFaultsExt (*C function*), 748
drivers_RT_RT1020_MIMXRT1024.PWM_SetupForceSignal (*C function*), 748
drivers_RT_RT1020_MIMXRT1024.PWM_SetupInputCapture (*C function*), 747
drivers_RT_RT1020_MIMXRT1024.PWM_SetupPwm (*C function*), 728
drivers_RT_RT1020_MIMXRT1024.PWM_SetupPwmPhaseShift (*C function*), 729
drivers_RT_RT1020_MIMXRT1024.PWM_SetupSwCtrlOut (*C function*), 750
drivers_RT_RT1020_MIMXRT1024.PWM_SetVALxValue (*C function*), 749
drivers_RT_RT1020_MIMXRT1024.pwm_signal_param_t (*C type*), 746
drivers_RT_RT1020_MIMXRT1024.PWM_StartTimer (*C function*), 732
drivers_RT_RT1020_MIMXRT1024.pwm_status_flags_t (*C type*), 746
drivers_RT_RT1020_MIMXRT1024.PWM_StopTimer (*C function*), 732
drivers_RT_RT1020_MIMXRT1024.PWM_SUBMODULE_CHANNEL (*C macro*), 754
drivers_RT_RT1020_MIMXRT1024.PWM_SUBMODULE_SWCONTROL_WIDTH (*C macro*), 754
drivers_RT_RT1020_MIMXRT1024.pwm_submodule_t (*C type*), 745
drivers_RT_RT1020_MIMXRT1024.PWM_UpdateCurrentPolarity (*C function*), 749
drivers_RT_RT1020_MIMXRT1024.PWM_UpdatePwmDutycycle (*C function*), 729
drivers_RT_RT1020_MIMXRT1024.PWM_UpdatePwmDutycycleHighAccuracy (*C function*), 729
drivers_RT_RT1020_MIMXRT1024.PWM_UpdatePwmPeriodAndDutycycle (*C function*), 730
drivers_RT_RT1020_MIMXRT1024.pwm_value_register_t (*C type*), 745
drivers_RT_RT1020_MIMXRT1024.pwm_watermark_control_t (*C type*), 746
drivers_RT_RT1020_MIMXRT1024.qtmr_channel_selection_t (*C type*), 766
drivers_RT_RT1020_MIMXRT1024.QTMR_ClearStatusFlags (*C function*), 758
drivers_RT_RT1020_MIMXRT1024.qtmr_config_t (*C type*), 766
drivers_RT_RT1020_MIMXRT1024.qtmr_counting_mode_t (*C type*), 765
drivers_RT_RT1020_MIMXRT1024.qtmr_debug_action_t (*C type*), 766
drivers_RT_RT1020_MIMXRT1024.QTMR_Deinit (*C function*), 757
drivers_RT_RT1020_MIMXRT1024.QTMR_DisableDma (*C function*), 760
drivers_RT_RT1020_MIMXRT1024.QTMR_DisableInterrupts (*C function*), 757
drivers_RT_RT1020_MIMXRT1024.qtmr_dma_enable_t (*C type*), 766
drivers_RT_RT1020_MIMXRT1024.QTMR_EnableDma (*C function*), 759
drivers_RT_RT1020_MIMXRT1024.QTMR_EnableInterrupts (*C function*), 757
drivers_RT_RT1020_MIMXRT1024.QTMR_GetCurrentTimerCount (*C function*), 759
drivers_RT_RT1020_MIMXRT1024.QTMR_GetDefaultConfig (*C function*), 757
drivers_RT_RT1020_MIMXRT1024.QTMR_GetEnabledInterrupts (*C function*), 757
drivers_RT_RT1020_MIMXRT1024.QTMR_GetPwmChannelStatus (*C function*), 767
drivers_RT_RT1020_MIMXRT1024.QTMR_GetPwmOutputStatus (*C function*), 767
drivers_RT_RT1020_MIMXRT1024.QTMR_GetStatus (*C function*), 758
drivers_RT_RT1020_MIMXRT1024.QTMR_Init (*C function*), 756
drivers_RT_RT1020_MIMXRT1024.qtmr_input_capture_edge_t (*C type*), 765
drivers_RT_RT1020_MIMXRT1024.qtmr_input_source_t (*C type*), 765

drivers_RT_RT1020_MIMXRT1024.qtmr_interrupt_enable_t (*C type*), 766
drivers_RT_RT1020_MIMXRT1024.qtmr_output_mode_t (*C type*), 765
drivers_RT_RT1020_MIMXRT1024.qtmr_preload_control_t (*C type*), 765
drivers_RT_RT1020_MIMXRT1024.qtmr_primary_count_source_t (*C type*), 765
drivers_RT_RT1020_MIMXRT1024.qtmr_pwm_out_state_t (*C type*), 765
drivers_RT_RT1020_MIMXRT1024.QTMR_SetCompareValue (*C function*), 758
drivers_RT_RT1020_MIMXRT1024.QTMR_SetLoadValue (*C function*), 759
drivers_RT_RT1020_MIMXRT1024.QTMR_SetPwmClockMode (*C function*), 767
drivers_RT_RT1020_MIMXRT1024.QTMR_SetPwmOutputToIdle (*C function*), 767
drivers_RT_RT1020_MIMXRT1024.QTMR_SetTimerPeriod (*C function*), 758
drivers_RT_RT1020_MIMXRT1024.QTMR_SetupInputCapture (*C function*), 766
drivers_RT_RT1020_MIMXRT1024.QTMR_SetupPwm (*C function*), 766
drivers_RT_RT1020_MIMXRT1024.QTMR_StartTimer (*C function*), 759
drivers_RT_RT1020_MIMXRT1024.qtmr_status_flags_t (*C type*), 766
drivers_RT_RT1020_MIMXRT1024.QTMR_StopTimer (*C function*), 759
drivers_RT_RT1020_MIMXRT1024.qtmrRegType (*C type*), 765
drivers_RT_RT1020_MIMXRT1024.RAMFUNCTION_SECTION_CODE (*C macro*), 612
drivers_RT_RT1020_MIMXRT1024.RTWDOG_ClearStatusFlags (*C function*), 771
drivers_RT_RT1020_MIMXRT1024.rtwdog_clock_prescaler_t (*C type*), 774
drivers_RT_RT1020_MIMXRT1024.rtwdog_clock_source_t (*C type*), 774
drivers_RT_RT1020_MIMXRT1024.RTWDOG_CLOCKS (*C macro*), 237
drivers_RT_RT1020_MIMXRT1024.rtwdog_config_t (*C type*), 774
drivers_RT_RT1020_MIMXRT1024.RTWDOG_CountToMesec (*C function*), 771
drivers_RT_RT1020_MIMXRT1024.RTWDOG_Deinit (*C function*), 769
drivers_RT_RT1020_MIMXRT1024.RTWDOG_Disable (*C function*), 769
drivers_RT_RT1020_MIMXRT1024.RTWDOG_DisableInterrupts (*C function*), 770
drivers_RT_RT1020_MIMXRT1024.RTWDOG_Enable (*C function*), 769
drivers_RT_RT1020_MIMXRT1024.RTWDOG_EnableInterrupts (*C function*), 770
drivers_RT_RT1020_MIMXRT1024.RTWDOG_EnableWindowMode (*C function*), 770
drivers_RT_RT1020_MIMXRT1024.RTWDOG_GetCounterValue (*C function*), 772
drivers_RT_RT1020_MIMXRT1024.RTWDOG_GetDefaultConfig (*C function*), 768
drivers_RT_RT1020_MIMXRT1024.RTWDOG_GetStatusFlags (*C function*), 770
drivers_RT_RT1020_MIMXRT1024.RTWDOG_Init (*C function*), 769
drivers_RT_RT1020_MIMXRT1024.RTWDOG_Refresh (*C function*), 772
drivers_RT_RT1020_MIMXRT1024.RTWDOG_SetTimeoutValue (*C function*), 771
drivers_RT_RT1020_MIMXRT1024.RTWDOG_SetWindowValue (*C function*), 771
drivers_RT_RT1020_MIMXRT1024.rtwdog_test_mode_t (*C type*), 774
drivers_RT_RT1020_MIMXRT1024.rtwdog_work_mode_t (*C type*), 774
drivers_RT_RT1020_MIMXRT1024.s_enetClock (*C var*), 332
drivers_RT_RT1020_MIMXRT1024.s_enetExtraClock (*C var*), 332
drivers_RT_RT1020_MIMXRT1024.s_flexioBases (*C var*), 410
drivers_RT_RT1020_MIMXRT1024.s_flexioClocks (*C var*), 410
drivers_RT_RT1020_MIMXRT1024.s_lpi2cMasterHandle (*C var*), 626
drivers_RT_RT1020_MIMXRT1024.s_lpi2cMasterIsr (*C var*), 626
drivers_RT_RT1020_MIMXRT1024.s_lpuartHandle (*C var*), 705
drivers_RT_RT1020_MIMXRT1024.s_lpuartIsr (*C var*), 705
drivers_RT_RT1020_MIMXRT1024.s_lpuartTxIRQ (*C var*), 705
drivers_RT_RT1020_MIMXRT1024.sai_bclk_source_t (*C type*), 797
drivers_RT_RT1020_MIMXRT1024.sai_bit_clock_t (*C type*), 798
drivers_RT_RT1020_MIMXRT1024.sai_clock_polarity_t (*C type*), 797
drivers_RT_RT1020_MIMXRT1024.SAI_CLOCKS (*C macro*), 237
drivers_RT_RT1020_MIMXRT1024.sai_config_t (*C type*), 798
drivers_RT_RT1020_MIMXRT1024.sai_data_order_t (*C type*), 797
drivers_RT_RT1020_MIMXRT1024.sai_data_pin_state_t (*C type*), 798
drivers_RT_RT1020_MIMXRT1024.SAI_Deinit (*C function*), 775
drivers_RT_RT1020_MIMXRT1024.SAI_DriverIRQHandler (*C function*), 789
drivers_RT_RT1020_MIMXRT1024.sai_edma_callback_t (*C type*), 810
drivers_RT_RT1020_MIMXRT1024.sai_edma_handle (*C struct*), 810

drivers_RT_RT1020_MIMXRT1024.sai_edma_handle_t (*C type*), 810
drivers_RT_RT1020_MIMXRT1024.sai_edma_handle.bytesPerFrame (*C var*), 811
drivers_RT_RT1020_MIMXRT1024.sai_edma_handle.callback (*C var*), 811
drivers_RT_RT1020_MIMXRT1024.sai_edma_handle.channel (*C var*), 811
drivers_RT_RT1020_MIMXRT1024.sai_edma_handle.channelMask (*C var*), 811
drivers_RT_RT1020_MIMXRT1024.sai_edma_handle.channelNums (*C var*), 811
drivers_RT_RT1020_MIMXRT1024.sai_edma_handle.count (*C var*), 811
drivers_RT_RT1020_MIMXRT1024.sai_edma_handle.dmaHandle (*C var*), 810
drivers_RT_RT1020_MIMXRT1024.sai_edma_handle.interleaveType (*C var*), 811
drivers_RT_RT1020_MIMXRT1024.sai_edma_handle.nbytes (*C var*), 811
drivers_RT_RT1020_MIMXRT1024.sai_edma_handle.queueDriver (*C var*), 811
drivers_RT_RT1020_MIMXRT1024.sai_edma_handle.queueUser (*C var*), 811
drivers_RT_RT1020_MIMXRT1024.sai_edma_handle.saiQueue (*C var*), 811
drivers_RT_RT1020_MIMXRT1024.sai_edma_handle.state (*C var*), 811
drivers_RT_RT1020_MIMXRT1024.sai_edma_handle.tcd (*C var*), 811
drivers_RT_RT1020_MIMXRT1024.sai_edma_handle.transferSize (*C var*), 811
drivers_RT_RT1020_MIMXRT1024.sai_edma_handle.userData (*C var*), 811
drivers_RT_RT1020_MIMXRT1024.sai_edma_interleave_t (*C type*), 810
drivers_RT_RT1020_MIMXRT1024.sai_fifo_combine_t (*C type*), 798
drivers_RT_RT1020_MIMXRT1024.sai_fifo_packing_t (*C type*), 798
drivers_RT_RT1020_MIMXRT1024.sai_fifo_t (*C type*), 798
drivers_RT_RT1020_MIMXRT1024.sai_frame_sync_len_t (*C type*), 798
drivers_RT_RT1020_MIMXRT1024.sai_frame_sync_t (*C type*), 798
drivers_RT_RT1020_MIMXRT1024.SAI_GetClassicI2SConfig (*C function*), 778
drivers_RT_RT1020_MIMXRT1024.SAI_GetDSPConfig (*C function*), 779
drivers_RT_RT1020_MIMXRT1024.SAI_GetLeftJustifiedConfig (*C function*), 779
drivers_RT_RT1020_MIMXRT1024.SAI_GetRightJustifiedConfig (*C function*), 779
drivers_RT_RT1020_MIMXRT1024.SAI_GetTDMConfig (*C function*), 779
drivers_RT_RT1020_MIMXRT1024.sai_handle_t (*C type*), 799
drivers_RT_RT1020_MIMXRT1024.SAI_Init (*C function*), 775
drivers_RT_RT1020_MIMXRT1024.sai_master_clock_t (*C type*), 798
drivers_RT_RT1020_MIMXRT1024.sai_master_slave_t (*C type*), 797
drivers_RT_RT1020_MIMXRT1024.sai_mono_stereo_t (*C type*), 797
drivers_RT_RT1020_MIMXRT1024.sai_protocol_t (*C type*), 797
drivers_RT_RT1020_MIMXRT1024.SAI_ReadBlocking (*C function*), 785
drivers_RT_RT1020_MIMXRT1024.SAI_ReadData (*C function*), 786
drivers_RT_RT1020_MIMXRT1024.SAI_ReadMultiChannelBlocking (*C function*), 786
drivers_RT_RT1020_MIMXRT1024.sai_reset_type_t (*C type*), 797
drivers_RT_RT1020_MIMXRT1024.SAI_RxClearStatusFlags (*C function*), 780
drivers_RT_RT1020_MIMXRT1024.SAI_RxDisableInterrupts (*C function*), 783
drivers_RT_RT1020_MIMXRT1024.SAI_RxEnable (*C function*), 776
drivers_RT_RT1020_MIMXRT1024.SAI_RxEnableDMA (*C function*), 784
drivers_RT_RT1020_MIMXRT1024.SAI_RxEnableInterrupts (*C function*), 783
drivers_RT_RT1020_MIMXRT1024.SAI_RxGetDataRegisterAddress (*C function*), 784
drivers_RT_RT1020_MIMXRT1024.SAI_RxGetStatusFlag (*C function*), 780
drivers_RT_RT1020_MIMXRT1024.SAI_RxReset (*C function*), 776
drivers_RT_RT1020_MIMXRT1024.SAI_RxSetBitclockConfig (*C function*), 777
drivers_RT_RT1020_MIMXRT1024.SAI_RxSetBitClockDirection (*C function*), 776
drivers_RT_RT1020_MIMXRT1024.SAI_RxSetBitClockPolarity (*C function*), 782
drivers_RT_RT1020_MIMXRT1024.SAI_RxSetBitClockRate (*C function*), 777
drivers_RT_RT1020_MIMXRT1024.SAI_RxSetChannelFIFOMask (*C function*), 781
drivers_RT_RT1020_MIMXRT1024.SAI_RxSetConfig (*C function*), 778
drivers_RT_RT1020_MIMXRT1024.SAI_RxSetDataOrder (*C function*), 781
drivers_RT_RT1020_MIMXRT1024.SAI_RxSetFifoConfig (*C function*), 778
drivers_RT_RT1020_MIMXRT1024.SAI_RxSetFIFOErrorContinue (*C function*), 782
drivers_RT_RT1020_MIMXRT1024.SAI_RxSetFIFOPacking (*C function*), 782
drivers_RT_RT1020_MIMXRT1024.SAI_RxSetFrameSyncConfig (*C function*), 778
drivers_RT_RT1020_MIMXRT1024.SAI_RxSetFrameSyncDirection (*C function*), 776

drivers_RT_RT1020_MIMXRT1024.SAI_RxSetFrameSyncPolarity (*C function*), 782
drivers_RT_RT1020_MIMXRT1024.SAI_RxSetSerialDataConfig (*C function*), 778
drivers_RT_RT1020_MIMXRT1024.SAI_RxSoftwareReset (*C function*), 781
drivers_RT_RT1020_MIMXRT1024.sai_sample_rate_t (*C type*), 798
drivers_RT_RT1020_MIMXRT1024.sai_serial_data_t (*C type*), 798
drivers_RT_RT1020_MIMXRT1024.SAI_SetMasterClockConfig (*C function*), 777
drivers_RT_RT1020_MIMXRT1024.sai_sync_mode_t (*C type*), 797
drivers_RT_RT1020_MIMXRT1024.sai_transceiver_t (*C type*), 798
drivers_RT_RT1020_MIMXRT1024.sai_transceiver_type_t (*C type*), 798
drivers_RT_RT1020_MIMXRT1024.sai_transfer_callback_t (*C type*), 799
drivers_RT_RT1020_MIMXRT1024.sai_transfer_format_t (*C type*), 798
drivers_RT_RT1020_MIMXRT1024.sai_transfer_t (*C type*), 798
drivers_RT_RT1020_MIMXRT1024.SAI_TransferAbortReceive (*C function*), 789
drivers_RT_RT1020_MIMXRT1024.SAI_TransferAbortReceiveEDMA (*C function*), 809
drivers_RT_RT1020_MIMXRT1024.SAI_TransferAbortSend (*C function*), 788
drivers_RT_RT1020_MIMXRT1024.SAI_TransferAbortSendEDMA (*C function*), 809
drivers_RT_RT1020_MIMXRT1024.SAI_TransferGetReceiveCount (*C function*), 788
drivers_RT_RT1020_MIMXRT1024.SAI_TransferGetReceiveCountEDMA (*C function*), 809
drivers_RT_RT1020_MIMXRT1024.SAI_TransferGetSendCount (*C function*), 788
drivers_RT_RT1020_MIMXRT1024.SAI_TransferGetSendCountEDMA (*C function*), 809
drivers_RT_RT1020_MIMXRT1024.SAI_TransferGetValidTransferSlotsEDMA (*C function*), 810
drivers_RT_RT1020_MIMXRT1024.SAI_TransferReceiveEDMA (*C function*), 807
drivers_RT_RT1020_MIMXRT1024.SAI_TransferReceiveLoopEDMA (*C function*), 808
drivers_RT_RT1020_MIMXRT1024.SAI_TransferReceiveNonBlocking (*C function*), 787
drivers_RT_RT1020_MIMXRT1024.SAI_TransferRxCreateHandle (*C function*), 786
drivers_RT_RT1020_MIMXRT1024.SAI_TransferRxCreateHandleEDMA (*C function*), 805
drivers_RT_RT1020_MIMXRT1024.SAI_TransferRxHandleIRQ (*C function*), 789
drivers_RT_RT1020_MIMXRT1024.SAI_TransferRxSetConfig (*C function*), 787
drivers_RT_RT1020_MIMXRT1024.SAI_TransferRxSetConfigEDMA (*C function*), 806
drivers_RT_RT1020_MIMXRT1024.SAI_TransferSendEDMA (*C function*), 806
drivers_RT_RT1020_MIMXRT1024.SAI_TransferSendLoopEDMA (*C function*), 807
drivers_RT_RT1020_MIMXRT1024.SAI_TransferSendNonBlocking (*C function*), 787
drivers_RT_RT1020_MIMXRT1024.SAI_TransferSetInterleaveType (*C function*), 805
drivers_RT_RT1020_MIMXRT1024.SAI_TransferTerminateReceive (*C function*), 789
drivers_RT_RT1020_MIMXRT1024.SAI_TransferTerminateReceiveEDMA (*C function*), 808
drivers_RT_RT1020_MIMXRT1024.SAI_TransferTerminateSend (*C function*), 789
drivers_RT_RT1020_MIMXRT1024.SAI_TransferTerminateSendEDMA (*C function*), 808
drivers_RT_RT1020_MIMXRT1024.SAI_TransferTxCreateHandle (*C function*), 786
drivers_RT_RT1020_MIMXRT1024.SAI_TransferTxCreateHandleEDMA (*C function*), 804
drivers_RT_RT1020_MIMXRT1024.SAI_TransferTxHandleIRQ (*C function*), 789
drivers_RT_RT1020_MIMXRT1024.SAI_TransferTxSetConfig (*C function*), 787
drivers_RT_RT1020_MIMXRT1024.SAI_TransferTxSetConfigEDMA (*C function*), 805
drivers_RT_RT1020_MIMXRT1024.SAI_TxClearStatusFlags (*C function*), 780
drivers_RT_RT1020_MIMXRT1024.SAI_TxDisableInterrupts (*C function*), 783
drivers_RT_RT1020_MIMXRT1024.SAI_TxEnable (*C function*), 776
drivers_RT_RT1020_MIMXRT1024.SAI_TxEnableDMA (*C function*), 784
drivers_RT_RT1020_MIMXRT1024.SAI_TxEnableInterrupts (*C function*), 783
drivers_RT_RT1020_MIMXRT1024.SAI_TxGetDataRegisterAddress (*C function*), 784
drivers_RT_RT1020_MIMXRT1024.SAI_TxGetStatusFlag (*C function*), 780
drivers_RT_RT1020_MIMXRT1024.SAI_TxReset (*C function*), 775
drivers_RT_RT1020_MIMXRT1024.SAI_TxSetBitclockConfig (*C function*), 777
drivers_RT_RT1020_MIMXRT1024.SAI_TxSetBitClockDirection (*C function*), 776
drivers_RT_RT1020_MIMXRT1024.SAI_TxSetBitClockPolarity (*C function*), 782
drivers_RT_RT1020_MIMXRT1024.SAI_TxSetBitClockRate (*C function*), 777
drivers_RT_RT1020_MIMXRT1024.SAI_TxSetChannelFIFOMask (*C function*), 781
drivers_RT_RT1020_MIMXRT1024.SAI_TxSetConfig (*C function*), 778
drivers_RT_RT1020_MIMXRT1024.SAI_TxSetDataOrder (*C function*), 781
drivers_RT_RT1020_MIMXRT1024.SAI_TxSetFifoConfig (*C function*), 777

drivers_RT_RT1020_MIMXRT1024.SAI_TxSetFIFOErrorContinue (*C function*), 782
drivers_RT_RT1020_MIMXRT1024.SAI_TxSetFIFOPacking (*C function*), 782
drivers_RT_RT1020_MIMXRT1024.SAI_TxSetFrameSyncConfig (*C function*), 778
drivers_RT_RT1020_MIMXRT1024.SAI_TxSetFrameSyncDirection (*C function*), 776
drivers_RT_RT1020_MIMXRT1024.SAI_TxSetFrameSyncPolarity (*C function*), 782
drivers_RT_RT1020_MIMXRT1024.SAI_TxSetSerialDataConfig (*C function*), 778
drivers_RT_RT1020_MIMXRT1024.SAI_TxSoftwareReset (*C function*), 781
drivers_RT_RT1020_MIMXRT1024.sai_word_width_t (*C type*), 798
drivers_RT_RT1020_MIMXRT1024.SAI_WriteBlocking (*C function*), 785
drivers_RT_RT1020_MIMXRT1024.SAI_WriteData (*C function*), 785
drivers_RT_RT1020_MIMXRT1024.SAI_WriteMultiChannelBlocking (*C function*), 785
drivers_RT_RT1020_MIMXRT1024.SAI_XFER_QUEUE_SIZE (*C macro*), 799
drivers_RT_RT1020_MIMXRT1024.SDK_ALIGN (*C macro*), 611
drivers_RT_RT1020_MIMXRT1024.SDK_ATOMIC_LOCAL_ADD (*C macro*), 610
drivers_RT_RT1020_MIMXRT1024.SDK_ATOMIC_LOCAL_CLEAR (*C macro*), 610
drivers_RT_RT1020_MIMXRT1024.SDK_ATOMIC_LOCAL_CLEAR_AND_SET (*C macro*), 610
drivers_RT_RT1020_MIMXRT1024.SDK_ATOMIC_LOCAL_COMPARE_AND_SET (*C macro*), 610
drivers_RT_RT1020_MIMXRT1024.SDK_ATOMIC_LOCAL_SET (*C macro*), 610
drivers_RT_RT1020_MIMXRT1024.SDK_ATOMIC_LOCAL_SUB (*C macro*), 610
drivers_RT_RT1020_MIMXRT1024.SDK_ATOMIC_LOCAL_TEST_AND_SET (*C macro*), 611
drivers_RT_RT1020_MIMXRT1024.SDK_ATOMIC_LOCAL_TOGGLE (*C macro*), 610
drivers_RT_RT1020_MIMXRT1024.SDK_DelayAtLeastUs (*C function*), 622
drivers_RT_RT1020_MIMXRT1024.SDK_DEVICE_MAXIMUM_CPU_CLOCK_FREQUENCY (*C macro*), 233
drivers_RT_RT1020_MIMXRT1024.SDK_Free (*C function*), 622
drivers_RT_RT1020_MIMXRT1024.SDK_IS_ALIGNED (*C macro*), 611
drivers_RT_RT1020_MIMXRT1024.SDK_ISR_EXIT_BARRIER (*C macro*), 611
drivers_RT_RT1020_MIMXRT1024.SDK_L1DCACHE_ALIGN (*C macro*), 611
drivers_RT_RT1020_MIMXRT1024.SDK_Malloc (*C function*), 621
drivers_RT_RT1020_MIMXRT1024.SDK_SIZEALIGN (*C macro*), 611
drivers_RT_RT1020_MIMXRT1024.SDK_SIZEALIGN_DOWN (*C macro*), 611
drivers_RT_RT1020_MIMXRT1024.SDK_SIZEALIGN_UP (*C macro*), 611
drivers_RT_RT1020_MIMXRT1024.sem_dbi_burst_len_t (*C type*), 829
drivers_RT_RT1020_MIMXRT1024.sem_nand_burst_len_t (*C type*), 829
drivers_RT_RT1020_MIMXRT1024.sem_norsram_burst_len_t (*C type*), 829
drivers_RT_RT1020_MIMXRT1024.sem_sdram_burst_len_t (*C type*), 829
drivers_RT_RT1020_MIMXRT1024.semc_addr_mode_t (*C type*), 830
drivers_RT_RT1020_MIMXRT1024.semc_adv_level_control_t (*C type*), 830
drivers_RT_RT1020_MIMXRT1024.semc_adv_polarity_t (*C type*), 830
drivers_RT_RT1020_MIMXRT1024.semc_axi_queueweight_t (*C type*), 831
drivers_RT_RT1020_MIMXRT1024.SEMC_BuildNandIPCommand (*C function*), 815
drivers_RT_RT1020_MIMXRT1024.semc_caslatency_t (*C type*), 829
drivers_RT_RT1020_MIMXRT1024.SEMC_ClearStatusFlags (*C function*), 814
drivers_RT_RT1020_MIMXRT1024.SEMC_CLOCKS (*C macro*), 237
drivers_RT_RT1020_MIMXRT1024.semc_config_t (*C type*), 831
drivers_RT_RT1020_MIMXRT1024.SEMC_ConfigureDBI (*C function*), 813
drivers_RT_RT1020_MIMXRT1024.SEMC_ConfigureNAND (*C function*), 812
drivers_RT_RT1020_MIMXRT1024.SEMC_ConfigureNOR (*C function*), 812
drivers_RT_RT1020_MIMXRT1024.SEMC_ConfigureSDRAM (*C function*), 812
drivers_RT_RT1020_MIMXRT1024.SEMC_ConfigureSRAM (*C function*), 813
drivers_RT_RT1020_MIMXRT1024.SEMC_ConfigureSRAMWithChipSelection (*C function*), 813
drivers_RT_RT1020_MIMXRT1024.semc_dbi_column_bit_num_t (*C type*), 829
drivers_RT_RT1020_MIMXRT1024.semc_dbi_config_t (*C type*), 831
drivers_RT_RT1020_MIMXRT1024.SEMC_Deinit (*C function*), 812
drivers_RT_RT1020_MIMXRT1024.SEMC_DisableInterrupts (*C function*), 814
drivers_RT_RT1020_MIMXRT1024.semc_dqs_mode_t (*C type*), 830
drivers_RT_RT1020_MIMXRT1024.SEMC_EnableInterrupts (*C function*), 813

drivers_RT_RT1020_MIMXRT1024.SEMC_EXSC_CLOCKS (*C macro*), 237
drivers_RT_RT1020_MIMXRT1024.SEMC_GetDefaultConfig (*C function*), 812
drivers_RT_RT1020_MIMXRT1024.SEMC_GetStatusFlag (*C function*), 814
drivers_RT_RT1020_MIMXRT1024.SEMC_Init (*C function*), 812
drivers_RT_RT1020_MIMXRT1024.semc_interrupt_enable_t (*C type*), 829
drivers_RT_RT1020_MIMXRT1024.semc_iomux_nora27_pin (*C type*), 830
drivers_RT_RT1020_MIMXRT1024.semc_iomux_pin (*C type*), 830
drivers_RT_RT1020_MIMXRT1024.semc_ipcmd_datasize_t (*C type*), 829
drivers_RT_RT1020_MIMXRT1024.semc_ipcmd_nand_addrmode_t (*C type*), 830
drivers_RT_RT1020_MIMXRT1024.semc_ipcmd_nand_cmdmode_t (*C type*), 830
drivers_RT_RT1020_MIMXRT1024.semc_ipcmd_nor_dbi_t (*C type*), 830
drivers_RT_RT1020_MIMXRT1024.semc_ipcmd_sdram_t (*C type*), 830
drivers_RT_RT1020_MIMXRT1024.semc_ipcmd_sram_t (*C type*), 830
drivers_RT_RT1020_MIMXRT1024.SEMC_IPCommandNandRead (*C function*), 815
drivers_RT_RT1020_MIMXRT1024.SEMC_IPCommandNandWrite (*C function*), 815
drivers_RT_RT1020_MIMXRT1024.SEMC_IPCommandNorRead (*C function*), 816
drivers_RT_RT1020_MIMXRT1024.SEMC_IPCommandNorWrite (*C function*), 815
drivers_RT_RT1020_MIMXRT1024.SEMC_IsInIdle (*C function*), 814
drivers_RT_RT1020_MIMXRT1024.SEMC_IsNandReady (*C function*), 815
drivers_RT_RT1020_MIMXRT1024.semc_mem_type_t (*C type*), 828
drivers_RT_RT1020_MIMXRT1024.semc_nand_access_type_t (*C type*), 829
drivers_RT_RT1020_MIMXRT1024.semc_nand_address_option_t (*C type*), 830
drivers_RT_RT1020_MIMXRT1024.semc_nand_column_bit_num_t (*C type*), 829
drivers_RT_RT1020_MIMXRT1024.semc_nand_config_t (*C type*), 831
drivers_RT_RT1020_MIMXRT1024.semc_nand_timing_config_t (*C type*), 831
drivers_RT_RT1020_MIMXRT1024.semc_nor_config_t (*C type*), 831
drivers_RT_RT1020_MIMXRT1024.semc_norsram_column_bit_num_t (*C type*), 829
drivers_RT_RT1020_MIMXRT1024.semc_queuea_weight_struct_t (*C type*), 831
drivers_RT_RT1020_MIMXRT1024.semc_queuea_weight_t (*C type*), 831
drivers_RT_RT1020_MIMXRT1024.semc_queueb_weight_struct_t (*C type*), 831
drivers_RT_RT1020_MIMXRT1024.semc_queueb_weight_t (*C type*), 831
drivers_RT_RT1020_MIMXRT1024.semc_rdy_polarity_t (*C type*), 830
drivers_RT_RT1020_MIMXRT1024.semc_refresh_time_t (*C type*), 829
drivers_RT_RT1020_MIMXRT1024.semc_sdram_column_bit_num_t (*C type*), 829
drivers_RT_RT1020_MIMXRT1024.semc_sdram_config_t (*C type*), 830
drivers_RT_RT1020_MIMXRT1024.semc_sdram_cs_t (*C type*), 829
drivers_RT_RT1020_MIMXRT1024.SEMC_SendIPCommand (*C function*), 814
drivers_RT_RT1020_MIMXRT1024.semc_sram_config_t (*C type*), 831
drivers_RT_RT1020_MIMXRT1024.semc_sram_cs_t (*C type*), 829
drivers_RT_RT1020_MIMXRT1024.semc_sync_mode_t (*C type*), 830
drivers_RT_RT1020_MIMXRT1024.semc_waitready_polarity_t (*C type*), 829
drivers_RT_RT1020_MIMXRT1024.smec_port_size_t (*C type*), 830
drivers_RT_RT1020_MIMXRT1024.SNVS_HP_ChangeSSMState (*C function*), 848
drivers_RT_RT1020_MIMXRT1024.SNVS_HP_ClearHighAssuranceCounter (*C function*), 845
drivers_RT_RT1020_MIMXRT1024.SNVS_HP_ClearSecurityViolationStatusFlags (*C function*), 850
drivers_RT_RT1020_MIMXRT1024.SNVS_HP_ClearStatusFlags (*C function*), 849
drivers_RT_RT1020_MIMXRT1024.SNVS_HP_Deinit (*C function*), 842
drivers_RT_RT1020_MIMXRT1024.SNVS_HP_EnableHighAssuranceCounter (*C function*), 845
drivers_RT_RT1020_MIMXRT1024.SNVS_HP_EnableMasterKeySelection (*C function*), 848
drivers_RT_RT1020_MIMXRT1024.SNVS_HP_GetHighAssuranceCounter (*C function*), 845
drivers_RT_RT1020_MIMXRT1024.SNVS_HP_GetSecurityViolationStatusFlags (*C function*), 850
drivers_RT_RT1020_MIMXRT1024.SNVS_HP_GetSSMState (*C function*), 849
drivers_RT_RT1020_MIMXRT1024.SNVS_HP_GetStatusFlags (*C function*), 849
drivers_RT_RT1020_MIMXRT1024.SNVS_HP_Init (*C function*), 842
drivers_RT_RT1020_MIMXRT1024.snvs_hp_interrupts_t (*C type*), 848
drivers_RT_RT1020_MIMXRT1024.SNVS_HP_LoadHighAssuranceCounter (*C function*), 845
drivers_RT_RT1020_MIMXRT1024.SNVS_HP_LockHighAssuranceCounter (*C function*), 846
drivers_RT_RT1020_MIMXRT1024.SNVS_HP_ProgramZeroizableMasterKey (*C function*), 848

drivers_RT_RT1020_MIMXRT1024.SNVS_HP_ResetLP (*C function*), 849
drivers_RT_RT1020_MIMXRT1024.SNVS_HP_RTC_ClearStatusFlags (*C function*), 844
drivers_RT_RT1020_MIMXRT1024.snvs_hp_rtc_config_t (*C type*), 848
drivers_RT_RT1020_MIMXRT1024.snvs_hp_rtc_datetime_t (*C type*), 848
drivers_RT_RT1020_MIMXRT1024.SNVS_HP_RTC_Deinit (*C function*), 843
drivers_RT_RT1020_MIMXRT1024.SNVS_HP_RTC_DisableInterrupts (*C function*), 844
drivers_RT_RT1020_MIMXRT1024.SNVS_HP_RTC_EnableInterrupts (*C function*), 844
drivers_RT_RT1020_MIMXRT1024.SNVS_HP_RTC_GetAlarm (*C function*), 843
drivers_RT_RT1020_MIMXRT1024.SNVS_HP_RTC_GetDatetime (*C function*), 843
drivers_RT_RT1020_MIMXRT1024.SNVS_HP_RTC_GetDefaultConfig (*C function*), 843
drivers_RT_RT1020_MIMXRT1024.SNVS_HP_RTC_GetEnabledInterrupts (*C function*), 844
drivers_RT_RT1020_MIMXRT1024.SNVS_HP_RTC_GetStatusFlags (*C function*), 844
drivers_RT_RT1020_MIMXRT1024.SNVS_HP_RTC_Init (*C function*), 842
drivers_RT_RT1020_MIMXRT1024.SNVS_HP_RTC_SetAlarm (*C function*), 843
drivers_RT_RT1020_MIMXRT1024.SNVS_HP_RTC_SetDatetime (*C function*), 843
drivers_RT_RT1020_MIMXRT1024.SNVS_HP_RTC_StartTimer (*C function*), 844
drivers_RT_RT1020_MIMXRT1024.SNVS_HP_RTC_StopTimer (*C function*), 845
drivers_RT_RT1020_MIMXRT1024.SNVS_HP_RTC_TimeSynchronize (*C function*), 844
drivers_RT_RT1020_MIMXRT1024.SNVS_HP_SetHighAssuranceCounterInitialValue (*C function*), 845
drivers_RT_RT1020_MIMXRT1024.SNVS_HP_SetSoftwareFatalSecurityViolation (*C function*), 849
drivers_RT_RT1020_MIMXRT1024.SNVS_HP_SetSoftwareSecurityViolation (*C function*), 849
drivers_RT_RT1020_MIMXRT1024.snvs_hp_ssm_state_t (*C type*), 848
drivers_RT_RT1020_MIMXRT1024.SNVS_HP_StartHighAssuranceCounter (*C function*), 845
drivers_RT_RT1020_MIMXRT1024.snvs_hp_status_flags_t (*C type*), 848
drivers_RT_RT1020_MIMXRT1024.snvs_hp_sv_status_flags_t (*C type*), 848
drivers_RT_RT1020_MIMXRT1024.SNVS_HPSVSR_SV0_MASK (*C macro*), 850
drivers_RT_RT1020_MIMXRT1024.SNVS_HPSVSR_SV1_MASK (*C macro*), 850
drivers_RT_RT1020_MIMXRT1024.SNVS_HPSVSR_SV2_MASK (*C macro*), 850
drivers_RT_RT1020_MIMXRT1024.SNVS_HPSVSR_SV4_MASK (*C macro*), 850
drivers_RT_RT1020_MIMXRT1024.SNVS_HPSVSR_SV5_MASK (*C macro*), 850
drivers_RT_RT1020_MIMXRT1024.snvs_lp_active_clock_t (*C type*), 859
drivers_RT_RT1020_MIMXRT1024.snvs_lp_active_tx_tamper_t (*C type*), 859
drivers_RT_RT1020_MIMXRT1024.SNVS_LP_CheckClockTamper (*C function*), 855
drivers_RT_RT1020_MIMXRT1024.SNVS_LP_CheckTemperatureTamper (*C function*), 855
drivers_RT_RT1020_MIMXRT1024.SNVS_LP_CheckVoltageTamper (*C function*), 854
drivers_RT_RT1020_MIMXRT1024.SNVS_LP_ClearAllExternalTamperStatus (*C function*), 856
drivers_RT_RT1020_MIMXRT1024.SNVS_LP_ClearExternalTamperStatus (*C function*), 856
drivers_RT_RT1020_MIMXRT1024.SNVS_LP_Deinit (*C function*), 852
drivers_RT_RT1020_MIMXRT1024.SNVS_LP_DisableAllExternalTamper (*C function*), 855
drivers_RT_RT1020_MIMXRT1024.SNVS_LP_DisableExternalTamper (*C function*), 855
drivers_RT_RT1020_MIMXRT1024.SNVS_LP_EnableMonotonicCounter (*C function*), 856
drivers_RT_RT1020_MIMXRT1024.SNVS_LP_EnablePassiveTamper (*C function*), 854
drivers_RT_RT1020_MIMXRT1024.SNVS_LP_EnableRxActiveTamper (*C function*), 854
drivers_RT_RT1020_MIMXRT1024.SNVS_LP_EnableTxActiveTamper (*C function*), 854
drivers_RT_RT1020_MIMXRT1024.SNVS_LP_EnableZeroizableMasterKeyECC (*C function*), 857
drivers_RT_RT1020_MIMXRT1024.snvs_lp_external_tamper_polarity_t (*C type*), 859
drivers_RT_RT1020_MIMXRT1024.snvs_lp_external_tamper_status_t (*C type*), 859
drivers_RT_RT1020_MIMXRT1024.snvs_lp_external_tamper_t (*C type*), 859
drivers_RT_RT1020_MIMXRT1024.SNVS_LP_GetExternalTamperStatus (*C function*), 855
drivers_RT_RT1020_MIMXRT1024.SNVS_LP_GetMonotonicCounter (*C function*), 856
drivers_RT_RT1020_MIMXRT1024.SNVS_LP_GetZeroizableMasterKeyValid (*C function*), 856
drivers_RT_RT1020_MIMXRT1024.SNVS_LP_IncreaseMonotonicCounter (*C function*), 856
drivers_RT_RT1020_MIMXRT1024.SNVS_LP_Init (*C function*), 851
drivers_RT_RT1020_MIMXRT1024.snvs_lp_master_key_mode_t (*C type*), 860
drivers_RT_RT1020_MIMXRT1024.SNVS_LP_MAX_TAMPER (*C macro*), 860
drivers_RT_RT1020_MIMXRT1024.snvs_lp_passive_tamper_t (*C struct*), 861

drivers_RT_RT1020_MIMXRT1024.SNVS_LP_PassiveTamperPin_GetDefaultConfig (*C function*), 855
drivers_RT_RT1020_MIMXRT1024.SNVS_LP_SetClockTamper (*C function*), 854
drivers_RT_RT1020_MIMXRT1024.SNVS_LP_SetMasterKeyMode (*C function*), 857
drivers_RT_RT1020_MIMXRT1024.SNVS_LP_SetTemperatureTamper (*C function*), 854
drivers_RT_RT1020_MIMXRT1024.SNVS_LP_SetVoltageTamper (*C function*), 854
drivers_RT_RT1020_MIMXRT1024.SNVS_LP_SetZeroizableMasterKeyProgramMode (*C function*), 857
drivers_RT_RT1020_MIMXRT1024.SNVS_LP_SetZeroizableMasterKeyValid (*C function*), 856
drivers_RT_RT1020_MIMXRT1024.SNVS_LP_SRTC_ClearStatusFlags (*C function*), 853
drivers_RT_RT1020_MIMXRT1024.snvs_lp_srtc_config_t (*C type*), 860
drivers_RT_RT1020_MIMXRT1024.snvs_lp_srtc_datetime_t (*C type*), 859
drivers_RT_RT1020_MIMXRT1024.SNVS_LP_SRTC_Deinit (*C function*), 860
drivers_RT_RT1020_MIMXRT1024.SNVS_LP_SRTC_DisableInterrupts (*C function*), 853
drivers_RT_RT1020_MIMXRT1024.SNVS_LP_SRTC_EnableInterrupts (*C function*), 853
drivers_RT_RT1020_MIMXRT1024.SNVS_LP_SRTC_GetAlarm (*C function*), 852
drivers_RT_RT1020_MIMXRT1024.SNVS_LP_SRTC_GetDatetime (*C function*), 852
drivers_RT_RT1020_MIMXRT1024.SNVS_LP_SRTC_GetDefaultConfig (*C function*), 860
drivers_RT_RT1020_MIMXRT1024.SNVS_LP_SRTC_GetEnabledInterrupts (*C function*), 853
drivers_RT_RT1020_MIMXRT1024.SNVS_LP_SRTC_GetStatusFlags (*C function*), 853
drivers_RT_RT1020_MIMXRT1024.SNVS_LP_SRTC_Init (*C function*), 860
drivers_RT_RT1020_MIMXRT1024.snvs_lp_srtc_interrupts_t (*C type*), 859
drivers_RT_RT1020_MIMXRT1024.SNVS_LP_SRTC_SetAlarm (*C function*), 852
drivers_RT_RT1020_MIMXRT1024.SNVS_LP_SRTC_SetDatetime (*C function*), 852
drivers_RT_RT1020_MIMXRT1024.SNVS_LP_SRTC_StartTimer (*C function*), 853
drivers_RT_RT1020_MIMXRT1024.snvs_lp_srtc_status_flags_t (*C type*), 859
drivers_RT_RT1020_MIMXRT1024.SNVS_LP_SRTC_StopTimer (*C function*), 853
drivers_RT_RT1020_MIMXRT1024.SNVS_LP_TamperPinRx_GetDefaultConfig (*C function*), 855
drivers_RT_RT1020_MIMXRT1024.SNVS_LP_TamperPinTx_GetDefaultConfig (*C function*), 855
drivers_RT_RT1020_MIMXRT1024.SNVS_LP_WriteZeroizableMasterKey (*C function*), 856
drivers_RT_RT1020_MIMXRT1024.snvs_lp_zmk_program_mode_t (*C type*), 860
drivers_RT_RT1020_MIMXRT1024.SNVS_MAKE_HP_SV_FLAG (*C macro*), 850
drivers_RT_RT1020_MIMXRT1024.SNVS_ZMK_REG_COUNT (*C macro*), 860
drivers_RT_RT1020_MIMXRT1024.SPDIF_ClearStatusFlags (*C function*), 863
drivers_RT_RT1020_MIMXRT1024.SPDIF_CLOCKS (*C macro*), 237
drivers_RT_RT1020_MIMXRT1024.spdif_config_t (*C type*), 875
drivers_RT_RT1020_MIMXRT1024.SPDIF_CopyRxChannelStatusToTx (*C function*), 867
drivers_RT_RT1020_MIMXRT1024.SPDIF_Deinit (*C function*), 862
drivers_RT_RT1020_MIMXRT1024.SPDIF_DisableInterrupts (*C function*), 864
drivers_RT_RT1020_MIMXRT1024.spdif_edma_callback_t (*C type*), 879
drivers_RT_RT1020_MIMXRT1024.spdif_edma_handle_t (*C type*), 879
drivers_RT_RT1020_MIMXRT1024.spdif_edma_transfer_t (*C type*), 879
drivers_RT_RT1020_MIMXRT1024.SPDIF_EnableDMA (*C function*), 864
drivers_RT_RT1020_MIMXRT1024.SPDIF_EnableInterrupts (*C function*), 863
drivers_RT_RT1020_MIMXRT1024.spdif_gain_select_t (*C type*), 874
drivers_RT_RT1020_MIMXRT1024.SPDIF_GetDefaultConfig (*C function*), 862
drivers_RT_RT1020_MIMXRT1024.SPDIF_GetInstance (*C function*), 862
drivers_RT_RT1020_MIMXRT1024.SPDIF_GetRxSampleRate (*C function*), 865
drivers_RT_RT1020_MIMXRT1024.SPDIF_GetStatusFlag (*C function*), 863
drivers_RT_RT1020_MIMXRT1024.spdif_handle_t (*C type*), 875
drivers_RT_RT1020_MIMXRT1024.SPDIF_Init (*C function*), 862
drivers_RT_RT1020_MIMXRT1024.SPDIF_ReadBlocking (*C function*), 866
drivers_RT_RT1020_MIMXRT1024.SPDIF_ReadChannelStatusHigh (*C function*), 866
drivers_RT_RT1020_MIMXRT1024.SPDIF_ReadChannelStatusLow (*C function*), 867
drivers_RT_RT1020_MIMXRT1024.SPDIF_ReadLeftData (*C function*), 866
drivers_RT_RT1020_MIMXRT1024.SPDIF_ReadQChannel (*C function*), 867
drivers_RT_RT1020_MIMXRT1024.SPDIF_ReadRightData (*C function*), 866
drivers_RT_RT1020_MIMXRT1024.SPDIF_ReadUChannel (*C function*), 867

drivers_RT_RT1020_MIMXRT1024.SPDIF_RxEnable (*C function*), 863
drivers_RT_RT1020_MIMXRT1024.SPDIF_RxFIFOReset (*C function*), 862
drivers_RT_RT1020_MIMXRT1024.spdif_rxfull_select_t (*C type*), 874
drivers_RT_RT1020_MIMXRT1024.SPDIF_RxGetLeftDataRegisterAddress (*C function*), 864
drivers_RT_RT1020_MIMXRT1024.SPDIF_RxGetRightDataRegisterAddress (*C function*), 865
drivers_RT_RT1020_MIMXRT1024.spdif_transfer_callback_t (*C type*), 875
drivers_RT_RT1020_MIMXRT1024.spdif_transfer_t (*C type*), 875
drivers_RT_RT1020_MIMXRT1024.SPDIF_TransferAbortReceive (*C function*), 869
drivers_RT_RT1020_MIMXRT1024.SPDIF_TransferAbortReceiveEDMA (*C function*), 878
drivers_RT_RT1020_MIMXRT1024.SPDIF_TransferAbortSend (*C function*), 869
drivers_RT_RT1020_MIMXRT1024.SPDIF_TransferAbortSendEDMA (*C function*), 878
drivers_RT_RT1020_MIMXRT1024.SPDIF_TransferGetReceiveCount (*C function*), 869
drivers_RT_RT1020_MIMXRT1024.SPDIF_TransferGetReceiveCountEDMA (*C function*), 879
drivers_RT_RT1020_MIMXRT1024.SPDIF_TransferGetSendCount (*C function*), 868
drivers_RT_RT1020_MIMXRT1024.SPDIF_TransferGetSendCountEDMA (*C function*), 879
drivers_RT_RT1020_MIMXRT1024.SPDIF_TransferReceiveEDMA (*C function*), 878
drivers_RT_RT1020_MIMXRT1024.SPDIF_TransferReceiveNonBlocking (*C function*), 868
drivers_RT_RT1020_MIMXRT1024.SPDIF_TransferRxCreateHandle (*C function*), 867
drivers_RT_RT1020_MIMXRT1024.SPDIF_TransferRxCreateHandleEDMA (*C function*), 877
drivers_RT_RT1020_MIMXRT1024.SPDIF_TransferRxHandleIRQ (*C function*), 869
drivers_RT_RT1020_MIMXRT1024.SPDIF_TransferSendEDMA (*C function*), 877
drivers_RT_RT1020_MIMXRT1024.SPDIF_TransferSendNonBlocking (*C function*), 868
drivers_RT_RT1020_MIMXRT1024.SPDIF_TransferTxCreateHandle (*C function*), 867
drivers_RT_RT1020_MIMXRT1024.SPDIF_TransferTxCreateHandleEDMA (*C function*), 877
drivers_RT_RT1020_MIMXRT1024.SPDIF_TransferTxHandleIRQ (*C function*), 869
drivers_RT_RT1020_MIMXRT1024.spdif_tx_source_t (*C type*), 874
drivers_RT_RT1020_MIMXRT1024.spdif_txempty_select_t (*C type*), 874
drivers_RT_RT1020_MIMXRT1024.SPDIF_TxEnable (*C function*), 863
drivers_RT_RT1020_MIMXRT1024.SPDIF_TxFIFOReset (*C function*), 862
drivers_RT_RT1020_MIMXRT1024.SPDIF_TxGetLeftDataRegisterAddress (*C function*), 864
drivers_RT_RT1020_MIMXRT1024.SPDIF_TxGetRightDataRegisterAddress (*C function*), 864
drivers_RT_RT1020_MIMXRT1024.SPDIF_TxSetSampleRate (*C function*), 865
drivers_RT_RT1020_MIMXRT1024.spdif_uchannel_source_t (*C type*), 874
drivers_RT_RT1020_MIMXRT1024.spdif_validity_config_t (*C type*), 874
drivers_RT_RT1020_MIMXRT1024.SPDIF_WriteBlocking (*C function*), 865
drivers_RT_RT1020_MIMXRT1024.SPDIF_WriteChannelStatusHigh (*C function*), 866
drivers_RT_RT1020_MIMXRT1024.SPDIF_WriteChannelStatusLow (*C function*), 866
drivers_RT_RT1020_MIMXRT1024.SPDIF_WriteLeftData (*C function*), 865
drivers_RT_RT1020_MIMXRT1024.SPDIF_WriteRightData (*C function*), 865
drivers_RT_RT1020_MIMXRT1024.SPDIF_XFER_QUEUE_SIZE (*C macro*), 875
drivers_RT_RT1020_MIMXRT1024.SPI_RETRY_TIMES (*C macro*), 456, 675
drivers_RT_RT1020_MIMXRT1024.SRC_AssertEIMReset (*C function*), 885
drivers_RT_RT1020_MIMXRT1024.SRC_ClearResetStatusFlags (*C function*), 886
drivers_RT_RT1020_MIMXRT1024.SRC_DoSoftwareResetARMCore (*C function*), 884
drivers_RT_RT1020_MIMXRT1024.SRC_DoSoftwareResetARMCore0 (*C function*), 884
drivers_RT_RT1020_MIMXRT1024.SRC_DoSoftwareResetARMCoreDebug (*C function*), 884
drivers_RT_RT1020_MIMXRT1024.SRC_EnableCoreDebugResetAfterPowerGate (*C function*), 884
drivers_RT_RT1020_MIMXRT1024.SRC_EnableCoreLockupReset (*C function*), 885
drivers_RT_RT1020_MIMXRT1024.SRC_EnableLockupReset (*C function*), 885
drivers_RT_RT1020_MIMXRT1024.SRC_EnableTemperatureSensorReset (*C function*), 884
drivers_RT_RT1020_MIMXRT1024.SRC_EnableWDOG3Reset (*C function*), 883
drivers_RT_RT1020_MIMXRT1024.SRC_EnableWDOGReset (*C function*), 885
drivers_RT_RT1020_MIMXRT1024.SRC_GetBootModeWord1 (*C function*), 885
drivers_RT_RT1020_MIMXRT1024.SRC_GetBootModeWord2 (*C function*), 886
drivers_RT_RT1020_MIMXRT1024.SRC_GetGeneralPurposeRegister (*C function*), 886
drivers_RT_RT1020_MIMXRT1024.SRC_GetResetStatusFlags (*C function*), 886
drivers_RT_RT1020_MIMXRT1024.SRC_GetSoftwareResetARMCore0Done (*C function*), 884
drivers_RT_RT1020_MIMXRT1024.SRC_GetSoftwareResetARMCoreDebugDone (*C function*), 884

drivers_RT_RT1020_MIMXRT1024.SRC_GetSoftwareResetARMCoreDone (*C function*), 885
drivers_RT_RT1020_MIMXRT1024.SRC_GetStatusFlags (*C function*), 885
drivers_RT_RT1020_MIMXRT1024.src_mix_reset_stretch_cycles_t (*C type*), 883
drivers_RT_RT1020_MIMXRT1024.SRC_SetGeneralPurposeRegister (*C function*), 886
drivers_RT_RT1020_MIMXRT1024.SRC_SetMixResetStretchCycles (*C function*), 883
drivers_RT_RT1020_MIMXRT1024.SRC_SetWdog3ResetOption (*C function*), 884
drivers_RT_RT1020_MIMXRT1024.src_warm_reset_bypass_count_t (*C type*), 883
drivers_RT_RT1020_MIMXRT1024.src_wdog3_reset_option_t (*C type*), 883
drivers_RT_RT1020_MIMXRT1024.status_t (*C type*), 621
drivers_RT_RT1020_MIMXRT1024.SUPPRESS_FALL_THROUGH_WARNING (*C macro*), 624
drivers_RT_RT1020_MIMXRT1024.tamper_active_rx_config_t (*C struct*), 860
drivers_RT_RT1020_MIMXRT1024.tamper_active_tx_config_t (*C struct*), 860
drivers_RT_RT1020_MIMXRT1024.tempmon_alarm_mode (*C type*), 887
drivers_RT_RT1020_MIMXRT1024.tempmon_config_t (*C type*), 887
drivers_RT_RT1020_MIMXRT1024.TEMPMON_Deinit (*C function*), 887
drivers_RT_RT1020_MIMXRT1024.TEMPMON_GetCurrentTemperature (*C function*), 888
drivers_RT_RT1020_MIMXRT1024.TEMPMON_GetDefaultConfig (*C function*), 887
drivers_RT_RT1020_MIMXRT1024.TEMPMON_HOTCOUNTMASK (*C macro*), 888
drivers_RT_RT1020_MIMXRT1024.TEMPMON_HOTCOUNTSHIFT (*C macro*), 888
drivers_RT_RT1020_MIMXRT1024.TEMPMON_HOTTEMPMASK (*C macro*), 888
drivers_RT_RT1020_MIMXRT1024.TEMPMON_HOTTEMPSHIFT (*C macro*), 888
drivers_RT_RT1020_MIMXRT1024.TEMPMON_Init (*C function*), 887
drivers_RT_RT1020_MIMXRT1024.TEMPMON_ROOMCOUNTMASK (*C macro*), 888
drivers_RT_RT1020_MIMXRT1024.TEMPMON_ROOMCOUNTSHIFT (*C macro*), 888
drivers_RT_RT1020_MIMXRT1024.TEMPMON_SetTempAlarm (*C function*), 888
drivers_RT_RT1020_MIMXRT1024.TEMPMON_StartMeasure (*C function*), 887
drivers_RT_RT1020_MIMXRT1024.TEMPMON_StopMeasure (*C function*), 888
drivers_RT_RT1020_MIMXRT1024.TMR_CLOCKS (*C macro*), 237
drivers_RT_RT1020_MIMXRT1024.TMR_CSCTRL_OFLAG_MASK (*C macro*), 768
drivers_RT_RT1020_MIMXRT1024.TMR_CSCTRL_OFLAG_SHIFT (*C macro*), 768
drivers_RT_RT1020_MIMXRT1024.trng_clock_mode_t (*C type*), 891
drivers_RT_RT1020_MIMXRT1024.TRNG_CLOCKS (*C macro*), 237
drivers_RT_RT1020_MIMXRT1024.trng_config_t (*C type*), 891
drivers_RT_RT1020_MIMXRT1024.TRNG_Deinit (*C function*), 892
drivers_RT_RT1020_MIMXRT1024.TRNG_GetDefaultConfig (*C function*), 891
drivers_RT_RT1020_MIMXRT1024.TRNG_GetRandomData (*C function*), 892
drivers_RT_RT1020_MIMXRT1024.TRNG_Init (*C function*), 892
drivers_RT_RT1020_MIMXRT1024.trng_ring_osc_div_t (*C type*), 891
drivers_RT_RT1020_MIMXRT1024.trng_sample_mode_t (*C type*), 891
drivers_RT_RT1020_MIMXRT1024.trng_statistical_check_limit_t (*C type*), 891
drivers_RT_RT1020_MIMXRT1024.UART_RETRY_TIMES (*C macro*), 468, 705
drivers_RT_RT1020_MIMXRT1024.UINT16_MAX (*C macro*), 609
drivers_RT_RT1020_MIMXRT1024.UINT32_MAX (*C macro*), 609
drivers_RT_RT1020_MIMXRT1024.UINT64_H (*C macro*), 609
drivers_RT_RT1020_MIMXRT1024.UINT64_L (*C macro*), 609
drivers_RT_RT1020_MIMXRT1024.UINTPTR_SIZE (*C macro*), 609
drivers_RT_RT1020_MIMXRT1024.USDHC_AdjustDelayForManualTuning (*C function*), 902
drivers_RT_RT1020_MIMXRT1024.USDHC_ADMA1_ADDRESS_ALIGN (*C macro*), 921
drivers_RT_RT1020_MIMXRT1024.USDHC_ADMA1_DESCRIPTOR_ADDRESS_MASK (*C macro*), 922
drivers_RT_RT1020_MIMXRT1024.USDHC_ADMA1_DESCRIPTOR_ADDRESS_SHIFT (*C macro*), 921
drivers_RT_RT1020_MIMXRT1024.USDHC_ADMA1_DESCRIPTOR_LENGTH_MASK (*C macro*), 922
drivers_RT_RT1020_MIMXRT1024.USDHC_ADMA1_DESCRIPTOR_LENGTH_SHIFT (*C macro*), 922
drivers_RT_RT1020_MIMXRT1024.USDHC_ADMA1_DESCRIPTOR_MAX_LENGTH_PER_ENTRY (*C macro*), 922

drivers_RT_RT1020_MIMXRT1024.usdhc_adma1_descriptor_t (*C type*), 920
drivers_RT_RT1020_MIMXRT1024.USDHC_ADMA1_LENGTH_ALIGN (*C macro*), 921
drivers_RT_RT1020_MIMXRT1024.USDHC_ADMA2_ADDRESS_ALIGN (*C macro*), 921
drivers_RT_RT1020_MIMXRT1024.USDHC_ADMA2_DESCRIPTOR_LENGTH_MASK (*C macro*), 923
drivers_RT_RT1020_MIMXRT1024.USDHC_ADMA2_DESCRIPTOR_LENGTH_SHIFT (*C macro*), 922
drivers_RT_RT1020_MIMXRT1024.USDHC_ADMA2_DESCRIPTOR_MAX_LENGTH_PER_ENTRY (*C macro*), 923
drivers_RT_RT1020_MIMXRT1024.usdhc_adma2_descriptor_t (*C type*), 920
drivers_RT_RT1020_MIMXRT1024.USDHC_ADMA2_LENGTH_ALIGN (*C macro*), 921
drivers_RT_RT1020_MIMXRT1024.usdhc_adma_config_t (*C type*), 920
drivers_RT_RT1020_MIMXRT1024.USDHC_AssertHardwareReset (*C function*), 898
drivers_RT_RT1020_MIMXRT1024.usdhc_boot_config_t (*C type*), 920
drivers_RT_RT1020_MIMXRT1024.usdhc_boot_mode_t (*C type*), 920
drivers_RT_RT1020_MIMXRT1024.usdhc_burst_len_t (*C type*), 920
drivers_RT_RT1020_MIMXRT1024.usdhc_capability_t (*C type*), 920
drivers_RT_RT1020_MIMXRT1024.usdhc_card_command_type_t (*C type*), 920
drivers_RT_RT1020_MIMXRT1024.usdhc_card_response_type_t (*C type*), 920
drivers_RT_RT1020_MIMXRT1024.USDHC_CardDetectByData3 (*C function*), 899
drivers_RT_RT1020_MIMXRT1024.USDHC_CheckStdTuningResult (*C function*), 902
drivers_RT_RT1020_MIMXRT1024.USDHC_CheckTuningError (*C function*), 903
drivers_RT_RT1020_MIMXRT1024.USDHC_ClearInterruptStatusFlags (*C function*), 897
drivers_RT_RT1020_MIMXRT1024.USDHC_CLOCKS (*C macro*), 237
drivers_RT_RT1020_MIMXRT1024.usdhc_command_t (*C type*), 920
drivers_RT_RT1020_MIMXRT1024.usdhc_config_t (*C type*), 920
drivers_RT_RT1020_MIMXRT1024.USDHC_ConfigStrobeDLL (*C function*), 903
drivers_RT_RT1020_MIMXRT1024.usdhc_data_bus_width_t (*C type*), 919
drivers_RT_RT1020_MIMXRT1024.usdhc_data_t (*C type*), 921
drivers_RT_RT1020_MIMXRT1024.USDHC_Deinit (*C function*), 895
drivers_RT_RT1020_MIMXRT1024.USDHC_DetectCardInsert (*C function*), 899
drivers_RT_RT1020_MIMXRT1024.USDHC_DisableInterruptSignal (*C function*), 897
drivers_RT_RT1020_MIMXRT1024.USDHC_DisableInterruptStatus (*C function*), 896
drivers_RT_RT1020_MIMXRT1024.usdhc_dma_mode_t (*C type*), 920
drivers_RT_RT1020_MIMXRT1024.USDHC_DriverIRQHandler (*C function*), 905
drivers_RT_RT1020_MIMXRT1024.USDHC_EnableAutoTuning (*C function*), 901
drivers_RT_RT1020_MIMXRT1024.USDHC_EnableAutoTuningForCmdAndData (*C function*), 901
drivers_RT_RT1020_MIMXRT1024.USDHC_EnableDDRMode (*C function*), 903
drivers_RT_RT1020_MIMXRT1024.USDHC_EnableHS400Mode (*C function*), 903
drivers_RT_RT1020_MIMXRT1024.USDHC_EnableInternalDMA (*C function*), 896
drivers_RT_RT1020_MIMXRT1024.USDHC_EnableInterruptSignal (*C function*), 897
drivers_RT_RT1020_MIMXRT1024.USDHC_EnableInterruptStatus (*C function*), 896
drivers_RT_RT1020_MIMXRT1024.USDHC_EnableManualTuning (*C function*), 901
drivers_RT_RT1020_MIMXRT1024.USDHC_EnableMmcBoot (*C function*), 900
drivers_RT_RT1020_MIMXRT1024.USDHC_EnableSdioControl (*C function*), 900
drivers_RT_RT1020_MIMXRT1024.USDHC_EnableStandardTuning (*C function*), 902
drivers_RT_RT1020_MIMXRT1024.USDHC_EnableStrobeDLL (*C function*), 903
drivers_RT_RT1020_MIMXRT1024.USDHC_EnableWakeupEvent (*C function*), 899
drivers_RT_RT1020_MIMXRT1024.usdhc_endian_mode_t (*C type*), 919
drivers_RT_RT1020_MIMXRT1024.USDHC_ForceClockOn (*C function*), 898
drivers_RT_RT1020_MIMXRT1024.USDHC_GetAdmaErrorStatusFlags (*C function*), 897
drivers_RT_RT1020_MIMXRT1024.USDHC_GetAutoCommand12ErrorStatusFlags (*C function*), 897
drivers_RT_RT1020_MIMXRT1024.USDHC_GetCapability (*C function*), 898
drivers_RT_RT1020_MIMXRT1024.USDHC_GetEnabledInterruptStatusFlags (*C function*), 897
drivers_RT_RT1020_MIMXRT1024.USDHC_GetExecuteStdTuningStatus (*C function*), 902
drivers_RT_RT1020_MIMXRT1024.USDHC_GetInterruptStatusFlags (*C function*), 897
drivers_RT_RT1020_MIMXRT1024.USDHC_GetPresentStatusFlags (*C function*), 898
drivers_RT_RT1020_MIMXRT1024.USDHC_GetStrobeDLLStatus (*C function*), 903

drivers_RT_RT1020_MIMXRT1024.USDHC_GetTuningDelayStatus (*C function*), 901
drivers_RT_RT1020_MIMXRT1024.usdhc_handle_t (*C type*), 921
drivers_RT_RT1020_MIMXRT1024.usdhc_host_t (*C type*), 921
drivers_RT_RT1020_MIMXRT1024.USDHC_Init (*C function*), 894
drivers_RT_RT1020_MIMXRT1024.USDHC_MAX_BLOCK_COUNT (*C macro*), 921
drivers_RT_RT1020_MIMXRT1024.USDHC_ReadData (*C function*), 899
drivers_RT_RT1020_MIMXRT1024.USDHC_RequestReTuning (*C function*), 901
drivers_RT_RT1020_MIMXRT1024.USDHC_RequestStopAtBlockGap (*C function*), 900
drivers_RT_RT1020_MIMXRT1024.USDHC_RequestTuningForSDR50 (*C function*), 901
drivers_RT_RT1020_MIMXRT1024.USDHC_Reset (*C function*), 895
drivers_RT_RT1020_MIMXRT1024.USDHC_ResetStrobeDLL (*C function*), 903
drivers_RT_RT1020_MIMXRT1024.usdhc_scatter_gather_data_list_t (*C type*), 920
drivers_RT_RT1020_MIMXRT1024.usdhc_scatter_gather_data_t (*C type*), 920
drivers_RT_RT1020_MIMXRT1024.usdhc_scatter_gather_transfer_t (*C type*), 921
drivers_RT_RT1020_MIMXRT1024.USDHC_SendCommand (*C function*), 899
drivers_RT_RT1020_MIMXRT1024.USDHC_SetADMA1Descriptor (*C function*), 896
drivers_RT_RT1020_MIMXRT1024.USDHC_SetADMA2Descriptor (*C function*), 896
drivers_RT_RT1020_MIMXRT1024.USDHC_SetAdmaTableConfig (*C function*), 895
drivers_RT_RT1020_MIMXRT1024.USDHC_SetCardActive (*C function*), 898
drivers_RT_RT1020_MIMXRT1024.USDHC_SetContinueRequest (*C function*), 900
drivers_RT_RT1020_MIMXRT1024.USDHC_SetDataBusWidth (*C function*), 899
drivers_RT_RT1020_MIMXRT1024.USDHC_SetDataConfig (*C function*), 904
drivers_RT_RT1020_MIMXRT1024.USDHC_SetForceEvent (*C function*), 900
drivers_RT_RT1020_MIMXRT1024.USDHC_SetInternalDmaConfig (*C function*), 895
drivers_RT_RT1020_MIMXRT1024.USDHC_SetMmcBootConfig (*C function*), 900
drivers_RT_RT1020_MIMXRT1024.USDHC_SetSdClock (*C function*), 898
drivers_RT_RT1020_MIMXRT1024.USDHC_SetStandardTuningCounter (*C function*), 902
drivers_RT_RT1020_MIMXRT1024.USDHC_SetStrobeDllOverride (*C function*), 903
drivers_RT_RT1020_MIMXRT1024.USDHC_SetTuningDelay (*C function*), 901
drivers_RT_RT1020_MIMXRT1024.usdhc_transfer_callback_t (*C type*), 921
drivers_RT_RT1020_MIMXRT1024.usdhc_transfer_direction_t (*C type*), 919
drivers_RT_RT1020_MIMXRT1024.usdhc_transfer_function_t (*C type*), 921
drivers_RT_RT1020_MIMXRT1024.usdhc_transfer_t (*C type*), 921
drivers_RT_RT1020_MIMXRT1024.USDHC_TransferBlocking (*C function*), 904
drivers_RT_RT1020_MIMXRT1024.USDHC_TransferCreateHandle (*C function*), 904
drivers_RT_RT1020_MIMXRT1024.USDHC_TransferHandleIRQ (*C function*), 905
drivers_RT_RT1020_MIMXRT1024.USDHC_TransferNonBlocking (*C function*), 904
drivers_RT_RT1020_MIMXRT1024.USDHC_WriteData (*C function*), 899
drivers_RT_RT1020_MIMXRT1024.USEC_TO_COUNT (*C macro*), 611
drivers_RT_RT1020_MIMXRT1024.WDOG_ClearInterruptStatus (*C function*), 931
drivers_RT_RT1020_MIMXRT1024.WDOG_CLOCKS (*C macro*), 237
drivers_RT_RT1020_MIMXRT1024.wdog_config_t (*C type*), 932
drivers_RT_RT1020_MIMXRT1024.WDOG_Deinit (*C function*), 929
drivers_RT_RT1020_MIMXRT1024.WDOG_Disable (*C function*), 930
drivers_RT_RT1020_MIMXRT1024.WDOG_DisablePowerDownEnable (*C function*), 931
drivers_RT_RT1020_MIMXRT1024.WDOG_Enable (*C function*), 929
drivers_RT_RT1020_MIMXRT1024.WDOG_EnableInterrupts (*C function*), 930
drivers_RT_RT1020_MIMXRT1024.WDOG_FIRST_WORD_OF_REFRESH (*C macro*), 772
drivers_RT_RT1020_MIMXRT1024.WDOG_FIRST_WORD_OF_UNLOCK (*C macro*), 772
drivers_RT_RT1020_MIMXRT1024.WDOG_GetDefaultConfig (*C function*), 929
drivers_RT_RT1020_MIMXRT1024.WDOG_GetStatusFlags (*C function*), 930
drivers_RT_RT1020_MIMXRT1024.WDOG_Init (*C function*), 929
drivers_RT_RT1020_MIMXRT1024.WDOG_Refresh (*C function*), 931
drivers_RT_RT1020_MIMXRT1024.WDOG_REFRESH_KEY (*C macro*), 932
drivers_RT_RT1020_MIMXRT1024.WDOG_SECOND_WORD_OF_REFRESH (*C macro*), 772
drivers_RT_RT1020_MIMXRT1024.WDOG_SECOND_WORD_OF_UNLOCK (*C macro*), 772
drivers_RT_RT1020_MIMXRT1024.WDOG_SetInterrputTimeoutValue (*C function*), 931
drivers_RT_RT1020_MIMXRT1024.WDOG_SetTimeoutValue (*C function*), 931

drivers_RT_RT1020_MIMXRT1024.WDOG_TriggerSoftwareSignal (*C function*), 930
drivers_RT_RT1020_MIMXRT1024.WDOG_TriggerSystemSoftwareReset (*C function*), 930
drivers_RT_RT1020_MIMXRT1024.wdog_work_mode_t (*C type*), 932
drivers_RT_RT1020_MIMXRT1024.xbara_active_edge_t (*C type*), 936
drivers_RT_RT1020_MIMXRT1024.XBARA_ClearStatusFlags (*C function*), 934
drivers_RT_RT1020_MIMXRT1024.XBARA_CLOCKS (*C macro*), 237
drivers_RT_RT1020_MIMXRT1024.xbara_control_config_t (*C type*), 936
drivers_RT_RT1020_MIMXRT1024.XBARA_Deinit (*C function*), 934
drivers_RT_RT1020_MIMXRT1024.XBARA_GetStatusFlags (*C function*), 934
drivers_RT_RT1020_MIMXRT1024.XBARA_Init (*C function*), 933
drivers_RT_RT1020_MIMXRT1024.xbara_request_t (*C type*), 936
drivers_RT_RT1020_MIMXRT1024.XBARA_SELx (*C macro*), 936
drivers_RT_RT1020_MIMXRT1024.XBARA_SetOutputSignalConfig (*C function*), 934
drivers_RT_RT1020_MIMXRT1024.XBARA_SetSignalsConnection (*C function*), 934
drivers_RT_RT1020_MIMXRT1024.xbara_status_flag_t (*C type*), 936
drivers_RT_RT1020_MIMXRT1024.XBARA_WR_SELx_SELx (*C macro*), 936
drivers_RT_RT1020_MIMXRT1024.XBARAControlConfig (*C struct*), 936
drivers_RT_RT1020_MIMXRT1024.XBARAControlConfig.activeEdge (*C var*), 937
drivers_RT_RT1020_MIMXRT1024.XBARAControlConfig.requestType (*C var*), 937
drivers_RT_RT1020_MIMXRT1024.XBARB_CLOCKS (*C macro*), 237
drivers_RT_RT1020_MIMXRT1024.XBARB_Deinit (*C function*), 937
drivers_RT_RT1020_MIMXRT1024.XBARB_Init (*C function*), 937
drivers_RT_RT1020_MIMXRT1024.XBARB_SELx (*C macro*), 937
drivers_RT_RT1020_MIMXRT1024.XBARB_SetSignalsConnection (*C function*), 937
drivers_RT_RT1020_MIMXRT1024.XBARB_WR_SELx_SELx (*C macro*), 937