MCUXpresso SDK Documentation

Release 25.06.00



NXP
Jun 26, 2025

Table of contents

1 MIMXRT1062 **3**

1.1 ADC: 12-bit Analog to Digital Converter Driver . . . 3
1.2 ADC_ETC: ADC External Trigger Control . . . 11
1.3 AIPSTZ: AHB to IP Bridge . . . 15
1.4 AOI: Crossbar AND/OR/INVERT Driver . . . 17
1.5 BEE: Bus Encryption Engine . . . 20
1.6 CACHE: ARMV7-M7 CACHE Memory Controller . . . 25
1.7 Clock Driver . . . 28
1.8 CMP: Analog Comparator Driver . . . 70
1.9 CSI: CMOS Sensor Interface . . . 74
1.10 DCDC: DCDC Converter . . . 84
1.11 DCP: Data Co-Processor . . . 93
1.12 DCP AES blocking driver . . . 97
1.13 DCP HASH driver . . . 99
1.14 DCP AES non-blocking driver . . . 101
1.15 DMAMUX: Direct Memory Access Multiplexer Driver . . . 103
1.16 eDMA: Enhanced Direct Memory Access (eDMA) Controller Driver . . . 104
1.17 eLCDIF: Enhanced LCD Interface . . . 124
1.18 ENC: Quadrature Encoder/Decoder . . . 134
1.19 ENET: Ethernet MAC Driver . . . 144
1.20 EWM: External Watchdog Monitor Driver . . . 175
1.21 FlexCAN: Flex Controller Area Network Driver . . . 178
1.22 FlexCAN Driver . . . 178
1.23 FlexCAN eDMA Driver . . . 217
1.24 FlexIO: FlexIO Driver . . . 219
1.25 FlexIO Driver . . . 219
1.26 FlexIO eDMA I2S Driver . . . 236
1.27 FlexIO eDMA SPI Driver . . . 240
1.28 FlexIO eDMA UART Driver . . . 243
1.29 FlexIO I2C Master Driver . . . 246
1.30 FlexIO I2S Driver . . . 255
1.31 FlexIO SPI Driver . . . 265
1.32 FlexIO UART Driver . . . 278
1.33 FLEXRAM: on-chip RAM manager . . . 289
1.34 FLEXSPI: Flexible Serial Peripheral Interface Driver . . . 296
1.35 FLEXSPI eDMA Driver . . . 313
1.36 GPC: General Power Controller Driver . . . 315
1.37 GPIO: General-Purpose Input/Output Driver . . . 317
1.38 GPT: General Purpose Timer . . . 321
1.39 IOMUXC: IOMUX Controller . . . 328
1.40 KPP: KeyPad Port Driver . . . 362
1.41 Common Driver . . . 365
1.42 Lin_lpuart_driver . . . 376
1.43 LPI2C: Low Power Inter-Integrated Circuit Driver . . . 384
1.44 LPI2C Master Driver . . . 385
1.45 LPI2C Master DMA Driver . . . 399

1.46 LPI2C Slave Driver . . . 402
1.47 LPSPI: Low Power Serial Peripheral Interface . . . 412
1.48 LPSPI Peripheral driver . . . 412
1.49 LPSPI eDMA Driver . . . 433
1.50 LPUART: Low Power Universal Asynchronous Receiver/Transmitter Driver . . . 441
1.51 LPUART Driver . . . 441
1.52 LPUART eDMA Driver . . . 459
1.53 Nic301 . . . 462
1.54 OCOTP: On Chip One-Time Programmable controller. . . . 467
1.55 PIT: Periodic Interrupt Timer . . . 470
1.56 PMU: Power Management Unit . . . 474
1.57 PWM: Pulse Width Modulator . . . 482
1.58 PXP: Pixel Pipeline . . . 506
1.59 QTMR: Quad Timer Driver . . . 542
1.60 Romapi . . . 552
1.61 RTWDOG: 32-bit Watchdog Timer . . . 566
1.62 SAI: Serial Audio Interface . . . 572
1.63 SAI Driver . . . 572
1.64 SAI EDMA Driver . . . 598
1.65 SEMC: Smart External DRAM Controller Driver . . . 605
1.66 SNVS: Secure Non-Volatile Storage . . . 630
1.67 Secure Non-Volatile Storage High-Power . . . 630
1.68 Secure Non-Volatile Storage Low-Power . . . 638
1.69 SPDIF: Sony/Philips Digital Interface . . . 648
1.70 SPDIF eDMA Driver . . . 661
1.71 SRC: System Reset Controller Driver . . . 665
1.72 TEMPMON: Temperature Monitor Module . . . 670
1.73 TRNG: True Random Number Generator . . . 672
1.74 TSC: Touch Screen Controller Driver . . . 676
1.75 USDHC: Ultra Secured Digital Host Controller Driver . . . 683
1.76 WDOG: Watchdog Timer Driver . . . 713
1.77 XBARA: Inter-Peripheral Crossbar Switch . . . 717
1.78 XBARB: Inter-Peripheral Crossbar Switch . . . 720

2 Middleware **723**
2.1 Boot . . . 723
2.1.1 MCUXpresso SDK : mcuxsdk-middleware-mcuboot_opensource . . . 723
2.1.2 MCUboot . . . 724
2.2 Cloud . . . 725
2.2.1 AWS IoT . . . 725
2.3 Connectivity . . . 734
2.3.1 lwIP . . . 734
2.4 eIQ . . . 735
2.4.1 eIQ . . . 735
2.5 File System . . . 763
2.5.1 FatFs . . . 763
2.6 Motor Control . . . 765
2.6.1 FreeMASTER . . . 765
2.7 MultiCore . . . 803
2.7.1 Multicore SDK . . . 803
2.8 Multimedia . . . 898
2.8.1 Audio Voice . . . 898
2.9 Wireless . . . 980
2.9.1 NXP Wireless Framework and Stacks . . . 980
2.9.2 EdgeFast Bluetooth . . . 1031

3 RTOS **1107**
3.1 FreeRTOS . . . 1107

3.1.1 FreeRTOS kernel 1107
3.1.2 FreeRTOS drivers 1107
3.1.3 backoffalgorithm 1107
3.1.4 corehttp 1107
3.1.5 corejson 1107
3.1.6 coremqtt 1108
3.1.7 coremqtt-agent 1108
3.1.8 corepkcs11 1108
3.1.9 freertos-plus-tcp 1108

This documentation contains information specific to the evkcmimxrt1060 board.

Chapter 1

MIMXRT1062

1.1 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.

1.2 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.

1.3 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.

1.4 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;

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

1.5 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

1.6 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 2.0.4.

1.7 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_ArmPllClk

ARMPLLCLK.

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_Usb1Sw120MClk
USB1PLLSw120MCLK

enumerator kCLOCK_Usb1Sw60MClk
USB1PLLSw60MCLK

enumerator kCLOCK_Usb1Sw80MClk
USB1PLLSw80MCLK

enumerator kCLOCK_Usb2PllClk
USB2PLLCLK.

enumerator kCLOCK_SysPllClk
SYSPLLCLK.

enumerator kCLOCK_SysPllPfd0Clk
SYSPLLPDF0CLK.

enumerator kCLOCK_SysPllPfd1Clk
SYSPLLPFD1CLK.

enumerator kCLOCK_SysPllPfd2Clk
SYSPLLPFD2CLK.

enumerator kCLOCK_SysPllPfd3Clk
SYSPLLPFD3CLK.

enumerator kCLOCK_EnetPll0Clk
Enet PLLCLK ref_enetpll0.

enumerator kCLOCK_EnetPll1Clk
Enet PLLCLK ref_enetpll1.

enumerator kCLOCK_EnetPll2Clk
Enet PLLCLK ref_enetpll2.

enumerator kCLOCK_AudioPllClk
Audio PLLCLK.

enumerator kCLOCK_VideoPllClk
Video 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

enumerator kCLOCK_FlexSpiExsc
CCGR0, CG3

enumerator kCLOCK_Sim_M_Main
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_Aoi2
CCGR1, CG7

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_Csi
CCGR2, CG1

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_Xbar3
CCGR2, CG7

enumerator kCLOCK_Ipmux1
CCGR2, CG8

enumerator kCLOCK_Ipmux2
CCGR2, CG9

enumerator kCLOCK_Ipmux3
CCGR2, CG10

enumerator kCLOCK_Xbar1
CCGR2, CG11

enumerator kCLOCK_Xbar2
CCGR2, CG12

enumerator kCLOCK_Gpio3
CCGR2, CG13

enumerator kCLOCK_Lcd
CCGR2, CG14

enumerator kCLOCK_Pxp
CCGR2, CG15

enumerator kCLOCK_Flexio2
CCGR3, CG0

enumerator kCLOCK_Lpuart5
CCGR3, CG1

enumerator kCLOCK_Semc
CCGR3, CG2

enumerator kCLOCK_Lpuart6
CCGR3, CG3

enumerator kCLOCK_Aoi1
CCGR3, CG4

enumerator kCLOCK_LcdPixel
CCGR3, CG5

enumerator kCLOCK_Gpio4
CCGR3, CG6

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_Ocram
CCGR3, CG14

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

enumerator kCLOCK_SimM
CCGR4, CG6

enumerator kCLOCK_SimEms
CCGR4, CG7

enumerator kCLOCK_Pwm1
CCGR4, CG8

enumerator kCLOCK_Pwm2
CCGR4, CG9

enumerator kCLOCK_Pwm3
CCGR4, CG10

enumerator kCLOCK_Pwm4
CCGR4, CG11

enumerator kCLOCK_Enc1
CCGR4, CG12

enumerator kCLOCK_Enc2
CCGR4, CG13

enumerator kCLOCK_Enc3
CCGR4, CG14

enumerator kCLOCK_Enc4
CCGR4, CG15

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_SimMain
CCGR5, CG8

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

enumerator kCLOCK_Timer3
CCGR6, CG15

enumerator kCLOCK_Enet2
CCGR7, CG0

enumerator kCLOCK_FlexSpi2
CCGR7, CG1

enumerator kCLOCK_Axbs_l
CCGR7, CG2

enumerator kCLOCK_Can3
CCGR7, CG3

enumerator kCLOCK_Can3S
CCGR7, CG4

enumerator kCLOCK_Aips_lite
CCGR7, CG5

enumerator kCLOCK_Flexio3
CCGR7, CG6

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_Flexspi2Mux
flexspi2 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_Flexio2Mux
flexio2 mux name

enumerator kCLOCK_CanMux
can mux name

enumerator kCLOCK_UartMux
uart mux name

enumerator kCLOCK_SpdifMux
spdif mux name

enumerator kCLOCK_Flexio1Mux
flexio1 mux name

enumerator kCLOCK_Lpi2cMux
lpi2c mux name

enumerator kCLOCK_LcdifPreMux
lcdif pre mux name

enumerator kCLOCK_CsiMux
csi mux name

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 clock divider set to divided by 1.

enumerator kCLOCK_SemcDivBy2
SEMC clock divider set to divided by 2.

enumerator kCLOCK_SemcDivBy3
SEMC clock divider set to divided by 3.

enumerator kCLOCK_SemcDivBy4
SEMC clock divider set to divided by 4.

enumerator kCLOCK_SemcDivBy5
SEMC clock divider set to divided by 5.

enumerator kCLOCK_SemcDivBy6
SEMC clock divider set to divided by 6.

enumerator kCLOCK_SemcDivBy7
SEMC clock divider set to divided by 7.

enumerator kCLOCK_SemcDivBy8
SEMC clock divider set to divided by 8.

enumerator kCLOCK_AhbDivBy1
AHB clock divider set to divided by 1.

enumerator kCLOCK_AhbDivBy2
AHB clock divider set to divided by 2.

enumerator kCLOCK_AhbDivBy3
AHB clock divider set to divided by 3.

enumerator kCLOCK_AhbDivBy4
AHB clock divider set to divided by 4.

enumerator kCLOCK_AhbDivBy5
AHB clock divider set to divided by 5.

enumerator kCLOCK_AhbDivBy6
AHB clock divider set to divided by 6.

enumerator kCLOCK_AhbDivBy7
AHB clock divider set to divided by 7.

enumerator kCLOCK_AhbDivBy8
AHB clock divider set to divided by 8.

enumerator kCLOCK_IpgDivBy1
IPG clock divider set to divided by 1.

enumerator kCLOCK_IpgDivBy2
IPG clock divider set to divided by 2.

enumerator kCLOCK_IpgDivBy3
IPG clock divider set to divided by 3.

enumerator kCLOCK_IpgDivBy4
IPG clock divider set to divided by 4.

enumerator kCLOCK_Flexspi2DivBy1
Flexspi2 divider set to divided by 1.

enumerator kCLOCK_Flexspi2DivBy2
Flexspi2 divider set to divided by 2.

enumerator kCLOCK_Flexspi2DivBy3
Flexspi2 divider set to divided by 3.

enumerator kCLOCK_Flexspi2DivBy4
Flexspi2 divider set to divided by 4.

enumerator kCLOCK_Flexspi2DivBy5
Flexspi2 divider set to divided by 5.

enumerator kCLOCK_Flexspi2DivBy6
Flexspi2 divider set to divided by 6.

enumerator kCLOCK_Flexspi2DivBy7
Flexspi2 divider set to divided by 7.

enumerator kCLOCK_Flexspi2DivBy8
Flexspi2 divider set to divided by 8.

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_LcdifDivBy1
Lcdif divider set to divided by 1.

enumerator kCLOCK_LcdifDivBy2
Lcdif divider set to divided by 2.

enumerator kCLOCK_LcdifDivBy3
Lcdif divider set to divided by 3.

enumerator kCLOCK_LcdifDivBy4
Lcdif divider set to divided by 4.

enumerator kCLOCK_LcdifDivBy5
Lcdif divider set to divided by 5.

enumerator kCLOCK_LcdifDivBy6
Lcdif divider set to divided by 6.

enumerator kCLOCK_LcdifDivBy7
Lcdif divider set to divided by 7.

enumerator kCLOCK_LcdifDivBy8
Lcdif 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_Flexio2DivBy1
Flexio2 divider set to divided by 1.

enumerator kCLOCK_Flexio2DivBy2
Flexio2 divider set to divided by 2.

enumerator kCLOCK_Flexio2DivBy3
Flexio2 divider set to divided by 3.

enumerator kCLOCK_Flexio2DivBy4
Flexio2 divider set to divided by 4.

enumerator kCLOCK_Flexio2DivBy5
Flexio2 divider set to divided by 5.

enumerator kCLOCK_Flexio2DivBy6
Flexio2 divider set to divided by 6.

enumerator kCLOCK_Flexio2DivBy7
Flexio2 divider set to divided by 7.

enumerator kCLOCK_Flexio2DivBy8
Flexio2 divider set to divided by 8.

enumerator kCLOCK_Sai3PreDivBy1
Sai3Pre divider set to divided by 1.

enumerator kCLOCK_Sai3PreDivBy2
Sai3Pre divider set to divided by 2.

enumerator kCLOCK_Sai3PreDivBy3
Sai3Pre divider set to divided by 3.

enumerator kCLOCK_Sai3PreDivBy4
Sai3Pre divider set to divided by 4.

enumerator kCLOCK_Sai3PreDivBy5
Sai3Pre divider set to divided by 5.

enumerator kCLOCK_Sai3PreDivBy6
Sai3Pre divider set to divided by 6.

enumerator kCLOCK_Sai3PreDivBy7
Sai3Pre divider set to divided by 7.

enumerator kCLOCK_Sai3PreDivBy8
Sai3Pre divider set to divided by 8.

enumerator kCLOCK_Flexio2PreDivBy1
Flexio2Pre divider set to divided by 1.

enumerator kCLOCK_Flexio2PreDivBy2
Flexio2Pre divider set to divided by 2.

enumerator kCLOCK_Flexio2PreDivBy3
Flexio2Pre divider set to divided by 3.

enumerator kCLOCK_Flexio2PreDivBy4
Flexio2Pre divider set to divided by 4.

enumerator kCLOCK_Flexio2PreDivBy5
Flexio2Pre divider set to divided by 5.

enumerator kCLOCK_Flexio2PreDivBy6
Flexio2Pre divider set to divided by 6.

enumerator kCLOCK_Flexio2PreDivBy7
Flexio2Pre divider set to divided by 7.

enumerator kCLOCK_Flexio2PreDivBy8
Flexio2Pre divider set to divided by 8.

enumerator kCLOCK_Sai1PreDivBy1
Sai1Pre divider set to divided by 1.

enumerator kCLOCK_Sai1PreDivBy2
Sai1Pre divider set to divided by 2.

enumerator kCLOCK_Sai1PreDivBy3
Sai1Pre divider set to divided by 3.

enumerator kCLOCK_Sai1PreDivBy4
Sai1Pre divider set to divided by 4.

enumerator kCLOCK_Sai1PreDivBy5
Sai1Pre divider set to divided by 5.

enumerator kCLOCK_Sai1PreDivBy6
Sai1Pre divider set to divided by 6.

enumerator kCLOCK_Sai1PreDivBy7
Sai1Pre divider set to divided by 7.

enumerator kCLOCK_Sai1PreDivBy8
Sai1Pre divider set to divided by 8.

enumerator kCLOCK_Sai2PreDivBy1
Sai2Pre divider set to divided by 1.

enumerator kCLOCK_Sai2PreDivBy2
Sai2Pre divider set to divided by 2.

enumerator kCLOCK_Sai2PreDivBy3
Sai2Pre divider set to divided by 3.

enumerator kCLOCK_Sai2PreDivBy4
Sai2Pre divider set to divided by 4.

enumerator kCLOCK_Sai2PreDivBy5
Sai2Pre divider set to divided by 5.

enumerator kCLOCK_Sai2PreDivBy6
Sai2Pre divider set to divided by 6.

enumerator kCLOCK_Sai2PreDivBy7
Sai2Pre divider set to divided by 7.

enumerator kCLOCK_Sai2PreDivBy8
Sai2Pre divider set to divided by 8.

enumerator kCLOCK_Spdif0PreDivBy1
Spdif0Pre divider set to divided by 1.

enumerator kCLOCK_Spdif0PreDivBy2
Spdif0Pre divider set to divided by 2.

enumerator kCLOCK_Spdif0PreDivBy3
Spdif0Pre divider set to divided by 3.

enumerator kCLOCK_Spdif0PreDivBy4
Spdif0Pre divider set to divided by 4.

enumerator kCLOCK_Spdif0PreDivBy5
Spdif0Pre divider set to divided by 5.

enumerator kCLOCK_Spdif0PreDivBy6
Spdif0Pre divider set to divided by 6.

enumerator kCLOCK_Spdif0PreDivBy7
Spdif0Pre divider set to divided by 7.

enumerator kCLOCK_Spdif0PreDivBy8
Spdif0Pre divider set to divided by 8.

enumerator kCLOCK_Spdif0DivBy1
Spdif0 divider set to divided by 1.

enumerator kCLOCK_Spdif0DivBy2
Spdif0 divider set to divided by 2.

enumerator kCLOCK_Spdif0DivBy3
Spdif0 divider set to divided by 3.

enumerator kCLOCK_Spdif0DivBy4
Spdif0 divider set to divided by 4.

enumerator kCLOCK_Spdif0DivBy5
Spdif0 divider set to divided by 5.

enumerator kCLOCK_Spdif0DivBy6
Spdif0 divider set to divided by 6.

enumerator kCLOCK_Spdif0DivBy7
Spdif0 divider set to divided by 7.

enumerator kCLOCK_Spdif0DivBy8
Spdif0 divider set to divided by 8.

enumerator kCLOCK_Flexio1PreDivBy1
Flexio1Pre divider set to divided by 1.

enumerator kCLOCK_Flexio1PreDivBy2
Flexio1Pre divider set to divided by 2.

enumerator kCLOCK_Flexio1PreDivBy3
Flexio1Pre divider set to divided by 3.

enumerator kCLOCK_Flexio1PreDivBy8
Flexio1Pre 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_LcdifPreDivBy1
LcdifPre divider set to divided by 1.

enumerator kCLOCK_LcdifPreDivBy2
LcdifPre divider set to divided by 2.

enumerator kCLOCK_LcdifPreDivBy3
LcdifPre divider set to divided by 3.

enumerator kCLOCK_LcdifPreDivBy4
LcdifPre divider set to divided by 4.

enumerator kCLOCK_LcdifPreDivBy5
LcdifPre divider set to divided by 5.

enumerator kCLOCK_LcdifPreDivBy6
LcdifPre divider set to divided by 6.

enumerator kCLOCK_LcdifPreDivBy7
LcdifPre divider set to divided by 7.

enumerator kCLOCK_LcdifPreDivBy8
LcdifPre divider set to divided by 8.

enumerator kCLOCK_CsiDivBy1
Csi divider set to divided by 1.

enumerator kCLOCK_CsiDivBy2
Csi divider set to divided by 2.

enumerator kCLOCK_CsiDivBy3
Csi divider set to divided by 3.

enumerator kCLOCK_CsiDivBy4
Csi divider set to divided by 4.

enumerator kCLOCK_CsiDivBy5
Csi divider set to divided by 5.

enumerator kCLOCK_CsiDivBy6
Csi divider set to divided by 6.

enumerator kCLOCK_CsiDivBy7
Csi divider set to divided by 7.

enumerator kCLOCK_CsiDivBy8
Csi 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_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_Flexspi2Div
flexspi2 div name

enumerator kCLOCK_LpspiDiv
lpspi div name

enumerator kCLOCK_LcdifDiv
lcdif 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_Flexio2Div
flexio2 pre div name

enumerator kCLOCK_Sai3PreDiv
sai3 pre div name

enumerator kCLOCK_Sai3Div
sai3 div name

enumerator kCLOCK_Flexio2PreDiv
sai3 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 div name

enumerator kCLOCK_Spdif0PreDiv
spdif pre div name

enumerator kCLOCK_Spdif0Div
spdif div name

enumerator kCLOCK_Flexio1PreDiv
flexio1 pre div name

enumerator kCLOCK_Flexio1Div
flexio1 div name

enumerator kCLOCK_Lpi2cDiv
lpi2c div name

enumerator kCLOCK_LcdifPreDiv
lcdif pre div name

enumerator kCLOCK_CsiDiv
csi div name

enumerator kCLOCK_NonePreDiv
None Pre div.

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_PllArm
PLL ARM

enumerator kCLOCK_PllSys
PLL SYS

enumerator kCLOCK_PllUsb1
PLL USB1

enumerator kCLOCK_PllAudio
PLL Audio

enumerator kCLOCK_PllVideo
PLL Video

enumerator kCLOCK_PllEnet
PLL Enet0

enumerator kCLOCK_PllEnet2
PLL Enet1

enumerator kCLOCK_PllEnet25M
PLL Enet2

enumerator kCLOCK_PllUsb2
PLL USB2

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_OutputPllUsb1
Selects USB1 PLL clock(Divided by 2) output.

enumerator kCLOCK_OutputPllSys
Selects SYS PLL clock(Divided by 2) output.

enumerator kCLOCK_OutputPllVideo
Selects Video PLL clock(Divided by 2) output.

enumerator kCLOCK_OutputSemcClk
Selects semc clock root output.

enumerator kCLOCK_OutputLcdifPixClk
Selects Lcdif pix 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_OutputCkilSyncClk
Selects Ckil 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_OutputCsiClk
Selects CSI clock root output.

enumerator kCLOCK_OutputOscClk
Selects OSC 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_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_Flexspi2ClkRoot
FLEXSPI2 clock root.

enumerator kCLOCK_CsiClkRoot
CSI 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_LcdifClkRoot
LCD clock root.

enumerator kCLOCK_SpdifClkRoot
SPDIF clock root.

enumerator kCLOCK_Flexio1ClkRoot
FLEXIO1 clock root.

enumerator kCLOCK_Flexio2ClkRoot
FLEXIO2 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_value* clock_div_value_t
Clock divider value.

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_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_arm_pll_config* clock_arm_pll_config_t

PLL configuration for ARM.

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_video_pll_config* clock_video_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_PerClk, kCLOCK_Lpi2cDiv, kCLOCK_CanDiv, kCLOCK_UartDiv, kCLOCK_Sai1Div, kCLOCK_Sai2Div, kCLOCK_Sai3Div 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 2.5.3.

SDK_DEVICE_MAXIMUM_CPU_CLOCK_FREQUENCY

CCM_ANALOG_PLL_BYPASS_SHIFT

CCM_ANALOG_PLL_BYPASS_CLK_SRC_MASK

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

CSCDR3_OFFSET

CACRR_OFFSET

CS1CDR_OFFSET

CS2CDR_OFFSET

PLL_ARM_OFFSET

CCM Analog registers offset.

PLL_SYS_OFFSET

PLL_USB1_OFFSET

PLL_AUDIO_OFFSET

PLL_VIDEO_OFFSET

PLL_ENET_OFFSET

PLL_USB2_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.

CSI_CLOCKS
: Clock ip name array for CSI.

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.

LCDIF_CLOCKS
: Clock ip name array for LCDIF.

LCDIF_PERIPH_CLOCKS
: Clock ip name array for LCDIF PIXEL.

LPI2C_CLOCKS
: Clock ip name array for LPI2C.

LPSPI_CLOCKS
: Clock ip name array for LPSPI.

LPUART_CLOCKS
: Clock ip name array for LPUART.

MQS_CLOCKS
: Clock ip name array for MQS.

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.

PXP_CLOCKS
: Clock ip name array for PXP.

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.

TSC_CLOCKS
: Clock ip name array for TSC.

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_SOURCE

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.

bool CLOCK_EnableUsbhs1Clock(*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_InitArmPll(const *clock_arm_pll_config_t* *config)

Initialize the ARM PLL.

This function initialize the ARM PLL with specific settings

Parameters

- config – configuration to set to PLL.

void CLOCK_DeinitArmPll(void)

De-initialize the ARM PLL.

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_InitUsb2Pll(const *clock_usb_pll_config_t* *config)

Initialize the USB2 PLL.

This function initializes the USB2 PLL with specific settings

Parameters

- config – Configuration to set to PLL.

void CLOCK_DeinitUsb2Pll(void)

Deinitialize the USB2 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_InitVideoPll(const *clock_video_pll_config_t* *config)

Initialize the video PLL.

This function configures the Video PLL with specific settings

Parameters

- config – configuration to set to PLL.

void CLOCK_DeinitVideoPll(void)

De-initialize the Video 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.

bool CLOCK_EnableUsbhs1PhyPllClock(*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_DisableUsbhs1PhyPllClock(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_arm_pll_config

#include <fsl_clock.h> PLL configuration for ARM.

Public Members

uint32_t loopDivider

PLL loop divider. Valid range for divider value: 54-108. Fout=Fin*loopDivider/2.

uint8_t src

Pll clock source, reference _clock_pll_clk_src

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_video_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 enableClkOutput25M

Power on and enable PLL clock output for ENET2 (ref_enetpll2).

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

bool enableClkOutput1

Power on and enable PLL clock output for ENET1 (ref_enetpll1).

uint8_t loopDivider1

Controls the frequency of the ENET1 reference clock. b00 25MHz b01 50MHz b10 100MHz (not 50% duty cycle) b11 125MHz

1.8 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.

1.9 CSI: CMOS Sensor Interface

status_t CSI__Init(CSI_Type *base, const *csi_config_t* *config)
: Initialize the CSI.

  This function enables the CSI peripheral clock, and resets the CSI registers.

  Parameters

  - base – CSI peripheral base address.

- config – Pointer to the configuration structure.

Return values

- kStatus_Success – Initialize successfully.
- kStatus_InvalidArgument – Initialize failed because of invalid argument.

void CSI_Deinit(CSI_Type *base)

De-initialize the CSI.

This function disables the CSI peripheral clock.

Parameters

- base – CSI peripheral base address.

void CSI_Reset(CSI_Type *base)

Reset the CSI.

This function resets the CSI peripheral registers to default status.

Parameters

- base – CSI peripheral base address.

void CSI_GetDefaultConfig(*csi_config_t* *config)

Get the default configuration for to initialize the CSI.

The default configuration value is:

```
config->width = 320U;
config->height = 240U;
config->polarityFlags = kCSI_HsyncActiveHigh | kCSI_DataLatchOnRisingEdge;
config->bytesPerPixel = 2U;
config->linePitch_Bytes = 320U * 2U;
config->workMode = kCSI_GatedClockMode;
config->dataBus = kCSI_DataBus8Bit;
config->useExtVsync = true;
```

Parameters

- config – Pointer to the CSI configuration.

void CSI_ClearFifo(CSI_Type *base, *csi_fifo_t* fifo)

Clear the CSI FIFO.

This function clears the CSI FIFO.

Parameters

- base – CSI peripheral base address.
- fifo – The FIFO to clear.

void CSI_ReflashFifoDma(CSI_Type *base, *csi_fifo_t* fifo)

Reflash the CSI FIFO DMA.

This function reflashes the CSI FIFO DMA.

For RXFIFO, there are two frame buffers. When the CSI module started, it saves the frames to frame buffer 0 then frame buffer 1, the two buffers will be written by turns. After reflash DMA using this function, the CSI is reset to save frame to buffer 0.

Parameters

- base – CSI peripheral base address.
- fifo – The FIFO DMA to reflash.

void CSI_EnableFifoDmaRequest(CSI_Type *base, *csi_fifo_t* fifo, bool enable)

Enable or disable the CSI FIFO DMA request.

Parameters

- base – CSI peripheral base address.
- fifo – The FIFO DMA reques to enable or disable.
- enable – True to enable, false to disable.

static inline void CSI_Start(CSI_Type *base)

Start to receive data.

Parameters

- base – CSI peripheral base address.

static inline void CSI_Stop(CSI_Type *base)

Stop to receiving data.

Parameters

- base – CSI peripheral base address.

void CSI_SetRxBufferAddr(CSI_Type *base, uint8_t index, uint32_t addr)

Set the RX frame buffer address.

Parameters

- base – CSI peripheral base address.
- index – Buffer index.
- addr – Frame buffer address to set.

void CSI_EnableInterrupts(CSI_Type *base, uint32_t mask)

Enables CSI interrupt requests.

Parameters

- base – CSI peripheral base address.
- mask – The interrupts to enable, pass in as OR'ed value of _csi_interrupt_enable.

void CSI_DisableInterrupts(CSI_Type *base, uint32_t mask)

Disable CSI interrupt requests.

Parameters

- base – CSI peripheral base address.
- mask – The interrupts to disable, pass in as OR'ed value of _csi_interrupt_enable.

static inline uint32_t CSI_GetStatusFlags(CSI_Type *base)

Gets the CSI status flags.

Parameters

- base – CSI peripheral base address.

Returns

status flag, it is OR'ed value of _csi_flags.

static inline void CSI_ClearStatusFlags(CSI_Type *base, uint32_t statusMask)

Clears the CSI status flag.

The flags to clear are passed in as OR'ed value of _csi_flags. The following flags are cleared automatically by hardware:

- kCSI_RxFifoFullFlag,
- kCSI_StatFifoFullFlag,
- kCSI_Field0PresentFlag,
- kCSI_Field1PresentFlag,
- kCSI_RxFifoDataReadyFlag,

Parameters

- base – CSI peripheral base address.
- statusMask – The status flags mask, OR'ed value of _csi_flags.

status_t CSI_TransferCreateHandle(CSI_Type *base, *csi_handle_t* *handle, *csi_transfer_callback_t* callback, void *userData)

Initializes the CSI handle.

This function initializes CSI handle, it should be called before any other CSI transactional functions.

Parameters

- base – CSI peripheral base address.
- handle – Pointer to the handle structure.
- callback – Callback function for CSI transfer.
- userData – Callback function parameter.

Return values

kStatus_Success – Handle created successfully.

status_t CSI_TransferStart(CSI_Type *base, *csi_handle_t* *handle)

Start the transfer using transactional functions.

When the empty frame buffers have been submit to CSI driver using function CSI_TransferSubmitEmptyBuffer, user could call this function to start the transfer. The incoming frame will be saved to the empty frame buffer, and user could be optionally notified through callback function.

Parameters

- base – CSI peripheral base address.
- handle – Pointer to the handle structure.

Return values

- kStatus_Success – Started successfully.
- kStatus_CSI_NoEmptyBuffer – Could not start because no empty frame buffer in queue.

status_t CSI_TransferStop(CSI_Type *base, *csi_handle_t* *handle)

Stop the transfer using transactional functions.

The driver does not clean the full frame buffers in queue. In other words, after calling this function, user still could get the full frame buffers in queue using function CSI_TransferGetFullBuffer.

Parameters

- base – CSI peripheral base address.
- handle – Pointer to the handle structure.

Return values

kStatus_Success – Stoped successfully.

status_t CSI_TransferSubmitEmptyBuffer(CSI_Type *base, *csi_handle_t* *handle, uint32_t frameBuffer)

Submit empty frame buffer to queue.

This function could be called before CSI_TransferStart or after CSI_TransferStart. If there is no room in queue to store the empty frame buffer, this function returns error.

Parameters

- base – CSI peripheral base address.
- handle – Pointer to the handle structure.
- frameBuffer – Empty frame buffer to submit.

Return values

- kStatus_Success – Started successfully.
- kStatus_CSI_QueueFull – Could not submit because there is no room in queue.

status_t CSI_TransferGetFullBuffer(CSI_Type *base, *csi_handle_t* *handle, uint32_t *frameBuffer)

Get one full frame buffer from queue.

After the transfer started using function CSI_TransferStart, the incoming frames will be saved to the empty frame buffers in queue. This function gets the full-filled frame buffer from the queue. If there is no full frame buffer in queue, this function returns error.

Parameters

- base – CSI peripheral base address.
- handle – Pointer to the handle structure.
- frameBuffer – Full frame buffer.

Return values

- kStatus_Success – Started successfully.
- kStatus_CSI_NoFullBuffer – There is no full frame buffer in queue.

void CSI_TransferHandleIRQ(CSI_Type *base, *csi_handle_t* *handle)

CSI IRQ handle function.

This function handles the CSI IRQ request to work with CSI driver transactional APIs.

Parameters

- base – CSI peripheral base address.
- handle – CSI handle pointer.

FSL_CSI_DRIVER_VERSION

Error codes for the CSI driver.

Values:

enumerator kStatus_CSI_NoEmptyBuffer
No empty frame buffer in queue to load to CSI.

enumerator kStatus_CSI_NoFullBuffer
No full frame buffer in queue to read out.

enumerator kStatus_CSI_QueueFull
Queue is full, no room to save new empty buffer.

enumerator kStatus_CSI_FrameDone
New frame received and saved to queue.

enum _csi_work_mode
CSI work mode.

The CCIR656 interlace mode is not supported currently.

Values:

enumerator kCSI_GatedClockMode
HSYNC, VSYNC, and PIXCLK signals are used.

enumerator kCSI_NonGatedClockMode
VSYNC, and PIXCLK signals are used.

enumerator kCSI_CCIR656ProgressiveMode
CCIR656 progressive mode.

enum _csi_data_bus
CSI data bus witdh.

Values:

enumerator kCSI_DataBus8Bit
8-bit data bus.

enumerator kCSI_DataBus16Bit
16-bit data bus.

enumerator kCSI_DataBus24Bit
24-bit data bus.

enum _csi_polarity_flags
CSI signal polarity.

Values:

enumerator kCSI_HsyncActiveLow
HSYNC is active low.

enumerator kCSI_HsyncActiveHigh
HSYNC is active high.

enumerator kCSI_DataLatchOnRisingEdge
Pixel data latched at rising edge of pixel clock.

enumerator kCSI_DataLatchOnFallingEdge
Pixel data latched at falling edge of pixel clock.

enumerator kCSI_VsyncActiveHigh
VSYNC is active high.

enumerator kCSI_VsyncActiveLow
VSYNC is active low.

enum _csi_fifo

The CSI FIFO, used for FIFO operation.

Values:

enumerator kCSI_RxFifo
RXFIFO.

enumerator kCSI_StatFifo
STAT FIFO.

enumerator kCSI_AllFifo
Both RXFIFO and STAT FIFO.

enum _csi_interrupt_enable

CSI feature interrupt source.

Values:

enumerator kCSI_EndOfFrameInterruptEnable
End of frame interrupt enable.

enumerator kCSI_ChangeOfFieldInterruptEnable
Change of field interrupt enable.

enumerator kCSI_StatFifoOverrunInterruptEnable
STAT FIFO overrun interrupt enable.

enumerator kCSI_RxFifoOverrunInterruptEnable
RXFIFO overrun interrupt enable.

enumerator kCSI_StatFifoDmaDoneInterruptEnable
STAT FIFO DMA done interrupt enable.

enumerator kCSI_StatFifoFullInterruptEnable
STAT FIFO full interrupt enable.

enumerator kCSI_RxBuffer1DmaDoneInterruptEnable
RX frame buffer 1 DMA transfer done.

enumerator kCSI_RxBuffer0DmaDoneInterruptEnable
RX frame buffer 0 DMA transfer done.

enumerator kCSI_RxFifoFullInterruptEnable
RXFIFO full interrupt enable.

enumerator kCSI_StartOfFrameInterruptEnable
Start of frame (SOF) interrupt enable.

enumerator kCSI_EccErrorInterruptEnable
ECC error detection interrupt enable.

enumerator kCSI_AhbResErrorInterruptEnable
AHB response Error interrupt enable.

enumerator kCSI_BaseAddrChangeErrorInterruptEnable
The DMA output buffer base address changes before DMA completed.

enumerator kCSI_Field0DoneInterruptEnable
Field 0 done interrupt enable.

enumerator kCSI_Field1DoneInterruptEnable
Field 1 done interrupt enable.

enum _csi_flags

CSI status flags.

The following status register flags can be cleared:

- kCSI_EccErrorFlag
- kCSI_AhbResErrorFlag
- kCSI_ChangeOfFieldFlag
- kCSI_StartOfFrameFlag
- kCSI_EndOfFrameFlag
- kCSI_RxBuffer1DmaDoneFlag
- kCSI_RxBuffer0DmaDoneFlag
- kCSI_StatFifoDmaDoneFlag
- kCSI_StatFifoOverrunFlag
- kCSI_RxFifoOverrunFlag
- kCSI_Field0DoneFlag
- kCSI_Field1DoneFlag
- kCSI_BaseAddrChangeErrorFlag

Values:

enumerator kCSI_RxFifoDataReadyFlag

RXFIFO data ready.

enumerator kCSI_EccErrorFlag

ECC error detected.

enumerator kCSI_AhbResErrorFlag

Hresponse (AHB bus response) Error.

enumerator kCSI_ChangeOfFieldFlag

Change of field.

enumerator kCSI_Field0PresentFlag

Field 0 present in CCIR mode.

enumerator kCSI_Field1PresentFlag

Field 1 present in CCIR mode.

enumerator kCSI_StartOfFrameFlag

Start of frame (SOF) detected.

enumerator kCSI_EndOfFrameFlag

End of frame (EOF) detected.

enumerator kCSI_RxFifoFullFlag

RXFIFO full (Number of data reaches trigger level).

enumerator kCSI_RxBuffer1DmaDoneFlag

RX frame buffer 1 DMA transfer done.

enumerator kCSI_RxBuffer0DmaDoneFlag

RX frame buffer 0 DMA transfer done.

enumerator kCSI_StatFifoFullFlag

STAT FIFO full (Reach trigger level).

enumerator kCSI_StatFifoDmaDoneFlag
STAT FIFO DMA transfer done.

enumerator kCSI_StatFifoOverrunFlag
STAT FIFO overrun.

enumerator kCSI_RxFifoOverrunFlag
RXFIFO overrun.

enumerator kCSI_Field0DoneFlag
Field 0 transfer done.

enumerator kCSI_Field1DoneFlag
Field 1 transfer done.

enumerator kCSI_BaseAddrChangeErrorFlag
The DMA output buffer base address changes before DMA completed.

typedef enum *_csi_work_mode* csi_work_mode_t
CSI work mode.

The CCIR656 interlace mode is not supported currently.

typedef enum *_csi_data_bus* csi_data_bus_t
CSI data bus witdh.

typedef struct *_csi_config* csi_config_t
Configuration to initialize the CSI module.

typedef enum *_csi_fifo* csi_fifo_t
The CSI FIFO, used for FIFO operation.

typedef struct *_csi_handle* csi_handle_t

typedef void (*csi_transfer_callback_t)(CSI_Type *base, *csi_handle_t* *handle, *status_t* status, void *userData)
CSI transfer callback function.

When a new frame is received and saved to the frame buffer queue, the callback is called and the pass the status kStatus_CSI_FrameDone to upper layer.

CSI_REG_CR1(base)

CSI_REG_CR2(base)

CSI_REG_CR3(base)

CSI_REG_CR18(base)

CSI_REG_SR(base)

CSI_REG_DMASA_FB1(base)

CSI_REG_DMASA_FB2(base)

CSI_REG_IMAG_PARA(base)

CSI_REG_FBUF_PARA(base)

CSI_DRIVER_QUEUE_SIZE
Size of the frame buffer queue used in CSI transactional function.

CSI_DRIVER_FRAG_MODE
Enable fragment capture function or not.

CSI_CR1_INT_EN_MASK

CSI_CR3_INT_EN_MASK

CSI_CR18_INT_EN_MASK

struct _csi_config

#include <fsl_csi.h> Configuration to initialize the CSI module.

Public Members

uint16_t width

Pixels of the input frame.

uint16_t height

Lines of the input frame.

uint32_t polarityFlags

Timing signal polarity flags, OR'ed value of _csi_polarity_flags.

uint8_t bytesPerPixel

Bytes per pixel, valid values are:

- 2: Used for RGB565, YUV422, and so on.
- 4: Used for XRGB8888, XYUV444, and so on.

uint16_t linePitch_Bytes

Frame buffer line pitch, must be 8-byte aligned.

csi_work_mode_t workMode

CSI work mode.

csi_data_bus_t dataBus

Data bus width.

bool useExtVsync

In CCIR656 progressive mode, set true to use external VSYNC signal, set false to use internal VSYNC signal decoded from SOF.

struct buf_queue_t

#include <fsl_csi.h>

struct _csi_handle

#include <fsl_csi.h> CSI handle structure.

Please see the user guide for the details of the CSI driver queue mechanism.

Public Members

volatile uint8_t activeBufferNum

How many frame buffers are in progress currently.

volatile uint8_t dmaDoneBufferIdx

Index of the current full-filled framebuffer.

volatile bool transferStarted

User has called CSI_TransferStart to start frame receiving.

csi_transfer_callback_t callback

Callback function.

void *userData
CSI callback function parameter.

1.10 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.

1.11 DCP: Data Co-Processor

FSL_DCP_DRIVER_VERSION
DCP driver version. Version 2.1.7.

Current version: 2.1.7

Change log:

- 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.

1.12 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

1.13 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.

1.14 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.

1.15 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 2.1.1.

1.16 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.

FSL_EDMA_DRIVER_VERSION

eDMA driver version

Version 2.4.5.

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.

1.17 eLCDIF: Enhanced LCD Interface

void ELCDIF_RgbModeInit(LCDIF_Type *base, const *elcdif_rgb_mode_config_t* *config)

Initializes the eLCDIF to work in RGB mode (DOTCLK mode).

This function ungates the eLCDIF clock and configures the eLCDIF peripheral according to the configuration structure.

Parameters

- base – eLCDIF peripheral base address.
- config – Pointer to the configuration structure.

void ELCDIF_RgbModeGetDefaultConfig(*elcdif_rgb_mode_config_t* *config)

Gets the eLCDIF default configuration structure for RGB (DOTCLK) mode.

This function sets the configuration structure to default values. The default configuration is set to the following values.

```
config->panelWidth = 480U;
config->panelHeight = 272U;
config->hsw = 41;
config->hfp = 4;
config->hbp = 8;
config->vsw = 10;
config->vfp = 4;
config->vbp = 2;
config->polarityFlags = kELCDIF_VsyncActiveLow |
                        kELCDIF_HsyncActiveLow |
                        kELCDIF_DataEnableActiveLow |
                        kELCDIF_DriveDataOnFallingClkEdge;
config->bufferAddr = 0U;
config->pixelFormat = kELCDIF_PixelFormatRGB888;
config->dataBus = kELCDIF_DataBus24Bit;
```

Parameters

- config – Pointer to the eLCDIF configuration structure.

void ELCDIF_Deinit(LCDIF_Type *base)

Deinitializes the eLCDIF peripheral.

Parameters

- base – eLCDIF peripheral base address.

void ELCDIF_RgbModeSetPixelFormat(LCDIF_Type *base, *elcdif_pixel_format_t* pixelFormat)

Set the pixel format in RGB (DOTCLK) mode.

Parameters

- base – eLCDIF peripheral base address.
- pixelFormat – The pixel format.

static inline void ELCDIF_RgbModeStart(LCDIF_Type *base)

Start to display in RGB (DOTCLK) mode.

Parameters

- base – eLCDIF peripheral base address.

void ELCDIF_RgbModeStop(LCDIF_Type *base)

Stop display in RGB (DOTCLK) mode and wait until finished.

Parameters

- base – eLCDIF peripheral base address.

static inline void ELCDIF_SetNextBufferAddr(LCDIF_Type *base, uint32_t bufferAddr)

Set the next frame buffer address to display.

Parameters

- base – eLCDIF peripheral base address.
- bufferAddr – The frame buffer address to set.

void ELCDIF_Reset(LCDIF_Type *base)

Reset the eLCDIF peripheral.

Parameters

- base – eLCDIF peripheral base address.

void ELCDIF_SetPixelComponentOrder(LCDIF_Type *base, *elcdif_pixel_component_order_t* order)

Set the order of the RGB components of each pixel in lines.

Parameters

- base – eLCDIF peripheral base address.
- order – The pixel component order

static inline uint32_t ELCDIF_GetCrcValue(const LCDIF_Type *base)

Get the CRC value of the frame sent out.

When a frame is sent complete (the interrupt kELCDIF_CurFrameDone assert), this function can be used to get the CRC value of the frame sent.

Note: The CRC value is dependent on the LCD_DATABUS_WIDTH.

Parameters

- base – eLCDIF peripheral base address.

Returns

The CRC value.

static inline uint32_t ELCDIF_GetBusMasterErrorAddr(const LCDIF_Type *base)

Get the bus master error virtual address.

When bus master error occurs (the interrupt kELCDIF_BusMasterError assert), this function can get the virtual address at which the AXI master received an error response from the slave.

Parameters

- base – eLCDIF peripheral base address.

Returns

The error virtual address.

static inline uint32_t ELCDIF_GetStatus(const LCDIF_Type *base)

Get the eLCDIF status.

The status flags are returned as a mask value, application could check the corresponding bit. Example:

```
uint32_t statusFlags;
statusFlags = ELCDIF_GetStatus(LCDIF);

if (kELCDIF_LFifoFull & statusFlags)
{
}

if (kELCDIF_TxFifoEmpty & statusFlags)
{
}
```

Parameters

- base – eLCDIF peripheral base address.

Returns

The mask value of status flags, it is OR'ed value of _elcdif_status_flags.

static inline uint32_t ELCDIF_GetLFifoCount(const LCDIF_Type *base)

Get current count in Latency buffer (LFIFO).

Parameters

- base – eLCDIF peripheral base address.

Returns

The LFIFO current count

static inline void ELCDIF_EnableInterrupts(LCDIF_Type *base, uint32_t mask)

Enables eLCDIF interrupt requests.

Parameters

- base – eLCDIF peripheral base address.
- mask – interrupt source, OR'ed value of _elcdif_interrupt_enable.

static inline void ELCDIF_DisableInterrupts(LCDIF_Type *base, uint32_t mask)

Disables eLCDIF interrupt requests.

Parameters

- base – eLCDIF peripheral base address.
- mask – interrupt source, OR'ed value of _elcdif_interrupt_enable.

static inline uint32_t ELCDIF_GetInterruptStatus(const LCDIF_Type *base)

Get eLCDIF interrupt peding status.

Parameters

- base – eLCDIF peripheral base address.

Returns

Interrupt pending status, OR'ed value of _elcdif_interrupt_flags.

static inline void ELCDIF_ClearInterruptStatus(LCDIF_Type *base, uint32_t mask)

Clear eLCDIF interrupt peding status.

Parameters

- base – eLCDIF peripheral base address.
- mask – of the flags to clear, OR'ed value of _elcdif_interrupt_flags.

static inline void ELCDIF_EnableLut(LCDIF_Type *base, bool enable)

Enable or disable the LUT.

Parameters

- base – eLCDIF peripheral base address.
- enable – True to enable, false to disable.

status_t ELCDIF_UpdateLut(LCDIF_Type *base, *elcdif_lut_t* lut, uint16_t startIndex, const uint32_t *lutData, uint16_t count)

Load the LUT value.

This function loads the LUT value to the specific LUT memory, user can specify the start entry index.

Parameters

- base – eLCDIF peripheral base address.
- lut – Which LUT to load.
- startIndex – The start index of the LUT entry to update.
- lutData – The LUT data to load.
- count – Count of lutData.

Return values

- kStatus_Success – Initialization success.
- kStatus_InvalidArgument – Wrong argument.

FSL_ELCDIF_DRIVER_VERSION

eLCDIF driver version

enum _elcdif_polarity_flags

eLCDIF signal polarity flags

Values:

enumerator kELCDIF_VsyncActiveLow

VSYNC active low.

enumerator kELCDIF_HsyncActiveLow

HSYNC active low.

enumerator kELCDIF_DataEnableActiveLow

Data enable line active low.

enumerator kELCDIF_DriveDataOnFallingClkEdge

Drive data on falling clock edge, capture data on rising clock edge.

enumerator kELCDIF_VsyncActiveHigh

VSYNC active high.

enumerator kELCDIF_HsyncActiveHigh

HSYNC active high.

enumerator kELCDIF_DataEnableActiveHigh

Data enable line active high.

enumerator kELCDIF_DriveDataOnRisingClkEdge

Drive data on falling clock edge, capture data on rising clock edge.

enum _elcdif_interrupt_enable

The eLCDIF interrupts to enable.

Values:

enumerator kELCDIF_BusMasterErrorInterruptEnable

Bus master error interrupt.

enumerator kELCDIF_TxFifoOverflowInterruptEnable

TXFIFO overflow interrupt.

enumerator kELCDIF_TxFifoUnderflowInterruptEnable

TXFIFO underflow interrupt.

enumerator kELCDIF_CurFrameDoneInterruptEnable

Interrupt when hardware enters vertical blanking state.

enumerator kELCDIF_VsyncEdgeInterruptEnable

Interrupt when hardware encounters VSYNC edge.

enum _elcdif_interrupt_flags

The eLCDIF interrupt status flags.

Values:

enumerator kELCDIF_BusMasterError

Bus master error interrupt.

enumerator kELCDIF_TxFifoOverflow

TXFIFO overflow interrupt.

enumerator kELCDIF_TxFifoUnderflow

TXFIFO underflow interrupt.

enumerator kELCDIF_CurFrameDone

Interrupt when hardware enters vertical blanking state.

enumerator kELCDIF_VsyncEdge

Interrupt when hardware encounters VSYNC edge.

enum _elcdif_status_flags

eLCDIF status flags

Values:

enumerator kELCDIF_LFifoFull

LFIFO full.

enumerator kELCDIF_LFifoEmpty

LFIFO empty.

enumerator kELCDIF_TxFifoFull

TXFIFO full.

enumerator kELCDIF_TxFifoEmpty

TXFIFO empty.

enum _elcdif_pixel_format

The pixel format.

This enumerator should be defined together with the array s_pixelFormatReg. To support new pixel format, enhance this enumerator and s_pixelFormatReg.

Values:

enumerator kELCDIF_PixelFormatRAW8
RAW 8 bit, four data use 32 bits.

enumerator kELCDIF_PixelFormatRGB565
RGB565, two pixel use 32 bits.

enumerator kELCDIF_PixelFormatRGB666
RGB666 unpacked, one pixel uses 32 bits, high byte unused, upper 2 bits of other bytes unused.

enumerator kELCDIF_PixelFormatXRGB8888
XRGB8888 unpacked, one pixel uses 32 bits, high byte unused.

enumerator kELCDIF_PixelFormatRGB888
RGB888 packed, one pixel uses 24 bits.

enum _elcdif_lcd_data_bus
The LCD data bus type.

Values:

enumerator kELCDIF_DataBus8Bit
8-bit data bus.

enumerator kELCDIF_DataBus16Bit
16-bit data bus, support RGB565.

enumerator kELCDIF_DataBus18Bit
18-bit data bus, support RGB666.

enumerator kELCDIF_DataBus24Bit
24-bit data bus, support RGB888.

enum _elcdif_as_pixel_format
eLCDIF alpha surface pixel format.

Values:

enumerator kELCDIF_AsPixelFormatARGB8888
32-bit pixels with alpha.

enumerator kELCDIF_AsPixelFormatRGB888
32-bit pixels without alpha (unpacked 24-bit format)

enumerator kELCDIF_AsPixelFormatARGB1555
16-bit pixels with alpha.

enumerator kELCDIF_AsPixelFormatARGB4444
16-bit pixels with alpha.

enumerator kELCDIF_AsPixelFormatRGB555
16-bit pixels without alpha.

enumerator kELCDIF_AsPixelFormatRGB444
16-bit pixels without alpha.

enumerator kELCDIF_AsPixelFormatRGB565
16-bit pixels without alpha.

enum _elcdif_alpha_mode
eLCDIF alpha mode during blending.

Values:

enumerator kELCDIF_AlphaEmbedded
The alpha surface pixel alpha value will be used for blend.

enumerator kELCDIF_AlphaOverride
The user defined alpha value will be used for blend directly.

enumerator kELCDIF_AlphaMultiply
The alpha surface pixel alpha value scaled the user defined alpha value will be used for blend, for example, pixel alpha set set to 200, user defined alpha set to 100, then the reault alpha is 200 * 100 / 255.

enumerator kELCDIF_AlphaRop
Raster operation.

enum _elcdif_rop_mode
eLCDIF ROP mode during blending.

Explanation:

- AS: Alpha surface
- PS: Process surface
- nAS: Alpha surface NOT value
- nPS: Process surface NOT value

Values:

enumerator kELCDIF_RopMaskAs
AS AND PS.

enumerator kELCDIF_RopMaskNotAs
nAS AND PS.

enumerator kELCDIF_RopMaskAsNot
AS AND nPS.

enumerator kELCDIF_RopMergeAs
AS OR PS.

enumerator kELCDIF_RopMergeNotAs
nAS OR PS.

enumerator kELCDIF_RopMergeAsNot
AS OR nPS.

enumerator kELCDIF_RopNotCopyAs
nAS.

enumerator kELCDIF_RopNot
nPS.

enumerator kELCDIF_RopNotMaskAs
AS NAND PS.

enumerator kELCDIF_RopNotMergeAs
AS NOR PS.

enumerator kELCDIF_RopXorAs
AS XOR PS.

enumerator kELCDIF_RopNotXorAs
AS XNOR PS.

enum _elcdif_lut

eLCDIF LUT

The Lookup Table (LUT) is used to expand the 8 bits pixel to 24 bits pixel before output to external displayer.

There are two 256x24 bits LUT memory in LCDIF, the LSB of frame buffer address determins which memory to use.

Values:

enumerator kELCDIF_Lut0

LUT 0.

enumerator kELCDIF_Lut1

LUT 1.

enum _elcdif_pixel_component_order

eLCDIF pixel component order.

Values:

enumerator kELCDIF_PixelComponentOrderRGB

Input order RGB.

enumerator kELCDIF_PixelComponentOrderRBG

Input order RBG.

enumerator kELCDIF_PixelComponentOrderGBR

Input order GBR.

enumerator kELCDIF_PixelComponentOrderGRB

Input order GRB.

enumerator kELCDIF_PixelComponentOrderBRG

Input order BRG.

enumerator kELCDIF_PixelComponentOrderBGR

Input order BGR.

typedef enum *_elcdif_pixel_format* elcdif_pixel_format_t

The pixel format.

This enumerator should be defined together with the array s_pixelFormatReg. To support new pixel format, enhance this enumerator and s_pixelFormatReg.

typedef enum *_elcdif_lcd_data_bus* elcdif_lcd_data_bus_t

The LCD data bus type.

typedef struct *_elcdif_pixel_format_reg* elcdif_pixel_format_reg_t

The register value when using different pixel format.

These register bits control the pixel format:

- CTRL[DATA_FORMAT_24_BIT]
- CTRL[DATA_FORMAT_18_BIT]
- CTRL[DATA_FORMAT_16_BIT]
- CTRL[WORD_LENGTH]
- CTRL1[BYTE_PACKING_FORMAT]

typedef struct *_elcdif_rgb_mode_config* elcdif_rgb_mode_config_t

eLCDIF configure structure for RGB mode (DOTCLK mode).

typedef enum *_elcdif_as_pixel_format* elcdif_as_pixel_format_t

eLCDIF alpha surface pixel format.

typedef struct *_elcdif_as_buffer_config* elcdif_as_buffer_config_t

eLCDIF alpha surface buffer configuration.

typedef enum *_elcdif_alpha_mode* elcdif_alpha_mode_t

eLCDIF alpha mode during blending.

typedef enum *_elcdif_rop_mode* elcdif_rop_mode_t

eLCDIF ROP mode during blending.

Explanation:

- AS: Alpha surface
- PS: Process surface
- nAS: Alpha surface NOT value
- nPS: Process surface NOT value

typedef struct *_elcdif_as_blend_config* elcdif_as_blend_config_t

eLCDIF alpha surface blending configuration.

typedef enum *_elcdif_lut* elcdif_lut_t

eLCDIF LUT

The Lookup Table (LUT) is used to expand the 8 bits pixel to 24 bits pixel before output to external displayer.

There are two 256x24 bits LUT memory in LCDIF, the LSB of frame buffer address determins which memory to use.

typedef enum *_elcdif_pixel_component_order* elcdif_pixel_component_order_t

eLCDIF pixel component order.

ELCDIF_CTRL1_IRQ_MASK

ELCDIF_CTRL1_IRQ_EN_MASK

ELCDIF_AS_CTRL_IRQ_MASK

ELCDIF_AS_CTRL_IRQ_EN_MASK

FSL_FEATURE_LCDIF_HAS_PXP_HANDSHAKE

ELCDIF_ADDR_CPU_2_IP(addr)

ELCDIF_LUT_ENTRY_NUM

struct _elcdif_pixel_format_reg

#include <fsl_elcdif.h> The register value when using different pixel format.

These register bits control the pixel format:

- CTRL[DATA_FORMAT_24_BIT]
- CTRL[DATA_FORMAT_18_BIT]
- CTRL[DATA_FORMAT_16_BIT]
- CTRL[WORD_LENGTH]
- CTRL1[BYTE_PACKING_FORMAT]

Public Members

uint32_t regCtrl
: Value of register CTRL.

uint32_t regCtrl1
: Value of register CTRL1.

struct _elcdif_rgb_mode_config
: *#include <fsl_elcdif.h>* eLCDIF configure structure for RGB mode (DOTCLK mode).

Public Members

uint16_t panelWidth
: Display panel width, pixels per line.

uint16_t panelHeight
: Display panel height, how many lines per panel.

uint8_t hsw
: HSYNC pulse width.

uint8_t hfp
: Horizontal front porch.

uint8_t hbp
: Horizontal back porch.

uint8_t vsw
: VSYNC pulse width.

uint8_t vfp
: Vrtical front porch.

uint8_t vbp
: Vertical back porch.

uint32_t polarityFlags
: OR'ed value of _elcdif_polarity_flags, used to contol the signal polarity.

uint32_t bufferAddr
: Frame buffer address.

elcdif_pixel_format_t pixelFormat
: Pixel format.

elcdif_lcd_data_bus_t dataBus
: LCD data bus.

struct _elcdif_as_buffer_config
: *#include <fsl_elcdif.h>* eLCDIF alpha surface buffer configuration.

Public Members

uint32_t bufferAddr
: Buffer address.

elcdif_as_pixel_format_t pixelFormat
: Pixel format.

struct _elcdif_as_blend_config
: *#include <fsl_elcdif.h>* eLCDIF alpha surface blending configuration.

Public Members

uint8_t alpha

User defined alpha value, only used when alphaMode is kELCDIF_AlphaOverride or kELCDIF_AlphaRop.

bool invertAlpha

Set true to invert the alpha.

elcdif_alpha_mode_t alphaMode

Alpha mode.

elcdif_rop_mode_t ropMode

ROP mode, only valid when alphaMode is kELCDIF_AlphaRop.

1.18 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_
```

(continues on next page)

(continued from previous page)

```
↪POSMATCHOnPositionCounterEqualToComapreValue;
config->positionCompareValue                = 0xFFFFFFFFU;
config->revolutionCountCondition             = kENC_RevolutionCountOnINDEXPulse;
config->enableModuloCountMode                  = false;
config->positionModulusValue                = 0U;
config->positionInitialValue               = 0U;
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.

1.19 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
    }
}
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.

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.

1.20 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

1.21 FlexCAN: Flex Controller Area Network Driver

1.22 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.

void FLEXCAN_EnterFreezeMode(CAN_Type *base)
Enter FlexCAN Freeze Mode.

This function makes the FlexCAN work under Freeze Mode.

Parameters

- base – FlexCAN peripheral base address.

void FLEXCAN_ExitFreezeMode(CAN_Type *base)
Exit FlexCAN Freeze Mode.

This function makes the FlexCAN leave Freeze Mode.

Parameters

- base – FlexCAN peripheral base address.

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.

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.

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 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.

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_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.

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 void 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.

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.

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.

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.

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.

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.

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.

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.

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_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 Legacy Rx Fifo transfer status during a interrupt non-blocking receive.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- count – Number of CAN messages receive so far by the non-blocking transaction.

Return values

- kStatus_InvalidArgument – count is Invalid.
- kStatus_Success – Successfully return the count.

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.

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_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_WakeUpInterruptEnable
Self Wake Up interrupt, use bit 26.

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_WakeUpIntFlag
Self Wake-Up 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_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

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_MEMORY_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.

struct _flexcan_timing_config

#include <fsl_flexcan.h> FlexCAN protocol timing characteristic configuration structure.

Public Members

uint16_t preDivider
: Classic CAN or CAN FD nominal phase bit rate prescaler.

uint8_t rJumpwidth
: Classic CAN or CAN FD nominal phase Re-sync Jump Width.

uint8_t phaseSeg1
: Classic CAN or CAN FD nominal phase Segment 1.

uint8_t phaseSeg2
: Classic CAN or CAN FD nominal phase Segment 2.

uint8_t propSeg
: Classic CAN or CAN FD nominal phase Propagation Segment.

uint16_t fpreDivider
: CAN FD data phase bit rate prescaler.

uint8_t frJumpwidth
: CAN FD data phase Re-sync Jump Width.

uint8_t fphaseSeg1
: CAN FD data phase Phase Segment 1.

uint8_t fphaseSeg2
: CAN FD data phase Phase Segment 2.

uint8_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 enableSelfWakeup
: Enable or Disable Self Wakeup Mode.

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 enableSupervisorMode

Enable or Disable Supervisor Mode, enable this mode will make registers allow only Supervisor access.

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.

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).

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

Number of CAN Message need to be received from Legacy or Ehanced 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 Ehanced Rx FIFO.

size_t rxFifoTransferTotalNum

Total CAN Message number need to be received from Legacy or Ehanced 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 ___unnamed20___

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 ___unnamed22___

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 ___unnamed24___

Public Members

struct _flexcan_frame

struct _flexcan_frame

struct ___unnamed26___

Public Members

uint32_t dataWord0

CAN Frame payload word0.

uint32_t dataWord1

CAN Frame payload word1.

struct ___unnamed28___

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 ___unnamed30___

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 or REMOTE).

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 ___unnamed32___

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 ___unnamed34___

Public Members

struct __flexcan__fd__frame

struct __flexcan__fd__frame

struct ___unnamed36___

Public Members

uint32_t dataWord[16]
: CAN FD Frame payload, 16 double word maximum.

struct ___unnamed38___

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 ___unnamed40___

Public Members

struct _flexcan_config

struct _flexcan_config

struct ___unnamed42___

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 ___unnamed44___

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 ___unnamed46___

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 ___unnamed50___

< 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 ___unnamed52___

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 ___unnamed48___

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 ___unnamed54___

< 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 ___unnamed56___

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.

1.23 FlexCAN eDMA Driver

void FLEXCAN_TransferCreateHandleEDMA(CAN_Type *base, *flexcan_edma_handle_t* *handle, *flexcan_edma_transfer_callback_t* callback, void *userData, *edma_handle_t* *rxFifoEdmaHandle)

Initializes the FlexCAN handle, which is used in transactional functions.

Parameters

- base – FlexCAN peripheral base address.
- handle – Pointer to flexcan_edma_handle_t structure.
- callback – The callback function.
- userData – The parameter of the callback function.
- rxFifoEdmaHandle – User-requested DMA handle for Rx FIFO DMA transfer.

void FLEXCAN_PrepareTransfConfiguration(CAN_Type *base, *flexcan_fifo_transfer_t* *pFifoXfer, *edma_transfer_config_t* *pEdmaConfig)

Prepares the eDMA transfer configuration for FLEXCAN Legacy RX FIFO.

This function prepares the eDMA transfer configuration structure according to FLEXCAN Legacy RX FIFO.

Parameters

- base – FlexCAN peripheral base address.
- pFifoXfer – FlexCAN Rx FIFO EDMA transfer structure, see flexcan_fifo_transfer_t.
- pEdmaConfig – The user configuration structure of type edma_transfer_t.

status_t FLEXCAN_StartTransferDatafromRxFIFO(CAN_Type *base, *flexcan_edma_handle_t* *handle, *edma_transfer_config_t* *pEdmaConfig)

Start Transfer Data from the FLEXCAN Legacy Rx FIFO using eDMA.

This function to Update edma transfer confiugration and Start eDMA transfer

Parameters

- base – FlexCAN peripheral base address.
- handle – Pointer to flexcan_edma_handle_t structure.
- pEdmaConfig – The user configuration structure of type edma_transfer_t.

Return values

- kStatus_Success – if succeed, others failed.
- kStatus_FLEXCAN_RxFifoBusy – Previous transfer ongoing.

status_t FLEXCAN_TransferReceiveFifoEDMA(CAN_Type *base, *flexcan_edma_handle_t* *handle, *flexcan_fifo_transfer_t* *pFifoXfer)

Receives the CAN Message from the Legacy Rx FIFO using eDMA.

This function receives the CAN Message using eDMA. This is a non-blocking function, which returns right away. After the CAN Message is received, the receive callback function is called.

Parameters

- base – FlexCAN peripheral base address.
- handle – Pointer to flexcan_edma_handle_t structure.
- pFifoXfer – FlexCAN Rx FIFO EDMA transfer structure, see flexcan_fifo_transfer_t.

Return values

- kStatus_Success – if succeed, others failed.
- kStatus_FLEXCAN_RxFifoBusy – Previous transfer ongoing.

status_t FLEXCAN_TransferGetReceiveFifoCountEMDA(CAN_Type *base, *flexcan_edma_handle_t* *handle, size_t *count)

Gets the Legacy Rx Fifo transfer status during a interrupt non-blocking receive.

Parameters

- base – FlexCAN peripheral base address.
- handle – FlexCAN handle pointer.
- count – Number of CAN messages receive so far by the non-blocking transaction.

Return values

- kStatus_InvalidArgument – count is Invalid.
- kStatus_Success – Successfully return the count.

void FLEXCAN_TransferAbortReceiveFifoEDMA(CAN_Type *base, *flexcan_edma_handle_t* *handle)

Aborts the receive Legacy/Enhanced Rx FIFO process which used eDMA.

This function aborts the receive Legacy/Enhanced Rx FIFO process which used eDMA.

Parameters

- base – FlexCAN peripheral base address.
- handle – Pointer to flexcan_edma_handle_t structure.

FSL_FLEXCAN_EDMA_DRIVER_VERSION

FlexCAN EDMA driver version.

typedef struct *_flexcan_edma_handle* flexcan_edma_handle_t

typedef void (*flexcan_edma_transfer_callback_t)(CAN_Type *base, *flexcan_edma_handle_t* *handle, *status_t* status, void *userData)

FlexCAN transfer callback function.

struct _flexcan_edma_handle

#include <fsl_flexcan_edma.h> FlexCAN eDMA handle.

Public Members

flexcan_edma_transfer_callback_t callback

Callback function.

void *userData

FlexCAN callback function parameter.

edma_handle_t *rxFifoEdmaHandle

The EDMA handler for Rx FIFO.

volatile uint8_t rxFifoState

Rx FIFO transfer state.

size_t frameNum

The number of messages that need to be received.

1.24 FlexIO: FlexIO Driver

1.25 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

1.26 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

1.27 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

1.28 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

1.29 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,
```

(continues on next page)

(continued from previous page)

```
.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.

1.30 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

1.31 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,
```

(continues on next page)

(continued from previous page)

```
.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.

1.32 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,
```

(continues on next page)

(continued from previous page)

```
.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 ___unnamed115___

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.

1.33 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

1.34 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.

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_PortB1
Access flash on B1 port.

enumerator kFLEXSPI_PortB2
Access flash on B2 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_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_config
#include <fsl_flexspi.h> FLEXSPI configuration structure.

Public Members

flexspi_read_sample_clock_t rxSampleClock
Sample Clock source selection for Flash Reading.

bool enableSckFreeRunning
Enable/disable SCK output free-running.

bool enableCombination
Enable/disable combining PORT A and B Data Pins (SIOA[3:0] and SIOB[3:0]) to support Flash Octal mode.

bool enableDoze
Enable/disable doze mode support.

bool enableHalfSpeedAccess
Enable/disable divide by 2 of the clock for half speed commands.

bool enableSckBDiffOpt

Enable/disable SCKB pad use as SCKA differential clock output, when enable, Port B flash access is not available.

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.

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.

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.

1.35 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

1.36 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.

1.37 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.

1.38 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;
```

(continues on next page)

(continued from previous page)

```
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.

1.39 IOMUXC: IOMUX Controller

enum _iomuxc_gpr_mode
Values:

enumerator kIOMUXC_GPR_GlobalInterruptRequest

enumerator kIOMUXC_GPR_ENET1RefClkMode

enumerator kIOMUXC_GPR_ENET2RefClkMode

enumerator kIOMUXC_GPR_ENET1TxClkOutputDir

enumerator kIOMUXC_GPR_ENET2TxClkOutputDir

enumerator kIOMUXC_GPR_SAI1MClkOutputDir

enumerator kIOMUXC_GPR_SAI2MClkOutputDir

enumerator kIOMUXC_GPR_SAI3MClkOutputDir

enumerator kIOMUXC_GPR_ExcMonitorSlavErrResponse

enumerator kIOMUXC_GPR_ENETIpgClkOn

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_FLEXPWM4_PWMA00

IOMUXC_GPIO_EMC_00_LPSPI2_SCK

IOMUXC_GPIO_EMC_00_XBAR1_XBAR_IN02

IOMUXC_GPIO_EMC_00_FLEXIO1_FLEXIO00

IOMUXC_GPIO_EMC_00_GPIO4_IO00

IOMUXC_GPIO_EMC_01_SEMC_DATA01

IOMUXC_GPIO_EMC_01_FLEXPWM4_PWMB00

IOMUXC_GPIO_EMC_01_LPSPI2_PCS0

IOMUXC_GPIO_EMC_01_XBAR1_IN03

IOMUXC_GPIO_EMC_01_FLEXIO1_FLEXIO01

IOMUXC_GPIO_EMC_01_GPIO4_IO01

IOMUXC_GPIO_EMC_02_SEMC_DATA02

IOMUXC_GPIO_EMC_02_FLEXPWM4_PWMA01

IOMUXC_GPIO_EMC_02_LPSPI2_SDO

IOMUXC_GPIO_EMC_02_XBAR1_INOUT04

IOMUXC_GPIO_EMC_02_FLEXIO1_FLEXIO02

IOMUXC_GPIO_EMC_02_GPIO4_IO02

IOMUXC_GPIO_EMC_03_SEMC_DATA03

IOMUXC_GPIO_EMC_03_FLEXPWM4_PWMB01

IOMUXC_GPIO_EMC_03_LPSPI2_SDI

IOMUXC_GPIO_EMC_03_XBAR1_INOUT05

IOMUXC_GPIO_EMC_03_FLEXIO1_FLEXIO03

IOMUXC_GPIO_EMC_03_GPIO4_IO03

IOMUXC_GPIO_EMC_04_SEMC_DATA04

IOMUXC_GPIO_EMC_04_FLEXPWM4_PWMA02

IOMUXC_GPIO_EMC_04_SAI2_TX_DATA

IOMUXC_GPIO_EMC_04_XBAR1_INOUT06

IOMUXC_GPIO_EMC_04_FLEXIO1_FLEXIO04

IOMUXC_GPIO_EMC_04_GPIO4_IO04

IOMUXC_GPIO_EMC_05_SEMC_DATA05

IOMUXC_GPIO_EMC_05_FLEXPWM4_PWMB02

IOMUXC_GPIO_EMC_05_SAI2_TX_SYNC

IOMUXC_GPIO_EMC_05_XBAR1_INOUT07

IOMUXC_GPIO_EMC_05_FLEXIO1_FLEXIO05

IOMUXC_GPIO_EMC_05_GPIO4_IO05

IOMUXC_GPIO_EMC_06_SEMC_DATA06

IOMUXC_GPIO_EMC_06_FLEXPWM2_PWMA00

IOMUXC_GPIO_EMC_06_SAI2_TX_BCLK

IOMUXC_GPIO_EMC_06_XBAR1_INOUT08

IOMUXC_GPIO_EMC_06_FLEXIO1_FLEXIO06

IOMUXC_GPIO_EMC_06_GPIO4_IO06

IOMUXC_GPIO_EMC_07_SEMC_DATA07

IOMUXC_GPIO_EMC_07_FLEXPWM2_PWMB00

IOMUXC_GPIO_EMC_07_SAI2_MCLK

IOMUXC_GPIO_EMC_07_XBAR1_INOUT09

IOMUXC_GPIO_EMC_07_FLEXIO1_FLEXIO07

IOMUXC_GPIO_EMC_07_GPIO4_IO07

IOMUXC_GPIO_EMC_08_SEMC_DM00

IOMUXC_GPIO_EMC_08_FLEXPWM2_PWMA01

IOMUXC_GPIO_EMC_08_SAI2_RX_DATA

IOMUXC_GPIO_EMC_08_XBAR1_INOUT17

IOMUXC_GPIO_EMC_08_FLEXIO1_FLEXIO08

IOMUXC_GPIO_EMC_08_GPIO4_IO08

IOMUXC_GPIO_EMC_09_SEMC_ADDR00

IOMUXC_GPIO_EMC_09_FLEXPWM2_PWMB01

IOMUXC_GPIO_EMC_09_SAI2_RX_SYNC

IOMUXC_GPIO_EMC_09_FLEXCAN2_TX

IOMUXC_GPIO_EMC_09_FLEXIO1_FLEXIO09

IOMUXC_GPIO_EMC_09_GPIO4_IO09

IOMUXC_GPIO_EMC_09_FLEXSPI2_B_SS1_B

IOMUXC_GPIO_EMC_10_SEMC_ADDR01

IOMUXC_GPIO_EMC_10_FLEXPWM2_PWMA02

IOMUXC_GPIO_EMC_10_SAI2_RX_BCLK

IOMUXC_GPIO_EMC_10_FLEXCAN2_RX

IOMUXC_GPIO_EMC_10_FLEXIO1_FLEXIO10

IOMUXC_GPIO_EMC_10_GPIO4_IO10

IOMUXC_GPIO_EMC_10_FLEXSPI2_B_SS0_B

IOMUXC_GPIO_EMC_11_SEMC_ADDR02

IOMUXC_GPIO_EMC_11_FLEXPWM2_PWMB02

IOMUXC_GPIO_EMC_11_LPI2C4_SDA

IOMUXC_GPIO_EMC_11_USDHC2_RESET_B

IOMUXC_GPIO_EMC_11_FLEXIO1_FLEXIO11

IOMUXC_GPIO_EMC_11_GPIO4_IO11

IOMUXC_GPIO_EMC_11_FLEXSPI2_B_DQS

IOMUXC_GPIO_EMC_12_SEMC_ADDR03

IOMUXC_GPIO_EMC_12_XBAR1_IN24

IOMUXC_GPIO_EMC_12_LPI2C4_SCL

IOMUXC_GPIO_EMC_12_USDHC1_WP

IOMUXC_GPIO_EMC_12_FLEXPWM1_PWMA03

IOMUXC_GPIO_EMC_12_GPIO4_IO12

IOMUXC_GPIO_EMC_12_FLEXSPI2_B_SCLK

IOMUXC_GPIO_EMC_13_SEMC_ADDR04

IOMUXC_GPIO_EMC_13_XBAR1_IN25

IOMUXC_GPIO_EMC_13_LPUART3_TX

IOMUXC_GPIO_EMC_13_MQS_RIGHT

IOMUXC_GPIO_EMC_13_FLEXPWM1_PWMB03

IOMUXC_GPIO_EMC_13_GPIO4_IO13

IOMUXC_GPIO_EMC_13_FLEXSPI2_B_DATA00

IOMUXC_GPIO_EMC_14_SEMC_ADDR05

IOMUXC_GPIO_EMC_14_XBAR1_INOUT19

IOMUXC_GPIO_EMC_14_LPUART3_RX

IOMUXC_GPIO_EMC_14_MQS_LEFT

IOMUXC_GPIO_EMC_14_LPSPI2_PCS1

IOMUXC_GPIO_EMC_14_GPIO4_IO14

IOMUXC_GPIO_EMC_14_FLEXSPI2_B_DATA01

IOMUXC_GPIO_EMC_15_SEMC_ADDR06

IOMUXC_GPIO_EMC_15_XBAR1_IN20

IOMUXC_GPIO_EMC_15_LPUART3_CTS_B

IOMUXC_GPIO_EMC_15_SPDIF_OUT

IOMUXC_GPIO_EMC_15_QTIMER3_TIMER0

IOMUXC_GPIO_EMC_15_GPIO4_IO15

IOMUXC_GPIO_EMC_15_FLEXSPI2_B_DATA02

IOMUXC_GPIO_EMC_16_SEMC_ADDR07

IOMUXC_GPIO_EMC_16_XBAR1_IN21

IOMUXC_GPIO_EMC_16_LPUART3_RTS_B

IOMUXC_GPIO_EMC_16_SPDIF_IN

IOMUXC_GPIO_EMC_16_QTIMER3_TIMER1

IOMUXC_GPIO_EMC_16_GPIO4_IO16

IOMUXC_GPIO_EMC_16_FLEXSPI2_B_DATA03

IOMUXC_GPIO_EMC_17_SEMC_ADDR08

IOMUXC_GPIO_EMC_17_FLEXPWM4_PWMA03

IOMUXC_GPIO_EMC_17_LPUART4_CTS_B

IOMUXC_GPIO_EMC_17_FLEXCAN1_TX

IOMUXC_GPIO_EMC_17_QTIMER3_TIMER2

IOMUXC_GPIO_EMC_17_GPIO4_IO17

IOMUXC_GPIO_EMC_18_SEMC_ADDR09

IOMUXC_GPIO_EMC_18_FLEXPWM4_PWMB03

IOMUXC_GPIO_EMC_18_LPUART4_RTS_B

IOMUXC_GPIO_EMC_18_FLEXCAN1_RX

IOMUXC_GPIO_EMC_18_QTIMER3_TIMER3

IOMUXC_GPIO_EMC_18_GPIO4_IO18

IOMUXC_GPIO_EMC_18_SNVS_VIO_5_CTL

IOMUXC_GPIO_EMC_19_SEMC_ADDR11

IOMUXC_GPIO_EMC_19_FLEXPWM2_PWMA03

IOMUXC_GPIO_EMC_19_LPUART4_TX

IOMUXC_GPIO_EMC_19_ENET_RDATA01

IOMUXC_GPIO_EMC_19_QTIMER2_TIMER0

IOMUXC_GPIO_EMC_19_GPIO4_IO19

IOMUXC_GPIO_EMC_19_SNVS_VIO_5

IOMUXC_GPIO_EMC_20_SEMC_ADDR12

IOMUXC_GPIO_EMC_20_FLEXPWM2_PWMB03

IOMUXC_GPIO_EMC_20_LPUART4_RX

IOMUXC_GPIO_EMC_20_ENET_RDATA00

IOMUXC_GPIO_EMC_20_QTIMER2_TIMER1

IOMUXC_GPIO_EMC_20_GPIO4_IO20

IOMUXC_GPIO_EMC_21_SEMC_BA0

IOMUXC_GPIO_EMC_21_FLEXPWM3_PWMA03

IOMUXC_GPIO_EMC_21_LPI2C3_SDA

IOMUXC_GPIO_EMC_21_ENET_TDATA01

IOMUXC_GPIO_EMC_21_QTIMER2_TIMER2

IOMUXC_GPIO_EMC_21_GPIO4_IO21

IOMUXC_GPIO_EMC_22_SEMC_BA1

IOMUXC_GPIO_EMC_22_FLEXPWM3_PWMB03

IOMUXC_GPIO_EMC_22_LPI2C3_SCL

IOMUXC_GPIO_EMC_22_ENET_TDATA00

IOMUXC_GPIO_EMC_22_QTIMER2_TIMER3

IOMUXC_GPIO_EMC_22_GPIO4_IO22

IOMUXC_GPIO_EMC_22_FLEXSPI2_A_SS1_B

IOMUXC_GPIO_EMC_23_SEMC_ADDR10

IOMUXC_GPIO_EMC_23_FLEXPWM1_PWMA00

IOMUXC_GPIO_EMC_23_LPUART5_TX

IOMUXC_GPIO_EMC_23_ENET_RX_EN

IOMUXC_GPIO_EMC_23_GPT1_CAPTURE2

IOMUXC_GPIO_EMC_23_GPIO4_IO23

IOMUXC_GPIO_EMC_23_FLEXSPI2_A_DQS

IOMUXC_GPIO_EMC_24_SEMC_CAS

IOMUXC_GPIO_EMC_24_FLEXPWM1_PWMB00

IOMUXC_GPIO_EMC_24_LPUART5_RX

IOMUXC_GPIO_EMC_24_ENET_TX_EN

IOMUXC_GPIO_EMC_24_GPT1_CAPTURE1

IOMUXC_GPIO_EMC_24_GPIO4_IO24

IOMUXC_GPIO_EMC_24_FLEXSPI2_A_SS0_B

IOMUXC_GPIO_EMC_25_SEMC_RAS

IOMUXC_GPIO_EMC_25_FLEXPWM1_PWMA01

IOMUXC_GPIO_EMC_25_LPUART6_TX

IOMUXC_GPIO_EMC_25_ENET_TX_CLK

IOMUXC_GPIO_EMC_25_ENET_REF_CLK

IOMUXC_GPIO_EMC_25_GPIO4_IO25

IOMUXC_GPIO_EMC_25_FLEXSPI2_A_SCLK

IOMUXC_GPIO_EMC_26_SEMC_CLK

IOMUXC_GPIO_EMC_26_FLEXPWM1_PWMB01

IOMUXC_GPIO_EMC_26_LPUART6_RX

IOMUXC_GPIO_EMC_26_ENET_RX_ER

IOMUXC_GPIO_EMC_26_FLEXIO1_FLEXIO12

IOMUXC_GPIO_EMC_26_GPIO4_IO26

IOMUXC_GPIO_EMC_26_FLEXSPI2_A_DATA00

IOMUXC_GPIO_EMC_27_SEMC_CKE

IOMUXC_GPIO_EMC_27_FLEXPWM1_PWMA02

IOMUXC_GPIO_EMC_27_LPUART5_RTS_B

IOMUXC_GPIO_EMC_27_LPSPI1_SCK

IOMUXC_GPIO_EMC_27_FLEXIO1_FLEXIO13

IOMUXC_GPIO_EMC_27_GPIO4_IO27

IOMUXC_GPIO_EMC_27_FLEXSPI2_A_DATA01

IOMUXC_GPIO_EMC_28_SEMC_WE

IOMUXC_GPIO_EMC_28_FLEXPWM1_PWMB02

IOMUXC_GPIO_EMC_28_LPUART5_CTS_B

IOMUXC_GPIO_EMC_28_LPSPI1_SDO

IOMUXC_GPIO_EMC_28_FLEXIO1_FLEXIO14

IOMUXC_GPIO_EMC_28_GPIO4_IO28

IOMUXC_GPIO_EMC_28_FLEXSPI2_A_DATA02

IOMUXC_GPIO_EMC_29_SEMC_CS0

IOMUXC_GPIO_EMC_29_FLEXPWM3_PWMA00

IOMUXC_GPIO_EMC_29_LPUART6_RTS_B

IOMUXC_GPIO_EMC_29_LPSPI1_SDI

IOMUXC_GPIO_EMC_29_FLEXIO1_FLEXIO15

IOMUXC_GPIO_EMC_29_GPIO4_IO29

IOMUXC_GPIO_EMC_29_FLEXSPI2_A_DATA03

IOMUXC_GPIO_EMC_30_SEMC_DATA08

IOMUXC_GPIO_EMC_30_FLEXPWM3_PWMB00

IOMUXC_GPIO_EMC_30_LPUART6_CTS_B

IOMUXC_GPIO_EMC_30_LPSPI1_PCS0

IOMUXC_GPIO_EMC_30_CSI_DATA23

IOMUXC_GPIO_EMC_30_GPIO4_IO30

IOMUXC_GPIO_EMC_30_ENET2_TDATA00

IOMUXC_GPIO_EMC_31_SEMC_DATA09

IOMUXC_GPIO_EMC_31_FLEXPWM3_PWMA01

IOMUXC_GPIO_EMC_31_LPUART7_TX

IOMUXC_GPIO_EMC_31_LPSPI1_PCS1

IOMUXC_GPIO_EMC_31_CSI_DATA22

IOMUXC_GPIO_EMC_31_GPIO4_IO31

IOMUXC_GPIO_EMC_31_ENET2_TDATA01

IOMUXC_GPIO_EMC_32_SEMC_DATA10

IOMUXC_GPIO_EMC_32_FLEXPWM3_PWMB01

IOMUXC_GPIO_EMC_32_LPUART7_RX

IOMUXC_GPIO_EMC_32_CCM_PMIC_RDY

IOMUXC_GPIO_EMC_32_CSI_DATA21

IOMUXC_GPIO_EMC_32_GPIO3_IO18

IOMUXC_GPIO_EMC_32_ENET2_TX_EN

IOMUXC_GPIO_EMC_33_SEMC_DATA11

IOMUXC_GPIO_EMC_33_FLEXPWM3_PWMA02

IOMUXC_GPIO_EMC_33_USDHC1_RESET_B

IOMUXC_GPIO_EMC_33_SAI3_RX_DATA

IOMUXC_GPIO_EMC_33_CSI_DATA20

IOMUXC_GPIO_EMC_33_GPIO3_IO19

IOMUXC_GPIO_EMC_33_ENET2_TX_CLK

IOMUXC_GPIO_EMC_33_ENET2_REF_CLK2

IOMUXC_GPIO_EMC_34_SEMC_DATA12

IOMUXC_GPIO_EMC_34_FLEXPWM3_PWMB02

IOMUXC_GPIO_EMC_34_USDHC1_VSELECT

IOMUXC_GPIO_EMC_34_SAI3_RX_SYNC

IOMUXC_GPIO_EMC_34_CSI_DATA19

IOMUXC_GPIO_EMC_34_GPIO3_IO20

IOMUXC_GPIO_EMC_34_ENET2_RX_ER

IOMUXC_GPIO_EMC_35_SEMC_DATA13

IOMUXC_GPIO_EMC_35_XBAR1_INOUT18

IOMUXC_GPIO_EMC_35_GPT1_COMPARE1

IOMUXC_GPIO_EMC_35_SAI3_RX_BCLK

IOMUXC_GPIO_EMC_35_CSI_DATA18

IOMUXC_GPIO_EMC_35_GPIO3_IO21

IOMUXC_GPIO_EMC_35_USDHC1_CD_B

IOMUXC_GPIO_EMC_35_ENET2_RDATA00

IOMUXC_GPIO_EMC_36_SEMC_DATA14

IOMUXC_GPIO_EMC_36_XBAR1_IN22

IOMUXC_GPIO_EMC_36_GPT1_COMPARE2

IOMUXC_GPIO_EMC_36_SAI3_TX_DATA

IOMUXC_GPIO_EMC_36_CSI_DATA17

IOMUXC_GPIO_EMC_36_GPIO3_IO22

IOMUXC_GPIO_EMC_36_USDHC1_WP

IOMUXC_GPIO_EMC_36_ENET2_RDATA01

IOMUXC_GPIO_EMC_36_FLEXCAN3_TX

IOMUXC_GPIO_EMC_37_SEMC_DATA15

IOMUXC_GPIO_EMC_37_XBAR1_IN23

IOMUXC_GPIO_EMC_37_GPT1_COMPARE3

IOMUXC_GPIO_EMC_37_SAI3_MCLK

IOMUXC_GPIO_EMC_37_CSI_DATA16

IOMUXC_GPIO_EMC_37_GPIO3_IO23

IOMUXC_GPIO_EMC_37_USDHC2_WP

IOMUXC_GPIO_EMC_37_ENET2_RX_EN

IOMUXC_GPIO_EMC_37_FLEXCAN3_RX

IOMUXC_GPIO_EMC_38_SEMC_DM01

IOMUXC_GPIO_EMC_38_FLEXPWM1_PWMA03

IOMUXC_GPIO_EMC_38_LPUART8_TX

IOMUXC_GPIO_EMC_38_SAI3_TX_BCLK

IOMUXC_GPIO_EMC_38_CSI_FIELD

IOMUXC_GPIO_EMC_38_GPIO3_IO24

IOMUXC_GPIO_EMC_38_USDHC2_VSELECT

IOMUXC_GPIO_EMC_38_ENET2_MDC

IOMUXC_GPIO_EMC_39_SEMC_DQS

IOMUXC_GPIO_EMC_39_FLEXPWM1_PWMB03

IOMUXC_GPIO_EMC_39_LPUART8_RX

IOMUXC_GPIO_EMC_39_SAI3_TX_SYNC

IOMUXC_GPIO_EMC_39_WDOG1_WDOG_B

IOMUXC_GPIO_EMC_39_GPIO3_IO25

IOMUXC_GPIO_EMC_39_USDHC2_CD_B

IOMUXC_GPIO_EMC_39_ENET2_MDIO

IOMUXC_GPIO_EMC_39_SEMC_DQS4

IOMUXC_GPIO_EMC_40_SEMC_RDY

IOMUXC_GPIO_EMC_40_GPT2_CAPTURE2

IOMUXC_GPIO_EMC_40_LPSPI1_PCS2

IOMUXC_GPIO_EMC_40_USB_OTG2_OC

IOMUXC_GPIO_EMC_40_ENET_MDC

IOMUXC_GPIO_EMC_40_GPIO3_IO26

IOMUXC_GPIO_EMC_40_USDHC2_RESET_B

IOMUXC_GPIO_EMC_40_SEMC_CLK5

IOMUXC_GPIO_EMC_41_SEMC_CSX00

IOMUXC_GPIO_EMC_41_GPT2_CAPTURE1

IOMUXC_GPIO_EMC_41_LPSPI1_PCS3

IOMUXC_GPIO_EMC_41_USB_OTG2_PWR

IOMUXC_GPIO_EMC_41_ENET_MDIO

IOMUXC_GPIO_EMC_41_GPIO3_IO27

IOMUXC_GPIO_EMC_41_USDHC1_VSELECT

IOMUXC_GPIO_AD_B0_00_FLEXPWM2_PWMA03

IOMUXC_GPIO_AD_B0_00_XBAR1_INOUT14

IOMUXC_GPIO_AD_B0_00_REF_CLK_32K

IOMUXC_GPIO_AD_B0_00_USB_OTG2_ID

IOMUXC_GPIO_AD_B0_00_LPI2C1_SCLS

IOMUXC_GPIO_AD_B0_00_GPIO1_IO00

IOMUXC_GPIO_AD_B0_00_USDHC1_RESET_B

IOMUXC_GPIO_AD_B0_00_LPSPI3_SCK

IOMUXC_GPIO_AD_B0_01_FLEXPWM2_PWMB03

IOMUXC_GPIO_AD_B0_01_XBAR1_INOUT15

IOMUXC_GPIO_AD_B0_01_REF_CLK_24M

IOMUXC_GPIO_AD_B0_01_USB_OTG1_ID

IOMUXC_GPIO_AD_B0_01_LPI2C1_SDAS

IOMUXC_GPIO_AD_B0_01_GPIO1_IO01

IOMUXC_GPIO_AD_B0_01_EWM_OUT_B

IOMUXC_GPIO_AD_B0_01_LPSPI3_SDO

IOMUXC_GPIO_AD_B0_02_FLEXCAN2_TX

IOMUXC_GPIO_AD_B0_02_XBAR1_INOUT16

IOMUXC_GPIO_AD_B0_02_LPUART6_TX

IOMUXC_GPIO_AD_B0_02_USB_OTG1_PWR

IOMUXC_GPIO_AD_B0_02_FLEXPWM1_PWMX00

IOMUXC_GPIO_AD_B0_02_GPIO1_IO02

IOMUXC_GPIO_AD_B0_02_LPI2C1_HREQ

IOMUXC_GPIO_AD_B0_02_LPSPI3_SDI

IOMUXC_GPIO_AD_B0_03_FLEXCAN2_RX

IOMUXC_GPIO_AD_B0_03_XBAR1_INOUT17

IOMUXC_GPIO_AD_B0_03_LPUART6_RX

IOMUXC_GPIO_AD_B0_03_USB_OTG1_OC

IOMUXC_GPIO_AD_B0_03_FLEXPWM1_PWMX01

IOMUXC_GPIO_AD_B0_03_GPIO1_IO03

IOMUXC_GPIO_AD_B0_03_REF_CLK_24M

IOMUXC_GPIO_AD_B0_03_LPSPI3_PCS0

IOMUXC_GPIO_AD_B0_04_SRC_BOOT_MODE00

IOMUXC_GPIO_AD_B0_04_MQS_RIGHT

IOMUXC_GPIO_AD_B0_04_ENET_TX_DATA03

IOMUXC_GPIO_AD_B0_04_SAI2_TX_SYNC

IOMUXC_GPIO_AD_B0_04_CSI_DATA09

IOMUXC_GPIO_AD_B0_04_GPIO1_IO04

IOMUXC_GPIO_AD_B0_04_PIT_TRIGGER00

IOMUXC_GPIO_AD_B0_04_LPSPI3_PCS1

IOMUXC_GPIO_AD_B0_05_SRC_BOOT_MODE01

IOMUXC_GPIO_AD_B0_05_MQS_LEFT

IOMUXC_GPIO_AD_B0_05_ENET_TX_DATA02

IOMUXC_GPIO_AD_B0_05_SAI2_TX_BCLK

IOMUXC_GPIO_AD_B0_05_CSI_DATA08

IOMUXC_GPIO_AD_B0_05_GPIO1_IO05

IOMUXC_GPIO_AD_B0_05_XBAR1_INOUT17

IOMUXC_GPIO_AD_B0_05_LPSPI3_PCS2

IOMUXC_GPIO_AD_B0_06_JTAG_TMS

IOMUXC_GPIO_AD_B0_06_GPT2_COMPARE1

IOMUXC_GPIO_AD_B0_06_ENET_RX_CLK

IOMUXC_GPIO_AD_B0_06_SAI2_RX_BCLK

IOMUXC_GPIO_AD_B0_06_CSI_DATA07

IOMUXC_GPIO_AD_B0_06_GPIO1_IO06

IOMUXC_GPIO_AD_B0_06_XBAR1_INOUT18

IOMUXC_GPIO_AD_B0_06_LPSPI3_PCS3

IOMUXC_GPIO_AD_B0_07_JTAG_TCK

IOMUXC_GPIO_AD_B0_07_GPT2_COMPARE2

IOMUXC_GPIO_AD_B0_07_ENET_TX_ER

IOMUXC_GPIO_AD_B0_07_SAI2_RX_SYNC

IOMUXC_GPIO_AD_B0_07_CSI_DATA06

IOMUXC_GPIO_AD_B0_07_GPIO1_IO07

IOMUXC_GPIO_AD_B0_07_XBAR1_INOUT19

IOMUXC_GPIO_AD_B0_07_ENET_1588_EVENT3_OUT

IOMUXC_GPIO_AD_B0_08_JTAG_MOD

IOMUXC_GPIO_AD_B0_08_GPT2_COMPARE3

IOMUXC_GPIO_AD_B0_08_ENET_RX_DATA03

IOMUXC_GPIO_AD_B0_08_SAI2_RX_DATA

IOMUXC_GPIO_AD_B0_08_CSI_DATA05

IOMUXC_GPIO_AD_B0_08_GPIO1_IO08

IOMUXC_GPIO_AD_B0_08_XBAR1_IN20

IOMUXC_GPIO_AD_B0_08_ENET_1588_EVENT3_IN

IOMUXC_GPIO_AD_B0_09_JTAG_TDI

IOMUXC_GPIO_AD_B0_09_FLEXPWM2_PWMA03

IOMUXC_GPIO_AD_B0_09_ENET_RX_DATA02

IOMUXC_GPIO_AD_B0_09_SAI2_TX_DATA

IOMUXC_GPIO_AD_B0_09_CSI_DATA04

IOMUXC_GPIO_AD_B0_09_GPIO1_IO09

IOMUXC_GPIO_AD_B0_09_XBAR1_IN21

IOMUXC_GPIO_AD_B0_09_GPT2_CLK

IOMUXC_GPIO_AD_B0_09_SEMC_DQS4

IOMUXC_GPIO_AD_B0_10_JTAG_TDO

IOMUXC_GPIO_AD_B0_10_FLEXPWM1_PWMA03

IOMUXC_GPIO_AD_B0_10_ENET_CRS

IOMUXC_GPIO_AD_B0_10_SAI2_MCLK

IOMUXC_GPIO_AD_B0_10_CSI_DATA03

IOMUXC_GPIO_AD_B0_10_GPIO1_IO10

IOMUXC_GPIO_AD_B0_10_XBAR1_IN22

IOMUXC_GPIO_AD_B0_10_ENET_1588_EVENT0_OUT

IOMUXC_GPIO_AD_B0_10_FLEXCAN3_TX

IOMUXC_GPIO_AD_B0_10_ARM_TRACE_SWO

IOMUXC_GPIO_AD_B0_11_JTAG_TRSTB

IOMUXC_GPIO_AD_B0_11_FLEXPWM1_PWMB03

IOMUXC_GPIO_AD_B0_11_ENET_COL

IOMUXC_GPIO_AD_B0_11_WDOG1_WDOG_B

IOMUXC_GPIO_AD_B0_11_CSI_DATA02

IOMUXC_GPIO_AD_B0_11_GPIO1_IO11

IOMUXC_GPIO_AD_B0_11_XBAR1_IN23

IOMUXC_GPIO_AD_B0_11_ENET_1588_EVENT0_IN

IOMUXC_GPIO_AD_B0_11_FLEXCAN3_RX

IOMUXC_GPIO_AD_B0_11_SEMC_CLK6

IOMUXC_GPIO_AD_B0_12_LPI2C4_SCL

IOMUXC_GPIO_AD_B0_12_CCM_PMIC_READY

IOMUXC_GPIO_AD_B0_12_LPUART1_TX

IOMUXC_GPIO_AD_B0_12_WDOG2_WDOG_B

IOMUXC_GPIO_AD_B0_12_FLEXPWM1_PWMX02

IOMUXC_GPIO_AD_B0_12_GPIO1_IO12

IOMUXC_GPIO_AD_B0_12_ENET_1588_EVENT1_OUT

IOMUXC_GPIO_AD_B0_12_NMI_GLUE_NMI

IOMUXC_GPIO_AD_B0_13_LPI2C4_SDA

IOMUXC_GPIO_AD_B0_13_GPT1_CLK

IOMUXC_GPIO_AD_B0_13_LPUART1_RX

IOMUXC_GPIO_AD_B0_13_EWM_OUT_B

IOMUXC_GPIO_AD_B0_13_FLEXPWM1_PWMX03

IOMUXC_GPIO_AD_B0_13_GPIO1_IO13

IOMUXC_GPIO_AD_B0_13_ENET_1588_EVENT1_IN

IOMUXC_GPIO_AD_B0_13_REF_CLK_24M

IOMUXC_GPIO_AD_B0_14_USB_OTG2_OC

IOMUXC_GPIO_AD_B0_14_XBAR1_IN24

IOMUXC_GPIO_AD_B0_14_LPUART1_CTS_B

IOMUXC_GPIO_AD_B0_14_ENET_1588_EVENT0_OUT

IOMUXC_GPIO_AD_B0_14_CSI_VSYNC

IOMUXC_GPIO_AD_B0_14_GPIO1_IO14

IOMUXC_GPIO_AD_B0_14_FLEXCAN2_TX

IOMUXC_GPIO_AD_B0_14_FLEXCAN3_TX

IOMUXC_GPIO_AD_B0_15_USB_OTG2_PWR

IOMUXC_GPIO_AD_B0_15_XBAR1_IN25

IOMUXC_GPIO_AD_B0_15_LPUART1_RTS_B

IOMUXC_GPIO_AD_B0_15_ENET_1588_EVENT0_IN

IOMUXC_GPIO_AD_B0_15_CSI_HSYNC

IOMUXC_GPIO_AD_B0_15_GPIO1_IO15

IOMUXC_GPIO_AD_B0_15_FLEXCAN2_RX

IOMUXC_GPIO_AD_B0_15_WDOG1_WDOG_RST_B_DEB

IOMUXC_GPIO_AD_B0_15_FLEXCAN3_RX

IOMUXC_GPIO_AD_B1_00_USB_OTG2_ID

IOMUXC_GPIO_AD_B1_00_QTIMER3_TIMER0

IOMUXC_GPIO_AD_B1_00_LPUART2_CTS_B

IOMUXC_GPIO_AD_B1_00_LPI2C1_SCL

IOMUXC_GPIO_AD_B1_00_WDOG1_B

IOMUXC_GPIO_AD_B1_00_GPIO1_IO16

IOMUXC_GPIO_AD_B1_00_USDHC1_WP

IOMUXC_GPIO_AD_B1_00_KPP_ROW07

IOMUXC_GPIO_AD_B1_00_ENET2_1588_EVENT0_OUT

IOMUXC_GPIO_AD_B1_00_FLEXIO3_FLEXIO00

IOMUXC_GPIO_AD_B1_01_USB_OTG1_PWR

IOMUXC_GPIO_AD_B1_01_QTIMER3_TIMER1

IOMUXC_GPIO_AD_B1_01_LPUART2_RTS_B

IOMUXC_GPIO_AD_B1_01_LPI2C1_SDA

IOMUXC_GPIO_AD_B1_01_CCM_PMIC_READY

IOMUXC_GPIO_AD_B1_01_GPIO1_IO17

IOMUXC_GPIO_AD_B1_01_USDHC1_VSELECT

IOMUXC_GPIO_AD_B1_01_KPP_COL07

IOMUXC_GPIO_AD_B1_01_ENET2_1588_EVENT0_IN

IOMUXC_GPIO_AD_B1_01_FLEXIO3_FLEXIO01

IOMUXC_GPIO_AD_B1_02_USB_OTG1_ID

IOMUXC_GPIO_AD_B1_02_QTIMER3_TIMER2

IOMUXC_GPIO_AD_B1_02_LPUART2_TX

IOMUXC_GPIO_AD_B1_02_SPDIF_OUT

IOMUXC_GPIO_AD_B1_02_ENET_1588_EVENT2_OUT

IOMUXC_GPIO_AD_B1_02_GPIO1_IO18

IOMUXC_GPIO_AD_B1_02_USDHC1_CD_B

IOMUXC_GPIO_AD_B1_02_KPP_ROW06

IOMUXC_GPIO_AD_B1_02_GPT2_CLK

IOMUXC_GPIO_AD_B1_02_FLEXIO3_FLEXIO02

IOMUXC_GPIO_AD_B1_03_USB_OTG1_OC

IOMUXC_GPIO_AD_B1_03_QTIMER3_TIMER3

IOMUXC_GPIO_AD_B1_03_LPUART2_RX

IOMUXC_GPIO_AD_B1_03_SPDIF_IN

IOMUXC_GPIO_AD_B1_03_ENET_1588_EVENT2_IN

IOMUXC_GPIO_AD_B1_03_GPIO1_IO19

IOMUXC_GPIO_AD_B1_03_USDHC2_CD_B

IOMUXC_GPIO_AD_B1_03_KPP_COL06

IOMUXC_GPIO_AD_B1_03_GPT2_CAPTURE1

IOMUXC_GPIO_AD_B1_03_FLEXIO3_FLEXIO03

IOMUXC_GPIO_AD_B1_04_FLEXSPIB_DATA03

IOMUXC_GPIO_AD_B1_04_ENET_MDC

IOMUXC_GPIO_AD_B1_04_LPUART3_CTS_B

IOMUXC_GPIO_AD_B1_04_SPDIF_SR_CLK

IOMUXC_GPIO_AD_B1_04_CSI_PIXCLK

IOMUXC_GPIO_AD_B1_04_GPIO1_IO20

IOMUXC_GPIO_AD_B1_04_USDHC2_DATA0

IOMUXC_GPIO_AD_B1_04_KPP_ROW05

IOMUXC_GPIO_AD_B1_04_GPT2_CAPTURE2

IOMUXC_GPIO_AD_B1_04_FLEXIO3_FLEXIO04

IOMUXC_GPIO_AD_B1_05_FLEXSPIB_DATA02

IOMUXC_GPIO_AD_B1_05_ENET_MDIO

IOMUXC_GPIO_AD_B1_05_LPUART3_RTS_B

IOMUXC_GPIO_AD_B1_05_SPDIF_OUT

IOMUXC_GPIO_AD_B1_05_CSI_MCLK

IOMUXC_GPIO_AD_B1_05_GPIO1_IO21

IOMUXC_GPIO_AD_B1_05_USDHC2_DATA1

IOMUXC_GPIO_AD_B1_05_KPP_COL05

IOMUXC_GPIO_AD_B1_05_GPT2_COMPARE1

IOMUXC_GPIO_AD_B1_05_FLEXIO3_FLEXIO05

IOMUXC_GPIO_AD_B1_06_FLEXSPIB_DATA01

IOMUXC_GPIO_AD_B1_06_LPI2C3_SDA

IOMUXC_GPIO_AD_B1_06_LPUART3_TX

IOMUXC_GPIO_AD_B1_06_SPDIF_LOCK

IOMUXC_GPIO_AD_B1_06_CSI_VSYNC

IOMUXC_GPIO_AD_B1_06_GPIO1_IO22

IOMUXC_GPIO_AD_B1_06_USDHC2_DATA2

IOMUXC_GPIO_AD_B1_06_KPP_ROW04

IOMUXC_GPIO_AD_B1_06_GPT2_COMPARE2

IOMUXC_GPIO_AD_B1_06_FLEXIO3_FLEXIO06

IOMUXC_GPIO_AD_B1_07_FLEXSPIB_DATA00

IOMUXC_GPIO_AD_B1_07_LPI2C3_SCL

IOMUXC_GPIO_AD_B1_07_LPUART3_RX

IOMUXC_GPIO_AD_B1_07_SPDIF_EXT_CLK

IOMUXC_GPIO_AD_B1_07_CSI_HSYNC

IOMUXC_GPIO_AD_B1_07_GPIO1_IO23

IOMUXC_GPIO_AD_B1_07_USDHC2_DATA3

IOMUXC_GPIO_AD_B1_07_KPP_COL04

IOMUXC_GPIO_AD_B1_07_GPT2_COMPARE3

IOMUXC_GPIO_AD_B1_07_FLEXIO3_FLEXIO07

IOMUXC_GPIO_AD_B1_08_FLEXSPIA_SS1_B

IOMUXC_GPIO_AD_B1_08_FLEXPWM4_PWMA00

IOMUXC_GPIO_AD_B1_08_FLEXCAN1_TX

IOMUXC_GPIO_AD_B1_08_CCM_PMIC_READY

IOMUXC_GPIO_AD_B1_08_CSI_DATA09

IOMUXC_GPIO_AD_B1_08_GPIO1_IO24

IOMUXC_GPIO_AD_B1_08_USDHC2_CMD

IOMUXC_GPIO_AD_B1_08_KPP_ROW03

IOMUXC_GPIO_AD_B1_08_FLEXIO3_FLEXIO08

IOMUXC_GPIO_AD_B1_09_FLEXSPIA_DQS

IOMUXC_GPIO_AD_B1_09_FLEXPWM4_PWMA01

IOMUXC_GPIO_AD_B1_09_FLEXCAN1_RX

IOMUXC_GPIO_AD_B1_09_SAI1_MCLK

IOMUXC_GPIO_AD_B1_09_CSI_DATA08

IOMUXC_GPIO_AD_B1_09_GPIO1_IO25

IOMUXC_GPIO_AD_B1_09_USDHC2_CLK

IOMUXC_GPIO_AD_B1_09_KPP_COL03

IOMUXC_GPIO_AD_B1_09_FLEXIO3_FLEXIO09

IOMUXC_GPIO_AD_B1_10_FLEXSPIA_DATA03

IOMUXC_GPIO_AD_B1_10_WDOG1_B

IOMUXC_GPIO_AD_B1_10_LPUART8_TX

IOMUXC_GPIO_AD_B1_10_SAI1_RX_SYNC

IOMUXC_GPIO_AD_B1_10_CSI_DATA07

IOMUXC_GPIO_AD_B1_10_GPIO1_IO26

IOMUXC_GPIO_AD_B1_10_USDHC2_WP

IOMUXC_GPIO_AD_B1_10_KPP_ROW02

IOMUXC_GPIO_AD_B1_10_ENET2_1588_EVENT1_OUT

IOMUXC_GPIO_AD_B1_10_FLEXIO3_FLEXIO10

IOMUXC_GPIO_AD_B1_11_FLEXSPIA_DATA02

IOMUXC_GPIO_AD_B1_11_EWM_OUT_B

IOMUXC_GPIO_AD_B1_11_LPUART8_RX

IOMUXC_GPIO_AD_B1_11_SAI1_RX_BCLK

IOMUXC_GPIO_AD_B1_11_CSI_DATA06

IOMUXC_GPIO_AD_B1_11_GPIO1_IO27

IOMUXC_GPIO_AD_B1_11_USDHC2_RESET_B

IOMUXC_GPIO_AD_B1_11_KPP_COL02

IOMUXC_GPIO_AD_B1_11_ENET2_1588_EVENT1_IN

IOMUXC_GPIO_AD_B1_11_FLEXIO3_FLEXIO11

IOMUXC_GPIO_AD_B1_12_FLEXSPIA_DATA01

IOMUXC_GPIO_AD_B1_12_ACMP_OUT00

IOMUXC_GPIO_AD_B1_12_LPSPI3_PCS0

IOMUXC_GPIO_AD_B1_12_SAI1_RX_DATA00

IOMUXC_GPIO_AD_B1_12_CSI_DATA05

IOMUXC_GPIO_AD_B1_12_GPIO1_IO28

IOMUXC_GPIO_AD_B1_12_USDHC2_DATA4

IOMUXC_GPIO_AD_B1_12_KPP_ROW01

IOMUXC_GPIO_AD_B1_12_ENET2_1588_EVENT2_OUT

IOMUXC_GPIO_AD_B1_12_FLEXIO3_FLEXIO12

IOMUXC_GPIO_AD_B1_13_FLEXSPIA_DATA00

IOMUXC_GPIO_AD_B1_13_ACMP_OUT01

IOMUXC_GPIO_AD_B1_13_LPSPI3_SDI

IOMUXC_GPIO_AD_B1_13_SAI1_TX_DATA00

IOMUXC_GPIO_AD_B1_13_CSI_DATA04

IOMUXC_GPIO_AD_B1_13_GPIO1_IO29

IOMUXC_GPIO_AD_B1_13_USDHC2_DATA5

IOMUXC_GPIO_AD_B1_13_KPP_COL01

IOMUXC_GPIO_AD_B1_13_ENET2_1588_EVENT2_IN

IOMUXC_GPIO_AD_B1_13_FLEXIO3_FLEXIO13

IOMUXC_GPIO_AD_B1_14_FLEXSPIA_SCLK

IOMUXC_GPIO_AD_B1_14_ACMP_OUT02

IOMUXC_GPIO_AD_B1_14_LPSPI3_SDO

IOMUXC_GPIO_AD_B1_14_SAI1_TX_BCLK

IOMUXC_GPIO_AD_B1_14_CSI_DATA03

IOMUXC_GPIO_AD_B1_14_GPIO1_IO30

IOMUXC_GPIO_AD_B1_14_USDHC2_DATA6

IOMUXC_GPIO_AD_B1_14_KPP_ROW00

IOMUXC_GPIO_AD_B1_14_ENET2_1588_EVENT3_OUT

IOMUXC_GPIO_AD_B1_14_FLEXIO3_FLEXIO14

IOMUXC_GPIO_AD_B1_15_FLEXSPIA_SS0_B

IOMUXC_GPIO_AD_B1_15_ACMP_OUT03

IOMUXC_GPIO_AD_B1_15_LPSPI3_SCK

IOMUXC_GPIO_AD_B1_15_SAI1_TX_SYNC

IOMUXC_GPIO_AD_B1_15_CSI_DATA02

IOMUXC_GPIO_AD_B1_15_GPIO1_IO31

IOMUXC_GPIO_AD_B1_15_USDHC2_DATA7

IOMUXC_GPIO_AD_B1_15_KPP_COL00

IOMUXC_GPIO_AD_B1_15_ENET2_1588_EVENT3_IN

IOMUXC_GPIO_AD_B1_15_FLEXIO3_FLEXIO15

IOMUXC_GPIO_B0_00_LCD_CLK

IOMUXC_GPIO_B0_00_QTIMER1_TIMER0

IOMUXC_GPIO_B0_00_MQS_RIGHT

IOMUXC_GPIO_B0_00_LPSPI4_PCS0

IOMUXC_GPIO_B0_00_FLEXIO2_FLEXIO00

IOMUXC_GPIO_B0_00_GPIO2_IO00

IOMUXC_GPIO_B0_00_SEMC_CSX01

IOMUXC_GPIO_B0_00_ENET2_MDC

IOMUXC_GPIO_B0_01_LCD_ENABLE

IOMUXC_GPIO_B0_01_QTIMER1_TIMER1

IOMUXC_GPIO_B0_01_MQS_LEFT

IOMUXC_GPIO_B0_01_LPSPI4_SDI

IOMUXC_GPIO_B0_01_FLEXIO2_FLEXIO01

IOMUXC_GPIO_B0_01_GPIO2_IO01

IOMUXC_GPIO_B0_01_SEMC_CSX02

IOMUXC_GPIO_B0_01_ENET2_MDIO

IOMUXC_GPIO_B0_02_LCD_HSYNC

IOMUXC_GPIO_B0_02_QTIMER1_TIMER2

IOMUXC_GPIO_B0_02_FLEXCAN1_TX

IOMUXC_GPIO_B0_02_LPSPI4_SDO

IOMUXC_GPIO_B0_02_FLEXIO2_FLEXIO02

IOMUXC_GPIO_B0_02_GPIO2_IO02

IOMUXC_GPIO_B0_02_SEMC_CSX03

IOMUXC_GPIO_B0_02_ENET2_1588_EVENT0_OUT

IOMUXC_GPIO_B0_03_LCD_VSYNC

IOMUXC_GPIO_B0_03_QTIMER2_TIMER0

IOMUXC_GPIO_B0_03_FLEXCAN1_RX

IOMUXC_GPIO_B0_03_LPSPI4_SCK

IOMUXC_GPIO_B0_03_FLEXIO2_FLEXIO03

IOMUXC_GPIO_B0_03_GPIO2_IO03

IOMUXC_GPIO_B0_03_WDOG2_RESET_B_DEB

IOMUXC_GPIO_B0_03_ENET2_1588_EVENT0_IN

IOMUXC_GPIO_B0_04_LCD_DATA00

IOMUXC_GPIO_B0_04_QTIMER2_TIMER1

IOMUXC_GPIO_B0_04_LPI2C2_SCL

IOMUXC_GPIO_B0_04_ARM_TRACE0

IOMUXC_GPIO_B0_04_FLEXIO2_FLEXIO04

IOMUXC_GPIO_B0_04_GPIO2_IO04

IOMUXC_GPIO_B0_04_SRC_BOOT_CFG00

IOMUXC_GPIO_B0_04_ENET2_TDATA03

IOMUXC_GPIO_B0_05_LCD_DATA01

IOMUXC_GPIO_B0_05_QTIMER2_TIMER2

IOMUXC_GPIO_B0_05_LPI2C2_SDA

IOMUXC_GPIO_B0_05_ARM_TRACE1

IOMUXC_GPIO_B0_05_FLEXIO2_FLEXIO05

IOMUXC_GPIO_B0_05_GPIO2_IO05

IOMUXC_GPIO_B0_05_SRC_BOOT_CFG01

IOMUXC_GPIO_B0_05_ENET2_TDATA02

IOMUXC_GPIO_B0_06_LCD_DATA02

IOMUXC_GPIO_B0_06_QTIMER3_TIMER0

IOMUXC_GPIO_B0_06_FLEXPWM2_PWMA00

IOMUXC_GPIO_B0_06_ARM_TRACE2

IOMUXC_GPIO_B0_06_FLEXIO2_FLEXIO06

IOMUXC_GPIO_B0_06_GPIO2_IO06

IOMUXC_GPIO_B0_06_SRC_BOOT_CFG02

IOMUXC_GPIO_B0_06_ENET2_RX_CLK

IOMUXC_GPIO_B0_07_LCD_DATA03

IOMUXC_GPIO_B0_07_QTIMER3_TIMER1

IOMUXC_GPIO_B0_07_FLEXPWM2_PWMB00

IOMUXC_GPIO_B0_07_ARM_TRACE3

IOMUXC_GPIO_B0_07_FLEXIO2_FLEXIO07

IOMUXC_GPIO_B0_07_GPIO2_IO07

IOMUXC_GPIO_B0_07_SRC_BOOT_CFG03

IOMUXC_GPIO_B0_07_ENET2_TX_ER

IOMUXC_GPIO_B0_08_LCD_DATA04

IOMUXC_GPIO_B0_08_QTIMER3_TIMER2

IOMUXC_GPIO_B0_08_FLEXPWM2_PWMA01

IOMUXC_GPIO_B0_08_LPUART3_TX

IOMUXC_GPIO_B0_08_FLEXIO2_FLEXIO08

IOMUXC_GPIO_B0_08_GPIO2_IO08

IOMUXC_GPIO_B0_08_SRC_BOOT_CFG04

IOMUXC_GPIO_B0_08_ENET2_RDATA03

IOMUXC_GPIO_B0_09_LCD_DATA05

IOMUXC_GPIO_B0_09_QTIMER4_TIMER0

IOMUXC_GPIO_B0_09_FLEXPWM2_PWMB01

IOMUXC_GPIO_B0_09_LPUART3_RX

IOMUXC_GPIO_B0_09_FLEXIO2_FLEXIO09

IOMUXC_GPIO_B0_09_GPIO2_IO09

IOMUXC_GPIO_B0_09_SRC_BOOT_CFG05

IOMUXC_GPIO_B0_09_ENET2_RDATA02

IOMUXC_GPIO_B0_10_LCD_DATA06

IOMUXC_GPIO_B0_10_QTIMER4_TIMER1

IOMUXC_GPIO_B0_10_FLEXPWM2_PWMA02

IOMUXC_GPIO_B0_10_SAI1_TX_DATA03

IOMUXC_GPIO_B0_10_FLEXIO2_FLEXIO10

IOMUXC_GPIO_B0_10_GPIO2_IO10

IOMUXC_GPIO_B0_10_SRC_BOOT_CFG06

IOMUXC_GPIO_B0_10_ENET2_CRS

IOMUXC_GPIO_B0_11_LCD_DATA07

IOMUXC_GPIO_B0_11_QTIMER4_TIMER2

IOMUXC_GPIO_B0_11_FLEXPWM2_PWMB02

IOMUXC_GPIO_B0_11_SAI1_TX_DATA02

IOMUXC_GPIO_B0_11_FLEXIO2_FLEXIO11

IOMUXC_GPIO_B0_11_GPIO2_IO11

IOMUXC_GPIO_B0_11_SRC_BOOT_CFG07

IOMUXC_GPIO_B0_11_ENET2_COL

IOMUXC_GPIO_B0_12_LCD_DATA08

IOMUXC_GPIO_B0_12_XBAR1_INOUT10

IOMUXC_GPIO_B0_12_ARM_TRACE_CLK

IOMUXC_GPIO_B0_12_SAI1_TX_DATA01

IOMUXC_GPIO_B0_12_FLEXIO2_FLEXIO12

IOMUXC_GPIO_B0_12_GPIO2_IO12

IOMUXC_GPIO_B0_12_SRC_BOOT_CFG08

IOMUXC_GPIO_B0_12_ENET2_TDATA00

IOMUXC_GPIO_B0_13_LCD_DATA09

IOMUXC_GPIO_B0_13_XBAR1_INOUT11

IOMUXC_GPIO_B0_13_ARM_TRACE_SWO

IOMUXC_GPIO_B0_13_SAI1_MCLK

IOMUXC_GPIO_B0_13_FLEXIO2_FLEXIO13

IOMUXC_GPIO_B0_13_GPIO2_IO13

IOMUXC_GPIO_B0_13_SRC_BOOT_CFG09

IOMUXC_GPIO_B0_13_ENET2_TDATA01

IOMUXC_GPIO_B0_14_LCD_DATA10

IOMUXC_GPIO_B0_14_XBAR1_INOUT12

IOMUXC_GPIO_B0_14_ARM_TXEV

IOMUXC_GPIO_B0_14_SAI1_RX_SYNC

IOMUXC_GPIO_B0_14_FLEXIO2_FLEXIO14

IOMUXC_GPIO_B0_14_GPIO2_IO14

IOMUXC_GPIO_B0_14_SRC_BOOT_CFG10

IOMUXC_GPIO_B0_14_ENET2_TX_EN

IOMUXC_GPIO_B0_15_LCD_DATA11

IOMUXC_GPIO_B0_15_XBAR1_INOUT13

IOMUXC_GPIO_B0_15_ARM_RXEV

IOMUXC_GPIO_B0_15_SAI1_RX_BCLK

IOMUXC_GPIO_B0_15_FLEXIO2_FLEXIO15

IOMUXC_GPIO_B0_15_GPIO2_IO15

IOMUXC_GPIO_B0_15_SRC_BOOT_CFG11

IOMUXC_GPIO_B0_15_ENET2_TX_CLK

IOMUXC_GPIO_B0_15_ENET2_REF_CLK2

IOMUXC_GPIO_B1_00_LCD_DATA12

IOMUXC_GPIO_B1_00_XBAR1_INOUT14

IOMUXC_GPIO_B1_00_LPUART4_TX

IOMUXC_GPIO_B1_00_SAI1_RX_DATA00

IOMUXC_GPIO_B1_00_FLEXIO2_FLEXIO16

IOMUXC_GPIO_B1_00_GPIO2_IO16

IOMUXC_GPIO_B1_00_FLEXPWM1_PWMA03

IOMUXC_GPIO_B1_00_ENET2_RX_ER

IOMUXC_GPIO_B1_00_FLEXIO3_FLEXIO16

IOMUXC_GPIO_B1_01_LCD_DATA13

IOMUXC_GPIO_B1_01_XBAR1_INOUT15

IOMUXC_GPIO_B1_01_LPUART4_RX

IOMUXC_GPIO_B1_01_SAI1_TX_DATA00

IOMUXC_GPIO_B1_01_FLEXIO2_FLEXIO17

IOMUXC_GPIO_B1_01_GPIO2_IO17

IOMUXC_GPIO_B1_01_FLEXPWM1_PWMB03

IOMUXC_GPIO_B1_01_ENET2_RDATA00

IOMUXC_GPIO_B1_01_FLEXIO3_FLEXIO17

IOMUXC_GPIO_B1_02_LCD_DATA14

IOMUXC_GPIO_B1_02_XBAR1_INOUT16

IOMUXC_GPIO_B1_02_LPSPI4_PCS2

IOMUXC_GPIO_B1_02_SAI1_TX_BCLK

IOMUXC_GPIO_B1_02_FLEXIO2_FLEXIO18

IOMUXC_GPIO_B1_02_GPIO2_IO18

IOMUXC_GPIO_B1_02_FLEXPWM2_PWMA03

IOMUXC_GPIO_B1_02_ENET2_RDATA01

IOMUXC_GPIO_B1_02_FLEXIO3_FLEXIO18

IOMUXC_GPIO_B1_03_LCD_DATA15

IOMUXC_GPIO_B1_03_XBAR1_INOUT17

IOMUXC_GPIO_B1_03_LPSPI4_PCS1

IOMUXC_GPIO_B1_03_SAI1_TX_SYNC

IOMUXC_GPIO_B1_03_FLEXIO2_FLEXIO19

IOMUXC_GPIO_B1_03_GPIO2_IO19

IOMUXC_GPIO_B1_03_FLEXPWM2_PWMB03

IOMUXC_GPIO_B1_03_ENET2_RX_EN

IOMUXC_GPIO_B1_03_FLEXIO3_FLEXIO19

IOMUXC_GPIO_B1_04_LCD_DATA16

IOMUXC_GPIO_B1_04_LPSPI4_PCS0

IOMUXC_GPIO_B1_04_CSI_DATA15

IOMUXC_GPIO_B1_04_ENET_RX_DATA00

IOMUXC_GPIO_B1_04_FLEXIO2_FLEXIO20

IOMUXC_GPIO_B1_04_GPIO2_IO20

IOMUXC_GPIO_B1_04_GPT1_CLK

IOMUXC_GPIO_B1_04_FLEXIO3_FLEXIO20

IOMUXC_GPIO_B1_05_LCD_DATA17

IOMUXC_GPIO_B1_05_LPSPI4_SDI

IOMUXC_GPIO_B1_05_CSI_DATA14

IOMUXC_GPIO_B1_05_ENET_RX_DATA01

IOMUXC_GPIO_B1_05_FLEXIO2_FLEXIO21

IOMUXC_GPIO_B1_05_GPIO2_IO21

IOMUXC_GPIO_B1_05_GPT1_CAPTURE1

IOMUXC_GPIO_B1_05_FLEXIO3_FLEXIO21

IOMUXC_GPIO_B1_06_LCD_DATA18

IOMUXC_GPIO_B1_06_LPSPI4_SDO

IOMUXC_GPIO_B1_06_CSI_DATA13

IOMUXC_GPIO_B1_06_ENET_RX_EN

IOMUXC_GPIO_B1_06_FLEXIO2_FLEXIO22

IOMUXC_GPIO_B1_06_GPIO2_IO22

IOMUXC_GPIO_B1_06_GPT1_CAPTURE2

IOMUXC_GPIO_B1_06_FLEXIO3_FLEXIO22

IOMUXC_GPIO_B1_07_LCD_DATA19

IOMUXC_GPIO_B1_07_LPSPI4_SCK

IOMUXC_GPIO_B1_07_CSI_DATA12

IOMUXC_GPIO_B1_07_ENET_TX_DATA00

IOMUXC_GPIO_B1_07_FLEXIO2_FLEXIO23

IOMUXC_GPIO_B1_07_GPIO2_IO23

IOMUXC_GPIO_B1_07_GPT1_COMPARE1

IOMUXC_GPIO_B1_07_FLEXIO3_FLEXIO23

IOMUXC_GPIO_B1_08_LCD_DATA20

IOMUXC_GPIO_B1_08_QTIMER1_TIMER3

IOMUXC_GPIO_B1_08_CSI_DATA11

IOMUXC_GPIO_B1_08_ENET_TX_DATA01

IOMUXC_GPIO_B1_08_FLEXIO2_FLEXIO24

IOMUXC_GPIO_B1_08_GPIO2_IO24

IOMUXC_GPIO_B1_08_FLEXCAN2_TX

IOMUXC_GPIO_B1_08_GPT1_COMPARE2

IOMUXC_GPIO_B1_08_FLEXIO3_FLEXIO24

IOMUXC_GPIO_B1_09_LCD_DATA21

IOMUXC_GPIO_B1_09_QTIMER2_TIMER3

IOMUXC_GPIO_B1_09_CSI_DATA10

IOMUXC_GPIO_B1_09_ENET_TX_EN

IOMUXC_GPIO_B1_09_FLEXIO2_FLEXIO25

IOMUXC_GPIO_B1_09_GPIO2_IO25

IOMUXC_GPIO_B1_09_FLEXCAN2_RX

IOMUXC_GPIO_B1_09_GPT1_COMPARE3

IOMUXC_GPIO_B1_09_FLEXIO3_FLEXIO25

IOMUXC_GPIO_B1_10_LCD_DATA22

IOMUXC_GPIO_B1_10_QTIMER3_TIMER3

IOMUXC_GPIO_B1_10_CSI_DATA00

IOMUXC_GPIO_B1_10_ENET_TX_CLK

IOMUXC_GPIO_B1_10_FLEXIO2_FLEXIO26

IOMUXC_GPIO_B1_10_GPIO2_IO26

IOMUXC_GPIO_B1_10_ENET_REF_CLK

IOMUXC_GPIO_B1_10_FLEXIO3_FLEXIO26

IOMUXC_GPIO_B1_11_LCD_DATA23

IOMUXC_GPIO_B1_11_QTIMER4_TIMER3

IOMUXC_GPIO_B1_11_CSI_DATA01

IOMUXC_GPIO_B1_11_ENET_RX_ER

IOMUXC_GPIO_B1_11_FLEXIO2_FLEXIO27

IOMUXC_GPIO_B1_11_GPIO2_IO27

IOMUXC_GPIO_B1_11_LPSPI4_PCS3

IOMUXC_GPIO_B1_11_FLEXIO3_FLEXIO27

IOMUXC_GPIO_B1_12_LPUART5_TX

IOMUXC_GPIO_B1_12_CSI_PIXCLK

IOMUXC_GPIO_B1_12_ENET_1588_EVENT0_IN

IOMUXC_GPIO_B1_12_FLEXIO2_FLEXIO28

IOMUXC_GPIO_B1_12_GPIO2_IO28

IOMUXC_GPIO_B1_12_USDHC1_CD_B

IOMUXC_GPIO_B1_12_FLEXIO3_FLEXIO28

IOMUXC_GPIO_B1_13_WDOG1_B

IOMUXC_GPIO_B1_13_LPUART5_RX

IOMUXC_GPIO_B1_13_CSI_VSYNC

IOMUXC_GPIO_B1_13_ENET_1588_EVENT0_OUT

IOMUXC_GPIO_B1_13_FLEXIO2_FLEXIO29

IOMUXC_GPIO_B1_13_GPIO2_IO29

IOMUXC_GPIO_B1_13_USDHC1_WP

IOMUXC_GPIO_B1_13_SEMC_DQS4

IOMUXC_GPIO_B1_13_FLEXIO3_FLEXIO29

IOMUXC_GPIO_B1_14_ENET_MDC

IOMUXC_GPIO_B1_14_FLEXPWM4_PWMA02

IOMUXC_GPIO_B1_14_CSI_HSYNC

IOMUXC_GPIO_B1_14_XBAR1_IN02

IOMUXC_GPIO_B1_14_FLEXIO2_FLEXIO30

IOMUXC_GPIO_B1_14_GPIO2_IO30

IOMUXC_GPIO_B1_14_USDHC1_VSELECT

IOMUXC_GPIO_B1_14_ENET2_TDATA00

IOMUXC_GPIO_B1_14_FLEXIO3_FLEXIO30

IOMUXC_GPIO_B1_15_ENET_MDIO

IOMUXC_GPIO_B1_15_FLEXPWM4_PWMA03

IOMUXC_GPIO_B1_15_CSI_MCLK

IOMUXC_GPIO_B1_15_XBAR1_IN03

IOMUXC_GPIO_B1_15_FLEXIO2_FLEXIO31

IOMUXC_GPIO_B1_15_GPIO2_IO31

IOMUXC_GPIO_B1_15_USDHC1_RESET_B

IOMUXC_GPIO_B1_15_ENET2_TDATA01

IOMUXC_GPIO_B1_15_FLEXIO3_FLEXIO31

IOMUXC_GPIO_SD_B0_00_USDHC1_CMD

IOMUXC_GPIO_SD_B0_00_FLEXPWM1_PWMA00

IOMUXC_GPIO_SD_B0_00_LPI2C3_SCL

IOMUXC_GPIO_SD_B0_00_XBAR1_INOUT04

IOMUXC_GPIO_SD_B0_00_LPSPI1_SCK

IOMUXC_GPIO_SD_B0_00_GPIO3_IO12

IOMUXC_GPIO_SD_B0_00_FLEXSPIA_SS1_B

IOMUXC_GPIO_SD_B0_00_ENET2_TX_EN

IOMUXC_GPIO_SD_B0_00_SEMC_DQS4

IOMUXC_GPIO_SD_B0_01_USDHC1_CLK

IOMUXC_GPIO_SD_B0_01_FLEXPWM1_PWMB00

IOMUXC_GPIO_SD_B0_01_LPI2C3_SDA

IOMUXC_GPIO_SD_B0_01_XBAR1_INOUT05

IOMUXC_GPIO_SD_B0_01_LPSPI1_PCS0

IOMUXC_GPIO_SD_B0_01_GPIO3_IO13

IOMUXC_GPIO_SD_B0_01_FLEXSPIB_SS1_B

IOMUXC_GPIO_SD_B0_01_ENET2_TX_CLK

IOMUXC_GPIO_SD_B0_01_ENET2_REF_CLK2

IOMUXC_GPIO_SD_B0_02_USDHC1_DATA0

IOMUXC_GPIO_SD_B0_02_FLEXPWM1_PWMA01

IOMUXC_GPIO_SD_B0_02_LPUART8_CTS_B

IOMUXC_GPIO_SD_B0_02_XBAR1_INOUT06

IOMUXC_GPIO_SD_B0_02_LPSPI1_SDO

IOMUXC_GPIO_SD_B0_02_GPIO3_IO14

IOMUXC_GPIO_SD_B0_02_ENET2_RX_ER

IOMUXC_GPIO_SD_B0_02_SEMC_CLK5

IOMUXC_GPIO_SD_B0_03_USDHC1_DATA1

IOMUXC_GPIO_SD_B0_03_FLEXPWM1_PWMB01

IOMUXC_GPIO_SD_B0_03_LPUART8_RTS_B

IOMUXC_GPIO_SD_B0_03_XBAR1_INOUT07

IOMUXC_GPIO_SD_B0_03_LPSPI1_SDI

IOMUXC_GPIO_SD_B0_03_GPIO3_IO15

IOMUXC_GPIO_SD_B0_03_ENET2_RDATA00

IOMUXC_GPIO_SD_B0_03_SEMC_CLK6

IOMUXC_GPIO_SD_B0_04_USDHC1_DATA2

IOMUXC_GPIO_SD_B0_04_FLEXPWM1_PWMA02

IOMUXC_GPIO_SD_B0_04_LPUART8_TX

IOMUXC_GPIO_SD_B0_04_XBAR1_INOUT08

IOMUXC_GPIO_SD_B0_04_FLEXSPIB_SS0_B

IOMUXC_GPIO_SD_B0_04_GPIO3_IO16

IOMUXC_GPIO_SD_B0_04_CCM_CLKO1

IOMUXC_GPIO_SD_B0_04_ENET2_RDATA01

IOMUXC_GPIO_SD_B0_05_USDHC1_DATA3

IOMUXC_GPIO_SD_B0_05_FLEXPWM1_PWMB02

IOMUXC_GPIO_SD_B0_05_LPUART8_RX

IOMUXC_GPIO_SD_B0_05_XBAR1_INOUT09

IOMUXC_GPIO_SD_B0_05_FLEXSPIB_DQS

IOMUXC_GPIO_SD_B0_05_GPIO3_IO17

IOMUXC_GPIO_SD_B0_05_CCM_CLKO2

IOMUXC_GPIO_SD_B0_05_ENET2_RX_EN

IOMUXC_GPIO_SD_B1_00_USDHC2_DATA3

IOMUXC_GPIO_SD_B1_00_FLEXSPIB_DATA03

IOMUXC_GPIO_SD_B1_00_FLEXPWM1_PWMA03

IOMUXC_GPIO_SD_B1_00_SAI1_TX_DATA03

IOMUXC_GPIO_SD_B1_00_LPUART4_TX

IOMUXC_GPIO_SD_B1_00_GPIO3_IO00

IOMUXC_GPIO_SD_B1_00_SAI3_RX_DATA

IOMUXC_GPIO_SD_B1_01_USDHC2_DATA2

IOMUXC_GPIO_SD_B1_01_FLEXSPIB_DATA02

IOMUXC_GPIO_SD_B1_01_FLEXPWM1_PWMB03

IOMUXC_GPIO_SD_B1_01_SAI1_TX_DATA02

IOMUXC_GPIO_SD_B1_01_LPUART4_RX

IOMUXC_GPIO_SD_B1_01_GPIO3_IO01

IOMUXC_GPIO_SD_B1_01_SAI3_TX_DATA

IOMUXC_GPIO_SD_B1_02_USDHC2_DATA1

IOMUXC_GPIO_SD_B1_02_FLEXSPIB_DATA01

IOMUXC_GPIO_SD_B1_02_FLEXPWM2_PWMA03

IOMUXC_GPIO_SD_B1_02_SAI1_TX_DATA01

IOMUXC_GPIO_SD_B1_02_FLEXCAN1_TX

IOMUXC_GPIO_SD_B1_02_GPIO3_IO02

IOMUXC_GPIO_SD_B1_02_CCM_WAIT

IOMUXC_GPIO_SD_B1_02_SAI3_TX_SYNC

IOMUXC_GPIO_SD_B1_03_USDHC2_DATA0

IOMUXC_GPIO_SD_B1_03_FLEXSPIB_DATA00

IOMUXC_GPIO_SD_B1_03_FLEXPWM2_PWMB03

IOMUXC_GPIO_SD_B1_03_SAI1_MCLK

IOMUXC_GPIO_SD_B1_03_FLEXCAN1_RX

IOMUXC_GPIO_SD_B1_03_GPIO3_IO03

IOMUXC_GPIO_SD_B1_03_CCM_PMIC_READY

IOMUXC_GPIO_SD_B1_03_SAI3_TX_BCLK

IOMUXC_GPIO_SD_B1_04_USDHC2_CLK

IOMUXC_GPIO_SD_B1_04_FLEXSPIB_SCLK

IOMUXC_GPIO_SD_B1_04_LPI2C1_SCL

IOMUXC_GPIO_SD_B1_04_SAI1_RX_SYNC

IOMUXC_GPIO_SD_B1_04_FLEXSPIA_SS1_B

IOMUXC_GPIO_SD_B1_04_GPIO3_IO04

IOMUXC_GPIO_SD_B1_04_CCM_STOP

IOMUXC_GPIO_SD_B1_04_SAI3_MCLK

IOMUXC_GPIO_SD_B1_05_USDHC2_CMD

IOMUXC_GPIO_SD_B1_05_FLEXSPIA_DQS

IOMUXC_GPIO_SD_B1_05_LPI2C1_SDA

IOMUXC_GPIO_SD_B1_05_SAI1_RX_BCLK

IOMUXC_GPIO_SD_B1_05_FLEXSPIB_SS0_B

IOMUXC_GPIO_SD_B1_05_GPIO3_IO05

IOMUXC_GPIO_SD_B1_05_SAI3_RX_SYNC

IOMUXC_GPIO_SD_B1_06_USDHC2_RESET_B

IOMUXC_GPIO_SD_B1_06_FLEXSPIA_SS0_B

IOMUXC_GPIO_SD_B1_06_LPUART7_CTS_B

IOMUXC_GPIO_SD_B1_06_SAI1_RX_DATA00

IOMUXC_GPIO_SD_B1_06_LPSPI2_PCS0

IOMUXC_GPIO_SD_B1_06_GPIO3_IO06

IOMUXC_GPIO_SD_B1_06_SAI3_RX_BCLK

IOMUXC_GPIO_SD_B1_07_SEMC_CSX01

IOMUXC_GPIO_SD_B1_07_FLEXSPIA_SCLK

IOMUXC_GPIO_SD_B1_07_LPUART7_RTS_B

IOMUXC_GPIO_SD_B1_07_SAI1_TX_DATA00

IOMUXC_GPIO_SD_B1_07_LPSPI2_SCK

IOMUXC_GPIO_SD_B1_07_GPIO3_IO07

IOMUXC_GPIO_SD_B1_08_USDHC2_DATA4

IOMUXC_GPIO_SD_B1_08_FLEXSPIA_DATA00

IOMUXC_GPIO_SD_B1_08_LPUART7_TX

IOMUXC_GPIO_SD_B1_08_SAI1_TX_BCLK

IOMUXC_GPIO_SD_B1_08_LPSPI2_SD0

IOMUXC_GPIO_SD_B1_08_GPIO3_IO08

IOMUXC_GPIO_SD_B1_08_SEMC_CSX02

IOMUXC_GPIO_SD_B1_09_USDHC2_DATA5

IOMUXC_GPIO_SD_B1_09_FLEXSPIA_DATA01

IOMUXC_GPIO_SD_B1_09_LPUART7_RX

IOMUXC_GPIO_SD_B1_09_SAI1_TX_SYNC

IOMUXC_GPIO_SD_B1_09_LPSPI2_SDI

IOMUXC_GPIO_SD_B1_09_GPIO3_IO09

IOMUXC_GPIO_SD_B1_10_USDHC2_DATA6

IOMUXC_GPIO_SD_B1_10_FLEXSPIA_DATA02

IOMUXC_GPIO_SD_B1_10_LPUART2_RX

IOMUXC_GPIO_SD_B1_10_LPI2C2_SDA

IOMUXC_GPIO_SD_B1_10_LPSPI2_PCS2

IOMUXC_GPIO_SD_B1_10_GPIO3_IO10

IOMUXC_GPIO_SD_B1_11_USDHC2_DATA7

IOMUXC_GPIO_SD_B1_11_FLEXSPIA_DATA03

IOMUXC_GPIO_SD_B1_11_LPUART2_TX

IOMUXC_GPIO_SD_B1_11_LPI2C2_SCL

IOMUXC_GPIO_SD_B1_11_LPSPI2_PCS3

IOMUXC_GPIO_SD_B1_11_GPIO3_IO11

IOMUXC_GPIO_SPI_B0_00_GPIO10_IO00

IOMUXC_GPIO_SPI_B0_01_FLEXSPI2_B_SCLK

IOMUXC_GPIO_SPI_B0_01_GPIO10_IO01

IOMUXC_GPIO_SPI_B0_02_FLEXSPI2_A_DATA00

IOMUXC_GPIO_SPI_B0_02_GPIO10_IO02

IOMUXC_GPIO_SPI_B0_03_FLEXSPI2_B_DATA02

IOMUXC_GPIO_SPI_B0_03_GPIO10_IO03

IOMUXC_GPIO_SPI_B0_04_FLEXSPI2_B_DATA03

IOMUXC_GPIO_SPI_B0_04_GPIO10_IO04

IOMUXC_GPIO_SPI_B0_05_FLEXSPI2_A_SS0_B

IOMUXC_GPIO_SPI_B0_05_GPIO10_IO05

IOMUXC_GPIO_SPI_B0_06_FLEXSPI2_A_DATA02

IOMUXC_GPIO_SPI_B0_06_GPIO10_IO06

IOMUXC_GPIO_SPI_B0_07_FLEXSPI2_B_DATA01

IOMUXC_GPIO_SPI_B0_07_GPIO10_IO07

IOMUXC_GPIO_SPI_B0_08_FLEXSPI2_A_SCLK

IOMUXC_GPIO_SPI_B0_08_GPIO10_IO08

IOMUXC_GPIO_SPI_B0_09_FLEXSPI2_A_DQS

IOMUXC_GPIO_SPI_B0_09_GPIO10_IO09

IOMUXC_GPIO_SPI_B0_10_FLEXSPI2_A_DATA03

IOMUXC_GPIO_SPI_B0_10_GPIO10_IO10

IOMUXC_GPIO_SPI_B0_11_FLEXSPI2_B_DATA00

IOMUXC_GPIO_SPI_B0_11_GPIO10_IO11

IOMUXC_GPIO_SPI_B0_12_FLEXSPI2_A_DATA01

IOMUXC_GPIO_SPI_B0_12_GPIO10_IO12

IOMUXC_GPIO_SPI_B0_13_GPIO10_IO13

IOMUXC_GPIO_SPI_B1_00_FLEXSPI2_A_DQS

IOMUXC_GPIO_SPI_B1_00_GPIO10_IO14

IOMUXC_GPIO_SPI_B1_01_FLEXSPI2_A_DATA03

IOMUXC_GPIO_SPI_B1_01_GPIO10_IO15

IOMUXC_GPIO_SPI_B1_02_FLEXSPI2_A_DATA02

IOMUXC_GPIO_SPI_B1_02_GPIO10_IO16

IOMUXC_GPIO_SPI_B1_03_FLEXSPI2_A_DATA01

IOMUXC_GPIO_SPI_B1_03_GPIO10_IO17

IOMUXC_GPIO_SPI_B1_04_FLEXSPI2_A_DATA00

IOMUXC_GPIO_SPI_B1_04_GPIO10_IO18

IOMUXC_GPIO_SPI_B1_05_FLEXSPI2_A_SCLK

IOMUXC_GPIO_SPI_B1_05_GPIO10_IO19

IOMUXC_GPIO_SPI_B1_06_FLEXSPI2_A_SS0_B

IOMUXC_GPIO_SPI_B1_06_GPIO10_IO20

IOMUXC_GPIO_SPI_B1_07_GPIO10_IO21

IOMUXC_SNVS_WAKEUP_GPIO5_IO00

IOMUXC_SNVS_WAKEUP_NMI

IOMUXC_SNVS_PMIC_ON_REQ_SNVS_PMIC_ON_REQ

IOMUXC_SNVS_PMIC_ON_REQ_GPIO5_IO01

IOMUXC_SNVS_PMIC_STBY_REQ_CCM_PMIC_STBY_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 2.0.3.

FSL_COMPONENT_ID

1.40 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.

void 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.

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”.

1.41 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.

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_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

CACHE_LINE_DATA

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_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.

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.

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

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_INVALID_IRQ_HANDLER
Invalid IRQ handler address.

1.42 Lin_lpuart_driver

FSL_LIN_LPUART_DRIVER_VERSION
LIN LPUART driver version.

enum _lin_lpuart_stop_bit_count
Values:

enumerator kLPUART_OneStopBit
One stop bit

enumerator kLPUART_TwoStopBit
Two stop bits

enum _lin_lpuart_flags
Values:

enumerator kLPUART_TxDataRegEmptyFlag
Transmit data register empty flag, sets when transmit buffer is empty

enumerator kLPUART_TransmissionCompleteFlag
Transmission complete flag, sets when transmission activity complete

enumerator kLPUART_RxDataRegFullFlag
Receive data register full flag, sets when the receive data buffer is full

enumerator kLPUART_IdleLineFlag
Idle line detect flag, sets when idle line detected

enumerator kLPUART_RxOverrunFlag
Receive Overrun, sets when new data is received before data is read from receive register

enumerator kLPUART_NoiseErrorFlag
Receive takes 3 samples of each received bit. If any of these samples differ, noise flag sets

enumerator kLPUART_FramingErrorFlag
Frame error flag, sets if logic 0 was detected where stop bit expected

enumerator kLPUART_ParityErrorFlag
If parity enabled, sets upon parity error detection

enumerator kLPUART_LinBreakFlag
LIN break detect interrupt flag, sets when LIN break char detected and LIN circuit enabled

enumerator kLPUART_RxActiveEdgeFlag
Receive pin active edge interrupt flag, sets when active edge detected

enumerator kLPUART_RxActiveFlag
Receiver Active Flag (RAF), sets at beginning of valid start bit

enumerator kLPUART_DataMatch1Flag
The next character to be read from LPUART_DATA matches MA1

enumerator kLPUART_DataMatch2Flag
The next character to be read from LPUART_DATA matches MA2

enumerator kLPUART_NoiseErrorInRxDataRegFlag
NOISY bit, sets if noise detected in current data word

enumerator kLPUART_ParityErrorInRxDataRegFlag
PARITY bit, sets if noise detected in current data word

enumerator kLPUART_TxFifoEmptyFlag
TXEMPT bit, sets if transmit buffer is empty

enumerator kLPUART_RxFifoEmptyFlag
RXEMPT bit, sets if receive buffer is empty

enumerator kLPUART_TxFifoOverflowFlag
TXOF bit, sets if transmit buffer overflow occurred

enumerator kLPUART_RxFifoUnderflowFlag
RXUF bit, sets if receive buffer underflow occurred

enum _lin_lpuart_interrupt_enable
Values:

enumerator kLPUART_LinBreakInterruptEnable
LIN break detect.

enumerator kLPUART_RxActiveEdgeInterruptEnable
Receive Active Edge.

enumerator kLPUART_TxDataRegEmptyInterruptEnable
Transmit data register empty.

enumerator kLPUART_TransmissionCompleteInterruptEnable
Transmission complete.

enumerator kLPUART_RxDataRegFullInterruptEnable
Receiver data register full.

enumerator kLPUART_IdleLineInterruptEnable
Idle line.

enumerator kLPUART_RxOverrunInterruptEnable
Receiver Overrun.

enumerator kLPUART_NoiseErrorInterruptEnable
Noise error flag.

enumerator kLPUART_FramingErrorInterruptEnable
Framing error flag.

enumerator kLPUART_ParityErrorInterruptEnable
Parity error flag.

enumerator kLPUART_TxFifoOverflowInterruptEnable
Transmit FIFO Overflow.

enumerator kLPUART_RxFifoUnderflowInterruptEnable
Receive FIFO Underflow.

enum _lin_lpuart_status
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.

enum lin_lpuart_bit_count_per_char_t
Values:

enumerator LPUART_8_BITS_PER_CHAR
8-bit data characters

enumerator LPUART_9_BITS_PER_CHAR
9-bit data characters

enumerator LPUART_10_BITS_PER_CHAR
10-bit data characters

typedef enum *_lin_lpuart_stop_bit_count* lin_lpuart_stop_bit_count_t

static inline bool LIN_LPUART_GetRxDataPolarity(const LPUART_Type *base)

static inline void LIN_LPUART_SetRxDataPolarity(LPUART_Type *base, bool polarity)

static inline void LIN_LPUART_WriteByte(LPUART_Type *base, uint8_t data)

static inline void LIN_LPUART_ReadByte(const LPUART_Type *base, uint8_t *readData)

status_t LIN_LPUART_CalculateBaudRate(LPUART_Type *base, uint32_t baudRate_Bps, uint32_t srcClock_Hz, uint32_t *osr, uint16_t *sbr)
Calculates the best osr and sbr value for configured baudrate.

Parameters

- base – LPUART peripheral base address
- baudRate_Bps – user configuration structure of type #lin_user_config_t
- srcClock_Hz – pointer to the LIN_LPUART driver state structure
- osr – pointer to osr value
- sbr – pointer to sbr value

Returns
An error code or lin_status_t

void LIN_LPUART_SetBaudRate(LPUART_Type *base, uint32_t *osr, uint16_t *sbr)
Configure baudrate according to osr and sbr value.

Parameters

- base – LPUART peripheral base address
- osr – pointer to osr value
- sbr – pointer to sbr value

lin_status_t LIN_LPUART_Init(LPUART_Type *base, lin_user_config_t *linUserConfig, lin_state_t *linCurrentState, uint32_t linSourceClockFreq)

Initializes an LIN_LPUART instance for LIN Network.

The caller provides memory for the driver state structures during initialization. The user must select the LIN_LPUART clock source in the application to initialize the LIN_LPUART. This function initializes a LPUART instance for operation. This function will initialize the run-time state structure to keep track of the on-going transfers, initialize the module to user defined settings and default settings, set break field length to be 13 bit times minimum, enable the break detect interrupt, Rx complete interrupt, frame error detect interrupt, and enable the LPUART module transmitter and receiver

Parameters

- base – LPUART peripheral base address
- linUserConfig – user configuration structure of type #lin_user_config_t
- linCurrentState – pointer to the LIN_LPUART driver state structure

Returns

An error code or lin_status_t

lin_status_t LIN_LPUART_Deinit(LPUART_Type *base)

Shuts down the LIN_LPUART by disabling interrupts and transmitter/receiver.

Parameters

- base – LPUART peripheral base address

Returns

An error code or lin_status_t

lin_status_t LIN_LPUART_SendFrameDataBlocking(LPUART_Type *base, const uint8_t *txBuff, uint8_t txSize, uint32_t timeoutMSec)

Sends Frame data out through the LIN_LPUART module using blocking method. This function will calculate the checksum byte and send it with the frame data. Blocking means that the function does not return until the transmission is complete.

Parameters

- base – LPUART peripheral base address
- txBuff – source buffer containing 8-bit data chars to send
- txSize – the number of bytes to send
- timeoutMSec – timeout value in milli seconds

Returns

An error code or lin_status_t

lin_status_t LIN_LPUART_SendFrameData(LPUART_Type *base, const uint8_t *txBuff, uint8_t txSize)

Sends frame data out through the LIN_LPUART module using non-blocking method. This enables an a-sync method for transmitting data. Non-blocking means that the function returns immediately. The application has to get the transmit status to know when the transmit is complete. This function will calculate the checksum byte and send it with the frame data.

Parameters

- base – LPUART peripheral base address
- txBuff – source buffer containing 8-bit data chars to send
- txSize – the number of bytes to send

Returns
An error code or lin_status_t

lin_status_t LIN_LPUART_GetTransmitStatus(LPUART_Type *base, uint8_t *bytesRemaining)

Get status of an on-going non-blocking transmission While sending frame data using non-blocking method, users can use this function to get status of that transmission. This function return LIN_TX_BUSY while sending, or LIN_TIMEOUT if timeout has occurred, or return LIN_SUCCESS when the transmission is complete. The bytesRemaining shows number of bytes that still needed to transmit.

Parameters

- base – LPUART peripheral base address
- bytesRemaining – Number of bytes still needed to transmit

Returns
lin_status_t LIN_TX_BUSY, LIN_SUCCESS or LIN_TIMEOUT

lin_status_t LIN_LPUART_RecvFrmDataBlocking(LPUART_Type *base, uint8_t *rxBuff, uint8_t rxSize, uint32_t timeoutMSec)

Receives frame data through the LIN_LPUART module using blocking method. This function will check the checksum byte. If the checksum is correct, it will receive the frame data. Blocking means that the function does not return until the reception is complete.

Parameters

- base – LPUART peripheral base address
- rxBuff – buffer containing 8-bit received data
- rxSize – the number of bytes to receive
- timeoutMSec – timeout value in milli seconds

Returns
An error code or lin_status_t

lin_status_t LIN_LPUART_RecvFrmData(LPUART_Type *base, uint8_t *rxBuff, uint8_t rxSize)

Receives frame data through the LIN_LPUART module using non-blocking method. This function will check the checksum byte. If the checksum is correct, it will receive it with the frame data. Non-blocking means that the function returns immediately. The application has to get the receive status to know when the reception is complete.

Parameters

- base – LPUART peripheral base address
- rxBuff – buffer containing 8-bit received data
- rxSize – the number of bytes to receive

Returns
An error code or lin_status_t

lin_status_t LIN_LPUART_AbortTransferData(LPUART_Type *base)

Aborts an on-going non-blocking transmission/reception. While performing a non-blocking transferring data, users can call this function to terminate immediately the transferring.

Parameters

- base – LPUART peripheral base address

Returns
An error code or lin_status_t

lin_status_t LIN_LPUART_GetReceiveStatus(LPUART_Type *base, uint8_t *bytesRemaining)

Get status of an on-going non-blocking reception While receiving frame data using non-blocking method, users can use this function to get status of that receiving. This function return the current event ID, LIN_RX_BUSY while receiving and return LIN_SUCCESS, or time-out (LIN_TIMEOUT) when the reception is complete. The bytesRemaining shows number of bytes that still needed to receive.

Parameters

- base – LPUART peripheral base address
- bytesRemaining – Number of bytes still needed to receive

Returns

lin_status_t LIN_RX_BUSY, LIN_TIMEOUT or LIN_SUCCESS

lin_status_t LIN_LPUART_GoToSleepMode(LPUART_Type *base)

This function puts current node to sleep mode This function changes current node state to LIN_NODE_STATE_SLEEP_MODE.

Parameters

- base – LPUART peripheral base address

Returns

An error code or lin_status_t

lin_status_t LIN_LPUART_GotoIdleState(LPUART_Type *base)

Puts current LIN node to Idle state This function changes current node state to LIN_NODE_STATE_IDLE.

Parameters

- base – LPUART peripheral base address

Returns

An error code or lin_status_t

lin_status_t LIN_LPUART_SendWakeupSignal(LPUART_Type *base)

Sends a wakeup signal through the LIN_LPUART interface.

Parameters

- base – LPUART peripheral base address

Returns

An error code or lin_status_t

lin_status_t LIN_LPUART_MasterSendHeader(LPUART_Type *base, uint8_t id)

Sends frame header out through the LIN_LPUART module using a non-blocking method. This function sends LIN Break field, sync field then the ID with correct parity.

Parameters

- base – LPUART peripheral base address
- id – Frame Identifier

Returns

An error code or lin_status_t

lin_status_t LIN_LPUART_EnableIRQ(LPUART_Type *base)

Enables LIN_LPUART hardware interrupts.

Parameters

- base – LPUART peripheral base address

Returns
An error code or lin_status_t

lin_status_t LIN_LPUART_DisableIRQ(LPUART_Type *base)

Disables LIN_LPUART hardware interrupts.

Parameters

- base – LPUART peripheral base address

Returns
An error code or lin_status_t

lin_status_t LIN_LPUART_AutoBaudCapture(uint32_t instance)

This function capture bits time to detect break char, calculate baudrate from sync bits and enable transceiver if autobaud successful. This function should only be used in Slave. The timer should be in mode input capture of both rising and falling edges. The timer input capture pin should be externally connected to RXD pin.

Parameters

- instance – LPUART instance

Returns
lin_status_t

void LIN_LPUART_IRQHandler(LPUART_Type *base)

LIN_LPUART RX TX interrupt handler.

Parameters

- base – LPUART peripheral base address

Returns
void

LIN_LPUART_TRANSMISSION_COMPLETE_TIMEOUT

Max loops to wait for LPUART transmission complete.

When de-initializing the LIN LPUART module, the program shall wait for the previous transmission to complete. This parameter defines how many loops to check completion before return error. If defined as 0, driver will wait forever until completion.

AUTOBAUD_BAUDRATE_TOLERANCE

BIT_RATE_TOLERANCE_UNSYNC

BIT_DURATION_MAX_19200

BIT_DURATION_MIN_19200

BIT_DURATION_MAX_14400

BIT_DURATION_MIN_14400

BIT_DURATION_MAX_9600

BIT_DURATION_MIN_9600

BIT_DURATION_MAX_4800

BIT_DURATION_MIN_4800

BIT_DURATION_MAX_2400

BIT_DURATION_MIN_2400

TWO_BIT_DURATION_MAX_19200

TWO_BIT_DURATION_MIN_19200

TWO_BIT_DURATION_MAX_14400

TWO_BIT_DURATION_MIN_14400

TWO_BIT_DURATION_MAX_9600

TWO_BIT_DURATION_MIN_9600

TWO_BIT_DURATION_MAX_4800

TWO_BIT_DURATION_MIN_4800

TWO_BIT_DURATION_MAX_2400

TWO_BIT_DURATION_MIN_2400

AUTOBAUD_BREAK_TIME_MIN

1.43 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 kLpi2cIrqs[]

Array to map LPI2C instance number to IRQ number, used internally for LPI2C master interrupt and EDMA transactional APIs.

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.

1.44 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_TransferRepeatedStartFlag
Send a repeated start condition

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.

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.

1.45 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[20]

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.

1.46 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.

1.47 LPSPI: Low Power Serial Peripheral Interface

1.48 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 calculates the values for 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 and the desired delay value in nano-seconds. The function calculates the value needed for the desired delay parameter and returns the actual calculated delay because an exact delay match may not be possible. In this case, the closest match is calculated without going below the desired delay value input. It is possible to input a very large delay value that exceeds the capability of the part, in which case the maximum supported delay is returned. It is up to the higher level peripheral driver to alert the user of an out of range delay input.

Note that the LPSPI module must be configured for master mode before configuring this. And note that the delayTime = LPSPI_clockSource / (PRESCALE * Delay_scaler).

Parameters

- base – LPSPI peripheral address.
- delayTimeInNanoSec – The desired delay value in nano-seconds.
- whichDelay – The desired delay to configuration, which must be of type lpspi_delay_type_t.
- srcClock_Hz – Module source input clock in Hertz.

Returns

actual Calculated delay value in nano-seconds.

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.

1.49 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

1.50 LPUART: Low Power Universal Asynchronous Receiver/Transmitter Driver

1.51 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

void 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.

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_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_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 enableTxCTS
TX CTS enable

lpuart_transmit_cts_source_t txCtsSource
TX CTS source

lpuart_transmit_cts_config_t txCtsConfig
TX CTS configure

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

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 ___unnamed69___

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 ___unnamed71___

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 ___unnamed73___

Public Members

uint8_t *volatile rxData
Address of remaining data to receive.

uint16_t *volatile rxData16
Address of remaining data to receive.

union ___unnamed75___

Public Members

uint8_t *rxRingBuffer
Start address of the receiver ring buffer.

uint16_t *rxRingBuffer16
Start address of the receiver ring buffer.

1.52 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

1.53 Nic301

enum __nic__reg

Values:

enumerator kNIC__REG__READ__QOS__LCD

enumerator kNIC__REG__READ__QOS__CSI

enumerator kNIC__REG__READ__QOS__PXP

enumerator kNIC__REG__READ__QOS__DCP

enumerator kNIC__REG__READ__QOS__ENET

enumerator kNIC__REG__READ__QOS__USBO2

enumerator kNIC__REG__READ__QOS__USDHC1

enumerator kNIC__REG__READ__QOS__USDHC2

enumerator kNIC__REG__READ__QOS__TestPort

enumerator kNIC__REG__READ__QOS__ENET2

enumerator kNIC__REG__READ__QOS__CM7

enumerator kNIC_REG_READ_QOS_DMA

enumerator kNIC_REG_WRITE_QOS_LCD

enumerator kNIC_REG_WRITE_QOS_CSI

enumerator kNIC_REG_WRITE_QOS_PXP

enumerator kNIC_REG_WRITE_QOS_DCP

enumerator kNIC_REG_WRITE_QOS_ENET

enumerator kNIC_REG_WRITE_QOS_USBO2

enumerator kNIC_REG_WRITE_QOS_USDHC1

enumerator kNIC_REG_WRITE_QOS_USDHC2

enumerator kNIC_REG_WRITE_QOS_TestPort

enumerator kNIC_REG_WRITE_QOS_ENET2

enumerator kNIC_REG_WRITE_QOS_CM7

enumerator kNIC_REG_WRITE_QOS_DMA

enumerator kNIC_REG_FN_MOD_LCD

enumerator kNIC_REG_FN_MOD_CSI

enumerator kNIC_REG_FN_MOD_PXP

enumerator kNIC_REG_FN_MOD_DCP

enumerator kNIC_REG_FN_MOD_ENET

enumerator kNIC_REG_FN_MOD_USBO2

enumerator kNIC_REG_FN_MOD_USDHC1

enumerator kNIC_REG_FN_MOD_USDHC2

enumerator kNIC_REG_FN_MOD_TestPort

enumerator kNIC_REG_FN_MOD_ENET2

enumerator kNIC_REG_FN_MOD_CM7

enumerator kNIC_REG_FN_MOD_DMA

enumerator kNIC_REG_FN_MOD2_DCP

enumerator kNIC_REG_FN_MOD_AHB_ENET

enumerator kNIC_REG_FN_MOD_AHB_TestPort

enumerator kNIC_REG_FN_MOD_AHB_ENET2

enumerator kNIC_REG_FN_MOD_AHB_DMA

enumerator kNIC_REG_WR_TIDEMARK_CM7

enum __nic_fn_mod2

Values:

enumerator kNIC_FN_MOD2_ENABLE

enumerator kNIC_FN_MOD2_BYPASS

enum _nic_fn_mod_ahb

Values:

enumerator kNIC_FN_MOD_AHB_RD_INCR_OVERRIDE

enumerator kNIC_FN_MOD_AHB_WR_INCR_OVERRIDE

enumerator kNIC_FN_MOD_AHB_LOCK_OVERRIDE

enum _nic_fn_mod

Values:

enumerator kNIC_FN_MOD_ReadIssue

enumerator kNIC_FN_MOD_WriteIssue

enum _nic_qos

Values:

enumerator kNIC_QOS_0

enumerator kNIC_QOS_1

enumerator kNIC_QOS_2

enumerator kNIC_QOS_3

enumerator kNIC_QOS_4

enumerator kNIC_QOS_5

enumerator kNIC_QOS_6

enumerator kNIC_QOS_7

enumerator kNIC_QOS_8

enumerator kNIC_QOS_9

enumerator kNIC_QOS_10

enumerator kNIC_QOS_11

enumerator kNIC_QOS_12

enumerator kNIC_QOS_13

enumerator kNIC_QOS_14

enumerator kNIC_QOS_15

typedef enum *_nic_reg* nic_reg_t

typedef enum *_nic_fn_mod2* nic_fn_mod2_t

typedef enum *_nic_fn_mod_ahb* nic_fn_mod_ahb_t

typedef enum *_nic_fn_mod* nic_fn_mod_t

typedef enum *_nic_qos* nic_qos_t

static inline void NIC_SetReadQos(*nic_reg_t* base, *nic_qos_t* value)

Set read_qos Value.

Parameters

- base – Base address of GPV address
- value – Target value (0 - 15)

static inline *nic_qos_t* NIC_GetReadQos(*nic_reg_t* base)

Get read_qos Value.

Parameters

- base – Base address of GPV address

Returns

Current value configured

static inline void NIC_SetWriteQos(*nic_reg_t* base, *nic_qos_t* value)

Set write_qos Value.

Parameters

- base – Base address of GPV address
- value – Target value (0 - 15)

static inline *nic_qos_t* NIC_GetWriteQos(*nic_reg_t* base)

Get write_qos Value.

Parameters

- base – Base address of GPV address

Returns

Current value configured

static inline void NIC_SetFnModAhb(*nic_reg_t* base, *nic_fn_mod_ahb_t* value)

Set fn_mod_ahb Value.

Parameters

- base – Base address of GPV address
- value – Target value

static inline *nic_fn_mod_ahb_t* NIC_GetFnModAhb(*nic_reg_t* base)

Get fn_mod_ahb Value.

Parameters

- base – Base address of GPV address

Returns

Current value configured

static inline void NIC_SetWrTideMark(*nic_reg_t* base, uint8_t value)

Set wr_tidemark Value.

Parameters

- base – Base address of GPV address
- value – Target value (0 - 7)

static inline uint8_t NIC_GetWrTideMark(*nic_reg_t* base)

Get wr_tidemark Value.

Parameters

- base – Base address of GPV address

Returns
Current value configured

static inline void NIC_SetFnMod(*nic_reg_t* base, *nic_fn_mod_t* value)
Set fn_mod Value.

Parameters

- base – Base address of GPV address
- value – Target value

static inline *nic_fn_mod_t* NIC_GetFnMod(*nic_reg_t* base)
Get fn_mod Value.

Parameters

- base – Base address of GPV address

Returns
Current value configured

static inline void NIC_SetFnMod2(*nic_reg_t* base, *nic_fn_mod_t* value)
Set fn_mod2 Value.

Parameters

- base – Base address of GPV address
- value – Target value

static inline *nic_fn_mod2_t* NIC_GetFnMod2(*nic_reg_t* base)
Get fn_mod2 Value.

Parameters

- base – Base address of GPV address

Returns
Current value configured

FSL_NIC301_DRIVER_VERSION
NIC301 driver version 2.0.1.

GPV0_BASE

GPV1_BASE

GPV4_BASE

NIC_FN_MOD2_OFFSET

NIC_FN_MOD_AHB_OFFSET

NIC_WR_TIDEMARK_OFFSET

NIC_READ_QOS_OFFSET

NIC_WRITE_QOS_OFFSET

NIC_FN_MOD_OFFSET

NIC_LCD_BASE

NIC_CSI_BASE

NIC_PXP_BASE

NIC_DCP_BASE

NIC_ENET_BASE

NIC_USBO2_BASE

NIC_USDHC1_BASE

NIC_USDHC2_BASE

NIC_TestPort_BASE

NIC_ENET2_BASE

NIC_CM7_BASE

NIC_DMA_BASE

NIC_QOS_MASK

NIC_WR_TIDEMARK_MASK

NIC_FN_MOD_AHB_MASK

NIC_FN_MOD_MASK

NIC_FN_MOD2_MASK

FSL_COMPONENT_ID

1.54 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_ReadFuseShadowRegister(OCOTP_Type *base, uint32_t address)

Read the fuse shadow register with the fuse addess.

Deprecated:

Use OCOTP_ReadFuseShadowRegisterExt 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_ReadFuseShadowRegisterExt(OCOTP_Type *base, uint32_t address, uint32_t *data, uint8_t fuseWords)

Read the fuse shadow register from the fuse addess.

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_WriteFuseShadowRegister(OCOTP_Type *base, uint32_t address, uint32_t data)

Write the fuse shadow register with the fuse addess and data. Please make sure the wrtie address is not locked while calling this API.

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_WriteFuseShadowRegisterWithLock(OCOTP_Type *base, uint32_t address, uint32_t data, bool lock)

Write the fuse shadow register and lock it.

Please make sure the wrtie 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.

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 shadow register 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

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.

1.55 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

1.56 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

static inline void PMU_GatePower(PMU_Type *base, uint32_t gates)

Gate the power to modules.

Parameters

- base – PMU peripheral base address.
- gates – Mask value for the module to be gated. See to _pmu_power_gate.

static inline void PMU_UngatePower(PMU_Type *base, uint32_t gates)

Ungate the power to modules.

Parameters

- base – PMU peripheral base address.
- gates – Mask value for the module to be gated. See to _pmu_power_gate.

static inline void PMU_EnableLowPowerBandgap(PMU_Type *base, bool enable)

Enable the low power bandgap.

Parameters

- base – PMU peripheral base address.
- enable – Enable the low power bandgap or use the normal power bandgap. @

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_gate

Mask values of power gate.

Values:

enumerator kPMU_PowerGateDisplay

Display power gate control.

enumerator kPMU_PowerGateDisplayLogic

Display logic power gate control.

enumerator kPMU_PowerGateL2

L2 power gate control.

enumerator kPMU_PowerGateL1

L1 power gate control.

enumerator kPMU_PowerGateRefTopIBias

Low power reftop ibias disable.

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.

1.57 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;
```

(continues on next page)

(continued from previous page)

```
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 2.9.0

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
The IPBus clock is used as the clock

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

enumerator kPWM_faultchannel_1

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

void PWM_SetupFaultInputFilter(PWM_Type *base, const *pwm_fault_input_filter_param_t* *faultInputFilterParams)

Sets up the PWM fault input filter.

Parameters

- base – PWM peripheral base address
- 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 protection.

PWM has 4 fault inputs.

Parameters

- base – PWM peripheral base address
- 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_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

1.58 PXP: Pixel Pipeline

void PXP_Init(PXP_Type *base)

Initialize the PXP.

This function enables the PXP peripheral clock, and resets the PXP registers to default status.

Parameters

- base – PXP peripheral base address.

void PXP_Deinit(PXP_Type *base)

De-initialize the PXP.

This function disables the PXP peripheral clock.

Parameters

- base – PXP peripheral base address.

void PXP_Reset(PXP_Type *base)

Reset the PXP.

This function resets the PXP peripheral registers to default status.

Parameters

- base – PXP peripheral base address.

void PXP_ResetControl(PXP_Type *base)

Reset the PXP and the control register to initialized state.

Parameters

- base – PXP peripheral base address.

static inline void PXP_Start(PXP_Type *base)

Start process.

Start PXP process using current configuration.

Parameters

- base – PXP peripheral base address.

static inline void PXP_EnableLcdHandShake(PXP_Type *base, bool enable)

Enable or disable LCD hand shake.

Parameters

- base – PXP peripheral base address.
- enable – True to enable, false to disable.

static inline void PXP_EnableContinousRun(PXP_Type *base, bool enable)

Enable or disable continous run.

If continous run not enabled, PXP_Start starts the PXP process. When completed, PXP enters idle mode and flag kPXP_CompleteFlag asserts.

If continous run enabled, the PXP will repeat based on the current configuration register settings.

Parameters

- base – PXP peripheral base address.
- enable – True to enable, false to disable.

static inline void PXP_SetProcessBlockSize(PXP_Type *base, *pxp_block_size_t* size)

Set the PXP processing block size.

This function chooses the pixel block size that PXP using during process. Larger block size means better performace, but be careful that when PXP is rotating, the output must be divisible by the block size selected.

Parameters

- base – PXP peripheral base address.
- size – The pixel block size.

static inline void PXP_EnableProcessEngine(PXP_Type *base, uint32_t mask, bool enable)

Enables or disables PXP engines in the process flow.

Parameters

- base – PXP peripheral base address.
- mask – The engines to enable. Logical OR of pxp_process_engine_name_t.

- enable – true to enable, false to disable.

static inline uint32_t PXP_GetStatusFlags(PXP_Type *base)

Gets PXP status flags.

This function gets all PXP status flags. The flags are returned as the logical OR value of the enumerators _pxp_flags. To check a specific status, compare the return value with enumerators in _pxp_flags. For example, to check whether the PXP has completed process, use like this:

```
if (kPXP_CompleteFlag & PXP_GetStatusFlags(PXP))
{
    ...
}
```

Parameters

- base – PXP peripheral base address.

Returns

PXP status flags which are OR'ed by the enumerators in the _pxp_flags.

static inline void PXP_ClearStatusFlags(PXP_Type *base, uint32_t statusMask)

Clears status flags with the provided mask.

This function clears PXP status flags with a provided mask.

Parameters

- base – PXP peripheral base address.
- statusMask – The status flags to be cleared; it is logical OR value of _pxp_flags.

static inline uint8_t PXP_GetAxiErrorId(PXP_Type *base, uint8_t axiIndex)

Gets the AXI ID of the failing bus operation.

Parameters

- base – PXP peripheral base address.
- axiIndex – Whitch AXI to get
  - 0: AXI0
  - 1: AXI1

Returns

The AXI ID of the failing bus operation.

static inline void PXP_EnableInterrupts(PXP_Type *base, uint32_t mask)

Enables PXP interrupts according to the provided mask.

This function enables the PXP interrupts according to the provided mask. The mask is a logical OR of enumeration members. See _pxp_interrupt_enable. For example, to enable PXP process complete interrupt and command loaded interrupt, do the following.

```
PXP_EnableInterrupts(PXP, kPXP_CommandLoadInterruptEnable | kPXP_
↪CompleteInterruptEnable);
```

Parameters

- base – PXP peripheral base address.
- mask – The interrupts to enable. Logical OR of _pxp_interrupt_enable.

static inline void PXP_DisableInterrupts(PXP_Type *base, uint32_t mask)

Disables PXP interrupts according to the provided mask.

This function disables the PXP interrupts according to the provided mask. The mask is a logical OR of enumeration members. See _pxp_interrupt_enable.

Parameters

- base – PXP peripheral base address.
- mask – The interrupts to disable. Logical OR of _pxp_interrupt_enable.

void PXP_SetAlphaSurfaceBufferConfig(PXP_Type *base, const *pxp_as_buffer_config_t* *config)

Set the alpha surface input buffer configuration.

Parameters

- base – PXP peripheral base address.
- config – Pointer to the configuration.

void PXP_SetAlphaSurfaceBlendConfig(PXP_Type *base, const *pxp_as_blend_config_t* *config)

Set the alpha surface blending configuration.

Parameters

- base – PXP peripheral base address.
- config – Pointer to the configuration structure.

void PXP_SetAlphaSurfaceBlendSecondaryConfig(PXP_Type *base, const *pxp_as_blend_secondary_config_t* *config)

Set the alpha surface blending configuration for the secondary engine.

Parameters

- base – PXP peripheral base address.
- config – Pointer to the configuration structure.

void PXP_SetAlphaSurfaceOverlayColorKey(PXP_Type *base, uint8_t num, uint32_t colorKeyLow, uint32_t colorKeyHigh)

Set the alpha surface overlay color key.

If a pixel in the current overlay image with a color that falls in the range from the colorKeyLow to colorKeyHigh range, it will use the process surface pixel value for that location. If no PS image is present or if the PS image also matches its colorkey range, the PS background color is used.

Note: Colorkey operations are higher priority than alpha or ROP operations

Parameters

- base – PXP peripheral base address.
- num – instance number. 0 for alpha engine A, 1 for alpha engine B.
- colorKeyLow – Color key low range.
- colorKeyHigh – Color key high range.

static inline void PXP_EnableAlphaSurfaceOverlayColorKey(PXP_Type *base, uint32_t num, bool enable)

Enable or disable the alpha surface color key.

Parameters

- base – PXP peripheral base address.
- num – instance number. 0 for alpha engine A, 1 for alpha engine B.
- enable – True to enable, false to disable.

void PXP_SetAlphaSurfacePosition(PXP_Type *base, uint16_t upperLeftX, uint16_t upperLeftY, uint16_t lowerRightX, uint16_t lowerRightY)

Set the alpha surface position in output buffer.

Parameters

- base – PXP peripheral base address.
- upperLeftX – X of the upper left corner.
- upperLeftY – Y of the upper left corner.
- lowerRightX – X of the lower right corner.
- lowerRightY – Y of the lower right corner.

static inline void PXP_SetProcessSurfaceBackGroundColor(PXP_Type *base, uint8_t num, uint32_t backGroundColor)

Set the back ground color of PS.

Parameters

- base – PXP peripheral base address.
- num – instance number. 0 for alpha engine A, 1 for alpha engine B.
- backGroundColor – Pixel value of the background color.

void PXP_SetProcessSurfaceBufferConfig(PXP_Type *base, const *pxp_ps_buffer_config_t* *config)

Set the process surface input buffer configuration.

Parameters

- base – PXP peripheral base address.
- config – Pointer to the configuration.

void PXP_SetProcessSurfaceScaler(PXP_Type *base, uint16_t inputWidth, uint16_t inputHeight, uint16_t outputWidth, uint16_t outputHeight)

Set the process surface scaler configuration.

The valid down scale fact is 1/(2^12) ~ 16.

Parameters

- base – PXP peripheral base address.
- inputWidth – Input image width.
- inputHeight – Input image height.
- outputWidth – Output image width.
- outputHeight – Output image height.

void PXP_SetProcessSurfacePosition(PXP_Type *base, uint16_t upperLeftX, uint16_t upperLeftY, uint16_t lowerRightX, uint16_t lowerRightY)

Set the process surface position in output buffer.

Parameters

- base – PXP peripheral base address.
- upperLeftX – X of the upper left corner.
- upperLeftY – Y of the upper left corner.

- lowerRightX – X of the lower right corner.
- lowerRightY – Y of the lower right corner.

void PXP_SetProcessSurfaceColorKey(PXP_Type *base, uint8_t num, uint32_t colorKeyLow, uint32_t colorKeyHigh)

Set the process surface color key.

If the PS image matches colorkey range, the PS background color is output. Set colorKeyLow to 0xFFFFFFFF and p colorKeyHigh to 0 will disable the colorkeying.

Parameters

- base – PXP peripheral base address.
- num – instance number. 0 for alpha engine A, 1 for alpha engine B.
- colorKeyLow – Color key low range.
- colorKeyHigh – Color key high range.

static inline void PXP_SetProcessSurfaceYUVFormat(PXP_Type *base, *pxp_ps_yuv_format_t* format)

Set the process surface input pixel format YUV or YCbCr.

If process surface input pixel format is YUV and CSC1 is not enabled, in other words, the process surface output pixel format is also YUV, then this function should be called to set whether input pixel format is YUV or YCbCr.

Parameters

- base – PXP peripheral base address.
- format – The YUV format.

void PXP_SetOutputBufferConfig(PXP_Type *base, const *pxp_output_buffer_config_t* *config)

Set the PXP outpt buffer configuration.

Parameters

- base – PXP peripheral base address.
- config – Pointer to the configuration.

static inline void PXP_SetOverwrittenAlphaValue(PXP_Type *base, uint8_t alpha)

Set the global overwritten alpha value.

If global overwritten alpha is enabled, the alpha component in output buffer pixels will be overwritten, otherwise the computed alpha value is used.

Parameters

- base – PXP peripheral base address.
- alpha – The alpha value.

static inline void PXP_EnableOverWrittenAlpha(PXP_Type *base, bool enable)

Enable or disable the global overwritten alpha value.

If global overwritten alpha is enabled, the alpha component in output buffer pixels will be overwritten, otherwise the computed alpha value is used.

Parameters

- base – PXP peripheral base address.
- enable – True to enable, false to disable.

static inline void PXP_SetRotateConfig(PXP_Type *base, *pxp_rotate_position_t* position, *pxp_rotate_degree_t* degree, *pxp_flip_mode_t* flipMode)

Set the rotation configuration.

The PXP could rotate the process surface or the output buffer. There are two PXP versions:

- Version 1: Only has one rotate sub module, the output buffer and process surface share the same rotate sub module, which means the process surface and output buffer could not be rotate at the same time. When pass in kPXP_RotateOutputBuffer, the process surface could not use the rotate, Also when pass in kPXP_RotateProcessSurface, output buffer could not use the rotate.
- Version 2: Has two seperate rotate sub modules, the output buffer and process surface could configure the rotation independently.

Upper layer could use the macro PXP_SHARE_ROTATE to check which version is. PXP_SHARE_ROTATE=1 means version 1.

Note: This function is different depends on the macro PXP_SHARE_ROTATE.

Parameters

- base – PXP peripheral base address.
- position – Rotate process surface or output buffer.
- degree – Rotate degree.
- flipMode – Flip mode.

void PXP_BuildRect(PXP_Type *base, *pxp_output_pixel_format_t* outFormat, uint32_t value, uint16_t width, uint16_t height, uint16_t pitch, uint32_t outAddr)

Build a solid rectangle of given pixel value.

Parameters

- base – PXP peripheral base address.
- outFormat – output pixel format.
- value – The value of the pixel to be filled in the rectangle in ARGB8888 format.
- width – width of the rectangle.
- height – height of the rectangle.
- pitch – output pitch in byte.
- outAddr – address of the memory to store the rectangle.

void PXP_SetNextCommand(PXP_Type *base, void *commandAddr)

Set the next command.

The PXP supports a primitive ability to queue up one operation while the current operation is running. Workflow:

a. Prepare the PXP register values except STAT, CSCCOEFn, NEXT in the memory in the order they appear in the register map.
b. Call this function sets the new operation to PXP.
c. There are two methods to check whether the PXP has loaded the new operation. The first method is using PXP_IsNextCommandPending. If there is new operation not loaded by the PXP, this function returns true. The second method is checking the flag

kPXP_CommandLoadFlag, if command loaded, this flag asserts. User could enable interrupt kPXP_CommandLoadInterruptEnable to get the loaded signal in interrupt way.

d. When command loaded by PXP, a new command could be set using this function.

```
uint32_t pxp_command1[48];
uint32_t pxp_command2[48];

pxp_command1[0] = ...;
pxp_command1[1] = ...;
...
pxp_command2[0] = ...;
pxp_command2[1] = ...;
...

while (PXP_IsNextCommandPending(PXP))
{
}

PXP_SetNextCommand(PXP, pxp_command1);

while (PXP_IsNextCommandPending(PXP))
{
}

PXP_SetNextCommand(PXP, pxp_command2);
```

Parameters

- base – PXP peripheral base address.
- commandAddr – Address of the new command.

static inline bool PXP_IsNextCommandPending(PXP_Type *base)

Check whether the next command is pending.

Parameters

- base – UART peripheral base address.

Returns

True is pending, false is not.

static inline void PXP_CancelNextCommand(PXP_Type *base)

Cancel command set by PXP_SetNextCommand.

Parameters

- base – UART peripheral base address.

void PXP_SetCsc1Mode(PXP_Type *base, *pxp_csc1_mode_t* mode)

Set the CSC1 mode.

The CSC1 module receives scaled YUV/YCbCr444 pixels from the scale engine and converts the pixels to the RGB888 color space. It could only be used by process surface.

Parameters

- base – PXP peripheral base address.
- mode – The conversion mode.

static inline void PXP_EnableCsc1(PXP_Type *base, bool enable)

Enable or disable the CSC1.

Parameters

- base – PXP peripheral base address.

- enable – True to enable, false to disable.

void PXP_SetInternalRamData(PXP_Type *base, *pxp_ram_t* ram, uint32_t bytesNum, uint8_t *data, uint16_t memStartAddr)

Write data to the PXP internal memory.

Parameters

- base – PXP peripheral base address.
- ram – Which internal memory to write.
- bytesNum – How many bytes to write.
- data – Pointer to the data to write.
- memStartAddr – The start address in the internal memory to write the data.

void PXP_SetDitherFinalLutData(PXP_Type *base, const *pxp_dither_final_lut_data_t* *data)

Set the dither final LUT data.

The dither final LUT is only applicble to dither engine 0. It takes the bits[7:4] of the output pixel and looks up and 8 bit value from the 16 value LUT to generate the final output pixel to the next process module.

Parameters

- base – PXP peripheral base address.
- data – Pointer to the LUT data to set.

static inline void PXP_SetDitherConfig(PXP_Type *base, const *pxp_dither_config_t* *config)

Set the configuration for the dither block.

If the pre-dither LUT, post-dither LUT or ordered dither is used, please call PXP_SetInternalRamData to set the LUT data to internal memory.

If the final LUT is used, please call PXP_SetDitherFinalLutData to set the LUT data.

Note: When using ordered dithering, please set the PXP process block size same with the ordered dithering matrix size using function PXP_SetProcessBlockSize.

Parameters

- base – PXP peripheral base address.
- config – Pointer to the configuration.

void PXP_EnableDither(PXP_Type *base, bool enable)

Enable or disable dither engine in the PXP process path.

After the initialize function PXP_Init, the dither engine is disabled and not use in the PXP processing path. This function enables the dither engine and routes the dither engine output to the output buffer. When the dither engine is enabled using this function, PXP_SetDitherConfig must be called to configure dither engine correctly, otherwise there is not output to the output buffer.

Parameters

- base – PXP peripheral base address.
- enable – Pass in true to enable, false to disable.

void PXP_SetPorterDuffConfig(PXP_Type *base, uint8_t num, const *pxp_porter_duff_config_t* *config)

Set the Porter Duff configuration for one of the alpha process engine.

Parameters

- base – PXP peripheral base address.
- num – instance number.
- config – Pointer to the configuration.

status_t PXP_GetPorterDuffConfigExt(*pxp_porter_duff_blend_mode_t* mode, *pxp_porter_duff_config_t* *config, uint8_t dstGlobalAlphaMode, uint8_t dstAlphaMode, uint8_t dstColorMode, uint8_t srcGlobalAlphaMode, uint8_t srcAlphaMode, uint8_t srcColorMode, uint8_t dstGlobalAlpha, uint8_t srcGlobalAlpha)

Get the Porter Duff configuration.

The FactorMode are selected based on blend mode, the other values are set based on input parameters. These values could be modified after calling this function. This function is extened PXP_GetPorterDuffConfig.

Parameters

- mode – The blend mode.
- config – Pointer to the configuration.
- dstGlobalAlphaMode – Destination layer (or PS, s0) global alpha mode, see pxp_porter_duff_global_alpha_mode
- dstAlphaMode – Destination layer (or PS, s0) alpha mode, see pxp_porter_duff_alpha_mode.
- dstColorMode – Destination layer (or PS, s0) color mode, see pxp_porter_duff_color_mode.
- srcGlobalAlphaMode – Source layer (or AS, s1) global alpha mode, see pxp_porter_duff_global_alpha_mode
- srcAlphaMode – Source layer (or AS, s1) alpha mode, see pxp_porter_duff_alpha_mode.
- srcColorMode – Source layer (or AS, s1) color mode, see pxp_porter_duff_color_mode.
- dstGlobalAlpha – Destination layer (or PS, s0) global alpha value, 0~255
- srcGlobalAlpha – Source layer (or AS, s1) global alpha value, 0~255

Return values

- kStatus_Success – Successfully get the configuratoin.
- kStatus_InvalidArgument – The blend mode not supported.

static inline *status_t* PXP_GetPorterDuffConfig(*pxp_porter_duff_blend_mode_t* mode, *pxp_porter_duff_config_t* *config)

Get the Porter Duff configuration by blend mode.

The FactorMode are selected based on blend mode, the AlphaMode are set to kPXP_PorterDuffAlphaStraight, the ColorMode are set to kPXP_PorterDuffColorWithAlpha, the GlobalAlphaMode are set to kPXP_PorterDuffLocalAlpha. These values could be modified after calling this function.

Parameters

- mode – The blend mode.
- config – Pointer to the configuration.

Return values

- kStatus_Success – Successfully get the configuratoin.
- kStatus_InvalidArgument – The blend mode not supported.

FSL_PXP_DRIVER_VERSION

enum _pxp_interrupt_enable

PXP interrupts to enable.

Values:

enumerator kPXP_CompleteInterruptEnable

PXP process completed. bit 1

enumerator kPXP_CommandLoadInterruptEnable

Interrupt to show that the command set by PXP_SetNextCommand has been loaded. bit 2

enumerator kPXP_CompressDoneInterruptEnable

Compress done interrupt enable. bit 15

enumerator kPXP_InputFetchCh0InterruptEnable

Input fetch channel 0 completed. bit 16

enumerator kPXP_InputFetchCh1InterruptEnable

Input fetch channel 1 completed. bit 17

enumerator kPXP_InputStoreCh0InterruptEnable

Input store channel 0 completed. bit 18

enumerator kPXP_InputStoreCh1InterruptEnable

Input store channel 1 completed. bit 19

enumerator kPXP_DitherFetchCh0InterruptEnable

Dither fetch channel 0 completed. bit 20

enumerator kPXP_DitherFetchCh1InterruptEnable

Dither fetch channel 1 completed. bit 21

enumerator kPXP_DitherStoreCh0InterruptEnable

Dither store channle 0 completed. bit 22

enumerator kPXP_DitherStoreCh1InterruptEnable

Dither store channle 1 completed. bit 23

enumerator kPXP_WfeaStoreCh0InterruptEnable

WFE-A store channel 0 completed. bit 24

enumerator kPXP_WfeaStoreCh1InterruptEnable

WFE-A store channel 1 completed. bit 25

enumerator kPXP_WfebStoreCh0InterruptEnable

WFE-B store channel 0 completed. bit 26

enumerator kPXP_WfebStoreCh1InterruptEnable

WFE-B store channel 1 completed. bit 27

enumerator kPXP_InputStoreInterruptEnable

Input store completed. bit 28

enumerator kPXP_DitherStoreInterruptEnable

Dither store completed. bit 29

enumerator kPXP_WfeaStoreInterruptEnable
WFE-A store completed. bit 30

enumerator kPXP_WfebStoreInterruptEnable
WFE-B store completed. bit 31

enum _pxp_flags
PXP status flags.

Note: These enumerations are meant to be OR'd together to form a bit mask.

Values:

enumerator kPXP_CompleteFlag
PXP process completed. bit 0

enumerator kPXP_Axi0WriteErrorFlag
PXP encountered an AXI write error and processing has been terminated. bit 1

enumerator kPXP_Axi0ReadErrorFlag
PXP encountered an AXI read error and processing has been terminated. bit 2

enumerator kPXP_CommandLoadFlag
The command set by PXP_SetNextCommand has been loaded, could set new command. bit 3

enumerator kPXP_CompressDoneFlag
Compress done. bit 15

enumerator kPXP_InputFetchCh0CompleteFlag
Input fetch channel 0 completed. bit 16

enumerator kPXP_InputFetchCh1CompleteFlag
Input fetch channel 1 completed. bit 17

enumerator kPXP_InputStoreCh0CompleteFlag
Input store channel 0 completed. bit 18

enumerator kPXP_InputStoreCh1CompleteFlag
Input store channel 1 completed. bit 19

enumerator kPXP_DitherFetchCh0CompleteFlag
Dither fetch channel 0 completed. bit 20

enumerator kPXP_DitherFetchCh1CompleteFlag
Dither fetch channel 1 completed. bit 21

enumerator kPXP_DitherStoreCh0CompleteFlag
Dither store channel 0 completed. bit 22

enumerator kPXP_DitherStoreCh1CompleteFlag
Dither store channel 1 completed. bit 23

enumerator kPXP_WfeaStoreCh0CompleteFlag
WFE-A store channel 0 completed. bit 24

enumerator kPXP_WfeaStoreCh1CompleteFlag
WFE-A store channel 1 completed. bit 25

enumerator kPXP_WfebStoreCh0CompleteFlag
WFE-B store channel 0 completed. bit 26

enumerator kPXP_WfebStoreCh1CompleteFlag
WFE-B store channel 1 completed. bit 27

enumerator kPXP_InputStoreCompleteFlag
Input store completed. bit 28

enumerator kPXP_DitherStoreCompleteFlag
Dither store completed. bit 29

enumerator kPXP_WfeaStoreCompleteFlag
WFE-A store completed. bit 30

enumerator kPXP_WfebStoreCompleteFlag
WFE-B store completed. bit 31

enum _pxp_flip_mode
PXP output flip mode.

Values:

enumerator kPXP_FlipDisable
Flip disable.

enumerator kPXP_FlipHorizontal
Horizontal flip.

enumerator kPXP_FlipVertical
Vertical flip.

enumerator kPXP_FlipBoth
Flip both directions.

enum _pxp_rotate_position
PXP rotate mode.

Values:

enumerator kPXP_RotateOutputBuffer
Rotate the output buffer.

enumerator kPXP_RotateProcessSurface
Rotate the process surface. Cannot be used together with flip, scale, or decimation function.

enum _pxp_rotate_degree
PXP rotate degree.

Values:

enumerator kPXP_Rotate0
Clock wise rotate 0 deg.

enumerator kPXP_Rotate90
Clock wise rotate 90 deg.

enumerator kPXP_Rotate180
Clock wise rotate 180 deg.

enumerator kPXP_Rotate270
Clock wise rotate 270 deg.

enum _pxp_interlaced_output_mode
PXP interlaced output mode.

Values:

enumerator kPXP_OutputProgressive
All data written in progressive format to output buffer 0.

enumerator kPXP_OutputField0
Only write field 0 data to output buffer 0.

enumerator kPXP_OutputField1
Only write field 1 data to output buffer 0.

enumerator kPXP_OutputInterlaced
Field 0 write to buffer 0, field 1 write to buffer 1.

enum _pxp_output_pixel_format
PXP output buffer format.

Values:

enumerator kPXP_OutputPixelFormatARGB8888
32-bit pixels with alpha.

enumerator kPXP_OutputPixelFormatRGB888
32-bit pixels without alpha (unpacked 24-bit format)

enumerator kPXP_OutputPixelFormatRGB888P
24-bit pixels without alpha (packed 24-bit format)

enumerator kPXP_OutputPixelFormatARGB1555
16-bit pixels with alpha.

enumerator kPXP_OutputPixelFormatARGB4444
16-bit pixels with alpha.

enumerator kPXP_OutputPixelFormatRGB555
16-bit pixels without alpha.

enumerator kPXP_OutputPixelFormatRGB444
16-bit pixels without alpha.

enumerator kPXP_OutputPixelFormatRGB565
16-bit pixels without alpha.

enumerator kPXP_OutputPixelFormatYUV1P444
32-bit pixels (1-plane XYUV unpacked).

enumerator kPXP_OutputPixelFormatUYVY1P422
16-bit pixels (1-plane U0,Y0,V0,Y1 interleaved bytes)

enumerator kPXP_OutputPixelFormatVYUY1P422
16-bit pixels (1-plane V0,Y0,U0,Y1 interleaved bytes)

enumerator kPXP_OutputPixelFormatY8
8-bit monochrome pixels (1-plane Y luma output)

enumerator kPXP_OutputPixelFormatY4
4-bit monochrome pixels (1-plane Y luma, 4 bit truncation)

enumerator kPXP_OutputPixelFormatYUV2P422
16-bit pixels (2-plane UV interleaved bytes)

enumerator kPXP_OutputPixelFormatYUV2P420
16-bit pixels (2-plane UV)

enumerator kPXP_OutputPixelFormatYVU2P422
16-bit pixels (2-plane VU interleaved bytes)

enumerator kPXP_OutputPixelFormatYVU2P420
16-bit pixels (2-plane VU)

enum _pxp_ps_pixel_format
PXP process surface buffer pixel format.

Values:

enumerator kPXP_PsPixelFormatRGB888
32-bit pixels without alpha (unpacked 24-bit format)

enumerator kPXP_PsPixelFormatRGB555
16-bit pixels without alpha.

enumerator kPXP_PsPixelFormatRGB444
16-bit pixels without alpha.

enumerator kPXP_PsPixelFormatRGB565
16-bit pixels without alpha.

enumerator kPXP_PsPixelFormatYUV1P444
32-bit pixels (1-plane XYUV unpacked).

enumerator kPXP_PsPixelFormatUYVY1P422
16-bit pixels (1-plane U0,Y0,V0,Y1 interleaved bytes)

enumerator kPXP_PsPixelFormatVYUY1P422
16-bit pixels (1-plane V0,Y0,U0,Y1 interleaved bytes)

enumerator kPXP_PsPixelFormatY8
8-bit monochrome pixels (1-plane Y luma output)

enumerator kPXP_PsPixelFormatY4
4-bit monochrome pixels (1-plane Y luma, 4 bit truncation)

enumerator kPXP_PsPixelFormatYUV2P422
16-bit pixels (2-plane UV interleaved bytes)

enumerator kPXP_PsPixelFormatYUV2P420
16-bit pixels (2-plane UV)

enumerator kPXP_PsPixelFormatYVU2P422
16-bit pixels (2-plane VU interleaved bytes)

enumerator kPXP_PsPixelFormatYVU2P420
16-bit pixels (2-plane VU)

enumerator kPXP_PsPixelFormatYVU422
16-bit pixels (3-plane)

enumerator kPXP_PsPixelFormatYVU420
16-bit pixels (3-plane)

enum _pxp_ps_yuv_format
PXP process surface buffer YUV format.

Values:

enumerator kPXP_PsYUVFormatYUV
YUV format.

enumerator kPXP_PsYUVFormatYCbCr
YCbCr format.

enum _pxp_as_pixel_format
PXP alpha surface buffer pixel format.

Values:

enumerator kPXP_AsPixelFormatARGB8888
32-bit pixels with alpha.

enumerator kPXP_AsPixelFormatRGB888
32-bit pixels without alpha (unpacked 24-bit format)

enumerator kPXP_AsPixelFormatARGB1555
16-bit pixels with alpha.

enumerator kPXP_AsPixelFormatARGB4444
16-bit pixels with alpha.

enumerator kPXP_AsPixelFormatRGB555
16-bit pixels without alpha.

enumerator kPXP_AsPixelFormatRGB444
16-bit pixels without alpha.

enumerator kPXP_AsPixelFormatRGB565
16-bit pixels without alpha.

enum _pxp_alpha_mode
PXP alpha mode during blending.

Values:

enumerator kPXP_AlphaEmbedded
The alpha surface pixel alpha value will be used for blend.

enumerator kPXP_AlphaOverride
The user defined alpha value will be used for blend directly.

enumerator kPXP_AlphaMultiply
The alpha surface pixel alpha value scaled the user defined alpha value will be used for blend, for example, pixel alpha set set to 200, user defined alpha set to 100, then the reault alpha is 200 * 100 / 255.

enumerator kPXP_AlphaRop
Raster operation.

enum _pxp_rop_mode
PXP ROP mode during blending.

Explanation:

- AS: Alpha surface
- PS: Process surface
- nAS: Alpha surface NOT value
- nPS: Process surface NOT value

Values:

enumerator kPXP_RopMaskAs
AS AND PS.

enumerator kPXP_RopMaskNotAs
nAS AND PS.

enumerator kPXP_RopMaskAsNot
AS AND nPS.

enumerator kPXP_RopMergeAs
AS OR PS.

enumerator kPXP_RopMergeNotAs
nAS OR PS.

enumerator kPXP_RopMergeAsNot
AS OR nPS.

enumerator kPXP_RopNotCopyAs
nAS.

enumerator kPXP_RopNot
nPS.

enumerator kPXP_RopNotMaskAs
AS NAND PS.

enumerator kPXP_RopNotMergeAs
AS NOR PS.

enumerator kPXP_RopXorAs
AS XOR PS.

enumerator kPXP_RopNotXorAs
AS XNOR PS.

enum _pxp_block_size
PXP process block size.

Values:

enumerator kPXP_BlockSize8
Process 8x8 pixel blocks.

enumerator kPXP_BlockSize16
Process 16x16 pixel blocks.

enum _pxp_csc1_mode
PXP CSC1 mode.

Values:

enumerator kPXP_Csc1YUV2RGB
YUV to RGB.

enumerator kPXP_Csc1YCbCr2RGB
YCbCr to RGB.

enum _pxp_csc2_mode
PXP CSC2 mode.

Values:

enumerator kPXP_Csc2YUV2RGB
YUV to RGB.

enumerator kPXP_Csc2YCbCr2RGB
YCbCr to RGB.

enumerator kPXP_Csc2RGB2YUV
RGB to YUV.

enumerator kPXP_Csc2RGB2YCbCr
RGB to YCbCr.

enum _pxp_ram
PXP internal memory.

Values:

enumerator kPXP_RamDither0Lut
Dither 0 LUT memory.

enumerator kPXP_RamDither1Lut
Dither 1 LUT memory.

enumerator kPXP_RamDither2Lut
Dither 2 LUT memory.

enumerator kPXP_RamDither0Err0
Dither 0 ERR0 memory.

enumerator kPXP_RamDither0Err1
Dither 0 ERR1 memory.

enumerator kPXP_RamAluA
ALU A instr memory.

enumerator kPXP_RamAluB
ALU B instr memory.

enumerator kPXP_WfeAFetch
WFE-A fetch memory.

enumerator kPXP_WfeBFetch
WFE-B fetch memory.

enum _pxp_dither_mode
PXP dither mode.

Values:

enumerator kPXP_DitherPassThrough
Pass through, no dither.

enumerator kPXP_DitherFloydSteinberg
Floyd-Steinberg. For dither engine 0 only.

enumerator kPXP_DitherAtkinson
Atkinson. For dither engine 0 only.

enumerator kPXP_DitherOrdered
Ordered dither.

enumerator kPXP_DitherQuantOnly
No dithering, only quantization.

enumerator kPXP_DitherSierra
Sierra. For dither engine 0 only.

enum _pxp_dither_lut_mode

PXP dither LUT mode.

Values:

enumerator kPXP_DitherLutOff

The LUT memory is not used for LUT, could be used as ordered dither index matrix.

enumerator kPXP_DitherLutPreDither

Use LUT at the pre-dither stage, The pre-dither LUT could only be used in Floyd mode or Atkinson mode, which are not supported by current PXP module.

enumerator kPXP_DitherLutPostDither

Use LUT at the post-dither stage.

enum _pxp_dither_matrix_size

PXP dither matrix size.

Values:

enumerator kPXP_DitherMatrix4

The dither index matrix is 4x4.

enumerator kPXP_DitherMatrix8

The dither index matrix is 8x8.

enumerator kPXP_DitherMatrix16

The dither index matrix is 16x16.

Porter Duff factor mode. .

Values:

enumerator kPXP_PorterDuffFactorOne

Use 1.

enumerator kPXP_PorterDuffFactorZero

Use 0.

enumerator kPXP_PorterDuffFactorStraight

Use straight alpha.

enumerator kPXP_PorterDuffFactorInversed

Use inversed alpha.

Porter Duff global alpha mode. .

Values:

enumerator kPXP_PorterDuffGlobalAlpha

Use global alpha.

enumerator kPXP_PorterDuffLocalAlpha

Use local alpha in each pixel.

enumerator kPXP_PorterDuffScaledAlpha

Use global alpha * local alpha.

Porter Duff alpha mode. .

Values:

enumerator kPXP_PorterDuffAlphaStraight
Use straight alpha, s0_alpha' = s0_alpha.

enumerator kPXP_PorterDuffAlphaInversed
Use inversed alpha, s0_alpha' = 0xFF - s0_alpha.

Porter Duff color mode. .

Values:

enumerator kPXP_PorterDuffColorStraight

Deprecated:
Use kPXP_PorterDuffColorNoAlpha.

enumerator kPXP_PorterDuffColorInversed

Deprecated:
Use kPXP_PorterDuffColorWithAlpha.

enumerator kPXP_PorterDuffColorNoAlpha
s0_pixel' = s0_pixel.

enumerator kPXP_PorterDuffColorWithAlpha
s0_pixel' = s0_pixel * s0_alpha".

enum _pxp_porter_duff_blend_mode
PXP Porter Duff blend mode. Note: don't change the enum item value.

Values:

enumerator kPXP_PorterDuffSrc
Source Only

enumerator kPXP_PorterDuffAtop
Source Atop

enumerator kPXP_PorterDuffOver
Source Over

enumerator kPXP_PorterDuffIn
Source In.

enumerator kPXP_PorterDuffOut
Source Out.

enumerator kPXP_PorterDuffDst
Destination Only.

enumerator kPXP_PorterDuffDstAtop
Destination Atop.

enumerator kPXP_PorterDuffDstOver
Destination Over.

enumerator kPXP_PorterDuffDstIn
Destination In.

enumerator kPXP_PorterDuffDstOut
Destination Out.

enumerator kPXP_PorterDuffXor
XOR.

enumerator kPXP_PorterDuffClear
Clear.

enumerator kPXP_PorterDuffMax

enum _pxp_process_engine_name
PXP process engine enumeration.

Values:

enumerator kPXP_PsAsOutEngine

enumerator kPXP_DitherEngine

enumerator kPXP_WfeaEngine

enumerator kPXP_WfebEngine

enumerator kPXP_InputFetchStoreEngine

enumerator kPXP_Alpha1Engine

enumerator kPXP_Csc2Engine

enumerator kPXP_LutEngine

enumerator kPXP_Rotate0Engine

enumerator kPXP_Rotate1Engine

enum _pxp_fetch_engine_name
PXP fetch engine enumeration.

There are actually 4 fetch engine implemented, the others are WFE-A fetch engine and WFE-B fetch engine, whose registers are reserved from developer.

Values:

enumerator kPXP_FetchInput

enumerator kPXP_FetchDither

enum _pxp_fetch_interface_mode
PXP fetch engine interface mode with the upstream store engine.

Values:

enumerator kPXP_FetchModeNormal

enumerator kPXP_FetchModeHandshake

enumerator kPXP_FetchModeBypass

enum _pxp_scanline_burst
PXP fetch/store engine burst length for scanline mode.

Values:

enumerator kPXP_Scanline8bytes

enumerator kPXP_Scanline16bytes

enumerator kPXP_Scanline32bytes

enumerator kPXP_Scanline64bytes

enum _pxp_activeBits

PXP fetch/store engine input/output active bits configuration.

Since fetch engine is 64-bit input and 32-bit output per channel, need to configure both channels to use 64-bit input mode. And expand configuration will have no effect.

Values:

enumerator kPXP_Active8Bits

enumerator kPXP_Active16Bits

enumerator kPXP_Active32Bits

enumerator kPXP_Active64Bits

enum _pxp_fetch_output_word_order

PXP fetch engine output word order when using 2 channels for 64-bit mode.

Values:

enumerator kPXP_FetchOutputChannel1channel0

In 64bit mode, channel 1 output high byte.

enumerator kPXP_FetchOutputChannel0channel1

In 64bit mode, channel 0 output high byte.

enum _pxp_fetch_pixel_format

PXP fetch engine input pixel format.

Values:

enumerator kPXP_FetchFormatRGB565

enumerator kPXP_FetchFormatRGB555

enumerator kPXP_FetchFormatARGB1555

enumerator kPXP_FetchFormatRGB444

enumerator kPXP_FetchFormatARGB4444

enumerator kPXP_FetchFormatYUYVorYVYU

enumerator kPXP_FetchFormatUYVYorVYUY

enumerator kPXP_FetchFormatYUV422_2P

enum _pxp_store_engine_name

PXP store engine enumeration.

There are actually 4 store engine implemented, the others are WFE-A store engine and WFE-B store engine, whose registers are reserved from developer.

Values:

enumerator kPXP_StoreInput

enumerator kPXP_StoreDither

enum _pxp_store_interface_mode

PXP store engine interface mode with the downstream fetch engine.

Values:

enumerator kPXP_StoreModeBypass

Store engine output the input data, after the shift function directly to the downstream Fetch Engine.

enumerator kPXP_StoreModeNormal

Store engine stores the input data to memory.

enumerator kPXP_StoreModeHandshake

Downstream fetch engine fetch data per scanline from memory using buffer sharing with store engine.

enumerator kPXP_StoreModeDual

Store engine outputs data directly to downstream fetch engine(Bypass) but also storing it to memory at the same time.

enum _pxp_store_yuv_mode

PXP store engine YUV output mode.

Values:

enumerator kPXP_StoreYUVDisable

Do not output YUV pixel format.

enumerator kPXP_StoreYUVPlane1

Use channel to output YUV422_1p pixel format, need to use shift operation to make sure each pixel component in its proper position: 64-bits of pixel data format and each 32 bits as {Y0, U0, Y1, V0}.

enumerator kPXP_StoreYUVPlane2

Use channel to output YUV422_2p pixel format, need to use shift operation to make sure each pixel component in its proper position: channel 0 {Y0,Y1}, channel 1 {U0,V0}.

enum _pxp_cfa_input_format

PXP pre-dither CFA engine input pixel format.

Values:

enumerator kPXP_CfaRGB888

enumerator kPXP_CfaRGB444

enum _pxp_histogram_mask_condition

PXP histogram mask condition.

Values:

enumerator kPXP_HistogramMaskEqual

Value that equal to value0 will pass the mask operation.

enumerator kPXP_HistogramMaskNotequal

Value that not equal to value0 will pass the mask operation.

enumerator kPXP_HistogramMaskIn

Value that within the range of value0-value1 will pass the mask operation.

enumerator kPXP_HistogramMaskOut

Value that without the range of value0-value1 will pass the mask operation.

enum _pxp_histgram_flags

PXP Histogram operation result flags.

Values:

enumerator kPXP_Histogram2levelMatch

enumerator kPXP_Histogram4levelMatch

enumerator kPXP_Histogram8levelMatch

enumerator kPXP_Histogram16levelMatch

enumerator kPXP_Histogram32levelMatch

typedef enum *_pxp_flip_mode* pxp_flip_mode_t
PXP output flip mode.

typedef enum *_pxp_rotate_position* pxp_rotate_position_t
PXP rotate mode.

typedef enum *_pxp_rotate_degree* pxp_rotate_degree_t
PXP rotate degree.

typedef enum *_pxp_interlaced_output_mode* pxp_interlaced_output_mode_t
PXP interlaced output mode.

typedef enum *_pxp_output_pixel_format* pxp_output_pixel_format_t
PXP output buffer format.

typedef struct *_pxp_output_buffer_config* pxp_output_buffer_config_t
PXP output buffer configuration.

typedef enum *_pxp_ps_pixel_format* pxp_ps_pixel_format_t
PXP process surface buffer pixel format.

typedef enum *_pxp_ps_yuv_format* pxp_ps_yuv_format_t
PXP process surface buffer YUV format.

typedef struct *_pxp_ps_buffer_config* pxp_ps_buffer_config_t
PXP process surface buffer configuration.

typedef enum *_pxp_as_pixel_format* pxp_as_pixel_format_t
PXP alpha surface buffer pixel format.

typedef struct *_pxp_as_buffer_config* pxp_as_buffer_config_t
PXP alphs surface buffer configuration.

typedef enum *_pxp_alpha_mode* pxp_alpha_mode_t
PXP alpha mode during blending.

typedef enum *_pxp_rop_mode* pxp_rop_mode_t
PXP ROP mode during blending.

Explanation:

- AS: Alpha surface
- PS: Process surface
- nAS: Alpha surface NOT value
- nPS: Process surface NOT value

typedef struct *_pxp_as_blend_config* pxp_as_blend_config_t
PXP alpha surface blending configuration.

typedef struct *_pxp_as_blend_secondary_config* pxp_as_blend_secondary_config_t
PXP secondary alpha surface blending engine configuration.

typedef enum *_pxp_block_size* pxp_block_size_t
PXP process block size.

typedef enum *_pxp_csc1_mode* pxp_csc1_mode_t

PXP CSC1 mode.

typedef enum *_pxp_csc2_mode* pxp_csc2_mode_t

PXP CSC2 mode.

typedef struct *_pxp_csc2_config* pxp_csc2_config_t

PXP CSC2 configuration.

Converting from YUV/YCbCr color spaces to the RGB color space uses the following equation structure:

R = A1(Y+D1) + A2(U+D2) + A3(V+D3) G = B1(Y+D1) + B2(U+D2) + B3(V+D3) B = C1(Y+D1) + C2(U+D2) + C3(V+D3)

Converting from the RGB color space to YUV/YCbCr color spaces uses the following equation structure:

Y = A1*R + A2*G + A3*B + D1 U = B1*R + B2*G + B3*B + D2 V = C1*R + C2*G + C3*B + D3

typedef enum *_pxp_ram* pxp_ram_t

PXP internal memory.

typedef struct *_pxp_dither_final_lut_data* pxp_dither_final_lut_data_t

PXP dither final LUT data.

typedef struct *_pxp_dither_config* pxp_dither_config_t

PXP dither configuration.

typedef enum *_pxp_porter_duff_blend_mode* pxp_porter_duff_blend_mode_t

PXP Porter Duff blend mode. Note: don't change the enum item value.

typedef struct *_pxp_pic_copy_config* pxp_pic_copy_config_t

PXP Porter Duff blend mode. Note: don't change the enum item value.

typedef enum *_pxp_process_engine_name* pxp_process_engine_name_t

PXP process engine enumeration.

typedef enum *_pxp_fetch_engine_name* pxp_fetch_engine_name_t

PXP fetch engine enumeration.

There are actually 4 fetch engine implemented, the others are WFE-A fetch engine and WFE-B fetch engine, whose registers are reserved from developer.

typedef enum *_pxp_fetch_interface_mode* pxp_fetch_interface_mode_t

PXP fetch engine interface mode with the upstream store engine.

typedef enum *_pxp_scanline_burst* pxp_scanline_burst_t

PXP fetch/store engine burst length for scanline mode.

typedef struct *_pxp_block_format_config* pxp_block_config_t

PXP fetch engine block configuration.

typedef enum *_pxp_activeBits* pxp_active_bits_t

PXP fetch/store engine input/output active bits configuration.

Since fetch engine is 64-bit input and 32-bit output per channel, need to configure both channels to use 64-bit input mode. And expand configuration will have no effect.

typedef enum *_pxp_fetch_output_word_order* pxp_fetch_output_word_order_t

PXP fetch engine output word order when using 2 channels for 64-bit mode.

typedef struct *_pxp_fetch_shift_component* pxp_fetch_shift_component_t
PXP fetch engine shift component configuration.

Fetch engine can divded each word into 4 components and shift them.

typedef struct *_pxp_fetch_shift_config* pxp_fetch_shift_config_t
PXP fetch engine shift configuration.

Fetch engine can divded each word into 4 components and shift them. For example, to change YUV444 to YVU444, U and V positions need to be shifted: OFFSET0=8, OFFSET1=0, OFFSET2=16, OFFSET3=24, WIDTH0/1/2/3=8

typedef enum *_pxp_fetch_pixel_format* pxp_fetch_pixel_format_t
PXP fetch engine input pixel format.

typedef struct *_pxp_fetch_engine_config* pxp_fetch_engine_config_t
PXP fetch engine configuration for one of the channel.

typedef enum *_pxp_store_engine_name* pxp_store_engine_name_t
PXP store engine enumeration.

There are actually 4 store engine implemented, the others are WFE-A store engine and WFE-B store engine, whose registers are reserved from developer.

typedef enum *_pxp_store_interface_mode* pxp_store_interface_mode_t
PXP store engine interface mode with the downstream fetch engine.

typedef enum *_pxp_store_yuv_mode* pxp_store_yuv_mode_t
PXP store engine YUV output mode.

typedef struct *_pxp_store_shift_config* pxp_store_shift_config_t
Shift configuration for PXP store engine.

typedef struct *_pxp_store_engine_config* pxp_store_engine_config_t
PXP store engine configuration for one of the channel.

typedef enum *_pxp_cfa_input_format* pxp_cfa_input_format_t
PXP pre-dither CFA engine input pixel format.

typedef struct *_pxp_cfa_config* pxp_cfa_config_t
PXP pre-dither CFA engine configuration.

typedef enum *_pxp_histogram_mask_condition* pxp_histogram_mask_condition_t
PXP histogram mask condition.

typedef struct *_pxp_histogram_config* pxp_histogram_config_t
PXP Histogram configuration.

typedef struct *_pxp_histogram_mask_result* pxp_histogram_mask_result_t
PXP Histogram mask result.

typedef struct *_pxp_wfea_engine_config* pxp_wfea_engine_config_t
PXP WFE-A engine configuration.

PXP_LUT_TABLE_BYTE

PXP_INTERNAL_RAM_LUT_BYTE

PXP_SHARE_ROTATE

PXP_USE_PATH

PXP_COMBINE_BYTE_TO_WORD(dataAddr)

struct pxp_output_buffer_config
#include <fsl_pxp.h> PXP output buffer configuration.

Public Members

pxp_output_pixel_format_t pixelFormat
Output buffer pixel format.

pxp_interlaced_output_mode_t interlacedMode
Interlaced output mode.

uint32_t buffer0Addr
Output buffer 0 address.

uint32_t buffer1Addr
Output buffer 1 address, used for UV data in YUV 2-plane mode, or field 1 in output interlaced mode.

uint16_t pitchBytes
Number of bytes between two vertically adjacent pixels.

uint16_t width
Pixels per line.

uint16_t height
How many lines in output buffer.

struct __pxp__ps__buffer__config
#include <fsl_pxp.h> PXP process surface buffer configuration.

Public Members

pxp_ps_pixel_format_t pixelFormat
PS buffer pixel format.

bool swapByte
For each 16 bit word, set true to swap the two bytes.

uint32_t bufferAddr
Input buffer address for the first panel.

uint32_t bufferAddrU
Input buffer address for the second panel.

uint32_t bufferAddrV
Input buffer address for the third panel.

uint16_t pitchBytes
Number of bytes between two vertically adjacent pixels.

struct __pxp__as__buffer__config
#include <fsl_pxp.h> PXP alphs surface buffer configuration.

Public Members

pxp_as_pixel_format_t pixelFormat
AS buffer pixel format.

uint32_t bufferAddr
Input buffer address.

uint16_t pitchBytes
Number of bytes between two vertically adjacent pixels.

struct __pxp_as_blend_config

#include <fsl_pxp.h> PXP alpha surface blending configuration.

Public Members

uint8_t alpha

User defined alpha value, only used when alphaMode is kPXP_AlphaOverride or kPXP_AlphaRop.

bool invertAlpha

Set true to invert the alpha.

pxp_alpha_mode_t alphaMode

Alpha mode.

pxp_rop_mode_t ropMode

ROP mode, only valid when alphaMode is kPXP_AlphaRop.

struct __pxp_as_blend_secondary_config

#include <fsl_pxp.h> PXP secondary alpha surface blending engine configuration.

Public Members

bool invertAlpha

Set true to invert the alpha.

bool ropEnable

Enable rop mode.

pxp_rop_mode_t ropMode

ROP mode, only valid when ropEnable is true.

struct __pxp_csc2_config

#include <fsl_pxp.h> PXP CSC2 configuration.

Converting from YUV/YCbCr color spaces to the RGB color space uses the following equation structure:

R = A1(Y+D1) + A2(U+D2) + A3(V+D3) G = B1(Y+D1) + B2(U+D2) + B3(V+D3) B = C1(Y+D1) + C2(U+D2) + C3(V+D3)

Converting from the RGB color space to YUV/YCbCr color spaces uses the following equation structure:

Y = A1*R + A2*G + A3*B + D1 U = B1*R + B2*G + B3*B + D2 V = C1*R + C2*G + C3*B + D3

Public Members

pxp_csc2_mode_t mode

Convertion mode.

float A1

A1.

float A2

A2.

float A3

A3.

float B1
B1.

float B2
B2.

float B3
B3.

float C1
C1.

float C2
C2.

float C3
C3.

int16_t D1
D1.

int16_t D2
D2.

int16_t D3
D3.

struct __pxp_dither_final_lut_data
#include <fsl_pxp.h> PXP dither final LUT data.

Public Members

uint32_t data_3_0
Data 3 to data 0. Data 0 is the least significant byte.

uint32_t data_7_4
Data 7 to data 4. Data 4 is the least significant byte.

uint32_t data_11_8
Data 11 to data 8. Data 8 is the least significant byte.

uint32_t data_15_12
Data 15 to data 12. Data 12 is the least significant byte.

struct __pxp_dither_config
#include <fsl_pxp.h> PXP dither configuration.

Public Members

uint32_t enableDither0
Enable dither engine 0 or not, set 1 to enable, 0 to disable.

uint32_t enableDither1
Enable dither engine 1 or not, set 1 to enable, 0 to disable.

uint32_t enableDither2
Enable dither engine 2 or not, set 1 to enable, 0 to disable.

uint32_t ditherMode0
Dither mode for dither engine 0. See _pxp_dither_mode.

uint32_t ditherMode1

Dither mode for dither engine 1. See _pxp_dither_mode.

uint32_t ditherMode2

Dither mode for dither engine 2. See _pxp_dither_mode.

uint32_t quantBitNum

Number of bits quantize down to, the valid value is 1~7.

uint32_t lutMode

How to use the memory LUT, see _pxp_dither_lut_mode. This must be set to kPXP_DitherLutOff if any dither engine uses kPXP_DitherOrdered mode.

uint32_t idxMatrixSize0

Size of index matrix used for dither for dither engine 0, see _pxp_dither_matrix_size.

uint32_t idxMatrixSize1

Size of index matrix used for dither for dither engine 1, see _pxp_dither_matrix_size.

uint32_t idxMatrixSize2

Size of index matrix used for dither for dither engine 2, see _pxp_dither_matrix_size.

uint32_t enableFinalLut

Enable the final LUT, set 1 to enable, 0 to disable.

struct pxp_porter_duff_config_t

#include <fsl_pxp.h> PXP Porter Duff configuration.

Public Members

uint32_t enable

Enable or disable Porter Duff.

uint32_t srcFactorMode

Source layer (or AS, s1) factor mode, see pxp_porter_duff_factor_mode.

uint32_t dstGlobalAlphaMode

Destination layer (or PS, s0) global alpha mode, see pxp_porter_duff_global_alpha_mode.

uint32_t dstAlphaMode

Destination layer (or PS, s0) alpha mode, see pxp_porter_duff_alpha_mode.

uint32_t dstColorMode

Destination layer (or PS, s0) color mode, see pxp_porter_duff_color_mode.

uint32_t dstFactorMode

Destination layer (or PS, s0) factor mode, see pxp_porter_duff_factor_mode.

uint32_t srcGlobalAlphaMode

Source layer (or AS, s1) global alpha mode, see pxp_porter_duff_global_alpha_mode.

uint32_t srcAlphaMode

Source layer (or AS, s1) alpha mode, see pxp_porter_duff_alpha_mode.

uint32_t srcColorMode

Source layer (or AS, s1) color mode, see pxp_porter_duff_color_mode.

uint32_t dstGlobalAlpha

Destination layer (or PS, s0) global alpha value, 0~255.

uint32_t srcGlobalAlpha

Source layer (or AS, s1) global alpha value, 0~255.

struct __pxp_pic_copy_config

#include <fsl_pxp.h> PXP Porter Duff blend mode. Note: don't change the enum item value.

Public Members

uint32_t srcPicBaseAddr

Source picture base address.

uint16_t srcPitchBytes

Pitch of the source buffer.

uint16_t srcOffsetX

Copy position in source picture.

uint16_t srcOffsetY

Copy position in source picture.

uint32_t destPicBaseAddr

Destination picture base address.

uint16_t destPitchBytes

Pitch of the destination buffer.

uint16_t destOffsetX

Copy position in destination picture.

uint16_t destOffsetY

Copy position in destination picture.

uint16_t width

Pixel number each line to copy.

uint16_t height

Lines to copy.

pxp_as_pixel_format_t pixelFormat

Buffer pixel format.

struct __pxp_block_format_config

#include <fsl_pxp.h> PXP fetch engine block configuration.

Public Members

bool enableblock

Enable to use block mode instead of scanline mode. Note: 1.Make sure to enable if rotate or flip mode is enabled. 2.Block mode cannot work on 64bpp data stream where activeBits = 64. 3. If LUT processing is in the path between the fetch and store engind, block mode must be enabled.

bool blockSize16

Enable to use 16*16 block, otherwise it will be 8*8 block.

pxp_scanline_burst_t burstLength

When using scanline mode, configure this for burst length.

struct __pxp_fetch_shift_component

#include <fsl_pxp.h> PXP fetch engine shift component configuration.

Fetch engine can divded each word into 4 components and shift them.

struct __pxp_fetch_shift_config

#include <fsl_pxp.h> PXP fetch engine shift configuration.

Fetch engine can divded each word into 4 components and shift them. For example, to change YUV444 to YVU444, U and V positions need to be shifted: OFFSET0=8, OFFSET1=0, OFFSET2=16, OFFSET3=24, WIDTH0/1/2/3=8

struct __pxp_fetch_engine_config

#include <fsl_pxp.h> PXP fetch engine configuration for one of the channel.

Public Members

bool channelEnable

Enable channel.

uint32_t inputBaseAddr0

The input base address. Used for Y plane input when pixel format is YUV422_2p.

uint32_t inputBaseAddr1

Must configure this for UV plane when input pixel format is YUV422_2p.

uint16_t totalHeight

Total height for the actual fetch size.

uint16_t totalWidth

Total width for the actual fetch size.

uint16_t pitchBytes

Channel input pitch

uint16_t ulcX

X coordinate of upper left coordinate in pixels of the active area of the total input memory

uint16_t ulcY

Y coordinate of upper left coordinate in pixels of the active area of the total input memory

uint16_t lrcX

X coordinate of Lower right coordinate in pixels of the active area of the total input memory

uint16_t lrcY

Y coordinate of Lower right coordinate in pixels of the active area of the total input memory

uint32_t backGroundColor

Pixel value of the background color for the space outside the active area.

pxp_fetch_interface_mode_t interface

Interface mode, normal/bypass/handshake

pxp_active_bits_t activeBits

Input active bits.

pxp_fetch_pixel_format_t pixelFormat

Input pixel fetch format

bool expandEnable

If enabled, input pixel will be expanded to ARGB8888, RGB888 or YUV444 of 32-bit format at the output.

pxp_flip_mode_t flipMode

Flip the fetched input.

pxp_rotate_degree_t rotateDegree

Rotate the fetched input.

pxp_block_config_t fetchFormat

Block mode configuration. Make sure to enable block if rotate or flip mode is enabled.

pxp_fetch_shift_config_t shiftConfig

Shift operation configuration.

pxp_fetch_output_word_order_t wordOrder

Output word order when using 2 channels for 64-bit mode.

struct __pxp__store__shift__config

#include <fsl_pxp.h> Shift configuration for PXP store engine.

Public Members

bool shiftBypass

Bypass the data shift

uint64_t *pDataShiftMask

Pointer to mask0~mask7 to mask the 64-bit of output data, data is masked first then shifted according to width.

uint8_t *pDataShiftWidth

Pointer to width0~width7. Bit 7 is for shifted direction, 0 to right. Bit0~5 is for shift width.

uint8_t *pFlagShiftMask

Pointer to mask0~mask7 to mask the 8-bit of output flag, flag is masked first then shifted according to width.

uint8_t *pFlagShiftWidth

Pointer to width0~width7. Bit 6 is for shifted direction, 0 to right. Bit0~5 is for shift width.

struct __pxp__store__engine__config

#include <fsl_pxp.h> PXP store engine configuration for one of the channel.

Public Members

bool channelEnable

Enable channel.

uint32_t outputBaseAddr0

The channel 0 output address if using 2 channels. If using 1 channel(must be channel 0) and YUV422_2p output format, is for Y plane address.

uint32_t outputBaseAddr1

The channel 1 output address if using 2 channels. If using 1 channel(must be channel 0) and YUV422_2p output format, is for UV plane address.

uint16_t totalHeight

Total height for the actual store size.

uint16_t totalWidth

Total width for the actual store size.

uint16_t pitchBytes

Channel input pitch

pxp_store_interface_mode_t interface

Interface mode, normal/bypass/handshake/dual. Make sure 2 channels use the same mode if both enabled.

pxp_active_bits_t activeBits

Output active bits.

pxp_store_yuv_mode_t yuvMode

Whether to output YUV pixel format.

bool useFixedData

Whether to use fixed value for the output data. Can be used to write fixed value to specific memory location for memory initialization.

uint32_t fixedData

The value of the fixed data.

bool packInSelect

When enabled, channel 0 will select low 32 bit shift out data to pack while channel i select high 32 bit, otherwise all 64-bit of data will be selected.

pxp_block_config_t storeFormat

The format to store data, block or otherwise.

pxp_store_shift_config_t shiftConfig

Shift operation configuration.

struct __pxp__cfa__config

#include <fsl_pxp.h> PXP pre-dither CFA engine configuration.

Public Members

bool bypass

Bypass the CFA process

pxp_cfa_input_format_t pixelInFormat

The pixel input format for CFA.

uint8_t arrayWidth

CFA array vertical size in pixels, min 3 max 15.

uint8_t arrayHeight

CFA array horizontal size in pixels, min 3 max 15.

uint16_t totalHeight

Total height for the buffer size, make sure it is aligned with the dither fetch engine and dither engine.

uint16_t totalWidth

Total width for the buffer size, make sure it is aligned with the dither fetch engine and dither engine.

uint32_t *cfaValue

Pointer to the value for the CFA array. 2-bit per component: 00-R,01-G,10-B,11-W. For a 4x4 array, 32 bits are need.

struct __pxp_histogram_config

#include <fsl_pxp.h> PXP Histogram configuration.

Public Members

bool enable

Enable histogram process.

uint8_t *pParamValue

Pointer to the 62(2+4+8+16+32) byte of param value for 2-level, 4-level.....32-level parameters. Only low 5-bit of each byte is valid.

uint8_t lutValueOffset

The starting bit position of the LUT value.

uint8_t lutValueWidth

The bit width of the LUT value, should be no more than 6 bits since only 63 LUTs are supported.

bool enableMask

Enable mask operation.

uint8_t maskValue0

Value 0 for the condition judgement.

uint8_t maskValue1

Value 1 for the condition judgement.

uint8_t maskOffset

The starting bit position of the field to be checked against mask condition.

uint8_t maskWidth

The width of the field to be checked against mask condition.

pxp_histogram_mask_condition_t condition

The mask condition.

uint16_t totalHeight

Total height for the buffer size, make sure it is aligned with the output of legacy flow or the WFE-A/B engine.

uint16_t totalWidth

Total width for the buffer size, make sure it is aligned with the output of legacy flow or the WFE-A/B engine.

struct __pxp_histogram_mask_result

#include <fsl_pxp.h> PXP Histogram mask result.

Public Members

uint32_t pixelCount

The total count of the pixels that pass the mask(collided pixels).

uint32_t minX

The x offset of the ULC of the minimal histogram that covers all passed pixels.

uint32_t minY
: The y offset of the ULC of the minimal histogram that covers all passed pixels.

uint32_t maxX
: The x offset of the LRC of the minimal histogram that covers all passed pixels.

uint32_t maxY
: The y offset of the LRC of the minimal histogram that covers all passed pixels.

uint64_t lutlist
: The 64-bit LUT list of collided pixels, if pixel of LUT17 is collided, bit17 in the list is set.

struct __pxp__wfea__engine__config
: *#include <fsl_pxp.h>* PXP WFE-A engine configuration.

Public Members

uint32_t y4Addr
: Address for Y4 buffer.

uint32_t y4cAddr
: Address for Y4C buffer, {Y4[3:0],3'b000,collision}, 8bpp.

uint32_t wbAddr
: Address for EPDC working buffer.

uint16_t updateWidth
: Width of the update area.

uint16_t updateHeight
: Height of the update area.

uint16_t updatePitch
: Pitch of the update area.

uint16_t ulcX
: X coordinate of upper left coordinate of the total input memory

uint16_t ulcY
: Y coordinate of upper left coordinate of the total input memory

uint16_t resX
: Horizontal resolution in pixels.

uint8_t lutNum
: The EPDC LUT number for the update.

bool fullUpdateEnable
: Enable full update.

bool alphaEnable
: Enable alpha field, upd is {Y4[3:0],3'b000,alpha} format, otherwise its {Y4[3:0],4'b0000}.

bool detectionOnly
: Detection only, do not write working buffer.

1.59 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

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

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 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

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

1.60 Romapi

Flash Pad Definitions.

Values:

enumerator kSerialFlash_1Pad

enumerator kSerialFlash_2Pads

enumerator kSerialFlash_4Pads

enumerator kSerialFlash_8Pads

FLEXSPI clock configuration type.

Values:

enumerator kFLEXSPIClk_SDR
Clock configure for SDR mode

enumerator kFLEXSPIClk_DDR
Clock configurat for DDR mode

enum _flexspi_read_sample_clk
FLEXSPI Read Sample Clock Source definition.

Values:

enumerator kFLEXSPIReadSampleClk_LoopbackInternally

enumerator kFLEXSPIReadSampleClk_LoopbackFromDqsPad

enumerator kFLEXSPIReadSampleClk_LoopbackFromSckPad

enumerator kFLEXSPIReadSampleClk_ExternalInputFromDqsPad

Flash Type Definition.

Values:

enumerator kFLEXSPIDeviceType_SerialNOR
Flash device is Serial NOR

Flash Configuration Command Type.

Values:

enumerator kDeviceConfigCmdType_Generic
Generic command, for example: configure dummy cycles, drive strength, etc

enumerator kDeviceConfigCmdType_QuadEnable
Quad Enable command

enumerator kDeviceConfigCmdType_Spi2Xpi
Switch from SPI to DPI/QPI/OPI mode

enumerator kDeviceConfigCmdType_Xpi2Spi
Switch from DPI/QPI/OPI to SPI mode

enumerator kDeviceConfigCmdType_Spi2NoCmd
Switch to 0-4-4/0-8-8 mode

enumerator kDeviceConfigCmdType_Reset
Reset device command

enum _flexspi_serial_clk_freq
Defintions for FLEXSPI Serial Clock Frequency.

Values:

enumerator kFLEXSPISerialClk_NoChange

enumerator kFLEXSPISerialClk_30MHz

enumerator kFLEXSPISerialClk_50MHz

enumerator kFLEXSPISerialClk_60MHz

enumerator kFLEXSPISerialClk_75MHz

enumerator kFLEXSPISerialClk_80MHz

enumerator kFLEXSPISerialClk_100MHz

enumerator kFLEXSPISerialClk_133MHz

enumerator kFLEXSPISerialClk_166MHz

enumerator kFLEXSPISerialClk_200MHz

Misc feature bit definitions.

Values:

enumerator kFLEXSPIMiscOffset_DiffClkEnable
Bit for Differential clock enable

enumerator kFLEXSPIMiscOffset_Ck2Enable
Bit for CK2 enable

enumerator kFLEXSPIMiscOffset_ParallelEnable
Bit for Parallel mode enable

enumerator kFLEXSPIMiscOffset_WordAddressableEnable
Bit for Word Addressable enable

enumerator kFLEXSPIMiscOffset_SafeConfigFreqEnable
Bit for Safe Configuration Frequency enable

enumerator kFLEXSPIMiscOffset_PadSettingOverrideEnable
Bit for Pad setting override enable

enumerator kFLEXSPIMiscOffset_DdrModeEnable
Bit for DDR clock confiuration indication.

enumerator kFLEXSPIMiscOffset_UseValidTimeForAllFreq
Bit for DLLCR settings under all modes

Manufacturer ID.

Values:

enumerator kSerialFlash_ISSI_ManufacturerID
Manufacturer ID of the ISSI serial flash

enumerator kSerialFlash_Adesto_ManufacturerID
Manufacturer ID of the Adesto Technologies serial flash

enumerator kSerialFlash_Winbond_ManufacturerID
Manufacturer ID of the Winbond serial flash

enumerator kSerialFlash_Cypress_ManufacturerID
Manufacturer ID for Cypress

enum _flexspi_nor_status
ROM FLEXSPI NOR flash status.

Values:

enumerator kStatus_ROM_FLEXSPI_SequenceExecutionTimeout
Status for Sequence Execution timeout

enumerator kStatus_ROM_FLEXSPI_InvalidSequence
Status for Invalid Sequence

enumerator kStatus_ROM_FLEXSPI_DeviceTimeout
Status for Device timeout

enumerator kStatus_FLEXSPINOR_DTRRead_DummyProbeFailed
Status for DDR Read dummy probe failure

enumerator kStatus_ROM_FLEXSPINOR_SFDP_NotFound
Status for SFDP read failure

enumerator kStatus_ROM_FLEXSPINOR_Flash_NotFound
Status for Flash detection failure

enum _flexspi_operation
FLEXSPI Operation Context.

Values:

enumerator kFLEXSPIOperation_Command
FLEXSPI operation: Only command, both TX and RX buffer are ignored.

enumerator kFLEXSPIOperation_Config
FLEXSPI operation: Configure device mode, the TX FIFO size is fixed in LUT.

enumerator kFLEXSPIOperation_Write
FLEXSPI operation: Write, only TX buffer is effective

enumerator kFLEXSPIOperation_Read
FLEXSPI operation: Read, only Rx Buffer is effective.

typedef struct *_flexspi_lut_seq* flexspi_lut_seq_t
FLEXSPI LUT Sequence structure.

typedef struct *_flexspi_mem_config* flexspi_mem_config_t
FLEXSPI Memory Configuration Block.

typedef struct *_flexspi_nor_config* flexspi_nor_config_t
Serial NOR configuration block.

typedef enum *_flexspi_operation* flexspi_operation_t
FLEXSPI Operation Context.

typedef struct *_flexspi_xfer* flexspi_xfer_t
FLEXSPI Transfer Context.

typedef struct *_serial_nor_config_option* serial_nor_config_option_t
Serial NOR Configuration Option.

void ROM_RunBootloader(void *arg)
Enter Bootloader.

Parameters

- arg – A pointer to the storage for the bootloader param. refer to System Boot Chapter in device reference manual for details.

status_t ROM_FLEXSPI_NorFlash_Init(uint32_t instance, *flexspi_nor_config_t* *config)
Initialize Serial NOR devices via FLEXSPI.

This function checks and initializes the FLEXSPI module for the other FLEXSPI APIs.

Parameters

- instance – storage the instance of FLEXSPI.
- config – A pointer to the storage for the driver runtime state.

Return values

- kStatus_Success – Api was executed succesfuly.
- kStatus_InvalidArgument – A invalid argument is provided.
- kStatus_ROM_FLEXSPI_InvalidSequence – A invalid Sequence is provided.
- kStatus_ROM_FLEXSPI_SequenceExecutionTimeout – Sequence Execution timeout.
- kStatus_ROM_FLEXSPI_DeviceTimeout – the device timeout

status_t ROM_FLEXSPI_NorFlash_ProgramPage(uint32_t instance, *flexspi_nor_config_t* *config, uint32_t dstAddr, const uint32_t *src)
Program data to Serial NOR via FLEXSPI.

This function programs the NOR flash memory with the dest address for a given flash area as determined by the dst address and the length.

Parameters

- instance – storage the instance of FLEXSPI.
- config – A pointer to the storage for the driver runtime state.
- dstAddr – A pointer to the desired flash memory to be programmed. NOTE: It is recommended that use page aligned access; If the dstAddr is not aligned to page,the driver automatically aligns address down with the page address.
- src – A pointer to the source buffer of data that is to be programmed into the NOR flash.

Return values

- kStatus_Success – Api was executed succesfuly.
- kStatus_InvalidArgument – A invalid argument is provided.
- kStatus_ROM_FLEXSPI_InvalidSequence – A invalid Sequence is provided.
- kStatus_ROM_FLEXSPI_SequenceExecutionTimeout – Sequence Execution timeout.
- kStatus_ROM_FLEXSPI_DeviceTimeout – the device timeout

status_t ROM_FLEXSPI_NorFlash_Read(uint32_t instance, *flexspi_nor_config_t* *config, uint32_t *dst, uint32_t start, uint32_t lengthInBytes)

Read data from Serial NOR via FLEXSPI.

This function read the NOR flash memory with the start address for a given flash area as determined by the dst address and the length.

Parameters

- instance – storage the instance of FLEXSPI.
- config – A pointer to the storage for the driver runtime state.
- dst – A pointer to the dest buffer of data that is to be read from the NOR flash. NOTE: It is recommended that use page aligned access; If the dstAddr is not aligned to page,the driver automatically aligns address down with the page address.
- start – The start address of the desired NOR flash memory to be read.
- lengthInBytes – The length, given in bytes to be read.

Return values

- kStatus_Success – Api was executed succesfuly.
- kStatus_InvalidArgument – A invalid argument is provided.
- kStatus_ROM_FLEXSPI_InvalidSequence – A invalid Sequence is provided.
- kStatus_ROM_FLEXSPI_SequenceExecutionTimeout – Sequence Execution timeout.
- kStatus_ROM_FLEXSPI_DeviceTimeout – the device timeout

status_t ROM_FLEXSPI_NorFlash_Erase(uint32_t instance, *flexspi_nor_config_t* *config, uint32_t start, uint32_t length)

Erase Flash Region specified by address and length.

This function erases the appropriate number of flash sectors based on the desired start address and length.

Parameters

- instance – storage the index of FLEXSPI.
- config – A pointer to the storage for the driver runtime state.
- start – The start address of the desired NOR flash memory to be erased. NOTE: It is recommended that use sector-aligned access nor device; If dstAddr is not aligned with the sector,the driver automatically aligns address down with the sector address.
- length – The length, given in bytes to be erased. NOTE: It is recommended that use sector-aligned access nor device; If length is not aligned with the sector,the driver automatically aligns up with the sector.

Return values

- kStatus_Success – Api was executed succesfuly.
- kStatus_InvalidArgument – A invalid argument is provided.
- kStatus_ROM_FLEXSPI_InvalidSequence – A invalid Sequence is provided.
- kStatus_ROM_FLEXSPI_SequenceExecutionTimeout – Sequence Execution timeout.
- kStatus_ROM_FLEXSPI_DeviceTimeout – the device timeout

status_t ROM_FLEXSPI_NorFlash_EraseAll(uint32_t instance, *flexspi_nor_config_t* *config)

Erase all the Serial NOR devices connected on FLEXSPI.

Parameters

- instance – storage the instance of FLEXSPI.
- config – A pointer to the storage for the driver runtime state.

Return values

- kStatus_Success – Api was executed succesfuly.
- kStatus_InvalidArgument – A invalid argument is provided.
- kStatus_ROM_FLEXSPI_InvalidSequence – A invalid Sequence is provided.
- kStatus_ROM_FLEXSPI_SequenceExecutionTimeout – Sequence Execution timeout.
- kStatus_ROM_FLEXSPI_DeviceTimeout – the device timeout

status_t ROM_FLEXSPI_NorFlash_CommandXfer(uint32_t instance, *flexspi_xfer_t* *xfer)

FLEXSPI command.

This function is used to perform the command write sequence to the NOR device.

Parameters

- instance – storage the index of FLEXSPI.
- xfer – A pointer to the storage FLEXSPI Transfer Context.

Return values

- kStatus_Success – Api was executed succesfuly.
- kStatus_InvalidArgument – A invalid argument is provided.
- kStatus_ROM_FLEXSPI_InvalidSequence – A invalid Sequence is provided.
- kStatus_ROM_FLEXSPI_SequenceExecutionTimeout – Sequence Execution timeout.

status_t ROM_FLEXSPI_NorFlash_UpdateLut(uint32_t instance, uint32_t seqIndex, const uint32_t *lutBase, uint32_t seqNumber)

Configure FLEXSPI Lookup table.

Parameters

- instance – storage the index of FLEXSPI.
- seqIndex – storage the sequence Id.
- lutBase – A pointer to the look-up-table for command sequences.
- seqNumber – storage sequence number.

Return values

- kStatus_Success – Api was executed succesfuly.
- kStatus_InvalidArgument – A invalid argument is provided.
- kStatus_ROM_FLEXSPI_InvalidSequence – A invalid Sequence is provided.
- kStatus_ROM_FLEXSPI_SequenceExecutionTimeout – Sequence Execution timeout.

void ROM_FLEXSPI_NorFlash_ClearCache(uint32_t instance)

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

- instance – storage the index of FLEXSPI.

status_t ROM_FLEXSPI_NorFlash_GetConfig(uint32_t instance, *flexspi_nor_config_t* *config, *serial_nor_config_option_t* *option)

Get FLEXSPI NOR Configuration Block based on specified option.

Parameters

- instance – storage the instance of FLEXSPI.
- config – A pointer to the storage for the driver runtime state.
- option – A pointer to the storage Serial NOR Configuration Option Context.

Return values

- kStatus_Success – Api was executed succesfuly.
- kStatus_InvalidArgument – A invalid argument is provided.

FSL_ROM_HAS_FLEXSPINOR_API

FSL_ROM_HAS_RUNBOOTLOADER_API

FSL_ROM_FLEXSPINOR_API_HAS_FEATURE_GET_CONFIG

FSL_ROM_FLEXSPINOR_API_HAS_FEATURE_FLASH_INIT

FSL_ROM_FLEXSPINOR_API_HAS_FEATURE_ERASE

FSL_ROM_FLEXSPINOR_API_HAS_FEATURE_ERASE_SECTOR

FSL_ROM_FLEXSPINOR_API_HAS_FEATURE_ERASE_BLOCK

FSL_ROM_FLEXSPINOR_API_HAS_FEATURE_ERASE_ALL

FSL_ROM_FLEXSPINOR_API_HAS_FEATURE_READ

FSL_ROM_FLEXSPINOR_API_HAS_FEATURE_UPDATE_LUT

FSL_ROM_FLEXSPINOR_API_HAS_FEATURE_CMD_XFER

kROM_StatusGroup_FLEXSPI
: ROM FLEXSPI status group number.

kROM_StatusGroup_FLEXSPINOR
: ROM FLEXSPI NOR status group number.

FSL_ROM_FLEXSPI_LUT_SEQ(cmd0, pad0, op0, cmd1, pad1, op1)

FSL_ROM_FLEXSPI_BITMASK(bit_offset)
: Generate bit mask.

FLEXSPI_CFG_BLK_TAG
: FLEXSPI memory config block related defintions.

  ascii “FCFB” Big Endian

FLEXSPI_CFG_BLK_VERSION
: V1.4.0

CMD_SDR

CMD_DDR

RADDR_SDR

RADDR_DDR

CADDR_SDR

CADDR_DDR

MODE1_SDR

MODE1_DDR

MODE2_SDR

MODE2_DDR

MODE4_SDR

MODE4_DDR

MODE8_SDR

MODE8_DDR

WRITE_SDR

WRITE_DDR

READ_SDR

READ_DDR

LEARN_SDR

LEARN_DDR

DATSZ_SDR

DATSZ_DDR

DUMMY_SDR

DUMMY_DDR

DUMMY_RWDS_SDR

DUMMY_RWDS_DDR

JMP_ON_CS

STOP

FLEXSPI_1PAD

FLEXSPI_2PAD

FLEXSPI_4PAD

FLEXSPI_8PAD

NOR_CMD_LUT_SEQ_IDX_READ
: NOR LUT sequence index used for default LUT assignment NOTE: The will take effect if the lut sequences are not customized. READ LUT sequence id in lookupTable stored in config block

NOR_CMD_LUT_SEQ_IDX_READSTATUS
: Read Status LUT sequence id in lookupTable stored in config block

NOR_CMD_LUT_SEQ_IDX_READSTATUS_XPI
: Read status DPI/QPI/OPI sequence id in lookupTable stored in config block

NOR_CMD_LUT_SEQ_IDX_WRITEENABLE
: Write Enable sequence id in lookupTable stored in config block

NOR_CMD_LUT_SEQ_IDX_WRITEENABLE_XPI
: Write Enable DPI/QPI/OPI sequence id in lookupTable stored in config block

NOR_CMD_LUT_SEQ_IDX_ERASESECTOR
: Erase Sector sequence id in lookupTable stored in config block

NOR_CMD_LUT_SEQ_IDX_READID

NOR_CMD_LUT_SEQ_IDX_ERASEBLOCK
: Erase Block sequence id in lookupTable stored in config block

NOR_CMD_LUT_SEQ_IDX_PAGEPROGRAM
: Program sequence id in lookupTable stored in config block

NOR_CMD_LUT_SEQ_IDX_CHIPERASE
: Chip Erase sequence in lookupTable id stored in config block

NOR_CMD_LUT_SEQ_IDX_READ_SFDP
: Read SFDP sequence in lookupTable id stored in config block

NOR_CMD_LUT_SEQ_IDX_RESTORE_NOCMD
: Restore 0-4-4/0-8-8 mode sequence id in lookupTable stored in config block

NOR_CMD_LUT_SEQ_IDX_EXIT_NOCMD
: Exit 0-4-4/0-8-8 mode sequence id in lookupTable stored in config blobk

uint32_t max_freq
: Maximum supported Frequency

uint32_t misc_mode
: miscellaneous mode

uint32_t quad_mode_setting
: Quad mode setting

uint32_t cmd_pads
: Command pads

uint32_t query_pads
: SFDP read pads

uint32_t device_type
: Device type

uint32_t option_size
: Option size, in terms of uint32_t, size = (option_size + 1) * 4

uint32_t tag
: Tag, must be 0x0E

struct *_serial_nor_config_option* B

uint32_t U

union *_serial_nor_config_option* option0

uint32_t dummy_cycles
: Dummy cycles before read

uint32_t status_override
: Override status register value during device mode configuration

uint32_t pinmux_group
: The pinmux group selection

uint32_t dqs_pinmux_group
: The DQS Pinmux Group Selection

uint32_t drive_strength
: The Drive Strength of FlexSPI Pads

uint32_t flash_connection
: Flash connection option: 0 - Single Flash connected to port A, 1 - Parallel mode, 2 - Single Flash connected to Port B

struct *_serial_nor_config_option* B

uint32_t U

union *_serial_nor_config_option* option1

uint8_t seqNum
: Sequence Number, valid number: 1-16

uint8_t seqId
: Sequence Index, valid number: 0-15

uint16_t reserved

uint8_t time_100ps
: Data valid time, in terms of 100ps

uint8_t delay_cells
: Data valid time, in terms of delay cells

uint32_t tag
: [0x000-0x003] Tag, fixed value 0x42464346UL

uint32_t version
: [0x004-0x007] Version,[31:24] -'V', [23:16] - Major, [15:8] - Minor, [7:0] - bugfix

uint32_t reserved0
: [0x008-0x00b] Reserved for future use

uint8_t readSampleClkSrc
: [0x00c-0x00c] Read Sample Clock Source, valid value: 0/1/3

uint8_t csHoldTime
: [0x00d-0x00d] Data hold time, default value: 3

uint8_t csSetupTime
: [0x00e-0x00e] Date setup time, default value: 3

uint8_t columnAddressWidth
: [0x00f-0x00f] Column Address with, for HyperBus protocol, it is fixed to 3, For Serial NAND, need to refer to datasheet

uint8_t deviceModeCfgEnable
: [0x010-0x010] Device Mode Configure enable flag, 1 - Enable, 0 - Disable

uint8_t deviceModeType
: [0x011-0x011] Specify the configuration command type:Quad Enable, DPI/QPI/OPI switch, Generic configuration, etc.

uint16_t waitTimeCfgCommands
: [0x012-0x013] Wait time for all configuration commands, unit: 100us, Used for DPI/QPI/OPI switch or reset command

flexspi_lut_seq_t deviceModeSeq
: [0x014-0x017] Device mode sequence info, [7:0] - LUT sequence id, [15:8] - LUt sequence number, [31:16] Reserved

uint32_t deviceModeArg
: [0x018-0x01b] Argument/Parameter for device configuration

uint8_t configCmdEnable
: [0x01c-0x01c] Configure command Enable Flag, 1 - Enable, 0 - Disable

uint8_t configModeType[3]
: [0x01d-0x01f] Configure Mode Type, similar as deviceModeTpe

flexspi_lut_seq_t configCmdSeqs[3]
: [0x020-0x02b] Sequence info for Device Configuration command, similar as deviceModeSeq

uint32_t reserved1
: [0x02c-0x02f] Reserved for future use

uint32_t configCmdArgs[3]
: [0x030-0x03b] Arguments/Parameters for device Configuration commands

uint32_t reserved2
: [0x03c-0x03f] Reserved for future use

uint32_t controllerMiscOption
: [0x040-0x043] Controller Misc Options, see Misc feature bit definitions for more details

uint8_t deviceType
: [0x044-0x044] Device Type: See Flash Type Definition for more details

uint8_t sflashPadType
: [0x045-0x045] Serial Flash Pad Type: 1 - Single, 2 - Dual, 4 - Quad, 8 - Octal

uint8_t serialClkFreq
: [0x046-0x046] Serial Flash Frequencey, device specific definitions, See System Boot Chapter for more details

uint8_t lutCustomSeqEnable
: [0x047-0x047] LUT customization Enable, it is required if the program/erase cannot be done using 1 LUT sequence, currently, only applicable to HyperFLASH

uint32_t reserved3[2]
: [0x048-0x04f] Reserved for future use

uint32_t sflashA1Size
: [0x050-0x053] Size of Flash connected to A1

uint32_t sflashA2Size
: [0x054-0x057] Size of Flash connected to A2

uint32_t sflashB1Size
: [0x058-0x05b] Size of Flash connected to B1

uint32_t sflashB2Size
: [0x05c-0x05f] Size of Flash connected to B2

uint32_t csPadSettingOverride
: [0x060-0x063] CS pad setting override value

uint32_t sclkPadSettingOverride
: [0x064-0x067] SCK pad setting override value

uint32_t dataPadSettingOverride
: [0x068-0x06b] data pad setting override value

uint32_t dqsPadSettingOverride
: [0x06c-0x06f] DQS pad setting override value

uint32_t timeoutInMs
: [0x070-0x073] Timeout threshold for read status command

uint32_t commandInterval
: [0x074-0x077] CS deselect interval between two commands

flexspi_dll_time_t dataValidTime[2]
: [0x078-0x07b] CLK edge to data valid time for PORT A and PORT B

uint16_t busyOffset
: [0x07c-0x07d] Busy offset, valid value: 0-31

uint16_t busyBitPolarity
: [0x07e-0x07f] Busy flag polarity, 0 - busy flag is 1 when flash device is busy, 1 - busy flag is 0 when flash device is busy

uint32_t lookupTable[64]
: [0x080-0x17f] Lookup table holds Flash command sequences

flexspi_lut_seq_t lutCustomSeq[12]
: [0x180-0x1af] Customizable LUT Sequences

uint32_t reserved4[4]
: [0x1b0-0x1bf] Reserved for future use

flexspi_mem_config_t memConfig
: Common memory configuration info via FLEXSPI

uint32_t pageSize
: Page size of Serial NOR

uint32_t sectorSize
: Sector size of Serial NOR

uint8_t ipcmdSerialClkFreq
: Clock frequency for IP command

uint8_t isUniformBlockSize
: Sector/Block size is the same

uint8_t isDataOrderSwapped
: Data order (D0, D1, D2, D3) is swapped (D1,D0, D3, D2)

uint8_t reserved0[1]
: Reserved for future use

uint8_t serialNorType
: Serial NOR Flash type: 0/1/2/3

uint8_t needExitNoCmdMode
: Need to exit NoCmd mode before other IP command

uint8_t halfClkForNonReadCmd
: Half the Serial Clock for non-read command: true/false

uint8_t needRestoreNoCmdMode
: Need to Restore NoCmd mode after IP commmand execution

uint32_t blockSize
: Block size

uint32_t reserve2[11]
: Reserved for future use

flexspi_operation_t operation
: FLEXSPI operation

uint32_t baseAddress
: FLEXSPI operation base address

uint32_t seqId
: Sequence Id

uint32_t seqNum
: Sequence Number

bool isParallelModeEnable
: Is a parallel transfer

uint32_t *txBuffer
: Tx buffer

uint32_t txSize
: Tx size in bytes

uint32_t *rxBuffer
Rx buffer

uint32_t rxSize
Rx size in bytes

FSL_ROM_ROMAPI_VERSION
ROM API version 1.0.2.

FSL_ROM_FLEXSPINOR_DRIVER_VERSION
ROM FLEXSPI NOR driver version 1.5.0.

struct _serial_nor_config_option
#include <fsl_romapi.h> Serial NOR Configuration Option.

struct _flexspi_lut_seq
#include <fsl_romapi.h> FLEXSPI LUT Sequence structure.

struct flexspi_dll_time_t
#include <fsl_romapi.h>

struct _flexspi_mem_config
#include <fsl_romapi.h> FLEXSPI Memory Configuration Block.

struct _flexspi_nor_config
#include <fsl_romapi.h> Serial NOR configuration block.

struct _flexspi_xfer
#include <fsl_romapi.h> FLEXSPI Transfer Context.

union option0

Public Members

struct *_serial_nor_config_option* B

uint32_t U

struct B

Public Members

uint32_t max_freq
Maximum supported Frequency

uint32_t misc_mode
miscellaneous mode

uint32_t quad_mode_setting
Quad mode setting

uint32_t cmd_pads
Command pads

uint32_t query_pads
SFDP read pads

uint32_t device_type
Device type

uint32_t option_size
Option size, in terms of uint32_t, size = (option_size + 1) * 4

uint32_t tag
Tag, must be 0x0E

union option1

Public Members

struct *_serial_nor_config_option* B

uint32_t U

struct B

Public Members

uint32_t dummy_cycles
Dummy cycles before read

uint32_t status_override
Override status register value during device mode configuration

uint32_t pinmux_group
The pinmux group selection

uint32_t dqs_pinmux_group
The DQS Pinmux Group Selection

uint32_t drive_strength
The Drive Strength of FlexSPI Pads

uint32_t flash_connection
Flash connection option: 0 - Single Flash connected to port A, 1 - Parallel mode, 2 - Single Flash connected to Port B

1.61 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.

AT_QUICKACCESS_SECTION_CODE (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

1.62 SAI: Serial Audio Interface

1.63 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 2.4.7

_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.

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 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

1.64 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 2.7.3

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

1.65 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.

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 readHoldTime_Ns
: read hold 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_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.

1.66 SNVS: Secure Non-Volatile Storage

1.67 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

1.68 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

1.69 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_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 2.0.7

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

1.70 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 2.0.8

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

1.71 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.

1.72 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.

1.73 TRNG: True Random Number Generator

FSL_TRNG_DRIVER_VERSION

TRNG driver version 2.0.18.

Current version: 2.0.18

Change log:

- 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.

1.74 TSC: Touch Screen Controller Driver

enum _tsc_detection_mode

@ Controller detection mode.

Values:

enumerator kTSC_Detection4WireMode

4-Wire Detection Mode.

enumerator kTSC_Detection5WireMode

5-Wire Detection Mode.

enum _tsc_corrdinate_value_selection

@ Coordinate value mask.

Values:

enumerator kTSC_XCoordinateValueSelection

X coordinate value is selected.

enumerator kTSC_YCoordinateValueSelection

Y coordinate value is selected.

enum _tsc_interrupt_signal_mask

@ Interrupt signal enable/disable mask.

Values:

enumerator kTSC_IdleSoftwareSignalEnable

Enable the interrupt signal when the controller has return to idle status. The signal is only valid after using TSC_ReturnToIdleStatus API.

enumerator kTSC_ValidSignalEnable

Enable the interrupt signal when controller receives a detect signal after measurement.

enumerator kTSC_DetectSignalEnable

Enable the interrupt signal when controller receives a detect signal.

enumerator kTSC_MeasureSignalEnable

Enable the interrupt signal after the touch detection which follows measurement.

enum _tsc_interrupt_mask

@ Interrupt enable/disable mask.

Values:

enumerator kTSC_IdleSoftwareInterruptEnable

Enable the interrupt when the controller has return to idle status. The interrupt is only valid after using TSC_ReturnToIdleStatus API.

enumerator kTSC_DetectInterruptEnable

Enable the interrupt when controller receive a detect signal.

enumerator kTSC_MeasureInterruptEnable

Enable the interrupt after the touch detection which follows measurement.

enum _tsc_interrupt_status_flag_mask

@ Interrupt Status flag mask.

Values:

enumerator kTSC_IdleSoftwareFlag

This flag is set if the controller has return to idle status. The flag is only valid after using TSC_ReturnToIdleStatus API.

enumerator kTSC_ValidSignalFlag

This flag is set if controller receives a detect signal after measurement.

enumerator kTSC_DetectSignalFlag

This flag is set if controller receives a detect signal.

enumerator kTSC_MeasureSignalFlag

This flag is set after the touch detection which follows measurement. Note: Valid signal falg will be cleared along with measure signal flag.

enum _tsc_adc_status_flag_mask

@ ADC status flag mask.

Values:

enumerator kTSC_ADCCOCOSignalFlag

This signal is generated by ADC when a conversion is completed.

enumerator kTSC_ADCConversionValueFlag

This signal is generated by ADC and indicates the result of an ADC conversion.

enum _tsc_status_flag_mask

@ TSC status flag mask.

Values:

enumerator kTSC_IntermediateStateFlag

This flag is set if TSC is in intermediate state, between two state machine states.

enumerator kTSC_DetectFiveWireFlag

This flag is set if TSC receives a 5-wire detect signal. It is only valid when the TSC in detect state and DETECT_ENABLE_FIVE_WIRE bit is set.

enumerator kTSC_DetectFourWireFlag

This flag is set if TSC receives a 4-wire detect signal. It is only valid when the TSC in detect state and DETECT_ENABLE_FOUR_WIRE bit is set.

enumerator kTSC_GlitchThresholdFlag

This field indicates glitch threshold.The threshold is defined by number of clock cycles. See “tsc_glitch_threshold_t”. If value = 00, Normal function: 0x1fff ipg clock cycles, Low power mode: 0x9 low power clock cycles. If value = 01, Normal function: 0xfff ipg clock cycles, Low power mode: :0x7 low power clock cycles. If value = 10, Normal function: 0x7ff ipg clock cycles, Low power mode:0x5 low power clock cycles. If value = 11, Normal function: 0x3 ipg clock cycles, Low power mode:0x3 low power clock cycles.

enumerator kTSC_StateMachineFlag

This field indicates the state of TSC. See “tsc_state_machine_t”; if value = 000, Controller is in idle state. if value = 001, Controller is in 1st-Pre-charge state. if value = 010, Controller is in 1st-detect state. if value = 011, Controller is in x-measure state. if value = 100, Controller is in y-measure state. if value = 101, Controller is in 2nd-Pre-charge state. if value = 110, Controller is in 2nd-detect state.

enum _tsc_state_machine

TSC state machine. These seven states are TSC complete workflow.

Values:

enumerator kTSC_IdleState

Controller is in idle state.

enumerator kTSC_1stPreChargeState

Controller is in 1st-Pre-charge state.

enumerator kTSC_1stDetectState
Controller is in 1st-detect state.

enumerator kTSC_XMeasureState
Controller is in x-measure state.

enumerator kTSC_YMeasureState
Controller is in y-measure state.

enumerator kTSC_2ndPreChargeState
Controller is in 2nd-Pre-charge state.

enumerator kTSC_2ndDetectState
Controller is in 2nd-detect state.

enum _tsc_glitch_threshold
TSC glitch threshold.

Values:

enumerator kTSC_glitchThresholdALT0
Normal function: 0x1fff ipg clock cycles, Low power mode: 0x9 low power clock cycles.

enumerator kTSC_glitchThresholdALT1
Normal function: 0xfff ipg clock cycles, Low power mode: :0x7 low power clock cycles.

enumerator kTSC_glitchThresholdALT2
Normal function: 0x7ff ipg clock cycles, Low power mode: :0x5 low power clock cycles.

enumerator kTSC_glitchThresholdALT3
Normal function: 0x3 ipg clock cycles, Low power mode: :0x3 low power clock cycles.

enum _tsc_trigger_signal
@ Hardware trigger select signal, select which ADC channel to start conversion.

Values:

enumerator kTSC_TriggerToChannel0
Trigger to ADC channel0. ADC_HC0 register will be used to conversion.

enumerator kTSC_TriggerToChannel1
Trigger to ADC channel1. ADC_HC1 register will be used to conversion.

enumerator kTSC_TriggerToChannel2
Trigger to ADC channel2. ADC_HC2 register will be used to conversion.

enumerator kTSC_TriggerToChannel3
Trigger to ADC channel3. ADC_HC3 register will be used to conversion.

enumerator kTSC_TriggerToChannel4
Trigger to ADC channel4. ADC_HC4 register will be used to conversion.

enum _tsc_port_source
@ TSC controller ports.

Values:

enumerator kTSC_WiperPortSource
TSC controller wiper port.

enumerator kTSC_YnlrPortSource
TSC controller ynlr port.

enumerator kTSC_YpllPortSource
TSC controller ypll port.

enumerator kTSC_XnurPortSource
TSC controller xnur port.

enumerator kTSC_XpulPortSource
TSC controller xpul port.

enum _tsc_port_mode
@ TSC port mode.

Values:

enumerator kTSC_PortOffMode
Disable pull up/down mode.

enumerator kTSC_Port200k_PullUpMode
200k-pull up mode.

enumerator kTSC_PortPullUpMode
Pull up mode.

enumerator kTSC_PortPullDownMode
Pull down mode.

typedef enum *_tsc_detection_mode* tsc_detection_mode_t
@ Controller detection mode.

typedef enum *_tsc_corrdinate_value_selection* tsc_corrdinate_value_selection_t
@ Coordinate value mask.

typedef enum *_tsc_state_machine* tsc_state_machine_t
TSC state machine. These seven states are TSC complete workflow.

typedef enum *_tsc_glitch_threshold* tsc_glitch_threshold_t
TSC glitch threshold.

typedef enum *_tsc_trigger_signal* tsc_trigger_signal_t
@ Hardware trigger select signal, select which ADC channel to start conversion.

typedef enum *_tsc_port_source* tsc_port_source_t
@ TSC controller ports.

typedef enum *_tsc_port_mode* tsc_port_mode_t
@ TSC port mode.

typedef struct *_tsc_config* tsc_config_t
@ Controller configuration.

void TSC_Init(TSC_Type *base, const *tsc_config_t* *config)
Initialize the TSC module.

Parameters

- base – TSC peripheral base address.
- config – Pointer to “tsc_config_t” structure.

void TSC_Deinit(TSC_Type *base)
De-initializes the TSC module.

Parameters

- base – TSC peripheral base address.

void TSC_GetDefaultConfig(*tsc_config_t* *config)

Gets an available pre-defined settings for the controller's configuration.

This function initializes the converter configuration structure with available settings. The default values of measureDelayTime and prechargeTime is tested on LCD8000-43T screen and work normally. The default values are:

```
config->enableAutoMeausre = false;
config->measureDelayTime = 0xFFFFU;
config->prechargeTime = 0xFFFFU;
config->detectionMode = kTSC_4WireDetectionMode;
```

Parameters

- config – Pointer to "tsc_config_t" structure.

static inline void TSC_ReturnToIdleStatus(TSC_Type *base)

Make the TSC module return to idle status after finish the current state operation. Application could check TSC status to confirm that the controller has return to idle status.

Parameters

- base – TSC peripheral base address.

static inline void TSC_StartSenseDetection(TSC_Type *base)

Start sense detection and (if work in auto-measure mode) measure after detect a touch.

Parameters

- base – TSC peripheral base address.

static inline void TSC_StartMeasure(TSC_Type *base)

start measure X/Y coordinate value after detect a touch.

Parameters

- base – TSC peripheral base address.

static inline void TSC_DropMeasure(TSC_Type *base)

Drop measure X/Y coordinate value after detect a touch and controller return to idle status.

Parameters

- base – TSC peripheral base address.

static inline void TSC_SoftwareReset(TSC_Type *base)

This is a synchronization reset, which resets every register except IPS directly access ones.

Parameters

- base – TSC peripheral base address.

uint32_t TSC_GetMeasureValue(TSC_Type *base, *tsc_corrdinate_value_selection_t* selection)

Get Y coordinate value or X coordinate value. The value is an ADC conversion value.

Parameters

- base – TSC peripheral base address.
- selection – Select alternative measure value which is Y coordinate value or X coordinate value. See "tsc_corrdinate_value_selection_t".

Returns

If selection is "kTSC_XCoordinateValueSelection", the API returns x-coordinate vlaue. If selection is "kTSC_YCoordinateValueSelection", the API returns y-coordinate vlaue.

static inline void TSC_EnableInterruptSignals(TSC_Type *base, uint32_t mask)

Enable the interrupt signals. Interrupt signal will be set when corresponding event happens. Specific events point to "_tsc_interrupt_signal_mask" . Specific interrupt signal point to "_tsc_interrupt_status_flag_mask";.

Parameters

- base – TSC peripheral base address.
- mask – Interrupt signals mask. See "_tsc_interrupt_signal_mask".

static inline void TSC_DisableInterruptSignals(TSC_Type *base, uint32_t mask)

Disable the interrupt signals. Interrupt signal will be set when corresponding event happens. Specific events point to "_tsc_interrupt_signal_mask". Specific interrupt signal point to "_tsc_interrupt_status_flag_mask";.

Parameters

- base – TSC peripheral base address.
- mask – Interrupt signals mask. See "_tsc_interrupt_signal_mask".

static inline void TSC_EnableInterrupts(TSC_Type *base, uint32_t mask)

Enable the interrupts. Notice: Only interrupts and signals are all enabled, interrupts could work normally.

Parameters

- base – TSC peripheral base address.
- mask – Interrupts mask. See "_tsc_interrupt_mask".

static inline void TSC_DisableInterrupts(TSC_Type *base, uint32_t mask)

Disable the interrupts.

Parameters

- base – TSC peripheral base address.
- mask – Interrupts mask. See "_tsc_interrupt_mask".

static inline uint32_t TSC_GetInterruptStatusFlags(TSC_Type *base)

Get interrupt status flags. Interrupt status falgs are valid when corresponding interrupt signals are enabled.

Parameters

- base – TSC peripheral base address.

Returns

Status flags asserted mask. See "_tsc_interrupt_status_flag_mask".

static inline void TSC_ClearInterruptStatusFlags(TSC_Type *base, uint32_t mask)

Clear interrupt status flags. Interrupt status falgs are valid when corresponding interrupt signals are enabled.

Parameters

- base – TSC peripheral base address.
- mask – Status flags mask. See "_tsc_interrupt_status_flag_mask".

static inline uint32_t TSC_GetADCStatusFlags(TSC_Type *base)

Get the status flags of ADC working with TSC.

Parameters

- base – TSC peripheral base address.

Returns
Status flags asserted mask. See "_tsc_adc_status_flag_mask".

static inline uint32_t TSC_GetStatusFlags(TSC_Type *base)

Get the status flags of TSC.

Parameters

- base – TSC peripheral base address.

Returns
Status flags asserted mask. See "_tsc_status_flag_mask".

FSL_TSC_DRIVER_VERSION

TSC driver version.

Version 2.0.3.

struct _tsc_config

#include <fsl_tsc.h> @ Controller configuration.

Public Members

bool enableAutoMeasure

Enable the auto-measure. It indicates after detect touch, whether automatic start measurement

uint32_t measureDelayTime

Set delay time(0U~0xFFFFFFU) to even potential distribution ready.It is a preparation for measure stage. If measure dalay time is too short, maybe it would have an undesired effect on measure value.

uint32_t prechargeTime

Set pre-charge time(1U~0xFFFFFFFFU) to make the upper layer of screen to charge to positive high. It is a preparation for detection stage. Pre-charge time must is greater than 0U, otherwise TSC could not work normally. If pre-charge dalay time is too short, maybe it would have an undesired effect on generation of valid signal(kTSC_ValidSignalFlag).

tsc_detection_mode_t detectionMode

Select the detection mode. See "tsc_detection_mode_t".

1.75 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.

FSL_USDHC_DRIVER_VERSION

Driver version 2.8.5.

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.

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.

1.76 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;
```

See also:

wdog_config_t

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.

1.77 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.

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.

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.

1.78 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 2

Middleware

2.1 Boot

2.1.1 MCUXpresso SDK : mcuxsdk-middleware-mcuboot_opensource

Overview

This repository is a fork of MCUboot (https://github.com/mcu-tools/mcuboot) 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 MCUboot - 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 MCUboot project. Once this MCUboot 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 MCUboot contains specific modifications and enhancements for NXP MCUXpresso SDK integration.

See *changelog* for details.

2.1.2 MCUboot



This is MCUboot version 2.1.0

MCUboot is a secure bootloader for 32-bits microcontrollers. It defines a common infrastructure for the bootloader and the system flash layout on microcontroller systems, and provides a secure bootloader that enables easy software upgrade.

MCUboot is not dependent on any specific operating system and hardware and relies on hardware porting layers from the operating system it works with. Currently, MCUboot works with the following operating systems and SoCs:

- Zephyr
- Apache Mynewt
- Apache NuttX
- RIOT
- Mbed OS
- Espressif
- Cypress/Infineon

RIOT is supported only as a boot target. We will accept any new port contributed by the community once it is good enough.

MCUboot How-tos

See the following pages for instructions on using MCUboot with different operating systems and SoCs:

- Zephyr
- Apache Mynewt
- Apache NuttX
- RIOT
- Mbed OS
- Espressif
- *Cypress/Infineon*

There are also instructions for the *Simulator*.

Roadmap

The issues being planned and worked on are tracked using GitHub issues. To give your input, visit MCUboot GitHub Issues.

Source files

You can find additional documentation on the bootloader in the source files. For more information, use the following links:

- boot/bootutil - The core of the bootloader itself.
- boot/boot_serial - Support for serial upgrade within the bootloader itself.
- boot/zephyr - Port of the bootloader to Zephyr.
- boot/mynewt - Bootloader application for Apache Mynewt.
- boot/nuttx - Bootloader application and port of MCUboot interfaces for Apache NuttX.
- boot/mbed - Port of the bootloader to Mbed OS.
- boot/espressif - Bootloader application and MCUboot port for Espressif SoCs.
- boot/cypress - Bootloader application and MCUboot port for Cypress/Infineon SoCs.
- imgtool - A tool to securely sign firmware images for booting by MCUboot.
- sim - A bootloader simulator for testing and regression.

Joining the project

Developers are welcome!

Use the following links to join or see more about the project:

- Our developer mailing list
- Our Slack channel Get your invite

2.2 Cloud

2.2.1 AWS IoT

Device Shadow Library

AWS IoT Device Shadow library The AWS IoT Device Shadow library enables you to store and retrieve the current state (the "shadow") of every registered device. The device's shadow is a persistent, virtual representation of your device that you can interact with from AWS IoT Core even if the device is offline. The device state is captured as its "shadow" within a JSON document. The device can send commands over MQTT to get, update and delete its latest state as well as receive notifications over MQTT about changes in its state. Each device's shadow is uniquely identified by the name of the corresponding "thing", a representation of a specific device or logical entity on the AWS Cloud. See Managing Devices with AWS IoT for more information on IoT "thing". More details about AWS IoT Device Shadow can be found in AWS IoT documentation. This library is distributed under the *MIT Open Source License*.

Note: From v1.1.0 release onwards, you can used named shadow, a feature of the AWS IoT Device Shadow service that allows you to create multiple shadows for a single IoT device.

This library has gone through code quality checks including verification that no function has a GNU Complexity score over 8, and checks against deviations from mandatory rules in the MISRA coding standard. Deviations from the MISRA C:2012 guidelines are documented under *MISRA Deviations*. This library has also undergone both static code analysis from Coverity static analysis, and validation of memory safety through the CBMC automated reasoning tool.

See memory requirements for this library here.

AWS IoT Device Shadow v1.3.0 source code is part of the FreeRTOS 202210.00 LTS release.

AWS IoT Device Shadow v1.0.2 source code is part of the FreeRTOS 202012.00 LTS release.

AWS IoT Device Shadow Config File The AWS IoT Device Shadow library exposes configuration macros that are required for building the library. A list of all the configurations and their default values are defined in *shadow_config_defaults.h*. To provide custom values for the configuration macros, a custom config file named shadow_config.h can be provided by the user application to the library.

By default, a shadow_config.h custom config is required to build the library. To disable this requirement and build the library with default configuration values, provide SHADOW_DO_NOT_USE_CUSTOM_CONFIG as a compile time preprocessor macro.

Building the Library The *shadowFilePaths.cmake* file contains the information of all source files and the header include path required to build the AWS IoT Device Shadow library.

As mentioned in the *previous section*, either a custom config file (i.e. shadow_config.h) OR the SHADOW_DO_NOT_USE_CUSTOM_CONFIG macro needs to be provided to build the AWS IoT Device Shadow library.

For a CMake example of building the AWS IoT Device Shadow library with the shadowFilePaths.cmake file, refer to the coverity_analysis library target in *test/CMakeLists.txt* file.

Building Unit Tests

Checkout CMock Submodule By default, the submodules in this repository are configured with update=none in *.gitmodules* to avoid increasing clone time and disk space usage of other repositories (like amazon-freertos that submodules this repository).

To build unit tests, the submodule dependency of CMock is required. Use the following command to clone the submodule:

```
git submodule update --checkout --init --recursive --test/unit-test/CMock
```

Platform Prerequisites

- For building the library, **CMake 3.13.0** or later and a **C90 compiler**.
- For running unit tests, **Ruby 2.0.0** or later is additionally required for the CMock test framework (that we use).
- For running the coverage target, **gcov** and **lcov** are additionally required.

Steps to build unit tests

1. Go to the root directory of this repository. (Make sure that the **CMock** submodule is cloned as described *above*.)
2. Run the *cmake* command: cmake -S test -B build

3. Run this command to build the library and unit tests: `make -C build all`
4. The generated test executables will be present in `build/bin/tests` folder.
5. Run `cd build && ctest` to execute all tests and view the test run summary.

CBMC To learn more about CBMC and proofs specifically, review the training material here.

The `test/cbmc/proofs` directory contains CBMC proofs.

In order to run these proofs you will need to install CBMC and other tools by following the instructions here.

Reference examples Please refer to the demos of the AWS IoT Device Shadow library in the following locations for reference examples on POSIX and FreeRTOS platforms:

Platform	Location	Transport Interface Implementation (for coreMQTT stack)
POSIX	AWS IoT Device SDK for Embedded C	POSIX sockets for TCP/IP and OpenSSL for TLS stack
FreeRTOS	FreeRTOS/FreeRTOS	FreeRTOS+TCP for TCP/IP and mbedTLS for TLS stack
FreeRTOS	FreeRTOS AWS Reference Integrations	Based on Secure Sockets Abstraction

Documentation

Existing Documentation For pre-generated documentation, please see the documentation linked in the locations below:

Location
AWS IoT Device SDK for Embedded C
FreeRTOS.org

Note that the latest included version of IoT Device Shadow library may differ across repositories.

Generating documentation The Doxygen references were created using Doxygen version 1.9.2. To generate the Doxygen pages, please run the following command from the root of this repository:

```
doxygen docs/doxygen/config.doxyfile
```

Contributing See *CONTRIBUTING.md* for information on contributing.

Device Defender Library

AWS IoT Device Defender Library The Device Defender library enables you to send device metrics to the AWS IoT Device Defender Service. This library also supports custom metrics, a feature that helps you monitor operational health metrics that are unique to your fleet or use case. For example, you can define a new metric to monitor the memory usage or CPU usage

on your devices. This library has no dependencies on any additional libraries other than the standard C library, and therefore, can be used with any MQTT client library. This library is distributed under the *MIT Open Source License*.

This library has gone through code quality checks including verification that no function has a GNU Complexity score over 8, and checks against deviations from mandatory rules in the MISRA coding standard. Deviations from the MISRA C:2012 guidelines are documented under *MISRA Deviations*. This library has also undergone static code analysis using Coverity static analysis, and validation of memory safety through the CBMC automated reasoning tool.

See memory requirements for this library *here*.

AWS IoT Device Defender v1.3.0 source code is part of the FreeRTOS 202210.00 LTS release.

AWS IoT Device Defender v1.1.0 source code is part of the FreeRTOS 202012.01 LTS release.

AWS IoT Device Defender Client Config File The AWS IoT Device Defender Client Library exposes build configuration macros that are required for building the library. A list of all the configurations and their default values are defined in *defender_config_defaults.h*. To provide custom values for the configuration macros, a config file named defender_config.h can be provided by the application to the library.

By default, a defender_config.h config file is required to build the library. To disable this requirement and build the library with default configuration values, provide DEFENDER_DO_NOT_USE_CUSTOM_CONFIG as a compile time preprocessor macro.

Thus, the Device Defender client library can be built by either:

- Defining a defender_config.h file in the application, and adding it to the include directories list of the library.

OR

- Defining the DEFENDER_DO_NOT_USE_CUSTOM_CONFIG preprocessor macro for the library build.

Building the Library The *defenderFilePaths.cmake* file contains the information of all source files and the header include paths required to build the Device Defender client library.

As mentioned in the previous section, either a custom config file (i.e. defender_config.h) or DEFENDER_DO_NOT_USE_CUSTOM_CONFIG macro needs to be provided to build the Device Defender client library.

For a CMake example of building the Device Defender client library with the defenderFilePaths.cmake file, refer to the coverity_analysis library target in *test/CMakeLists.txt* file.

Building Unit Tests

Platform Prerequisites

- For running unit tests:
  - **C90 compiler** like gcc.
  - **CMake 3.13.0 or later**.
  - **Ruby 2.0.0 or later** is additionally required for the CMock test framework (that we use).
- For running the coverage target, **gcov** and **lcov** are additionally required.

Steps to build Unit Tests

1. Go to the root directory of this repository.
2. Run the *cmake* command: `cmake -S test -B build -DBUILD_CLONE_SUBMODULES=ON`.
3. Run this command to build the library and unit tests: `make -C build all`.
4. The generated test executables will be present in `build/bin/tests` folder.
5. Run `cd build && ctest` to execute all tests and view the test run summary.

CBMC To learn more about CBMC and proofs specifically, review the training material here.

The `test/cbmc/proofs` directory contains CBMC proofs.

In order to run these proofs you will need to install CBMC and other tools by following the instructions here.

Reference examples The AWS IoT Embedded C-SDK repository contains a demo showing the use of AWS IoT Device Defender Client Library here on a POSIX platform.

Documentation

Existing documentation For pre-generated documentation, please see the documentation linked in the locations below:

Location
AWS IoT Device SDK for Embedded C
FreeRTOS.org

Note that the latest included version of the AWS IoT Device Defender library may differ across repositories.

Generating documentation The Doxygen references were created using Doxygen version 1.9.2. To generate the Doxygen pages, please run the following command from the root of this repository:

```
doxygen docs/doxygen/config.doxyfile
```

Contributing See *CONTRIBUTING.md* for information on contributing.

Jobs Library

README

AWS IoT Jobs library The AWS IoT Jobs library helps you notify connected IoT devices of a pending Job. A Job can be used to manage your fleet of devices, update firmware and security certificates on your devices, or perform administrative tasks such as restarting devices and performing diagnostics. It interacts with the AWS IoT Jobs service using MQTT, a lightweight publish-subscribe protocol. This library provides a convenience API to compose and recognize the MQTT topic strings used by the Jobs service. The library is written in C compliant with ISO C90 and MISRA C:2012, and is distributed under the *MIT Open Source License*.

This library has gone through code quality checks including verification that no function has a GNU Complexity score over 8, and checks against deviations from mandatory rules in the MISRA coding standard . Deviations from the MISRA C:2012 guidelines are documented under *MISRA Deviations*. This library has also undergone both static code analysis from Coverity, and validation of memory safety with the CBMC bounded model checker.

See memory requirements for this library *here*.

AWS IoT Jobs v1.3.0 source code is part of the FreeRTOS 202210.00 LTS release.

AWS IoT Jobs v1.1.0 source code is part of the FreeRTOS 202012.01 LTS release.

Building the Jobs library A compiler that supports **C90 or later** such as *gcc* is required to build the library.

Given an application in a file named `example.c`, *gcc* can be used like so:

```
gcc -I source/include example.c source/jobs.c -o example
```

gcc can also produce an object file to be linked later:

```
gcc -I source/include -c source/jobs.c
```

CBMC To learn more about CBMC and proofs specifically, review the training material here.

The `test/cbmc/proofs` directory contains CBMC proofs.

In order to run these proofs you will need to install CBMC and other tools by following the instructions here.

Reference example The AWS IoT Device SDK for Embedded C repository contains a demo using the jobs library on a POSIX platform. https://github.com/aws/aws-iot-device-sdk-embedded-C/tree/main/demos/jobs/jobs_demo_mosquitto

Documentation

Existing Documentation For pre-generated documentation, please see the documentation linked in the locations below:

Location
AWS IoT Device SDK for Embedded C
FreeRTOS.org

Note that the latest included version of the AWS IoT Jobs library may differ across repositories.

Generating Documentation The Doxygen references were created using Doxygen version 1.9.2. To generate the Doxygen pages, please run the following command from the root of this repository:

```
doxygen docs/doxygen/config.doxyfile
```

Building unit tests

Checkout Unity Submodule By default, the submodules in this repository are configured with `update=none` in *.gitmodules* to avoid increasing clone time and disk space usage of other repositories that submodule this repository.

To build unit tests, the submodule dependency of Unity is required. Use the following command to clone the submodule:

```
git submodule update --checkout --init --recursive --test/unit-test/Unity
```

Platform Prerequisites

- For running unit tests
  - C90 compiler like gcc
  - CMake 3.13.0 or later
  - Ruby 2.0.0 or later is additionally required for the Unity test framework (that we use).
- For running the coverage target, lcov is additionally required.

Steps to build Unit Tests

1. Go to the root directory of this repository. (Make sure that the **Unity** submodule is cloned as described *above*.)
2. Create build directory: `mkdir build && cd build`
3. Run *cmake* while inside build directory: `cmake -S ../test`
4. Run this command to build the library and unit tests: `make all`
5. The generated test executables will be present in `build/bin/tests` folder.
6. Run `ctest` to execute all tests and view the test run summary.

Contributing See *CONTRIBUTING.md* for information on contributing.

Over-the-air Update Library

AWS IoT Over-the-air Update Library The OTA library enables you to manage the notification of a newly available update, download the update, and perform cryptographic verification of the firmware update. Using the library, you can logically separate firmware updates from the application running on your devices. The OTA library can share a network connection with the application, saving memory in resource-constrained devices. In addition, the OTA library lets you define application-specific logic for testing, committing, or rolling back a firmware update. The library supports different application protocols like Message Queuing Telemetry Transport (MQTT) and Hypertext Transfer Protocol (HTTP), and provides various configuration options you can fine tune depending on network type and conditions. This library is distributed under the *MIT Open Source License*.

This library has gone through code quality checks including verification that no function has a GNU Complexity score over 8. This library has also undergone static code analysis from Coverity static analysis.

See memory requirements for this library *here*.

AWS IoT Over-the-air Update Library v3.4.0 source code is part of the FreeRTOS 202210.00 LTS release.

AWS IoT Over-the-air Update Library v3.3.0 source code is part of the FreeRTOS 202012.01 LTS release.

AWS IoT Over-the-air Updates Config File The AWS IoT Over-the-air Updates library exposes configuration macros that are required for building the library. A list of all the configurations and their default values are defined in *ota_config_defaults.h*. To provide custom values for the configuration macros, a custom config file named ota_config.h can be provided by the user application to the library.

By default, a ota_config.h custom config is required to build the library. To disable this requirement and build the library with default configuration values, provide OTA_DO_NOT_USE_CUSTOM_CONFIG as a compile time preprocessor macro.

Building the Library The *otaFilePaths.cmake* file contains the information of all source files and the header include paths required to build the AWS IoT Over-the-air Updates library.

As mentioned in the previous section, either a custom config file (i.e. ota_config.h) OR the OTA_DO_NOT_USE_CUSTOM_CONFIG macro needs to be provided to build the AWS IoT Over-the-air Updates library.

For a CMake example of building the AWS IoT Over-the-air Updates library with the otaFilePaths.cmake file, refer to the coverity_analysis library target in the *test/CMakeLists.txt* file.

Building Unit Tests

Checkout CMock Submodule By default, the submodules in this repository are configured with update=none in *.gitmodules* to avoid increasing clone time and disk space usage of other repositories (like AWS IoT Device SDK for Embedded C that submodules this repository).

To build unit tests, the submodule dependency of CMock is required. Use the following command to clone the submodule:

```
git submodule update --checkout --init --recursive test/unit-test/CMock
```

Platform Prerequisites

- For building the library, **CMake 3.13.0** or later and a **C90 compiler**.
- For running unit tests, **Ruby 2.0.0** or later is additionally required for the CMock test framework (that we use).
- For running the coverage target, **gcov** and **lcov** are additionally required.

Steps to build unit tests

1. Go to the root directory of this repository. (Make sure that the **CMock** submodule is cloned as described *above*.)
2. Run the *cmake* command: cmake -S test -B build

3. Run this command to build the library and unit tests: `make -C build all`
4. The generated test executables will be present in `build/bin/tests` folder.
5. Run `cd build && ctest` to execute all tests and view the test run summary.

Migration Guide

How to migrate from v2.0.0 (Release Candidate) to v3.4.0 The following table lists equivalent API function signatures in v2.0.0 (Release Candidate) and v3.4.0 declared in *ota.h*

v2.0.0 (Release Candidate)	v3.4.0	Notes
`OtaState_t OTA_Shutdown( uint32_t ticksToWait );`	`OtaState_t OTA_Shutdown( uint32_t ticksToWait, uint8_t unsubscribeFlag );`	`unsubscribeFlag` indicates if unsubscribe operations should be performed from the job topics when shutdown is called. Set this as 1 to unsubscribe, 0 otherwise.

How to migrate from version 1.0.0 to version 3.4.0 for OTA applications Refer to OTA Migration document for the summary of updates to the API. Migration document for OTA PAL also provides a summary of updates required for upgrading the OTA-PAL to work with v3.4.0 of the library.

Porting In order to support AWS IoT Over-the-air Updates on your device, it is necessary to provide the following components:

1. Port for the OTA Portable Abstraction Layer (PAL).
2. OS Interface
3. MQTT Interface

For enabling data transfer over HTTP dataplane the following component should also be provided:

1. HTTP Interface

NOTE When using OTA over HTTP dataplane, MQTT is required for control plane operations and should also be provided.

CBMC To learn more about CBMC and proofs specifically, review the training material here.

The `test/cbmc/proofs` directory contains CBMC proofs.

In order to run these proofs you will need to install CBMC and other tools by following the instructions here.

Reference examples Please refer to the demos of the AWS IoT Over-the-air Updates library in the following location for reference examples on POSIX and FreeRTOS:

Platform	Location
POSIX	AWS IoT Device SDK for Embedded C
FreeRTOS	FreeRTOS/FreeRTOS
FreeRTOS	FreeRTOS AWS Reference Integrations

Documentation

Existing Documentation For pre-generated documentation, please see the documentation linked in the locations below:

Location
AWS IoT Device SDK for Embedded C
FreeRTOS.org

Note that the latest included version of coreMQTT may differ across repositories.

Generating documentation The Doxygen references were created using Doxygen version 1.9.2. To generate the Doxygen pages, please run the following command from the root of this repository:

```
doxygen docs/doxygen/config.doxyfile
```

Contributing See *CONTRIBUTING.md* for information on contributing.

2.3 Connectivity

2.3.1 lwIP

This is the NXP fork of the lwIP networking stack.

- For details about changes and additions made by NXP, see CHANGELOG.
- For details about the NXP porting layer, see *The NXP lwIP Port*.
- For usage and API of lwIP, use official documentation at http://www.nongnu.org/lwip/.

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.

2.4 eIQ

2.4.1 eIQ

eIQ TensorFlow Lite for Micro Library User Guide

- *Overview*
- *TensorFlow Lite for Microcontrollers*
- *Build Status*
  - *Official Builds*
  - *Community Supported TFLM Examples*
  - *Community Supported Kernels and Unit Tests*
- *Contributing*

- *Getting Help*
- *Additional Documentation*
- *RFCs*

Overview TensorFlow Lite is an open source software library for running machine learning models on mobile and embedded devices. For more information, see www.tensorflow.org/lite.

For memory constrained devices, the library contains TensorFlow Lite for Microcontrollers. For more information, see *www.tensorflow.org/lite/microcontrollers*.

The MCUXpresso Software Development Kit (MCUXpresso SDK) provides a comprehensive software package with a pre-integrated TensorFlow Lite for Microcontrollers based on version 25-04-08 (from the 8th of April 2025 with commit). This document describes the steps required to download and start using the library. Additionally, the document describes the steps required to create an application for running pre-trained models.

Note: The document also assumes knowledge of machine learning frameworks for model training.

TensorFlow Lite for Microcontrollers TensorFlow Lite for Microcontrollers is a port of TensorFlow Lite designed to run machine learning models on DSPs, microcontrollers and other devices with limited memory.

Additional Links:

- Tensorflow github repository
- TFLM at tensorflow.org

Build Status

- GitHub Status

Official Builds

Build Type	Status
CI (Linux)	Run-CI passing
Code Sync	Sync from Upstream TF passing

Community Supported TFLM Examples This table captures platforms that TFLM has been ported to. Please see *New Platform Support* for additional documentation.

Platform	Status
Arduino	CI no status Arduino examples tests no status
Coral Dev Board Micro	TFLM + EdgeTPU Examples for Coral Dev Board Micro
Espressif Systems Dev Boards	CI passing
Renesas Boards	TFLM Examples for Renesas Boards
Silicon Labs Dev Kits	TFLM Examples for Silicon Labs Dev Kits
Sparkfun Edge	CI no status
Texas Instruments Dev Boards	

Community Supported Kernels and Unit Tests This is a list of targets that have optimized kernel implementations and/or run the TFLM unit tests using software emulation or instruction set simulators.

Build Type	Status
Cortex-M	Cortex-M passing
Hexagon	Run-Hexagon passing
RISC-V	RISC-V passing
Xtensa	Run-Xtensa passing
Generate Integration Test	Generate Integration Tests passing

Contributing See our *contribution documentation*.

Getting Help A Github issue should be the primary method of getting in touch with the TensorFlow Lite Micro (TFLM) team.

The following resources may also be useful:

1. SIG Micro email group and monthly meetings.
2. SIG Micro gitter chat room.
3. For questions that are not specific to TFLM, please consult the broader TensorFlow project, e.g.:
   - Create a topic on the TensorFlow Discourse forum
   - Send an email to the TensorFlow Lite mailing list
   - Create a TensorFlow issue
   - Create a Model Optimization Toolkit issue

Additional Documentation

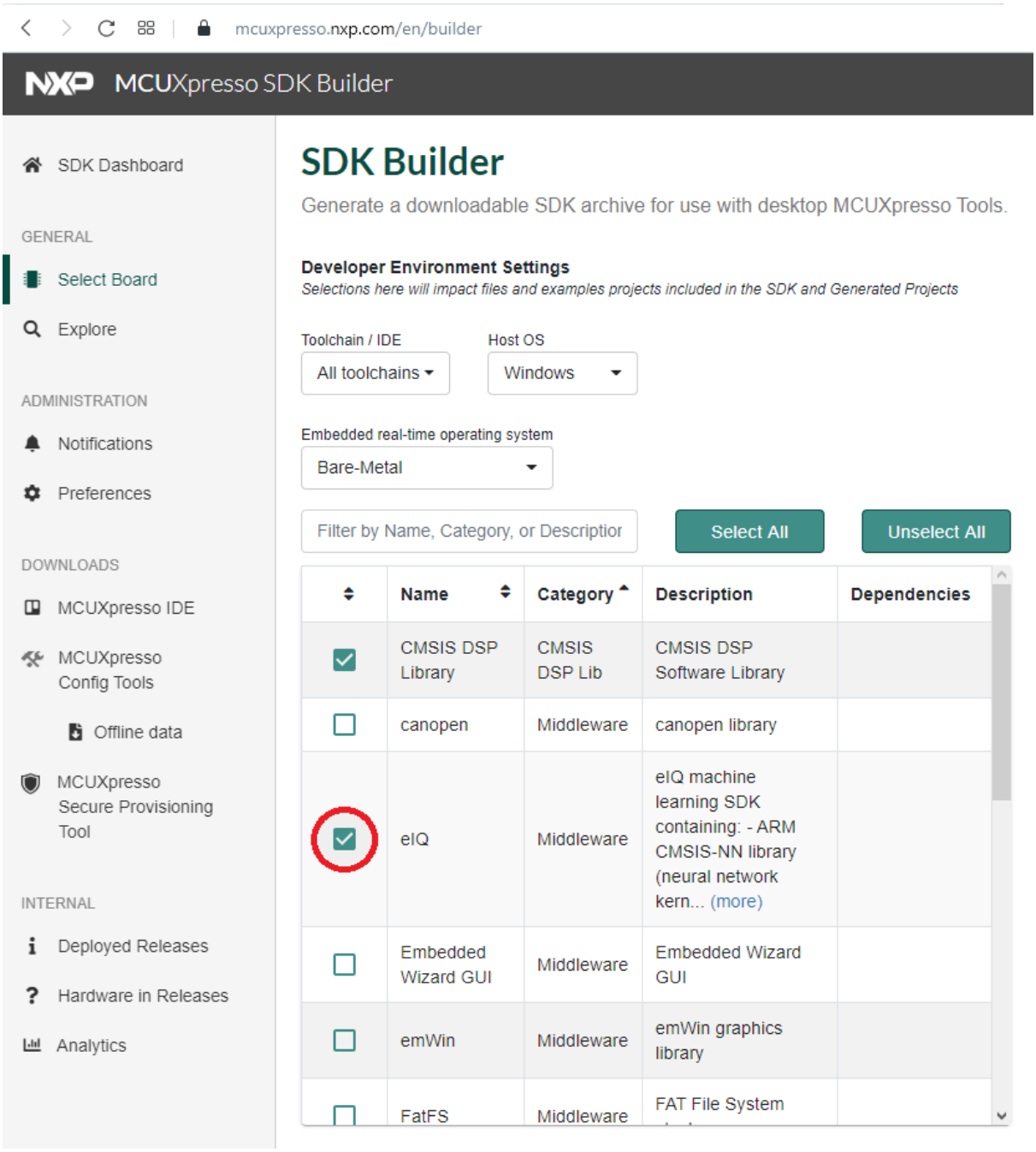
- *Continuous Integration*

- *Benchmarks*
- *Profiling*
- *Memory Management*
- *Logging*
- *Porting Reference Kernels from TfLite to TFLM*
- *Optimized Kernel Implementations*
- *New Platform Support*
- Platform/IP support
  - *Arm IP support*
- *Software Emulation with Renode*
- *Software Emulation with QEMU*
- *Python Dev Guide*
- *Automatically Generated Files*
- *Python Interpreter Guide*

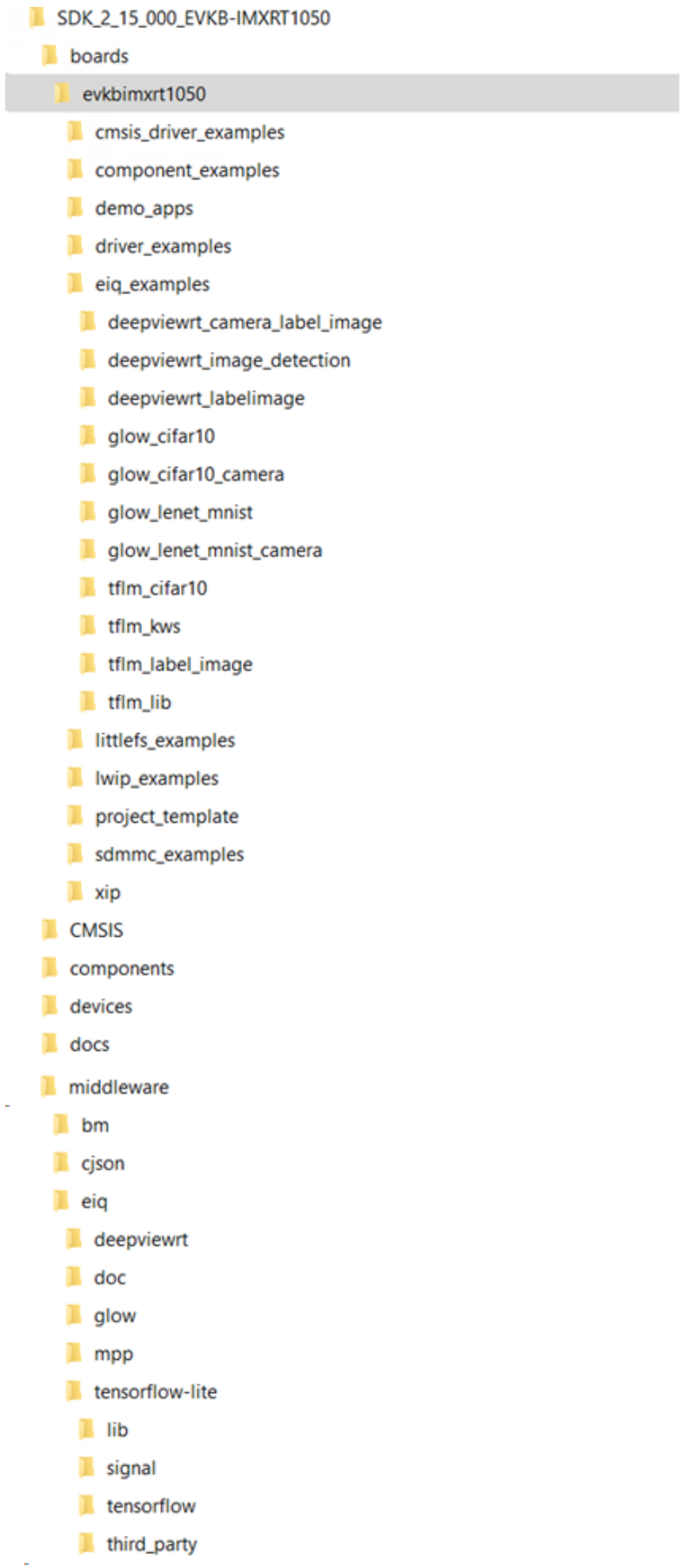
RFCs

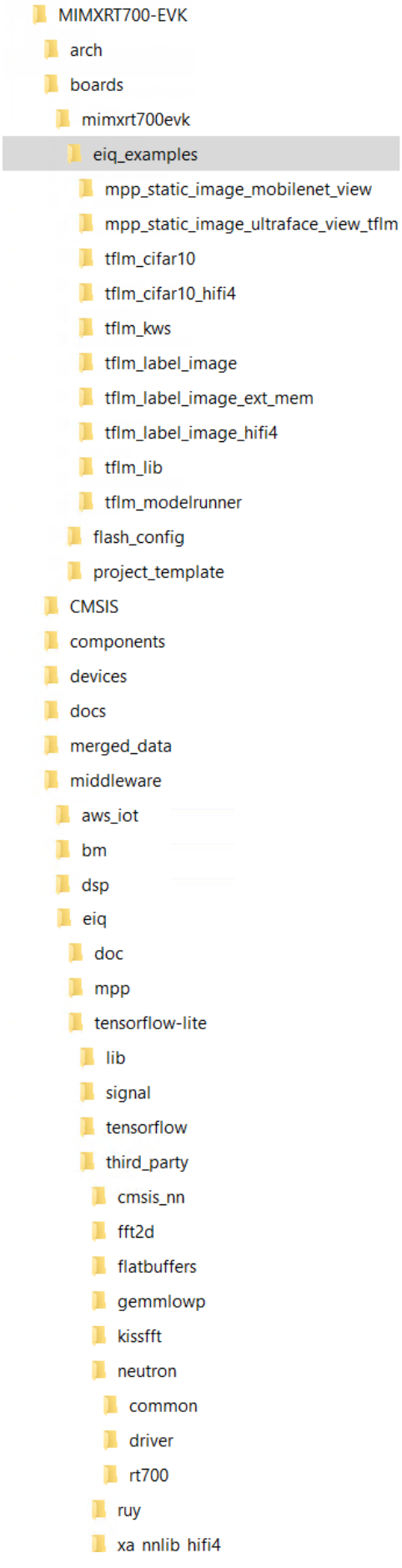
1. *Pre-allocated tensors*
2. *TensorFlow Lite for Microcontrollers Port of 16x8 Quantized Operators*

Deployment The eIQ TensorFlow Lite for Microcontrollers library is part of the eIQ machine learning software package, which is an optional middleware component of MCUXpresso SDK. The eIQ component is integrated into the MCUXpresso SDK Builder delivery system available on mcuxpresso.nxp.com. To include eIQ machine learning into the MCUXpresso SDK package, the eIQ middleware component is selected in the software component selector on the SDK Builder page when building a new package. See *Figure 1*.



Once the MCUXpresso SDK package is downloaded, it can be extracted on a local machine or imported into the MCUXpresso IDE. For more information on the MCUXpresso SDK folder structure, see the Getting Started with MCUXpresso SDK User's Guide (document: MCUXSDKGSUG). The package directory structure is similar to *Figure 2*. The eIQ TensorFlow Lite library directories are highlighted in red.


SDK_2_15_000_EVKB-IMXRT1050
boards
evkbimxrt1050
cmsis_driver_examples
component_examples
demo_apps
driver_examples
eiq_examples
deepviewrt_camera_label_image
deepviewrt_image_detection
deepviewrt_labelimage
glow_cifar10
glow_cifar10_camera
glow_lenet_mnist
glow_lenet_mnist_camera
tflm_cifar10
tflm_kws
tflm_label_image
tflm_lib
littlefs_examples
lwip_examples
project_template
sdmmc_examples
xip
CMSIS
components
devices
docs
middleware
bm
cjson
eiq
deepviewrt
doc
glow
mpp
tensorflow-lite
lib
signal
tensorflow
third_party


MIMXRT700-EVK
arch
boards
mimxrt700evk
eiq_examples
mpp_static_image_mobilenet_view
mpp_static_image_ultraface_view_tflm
tflm_cifar10
tflm_cifar10_hifi4
tflm_kws
tflm_label_image
tflm_label_image_ext_mem
tflm_label_image_hifi4
tflm_lib
tflm_modelrunner
flash_config
project_template
CMSIS
components
devices
docs
merged_data
middleware
aws_iot
bm
dsp
eiq
doc
mpp
tensorflow-lite
lib
signal
tensorflow
third_party
cmsis_nn
fft2d
flatbuffers
gemmlowp
kissfft
neutron
common
driver
rt700
ruy
xa_nnlib_hifi4

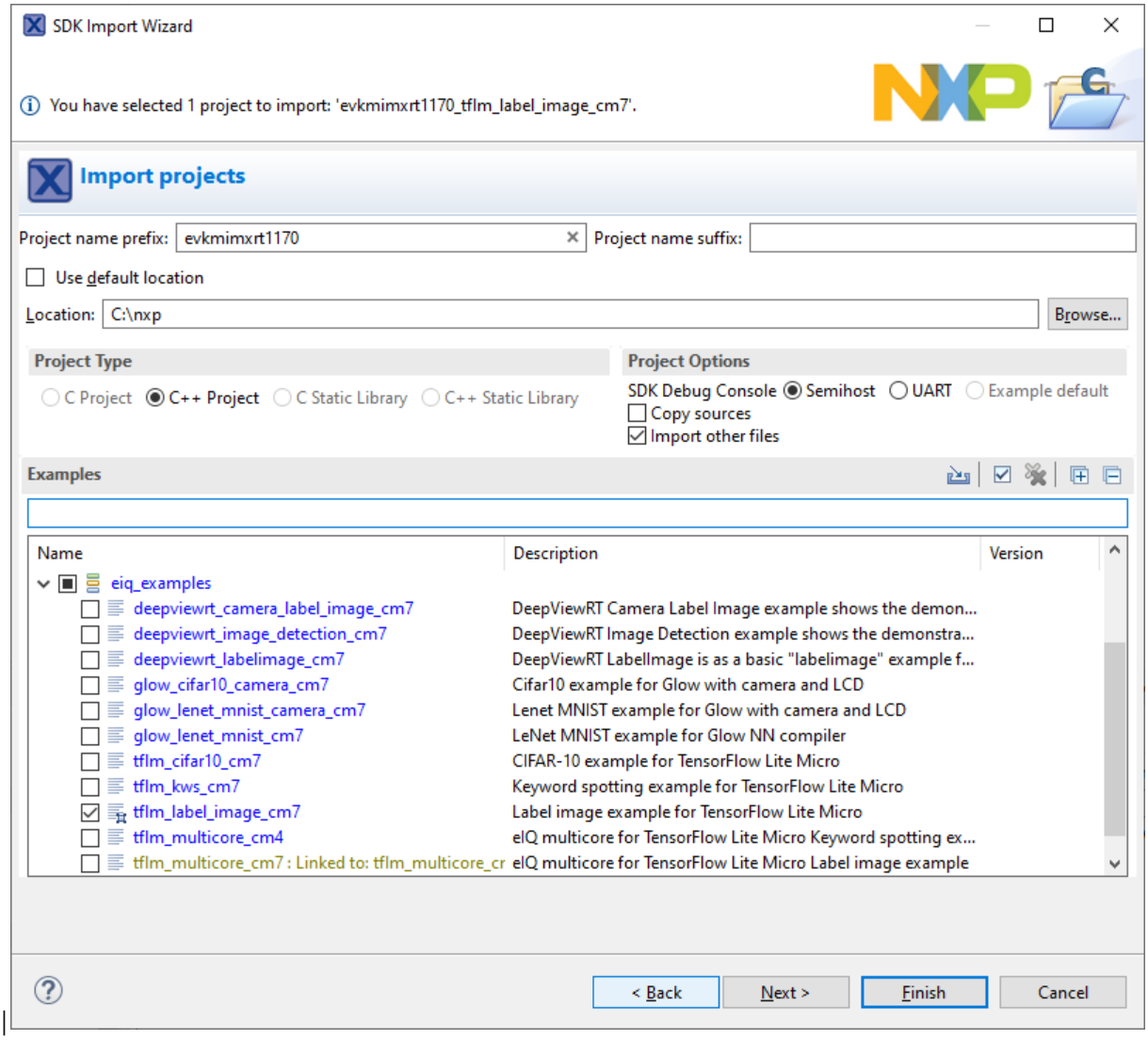
|

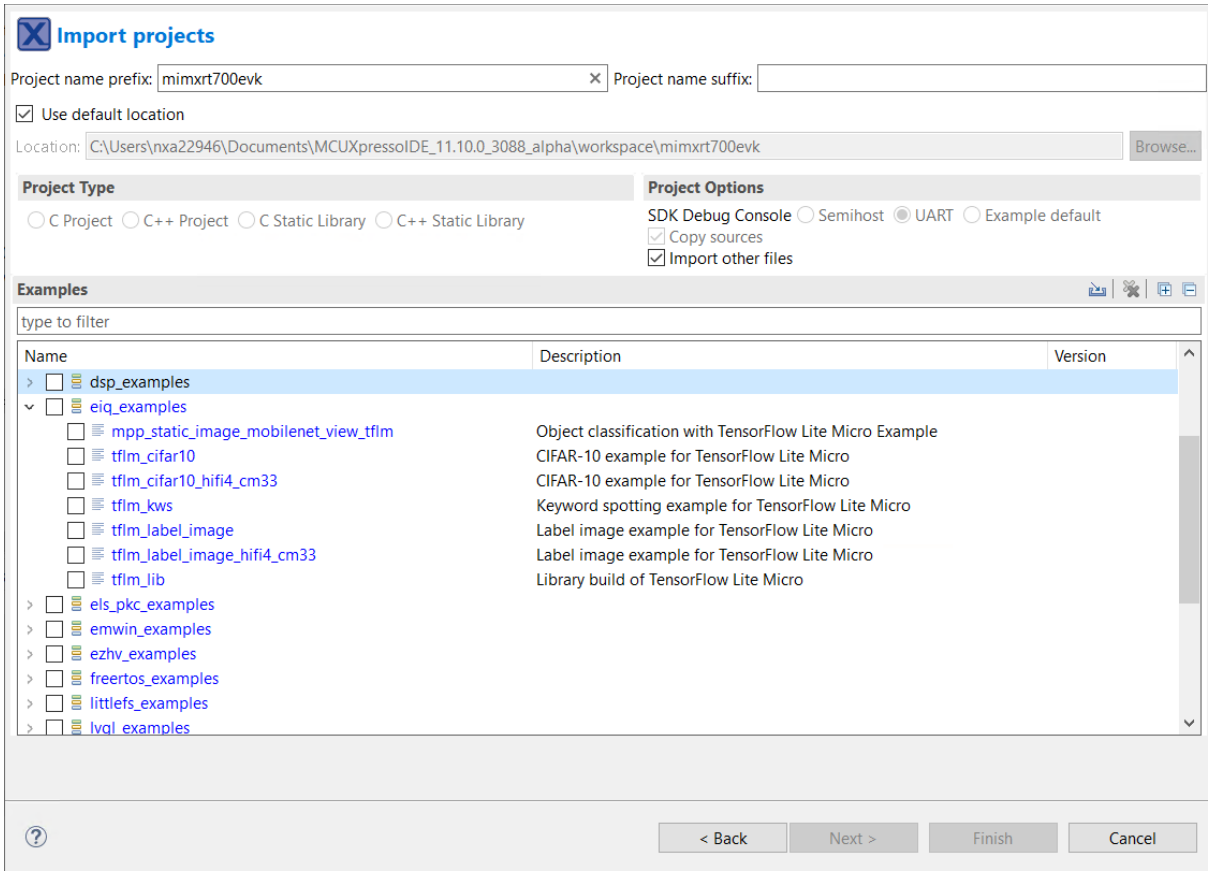
The *boards* directory contains example application projects for supported toolchains. For the list of supported toolchains, see the *MCUXpresso SDK Release Notes*. The *middleware* directory contains the eIQ library source code and example application source code and data.

Example applications The eIQ TensorFlow Lite library is provided with a set of example applications. For details, see *Table 1*. The applications demonstrate the usage of the library in several use cases.

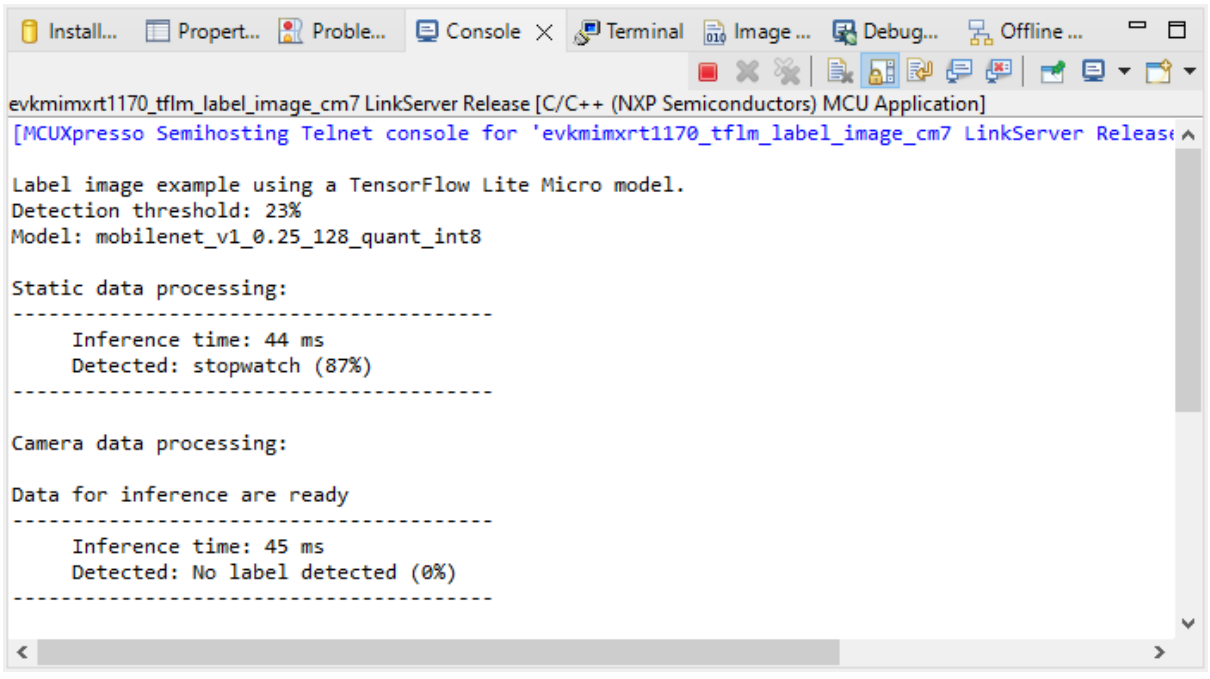
Name	Description	Availability
tflm_c	CIFAR-10 classification of 32 × 32 RGB pixel images into 10 categories using a small Convolutional Neural Network (CNN).	MCX-N947-EVK (no camera and display support) MCX-N947-FRDM (no camera and display support) MCX-N547-EVK (no camera and display support) MIMXRT700-EVK (no camera and display support)
tflm_l	Keyword spotting application using a neural network for word detection in pre-processed audio input.	MCX-N947-EVK (no audio support) MCX-N947-FRDM (no audio support) MCX-N547-EVK (no audio support) MIMXRT700-EVK (no audio support)
tflm_l	Image recognition application using a MobileNet model architecture to classify 128 × 128 RGB pixel images into 1000 categorieswith eIQ Neutron NPU.	MCX-N947-EVK (no camera and display support) MCX-N947-FRDM (no camera and display support) MCX-N547-EVK (no camera and display support) MIMXRT700-EVK (no camera and display support)
tflm_l	Image recognition application using a MobileNet model architecture to classify 224 × 224 RGB pixel images into 1000 categorieswith eIQ Neutron NPU. In this example, it demonstrates how to fetch model's weight from external memory(xSPI flash) to internal SRAM for Neutron NPU execution.	MIMXRT700-EVK (no camera and display support)
tflm_c	CIFAR-10 classification of 32 × 32 RGBpixel images into 10 categories using a small Convolutional Neural Network. In this example, M33 core0 starts HiFi4 DSP core with HiFi4 DSP image. HiFi4 DSP does the inference for CIFAR-10 classification.	MIMXRT700-EVK (no camera and display support)
tflm_l	Image recognition application using a MobileNet model architecture to classify 128 × 128 RGB pixel images into 1000 categories. In this example, M33 core0 starts HiFi4 DSP core with HiFi4 DSP image. HiFi4 DSP does the inference for image recognition application.	MIMXRT700-EVK (no camera and display support)

For details on how to build and run the example applications with supported toolchains, see *Getting Started with MCUXpresso SDK User's Guide* (document: MCUXSDKGSUG). When using MCUXpresso IDE, the example applications can be imported through the SDK Import Wizard as shown in *Figure 1*.


SDK Import Wizard
You have selected 1 project to import: 'evkmimxrt1170_tflm_label_image_cm7'.
Import projects
Project name prefix: evkmimxrt1170
Project name suffix:
Use default location
Location: C:\nxp
Browse...
Project Type
C Project
C++ Project
C Static Library
C++ Static Library
Project Options
SDK Debug Console
Semihost
UART
Example default
Copy sources
Import other files
Examples
Name
Description
Version
eiq_examples
deepviewrt_camera_label_image_cm7
DeepViewRT Camera Label Image example shows the demon...
deepviewrt_image_detection_cm7
DeepViewRT Image Detection example shows the demonstra...
deepviewrt_labelimage_cm7
DeepViewRT LabelImage is as a basic "labelimage" example f...
glow_cifar10_camera_cm7
Cifar10 example for Glow with camera and LCD
glow_lenet_mnist_camera_cm7
Lenet MNIST example for Glow with camera and LCD
glow_lenet_mnist_cm7
LeNet MNIST example for Glow NN compiler
tflm_cifar10_cm7
CIFAR-10 example for TensorFlow Lite Micro
tflm_kws_cm7
Keyword spotting example for TensorFlow Lite Micro
tflm_label_image_cm7
Label image example for TensorFlow Lite Micro
tflm_multicore_cm4
eIQ multicore for TensorFlow Lite Micro Keyword spotting ex...
tflm_multicore_cm7 : Linked to: tflm_multicore_cr
eIQ multicore for TensorFlow Lite Micro Label image example
< Back
Next >
Finish
Cancel



After building the example application and downloading it to the target, the execution stops in the *main* function. When the execution resumes, an output message displays on the connected terminal. For example, *Figure 2* shows the output of the tflm_label_image_cm7``tflm_label_image example application printed to the MCUXpresso IDE Console window when semihosting debug console is selected in the SDK Import Wizard.



```
Label image example using a TensorFlow Lite Micro model.
Detection threshold: 23%
Model: mobilenet_v1_0.25_128_quant_int8_npu

Static data processing:
---------------------------------------
     Inference time: 3987 us
     Detected: stopwatch (87%)
---------------------------------------
```

|

Model Conversion to TensorFlow Lite Format The eIQ® Toolkit provides a comprehensive end-to-end environment for machine learning (ML) model development and deployment. Designed for NXP EdgeVerse processors, the toolkit includes both an intuitive GUI-based tool (eIQ Portal) and command-line utilities for advanced workflows.

One key component, the eIQ ModelTool, enables seamless conversion of ML models from popular formats such as TensorFlow, PyTorch, and ONNX into the TensorFlow Lite (TFLite) format. These converted models can be further optimized and deployed on NXP platforms for inference acceleration.

Model Conversion for NXP eIQ Neutron NPU To leverage the NXP eIQ Neutron NPU for hardware acceleration, models must undergo additional processing using the Neutron Converter Tool. This tool transforms standard quantized TensorFlow Lite models into a format optimized for execution on the Neutron NPU.

The key steps involved in this process are as follows:

1. Convert to Quantized TensorFlow Lite Model: Ensure the model is in a quantized TFLite format before running the Neutron Converter.
2. Run the Neutron Converter Tool: The Neutron Converter analyzes the TFLite model, identifies supported operators, and replaces them with specialized NPU-compatible nodes. Unsupported operations are executed using fallback mechanisms, such as:

- CMSIS-NN for optimized CPU execution
- Reference Operators for unsupported cases

3. Execute on Target Platform: The converted model runs efficiently on the Neutron NPU using a custom TFLite Micro-operator implementation.

Example: Converting a Quantized TensorFlow Lite Model for Neutron NPU The following is a sample command-line invocation for the Neutron Converter tool:

```
neutron-converter --input mobilenet_v1_0.25_128_quant.tflite \
                  --output mobilenet_v1_0.25_128_quant_npu.tflite \
                  --target imxrt700 \
                  --dump-header-file-output
```

Note: This will convert the source tflite model to neutron compatable model, meanwhile, it will dump the model as one headfile name as “mobilenet_v1_0.25_128_quant_npu.h”.

Run and debug eIQ HiFi4 and HiFi1 DSP examples using Xplorer IDE This section lists the steps to Prepare CM33 Core for the examples and Prepare DSP core for the examples.

Prepare CM33 Core for the examples

1. The tflm_cifar10_hifi4 and tflm_label_image_hifi4 examples consist of two separate applications that run on the CM33 core0 and DSP core. The CM33 core0 application initializes the DSP core and starts it.

   To debug the application:

   1. Set up and execute the CM33 application using an environment of your choice.
   2. Build and execute the examples located in:

      <SDK_ROOT>/boards/mimxrt700evk/eiq_examples/tflm_cifar10_hifi4/cm33/

      <SDK_ROOT>/ boards/mimxrt700evk/eiq_examples/tflm_label_image_hifi4/cm33/

2. The tflm_cifar10_hifi1 example consists of three separate applications that run on the CM33 core0, CM33 core1, and DSP core. The CM33 core0 application initializes the CM33 core1 core and starts it. The CM33 core1 application initializes the DSP core and starts it.

   To debug the application:

   1. Set up and build the CM33 core1 application using an environment of your choice.
   2. Set up and execute the CM33 core0 application using an environment of your choice.
   3. Build and execute the example located in:

      <SDK_ROOT>/boards/mimxrt700evk/eiq_examples/tflm_cifar10_hifi1/cm33_core1/

      <SDK_ROOT>/boards/mimxrt700evk/eiq_examples/tflm_cifar10_hifi1/cm33_core0/

      Note: ARMGCC toolchain and IAR Embedded Workbench are both supported. To enable compatibility with RT700, IAR Embedded Workbench may require a patch. There are default DSP core images in the SDK. For details on how to build the examples, refer to Prepare DSP core for the examples.

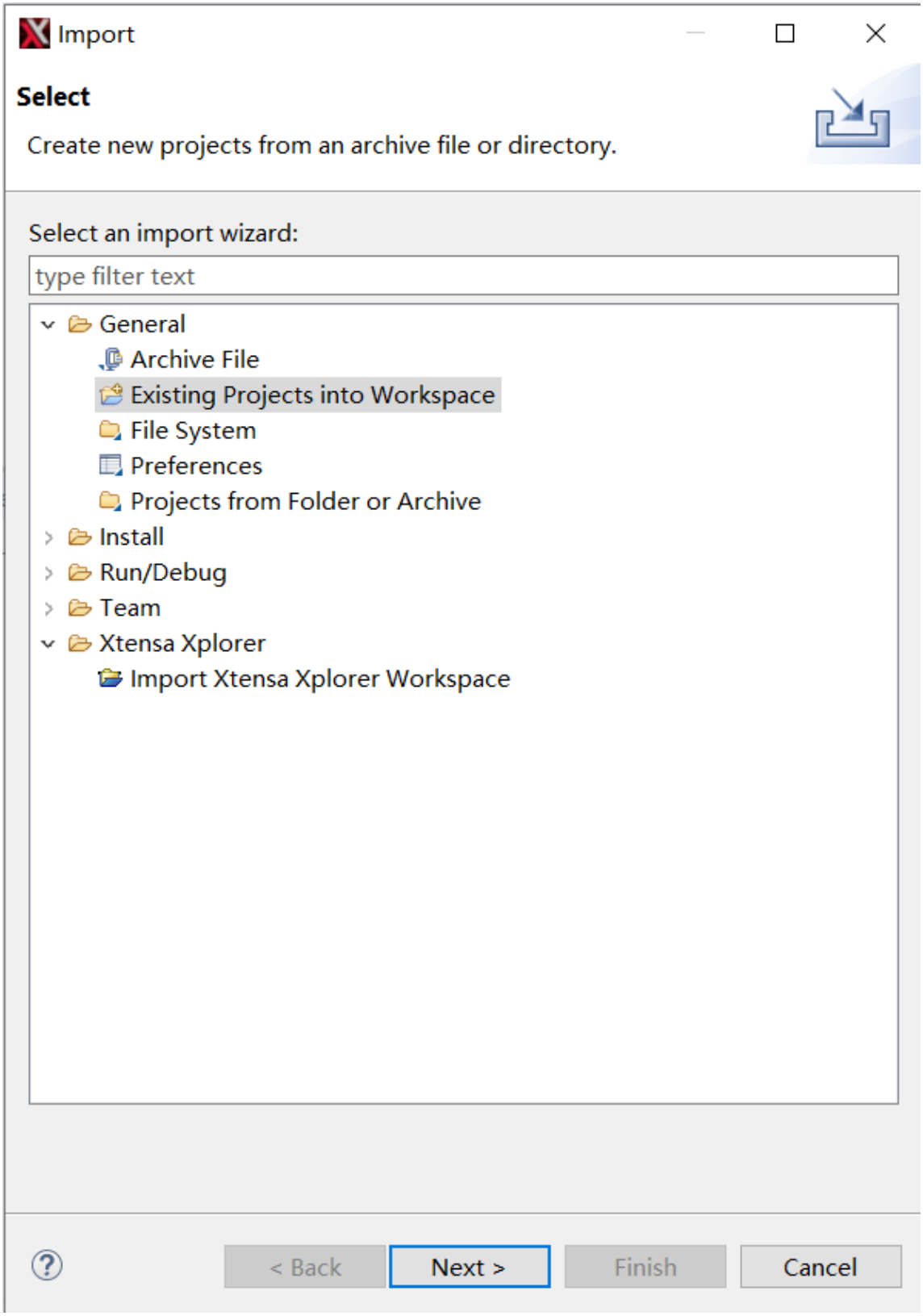
Parent topic:*Run and debug eIQ HiFi4 and HiFi1 DSP examples using Xplorer IDE*

Prepare DSP core for the examples

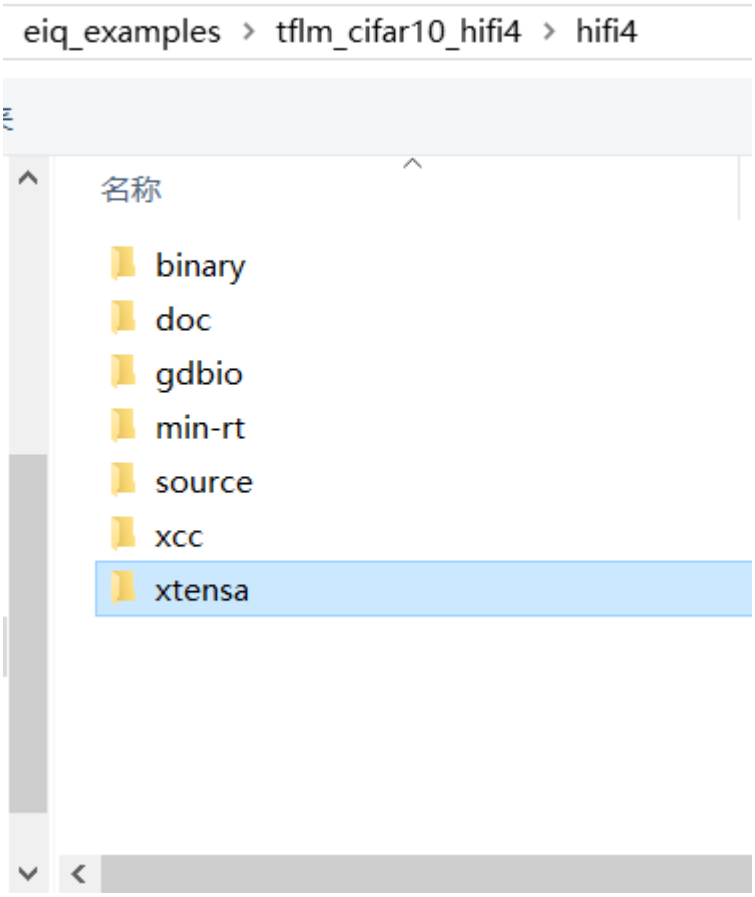
The projects for different supported toolchains are built. The “xcc” project builds on the command line and the “xtensa” directory is an Xplorer IDE project.

To run the tflm_cifar10_hifi4 example, import the SDK sources into the Xplorer IDE.

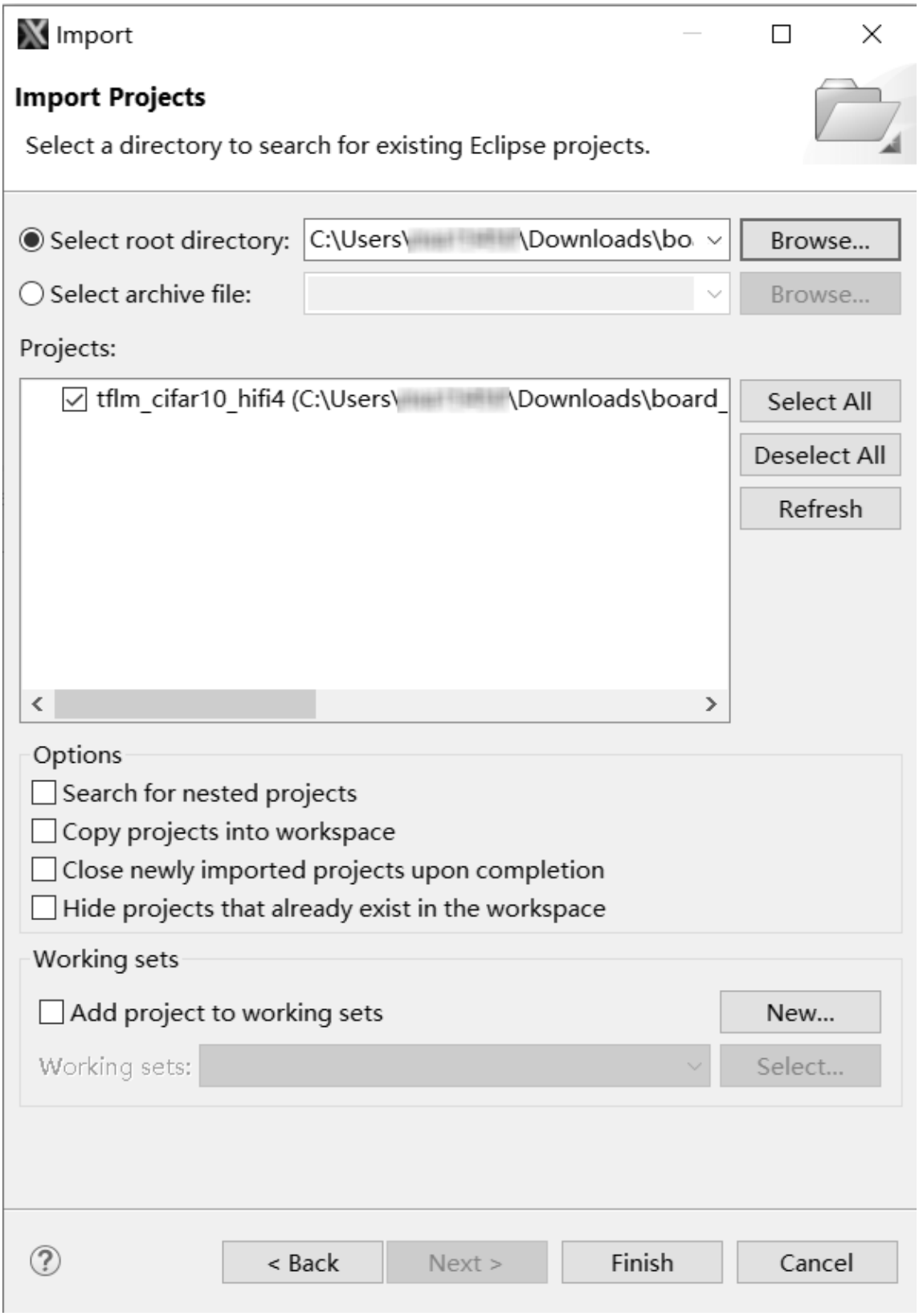
1. Select **File** > **Import** > **General** > **Existing Projects into Workspace**.



2. Click **Next**.
3. Select the SDK directory/boards/mimxrt700evk/eiq_examples/tflm_cifar10_hifi4/hifi4/xtensa as the root directory.

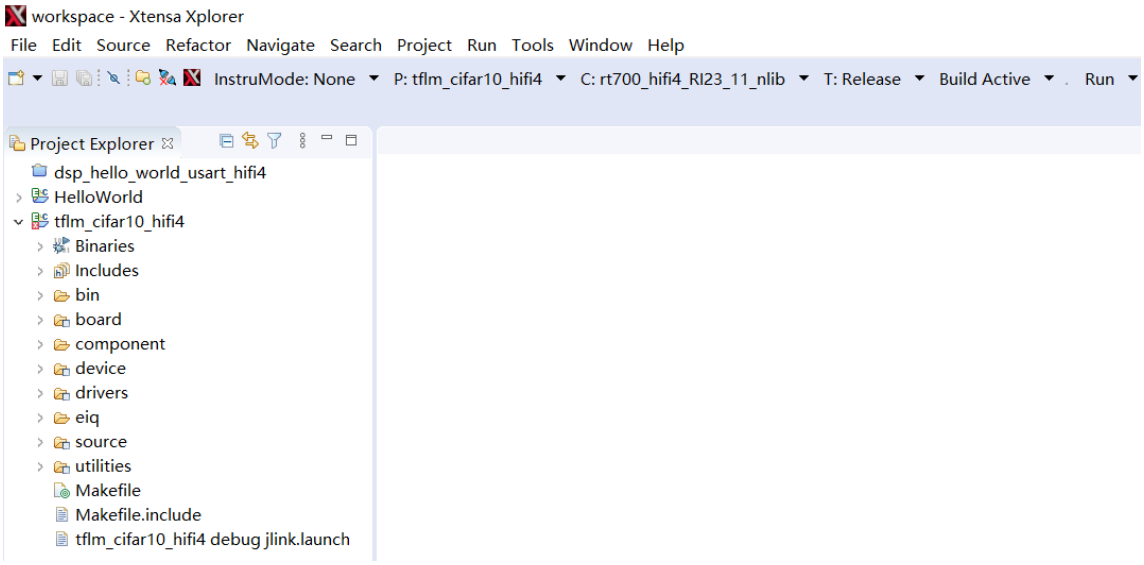


4. Click **Select Folder**.
5. Leave all the other options check boxes blank.

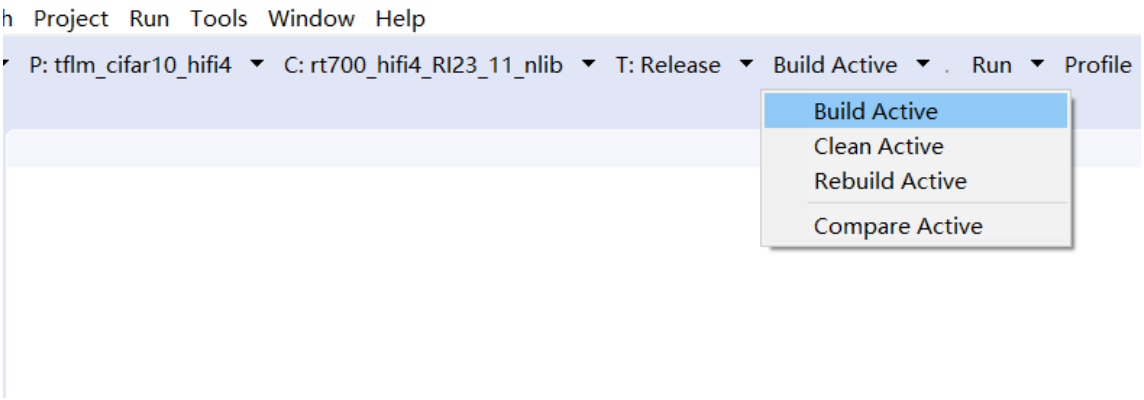


Once imported, the tflm_cifar10_hifi4 example appears in the **Project Explorer**.

6. To make a build selection for the project and hardware target configuration, use the drop-down buttons on the menu bar.



7. To build the DSP application image for the CM33 application, select the **Release target** option in the Xplorer IDE as below.



8. Three DSP binaries are generated and are loaded into different TCM or SRAM address segments:
   - <SDK_ROOT/>/boards/mimxrt700evk/eiq_examples/tflm_cifar10_hifi4/hifi4/binary/dsp_data_release.bin
   - <SDK_ROOT/>/boards/mimxrt700evk/eiq_examples/tflm_cifar10_hifi4/hifi4/binary/dsp_literal_release.bin
   - <SDK_ROOT/>/boards/mimxrt700evk/eiq_examples/tflm_cifar10_hifi4/hifi4/binary/dsp_text_release.bin

Parent topic:*Run and debug eIQ HiFi4 and HiFi1 DSP examples using Xplorer IDE*

Running an inference After converting the model to the TensorFlow Lite format, it is converted into a C language array to include it in the application source code. The *xxd* utility can be used for this purpose (distributed with the *Vim* editor for many platforms on https://www.vim.org/) as shown in *Converting a model to a C language header file*. The utility converts a TensorFlow Lite model into a C header file with an array definition containing the binary image of the model and a variable containing the data size.

Converting a model to a C language header file {#EXAMPLE_4 .section}

```
xxd -i mobilenet_v1_0.25_128_quant.tflite > mobilenet_v1_0.25_128_quant_model.h
```

After the header file is generated, the type of the array is changed from unsigned char to const char to match the library API input parameters and the default array name can be changed to a more convenient one. The user must align the buffer to at least 64-bit boundary (the size of a double-precision floating-point number) to avoid misaligned memory access. The alignment can be achieved by using the __ALIGNED(16) macro from the cmsis_compiler.h header file (available in the MCUXpresso SDK) in the array declaration before the data assignment.

The easiest way to create an application with the proper configuration is to copy and modify an existing example application. To learn where to find the example applications and how to build them, see the *Example applications*.

Running an inference using TensorFlow Lite for Microcontrollers involves several steps (shown for quantized model with signed 8-bit values as input and 32-floating point values as output):

1. Include the necessary eIQ TensorFlow Lite Micro library header files and the converted model.

   Including header files

   ```
   #include "tensorflow/lite/micro/micro_error_reporter.h"
   #include "tensorflow/lite/micro/micro_interpreter.h"
   #include "tensorflow/lite/micro/all_ops_resolver.h"
   #include "mobilenet_v1_0.25_128_quant_model.h"
   ```

2. Allocate a static memory buffer for input and output tensors and intermediate arrays. Load the FlatBuffer model image (assuming the mobilenet_v1_0.25_128_quant_model.h file generated in *Converting a model to a C language header file* defines an array named mobilenet_model and a size variable named mobilenet_model_len), build the interpreter object and allocate memory for tensors.

   Loading the FlatBuffer model

   ```
   constexpr int kTensorArenaSize = 1024 * 1024;
   static uint8_t tensorArena[kTensorArenaSize];
   const tflite::Model* model = tflite::GetModel(mobilenet_model);
   // TODO: Report an error if model->version() != TFLITE_SCHEMA_VERSION
   static tflite::AllOpsResolver microOpResolver;
   static tflite::MicroErrorReporter microErrorReporter;
   static tflite::MicroInterpreter interpreter(model,
     microOpResolver, tensorArena, kTensorArenaSize,
     microErrorReporter);
   interpreter->AllocateTensors();
   // TODO: Check return value for kTfLiteOk
   ```

3. Fillhe input data into the input tensor. For example, if a speech recognition model, image data from a camera or audio data from a microphone. The dimensions of the input data must be the same as the dimensions of the input tensor. These dimensions were specified when the model was created.

   Fill-in input data

   ```
   // Get access to the input tensor data
   TfLiteTensor* inputTensor = interpreter->input(0);
   // Copy the input tensor data from an application buffer
   for (int i = 0; i < inputTensor->bytes; i++)
     inputTensor->data.int8[i] = input_data[i];
   ```

4. Run the inference and read the output data from the output tensor. The dimensions of the output data must be the same as the dimensions of the output tensor. These dimensions were specified when the model was created.

   Running inference and reading output data

```
// Run the inference
interpreter->Invoke();
// TODO: Check the return value for TfLiteOk
// Get access to the output tensor data
TfLiteTensor* outputTensor = interpreter->output(0);
// Copy the output tensor data to an application buffer
for (int i = 0; i < outputTensor->bytes / sizeof(float32); i++)
  output_data[i] = outputTensor->data.f[i];
```

NPU inference {#npu_infer .section} Running an inference using a model converted for the NPU requires registration of a custom operator implementation. First the header file with the custom operator implementation interface must be included.

```
#include "tensorflow/lite/micro/kernels/micro_ops.h"
#include "tensorflow/lite/micro/all_ops_resolver.h"
#include "tensorflow/lite/micro/kernels/neutron/neutron.h"
```

Next, the specialized implemetation has to be registered in the operator resolver object.

```
static tflite::AllOpsResolver microOpResolver;
microOpResolver.AddCustom(tflite::GetString_NEUTRON_GRAPH(),
      tflite::Register_NEUTRON_GRAPH());
```

The specialized NPU nodes from the converted model are the executed using this newly registered implementation.

Adjusting the tensor arena size {#adjust_arena .section} The tensor arena is a static memory buffer used for intermediate tensor and scratch buffer allocation. The size of the tensor arena buffer is set by the kTensorArenaSize constant in the example above. The value depends on the tensor sizes used in the model and on the hardware-specific implementations of kernels, which may require various sizes of scratch buffers for intermediate computations. The value can be determined experimentally by running an inference with a small value, so the library fails with an insufficient tensor memory error and prints the missing amount. Continue adjusting the size until the error stops being reported. If the target hardware changes, readjust the value.

Code size optimization Typically, models do not use all the operators that are available in TensorFlow Lite. However, because of the default operator registration mechanism used in the library, the toolchain linker is not able to remove the code of unused operators. In order to reduce code size, it is possible to only register the specific operators used by a model. To determine which operators are used by a particular model, a model visualizer tool like Netron can be used. Then a mutable operator resolver object can be created that only registers the operators that are used by the model being inferenced.

Use the tflite::MicroMutableOpResolver object template, which is later passed to the tflite::MicroInterpreter object. Depending on the list of used operators, the result should be similar to the following code snippet. Make sure to update the MicroMutableOpResolver template parameter to reflect the number of operators that need to be registered.

Register only used operators in TensorFlow Lite Micro {#SECTION_SS1_DJQ_QPB .section}

```
#include "tensorflow/lite/micro/kernels/micro_ops.h"
      #include "tensorflow/lite/micro/micro_mutable_op_resolver.h"
      tflite::MicroMutableOpResolver<6> microOpResolver;
      microOpResolver.AddAveragePool2D();
      microOpResolver.AddConv2D();
      microOpResolver.AddDepthwiseConv2D();
```

(continues on next page)

(continued from previous page)

```
microOpResolver.AddDequantize();
microOpResolver.AddReshape();
microOpResolver.AddSoftmax();
static tflite::MicroInterpreter interpreter(
model, microOpResolver, tensorArena, kTensorArenaSize, microErrorReporter);
```

Note about the source code in the document Example code shown in this document has the following copyright and BSD-3-Clause license:

Copyright 2024 NXP Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

eIQ ExecuTorch Library User Guide

Overview ExecuTorch is an end-to-end solution for enabling on-device inference capabilities across mobile and edge devices including wearables, embedded devices and microcontrollers. It is part of the PyTorch Edge ecosystem and enables efficient deployment of PyTorch models to edge devices. For more information, see https://pytorch.org/executorch-overview.

The MCUXpresso Software Development Kit (MCUXpresso SDK) provides a comprehensive software package with a pre-integrated ExecuTorch based on version v0.5.0 with initial support for Neutron Backend. Neutron Backend enables acceleration of ML models on the eIQ® Neutron Neural Processing Unit (NPU).

This document describes the steps required to download and start using the ExecuTorch. Additionally, the document describes the steps required to create an application for running pretrained models.

Note: The document also assumes knowledge of machine learning frameworks for model training.

Supported platforms:

- i.MX RT700

Installation The ExecuTorch, with the Neutron Backend consists of:

- ExecuTorch with Neutron Backend for Ahead of Time ML Model Compilation
- Neutron Converter
- MCUXpresso SDK

Here we briefly describe each components purpose and steps to install them.

The **ExecuTorch AoT** and **Neutron Converter** are needed to convert a PyTorch model to ExecuTorch and Delegate it to eIQ Neutron NPU using the Neutron Backend. The **MCUXpresso SDK** provides project to build the ExecuTorch Runtime Library, the example application with simple CNN, toolchains and other middleware libraries to build and deploy the application on the target platform.

If you want run to prepared example application on the i.MX RT700 platform, and skip the model preparation phase continue with the *MCUXpresso SDK Part*.

ExecuTorch for Ahead of Time model preparation The ExecuTorch enables to deploy PyTorch models on edge devices. For this purpose the PyTorch model must be processed and converter by the ExecuTorch Ahead of Time (AoT) part. You can obtain the full ExecuTorch including the AoT part aligned with this version of MCUX SDK from the mcuxsdk-middleware-executorch release/mcux-full branch.

Installation Prerequisities:

- x86 Linux Machine with GLIBC-2.29 or higher (e.g. Ubuntu 20.04 or higher)
- Python 3.10, 3.11 or 3.12

To build and install the ExecuTorch follow these steps:

1. (Optional) Setup python virtual environment on desired location and activate it.

```
$ python3 -m venv venv
$ source venv/bin/activate
```

2. Clone the ExecuTorch from mcuxsdk-middleware-executorch

```
$ git clone --branch release/mcux-full https://github.com/nxp-mcuxpresso/mcuxsdk-middleware-executorch.git
$ cd mcuxsdk-middleware-executorch
$ git submodule update --init --recursive
```

3. Build and install the ExecuTorch and its dependencies:

```
$ ./install_requirements.sh
```

[!WARNING] The install_requirements.sh installs the CPU version of torch from https://download.pytorch.org/whl/cpu. If you are behind corporate proxy, it might have issues accessing it and you will see warnings like:

```
WARNING: Retrying (Retry(total=4, connect=None, read=None, redirect=None,␣
↪status=None)) after connection broken by 'SSLError(SSLCertVerificationError(1, '[SSL:␣
↪CERTIFICATE_VERIFY_FAILED] certificate verify failed: unable to get local issuer␣
↪certificate (_ssl.c:1006)'))': /whl/test/cpu/torch/
```

In this case the CUDA version of torch is installed and the install_requirements.sh script fails with:

```
PyTorch: CUDA cannot be found.  Depending on whether you are building
```

Make sure the pip can access the https://download.pytorch.org/whl/cpu PyPI.

Next continue with installation of the *Neutron Converter*

Neutron Converter The eIQ Neutron Backend uses the Neutron Converter to convert the ExecuTorch program to the eIQ Neutron NPU microcode.

Installation The Neutron Converter is available as a Python package and can be installed by the pip command from eiq.nxp.com/repository:

```
pip install --extra-index-url eiq.nxp.com/repository neutron_converter_SDK_25_06==1.0.0
```

The Neutron Converter is used internally by the ExecuTorch, and it is tight to the particular BSP you are using - the suffix of the python package name. In the code snippet above the flavor is the SDK_25_06. In the *aot_neutron_convert.py* example script by the --neutron_converter_flavor parameter.

MCUXpresso SDK The MCUXpresso SDK is used to build, debug and deploy the application using the ExecuTorch on the target platform.

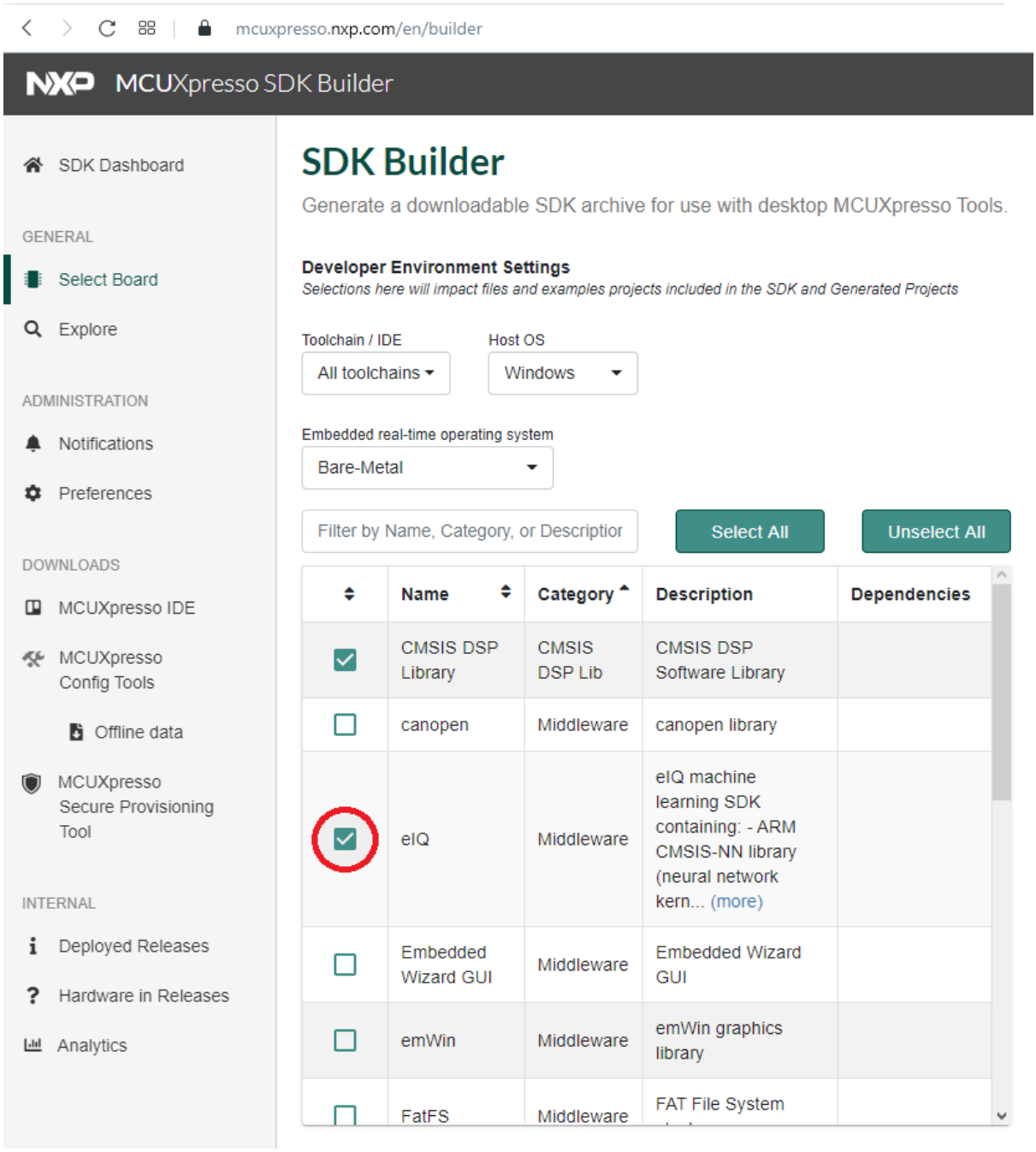
You can obtain the MCUXpresso SDK from MCUXpresso SDK Builder including the IDE. See the *getting_mcuxpress* for details.

In the MCUXpresso SDK, there are 2 project available related to ExecuTorch:

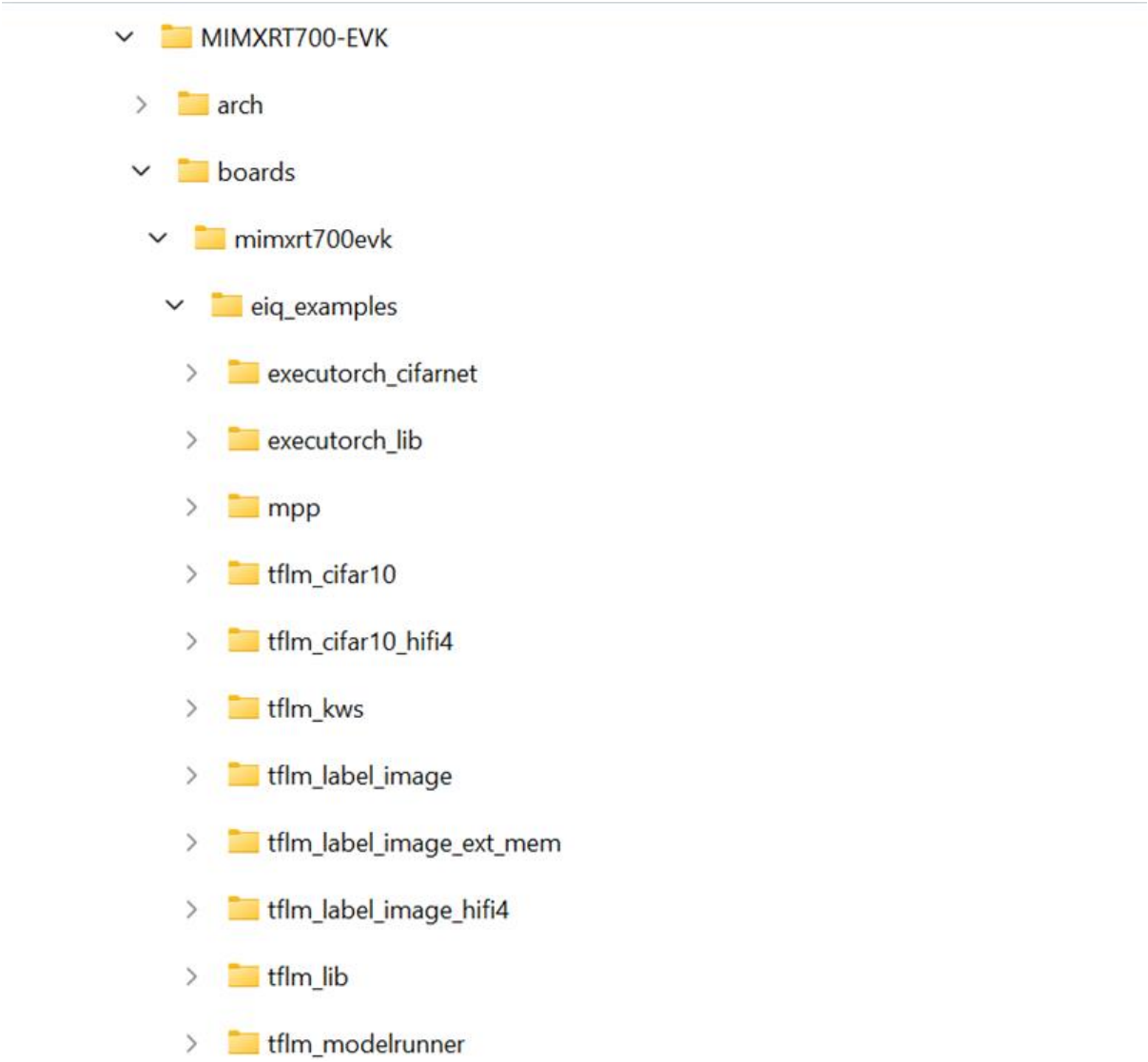
- executorch_lib
- executorch_cifarnet

For more details see *example_applications*. Here you will find the details to run build and run the demo applications.

Getting the MCUXpresso SDK with eIQ ExecuTorch The eIQ ExecuTorch library is part of the eIQ machine learning software package, which is an optional middleware component of MCUXpresso SDK. The eIQ component is integrated into the MCUXpresso SDK Builder delivery system available on mcuxpresso.nxp.com. To include eIQ machine learning into the MCUXpresso SDK package, the eIQ middleware component is selected in the software component selector on the SDK Builder page when building a new package. See *Figure 1*.



Once the MCUXpresso SDK package is downloaded, it can be extracted on a local machine or imported into the MCUXpresso IDE. For more information on the MCUXpresso SDK folder structure, see the Getting Started with MCUXpresso SDK User's Guide (document: MCUXSDKGSUG). The package directory structure is similar to *Figure 2*.



The *boards* directory contains example application projects for supported toolchains. For the list of supported toolchains, see the *MCUXpresso SDK Release Notes*. The *middleware* directory contains the eIQ library source code and example application source code and data.

PyTorch Model Conversion to ExecuTorch Format In this guideline we will show how to use the ExecuTorch AoT part to convert a PyTorch model to ExecuTorch format and delegate the model computation to eIQ Neutron NPU using the eIQ Neutron Backend.

First we will start with an example script converting the model. This example show the CifarNet model preparation. It is the same model which is part of the example_cifarnet

The steps are expected to be executed from the executorch root folder, in our case the mcuxsdk-middleware-executorch

1. After building the ExecuTorch you shall have the libquantized_ops_aot_lib.so located in the pip-out folder. We will need this library when generating the quantized cifarnet ExecuTorch model. So as first step we will find it:

```
$ find ./pip-out -name 'libquantized_ops_aot_lib.so'
./pip-out/temp.linux-x86_64-cpython-310/cmake-out/kernels/quantized/libquantized_ops_aot_lib.so
./pip-out/lib.linux-x86_64-cpython-310/executorch/kernels/quantized/libquantized_ops_aot_lib.so
```

2. Now run the aot_neutron_compile.py example with the cifar10 model

```
$ python examples/nxp/aot_neutron_compile.py \
  --quantize --so_library ./pip-out/lib.linux-x86_64-cpython-310/executorch/kernels/quantized/libquantized_
↪ops_aot_lib.so \
  --delegate --neutron_converter_flavor SDK_25_06 -m cifar10
```

3. It will generate you cifar10_nxp_delegate.pte file which can be used with the MXUXpresso SDK cifarnet_example project.

The generated PTE file is used in the executorch_cifarnet example application, see *example_application*.

MCUXpresso SDK Example applications The MCUXpresso SDK provides a set of projects and example application with the eIQ ExecuTorch. For details, see *Table 1*.

The eIQ ExecuTorch library is provided with a set of example applications. For details, see *Table 1*. The applications demonstrate the usage of the library in several use cases.

Name	Description	Availability
executorch_	This project contains the ExecuTorch Runtime Library source code and is used to build the ExecuTorch Runtime Library. The library is further used to build a full application using the leveraging ExecuTorch.	MIMXRT700-EVK (no camera and display support)
executorch_	Example application demonstrating the use of the ExecuTorch running a CifarNet classification model accelerated on the eIQ Neutron NPU. The Cifarnet is a small Convolutional Neural Network (CNN), trained on CIFAR-10 [1] dataset. The model clasifies the input images into 10 caterories.	MIMXRT700-EVK (no camera and display support)

For details on how to build and run the example applications with supported toolchains, see *Getting Started with MCUXpresso SDK User's Guide* (document: MCUXSDKGSUG).

How to build and run executorch_cifarnet example The example needs ExecuTorch Runtime Library and Neutron Libraries.

ExecuTorch Runtime Library:

- middleware/eiq/executorch/lib/cm33/armgcc/libexecutorch.a

Neutron Libraries:

- middleware/eiq/executorch/third-party/neutron/rt700/libNeutronDriver.a and
- middleware/eiq/executorch/third-party/neutron/rt700/libNeutronFirmware.a

In the example the model and the input image is already embedded into the program and ready to build and deploy to i.MX RT700, so you can continue right to the *building and deployment* section.

Convert the model and example input to C array In this section we describe where the model and example input is located in the example application sources, and how it was generated.

The **cifar10 model** ExecuTorch model is stored in boards/mimxrt700evk/eiq_examples/executorch_cifarnet/cm33_core0/model_pte.h. and was generated from the cifar10_nxp_delegate.pte (see *convert_model*).

We use the xxd command to get the C array containing the model data and array size:

```
$ xxd -i cifar10_nxp_delegate.pte > model_pte_data.h
```

then use the array data and size in the model_pte.h.

As **input image** we use the image from CIFAR-10 dataset [1]. After preprocessing and normalization it is converted to bytes and located here boards/mimxrt700evk/eiq_examples/executorch_cifarnet/cm33_core0/image_data.h. The preprocessing is performed as follows:

```
import torch
import torchvision
import numpy as np

batch_size = 1

transform = torchvision.transforms.Compose([
  torchvision.transforms.ToTensor(),
  torchvision.transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

test_set = torchvision.datasets.CIFAR10(root='./data', train=False, download=True, transform=transform)
test_loader = torch.utils.data.DataLoader(test_set, batch_size=batch_size, shuffle=False, num_workers=0)

index = 0
num_images = 10
for data in test_loader:
  images, labels = data
  for image, label in zip(images,labels):
    arr = image.numpy().astype(np.float32)
    arr.tofile("img" + str(index) + "_" + str(int(label)) + ".bin")
    index = index + 1
    if index >= num_images:
      break
  if index >= num_images:
    break
```

This generates the num_images count of images from Cifar10 dataset, as input tensors for the cifar10 model and store them in corresponding .bin files. Then we use the xxd command to get the C array data and size:

```
$ xxd -i img0_3.bin > image_data_base.h
```

and again copy the array data and size in the image_data.h

Note, the img0 is the image picturing a cat, what is a class number 3.

Build, Deploy and Run

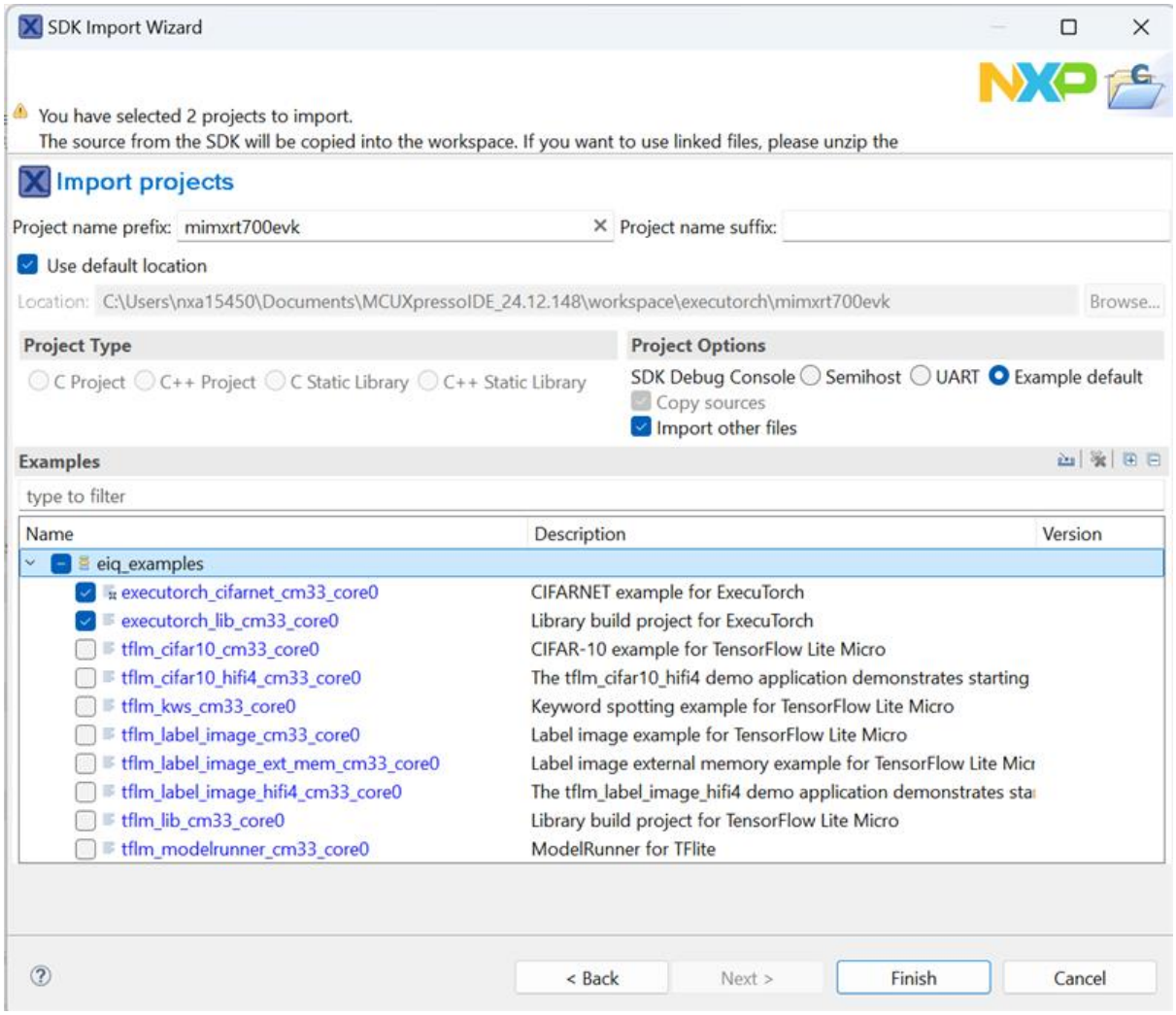
1. When using ARMGCC toolchain, the example application can be built as below. After building the example application, download it to the target with JLink as shown in *Figure 3*, an output message displays on the connected terminal as *Figure 4*.

```
$ boards/mimxrt700evk/eiq_examples/executorch_cifarnet/cm33_core0/armgcc$ ./build_flash_release.sh
```

```
J-Link>loadfile C:\rt700\executorch_cifarnet_cm33_core0.elf
'loadfile': Performing implicit reset & halt of MCU.
ResetTarget() start
-- JLINK ResetTarget --
Set CM33_SYSRESETREQ_EN
CPU0 CSW: 0x03000002
ResetTarget() end - Took 88.9ms
Downloading file [C:\rt700\executorch_cifarnet_cm33_core0.elf]...
J-Link: Flash download: Bank 0 @ 0x28000000: 1 range affected (315392 bytes)
J-Link: Flash download: Total: 20.220s (Prepare: 0.145s, Compare: 2.874s, Erase: 2.558s, Program: 13.212s, Verify: 1.392s, Restore: 0
.036s)
J-Link: Flash download: Program speed: 22 KB/s
O.K.
J-Link>reset
Reset delay: 0 ms
Reset type NORMAL: Resets core & peripherals via SYSRESETREQ & VECTRESET bit.
ResetTarget() start
-- JLINK ResetTarget --
Set CM33_SYSRESETREQ_EN
CPU0 CSW: 0x03000002
ResetTarget() end - Took 71.5ms
J-Link>go
Memory map 'after startup completion point' is active
```

```
Model PTE file loaded. Size: 99376 bytes.
Model buffer loaded, has 1 methods
Running method forward
Setting up planned buffer 0, size 53760.
Method loaded.
Preparing inputs...
Input prepared.
Starting the model execution...
Model executed successfully.
---------------------------------------
     Inference time: 11950 us
---------------------------------------
1 outputs:
Output[0][0]: 0
Output[0][1]: 0
Output[0][2]: 0
Output[0][3]: 0.996094
Output[0][4]: 0
Output[0][5]: 0
Output[0][6]: 0
Output[0][7]: 0
Output[0][8]: 0
Output[0][9]: 0
Program complete, exiting.
```

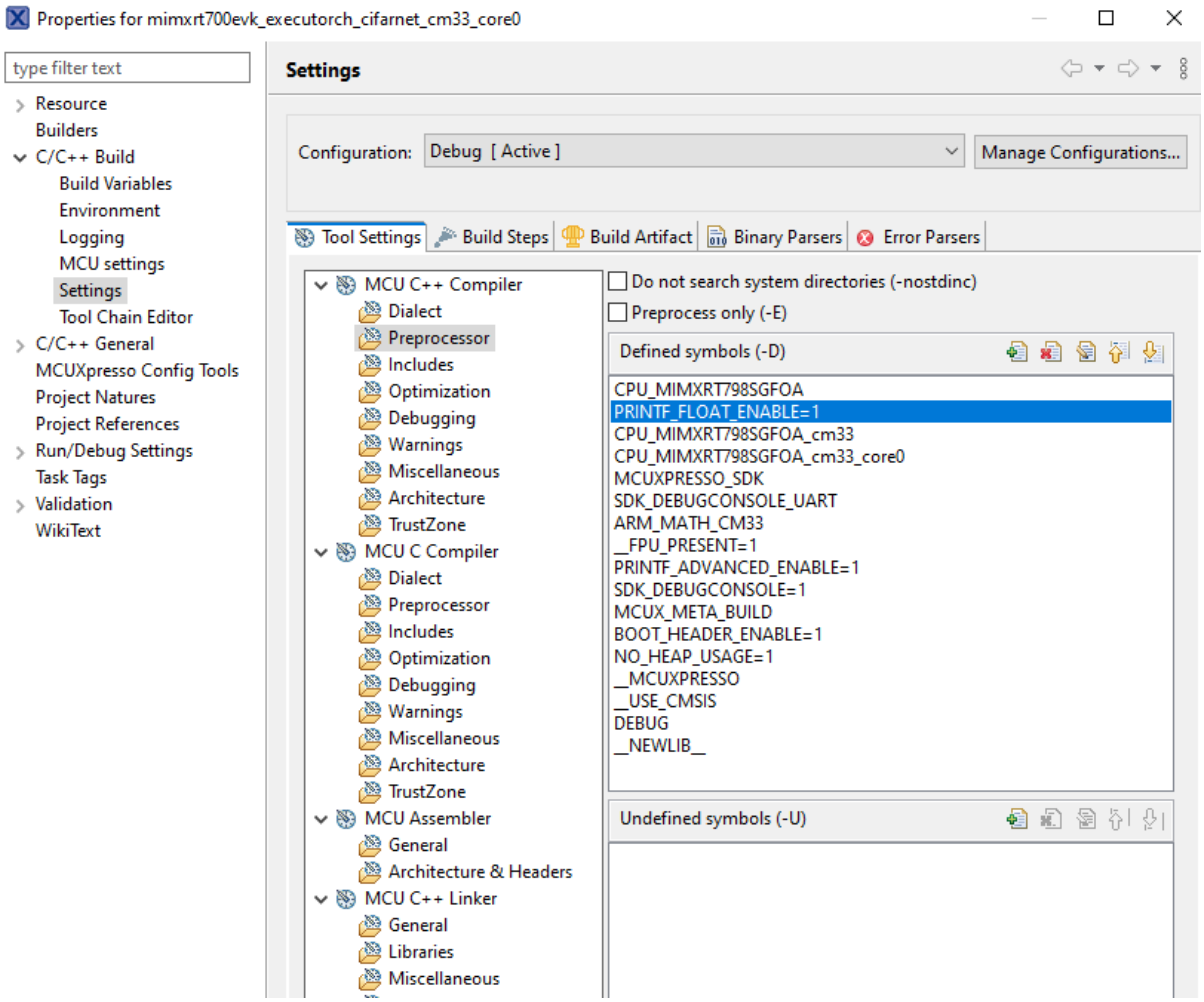
2. When using MCUXpresso IDE, the example applications can be imported through the SDK Import Wizard as shown in *Figure 5*.



After building the example application and downloading it to the target, the execution stops in the *main* function. When the execution resumes, an output message displays on the connected terminal. For example, *Figure 6* shows the output of the executorch_cifarnet example application.

```
Model PTE file loaded. Size: 99376 bytes.
Model buffer loaded, has 1 methods
Running method forward
Setting up planned buffer 0, size 53760.
Method loaded.
Preparing inputs...
Input prepared.
Starting the model execution...
Model executed successfully.
---------------------------------------
     Inference time: 14855 us
---------------------------------------
1 outputs:
Output[0][0]: 0
Output[0][1]: 0
Output[0][2]: 0
Output[0][3]: 0.996094
Output[0][4]: 0
Output[0][5]: 0
Output[0][6]: 0
Output[0][7]: 0
Output[0][8]: 0
Output[0][9]: 0
Program complete, exiting.
```

In case of missing probabilities in the printed output, add PRINTF_FLOAT_ENABLE=1 to the Preprocessor settings for C++ and C compiler:



How to build executorch_lib **example** If you want to build a new ExecuTorch Runtime Library, follow the commands as below and use the new library to replace the default Runtime library middleware/eiq/executorch/lib/cm33/armgcc/libexecutorch.a.

1. When using ARMGCC toolchain, the example application can be built as below.

```
$ boards/mimxrt700evk/eiq_examples/executorch_lib/cm33_core0/armgcc$ ./build_release.sh
$ boards/mimxrt700evk/eiq_examples/executorch_lib/cm33_core0/armgcc$ cp release/libexecutorch_lib_
↪cm33_core0.a.a ../../../../../../middleware/eiq/executorch/lib/cm33/armgcc/libexecutorch.a
```

2. When using MCUXpresso IDE, the example applications can be imported through the SDK Import Wizard as shown in the above *Figure 5*.

After building the example application, copy the new library mimxrt700evk_executorch_lib_cm33_core0\Debug\libmimxrt700evk_executorch_lib_cm33_core0.a to replace the default Runtime library mimxrt700evk_executorch_cifarnet_cm33_core0\eiq\executorch\lib\cm33\armgcc\libexecutorch.a.

[1] Learning Multiple Layers of Features from Tiny Images, Alex Krizhevsky, 2009

2.5 File System

2.5.1 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

[R0.15_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.

2.6 Motor Control

2.6.1 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")
```

(continues on next page)

(continued from previous page)

```
/* 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.

2.7 MultiCore

2.7.1 Multicore SDK

Multicore Software Development Kit (MCSDK) is a Software Development Kit that provides comprehensive software support for NXP dual/multicore devices. The MCSDK is combined with the MCUXpresso SDK to make the software framework for easy development of multicore applications.

Multicore SDK (MCSDK) Release Notes

Overview These are the release notes for the NXP Multicore Software Development Kit (MCSDK) version 25.06.00.
This software package contains components for efficient work with multicore devices as well as for the
multiprocessor communication.

What is new

- eRPC CHANGELOG
- RPMsg-Lite CHANGELOG
- MCMgr CHANGELOG
- Supported evaluation boards (multicore examples):
  - LPCXpresso55S69
  - FRDM-K32L3A6
  - MIMXRT1170-EVKB
  - MIMXRT1160-EVK
  - MIMXRT1180-EVK
  - MCX-N5XX-EVK
  - MCX-N9XX-EVK
  - FRDM-MCXN947
  - MIMXRT700-EVK
  - KW47-EVK
  - KW47-LOC
  - FRDM-MCXW72
  - MCX-W72-EVK
- Supported evaluation boards (multiprocessor examples):
  - LPCXpresso55S36
  - FRDM-K22F
  - FRDM-K32L2B
  - MIMXRT685-EVK
  - MIMXRT1170-EVKB
  - MIMXRT1180
  - FRDM-MCXN236
  - FRDM-MCXC242
  - FRDM-MCXC444
  - MCX-N9XX-EVK
  - FRDM-MCXN947
  - MIMXRT700-EVK

Development tools The Multicore SDK (MCSDK) was compiled and tested with development tools referred in: Development tools

Release contents This table describes the release contents. Not all MCUXpresso SDK packages contain the whole set of these components.

Deliverable	Location
Multicore SDK location <MCSDK_dir>	<MCUXpressoSDK_install_dir>/middleware/multicore/
Documentation	<MCSDK_dir>/mcuxsdk-doc/
Embedded Remote Procedure Call component	<MCSDK_dir>/erpc/
Multicore Manager component	<MCSDK_dir>/mcmgr/
RPMsg-Lite	<MCSDK_dir>/rpmsg_lite/
Multicore demo applications	<MCUXpressoSDK_install_dir>/examples/multicore_examples/
Multiprocessor demo applications	<MCUXpressoSDK_install_dir>/examples/multiprocessor_examples/

Multicore SDK release overview Together, the Multicore SDK (MCSDK) and the MCUXpresso SDK (SDK) form a framework for the development of software for NXP multicore devices. The MCSDK release consists of the following elementary software components for multicore:

- Embedded Remote Procedure Call (eRPC)
- Multicore Manager (MCMGR) - included just in SDK for multicore devices
- Remote Processor Messaging - Lite (RPMsg-Lite) - included just in SDK for multicore devices

The MCSDK is also accompanied with documentation and several multicore and multiprocessor demo applications.

Demo applications The multicore demo applications demonstrate the usage of the MCSDK software components on supported multicore development boards.
The following multicore demo applications are located together with other MCUXpresso SDK examples in
the <MCUXpressoSDK_install_dir>/examples/multicore_examples subdirectories.

- erpc_matrix_multiply_mu
- erpc_matrix_multiply_mu_rtos
- erpc_matrix_multiply_rpmsg
- erpc_matrix_multiply_rpmsg_rtos
- erpc_two_way_rpc_rpmsg_rtos
- freertos_message_buffers
- hello_world
- multicore_manager
- rpmsg_lite_pingpong
- rpmsg_lite_pingpong_rtos
- rpmsg_lite_pingpong_tzm

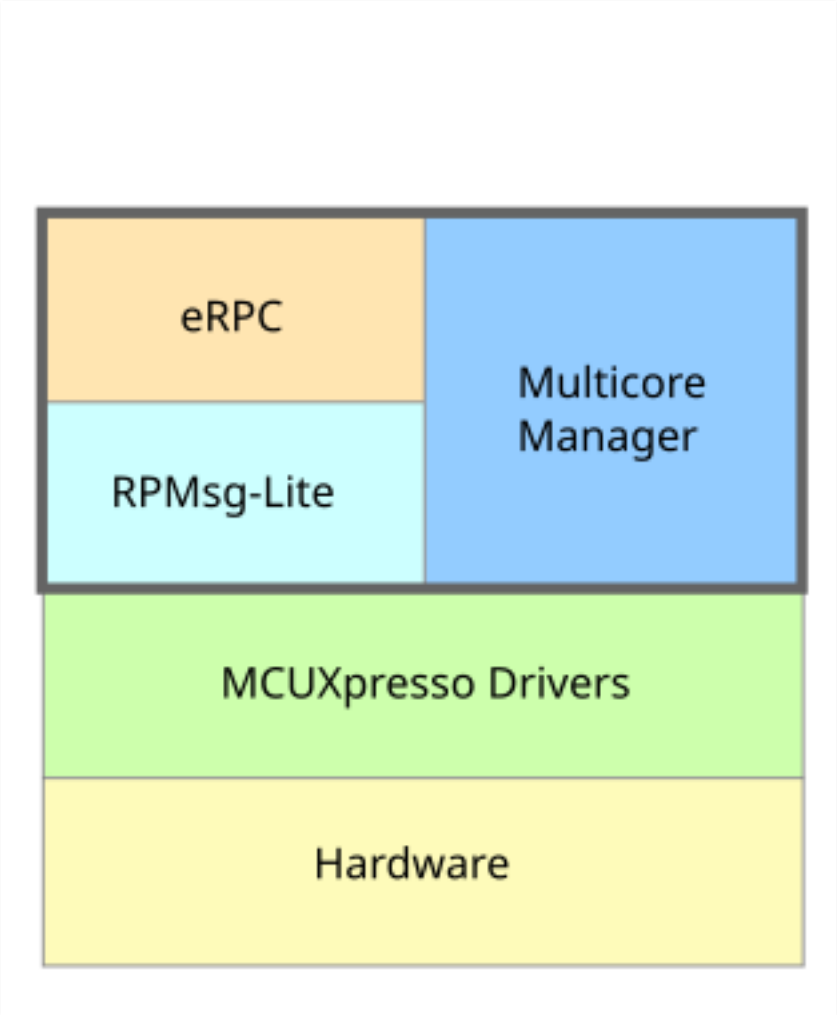
The eRPC multicore component can be leveraged for inter-processor communication and remote procedure calls between SoCs / development boards.
The following multiprocessor demo applications are located together with other MCUXpresso SDK examples in
the <MCUXpressoSDK_install_dir>/examples/multiprocessor_examples subdirectories.

- erpc_client_matrix_multiply_spi
- erpc_server_matrix_multiply_spi
- erpc_client_matrix_multiply_uart
- erpc_server_matrix_multiply_uart
- erpc_server_dac_adc
- erpc_remote_control

Getting Started with Multicore SDK (MCSDK)

Overview Multicore Software Development Kit (MCSDK) is a Software Development Kit that provides comprehensive software support for NXP dual/multicore devices. The MCSDK is combined with the MCUXpresso SDK to make the software framework for easy development of multicore applications.

The following figure highlights the layers and main software components of the MCSDK.

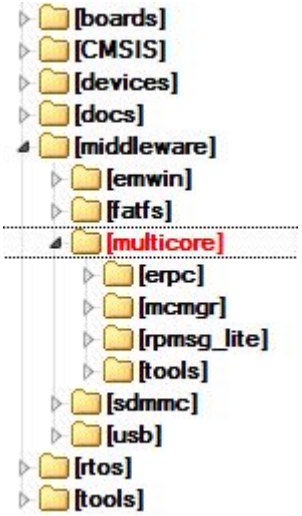


All the MCSDK-related files are located in <MCUXpressoSDK_install_dir>/middleware/multicore folder.

For supported toolchain versions, see the *Multicore SDK v25.06.00 Release Notes* (document MCSDKRN). For the latest version of this and other MCSDK documents, visit www.nxp.com.

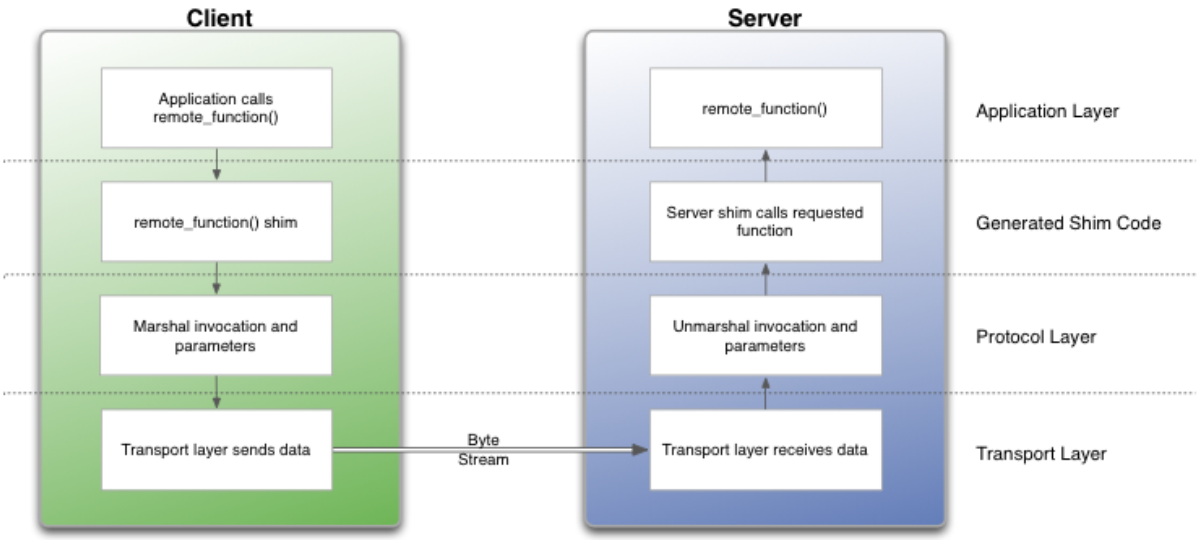
Multicore SDK (MCSDK) components The MCSDK consists of the following software components:

- **Embedded Remote Procedure Call (eRPC):** This component is a combination of a library and code generator tool that implements a transparent function call interface to remote services (running on a different core).
- **Multicore Manager (MCMGR):** This library maintains information about all cores and starts up secondary/auxiliary cores.
- **Remote Processor Messaging - Lite (RPMsg-Lite):** Inter-Processor Communication library.



Embedded Remote Procedure Call (eRPC) The Embedded Remote Procedure Call (eRPC) is the RPC system created by NXP. The RPC is a mechanism used to invoke a software routine on a remote system via a simple local function call.

When a remote function is called by the client, the function's parameters and an identifier for the called routine are marshaled (or serialized) into a stream of bytes. This byte stream is transported to the server through a communications channel (IPC, TPC/IP, UART, and so on). The server unmarshaled the parameters, determines which function was invoked, and calls it. If the function returns a value, it is marshaled and sent back to the client.



RPC implementations typically use a combination of a tool (erpcgen) and IDL (interface definition language) file to generate source code to handle the details of marshaling a function's parameters and building the data stream.

Main eRPC features:

- Scalable from BareMetal to Linux OS - configurable memory and threading policies.
- Focus on embedded systems - intrinsic support for C, modular, and lightweight implementation.
- Abstracted transport interface - RPMsg is the primary transport for multicore, UART, or SPI-based solutions can be used for multichip.

The eRPC library is located in the <MCUXpressoSDK_install_dir>/middleware/multicore/erpc folder. For detailed information about the eRPC, see the documentation available in the <MCUXpressoSDK_install_dir>/middleware/multicore/erpc/doc folder.

Multicore Manager (MCMGR) The Multicore Manager (MCMGR) software library provides a number of services for multicore systems.

The main MCMGR features:

- Maintains information about all cores in system.
- Secondary/auxiliary cores startup and shutdown.
- Remote core monitoring and event handling.

The MCMGR library is located in the <MCUXpressoSDK_install_dir>/middleware/multicore/mcmgr folder. For detailed information about the MCMGR library, see the documentation available in the <MCUXpressoSDK_install_dir>/middleware/multicore/mcmgr/doc folder.

Remote Processor Messaging Lite (RPMsg-Lite) RPMsg-Lite is a lightweight implementation of the RPMsg protocol. The RPMsg protocol defines a standardized binary interface used to communicate between multiple cores in a heterogeneous multicore system. Compared to the legacy OpenAMP implementation, RPMsg-Lite offers a code size reduction, API simplification, and improved modularity.

The main RPMsg protocol features:

- Shared memory interprocessor communication.
- Virtio-based messaging bus.
- Application-defined messages sent between endpoints.

- Portable to different environments/platforms.
- Available in upstream Linux OS.

The RPMsg-Lite library is located in the <MCUXpressoSDK_install_dir>/middleware/multicore/rpmsg-lite folder. For detailed information about the RPMsg-Lite, see the RPMsg-Lite User's Guide located in the <MCUXpressoSDK_install_dir>/middleware/multicore/rpmsg_lite/doc folder.

MCSDK demo applications Multicore and multiprocessor example applications are stored together with other MCUXpresso SDK examples, in the dedicated multicore subfolder.

Location	Folder
Multicore example projects	<MCUXpressoSDK_install_dir>/examples/multicore_examples/<application_name>/
Multiprocessor example projects	<MCUXpressoSDK_install_dir>/examples/multiprocessor_examples/<application_name>/

See the *Getting Started with MCUXpresso SDK* (document MCUXSDKGSUG) and *Getting Started with MCUXpresso SDK for XXX Derivatives* documents for more information about the MCUXpresso SDK example folder structure and the location of individual files that form the example application projects. These documents also contain information about building, running, and debugging multicore demo applications in individual supported IDEs. Each example application also contains a readme file that describes the operation of the example and required setup steps.

Inter-Processor Communication (IPC) levels The MCSDK provides several mechanisms for Inter-Processor Communication (IPC). Particular ways and levels of IPC are described in this chapter.

IPC using low-level drivers

The NXP multicore SoCs are equipped with peripheral modules dedicated for data exchange between individual cores. They deal with the Mailbox peripheral for LPC parts and the Messaging Unit (MU) peripheral for Kinetis and i.MX parts. The common attribute of both modules is the ability to provide a means of IPC, allowing multiple CPUs to share resources and communicate with each other in a simple manner.

The most lightweight method of IPC uses the MCUXpresso SDK low-level drivers for these peripherals. Using the Mailbox/MU driver API functions, it is possible to pass a value from core to core via the dedicated registers (could be a scalar or a pointer to shared memory) and also to trigger inter-core interrupts for notifications.

For details about individual driver API functions, see the MCUXpresso SDK API Reference Manual of the specific multicore device. The MCUXpresso SDK is accompanied with the RPMsg-Lite documentation that shows how to use this API in multicore applications.

Messaging mechanism

On top of Mailbox/MU drivers, a messaging system can be implemented, allowing messages to send between multiple endpoints created on each of the CPUs. The RPMsg-Lite library of the MCSDK provides this ability and serves as the preferred MCUXpresso SDK messaging library. It implements ring buffers in shared memory for messages exchange without the need of a locking mechanism.

The RPMsg-Lite provides the abstraction layer and can be easily ported to different multicore platforms and environments (Operating Systems). The advantages of such a messaging system are ease of use (there is no need to study behavior of the used underlying hardware) and smooth application code portability between platforms due to unified messaging API.

However, this costs several kB of code and data memory. The MCUXpresso SDK is accompanied by the RPMsg-Lite documentation and several multicore examples. You can also obtain the latest RPMsg-Lite code from the GitHub account github.com/nxp-mcuxpresso/rpmsg-lite.

Remote procedure calls

To facilitate the IPC even more and to allow the remote functions invocation, the remote procedure call mechanism can be implemented. The eRPC of the MCSDK serves for these purposes and allows the ability to invoke a software routine on a remote system via a simple local function call. Utilizing different transport layers, it is possible to communicate between individual cores of multicore SoCs (via RPMsg-Lite) or between separate processors (via SPI, UART, or TCP/IP). The eRPC is mostly applicable to the MPU parts with enough of memory resources like i.MX parts.

The eRPC library allows you to export existing C functions without having to change their prototypes (in most cases). It is accompanied by the code generator tool that generates the shim code for serialization and invocation based on the IDL file with definitions of data types and remote interfaces (API).

If the communicating peer is running as a Linux OS user-space application, the generated code can be either in C/C++ or Python.

Using the eRPC simplifies the access to services implemented on individual cores. This way, the following types of applications running on dedicated cores can be easily interfaced:

- Communication stacks (USB, Thread, Bluetooth Low Energy, Zigbee)
- Sensor aggregation/fusion applications
- Encryption algorithms
- Virtual peripherals

The eRPC is publicly available from the following GitHub account: github.com/EmbeddedRPC/erpc. Also, the MCUXpresso SDK is accompanied by the eRPC code and several multicore and multiprocessor eRPC examples.

The mentioned IPC levels demonstrate the scalability of the Multicore SDK library. Based on application needs, different IPC techniques can be used. It depends on the complexity, required speed, memory resources, system design, and so on. The MCSDK brings users the possibility for quick and easy development of multicore and multiprocessor applications.

Changelog Multicore SDK

All notable changes to this project will be documented in this file.

The format is based on Keep a Changelog, and this project adheres to Semantic Versioning.

[25.06.00]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.14.0
  - eRPC generator (erpcgen) v1.14.0
  - Multicore Manager (MCMgr) v5.0.0
  - RPMsg-Lite v5.2.0

[25.03.00]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.13.0

  - eRPC generator (erpcgen) v1.13.0
  - Multicore Manager (MCMgr) v4.1.7
  - RPMsg-Lite v5.1.4

[24.12.00]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.13.0
  - eRPC generator (erpcgen) v1.13.0
  - Multicore Manager (MCMgr) v4.1.6
  - RPMsg-Lite v5.1.3

[2.16.0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.13.0
  - eRPC generator (erpcgen) v1.13.0
  - Multicore Manager (MCMgr) v4.1.5
  - RPMsg-Lite v5.1.2

[2.15.0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.12.0
  - eRPC generator (erpcgen) v1.12.0
  - Multicore Manager (MCMgr) v4.1.5
  - RPMsg-Lite v5.1.1

[2.14.0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.11.0
  - eRPC generator (erpcgen) v1.11.0
  - Multicore Manager (MCMgr) v4.1.4
  - RPMsg-Lite v5.1.0

[2.13.0_imxrt1180a0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.10.0
  - eRPC generator (erpcgen) v1.10.0
  - Multicore Manager (MCMgr) v4.1.3
  - RPMsg-Lite v5.0.0

[2.13.0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.10.0
  - eRPC generator (erpcgen) v1.10.0
  - Multicore Manager (MCMgr) v4.1.3
  - RPMsg-Lite v5.0.0

[2.12.0_imx93]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.9.1
  - eRPC generator (erpcgen) v1.9.1
  - Multicore Manager (MCMgr) v4.1.2
  - RPMsg-Lite v4.0.1

[2.12.0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.9.1
  - eRPC generator (erpcgen) v1.9.1
  - Multicore Manager (MCMgr) v4.1.2
  - RPMsg-Lite v4.0.0

[2.11.1]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.9.0
  - eRPC generator (erpcgen) v1.9.0
  - Multicore Manager (MCMgr) v4.1.1
  - RPMsg-Lite v3.2.1

[2.11.0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.9.0
  - eRPC generator (erpcgen) v1.9.0
  - Multicore Manager (MCMgr) v4.1.1
  - RPMsg-Lite v3.2.0

[2.10.0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.8.1
  - eRPC generator (erpcgen) v1.8.1
  - Multicore Manager (MCMgr) v4.1.1
  - RPMsg-Lite v3.1.2

[2.9.0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.8.0
  - eRPC generator (erpcgen) v1.8.0
  - Multicore Manager (MCMgr) v4.1.1
  - RPMsg-Lite v3.1.1

[2.8.0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.7.4
  - eRPC generator (erpcgen) v1.7.4
  - Multicore Manager (MCMgr) v4.1.0
  - RPMsg-Lite v3.1.0

[2.7.0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.7.3
  - eRPC generator (erpcgen) v1.7.3
  - Multicore Manager (MCMgr) v4.1.0
  - RPMsg-Lite v3.0.0

[2.6.0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.7.2
  - eRPC generator (erpcgen) v1.7.2
  - Multicore Manager (MCMgr) v4.0.3
  - RPMsg-Lite v2.2.0

[2.5.0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.7.1
  - eRPC generator (erpcgen) v1.7.1
  - Multicore Manager (MCMgr) v4.0.2
  - RPMsg-Lite v2.0.2

[2.4.0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.7.0
  - eRPC generator (erpcgen) v1.7.0
  - Multicore Manager (MCMgr) v4.0.1
  - RPMsg-Lite v2.0.1

[2.3.1]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.6.0
  - eRPC generator (erpcgen) v1.6.0
  - Multicore Manager (MCMgr) v4.0.0
  - RPMsg-Lite v1.2.0

[2.3.0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.5.0
  - eRPC generator (erpcgen) v1.5.0
  - Multicore Manager (MCMgr) v3.0.0
  - RPMsg-Lite v1.2.0

[2.2.0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.4.0
  - eRPC generator (erpcgen) v1.4.0
  - Multicore Manager (MCMgr) v2.0.1
  - RPMsg-Lite v1.1.0

[2.1.0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.3.0
  - eRPC generator (erpcgen) v1.3.0

[2.0.0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.2.0
  - eRPC generator (erpcgen) v1.2.0
  - Multicore Manager (MCMgr) v2.0.0
  - RPMsg-Lite v1.0.0

[1.1.0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.1.0
  - Multicore Manager (MCMgr) v1.1.0
  - Open-AMP / RPMsg based on SHA1 ID 44b5f3c0a6458f3cf80 rev01

[1.0.0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.0.0
  - Multicore Manager (MCMgr) v1.0.0
  - Open-AMP / RPMsg based on SHA1 ID 44b5f3c0a6458f3cf80 rev00

Multicore SDK Components

RPMSG-Lite

MCUXpresso SDK : mcuxsdk-middleware-rpmsg-lite

Overview This repository is for MCUXpresso SDK RPMSG-Lite middleware delivery and it contains RPMSG-Lite component 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 for the complete delivery of MCUXpresso SDK to be able to build and run RPMSG-Lite examples that are based on mcux-sdk-middleware-rpmsg-lite component.

Documentation Overall details can be reviewed here: MCUXpresso SDK Online Documentation

Visit RPMSG-Lite - 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 We welcome and encourage the community to submit patches directly to the rpmsg-lite project placed on github. Contributing can be managed via pull-requests. Before a pull-request is created the code should be tested and properly formatted.

RPMSG-Lite This documentation describes the RPMsg-Lite component, which is a lightweight implementation of the Remote Processor Messaging (RPMsg) protocol. The RPMsg protocol defines a standardized binary interface used to communicate between multiple cores in a heterogeneous multicore system.

Compared to the RPMsg implementation of the Open Asymmetric Multi Processing (OpenAMP) framework (https://github.com/OpenAMP/open-amp), the RPMsg-Lite offers a code size reduction, API simplification, and improved modularity. On smaller Cortex-M0+ based systems, it is recommended to use RPMsg-Lite.

The RPMsg-Lite is an open-source component developed by NXP Semiconductors and released under the BSD-compatible license.

For Further documentation, please look at doxygen documentation at: https://nxp-mcuxpresso.github.io/rpmsg-lite/

Motivation to create RPMsg-Lite There are multiple reasons why RPMsg-Lite was developed. One reason is the need for the small footprint of the RPMsg protocol-compatible communication component, another reason is the simplification of extensive API of OpenAMP RPMsg implementation.

RPMsg protocol was not documented, and its only definition was given by the Linux Kernel and legacy OpenAMP implementations. This has changed with [1] which is a standardization protocol allowing multiple different implementations to coexist and still be mutually compatible.

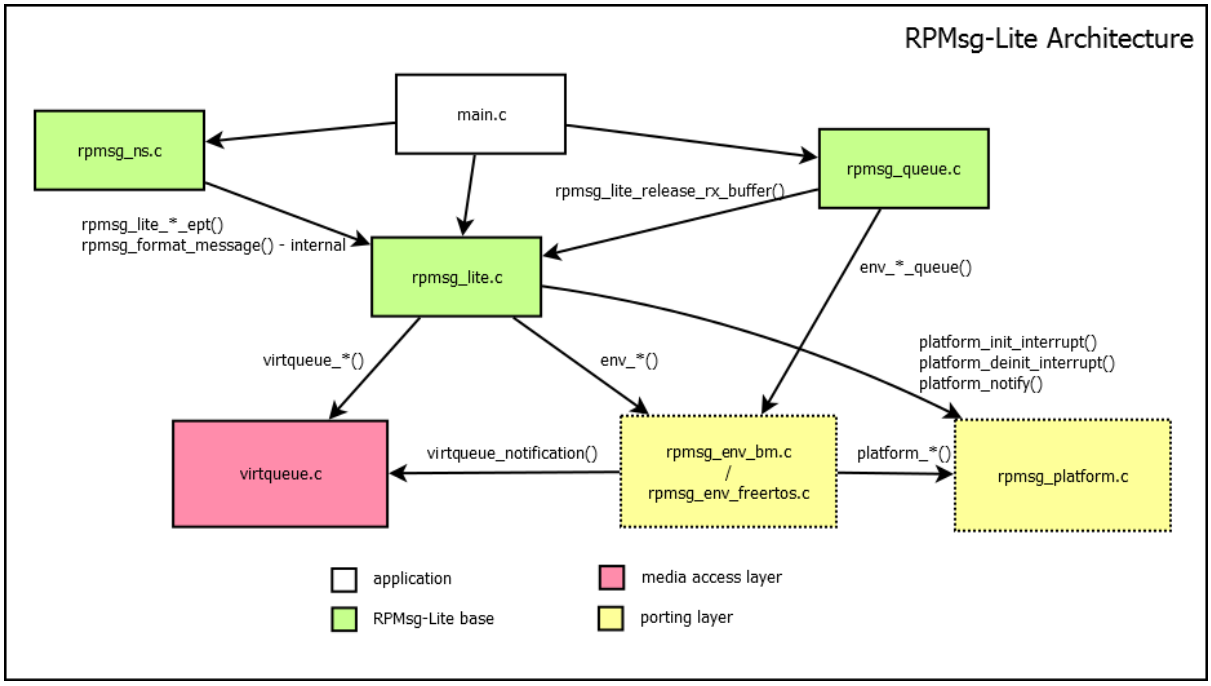
Small MCU-based systems often do not implement dynamic memory allocation. The creation of static API in RPMsg-Lite enables another reduction of resource usage. Not only does the dynamic allocation adds another 5 KB of code size, but also communication is slower and less deterministic, which is a property introduced by dynamic memory. The following table shows some rough comparison data between the OpenAMP RPMsg implementation and new RPMsg-Lite implementation:

Component / Configuration	Flash [B]	RAM [B]
OpenAMP RPMsg / Release (reference)	5547	456 + dynamic
RPMsg-Lite / Dynamic API, Release	3462	56 + dynamic
Relative Difference [%]	~62.4%	~12.3%
RPMsg-Lite / Static API (no malloc), Release	2926	352
Relative Difference [%]	~52.7%	~77.2%

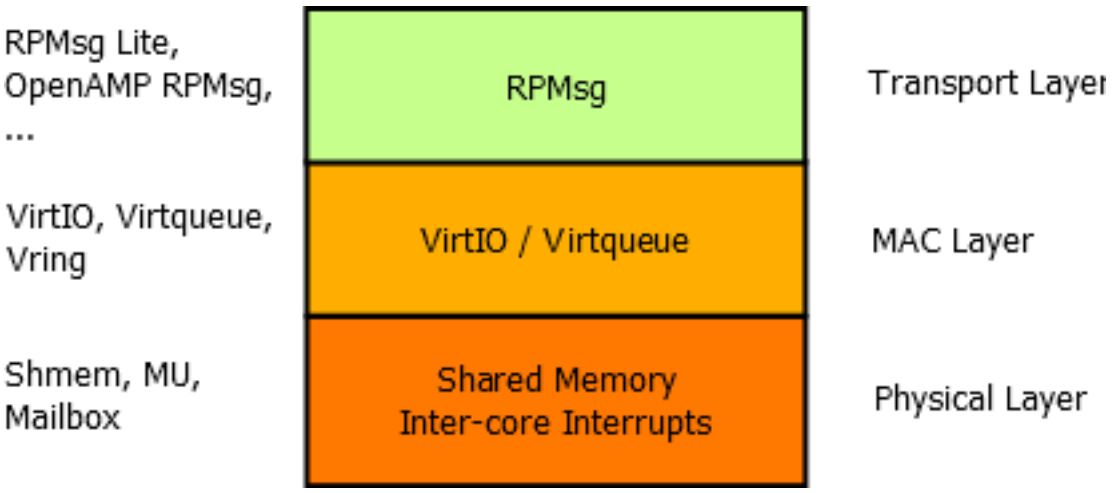
Implementation The implementation of RPMsg-Lite can be divided into three subcomponents, from which two are optional. The core component is situated in rpmsg_lite.c. Two optional components are used to implement a blocking receive API (in rpmsg_queue.c) and dynamic “named” endpoint creation and deletion announcement service (in rpmsg_ns.c).

The actual “media access” layer is implemented in virtqueue.c, which is one of the few files shared with the OpenAMP implementation. This layer mainly defines the shared memory model, and internally defines used components such as vring or virtqueue.

The porting layer is split into two sub-layers: the environment layer and the platform layer. The first sublayer is to be implemented separately for each environment. (The bare metal environment already exists and is implemented in rpmsg_env_bm.c, and the FreeRTOS environment is implemented in rpmsg_env_freertos.c etc.) Only the source file, which matches the used environment, is included in the target application project. The second sublayer is implemented in rpmsg_platform.c and defines low-level functions for interrupt enabling, disabling, and triggering mainly. The situation is described in the following figure:



RPMsg-Lite core sub-component This subcomponent implements a blocking send API and callback-based receive API. The RPMsg protocol is part of the transport layer. This is realized by using so-called endpoints. Each endpoint can be assigned a different receive callback function. However, it is important to notice that the callback is executed in an interrupt environment in current design. Therefore, certain actions like memory allocation are discouraged to execute in the callback. The following figure shows the role of RPMsg in an ISO/OSI-like layered model:



Queue sub-component (optional) This subcomponent is optional and requires implementation of the env_*_queue() functions in the environment porting layer. It uses a blocking receive API, which is common in RTOS-environments. It supports both copy and nocopy blocking receive functions.

Name Service sub-component (optional) This subcomponent is a minimum implementation of the name service which is present in the Linux Kernel implementation of RPMsg. It allows the communicating node both to send announcements about “named” endpoint (in other words, channel) creation or deletion and to receive these announcement taking any user-defined action

in an application callback. The endpoint address used to receive name service announcements is arbitrarily fixed to be 53 (0x35).

Usage The application should put the /rpmsg_lite/lib/include directory to the include path and in the application, include either the rpmsg_lite.h header file, or optionally also include the rpmsg_queue.h and/or rpmsg_ns.h files. Both porting sublayers should be provided for you by NXP, but if you plan to use your own RTOS, all you need to do is to implement your own environment layer (in other words, rpmsg_env_myrtos.c) and to include it in the project build.

The initialization of the stack is done by calling the rpmsg_lite_master_init() on the master side and the rpmsg_lite_remote_init() on the remote side. This initialization function must be called prior to any RPMsg-Lite API call. After the init, it is wise to create a communication endpoint, otherwise communication is not possible. This can be done by calling the rpmsg_lite_create_ept() function. It optionally accepts a last argument, where an internal context of the endpoint is created, just in case the RL_USE_STATIC_API option is set to 1. If not, the stack internally calls env_alloc() to allocate dynamic memory for it. In case a callback-based receiving is to be used, an ISR-callback is registered to each new endpoint with user-defined callback data pointer. If a blocking receive is desired (in case of RTOS environment), the rpmsg_queue_create() function must be called before calling rpmsg_lite_create_ept(). The queue handle is passed to the endpoint creation function as a callback data argument and the callback function is set to rpmsg_queue_rx_cb(). Then, it is possible to use rpmsg_queue_receive() function to listen on a queue object for incoming messages. The rpmsg_lite_send() function is used to send messages to the other side.

The RPMsg-Lite also implements no-copy mechanisms for both sending and receiving operations. These methods require specifics that have to be considered when used in an application.

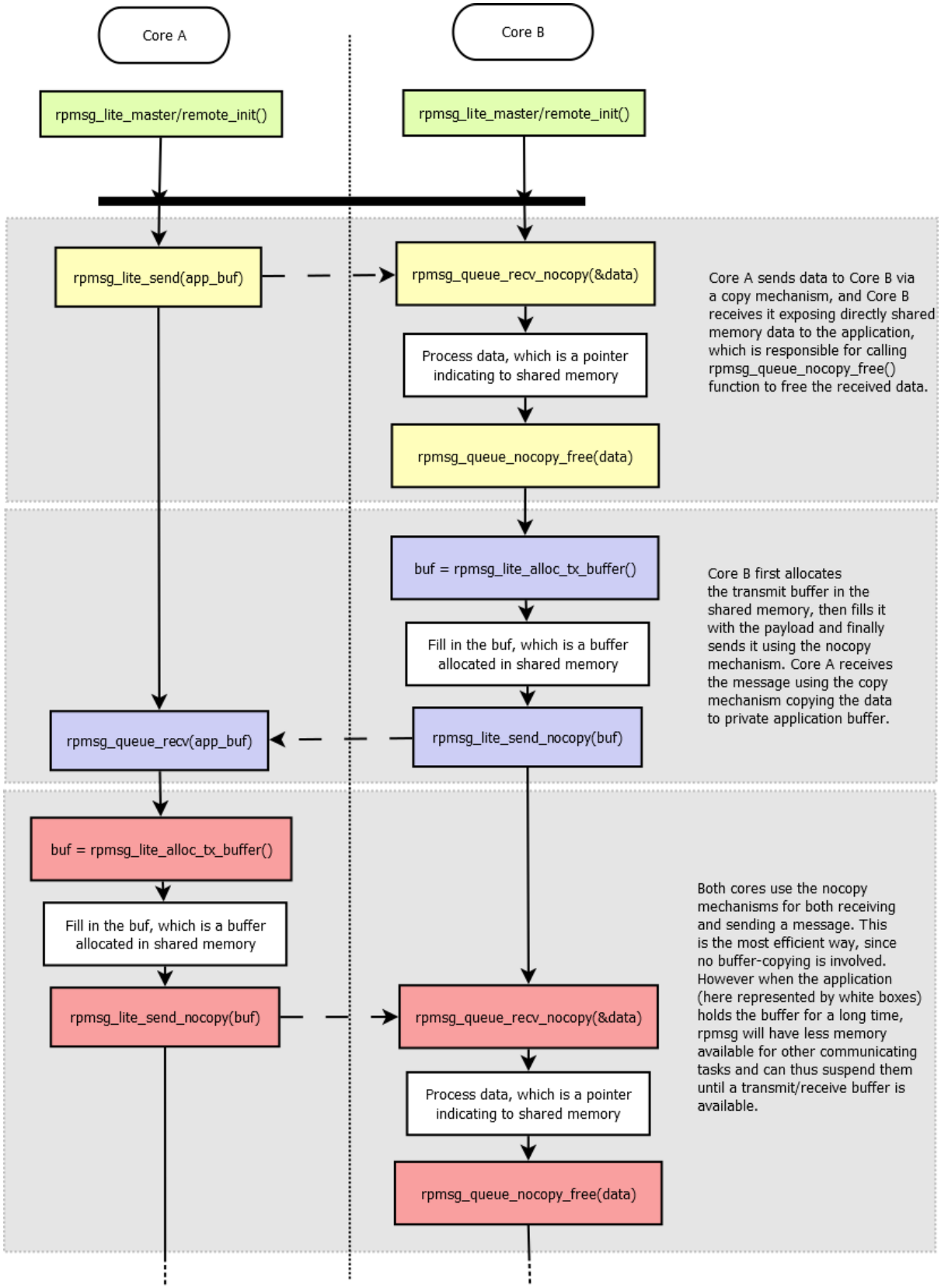
no-copy-send mechanism: This mechanism allows sending messages without the cost for copying data from the application buffer to the RPMsg/virtio buffer in the shared memory. The sequence of no-copy sending steps to be performed is as follows:

- Call the rpmsg_lite_alloc_tx_buffer() function to get the virtio buffer and provide the buffer pointer to the application.
- Fill the data to be sent into the pre-allocated virtio buffer. Ensure that the filled data does not exceed the buffer size (provided as the rpmsg_lite_alloc_tx_buffer() size output parameter).
- Call the rpmsg_lite_send_nocopy() function to send the message to the destination endpoint. Consider the cache functionality and the virtio buffer alignment. See the rpmsg_lite_send_nocopy() function description below.

no-copy-receive mechanism: This mechanism allows reading messages without the cost for copying data from the virtio buffer in the shared memory to the application buffer. The sequence of no-copy receiving steps to be performed is as follows:

- Call the rpmsg_queue_recv_nocopy() function to get the virtio buffer pointer to the received data.
- Read received data directly from the shared memory.
- Call the rpmsg_queue_nocopy_free() function to release the virtio buffer and to make it available for the next data transfer.

The user is responsible for destroying any RPMsg-Lite objects he has created in case of deinitialization. In order to do this, the function rpmsg_queue_destroy() is used to destroy a queue, rpmsg_lite_destroy_ept() is used to destroy an endpoint and finally, rpmsg_lite_deinit() is used to deinitialize the RPMsg-Lite intercore communication stack. Deinitialize all endpoints using a queue before deinitializing the queue. Otherwise, you are actively invalidating the used queue handle, which is not allowed. RPMsg-Lite does not check this internally, since its main aim is to be lightweight.



Examples RPMsg_Lite multicore examples are part of NXP MCUXpressoSDK packages. Visit https://mcuxpresso.nxp.com to configure, build and download these packages. To get the board list with multicore support (RPMsg_Lite included) use filtering based on Middleware and search for 'multicore' string. Once the selected package with the multicore middleware is downloaded,

see

<MCUXpressoSDK_install_dir>/boards/<board_name>/multicore_examples for RPMsg_Lite multicore examples with 'rpmsg_lite_' name prefix.

Another way of getting NXP MCUXpressoSDK RPMsg_Lite multicore examples is using the mcuxsdk-manifests Github repo. Follow the description how to use the West tool to clone and update the mcuxsdk-manifests repo in readme section. Once done the armgcc rpmsg_lite examples can be found in

mcuxsdk/examples/_<board_name>/multicore_examples

You can use the evkmimxrt1170 as the board_name for instance. Similar to MCUXpressoSDK packages the RPMsg_Lite examples use the 'rpmsg_lite_' name prefix.

Notes

Environment layers implementation Several environment layers are provided in lib/rpmsg_lite/porting/environment folder. Not all of them are fully tested however. Here is the list of environment layers that passed testing:

- rpmsg_env_bm.c
- rpmsg_env_freertos.c
- rpmsg_env_xos.c
- rpmsg_env_threadx.c

The rest of environment layers has been created and used in some experimental projects, it has been running well at the time of creation but due to the lack of unit testing there is no guarantee it is still fully functional.

Shared memory configuration It is important to correctly initialize/configure the shared memory for data exchange in the application. The shared memory must be accessible from both the master and the remote core and it needs to be configured as Non-Cacheable memory. Dedicated shared memory section in liker file is also a good practise, it is recommended to use linker files from MCUXpressSDK packages for NXP devices based applications. It needs to be ensured no other application part/component is unintentionally accessing this part of memory.

Configuration options The RPMsg-Lite can be configured at the compile time. The default configuration is defined in the rpmsg_default_config.h header file. This configuration can be customized by the user by including rpmsg_config.h file with custom settings. The following table summarizes all possible RPMsg-Lite configuration options.

Configuration option	Default value	Usage
RL_MS_PE	(1)	Delay in milliseconds used in non-blocking API functions for polling.
RL_BUFFE	(496)	Size of the buffer payload, it must be equal to (240, 496, 1008, ...) [2^n - 16]
RL_BUFFE	(2)	Number of the buffers, it must be power of two (2, 4, ...)
RL_API_H	(1)	Zero-copy API functions enabled/disabled.
RL_USE_S	(0)	Static API functions (no dynamic allocation) enabled/disabled.
RL_USE_D	(0)	Memory cache management of shared memory. Use in case of data cache is enabled for shared memory.
RL_CLEAF	(0)	Clearing used buffers before returning back to the pool of free buffers enabled/disabled.
RL_USE_M	(0)	When enabled IPC interrupts are managed by the Multicore Manager (IPC interrupts router), when disabled RPMsg-Lite manages IPC interrupts by itself.
RL_USE_E	(0)	When enabled the environment layer uses its own context. Required for some environments (QNX). The default value is 0 (no context, saves some RAM).
RL_DEBU(	(0)	When enabled buffer pointers passed to rpmsg_lite_send_nocopy() and rpmsg_lite_release_rx_buffer() functions (enabled by RL_API_HAS_ZEROCOPY config) are checked to avoid passing invalid buffer pointer. The default value is 0 (disabled). Do not use in RPMsg-Lite to Linux configuration.
RL_ALLO\	(0)	When enabled the opposite side is notified each time received buffers are consumed and put into the queue of available buffers. Enable this option in RPMsg-Lite to Linux configuration to allow unblocking of the Linux blocking send. The default value is 0 (RPMsg-Lite to RPMsg-Lite communication).
RL_ALLO\	(0)	It allows to define custom shared memory configuration and replacing the shared memory related global settings from rpmsg_config.h This is useful when multiple instances are running in parallel but different shared memory arrangement (vring size & alignment, buffers size & count) is required. The default value is 0 (all RPMsg_Lite instances use the same shared memory arrangement as defined by common config macros).
RL_ASSER	see rpmsg	Assert implementation.

How to format rpmsg-lite code To format code, use the application developed by Google, named *clang-format*. This tool is part of the llvm project. Currently, the clang-format 10.0.0 version is used for rpmsg-lite. The set of style settings used for clang-format is defined in the .clang-format file, placed in a root of the rpmsg-lite directory where Python script run_clang_format.py can be executed. This script executes the application named *clang-format.exe*. You need to have the path of this application in the OS's environment path, or you need to change the script.

References

[1] M. Novak, M. Cingel, Lockless Shared Memory Based Multicore Communication Protocol
Copyright © 2016 Freescale Semiconductor, Inc. Copyright © 2016-2025 NXP

Changelog RPMSG-Lite All notable changes to this project will be documented in this file.

The format is based on Keep a Changelog, and this project adheres to Semantic Versioning.

v5.2.0

Added

- Add MCXL20 porting layer and unit testing
- New utility macro RL_CALCULATE_BUFFER_COUNT_DOWN_SAFE to safely determine maximum buffer count within shared memory while preventing integer underflow.
- RT700 platform add support for MCMGR in DSPs

Changed

- Change rpmsg_platform.c to support new MCMGR API
- Improved input validation in initialization functions to properly handle insufficient memory size conditions.
- Refactored repeated buffer count calculation pattern for better code maintainability.
- To make sure that remote has already registered IRQ there is required App level IPC mechanism to notify master about it

Fixed

- Fixed env_wait_for_link_up function to handle timeout in link state checks for baremetal and qnx environment, RL_BLOCK mode can be used to wait indefinitely.
- Fixed CERT-C INT31-C violation by adding compile-time check to ensure RL_PLATFORM_HIGHEST_LINK_ID remains within safe range for 16-bit casting in virtqueue ID creation.
- Fixed CERT-C INT30-C violations by adding protection against unsigned integer underflow in shared memory calculations, specifically in shmem_length - (uint32_t)RL_VRING_OVERHEAD and shmem_length - 2U * shmem_config.vring_size expressions.
- Fixed CERT INT31-C violation in platform_interrupt_disable() and similar functions by replacing unsafe cast from uint32_t to int32_t with a return of 0 constant.
- Fixed unsigned integer underflow in rpmsg_lite_alloc_tx_buffer() where subtracting header size from buffer size could wrap around if buffer was too small, potentially leading to incorrect buffer sizing.
- Fixed CERT-C INT31-C violation in rpmsg_lite.c where size parameter was cast from uint32_t to uint16_t without proper validation.
  - Applied consistent masking approach to both size and flags parameters: (uint16_t)(value & 0xFFFFU).
  - This fix prevents potential data loss when size values exceed 65535.
- Fixed CERT INT31-C violation in env_memset functions by explicitly converting int32_t values to unsigned char using bit masking. This prevents potential data loss or misinterpretation when passing values outside the unsigned char range (0-255) to the standard memset() function.
- Fixed CERT-C INT31-C violations in RPMsg-Lite environment porting: Added validation checks for signed-to-unsigned integer conversions to prevent data loss and misinterpretation.
  - rpmsg_env_freertos.c: Added validation before converting int32_t to UBaseType_t.
  - rpmsg_env_qnx.c: Fixed format string and added validation before assigning to mqstat fields.

  - rpmsg_env_threadx.c: Added validation to prevent integer overflow and negative values.
  - rpmsg_env_xos.c: Added range checking before casting to uint16_t.
  - rpmsg_env_zephyr.c: Added validation before passing values to k_msgq_init.
- Fixed a CERT INT31-C compliance issue in env_get_current_queue_size() function where an unsigned queue count was cast to a signed int32_t without proper validation, which could lead to lost or misinterpreted data if queue size exceeded INT32_MAX.
- Fixed CERT INT31-C violation in rpmsg_platform.c where memcmp() return value (signed int) was compared with unsigned constant without proper type handling.
- Fixed CERT INT31-C violation in rpmsg_platform.c where casting from uint32_t to uint16_t could potentially result in data loss. Changed length variable type from uint16_t to uint32_t to properly handle memory address differences without truncation.
- Fixed potential integer overflow in env_sleep_msec() function in ThreadX environment implementation by rearranging calculation order in the sleep duration formula.
- Fixed CERT-C INT31-C violation in RPMsg-Lite where bitwise NOT operations on integer constants were performed in signed integer context before being cast to unsigned. This could potentially lead to misinterpreted data on imx943 platform.
- Added RL_MAX_BUFFER_COUNT (32768U) and RL_MAX_VRING_ALIGN (65536U) limit to ensure alignment values cannot contribute to integer overflow
- Fixed CERT INT31-C violation in vring_need_event(), added cast to uint16_t for each operand.

v5.1.4 - 27-Mar-2025

Added

- Add KW43B43 porting layer

Changed

- Doxygen bump to version 1.9.6

v5.1.3 - 13-Jan-2025

Added

- Memory cache management of shared memory. Enable with #define RL_USE_DCACHE (1) in rpmsg_config.h in case of data cache is used.
- Cmake/Kconfig support added.
- Porting layers for imx95, imxrt700, mcmxw71x, mcmxw72x, kw47b42 added.

v5.1.2 - 08-Jul-2024

Changed

- Zephyr-related changes.
- Minor Misra corrections.

v5.1.1 - 19-Jan-2024

Added

- Test suite provided.
- Zephyr support added.

Changed

- Minor changes in platform and env. layers, minor test code updates.

v5.1.0 - 02-Aug-2023

Added

- RPMsg-Lite: Added aarch64 support.

Changed

- RPMsg-Lite: Increased the queue size to (2 * RL_BUFFER_COUNT) to cover zero copy cases.
- Code formatting using LLVM16.

Fixed

- Resolved issues in ThreadX env. layer implementation.

v5.0.0 - 19-Jan-2023

Added

- Timeout parameter added to rpmsg_lite_wait_for_link_up API function.

Changed

- Improved debug check buffers implementation - instead of checking the pointer fits into shared memory check the presence in the VirtIO ring descriptors list.
- VRING_SIZE is set based on number of used buffers now (as calculated in vring_init) - updated for all platforms that are not communicating to Linux rpmsg counterpart.

Fixed

- Fixed wrong RL_VRING_OVERHEAD macro comment in platform.h files
- Misra corrections.

v4.0.0 - 20-Jun-2022

Added

- Added support for custom shared memory arrangement per the RPMsg_Lite instance.
- Introduced new rpmsg_lite_wait_for_link_up() API function - this allows to avoid using busy loops in rtos environments, GitHub PR #21.

Changed

- Adjusted rpmsg_lite_is_link_up() to return RL_TRUE/RL_FALSE.

v3.2.0 - 17-Jan-2022

Added

- Added support for i.MX8 MP multicore platform.

Changed

- Improved static allocations - allow OS-specific objects being allocated statically, GitHub PR #14.
- Aligned rpmsg_env_xos.c and some platform layers to latest static allocation support.

Fixed

- Minor Misra and typo corrections, GitHub PR #19, #20.

v3.1.2 - 16-Jul-2021

Added

- Addressed MISRA 21.6 rule violation in rpmsg_env.h (use SDK's PRINTF in MCUXpressoSDK examples, otherwise stdio printf is used).
- Added environment layers for XOS.
- Added support for i.MX RT500, i.MX RT1160 and i.MX RT1170 multicore platforms.

Fixed

- Fixed incorrect description of the rpmsg_lite_get_endpoint_from_addr function.

Changed

- Updated RL_BUFFER_COUNT documentation (issue #10).
- Updated imxrt600_hifi4 platform layer.

v3.1.1 - 15-Jan-2021

Added

- Introduced RL_ALLOW_CONSUMED_BUFFERS_NOTIFICATION config option to allow opposite side notification sending each time received buffers are consumed and put into the queue of available buffers.
- Added environment layers for Threadx.
- Added support for i.MX8QM multicore platform.

Changed

- Several MISRA C-2012 violations addressed.

v3.1.0 - 22-Jul-2020

Added

- Added support for several new multicore platforms.

Fixed

- MISRA C-2012 violations fixed (7.4).
- Fixed missing lock in rpmsg_lite_rx_callback() for QNX env.
- Correction of rpmsg_lite_instance structure members description.
- Address -Waddress-of-packed-member warnings in GCC9.

Changed

- Clang update to v10.0.0, code re-formatted.

v3.0.0 - 20-Dec-2019

Added

- Added support for several new multicore platforms.

Fixed

- MISRA C-2012 violations fixed, incl. data types consolidation.
- Code formatted.

v2.2.0 - 20-Mar-2019

Added

- Added configuration macro RL_DEBUG_CHECK_BUFFERS.
- Several MISRA violations fixed.
- Added environment layers for QNX and Zephyr.
- Allow environment context required for some environment (controlled by the RL_USE_ENVIRONMENT_CONTEXT configuration macro).
- Data types consolidation.

v1.1.0 - 28-Apr-2017

Added

- Supporting i.MX6SX and i.MX7D MPU platforms.
- Supporting LPC5411x MCU platform.
- Baremental and FreeRTOS support.
- Support of copy and zero-copy transfer.
- Support of static API (without dynamic allocations).

Multicore Manager

MCUXpresso SDK : mcuxsdk-middleware-mcmgr (Multicore Manager)

Overview This repository is for MCUXpresso SDK Multicore Manager middleware delivery and it contains Multicore Manager component 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 for the complete delivery of MCUXpresso SDK to be able to build and run Multicore Manager examples that are based on mcux-sdk-middleware-mcmgr component.

Documentation Overall details can be reviewed here: MCUXpresso SDK Online Documentation

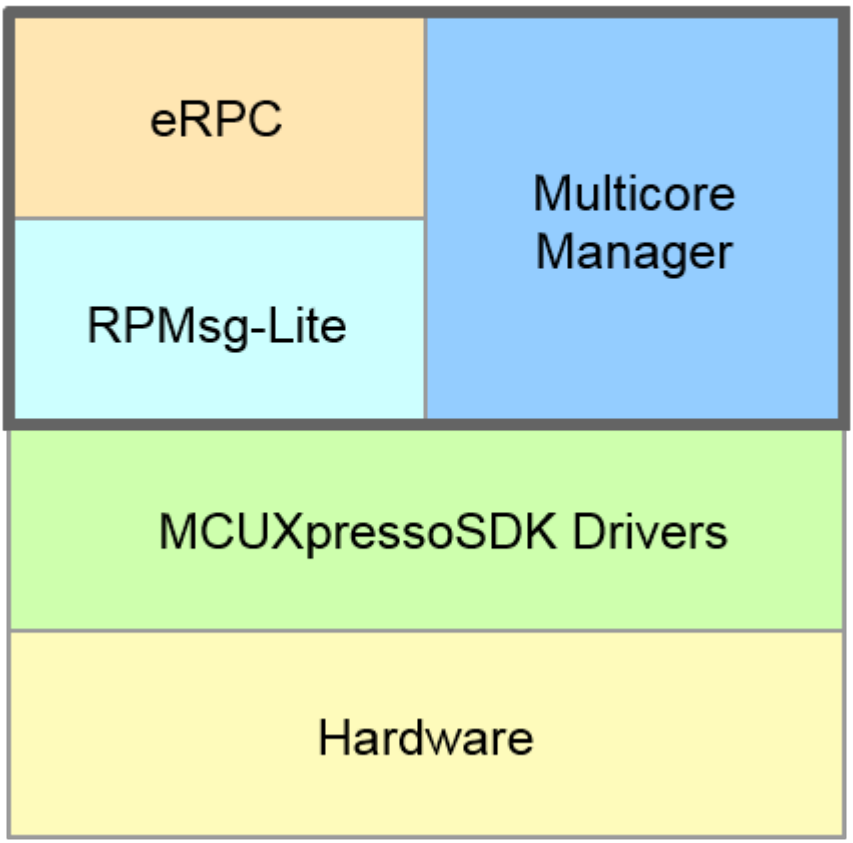
Visit Multicore Manager - Documentation to review details on the contents in this sub-repo.

Setup Instructions on how to install the MCUXpresso SDK provided from GitHub via west manifest Getting Started with SDK - Detailed Installation Instructions

Contribution We welcome and encourage the community to submit patches directly to the mcmgr project placed on github. Contributing can be managed via pull-requests. Before a pull-request is created the code should be tested and properly formatted.

Multicore Manager (MCMGR) The Multicore Manager (MCMGR) software library provides a number of services for multicore systems. This library is distributed as a part of the Multicore SDK (MCSDK). Together, the MCSDK and the MCUXpresso SDK (SDK) form a framework for development of software for NXP multicore devices.

The MCMGR component is located in the <MCUXpressoSDK_install_dir>/middleware/multicore/mcmgr directory.



The Multicore Manager provides the following major functions:

- Maintains information about all cores in system.
- Secondary/auxiliary core(s) startup and shutdown.
- Remote core monitoring and event handling.

Usage of the MCMGR software component The main use case of MCMGR is the secondary/auxiliary core start. This functionality is performed by the public API function.

Example of MCMGR usage to start secondary core:

```
#include "mcmgr.h"

void main()
```

(continues on next page)

(continued from previous page)

```
{
    /* Initialize MCMGR - low level multicore management library.
       Call this function as close to the reset entry as possible,
       (into the startup sequence) to allow CoreUp event triggering. */
    MCMGR_EarlyInit();

    /* Initialize MCMGR, install generic event handlers */
    MCMGR_Init();

    /* Boot secondary core application from the CORE1_BOOT_ADDRESS, pass "1" as startup data,␣
↪starting synchronously. */
    MCMGR_StartCore(kMCMGR_Core1, CORE1_BOOT_ADDRESS, 1, kMCMGR_Start_Synchronous);
.
.
.
    /* Stop secondary core execution. */
    MCMGR_StopCore(kMCMGR_Core1);
}
```

Some platforms allow stopping and re-starting the secondary core application again, using the MCMGR_StopCore / MCMGR_StartCore API calls. It is necessary to ensure the initially loaded image is not corrupted before re-starting, especially if it deals with the RAM target. Cache coherence has to be considered/ensured as well.

Another important MCMGR feature is the ability for remote core monitoring and handling of events such as reset, exception, and application events. Application-specific callback functions for events are registered by the MCMGR_RegisterEvent() API. Triggering these events is done using the MCMGR_TriggerEvent() API. mcmgr_event_type_t enums all possible event types.

An example of MCMGR usage for remote core monitoring and event handling. Code for the primary side:

```
#include "mcmgr.h"

#define APP_RPMSG_READY_EVENT_DATA  (1)
#define APP_NUMBER_OF_CORES (2)
#define APP_SECONDARY_CORE kMCMGR_Core1

/* Callback function registered via the MCMGR_RegisterEvent() and triggered by MCMGR_TriggerEvent()␣
↪called on the secondary core side */
void RPMsgRemoteReadyEventHandler(mcmgr_core_t coreNum, uint16_t eventData, void *context)
{
    uint16_t *data = &((uint16_t *)context)[coreNum];

    *data = eventData;
}

void main()
{
    uint16_t RPMsgRemoteReadyEventData[NUMBER_OF_CORES] = {0};

    /* Initialize MCMGR - low level multicore management library.
       Call this function as close to the reset entry as possible,
       (into the startup sequence) to allow CoreUp event triggering. */
    MCMGR_EarlyInit();

    /* Initialize MCMGR, install generic event handlers */
    MCMGR_Init();

    /* Register the application event before starting the secondary core */
    MCMGR_RegisterEvent(kMCMGR_RemoteApplicationEvent, RPMsgRemoteReadyEventHandler, (void␣
```

(continues on next page)

(continued from previous page)

```
↪*)RPMsgRemoteReadyEventData);

    /* Boot secondary core application from the CORE1_BOOT_ADDRESS, pass rpmsg_lite_base address␣
↪as startup data, starting synchronously. */
    MCMGR_StartCore(APP_SECONDARY_CORE, CORE1_BOOT_ADDRESS, (uint32_t)rpmsg_lite_
↪base, kMCMGR_Start_Synchronous);

    /* Wait until the secondary core application signals the rpmsg remote has been initialized and is ready to␣
↪communicate. */
    while(APP_RPMSG_READY_EVENT_DATA != RPMsgRemoteReadyEventData[APP_SECONDARY_
↪CORE]) {};
.
.
.
}
```

Code for the secondary side:

```
#include "mcmgr.h"

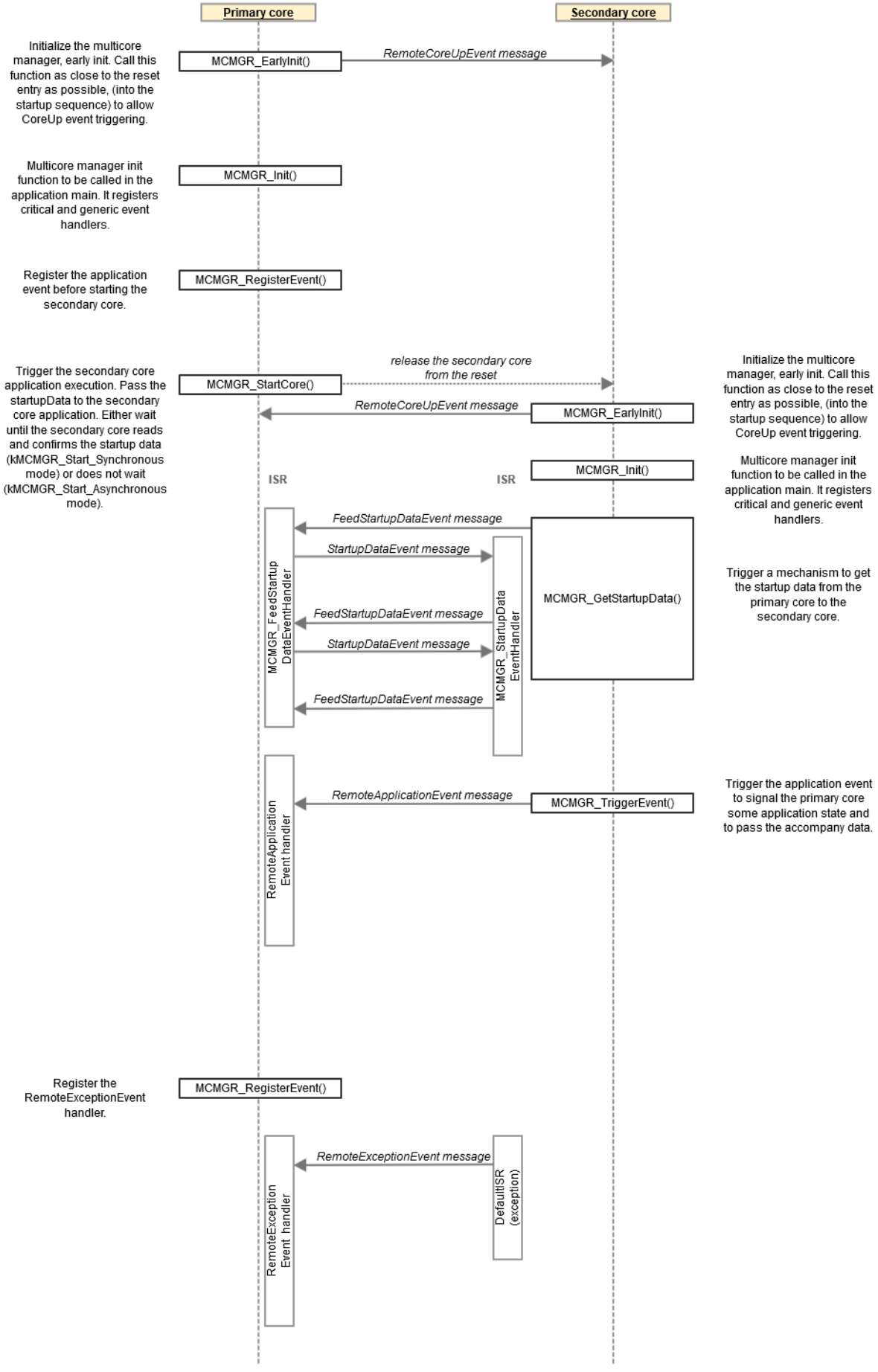
#define APP_RPMSG_READY_EVENT_DATA  (1)

void main()
{
    /* Initialize MCMGR - low level multicore management library.
       Call this function as close to the reset entry as possible,
       (into the startup sequence) to allow CoreUp event triggering. */
    MCMGR_EarlyInit();

    /* Initialize MCMGR, install generic event handlers */
    MCMGR_Init();
.
.
.

    /* Signal the to other core that we are ready by triggering the event and passing the APP_RPMSG_
↪READY_EVENT_DATA */
    MCMGR_TriggerEvent(kMCMGR_Core0, kMCMGR_RemoteApplicationEvent, APP_RPMSG_
↪READY_EVENT_DATA);
.
.
.
}
```

MCMGR Data Exchange Diagram The following picture shows how the handshakes are supposed to work between the two cores in the MCMGR software.


Primary core
Secondary core
Initialize the multicore manager, early init. Call this function as close to the reset entry as possible, (into the startup sequence) to allow CoreUp event triggering.
MCMGR_EarlyInit()
RemoteCoreUpEvent message
Multicore manager init function to be called in the application main. It registers critical and generic event handlers.
MCMGR_Init()
Register the application event before starting the secondary core.
MCMGR_RegisterEvent()
Trigger the secondary core application execution. Pass the startupData to the secondary core application. Either wait until the secondary core reads and confirms the startup data (kMCMGR_Start_Synchronous mode) or does not wait (kMCMGR_Start_Asynchronous mode).
MCMGR_StartCore()
release the secondary core from the reset
RemoteCoreUpEvent message
MCMGR_EarlyInit()
Initialize the multicore manager, early init. Call this function as close to the reset entry as possible, (into the startup sequence) to allow CoreUp event triggering.
MCMGR_Init()
Multicore manager init function to be called in the application main. It registers critical and generic event handlers.
ISR
ISR
FeedStartupDataEvent message
StartupDataEvent message
FeedStartupDataEvent message
StartupDataEvent message
FeedStartupDataEvent message
MCMGR_FeedStartup DataEventHandler
MCMGR_StartupData EventHandler
MCMGR_GetStartupData()
Trigger a mechanism to get the startup data from the primary core to the secondary core.
RemoteApplicationEvent message
MCMGR_TriggerEvent()
RemoteApplication Event handler
Trigger the application event to signal the primary core some application state and to pass the accompany data.
Register the RemoteExceptionEvent handler.
MCMGR_RegisterEvent()
RemoteExceptionEvent message
RemoteException Event handler
DefaultISR (exception)

Changelog Multicore Manager All notable changes to this project will be documented in this file.

The format is based on Keep a Changelog, and this project adheres to Semantic Versioning.

v5.0.0

Added

- Added MCMGR_BUSY_POLL_COUNT macro to prevent infinite polling loops in MCMGR operations.
- Implemented timeout mechanism for all polling loops in MCMGR code.
- Added support to handle more then two cores. Breaking API change by adding parameter coreNum specifying core number in functions bellow.
  - MCMGR_GetStartupData(uint32_t *startupData, mcmgr_core_t coreNum)
  - MCMGR_TriggerEvent(mcmgr_event_type_t type, uint16_t eventData, mcmgr_core_t coreNum)
  - MCMGR_TriggerEventForce(mcmgr_event_type_t type, uint16_t eventData, mcmgr_core_t coreNum)
  - typedef void (*mcmgr_event_callback_t)(uint16_t data, void *context, mcmgr_core_t coreNum);

  When registering the event with function MCMGR_RegisterEvent() user now needs to provide callbackData pointer to array of elements per every core in system (see README.md for example).In case of systems with only two cores the coreNum in callback can be ignored as events can arrive only from one core. Please see Porting guide for more details: Porting-GuideTo_v5.md
- Updated all porting files to support new MCMGR API.
- Added new platform specific include file mcmgr_platform.h. It will contain common platform specific macros that can be then used in mcmgr and application. e.g. platform core count MCMGR_CORECOUNT 4.
- Move all header files to new inc directory.
- Added new platform-specific include files inc/platform/<platform_name>/mcmgr_platform.h.

Added

- Add MCXL20 porting layer and unit testing

v4.1.7

Fixed

- mcmgr_stop_core_internal() function now returns kStatus_MCMGR_NotImplemented status code instead of kStatus_MCMGR_Success when device does not support stop of secondary core. Ports affected: kw32w1, kw45b41, kw45b42, mcxw716, mcxw727.

[v4.1.6]

Added

- Multicore Manager moved to standalone repository.
- Add porting layers for imxrt700, mcmxw727, kw47b42.
- New MCMGR_ProcessDeferredRxIsr() API added.

[v4.1.5]

Added

- Add notification into MCMGR_EarlyInit and mcmgr_early_init_internal functions to avoid using uninitialized data in their implementations.

[v4.1.4]

Fixed

- Avoid calling tx isr callbacks when respective Messaging Unit Transmit Interrupt Enable flag is not set in the CR/TCR register.
- Messaging Unit RX and status registers are cleared after the initialization.

[v4.1.3]

Added

- Add porting layers for imxrt1180.

Fixed

- mu_isr() updated to avoid calling tx isr callbacks when respective Transmit Interrupt Enable flag is not set in the CR/TCR register.
- mcmgr_mu_internal.c code adaptation to new supported SoCs.

[v4.1.2]

Fixed

- Update mcmgr_stop_core_internal() implementations to set core state to kMCMGR_ResetCoreState.

[v4.1.0]

Fixed

- Code adjustments to address MISRA C-2012 Rules

[v4.0.3]

Fixed

- Documentation updated to describe handshaking in a graphic form.
- Minor code adjustments based on static analysis tool findings

[v4.0.2]

Fixed

- Align porting layers to the updated MCUXpressoSDK feature files.

[v4.0.1]

Fixed

- Code formatting, removed unused code

[v4.0.0]

Added

- Add new MCMGR_TriggerEventForce() API.

[v3.0.0]

Removed

- Removed MCMGR_LoadApp(), MCMGR_MapAddress() and MCMGR_SignalReady()

Modified

- Modified MCMGR_GetStartupData()

Added

- Added MCMGR_EarlyInit(), MCMGR_RegisterEvent() and MCMGR_TriggerEvent()
- Added the ability for remote core monitoring and event handling

[v2.0.1]

Fixed

- Updated to be Misra compliant.

[v2.0.0]

Added

- Support for lpcxpresso54114 board.

[v1.1.0]

Fixed

- Ported to KSDK 2.0.0.

[v1.0.0]

Added

- Initial release.

eRPC

MCUXpresso SDK : mcuxsdk-middleware-erpc

Overview This repository is for MCUXpresso SDK eRPC middleware delivery and it contains eRPC component 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 for the complete delivery of MCUXpresso SDK to be able to build and run eRPC examples that are based on mcux-sdk-middleware-erpc component.

Documentation Overall details can be reviewed here: MCUXpresso SDK Online Documentation

Visit eRPC - 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 We welcome and encourage the community to submit patches directly to the eRPC project placed on github. Contributing can be managed via pull-requests. Before a pull-request is created the code should be tested and properly formatted.

---

eRPC

- *MCUXpresso SDK : mcuxsdk-middleware-erpc*
  - *Overview*
  - *Documentation*
  - *Setup*
  - *Contribution*
- *eRPC*

- *About*
- *Releases*
  - *Edge releases*
- *Documentation*
- *Examples*
- *References*
- *Directories*
- *Building and installing*
  - *Requirements*
    - *Windows*
    - *Mac OS X*
  - *Building*
    - *CMake and KConfig*
    - *Make*
  - *Installing for Python*
- *Known issues and limitations*
- *Code providing*

About

eRPC (Embedded RPC) is an open source Remote Procedure Call (RPC) system for multichip embedded systems and heterogeneous multicore SoCs.

Unlike other modern RPC systems, such as the excellent Apache Thrift, eRPC distinguishes itself by being designed for tightly coupled systems, using plain C for remote functions, and having a small code size (<5kB). It is not intended for high performance distributed systems over a network.

eRPC does not force upon you any particular API style. It allows you to export existing C functions, without having to change their prototypes. (There are limits, of course.) And although the internal infrastructure is written in C++, most users will be able to use only the simple C setup APIs shown in the examples below.

A code generator tool called erpcgen is included. It accepts input IDL files, having an .erpc extension, that have definitions of your data types and remote interfaces, and generates the shim code that handles serialization and invocation. erpcgen can generate either C/C++ or Python code.

Example .erpc file:

```
// Define a data type.
enum LEDName { kRed, kGreen, kBlue }

// An interface is a logical grouping of functions.
interface IO {
    // Simple function declaration with an empty reply.
    set_led(LEDName whichLed, bool onOrOff) -> void
}
```

Client side usage:

```
void example_client(void) {
    erpc_transport_t transport;
    erpc_mbf_t message_buffer_factory;
    erpc_client_t client_manager;

    /* Init eRPC client infrastructure */
    transport = erpc_transport_cmsis_uart_init(Driver_USART0);
    message_buffer_factory = erpc_mbf_dynamic_init();
    client_manager = erpc_client_init(transport, message_buffer_factory);

    /* init eRPC client IO service */
    initIO_client(client_manager);

    // Now we can call the remote function to turn on the green LED.
    set_led(kGreen, true);

    /* deinit objects */
    deinitIO_client();
    erpc_client_deinit(client_manager);
    erpc_mbf_dynamic_deinit(message_buffer_factory);
    erpc_transport_tcp_deinit(transport);
}
```

```
void example_client(void) {
    erpc_transport_t transport;
    erpc_mbf_t message_buffer_factory;
    erpc_client_t client_manager;

    /* Init eRPC client infrastructure */
    transport = erpc_transport_cmsis_uart_init(Driver_USART0);
    message_buffer_factory = erpc_mbf_dynamic_init();
    client_manager = erpc_client_init(transport, message_buffer_factory);

    /* scope for client service */
    {
        /* init eRPC client IO service */
        IO_client client(client_manager);

        // Now we can call the remote function to turn on the green LED.
        client.set_led(kGreen, true);
    }

    /* deinit objects */
    erpc_client_deinit(client_manager);
    erpc_mbf_dynamic_deinit(message_buffer_factory);
    erpc_transport_tcp_deinit(transport);
}
```

Server side usage:

```
// Implement the remote function.
void set_led(LEDName whichLed, bool onOrOff) {
    // implementation goes here
}

void example_server(void) {
    erpc_transport_t transport;
    erpc_mbf_t message_buffer_factory;
    erpc_server_t server;
    erpc_service_t service = create_IO_service();
```

(continues on next page)

(continued from previous page)

```
    /* Init eRPC server infrastructure */
    transport = erpc_transport_cmsis_uart_init(Driver_USART0);
    message_buffer_factory = erpc_mbf_dynamic_init();
    server = erpc_server_init(transport, message_buffer_factory);

    /* add custom service implementation to the server */
    erpc_add_service_to_server(server, service);

    // Run the server.
    erpc_server_run();

    /* deinit objects */
    destroy_IO_service(service);
    erpc_server_deinit(server);
    erpc_mbf_dynamic_deinit(message_buffer_factory);
    erpc_transport_tcp_deinit(transport);
}
```

```
// Implement the remote function.
class IO : public IO_interface
{
    /* eRPC call definition */
    void set_led(LEDName whichLed, bool onOrOff) override {
        // implementation goes here
    }
}

void example_server(void) {
    erpc_transport_t transport;
    erpc_mbf_t message_buffer_factory;
    erpc_server_t server;
    IO IOImpl;
    IO_service io(&IOImpl);

    /* Init eRPC server infrastructure */
    transport = erpc_transport_cmsis_uart_init(Driver_USART0);
    message_buffer_factory = erpc_mbf_dynamic_init();
    server = erpc_server_init(transport, message_buffer_factory);

    /* add custom service implementation to the server */
    erpc_add_service_to_server(server, &io);

    /* poll for requests */
    erpc_status_t err = server.run();

    /* deinit objects */
    erpc_server_deinit(server);
    erpc_mbf_dynamic_deinit(message_buffer_factory);
    erpc_transport_tcp_deinit(transport);
}
```

A number of transports are supported, and new transport classes are easy to write.

Supported transports can be found in *erpc/erpc_c/transport* folder. E.g:

- CMSIS UART
- NXP Kinetis SPI and DSPI
- POSIX and Windows serial port
- TCP/IP (mostly for testing)

- NXP RPMsg-Lite / RPMsg TTY
- SPIdev Linux
- USB CDC
- NXP Messaging Unit

eRPC is available with an unrestrictive BSD 3-clause license. See the LICENSE file for the full license text.

Releases eRPC releases

Edge releases Edge releases can by found on eRPC CircleCI webpage. Choose build of interest, then platform target and choose ARTIFACTS tab. Here you can find binary application from chosen build.

Documentation Documentation is in the wiki section.

eRPC Infrastructure documentation

Examples *Example IDL* is available in the *examples/* folder.

Plenty of eRPC multicore and multiprocessor examples can be also found in NXP MCUXpressoSDK packages. Visit https://mcuxpresso.nxp.com to configure, build and download these packages.

To get the board list with multicore support (eRPC included) use filtering based on Middleware and search for 'multicore' string. Once the selected package with the multicore middleware is downloaded, see

<MCUXpressoSDK_install_dir>/boards/<board_name>/multicore_examples for eRPC multicore examples (RPMsg_Lite or Messaging Unit transports used) or

<MCUXpressoSDK_install_dir>/boards/<board_name>/multiprocessor_examples for eRPC multiprocessor examples (UART or SPI transports used).

eRPC examples use the 'erpc_' name prefix.

Another way of getting NXP MCUXpressoSDK eRPC multicore and multiprocessor examples is using the mcux-sdk Github repo. Follow the description how to use the West tool to clone and update the mcuxsdk repo in readme Overview section. Once done the armgcc eRPC examples can be found in

mcuxsdk/examples/<board_name>/multicore_examples or in

mcuxsdk/examples/<board_name>/multiprocessor_examples folders.

You can use the evkmimxrt1170 as the board_name for instance. Similar to MCUXpressoSDK packages the eRPC examples use the 'erpc_' name prefix.

References This section provides links to interesting erpc-based projects, articles, blogs or guides:

- erpc (EmbeddedRPC) getting started notes
- ERPC Linux Local Environment Construction and Use
- The New Wio Terminal eRPC Firmware

Directories *doc* - Documentation.

doxygen - Configuration and support files for running Doxygen over the eRPC C++ infrastructure and erpcgen code.

erpc_c - Holds C/C++ infrastructure for eRPC. This is the code you will include in your application.

erpc_python - Holds Python version of the eRPC infrastructure.

erpcgen - Holds source code for erpcgen and makefiles or project files to build erpcgen on Windows, Linux, and OS X.

erpcsniffer - Holds source code for erpcsniffer application.

examples - Several example IDL files.

mk - Contains common makefiles for building eRPC components.

test - Client/server tests. These tests verify the entire communications path from client to server and back.

utilities - Holds utilities which bring additional benefit to eRPC apps developers.

Building and installing These build instructions apply to host PCs and embedded Linux. For bare metal or RTOS embedded environments, you should copy the *erpc_c* directory into your application sources.

CMake and KConfig build:

It builds a static library of the eRPC C/C++ infrastructure, the erpcgen executable, and optionally the unit tests and examples.

CMake is compatible with gcc and clang. On Windows local MingGW downloaded by *script* can be used.

Make build:

It builds a static library of the eRPC C/C++ infrastructure, the erpcgen executable, and optionally the unit tests.

The makefiles are compatible with gcc or clang on Linux, OS X, and Cygwin. A Windows build of erpcgen using Visual Studio is also available in the *erpcgen/VisualStudio_v14* directory. There is also an Xcode project file in the *erpcgen* directory, which can be used to build erpcgen for OS X.

Requirements eRPC now support building **erpcgen**, **erpc_lib**, **tests** and **C examples** using CMake.

Requirements when using CMake:

- **CMake** (minimal version 3.20.0)
- Generator - **Make**, **Ninja**, ...
- **C/C++ compiler** - **GCC**, **CLANG**, ...
- **Binson** - https://www.gnu.org/software/bison/
- **Flex** - https://github.com/westes/flex/

Requirements when using Make:

- **Make**
- **C/C++ compiler** - **GCC**, **CLANG**, ...
- **Binson** - https://www.gnu.org/software/bison/
- **Flex** - https://github.com/westes/flex/

Windows Related steps to build **erpcgen** using **Visual Studio** are described in erpcgen/VisualStudio_v14/readme_erpcgen.txt.

To install MinGW, Bison, Flex locally on Windows:

```
./install_dependencies.ps1
* ```

#### Linux

```bash
./install_dependencies.sh
```

Mandatory for case, when build for different architecture is needed

- gcc-multilib, g++-multilib

Mac OS X

```
./install_dependencies.sh
```

Building

CMake and KConfig eRPC use CMake and KConfig to configurate and build eRPC related targets. KConfig can be edited by *prj.conf* or *menuconfig* when building.

Generate project, config and build. In *erpc/* execute:

```
cmake -B ./build # in erpc/build generate cmake project
cmake --build ./build --target menuconfig # Build menuconfig and configurate erpcgen, erpc_lib, tests and␣
↪examples
cmake --build ./build # Build all selected target from prj.conf/menuconfig
```

**CMake will use the system's default compilers and generator

If you want to use Windows and locally installed MinGW, use *CMake preset* :

```
cmake --preset mingw64 # Generate project in ./build using mingw64's make and compilers
cmake --build ./build --target menuconfig # Build menuconfig and configurate erpcgen, erpc_lib, tests and␣
↪examples
cmake --build ./build # Build all selected target from prj.conf/menuconfig
```

Make To build the library and erpcgen, run from the repo root directory:

```
make
```

To install the library, erpcgen, and include files, run:

```
make install
```

You may need to sudo the make install.

By default this will install into /usr/local. If you want to install elsewhere, set the PREFIX environment variable. Example for installing into /opt:

```
make install PREFIX=/opt
```

List of top level Makefile targets:

- erpc: build the liberpc.a static library
```

- erpcgen: build the erpcgen tool
- erpcsniffer: build the sniffer tool
- test: build the unit tests under the *test* directory
- all: build all of the above
- install: install liberpc.a, erpcgen, and include files

eRPC code is validated with respect to the C++ 11 standard.

Installing for Python To install the Python infrastructure for eRPC see instructions in the *erpc python readme*.

Known issues and limitations

- Static allocations controlled by the ERPC_ALLOCATION_POLICY config macro are not fully supported yet, i.e. not all erpc objects can be allocated statically now. It deals with the ongoing process and the full static allocations support will be added in the future.

Code providing Repository on Github contains two main branches: **main** and **develop**. Code is developed on **develop** branch. Release version is created via merging **develop** branch into **main** branch.

Copyright 2014-2016 Freescale Semiconductor, Inc.

Copyright 2016-2025 NXP

eRPC Getting Started

Overview This *Getting Started User Guide* shows software developers how to use Remote Procedure Calls (RPC) in embedded multicore microcontrollers (eRPC).

The eRPC documentation is located in the *<MCUXpressoSDK_install_dir>/ middleware/multicore/erpc/doc* folder.

Create an eRPC application This section describes a generic way to create a client/server eRPC application:

1. **Design the eRPC application:** Decide which data types are sent between applications, and define functions that send/receive this data.
2. **Create the IDL file:** The IDL file contains information about data types and functions used in an eRPC application, and is written in the IDL language.
3. **Use the eRPC generator tool:** This tool takes an IDL file and generates the shim code for the client and the server-side applications.
4. **Create an eRPC application:**
    1. Create two projects, where one project is for the client side (primary core) and the other project is for the server side (secondary core).
    2. Add generated files for the client application to the client project, and add generated files for the server application to the server project.
    3. Add infrastructure files.

4. Add user code for client and server applications.
5. Set the client and server project options.

5. **Run the eRPC application:** Run both the server and the client applications. Make sure that the server has been run before the client request was sent.

A specific example follows in the next section.

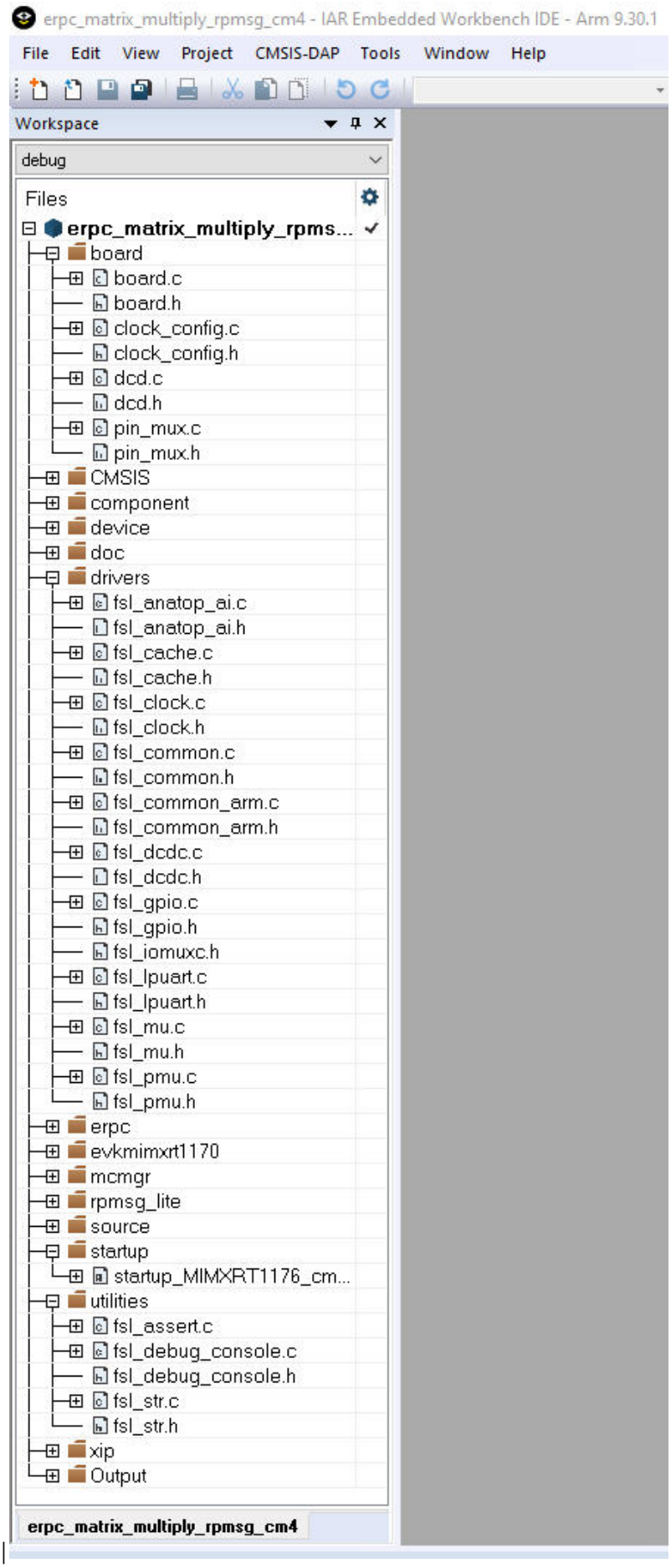
Multicore server application The "Matrix multiply" eRPC server project is located in the following folder:

<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_matrix_multiply_rpmsg/cm4/iar

The project files for the eRPC server have the _cm4 suffix.

Server project basic source files The startup files, board-related settings, peripheral drivers, and utilities belong to the basic project source files and form the skeleton of all MCUXpresso SDK applications. These source files are located in:

- *<MCUXpressoSDK_install_dir>/devices/<device>*
- *<MCUXpressoSDK_install_dir>/boards/<board_name>/multicore_examples/<example_name>/*


erpc_matrix_multiply_rpmsg_cm4 - IAR Embedded Workbench IDE - Arm 9.30.1
File Edit View Project CMSIS-DAP Tools Window Help
Workspace
debug
Files
erpc_matrix_multiply_rpms...
board
board.c
board.h
clock_config.c
clock_config.h
dcd.c
dcd.h
pin_mux.c
pin_mux.h
CMSIS
component
device
doc
drivers
fsl_anatop_ai.c
fsl_anatop_ai.h
fsl_cache.c
fsl_cache.h
fsl_clock.c
fsl_clock.h
fsl_common.c
fsl_common.h
fsl_common_arm.c
fsl_common_arm.h
fsl_dcdc.c
fsl_dcdc.h
fsl_gpio.c
fsl_gpio.h
fsl_iomuxc.h
fsl_lpuart.c
fsl_lpuart.h
fsl_mu.c
fsl_mu.h
fsl_pmu.c
fsl_pmu.h
erpc
evkmimxrt1170
mcmgr
rpmsg_lite
source
startup
startup_MIMXRT1176_cm...
utilities
fsl_assert.c
fsl_debug_console.c
fsl_debug_console.h
fsl_str.c
fsl_str.h
xip
Output
erpc_matrix_multiply_rpmsg_cm4

|

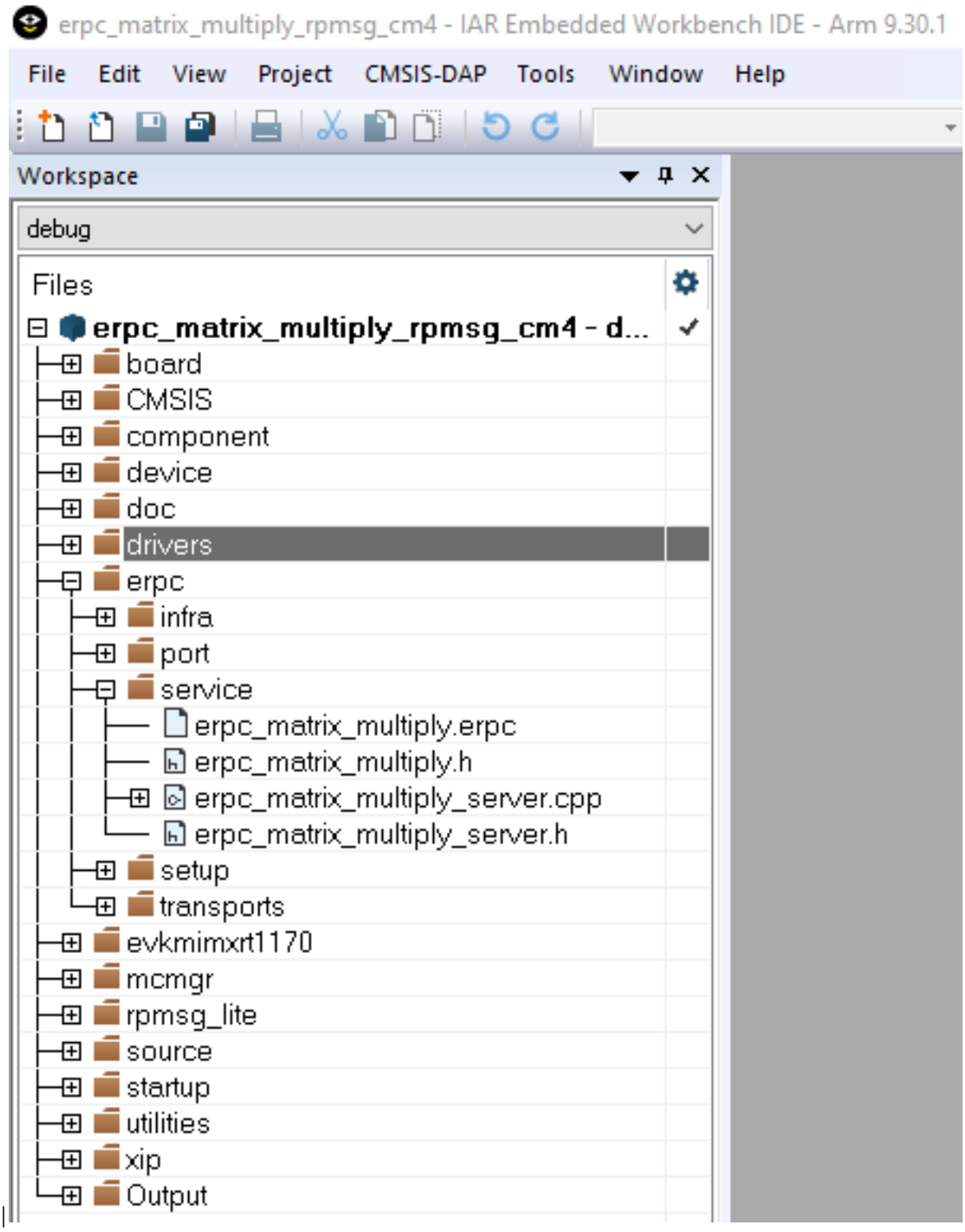
Parent topic: Multicore server application

Server related generated files The server-related generated files are:

- erpc_matric_multiply.h
- erpc_matrix_multiply_server.h
- erpc_matrix_multiply_server.cpp

The server-related generated files contain the shim code for functions and data types declared in the IDL file. These files also contain functions for the identification of client requested functions, data deserialization, calling requested function's implementations, and data serialization and return, if requested by the client. These shim code files can be found in the following folder:

<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_common/erpc_matrix_multiply/s



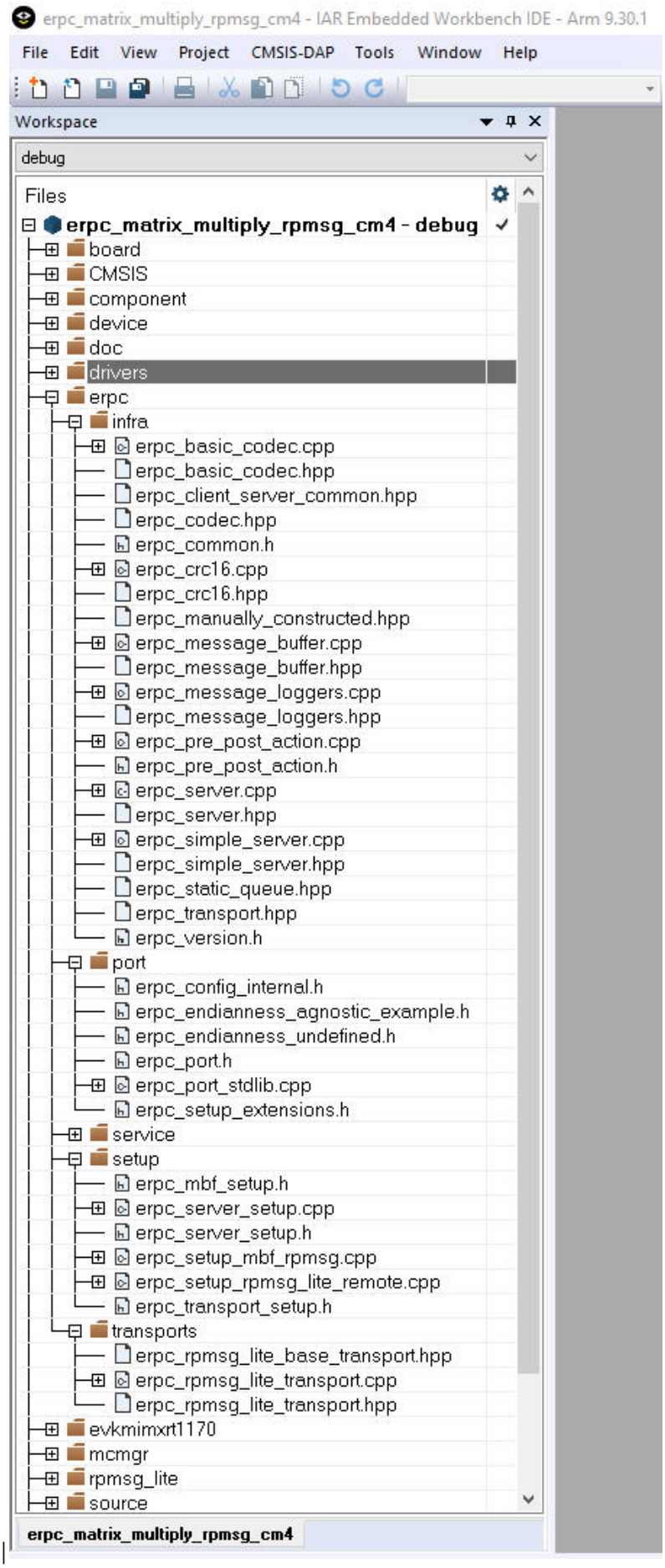
Parent topic: Multicore server application

Server infrastructure files The eRPC infrastructure files are located in the following folder:

<MCUXpressoSDK_install_dir>/middleware/multicore/erpc/erpc_c

The **erpc_c** folder contains files for creating eRPC client and server applications in the *C/C++* language. These files are distributed into subfolders.

- The **infra** subfolder contains C++ infrastructure code used to build server and client applications.
  - Four files, erpc_server.hpp, erpc_server.cpp, erpc_simple_server.hpp, and erpc_simple_server.cpp, are used for running the eRPC server on the server-side applications. The simple server is currently the only implementation of the server, and its role is to catch client requests, identify and call requested functions, and send data back when requested.
  - Three files (erpc_codec.hpp, erpc_basic_codec.hpp, and erpc_basic_codec.cpp) are used for codecs. Currently, the basic codec is the initial and only implementation of the codecs.
  - The erpc_common.hpp file is used for common eRPC definitions, typedefs, and enums.
  - The erpc_manually_constructed.hpp file is used for allocating static storage for the used objects.
  - Message buffer files are used for storing serialized data: erpc_message_buffer.h and erpc_message_buffer.cpp.
  - The erpc_transport.h file defines the abstract interface for transport layer.
- The **port** subfolder contains the eRPC porting layer to adapt to different environments.
  - erpc_port.h file contains definition of erpc_malloc() and erpc_free() functions.
  - erpc_port_stdlib.cpp file ensures adaptation to stdlib.
  - erpc_config_internal.h internal erpc configuration file.
- The **setup** subfolder contains a set of plain C APIs that wrap the C++ infrastructure, providing client and server init and deinit routines that greatly simplify eRPC usage in C-based projects. No knowledge of C++ is required to use these APIs.
  - The erpc_server_setup.h and erpc_server_setup.cpp files needs to be added into the "Matrix multiply" example project to demonstrate the use of C-wrapped functions in this example.
  - The erpc_transport_setup.h and erpc_setup_rpmsg_lite_remote.cpp files needs to be added into the project in order to allow the C-wrapped function for transport layer setup.
  - The erpc_mbf_setup.h and erpc_setup_mbf_rpmsg.cpp files needs to be added into the project in order to allow message buffer factory usage.
- The **transports** subfolder contains transport classes for the different methods of communication supported by eRPC. Some transports are applicable only to host PCs, while others are applicable only to embedded or multicore systems. Most transports have corresponding client and server setup functions in the setup folder.
  - RPMsg-Lite is used as the transport layer for the communication between cores, erpc_rpmsg_lite_base_transport.hpp, erpc_rpmsg_lite_transport.hpp, and erpc_rpmsg_lite_transport.cpp files need to be added into the server project.


erpc_matrix_multiply_rpmsg_cm4 - IAR Embedded Workbench IDE - Arm 9.30.1
File Edit View Project CMSIS-DAP Tools Window Help
Workspace
debug
Files
erpc_matrix_multiply_rpmsg_cm4 - debug
board
CMSIS
component
device
doc
drivers
erpc
infra
erpc_basic_codec.cpp
erpc_basic_codec.hpp
erpc_client_server_common.hpp
erpc_codec.hpp
erpc_common.h
erpc_crc16.cpp
erpc_crc16.hpp
erpc_manually_constructed.hpp
erpc_message_buffer.cpp
erpc_message_buffer.hpp
erpc_message_loggers.cpp
erpc_message_loggers.hpp
erpc_pre_post_action.cpp
erpc_pre_post_action.h
erpc_server.cpp
erpc_server.hpp
erpc_simple_server.cpp
erpc_simple_server.hpp
erpc_static_queue.hpp
erpc_transport.hpp
erpc_version.h
port
erpc_config_internal.h
erpc_endianness_agnostic_example.h
erpc_endianness_undefined.h
erpc_port.h
erpc_port_stdlib.cpp
erpc_setup_extensions.h
service
setup
erpc_mbf_setup.h
erpc_server_setup.cpp
erpc_server_setup.h
erpc_setup_mbf_rpmsg.cpp
erpc_setup_rpmsg_lite_remote.cpp
erpc_transport_setup.h
transports
erpc_rpmsg_lite_base_transport.hpp
erpc_rpmsg_lite_transport.cpp
erpc_rpmsg_lite_transport.hpp
evkmimxrt1170
mcmgr
rpmsg_lite
source
erpc_matrix_multiply_rpmsg_cm4

|

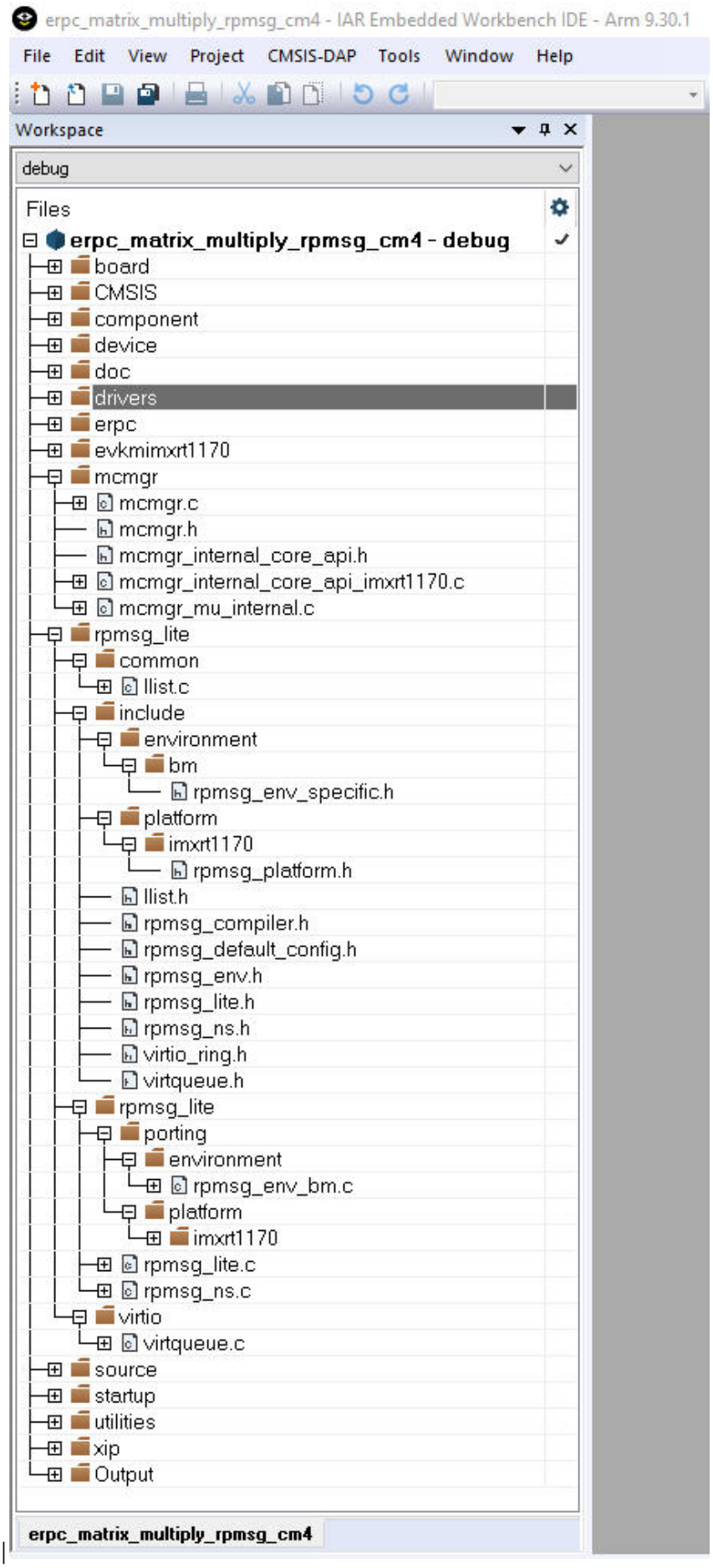
Parent topic: Multicore server application

Server multicore infrastructure files Because of the RPMsg-Lite (transport layer), it is also necessary to include RPMsg-Lite related files, which are in the following folder:

<MCUXpressoSDK_install_dir>/middleware/multicore/rpmsg_lite/

The multicore example applications also use the Multicore Manager software library to control the secondary core startup and shutdown. These source files are located in the following folder:

<MCUXpressoSDK_install_dir>/middleware/multicore/mcmgr/


erpc_matrix_multiply_rpmsg_cm4 - IAR Embedded Workbench IDE - Arm 9.30.1
File Edit View Project CMSIS-DAP Tools Window Help
Workspace
debug
Files
erpc_matrix_multiply_rpmsg_cm4 - debug
board
CMSIS
component
device
doc
drivers
erpc
evkmimxrt1170
mcmgr
mcmgr.c
mcmgr.h
mcmgr_internal_core_api.h
mcmgr_internal_core_api_imxrt1170.c
mcmgr_mu_internal.c
rpmsg_lite
common
llist.c
include
environment
bm
rpmsg_env_specific.h
platform
imxrt1170
rpmsg_platform.h
llist.h
rpmsg_compiler.h
rpmsg_default_config.h
rpmsg_env.h
rpmsg_lite.h
rpmsg_ns.h
virtio_ring.h
virtqueue.h
rpmsg_lite
porting
environment
rpmsg_env_bm.c
platform
imxrt1170
rpmsg_lite.c
rpmsg_ns.c
virtio
virtqueue.c
source
startup
utilities
xip
Output
erpc_matrix_multiply_rpmsg_cm4

|

Parent topic: Multicore server application

Server user code The server's user code is stored in the main_core1.c file, located in the following folder:

<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_matrix_multiply_rpmsg/cm4

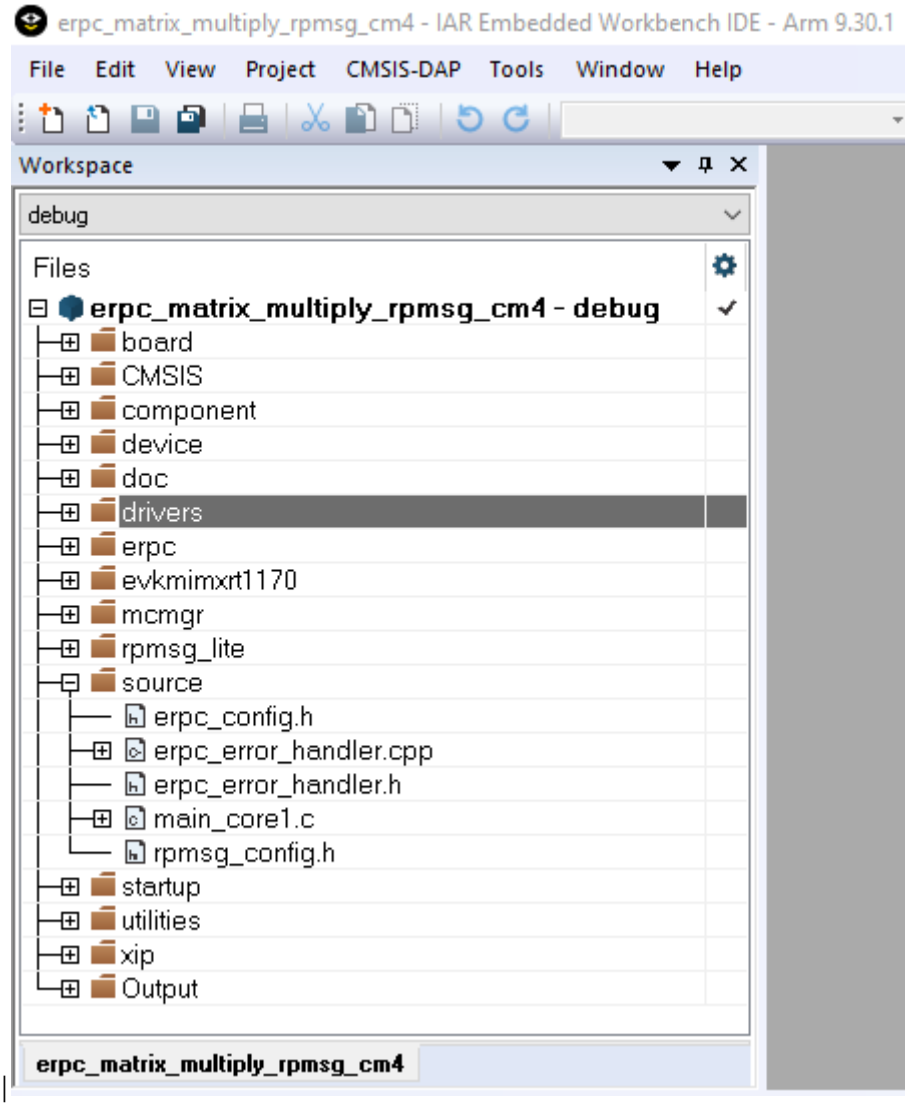
The main_core1.c file contains two functions:

- The **main()** function contains the code for the target board and eRPC server initialization. After the initialization, the matrix multiply service is added and the eRPC server waits for client's requests in the while loop.
- The **erpcMatrixMultiply()** function is the user implementation of the eRPC function defined in the IDL file.
- There is the possibility to write the application-specific eRPC error handler. The eRPC error handler of the matrix multiply application is implemented in the erpc_error_handler.h and erpc_error_handler.cpp files.

The eRPC-relevant code is captured in the following code snippet:

```
/* erpcMatrixMultiply function user implementation */
void erpcMatrixMultiply(const Matrix *matrix1, const Matrix *matrix2, Matrix *result_matrix)
{
  ...
}
int main()
{
  ...
  /* RPMsg-Lite transport layer initialization */
  erpc_transport_t transport;
  transport = erpc_transport_rpmsg_lite_remote_init(src, dst, (void*)startupData,
  ERPC_TRANSPORT_RPMSG_LITE_LINK_ID, SignalReady, NULL);
  ...
  /* MessageBufferFactory initialization */
  erpc_mbf_t message_buffer_factory;
  message_buffer_factory = erpc_mbf_rpmsg_init(transport);
  ...
  /* eRPC server side initialization */
  erpc_server_t server;
  server = erpc_server_init(transport, message_buffer_factory);
  ...
  /* Adding the service to the server */
  erpc_service_t service = create_MatrixMultiplyService_service();
  erpc_add_service_to_server(server, service);
  ...
  while (1)
  {
    /* Process eRPC requests */
    erpc_status_t status = erpc_server_poll(server);
    /* handle error status */
    if (status != kErpcStatus_Success)
    {
      /* print error description */
      erpc_error_handler(status, 0);
      ...
    }
    ...
  }
}
```

Except for the application main file, there are configuration files for the RPMsg-Lite (rpmsg_config.h) and eRPC (erpc_config.h), located in the *<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_matrix_multiply_rpmsg* folder.



Parent topic: Multicore server application

Parent topic:*Create an eRPC application*

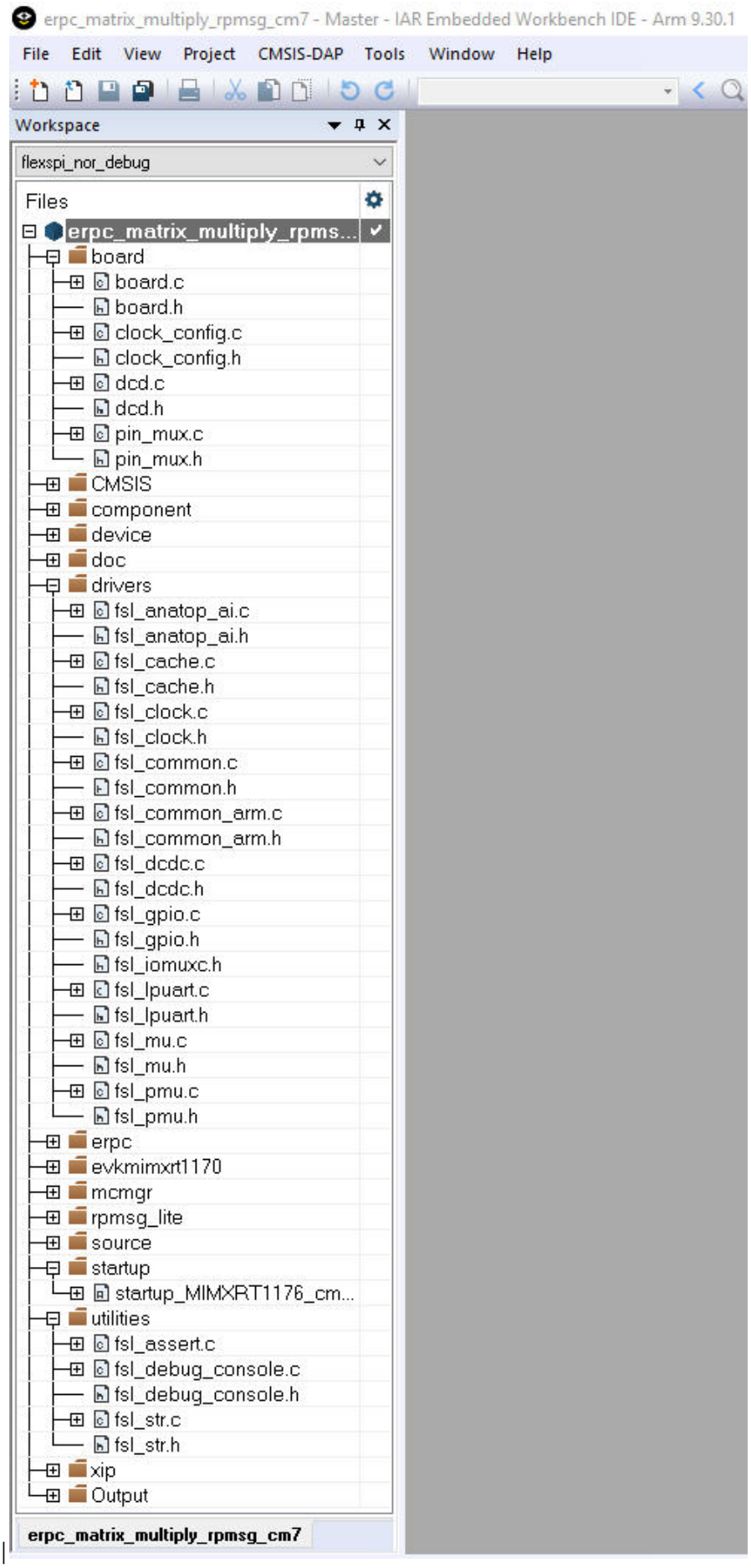
Multicore client application The "Matrix multiply" eRPC client project is located in the following folder:

<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_matrix_multiply_rpmsg/cm7/iar

Project files for the eRPC client have the _cm7 suffix.

Client project basic source files The startup files, board-related settings, peripheral drivers, and utilities belong to the basic project source files and form the skeleton of all MCUXpresso SDK applications. These source files are located in the following folders:

- *<MCUXpressoSDK_install_dir>/devices/<device>*
- *<MCUXpressoSDK_install_dir>/boards/<board_name>/multicore_examples/<example_name>/*


erpc_matrix_multiply_rpmsg_cm7 - Master - IAR Embedded Workbench IDE - Arm 9.30.1
File Edit View Project CMSIS-DAP Tools Window Help
Workspace
flexspi_nor_debug
Files
erpc_matrix_multiply_rpms...
board
board.c
board.h
clock_config.c
clock_config.h
dcd.c
dcd.h
pin_mux.c
pin_mux.h
CMSIS
component
device
doc
drivers
fsl_anatop_ai.c
fsl_anatop_ai.h
fsl_cache.c
fsl_cache.h
fsl_clock.c
fsl_clock.h
fsl_common.c
fsl_common.h
fsl_common_arm.c
fsl_common_arm.h
fsl_dcdc.c
fsl_dcdc.h
fsl_gpio.c
fsl_gpio.h
fsl_iomuxc.h
fsl_lpuart.c
fsl_lpuart.h
fsl_mu.c
fsl_mu.h
fsl_pmu.c
fsl_pmu.h
erpc
evkmimxrt1170
mcmgr
rpmsg_lite
source
startup
startup_MIMXRT1176_cm...
utilities
fsl_assert.c
fsl_debug_console.c
fsl_debug_console.h
fsl_str.c
fsl_str.h
xip
Output
erpc_matrix_multiply_rpmsg_cm7

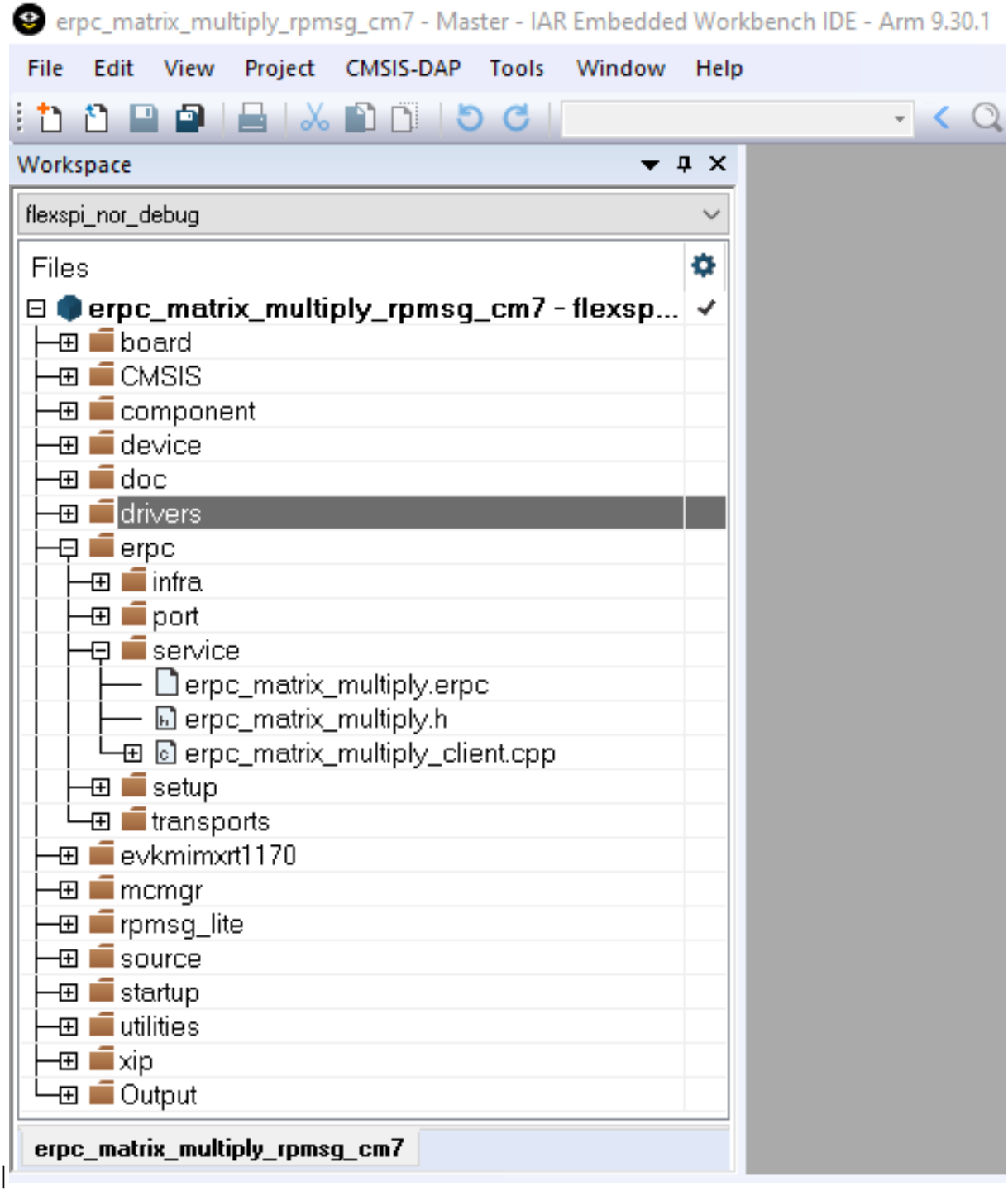
|

Parent topic: Multicore client application

Client-related generated files The client-related generated files are:

- erpc_matric_multiply.h
- erpc_matrix_multiply_client.cpp

These files contain the shim code for the functions and data types declared in the IDL file. These functions also call methods for codec initialization, data serialization, performing eRPC requests, and de-serializing outputs into expected data structures (if return values are expected). These shim code files can be found in the *<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_common/erpc_matrix_multiply/service/* folder.



Parent topic: Multicore client application

Client infrastructure files The eRPC infrastructure files are located in the following folder:

<MCUXpressoSDK_install_dir>/middleware/multicore/erpc/erpc_c

The **erpc_c** folder contains files for creating eRPC client and server applications in the C/C++ language. These files are distributed into subfolders.

- The **infra** subfolder contains C++ infrastructure code used to build server and client applications.

- Two files, erpc_client_manager.h and erpc_client_manager.cpp, are used for managing the client-side application. The main purpose of the client files is to create, perform, and release eRPC requests.
- Three files (erpc_codec.hpp, erpc_basic_codec.hpp, and erpc_basic_codec.cpp) are used for codecs. Currently, the basic codec is the initial and only implementation of the codecs.
- erpc_common.h file is used for common eRPC definitions, typedefs, and enums.
- erpc_manually_constructed.hpp file is used for allocating static storage for the used objects.
- Message buffer files are used for storing serialized data: erpc_message_buffer.hpp and erpc_message_buffer.cpp.
- erpc_transport.hpp file defines the abstract interface for transport layer.

The **port** subfolder contains the eRPC porting layer to adapt to different environments.

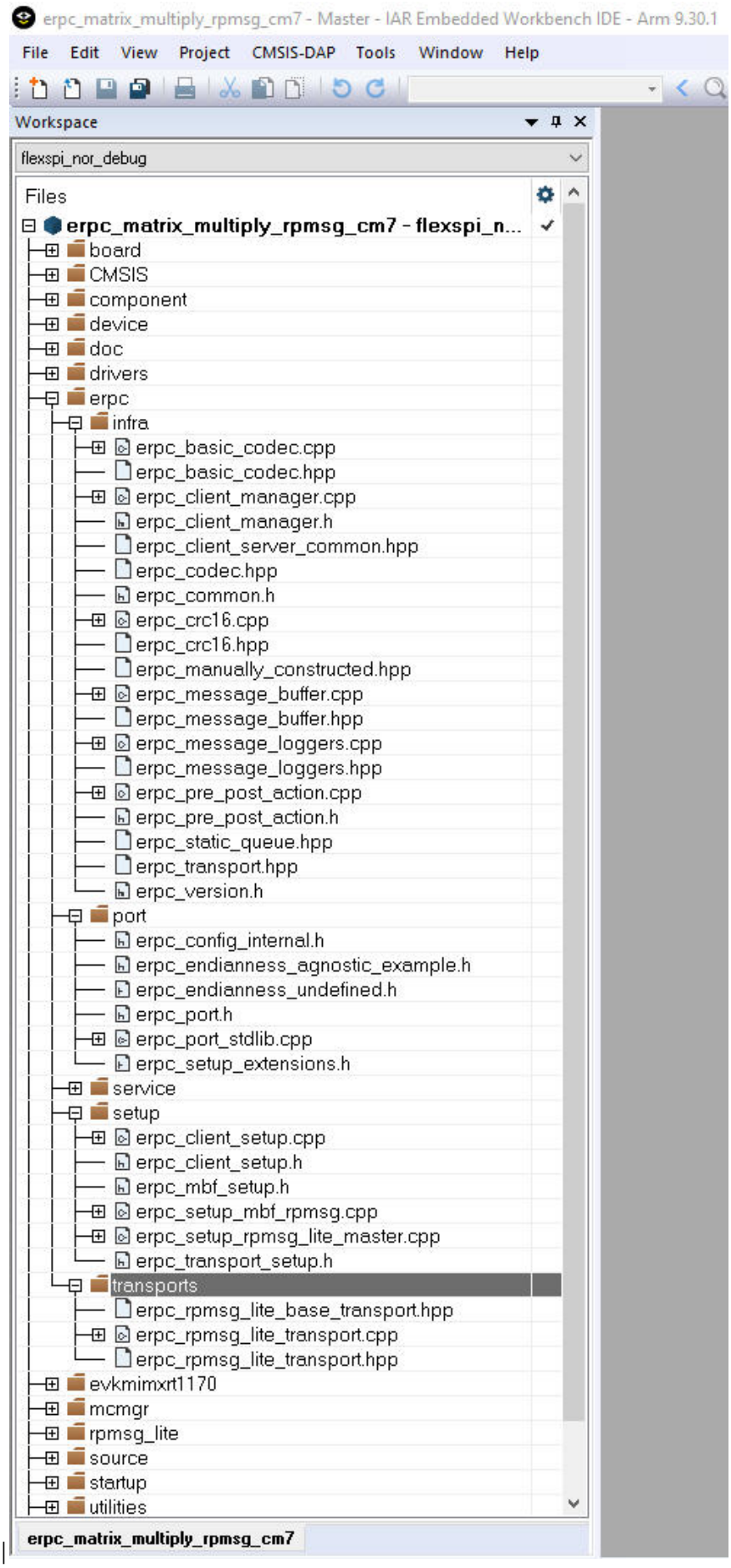
- erpc_port.h file contains definition of erpc_malloc() and erpc_free() functions.
- erpc_port_stdlib.cpp file ensures adaptation to stdlib.
- erpc_config_internal.h internal eRPC configuration file.

The **setup** subfolder contains a set of plain C APIs that wrap the C++ infrastructure, providing client and server init and deinit routines that greatly simplify eRPC usage in C-based projects. No knowledge of C++ is required to use these APIs.

- erpc_client_setup.h and erpc_client_setup.cpp files needs to be added into the "Matrix multiply" example project to demonstrate the use of C-wrapped functions in this example.
- erpc_transport_setup.h and erpc_setup_rpmsg_lite_master.cpp files needs to be added into the project in order to allow C-wrapped function for transport layer setup.
- erpc_mbf_setup.h and erpc_setup_mbf_rpmsg.cpp files needs to be added into the project in order to allow message buffer factory usage.

The **transports** subfolder contains transport classes for the different methods of communication supported by eRPC. Some transports are applicable only to host PCs, while others are applicable only to embedded or multicore systems. Most transports have corresponding client and server setup functions, in the setup folder.

- RPMsg-Lite is used as the transport layer for the communication between cores, erpc_rpmsg_lite_base_transport.hpp, erpc_rpmsg_lite_transport.hpp, and erpc_rpmsg_lite_transport.cpp files needs to be added into the client project.


erpc_matrix_multiply_rpmsg_cm7 - Master - IAR Embedded Workbench IDE - Arm 9.30.1
File Edit View Project CMSIS-DAP Tools Window Help
Workspace
flexspi_nor_debug
Files
erpc_matrix_multiply_rpmsg_cm7 - flexspi_n...
board
CMSIS
component
device
doc
drivers
erpc
infra
erpc_basic_codec.cpp
erpc_basic_codec.hpp
erpc_client_manager.cpp
erpc_client_manager.h
erpc_client_server_common.hpp
erpc_codec.hpp
erpc_common.h
erpc_crc16.cpp
erpc_crc16.hpp
erpc_manually_constructed.hpp
erpc_message_buffer.cpp
erpc_message_buffer.hpp
erpc_message_loggers.cpp
erpc_message_loggers.hpp
erpc_pre_post_action.cpp
erpc_pre_post_action.h
erpc_static_queue.hpp
erpc_transport.hpp
erpc_version.h
port
erpc_config_internal.h
erpc_endianness_agnostic_example.h
erpc_endianness_undefined.h
erpc_port.h
erpc_port_stdlib.cpp
erpc_setup_extensions.h
service
setup
erpc_client_setup.cpp
erpc_client_setup.h
erpc_mbf_setup.h
erpc_setup_mbf_rpmsg.cpp
erpc_setup_rpmsg_lite_master.cpp
erpc_transport_setup.h
transports
erpc_rpmsg_lite_base_transport.hpp
erpc_rpmsg_lite_transport.cpp
erpc_rpmsg_lite_transport.hpp
evkmimxrt1170
mcmgr
rpmsg_lite
source
startup
utilities
erpc_matrix_multiply_rpmsg_cm7

|

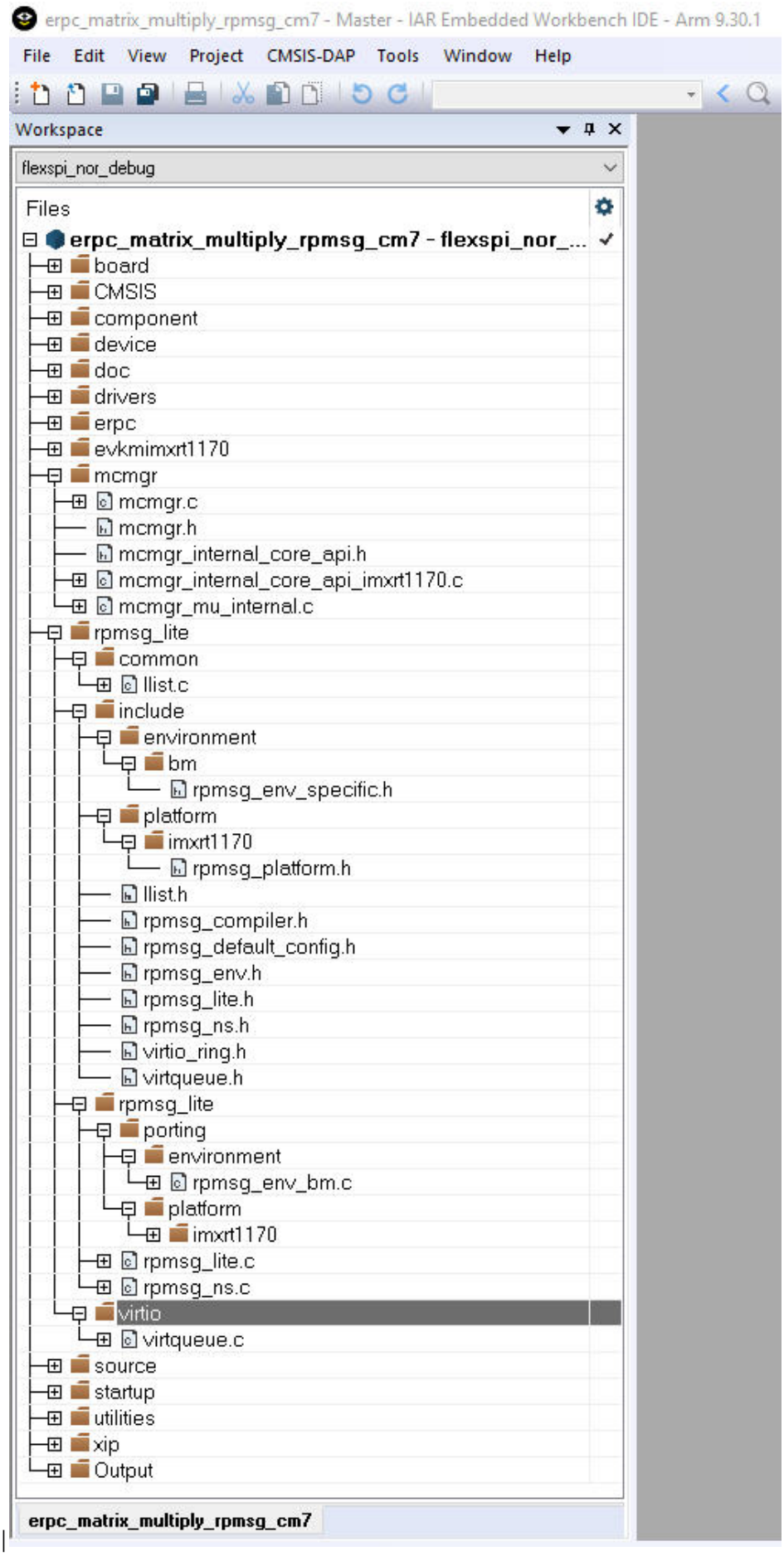
Parent topic: Multicore client application

Client multicore infrastructure files Because of the RPMsg-Lite (transport layer), it is also necessary to include RPMsg-Lite related files, which are in the following folder:

<MCUXpressoSDK_install_dir>/middleware/multicore/rpmsg_lite/

The multicore example applications also use the Multicore Manager software library to control the secondary core startup and shutdown. These source files are located in the following folder:

<MCUXpressoSDK_install_dir>/middleware/multicore/mcmgr/


erpc_matrix_multiply_rpmsg_cm7 - Master - IAR Embedded Workbench IDE - Arm 9.30.1
File Edit View Project CMSIS-DAP Tools Window Help
Workspace
flexspi_nor_debug
Files
erpc_matrix_multiply_rpmsg_cm7 - flexspi_nor_...
board
CMSIS
component
device
doc
drivers
erpc
evkmimxrt1170
mcmgr
mcmgr.c
mcmgr.h
mcmgr_internal_core_api.h
mcmgr_internal_core_api_imxrt1170.c
mcmgr_mu_internal.c
rpmsg_lite
common
llist.c
include
environment
bm
rpmsg_env_specific.h
platform
imxrt1170
rpmsg_platform.h
llist.h
rpmsg_compiler.h
rpmsg_default_config.h
rpmsg_env.h
rpmsg_lite.h
rpmsg_ns.h
virtio_ring.h
virtqueue.h
rpmsg_lite
porting
environment
rpmsg_env_bm.c
platform
imxrt1170
rpmsg_lite.c
rpmsg_ns.c
virtio
virtqueue.c
source
startup
utilities
xip
Output
erpc_matrix_multiply_rpmsg_cm7

|

Parent topic: Multicore client application

Client user code The client's user code is stored in the main_core0.c file, located in the following folder:

<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_example/erpc_matrix_multiply_rpmsg/cm7

The main_core0.c file contains the code for target board and eRPC initialization.

- After initialization, the secondary core is released from reset.
- When the secondary core is ready, the primary core initializes two matrix variables.
- The erpcMatrixMultiply eRPC function is called to issue the eRPC request and get the result.

It is possible to write the application-specific eRPC error handler. The eRPC error handler of the matrix multiply application is implemented in erpc_error_handler.h and erpc_error_handler.cpp files.

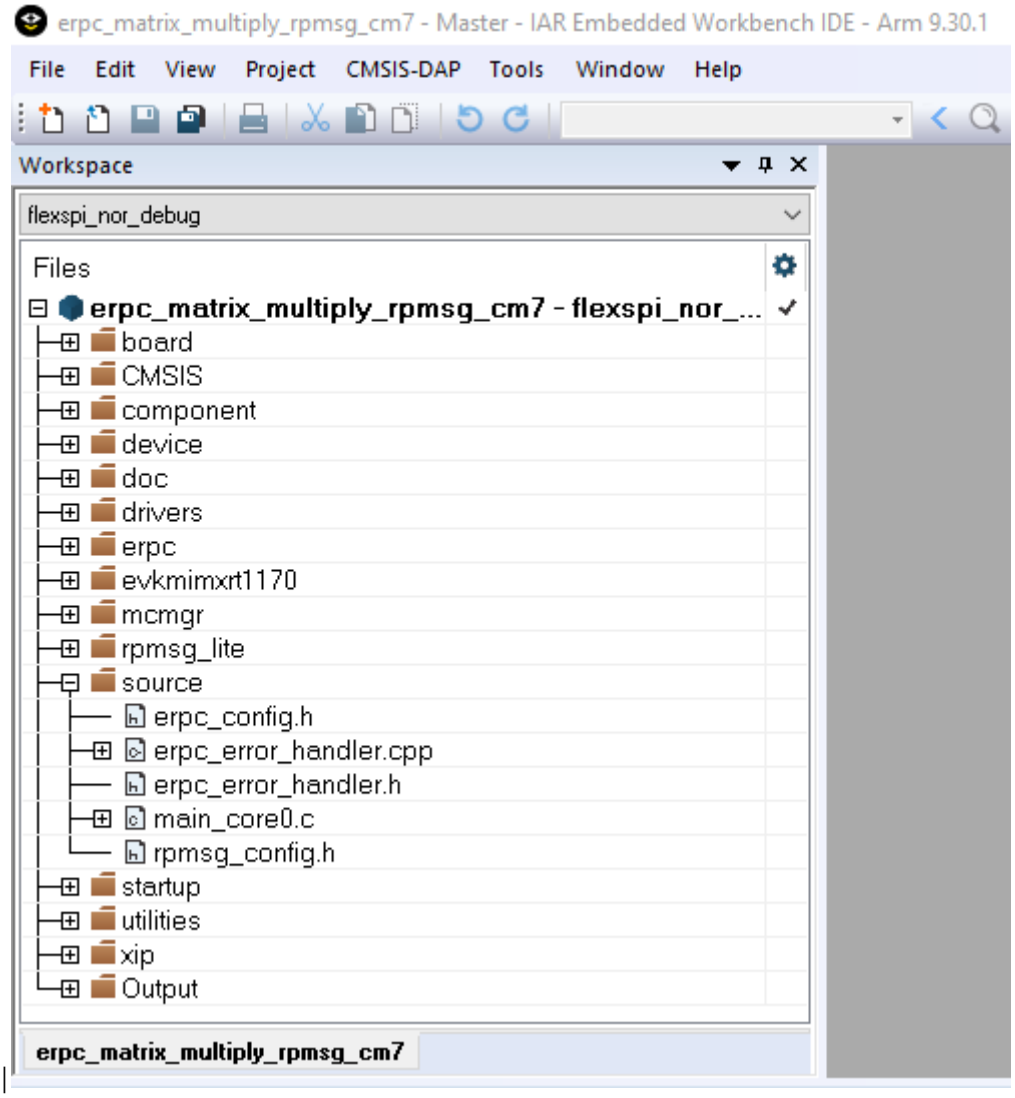
The matrix multiplication can be issued repeatedly, when pressing a software board button.

The eRPC-relevant code is captured in the following code snippet:

```
...
extern bool g_erpc_error_occurred;
...
/* Declare matrix arrays */
Matrix matrix1 = {0}, matrix2 = {0}, result_matrix = {0};
...
/* RPMsg-Lite transport layer initialization */
erpc_transport_t transport;
transport = erpc_transport_rpmsg_lite_master_init(src, dst,
ERPC_TRANSPORT_RPMSG_LITE_LINK_ID);
...
/* MessageBufferFactory initialization */
erpc_mbf_t message_buffer_factory;
message_buffer_factory = erpc_mbf_rpmsg_init(transport);
...
/* eRPC client side initialization */
erpc_client_t client;
client = erpc_client_init(transport, message_buffer_factory);
...
/* Set default error handler */
erpc_client_set_error_handler(client, erpc_error_handler);
...
while (1)
{
  /* Invoke the erpcMatrixMultiply function */
  erpcMatrixMultiply(matrix1, matrix2, result_matrix);
  ...
  /* Check if some error occured in eRPC */
  if (g_erpc_error_occurred)
  {
    /* Exit program loop */
    break;
  }
  ...
}
```

Except for the application main file, there are configuration files for the RPMsg-Lite (rpmsg_config.h) and eRPC (erpc_config.h), located in the following folder:

<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_matrix_multiply_rpmsg



Parent topic: Multicore client application

Parent topic:*Create an eRPC application*

Multiprocessor server application The "Matrix multiply" eRPC server project for multiprocessor applications is located in the *<MCUXpressoSDK_install_dir»/boards/<board_name>/multiprocessor_examples/erpc_server_matrix_multiply_<transport_layer>* folder.

Most of the multiprocessor application setup is the same as for the multicore application. The multiprocessor server application requires server-related generated files (server shim code), server infrastructure files, and the server user code. There is no need for server multicore infrastructure files (MCMGR and RPMsg-Lite). The RPMsg-Lite transport layer is replaced either by SPI or UART transports. The following table shows the required transport-related files per each transport type.

|SPI|<eRPC base directory>/erpc_c/setup/erpc_setup_(d)spi_slave.cpp

<eRPC base directory>/erpc_c/transports/erpc_(d)spi_slave_transport.hpp

<eRPC base directory>/erpc_c/transports/erpc_(d)spi_slave_transport.cpp

| |UART|<eRPC base directory>/erpc_c/setup/erpc_setup_uart_cmsis.cpp

<eRPC base directory>/erpc_c/transports/erpc_uart_cmsis_transport.hpp

<eRPC base directory>/erpc_c/transports/erpc_uart_cmsis_transport.cpp

|

Server user code The server's user code is stored in the main_server.c file, located in the *<MCUXpressoSDK_install_dir>/boards/<board_name>/multiprocessor_examples/erpc_server_matrix_multiply_<transport_layer>/* folder.

The eRPC-relevant code with UART as a transport is captured in the following code snippet:

```
/* erpcMatrixMultiply function user implementation */
void erpcMatrixMultiply(Matrix matrix1, Matrix matrix2, Matrix result_matrix)
{
  ...
}
int main()
{
  ...
  /* UART transport layer initialization, ERPC_DEMO_UART is the structure of CMSIS UART driver␣
↪operations */
  erpc_transport_t transport;
  transport = erpc_transport_cmsis_uart_init((void *)&ERPC_DEMO_UART);
  ...
  /* MessageBufferFactory initialization */
  erpc_mbf_t message_buffer_factory;
  message_buffer_factory = erpc_mbf_dynamic_init();
  ...
  /* eRPC server side initialization */
  erpc_server_t server;
  server = erpc_server_init(transport, message_buffer_factory);
  ...
  /* Adding the service to the server */
  erpc_service_t service = create_MatrixMultiplyService_service();
  erpc_add_service_to_server(server, service);
  ...
  while (1)
  {
    /* Process eRPC requests */
    erpc_status_t status = erpc_server_poll(server)
    /* handle error status */
    if (status != kErpcStatus_Success)
    {
      /* print error description */
      erpc_error_handler(status, 0);
      ...
    }
    ...
  }
}
```

Parent topic: Multiprocessor server application

Multiprocessor client application The "Matrix multiply" eRPC client project for multiprocessor applications is located in the *<MCUXpressoSDK_install_dir>/boards/<board_name>/multiprocessor_examples/erpc_client_matrix_multiply_<transport_layer>/iar/* folder.

Most of the multiprocessor application setup is the same as for the multicore application. The multiprocessor server application requires client-related generated files (server shim code),

client infrastructure files, and the client user code. There is no need for client multicore infrastructure files (MCMGR and RPMsg-Lite). The RPMsg-Lite transport layer is replaced either by SPI or UART transports. The following table shows the required transport-related files per each transport type.

|SPI|<eRPC base directory>/erpc_c/setup/erpc_setup_(d)spi_master.cpp

<eRPC base directory>/erpc_c/transports/ erpc_(d)spi_master_transport.hpp

<eRPC base directory>/erpc_c/transports/ erpc_(d)spi_master_transport.cpp

| |UART|<eRPC base directory>/erpc_c/setup/erpc_setup_uart_cmsis.cpp

<eRPC base directory>/erpc_c/transports/erpc_uart_cmsis_transport.hpp

<eRPC base directory>/erpc_c/transports/erpc_uart_cmsis_transport.cpp

|

Client user code The client's user code is stored in the main_client.c file, located in the *<MCUXpressoSDK_install_dir>/boards/<board_name>/multiprocessor_examples/erpc_client_matrix_multiply_<transport_layer>/* folder.

The eRPC-relevant code with UART as a transport is captured in the following code snippet:

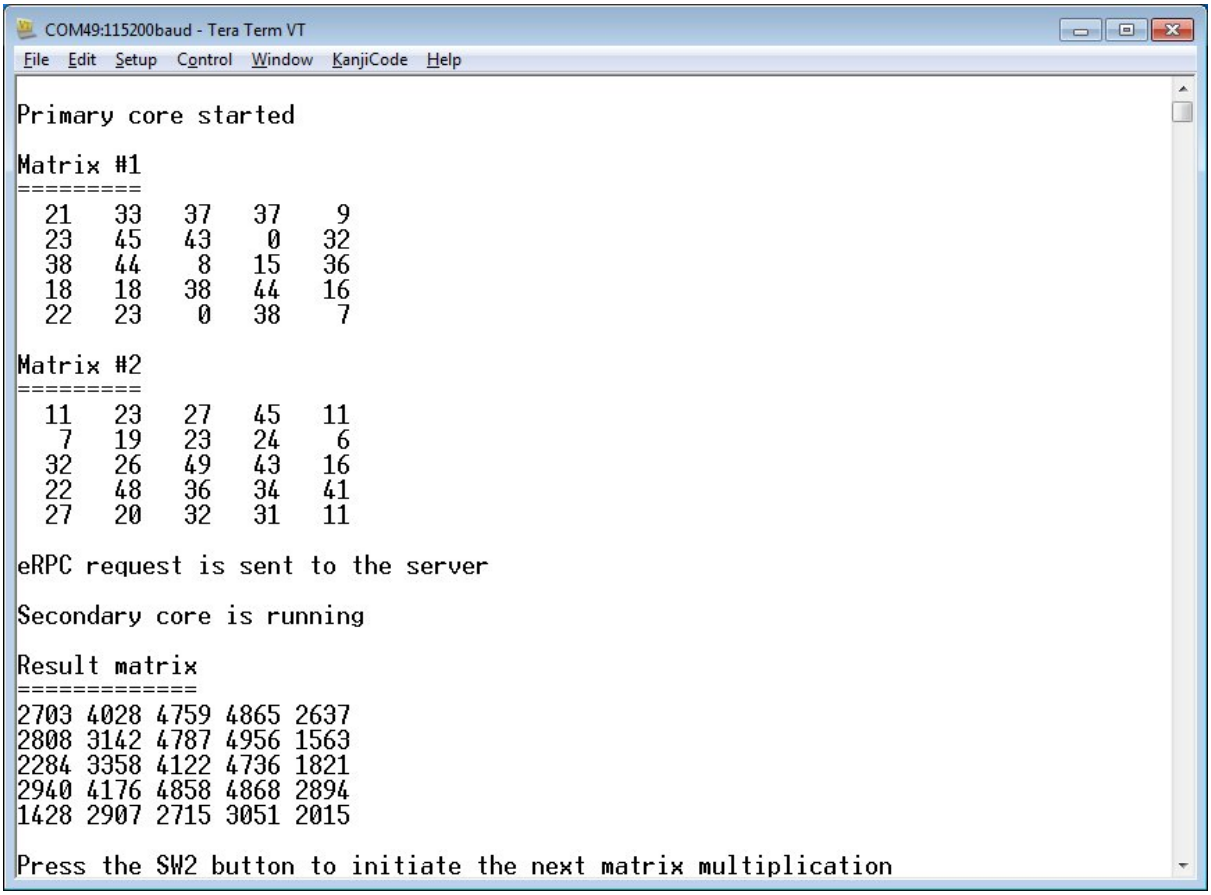
```
...
extern bool g_erpc_error_occurred;
...
/* Declare matrix arrays */
Matrix matrix1 = {0}, matrix2 = {0}, result_matrix = {0};
...
/* UART transport layer initialization, ERPC_DEMO_UART is the structure of CMSIS UART driver␣
↪operations */
erpc_transport_t transport;
transport = erpc_transport_cmsis_uart_init((void *)&ERPC_DEMO_UART);
...
/* MessageBufferFactory initialization */
erpc_mbf_t message_buffer_factory;
message_buffer_factory = erpc_mbf_dynamic_init();
...
/* eRPC client side initialization */
erpc_client_t client;
client = erpc_client_init(transport,message_buffer_factory);
...
/* Set default error handler */
erpc_client_set_error_handler(client, erpc_error_handler);
...
while (1)
{
  /* Invoke the erpcMatrixMultiply function */
  erpcMatrixMultiply(matrix1, matrix2, result_matrix);
  ...
  /* Check if some error occured in eRPC */
  if (g_erpc_error_occurred)
  {
    /* Exit program loop */
    break;
  }
  ...
}
```

Parent topic: Multiprocessor client application

Parent topic: Multiprocessor server application

Parent topic:*Create an eRPC application*

Running the eRPC application Follow the instructions in *Getting Started with MCUXpresso SDK* (document MCUXSDKGSUG) (located in the *<MCUXpressoSDK_install_dir>/docs* folder), to load both the primary and the secondary core images into the on-chip memory, and then effectively debug the dual-core application. After the application is running, the serial console should look like:



For multiprocessor applications that are running between PC and the target evaluation board or between two boards, follow the instructions in the accompanied example readme files that provide details about the proper board setup and the PC side setup (Python).

Parent topic:*Create an eRPC application*

Parent topic:*eRPC example*

eRPC example This section shows how to create an example eRPC application called “Matrix multiply”, which implements one eRPC function (matrix multiply) with two function parameters (two matrices). The client-side application calls this eRPC function, and the server side performs the multiplication of received matrices. The server side then returns the result.

For example, use the NXP MIMXRT1170-EVK board as the target dual-core platform, and the IAR Embedded Workbench for ARM (EWARM) as the target IDE for developing the eRPC example.

- The primary core (CM7) runs the eRPC client.
- The secondary core (CM4) runs the eRPC server.
- RPMsg-Lite (Remote Processor Messaging Lite) is used as the eRPC transport layer.

The “Matrix multiply” application can be also run in the multi-processor setup. In other words, the eRPC client running on one SoC comunicates with the eRPC server that runs on anothe SoC, utilizing different transport channels. It is possible to run the board-to-PC example (PC as the eRPC server and a board as the eRPC client, and vice versa) and also the board-to-board example. These multiprocessor examples are prepared for selected boards only.

|Multicore application source and project files|*<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore*
|Multiprocessor application source and project files|*<MCUXpressoSDK_install_dir>/boards/<board_name>/multi*

<MCUXpressoSDK_install_dir>/boards/<board_name>/multiprocessor_examples/erpc_server_matrix_multiply_<tr

| |eRPC source files|*<MCUXpressoSDK_install_dir>/middleware/multicore/erpc/*| |RPMsg-Lite source files|*<MCUXpressoSDK_install_dir>/middleware/multicore/rpmsg_lite/*|

Designing the eRPC application The matrix multiply application is based on calling single eRPC function that takes 2 two-dimensional arrays as input and returns matrix multiplication results as another 2 two-dimensional array. The IDL file syntax supports arrays with the dimension length set by the number only (in the current eRPC implementation). Because of this, a variable is declared in the IDL dedicated to store information about matrix dimension length, and to allow easy maintenance of the user and server code.

For a simple use of the two-dimensional array, the alias name (new type definition) for this data type has is declared in the IDL. Declaring this alias name ensures that the same data type can be used across the client and server applications.

Parent topic:*eRPC example*

Creating the IDL file The created IDL file is located in the following folder:

<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_common/erpc_matrix_multiply/s

The created IDL file contains the following code:

```
program erpc_matrix_multiply
/*! This const defines the matrix size. The value has to be the same as the
Matrix array dimension. Do not forget to re-generate the erpc code once the
matrix size is changed in the erpc file */
const int32 matrix_size = 5;
/*! This is the matrix array type. The dimension has to be the same as the
matrix size const. Do not forget to re-generate the erpc code once the
matrix size is changed in the erpc file */
type Matrix = int32[matrix_size][matrix_size];
interface MatrixMultiplyService {
erpcMatrixMultiply(in Matrix matrix1, in Matrix matrix2, out Matrix result_matrix) ->
void
}
```

Details:

- The IDL file starts with the program name (*erpc_matrix_multiply*), and this program name is used in the naming of all generated outputs.
- The declaration and definition of the constant variable named *matrix_size* follows next. The *matrix_size* variable is used for passing information about the length of matrix dimensions to the client/server user code.
- The alias name for the two-dimensional array type (*Matrix*) is declared.
- The interface group *MatrixMultiplyService* is located at the end of the IDL file. This interface group contains only one function declaration *erpcMatrixMultiply*.
- As shown above, the function’s declaration contains three parameters of Matrix type: *matrix1* and *matrix2* are input parameters, while *result_matrix* is the output parameter. Additionally, the returned data type is declared as void.

When writing the IDL file, the following order of items is recommended:

1. Program name at the top of the IDL file.
2. New data types and constants declarations.
3. Declarations of interfaces and functions at the end of the IDL file.

Parent topic:*eRPC example*

Using the eRPC generator tool |Windows OS|<MCUXpressoSDK_install_dir>/middleware/multicore/tools/erp
|Linux OS|<MCUXpressoSDK_install_dir>/middleware/multicore/tools/erpcgen/Linux_x64

<MCUXpressoSDK_install_dir>/middleware/multicore/tools/erpcgen/Linux_x86

| |Mac OS|<MCUXpressoSDK_install_dir>/middleware/multicore/tools/erpcgen/Mac|

The files for the "Matrix multiply" example are pre-generated and already a part of the application projects. The following section describes how they have been created.

- The easiest way to create the shim code is to copy the erpcgen application to the same folder where the IDL file (*.erpc) is located; then run the following command:

  erpcgen <IDL_file>.erpc

- In the "Matrix multiply" example, the command should look like:

  erpcgen erpc_matrix_multiply.erpc

Additionally, another method to create the shim code is to execute the eRPC application using input commands:

- "-?"/"—help" – Shows supported commands.
- "-o <filePath>"/"—output<filePath>" – Sets the output directory.

For example,

```
<path_to_erpcgen>/erpcgen –o <path_to_output>
<path_to_IDL>/<IDL_file_name>.erpc
```

For the "Matrix multiply" example, when the command is executed from the default erpcgen location, it looks like:

erpcgen –o

../../../../../boards/evkmimxrt1170/multicore_examples/erpc_common/erpc_matrix_multiply/service
../../../../../boards/evkmimxrt1170/multicore_examples/erpc_common/erpc_matrix_multiply/service/erpc_matrix_mu

In both cases, the following four files are generated into the *<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_common/erpc_matrix_multiply/service folder.*

- erpc_matrix_multiply.h
- erpc_matrix_multiply_client.cpp
- erpc_matrix_multiply_server.h
- erpc_matrix_multiply_server.cpp

For multiprocessor examples, the eRPC file and pre-generated files can be found in the *<MCUXpressoSDK_install_dir>/boards/<board_name>/multiprocessor_examples/erpc_common/erpc_matrix_multiply/serv* folder.

For Linux OS users:

- Do not forget to set the permissions for the eRPC generator application.
- Run the application as ./erpcgen... instead of as erpcgen ....

Parent topic:*eRPC example*

Create an eRPC application This section describes a generic way to create a client/server eRPC application:

1. **Design the eRPC application:** Decide which data types are sent between applications, and define functions that send/receive this data.
2. **Create the IDL file:** The IDL file contains information about data types and functions used in an eRPC application, and is written in the IDL language.
3. **Use the eRPC generator tool:** This tool takes an IDL file and generates the shim code for the client and the server-side applications.
4. **Create an eRPC application:**
    1. Create two projects, where one project is for the client side (primary core) and the other project is for the server side (secondary core).
    2. Add generated files for the client application to the client project, and add generated files for the server application to the server project.
    3. Add infrastructure files.
    4. Add user code for client and server applications.
    5. Set the client and server project options.
5. **Run the eRPC application:** Run both the server and the client applications. Make sure that the server has been run before the client request was sent.

A specific example follows in the next section.

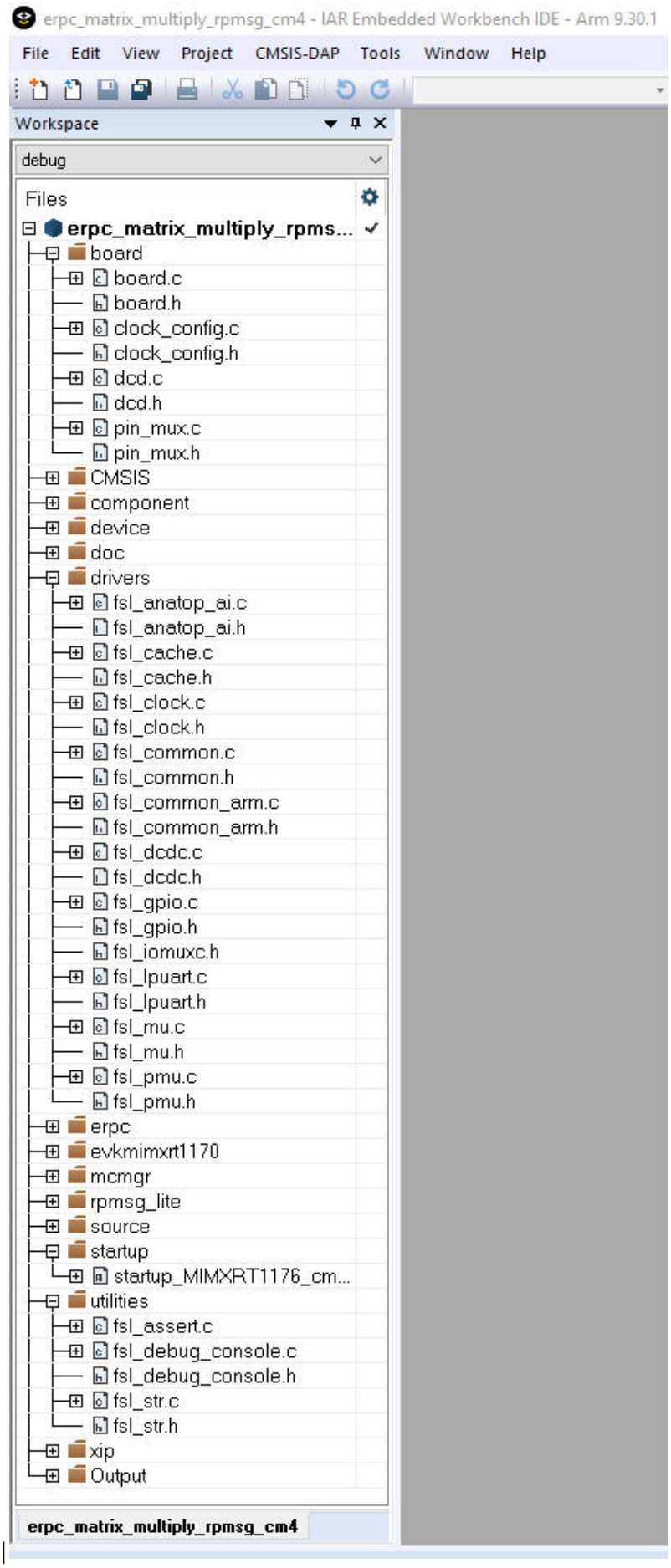
Multicore server application The “Matrix multiply” eRPC server project is located in the following folder:

<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_matrix_multiply_rpmsg/cm4/iar

The project files for the eRPC server have the `_cm4` suffix.

Server project basic source files The startup files, board-related settings, peripheral drivers, and utilities belong to the basic project source files and form the skeleton of all MCUXpresso SDK applications. These source files are located in:

- *<MCUXpressoSDK_install_dir>/devices/<device>*
- *<MCUXpressoSDK_install_dir>/boards/<board_name>/multicore_examples/<example_name>/*


erpc_matrix_multiply_rpmsg_cm4 - IAR Embedded Workbench IDE - Arm 9.30.1
File Edit View Project CMSIS-DAP Tools Window Help
Workspace
debug
Files
erpc_matrix_multiply_rpms...
board
board.c
board.h
clock_config.c
clock_config.h
dcd.c
dcd.h
pin_mux.c
pin_mux.h
CMSIS
component
device
doc
drivers
fsl_anatop_ai.c
fsl_anatop_ai.h
fsl_cache.c
fsl_cache.h
fsl_clock.c
fsl_clock.h
fsl_common.c
fsl_common.h
fsl_common_arm.c
fsl_common_arm.h
fsl_dcdc.c
fsl_dcdc.h
fsl_gpio.c
fsl_gpio.h
fsl_iomuxc.h
fsl_lpuart.c
fsl_lpuart.h
fsl_mu.c
fsl_mu.h
fsl_pmu.c
fsl_pmu.h
erpc
evkmimxrt1170
mcmgr
rpmsg_lite
source
startup
startup_MIMXRT1176_cm...
utilities
fsl_assert.c
fsl_debug_console.c
fsl_debug_console.h
fsl_str.c
fsl_str.h
xip
Output
erpc_matrix_multiply_rpmsg_cm4

|

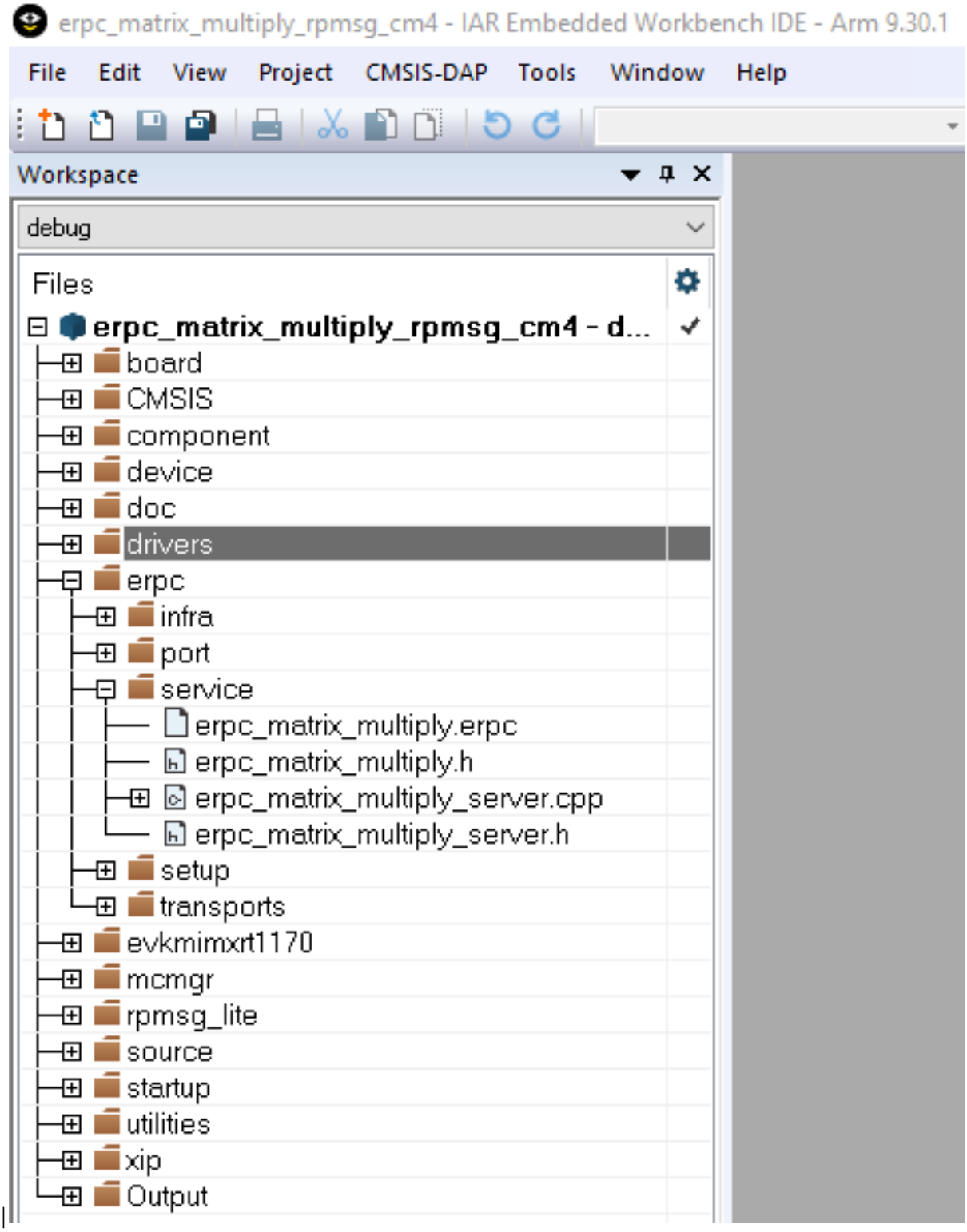
Parent topic: Multicore server application

Server related generated files The server-related generated files are:

- erpc_matric_multiply.h
- erpc_matrix_multiply_server.h
- erpc_matrix_multiply_server.cpp

The server-related generated files contain the shim code for functions and data types declared in the IDL file. These files also contain functions for the identification of client requested functions, data deserialization, calling requested function's implementations, and data serialization and return, if requested by the client. These shim code files can be found in the following folder:

<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_common/erpc_matrix_multiply/s



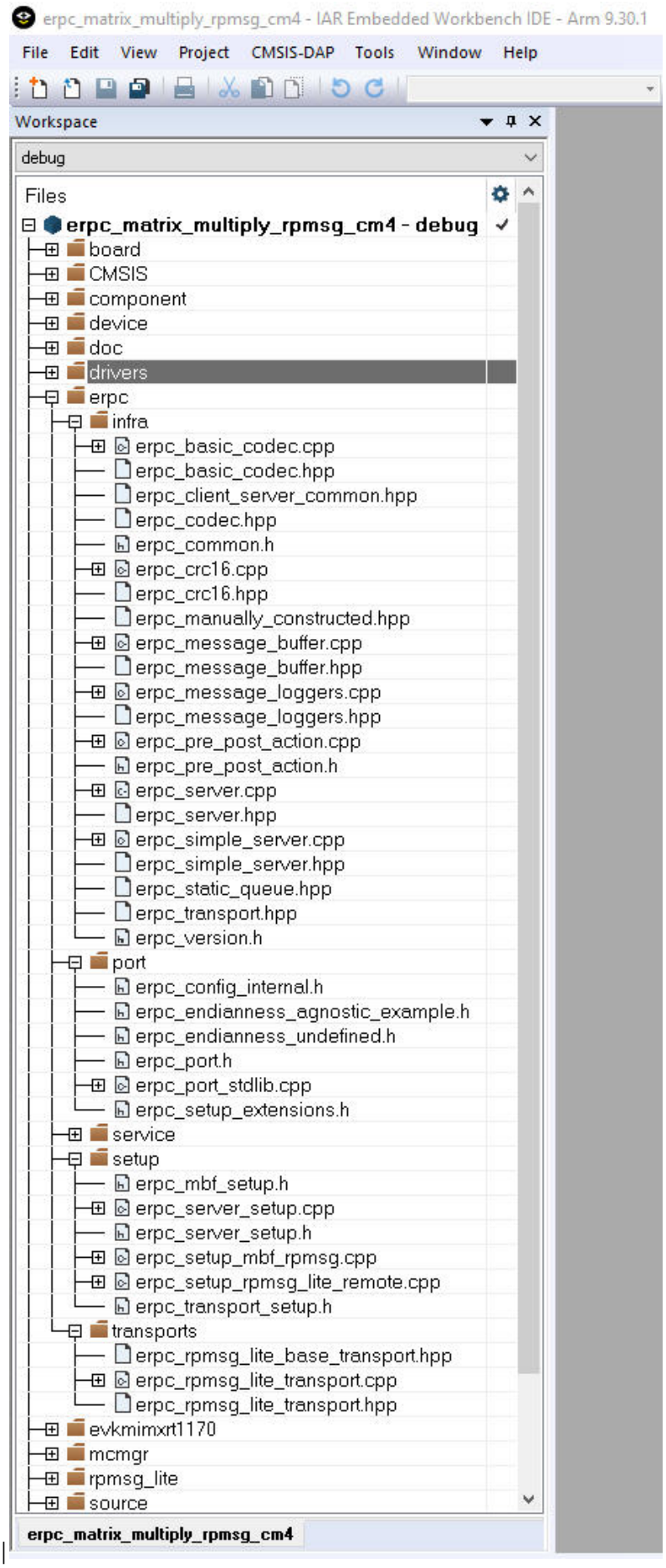
Parent topic: Multicore server application

Server infrastructure files The eRPC infrastructure files are located in the following folder:

<MCUXpressoSDK_install_dir>/middleware/multicore/erpc/erpc_c

The **erpc_c** folder contains files for creating eRPC client and server applications in the *C/C++* language. These files are distributed into subfolders.

- The **infra** subfolder contains C++ infrastructure code used to build server and client applications.
  - Four files, erpc_server.hpp, erpc_server.cpp, erpc_simple_server.hpp, and erpc_simple_server.cpp, are used for running the eRPC server on the server-side applications. The simple server is currently the only implementation of the server, and its role is to catch client requests, identify and call requested functions, and send data back when requested.
  - Three files (erpc_codec.hpp, erpc_basic_codec.hpp, and erpc_basic_codec.cpp) are used for codecs. Currently, the basic codec is the initial and only implementation of the codecs.
  - The erpc_common.hpp file is used for common eRPC definitions, typedefs, and enums.
  - The erpc_manually_constructed.hpp file is used for allocating static storage for the used objects.
  - Message buffer files are used for storing serialized data: erpc_message_buffer.h and erpc_message_buffer.cpp.
  - The erpc_transport.h file defines the abstract interface for transport layer.
- The **port** subfolder contains the eRPC porting layer to adapt to different environments.
  - erpc_port.h file contains definition of erpc_malloc() and erpc_free() functions.
  - erpc_port_stdlib.cpp file ensures adaptation to stdlib.
  - erpc_config_internal.h internal erpc configuration file.
- The **setup** subfolder contains a set of plain C APIs that wrap the C++ infrastructure, providing client and server init and deinit routines that greatly simplify eRPC usage in C-based projects. No knowledge of C++ is required to use these APIs.
  - The erpc_server_setup.h and erpc_server_setup.cpp files needs to be added into the "Matrix multiply" example project to demonstrate the use of C-wrapped functions in this example.
  - The erpc_transport_setup.h and erpc_setup_rpmsg_lite_remote.cpp files needs to be added into the project in order to allow the C-wrapped function for transport layer setup.
  - The erpc_mbf_setup.h and erpc_setup_mbf_rpmsg.cpp files needs to be added into the project in order to allow message buffer factory usage.
- The **transports** subfolder contains transport classes for the different methods of communication supported by eRPC. Some transports are applicable only to host PCs, while others are applicable only to embedded or multicore systems. Most transports have corresponding client and server setup functions in the setup folder.
  - RPMsg-Lite is used as the transport layer for the communication between cores, erpc_rpmsg_lite_base_transport.hpp, erpc_rpmsg_lite_transport.hpp, and erpc_rpmsg_lite_transport.cpp files need to be added into the server project.


erpc_matrix_multiply_rpmsg_cm4 - IAR Embedded Workbench IDE - Arm 9.30.1
File Edit View Project CMSIS-DAP Tools Window Help
Workspace
debug
Files
erpc_matrix_multiply_rpmsg_cm4 - debug
board
CMSIS
component
device
doc
drivers
erpc
infra
erpc_basic_codec.cpp
erpc_basic_codec.hpp
erpc_client_server_common.hpp
erpc_codec.hpp
erpc_common.h
erpc_crc16.cpp
erpc_crc16.hpp
erpc_manually_constructed.hpp
erpc_message_buffer.cpp
erpc_message_buffer.hpp
erpc_message_loggers.cpp
erpc_message_loggers.hpp
erpc_pre_post_action.cpp
erpc_pre_post_action.h
erpc_server.cpp
erpc_server.hpp
erpc_simple_server.cpp
erpc_simple_server.hpp
erpc_static_queue.hpp
erpc_transport.hpp
erpc_version.h
port
erpc_config_internal.h
erpc_endianness_agnostic_example.h
erpc_endianness_undefined.h
erpc_port.h
erpc_port_stdlib.cpp
erpc_setup_extensions.h
service
setup
erpc_mbf_setup.h
erpc_server_setup.cpp
erpc_server_setup.h
erpc_setup_mbf_rpmsg.cpp
erpc_setup_rpmsg_lite_remote.cpp
erpc_transport_setup.h
transports
erpc_rpmsg_lite_base_transport.hpp
erpc_rpmsg_lite_transport.cpp
erpc_rpmsg_lite_transport.hpp
evkmimxrt1170
mcmgr
rpmsg_lite
source
erpc_matrix_multiply_rpmsg_cm4

|

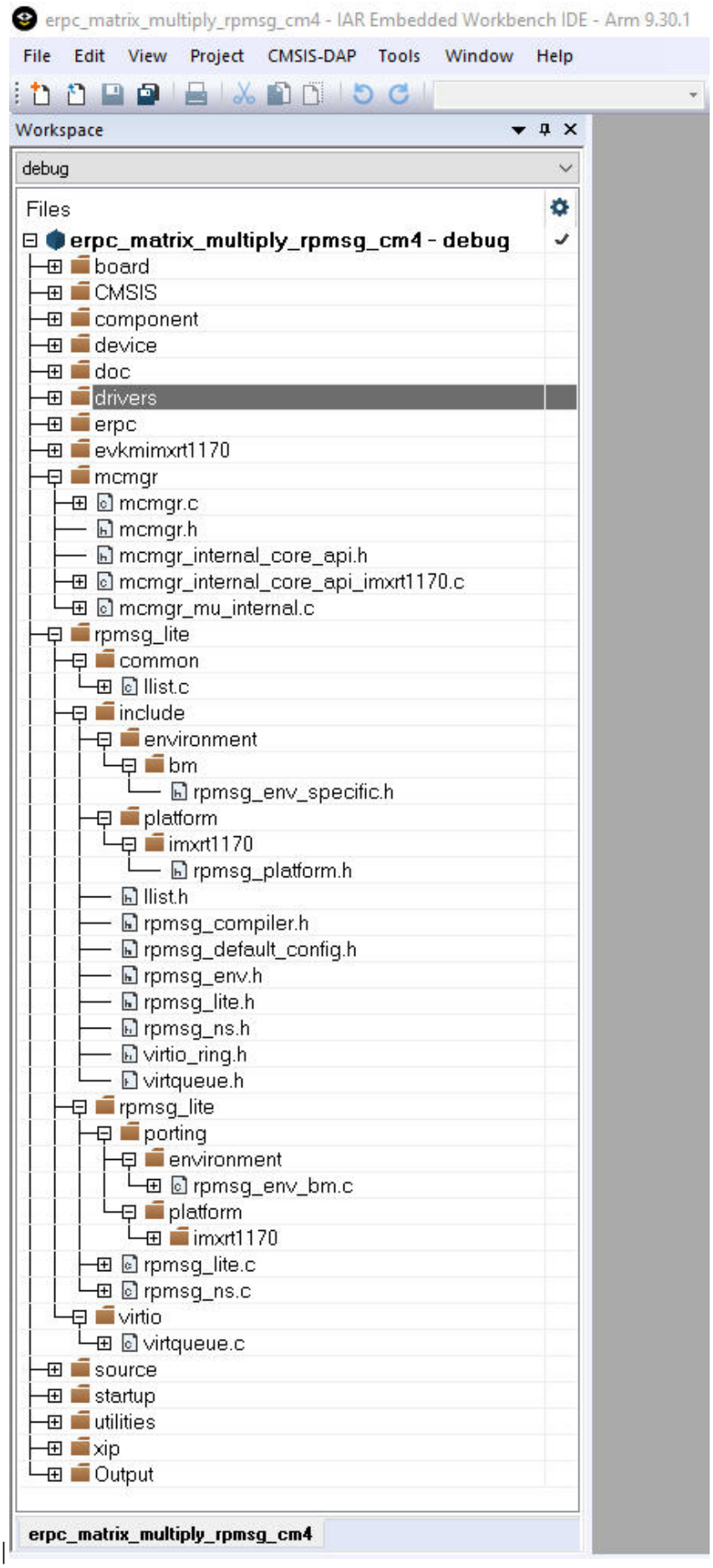
Parent topic: Multicore server application

Server multicore infrastructure files Because of the RPMsg-Lite (transport layer), it is also necessary to include RPMsg-Lite related files, which are in the following folder:

<MCUXpressoSDK_install_dir>/middleware/multicore/rpmsg_lite/

The multicore example applications also use the Multicore Manager software library to control the secondary core startup and shutdown. These source files are located in the following folder:

<MCUXpressoSDK_install_dir>/middleware/multicore/mcmgr/


erpc_matrix_multiply_rpmsg_cm4 - IAR Embedded Workbench IDE - Arm 9.30.1
File Edit View Project CMSIS-DAP Tools Window Help
Workspace
debug
Files
erpc_matrix_multiply_rpmsg_cm4 - debug
board
CMSIS
component
device
doc
drivers
erpc
evkmimxrt1170
mcmgr
mcmgr.c
mcmgr.h
mcmgr_internal_core_api.h
mcmgr_internal_core_api_imxrt1170.c
mcmgr_mu_internal.c
rpmsg_lite
common
llist.c
include
environment
bm
rpmsg_env_specific.h
platform
imxrt1170
rpmsg_platform.h
llist.h
rpmsg_compiler.h
rpmsg_default_config.h
rpmsg_env.h
rpmsg_lite.h
rpmsg_ns.h
virtio_ring.h
virtqueue.h
rpmsg_lite
porting
environment
rpmsg_env_bm.c
platform
imxrt1170
rpmsg_lite.c
rpmsg_ns.c
virtio
virtqueue.c
source
startup
utilities
xip
Output
erpc_matrix_multiply_rpmsg_cm4

|

Parent topic: Multicore server application

Server user code The server's user code is stored in the main_core1.c file, located in the following folder:

<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_matrix_multiply_rpmsg/cm4

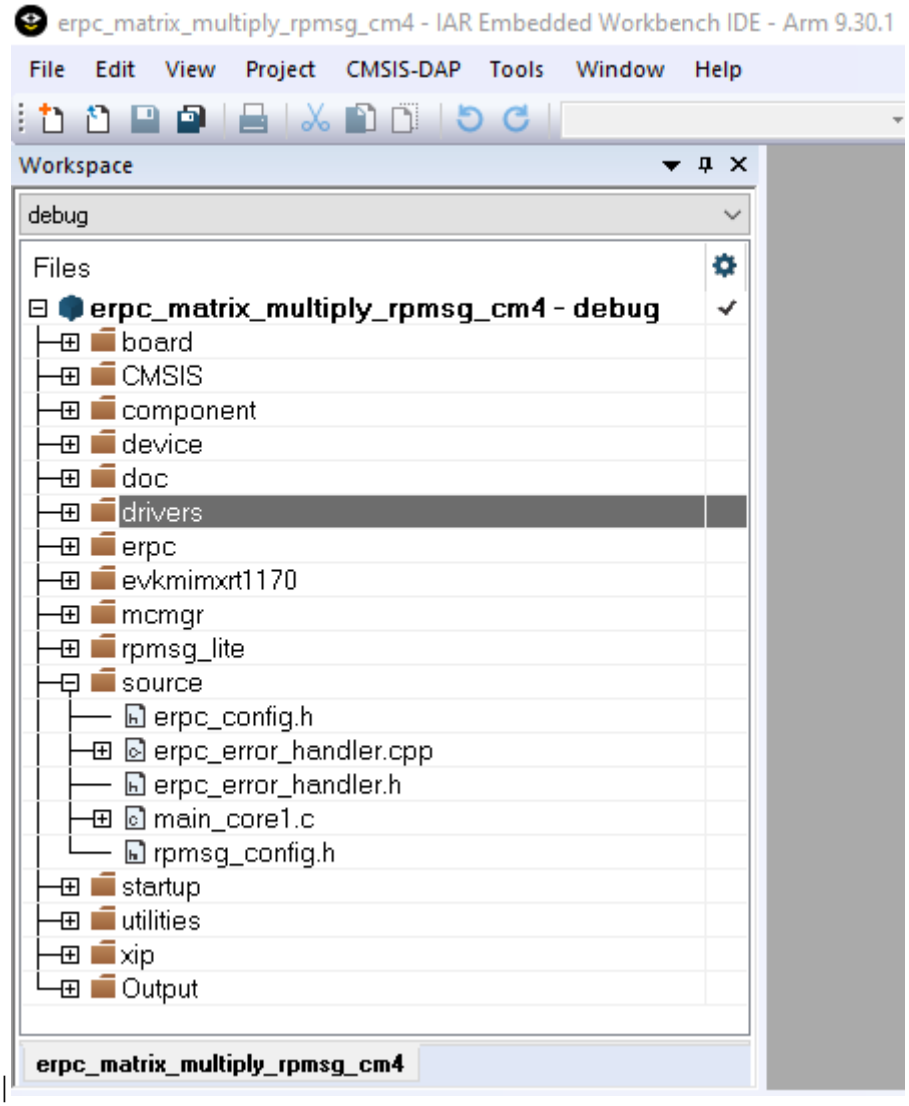
The main_core1.c file contains two functions:

- The **main()** function contains the code for the target board and eRPC server initialization. After the initialization, the matrix multiply service is added and the eRPC server waits for client's requests in the while loop.
- The **erpcMatrixMultiply()** function is the user implementation of the eRPC function defined in the IDL file.
- There is the possibility to write the application-specific eRPC error handler. The eRPC error handler of the matrix multiply application is implemented in the erpc_error_handler.h and erpc_error_handler.cpp files.

The eRPC-relevant code is captured in the following code snippet:

```
/* erpcMatrixMultiply function user implementation */
void erpcMatrixMultiply(const Matrix *matrix1, const Matrix *matrix2, Matrix *result_matrix)
{
  ...
}
int main()
{
  ...
  /* RPMsg-Lite transport layer initialization */
  erpc_transport_t transport;
  transport = erpc_transport_rpmsg_lite_remote_init(src, dst, (void*)startupData,
  ERPC_TRANSPORT_RPMSG_LITE_LINK_ID, SignalReady, NULL);
  ...
  /* MessageBufferFactory initialization */
  erpc_mbf_t message_buffer_factory;
  message_buffer_factory = erpc_mbf_rpmsg_init(transport);
  ...
  /* eRPC server side initialization */
  erpc_server_t server;
  server = erpc_server_init(transport, message_buffer_factory);
  ...
  /* Adding the service to the server */
  erpc_service_t service = create_MatrixMultiplyService_service();
  erpc_add_service_to_server(server, service);
  ...
  while (1)
  {
    /* Process eRPC requests */
    erpc_status_t status = erpc_server_poll(server);
    /* handle error status */
    if (status != kErpcStatus_Success)
    {
      /* print error description */
      erpc_error_handler(status, 0);
      ...
    }
    ...
  }
}
```

Except for the application main file, there are configuration files for the RPMsg-Lite (rpmsg_config.h) and eRPC (erpc_config.h), located in the *<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_matrix_multiply_rpmsg* folder.



Parent topic: Multicore server application

Parent topic:*Create an eRPC application*

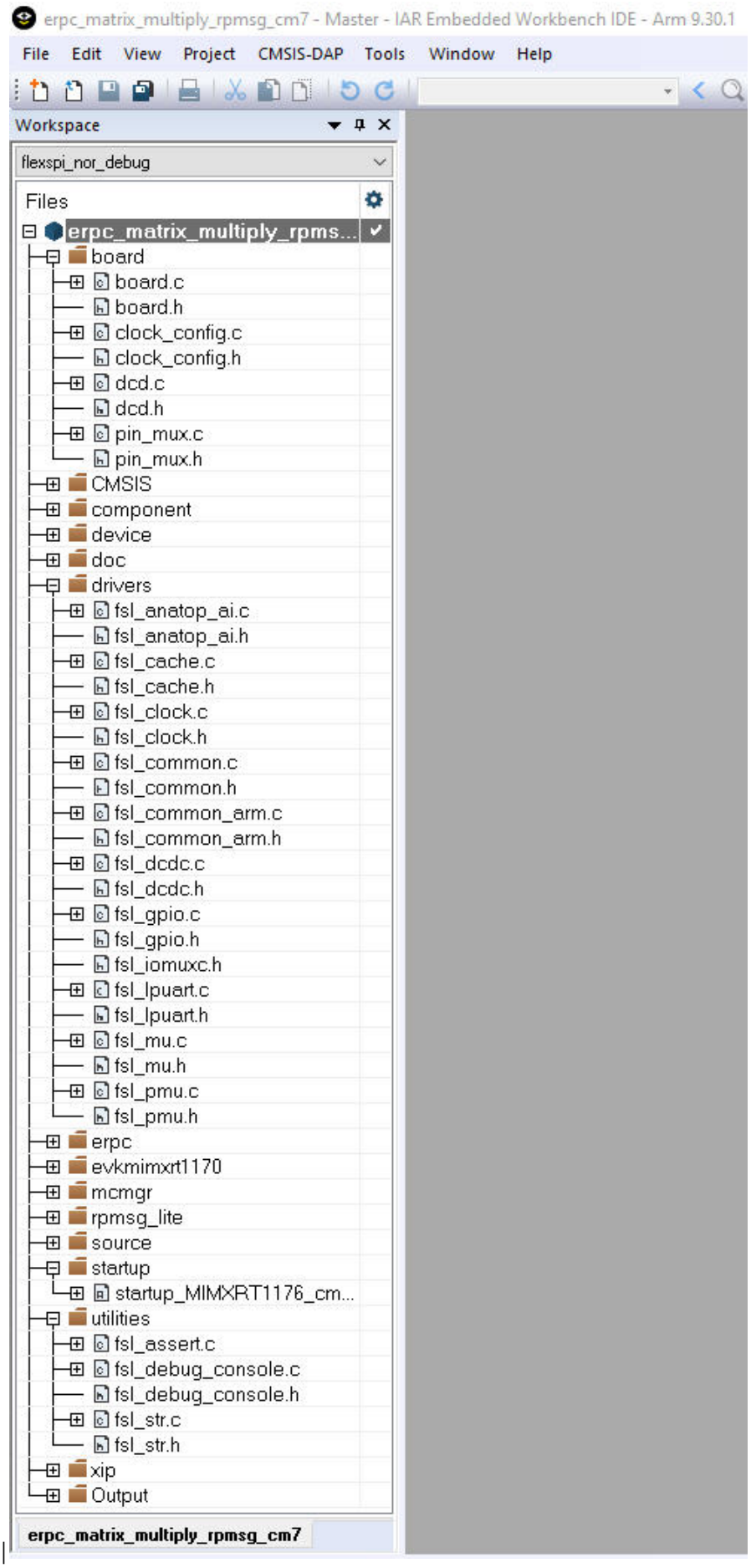
Multicore client application The "Matrix multiply" eRPC client project is located in the following folder:

<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_matrix_multiply_rpmsg/cm7/iar

Project files for the eRPC client have the _cm7 suffix.

Client project basic source files The startup files, board-related settings, peripheral drivers, and utilities belong to the basic project source files and form the skeleton of all MCUXpresso SDK applications. These source files are located in the following folders:

- *<MCUXpressoSDK_install_dir>/devices/<device>*
- *<MCUXpressoSDK_install_dir>/boards/<board_name>/multicore_examples/<example_name>/*


erpc_matrix_multiply_rpmsg_cm7 - Master - IAR Embedded Workbench IDE - Arm 9.30.1
File Edit View Project CMSIS-DAP Tools Window Help
Workspace
flexspi_nor_debug
Files
erpc_matrix_multiply_rpms...
board
board.c
board.h
clock_config.c
clock_config.h
dcd.c
dcd.h
pin_mux.c
pin_mux.h
CMSIS
component
device
doc
drivers
fsl_anatop_ai.c
fsl_anatop_ai.h
fsl_cache.c
fsl_cache.h
fsl_clock.c
fsl_clock.h
fsl_common.c
fsl_common.h
fsl_common_arm.c
fsl_common_arm.h
fsl_dcdc.c
fsl_dcdc.h
fsl_gpio.c
fsl_gpio.h
fsl_iomuxc.h
fsl_lpuart.c
fsl_lpuart.h
fsl_mu.c
fsl_mu.h
fsl_pmu.c
fsl_pmu.h
erpc
evkmimxrt1170
mcmgr
rpmsg_lite
source
startup
startup_MIMXRT1176_cm...
utilities
fsl_assert.c
fsl_debug_console.c
fsl_debug_console.h
fsl_str.c
fsl_str.h
xip
Output
erpc_matrix_multiply_rpmsg_cm7

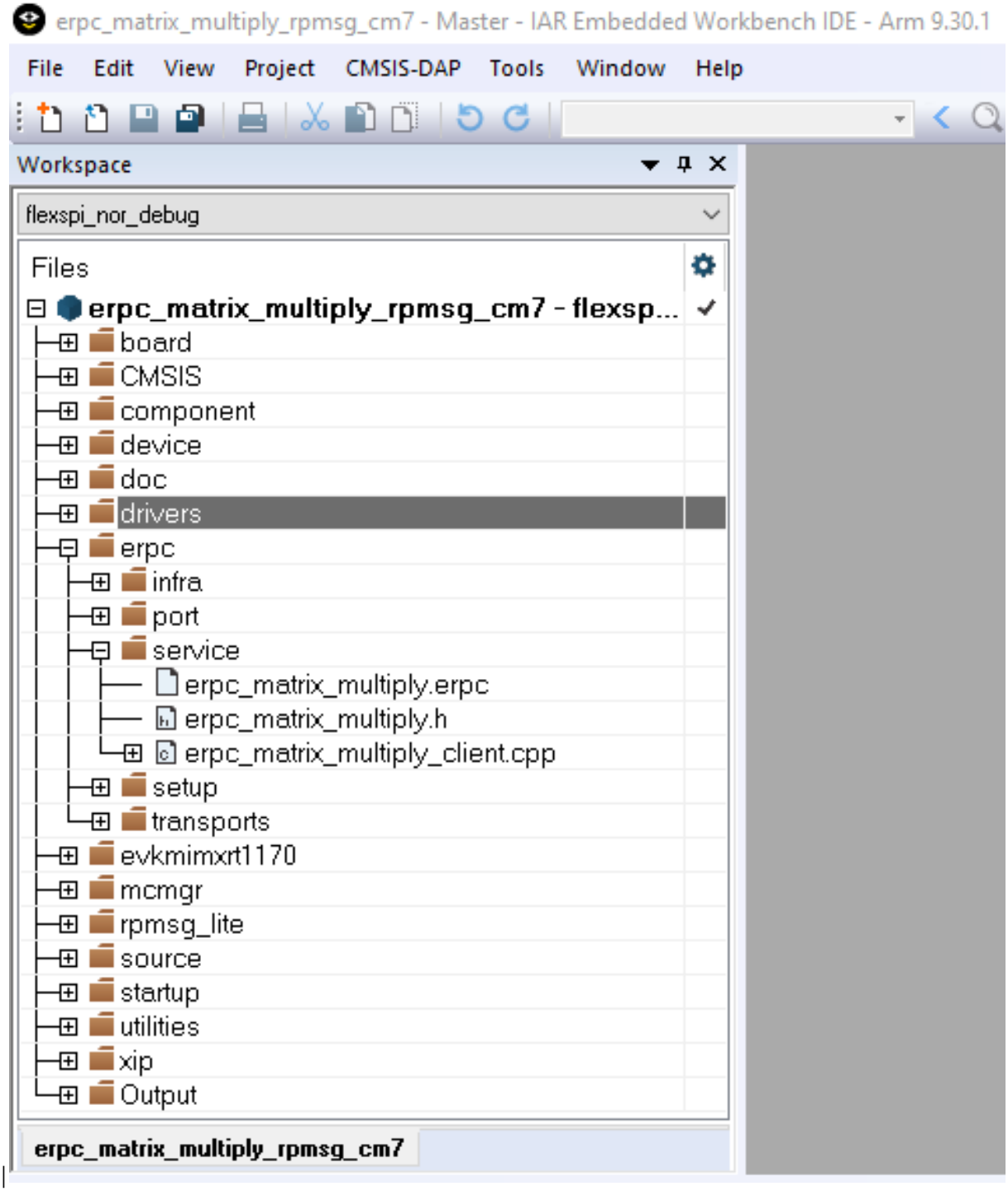
|

Parent topic: Multicore client application

Client-related generated files The client-related generated files are:

- erpc_matric_multiply.h
- erpc_matrix_multiply_client.cpp

These files contain the shim code for the functions and data types declared in the IDL file. These functions also call methods for codec initialization, data serialization, performing eRPC requests, and de-serializing outputs into expected data structures (if return values are expected). These shim code files can be found in the *<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_common/erpc_matrix_multiply/service/* folder.



Parent topic: Multicore client application

Client infrastructure files The eRPC infrastructure files are located in the following folder:

<MCUXpressoSDK_install_dir>/middleware/multicore/erpc/erpc_c

The **erpc_c** folder contains files for creating eRPC client and server applications in the C/C++ language. These files are distributed into subfolders.

- The **infra** subfolder contains C++ infrastructure code used to build server and client applications.

- Two files, erpc_client_manager.h and erpc_client_manager.cpp, are used for managing the client-side application. The main purpose of the client files is to create, perform, and release eRPC requests.
- Three files (erpc_codec.hpp, erpc_basic_codec.hpp, and erpc_basic_codec.cpp) are used for codecs. Currently, the basic codec is the initial and only implementation of the codecs.
- erpc_common.h file is used for common eRPC definitions, typedefs, and enums.
- erpc_manually_constructed.hpp file is used for allocating static storage for the used objects.
- Message buffer files are used for storing serialized data: erpc_message_buffer.hpp and erpc_message_buffer.cpp.
- erpc_transport.hpp file defines the abstract interface for transport layer.

The **port** subfolder contains the eRPC porting layer to adapt to different environments.

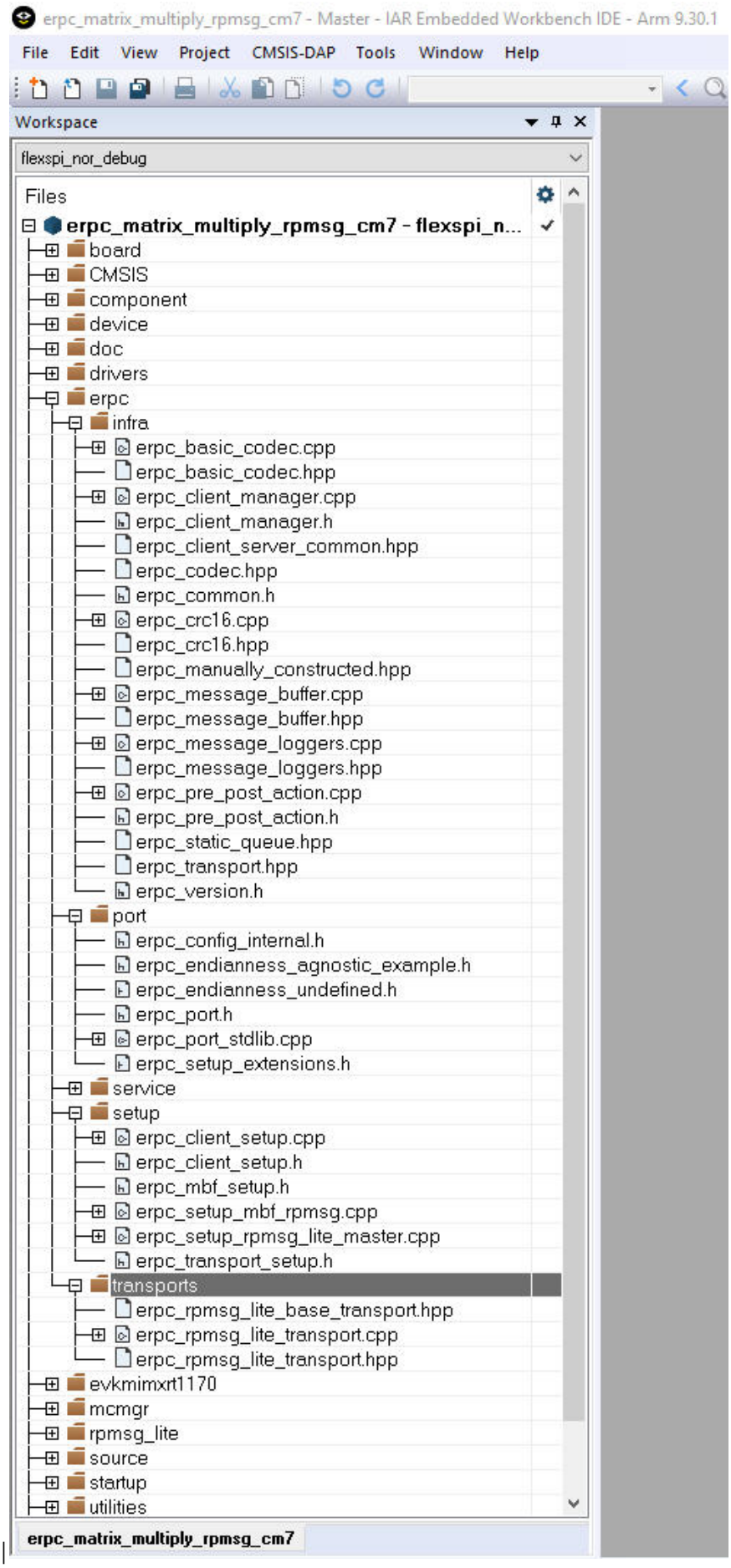
- erpc_port.h file contains definition of erpc_malloc() and erpc_free() functions.
- erpc_port_stdlib.cpp file ensures adaptation to stdlib.
- erpc_config_internal.h internal eRPC configuration file.

The **setup** subfolder contains a set of plain C APIs that wrap the C++ infrastructure, providing client and server init and deinit routines that greatly simplify eRPC usage in C-based projects. No knowledge of C++ is required to use these APIs.

- erpc_client_setup.h and erpc_client_setup.cpp files needs to be added into the "Matrix multiply" example project to demonstrate the use of C-wrapped functions in this example.
- erpc_transport_setup.h and erpc_setup_rpmsg_lite_master.cpp files needs to be added into the project in order to allow C-wrapped function for transport layer setup.
- erpc_mbf_setup.h and erpc_setup_mbf_rpmsg.cpp files needs to be added into the project in order to allow message buffer factory usage.

The **transports** subfolder contains transport classes for the different methods of communication supported by eRPC. Some transports are applicable only to host PCs, while others are applicable only to embedded or multicore systems. Most transports have corresponding client and server setup functions, in the setup folder.

- RPMsg-Lite is used as the transport layer for the communication between cores, erpc_rpmsg_lite_base_transport.hpp, erpc_rpmsg_lite_transport.hpp, and erpc_rpmsg_lite_transport.cpp files needs to be added into the client project.


erpc_matrix_multiply_rpmsg_cm7 - Master - IAR Embedded Workbench IDE - Arm 9.30.1
File Edit View Project CMSIS-DAP Tools Window Help
Workspace
flexspi_nor_debug
Files
erpc_matrix_multiply_rpmsg_cm7 - flexspi_n...
board
CMSIS
component
device
doc
drivers
erpc
infra
erpc_basic_codec.cpp
erpc_basic_codec.hpp
erpc_client_manager.cpp
erpc_client_manager.h
erpc_client_server_common.hpp
erpc_codec.hpp
erpc_common.h
erpc_crc16.cpp
erpc_crc16.hpp
erpc_manually_constructed.hpp
erpc_message_buffer.cpp
erpc_message_buffer.hpp
erpc_message_loggers.cpp
erpc_message_loggers.hpp
erpc_pre_post_action.cpp
erpc_pre_post_action.h
erpc_static_queue.hpp
erpc_transport.hpp
erpc_version.h
port
erpc_config_internal.h
erpc_endianness_agnostic_example.h
erpc_endianness_undefined.h
erpc_port.h
erpc_port_stdlib.cpp
erpc_setup_extensions.h
service
setup
erpc_client_setup.cpp
erpc_client_setup.h
erpc_mbf_setup.h
erpc_setup_mbf_rpmsg.cpp
erpc_setup_rpmsg_lite_master.cpp
erpc_transport_setup.h
transports
erpc_rpmsg_lite_base_transport.hpp
erpc_rpmsg_lite_transport.cpp
erpc_rpmsg_lite_transport.hpp
evkmimxrt1170
mcmgr
rpmsg_lite
source
startup
utilities
erpc_matrix_multiply_rpmsg_cm7

|

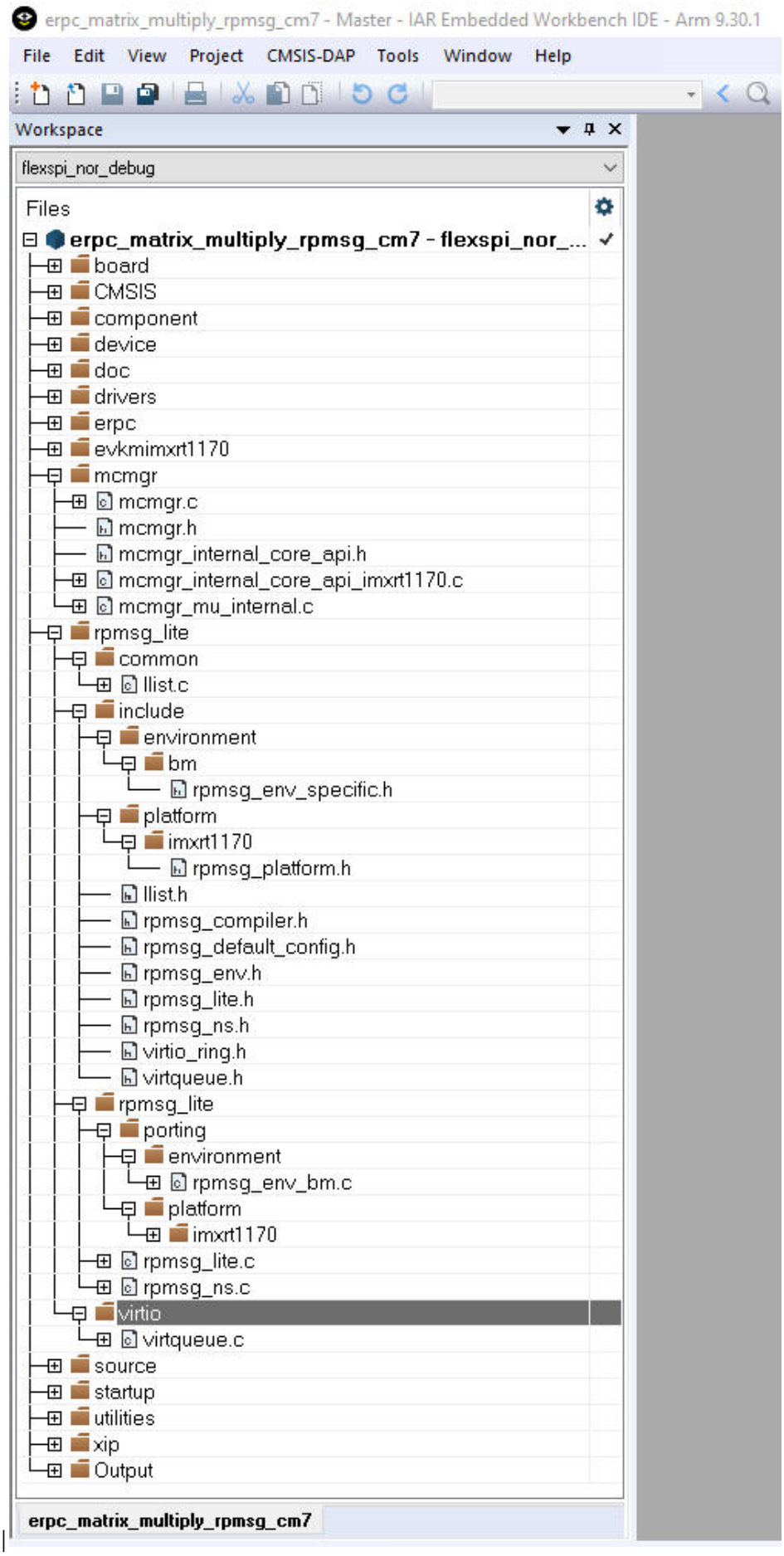
Parent topic: Multicore client application

Client multicore infrastructure files Because of the RPMsg-Lite (transport layer), it is also necessary to include RPMsg-Lite related files, which are in the following folder:

<MCUXpressoSDK_install_dir>/middleware/multicore/rpmsg_lite/

The multicore example applications also use the Multicore Manager software library to control the secondary core startup and shutdown. These source files are located in the following folder:

<MCUXpressoSDK_install_dir>/middleware/multicore/mcmgr/


erpc_matrix_multiply_rpmsg_cm7 - Master - IAR Embedded Workbench IDE - Arm 9.30.1
File Edit View Project CMSIS-DAP Tools Window Help
Workspace
flexspi_nor_debug
Files
erpc_matrix_multiply_rpmsg_cm7 - flexspi_nor_...
board
CMSIS
component
device
doc
drivers
erpc
evkmimxrt1170
mcmgr
mcmgr.c
mcmgr.h
mcmgr_internal_core_api.h
mcmgr_internal_core_api_imxrt1170.c
mcmgr_mu_internal.c
rpmsg_lite
common
llist.c
include
environment
bm
rpmsg_env_specific.h
platform
imxrt1170
rpmsg_platform.h
llist.h
rpmsg_compiler.h
rpmsg_default_config.h
rpmsg_env.h
rpmsg_lite.h
rpmsg_ns.h
virtio_ring.h
virtqueue.h
rpmsg_lite
porting
environment
rpmsg_env_bm.c
platform
imxrt1170
rpmsg_lite.c
rpmsg_ns.c
virtio
virtqueue.c
source
startup
utilities
xip
Output
erpc_matrix_multiply_rpmsg_cm7

|

Parent topic: Multicore client application

Client user code The client's user code is stored in the main_core0.c file, located in the following folder:

<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_example/erpc_matrix_multiply_rpmsg/cm7

The main_core0.c file contains the code for target board and eRPC initialization.

- After initialization, the secondary core is released from reset.
- When the secondary core is ready, the primary core initializes two matrix variables.
- The erpcMatrixMultiply eRPC function is called to issue the eRPC request and get the result.

It is possible to write the application-specific eRPC error handler. The eRPC error handler of the matrix multiply application is implemented in erpc_error_handler.h and erpc_error_handler.cpp files.

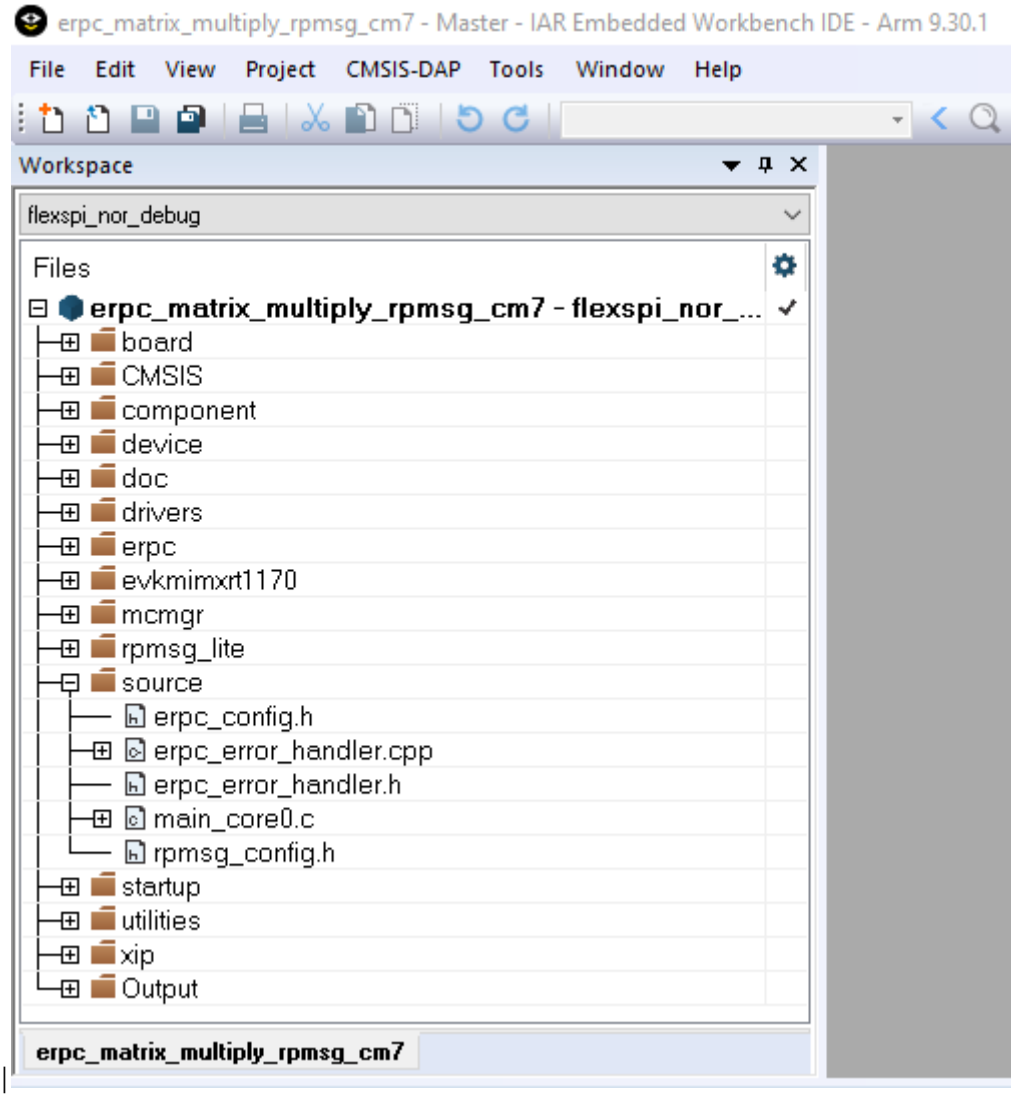
The matrix multiplication can be issued repeatedly, when pressing a software board button.

The eRPC-relevant code is captured in the following code snippet:

```
...
extern bool g_erpc_error_occurred;
...
/* Declare matrix arrays */
Matrix matrix1 = {0}, matrix2 = {0}, result_matrix = {0};
...
/* RPMsg-Lite transport layer initialization */
erpc_transport_t transport;
transport = erpc_transport_rpmsg_lite_master_init(src, dst,
ERPC_TRANSPORT_RPMSG_LITE_LINK_ID);
...
/* MessageBufferFactory initialization */
erpc_mbf_t message_buffer_factory;
message_buffer_factory = erpc_mbf_rpmsg_init(transport);
...
/* eRPC client side initialization */
erpc_client_t client;
client = erpc_client_init(transport, message_buffer_factory);
...
/* Set default error handler */
erpc_client_set_error_handler(client, erpc_error_handler);
...
while (1)
{
  /* Invoke the erpcMatrixMultiply function */
  erpcMatrixMultiply(matrix1, matrix2, result_matrix);
  ...
  /* Check if some error occured in eRPC */
  if (g_erpc_error_occurred)
  {
    /* Exit program loop */
    break;
  }
  ...
}
```

Except for the application main file, there are configuration files for the RPMsg-Lite (rpmsg_config.h) and eRPC (erpc_config.h), located in the following folder:

<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_matrix_multiply_rpmsg



Parent topic: Multicore client application

Parent topic:*Create an eRPC application*

Multiprocessor server application The "Matrix multiply" eRPC server project for multiprocessor applications is located in the *<MCUXpressoSDK_install_dir»/boards/<board_name>/multiprocessor_examples/erpc_server_matrix_multiply_<transport_layer>* folder.

Most of the multiprocessor application setup is the same as for the multicore application. The multiprocessor server application requires server-related generated files (server shim code), server infrastructure files, and the server user code. There is no need for server multicore infrastructure files (MCMGR and RPMsg-Lite). The RPMsg-Lite transport layer is replaced either by SPI or UART transports. The following table shows the required transport-related files per each transport type.

|SPI|<eRPC base directory>/erpc_c/setup/erpc_setup_(d)spi_slave.cpp

<eRPC base directory>/erpc_c/transports/erpc_(d)spi_slave_transport.hpp

<eRPC base directory>/erpc_c/transports/erpc_(d)spi_slave_transport.cpp

| |UART|<eRPC base directory>/erpc_c/setup/erpc_setup_uart_cmsis.cpp

<eRPC base directory>/erpc_c/transports/erpc_uart_cmsis_transport.hpp

<eRPC base directory>/erpc_c/transports/erpc_uart_cmsis_transport.cpp

|

Server user code The server's user code is stored in the main_server.c file, located in the *<MCUXpressoSDK_install_dir>/boards/<board_name>/multiprocessor_examples/erpc_server_matrix_multiply_<transport_layer>/* folder.

The eRPC-relevant code with UART as a transport is captured in the following code snippet:

```
/* erpcMatrixMultiply function user implementation */
void erpcMatrixMultiply(Matrix matrix1, Matrix matrix2, Matrix result_matrix)
{
  ...
}
int main()
{
  ...
  /* UART transport layer initialization, ERPC_DEMO_UART is the structure of CMSIS UART driver␣
↪operations */
  erpc_transport_t transport;
  transport = erpc_transport_cmsis_uart_init((void *)&ERPC_DEMO_UART);
  ...
  /* MessageBufferFactory initialization */
  erpc_mbf_t message_buffer_factory;
  message_buffer_factory = erpc_mbf_dynamic_init();
  ...
  /* eRPC server side initialization */
  erpc_server_t server;
  server = erpc_server_init(transport, message_buffer_factory);
  ...
  /* Adding the service to the server */
  erpc_service_t service = create_MatrixMultiplyService_service();
  erpc_add_service_to_server(server, service);
  ...
  while (1)
  {
    /* Process eRPC requests */
    erpc_status_t status = erpc_server_poll(server)
    /* handle error status */
    if (status != kErpcStatus_Success)
    {
      /* print error description */
      erpc_error_handler(status, 0);
      ...
    }
    ...
  }
}
```

Parent topic: Multiprocessor server application

Multiprocessor client application The "Matrix multiply" eRPC client project for multiprocessor applications is located in the *<MCUXpressoSDK_install_dir>/boards/<board_name>/multiprocessor_examples/erpc_client_matrix_multiply_<transport_layer>/iar/* folder.

Most of the multiprocessor application setup is the same as for the multicore application. The multiprocessor server application requires client-related generated files (server shim code),

client infrastructure files, and the client user code. There is no need for client multicore infrastructure files (MCMGR and RPMsg-Lite). The RPMsg-Lite transport layer is replaced either by SPI or UART transports. The following table shows the required transport-related files per each transport type.

|SPI|<eRPC base directory>/erpc_c/setup/erpc_setup_(d)spi_master.cpp

<eRPC base directory>/erpc_c/transports/ erpc_(d)spi_master_transport.hpp

<eRPC base directory>/erpc_c/transports/ erpc_(d)spi_master_transport.cpp

| |UART|<eRPC base directory>/erpc_c/setup/erpc_setup_uart_cmsis.cpp

<eRPC base directory>/erpc_c/transports/erpc_uart_cmsis_transport.hpp

<eRPC base directory>/erpc_c/transports/erpc_uart_cmsis_transport.cpp

|

Client user code The client's user code is stored in the main_client.c file, located in the *<MCUXpressoSDK_install_dir>/boards/<board_name>/multiprocessor_examples/erpc_client_matrix_multiply_<transport_layer>/* folder.

The eRPC-relevant code with UART as a transport is captured in the following code snippet:

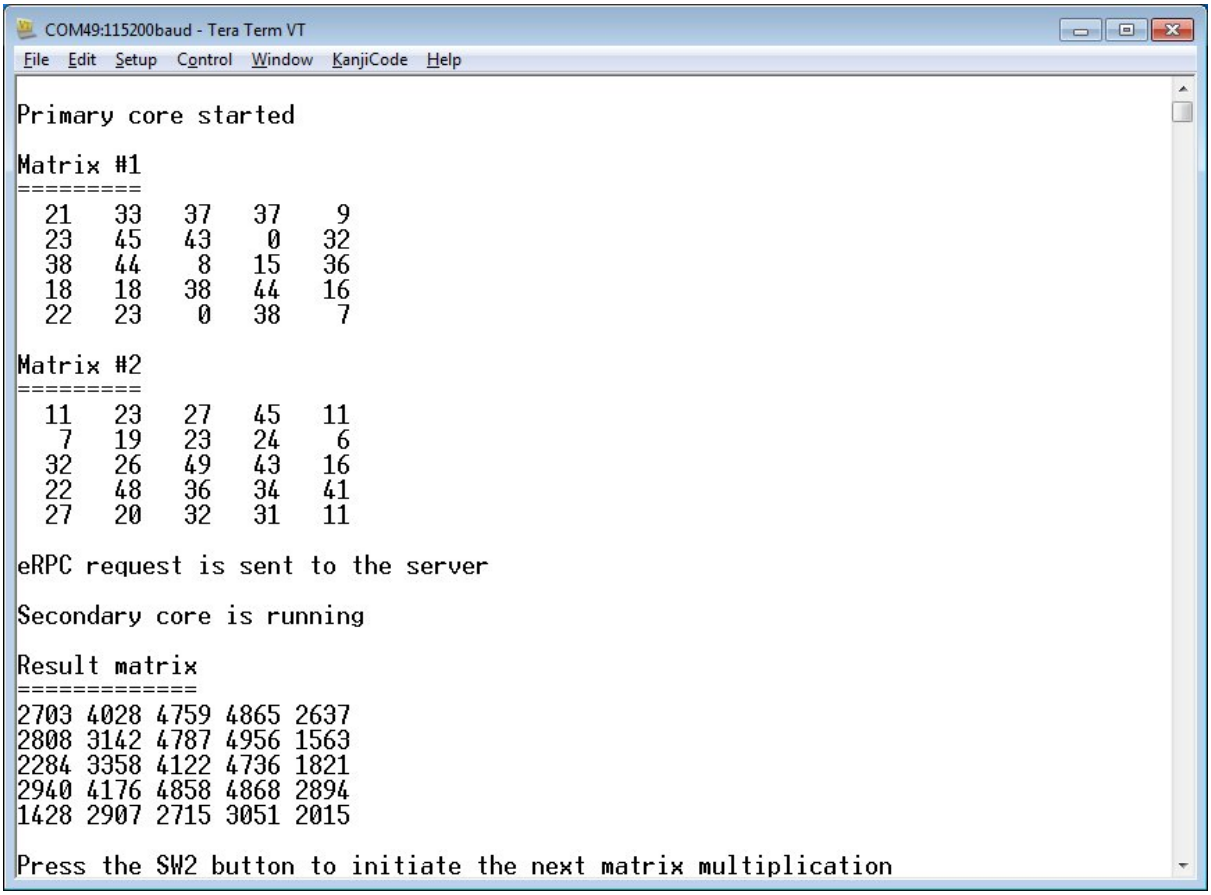
```
...
extern bool g_erpc_error_occurred;
...
/* Declare matrix arrays */
Matrix matrix1 = {0}, matrix2 = {0}, result_matrix = {0};
...
/* UART transport layer initialization, ERPC_DEMO_UART is the structure of CMSIS UART driver␣
↪operations */
erpc_transport_t transport;
transport = erpc_transport_cmsis_uart_init((void *)&ERPC_DEMO_UART);
...
/* MessageBufferFactory initialization */
erpc_mbf_t message_buffer_factory;
message_buffer_factory = erpc_mbf_dynamic_init();
...
/* eRPC client side initialization */
erpc_client_t client;
client = erpc_client_init(transport,message_buffer_factory);
...
/* Set default error handler */
erpc_client_set_error_handler(client, erpc_error_handler);
...
while (1)
{
  /* Invoke the erpcMatrixMultiply function */
  erpcMatrixMultiply(matrix1, matrix2, result_matrix);
  ...
  /* Check if some error occured in eRPC */
  if (g_erpc_error_occurred)
  {
    /* Exit program loop */
    break;
  }
  ...
}
```

Parent topic: Multiprocessor client application

Parent topic: Multiprocessor server application

Parent topic:*Create an eRPC application*

Running the eRPC application Follow the instructions in *Getting Started with MCUXpresso SDK* (document MCUXSDKGSUG) (located in the *<MCUXpressoSDK_install_dir>/docs* folder), to load both the primary and the secondary core images into the on-chip memory, and then effectively debug the dual-core application. After the application is running, the serial console should look like:



For multiprocessor applications that are running between PC and the target evaluation board or between two boards, follow the instructions in the accompanied example readme files that provide details about the proper board setup and the PC side setup (Python).

Parent topic:*Create an eRPC application*

Parent topic:*eRPC example*

Other uses for an eRPC implementation The eRPC implementation is generic, and its use is not limited to just embedded applications. When creating an eRPC application outside the embedded world, the same principles apply. For example, this manual can be used to create an eRPC application for a PC running the Linux operating system. Based on the used type of transport medium, existing transport layers can be used, or new transport layers can be implemented.

For more information and erpc updates see the github.com/EmbeddedRPC.

Note about the source code in the document Example code shown in this document has the following copyright and BSD-3-Clause license:

Copyright 2024 NXP Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Changelog eRPC All notable changes to this project will be documented in this file.

The format is based on Keep a Changelog, and this project adheres to Semantic Versioning.

Unreleased

1.14.0

Added

- Added Cmake/Kconfig support.
- Made java code jdk11 compliant, GitHub PR #432.
- Added imxrt1186 support into mu transport layer.
- erpcgen: Added assert for listType before usage, GitHub PR #406.

Fixed

- eRPC: Sources reformatted.
- erpc: Fixed typo in semaphore get (mutex -> semaphore), and write it can fail in case of timeout, GitHub PR #446.
- erpc: Free the arbitrated client token from client manager, GitHub PR #444.
- erpc: Fixed Makefile, install the erpc_simple_server header, GitHub PR #447.
- erpc_python: Fixed possible AttributeError and OSError on calling TCPTransport.close(), GitHub PR #438.
- Examples and tests consolidated.

1.13.0

Added

- erpc: Add BSD-3 license to endianness agnostic files, GitHub PR #417.
- eRPC: Add new Zephyr-related transports (zephyr_uart, zephyr_mbox).
- eRPC: Add new Zephyr-related examples.

Fixed

- eRPC,erpcgen: Fixing/improving markdown files, GitHub PR #395.
- eRPC: Fix Python client TCPTransports not being able to close, GitHub PR #390.
- eRPC,erpcgen: Align switch brackets, GitHub PR #396.
- erpc: Fix zephyr uart transport, GitHub PR #410.
- erpc: UART ZEPHYR Transport stop to work after a few transactions when using USB-CDC resolved, GitHub PR #420.

Removed

- eRPC,erpcgen: Remove cstbool library, GitHub PR #403.

1.12.0

Added

- eRPC: Add dynamic/static option for transport init, GitHub PR #361.
- eRPC,erpcgen: Winsock2 support, GitHub PR #365.
- eRPC,erpcgen: Feature/support multiple clients, GitHub PR #271.
- eRPC,erpcgen: Feature/buffer head - Framed transport header data stored in MessageBuffer, GitHub PR #378.
- eRPC,erpcgen: Add experimental Java support.

Fixed

- eRPC: Fix receive error value for spidev, GitHub PR #363.
- eRPC: UartTransport::init adaptation to changed driver.
- eRPC: Fix typo in assert, GitHub PR #371.
- eRPC,erpcgen: Move enums to enum classes, GitHub PR #379.
- eRPC: Fixed rpmsg tty transport to work with serial transport, GitHub PR #373.

1.11.0

Fixed

- eRPC: Makefiles update, GitHub PR #301.
- eRPC: Resolving warnings in Python, GitHub PR #325.
- eRPC: Python3.8 is not ready for usage of typing.Any type, GitHub PR #325.
- eRPC: Improved codec function to use reference instead of address, GitHub PR #324.

- eRPC: Fix NULL check for pending client creation, GitHub PR #341.
- eRPC: Replace sprintf with snprintf, GitHub PR #343.
- eRPC: Use MU_SendMsg blocking call in MU transport.
- eRPC: New LPSPI and LPI2C transport layers.
- eRPC: Freeing static objects, GitHub PR #353.
- eRPC: Fixed casting in deinit functions, GitHub PR #354.
- eRPC: Align LIBUSBSIO.GetNumPorts API use with libusbsio python module v. 2.1.11.
- erpcgen: Renamed temp variable to more generic one, GitHub PR #321.
- erpcgen: Add check that string read is not more than max length, GitHub PR #328.
- erpcgen: Move to g++ in pytest, GitHub PR #335.
- erpcgen: Use build=release for make, GitHub PR #334.
- erpcgen: Removed boost dependency, GitHub PR #346.
- erpcgen: Mingw support, GitHub PR #344.
- erpcgen: VS build update, GitHub PR #347.
- erpcgen: Modified name for common types macro scope, GitHub PR #337.
- erpcgen: Fixed memcpy for template, GitHub PR #352.
- eRPC,erpcgen: Change default build target to release + adding artefacts, GitHub PR #334.
- eRPC,erpcgen: Remove redundant includes, GitHub PR #338.
- eRPC,erpcgen: Many minor code improvements, GitHub PR #323.

1.10.0

Fixed

- eRPC: MU transport layer switched to blocking MU_SendMsg() API use.

1.10.0

Added

- eRPC: Add TCP_NODELAY option to python, GitHub PR #298.

Fixed

- eRPC: MUTransport adaptation to new supported SoCs.
- eRPC: Simplifying CI with installing dependencies using shell script, GitHub PR #267.
- eRPC: Using event for waiting for sock connection in TCP python server, formatting python code, C specific includes, GitHub PR #269.
- eRPC: Endianness agnostic update, GitHub PR #276.
- eRPC: Assertion added for functions which are returning status on freeing memory, GitHub PR #277.
- eRPC: Fixed closing arbitrator server in unit tests, GitHub PR #293.
- eRPC: Makefile updated to reflect the correct header names, GitHub PR #295.

- eRPC: Compare value length to used length() in reading data from message buffer, GitHub PR #297.
- eRPC: Replace EXPECT_TRUE with EXPECT_EQ in unit tests, GitHub PR #318.
- eRPC: Adapt rpmsg_lite based transports to changed rpmsg_lite_wait_for_link_up() API parameters.
- eRPC, erpcgen: Better distuingish which file can and cannot by linked by C linker, GitHub PR #266.
- eRPC, erpcgen: Stop checking if pointer is NULL before sending it to the erpc_free function, GitHub PR #275.
- eRPC, erpcgen: Changed api to count with more interfaces, GitHub PR #304.
- erpcgen: Check before reading from heap the buffer boundaries, GitHub PR #287.
- erpcgen: Several fixes for tests and CI, GitHub PR #289.
- erpcgen: Refactoring erpcgen code, GitHub PR #302.
- erpcgen: Fixed assigning const value to enum, GitHub PR #309.
- erpcgen: Enable runTesttest_enumErrorCode_allDirection, serialize enums as int32 instead of uint32.

1.9.1

Fixed

- eRPC: Construct the USB CDC transport, rather than a client, GitHub PR #220.
- eRPC: Fix premature import of package, causing failure when attempting installation of Python library in a clean environment, GitHub PR #38, #226.
- eRPC: Improve python detection in make, GitHub PR #225.
- eRPC: Fix several warnings with deprecated call in pytest, GitHub PR #227.
- eRPC: Fix freeing union members when only default need be freed, GitHub PR #228.
- eRPC: Fix making test under Linux, GitHub PR #229.
- eRPC: Assert costumizing, GitHub PR #148.
- eRPC: Fix corrupt clientList bug in TransportArbitrator, GitHub PR #199.
- eRPC: Fix build issue when invoking g++ with -Wno-error=free-nonheap-object, GitHub PR #233.
- eRPC: Fix inout cases, GitHub PR #237.
- eRPC: Remove ERPC_PRE_POST_ACTION dependency on return type, GitHub PR #238.
- eRPC: Adding NULL to ptr when codec function failed, fixing memcpy when fail is present during deserialization, GitHub PR #253.
- eRPC: MessageBuffer usage improvement, GitHub PR #258.
- eRPC: Get rid for serial and enum34 dependency (enum34 is in python3 since 3.4 (from 2014)), GitHub PR #247.
- eRPC: Several MISRA violations addressed.
- eRPC: Fix timeout for Freertos semaphore, GitHub PR #251.
- eRPC: Use of rpmsg_lite_wait_for_link_up() in rpmsg_lite based transports, GitHub PR #223.
- eRPC: Fix codec nullptr dereferencing, GitHub PR #264.

- erpcgen: Fix two syntax errors in erpcgen Python output related to non-encapsulated unions, improved test for union, GitHub PR #206, #224.
- erpcgen: Fix serialization of list/binary types, GitHub PR #240.
- erpcgen: Fix empty list parsing, GitHub PR #72.
- erpcgen: Fix templates for malloc errors, GitHub PR #110.
- erpcgen: Get rid of encapsulated union declarations in global scale, improve enum usage in unions, GitHub PR #249, #250.
- erpcgen: Fix compile error:UniqueIdChecker.cpp:156:104:'sort' was not declared, GitHub PR #265.

1.9.0

Added

- eRPC: Allow used LIBUSBSIO device index being specified from the Python command line argument.

Fixed

- eRPC: Improving template usage, GitHub PR #153.
- eRPC: run_clang_format.py cleanup, GitHub PR #177.
- eRPC: Build TCP transport setup code into liberpc, GitHub PR #179.
- eRPC: Fix multiple definitions of g_client error, GitHub PR #180.
- eRPC: Fix memset past end of buffer in erpc_setup_mbf_static.cpp, GitHub PR #184.
- eRPC: Fix deprecated error with newer pytest version, GitHub PR #203.
- eRPC, erpcgen: Static allocation support and usage of rpmsg static FreeRTOSs related APi, GitHub PR #168, #169.
- erpcgen: Remove redundant module imports in erpcgen, GitHub PR #196.

1.8.1

Added

- eRPC: New i2c_slave_transport trasnport introduced.

Fixed

- eRPC: Fix misra erpc c, GitHub PR #158.
- eRPC: Allow conditional compilation of message_loggers and pre_post_action.
- eRPC: (D)SPI slave transports updated to avoid busy loops in rtos environments.
- erpcgen: Re-implement EnumMember::hasValue(), GitHub PR #159.
- erpcgen: Fixing several misra issues in shim code, erpcgen and unit tests updated, GitHub PR #156.
- erpcgen: Fix bison file, GitHub PR #156.

1.8.0

Added

- eRPC: Support win32 thread, GitHub PR #108.
- eRPC: Add mbed support for malloc() and free(), GitHub PR #92.
- eRPC: Introduced pre and post callbacks for eRPC call, GitHub PR #131.
- eRPC: Introduced new USB CDC transport.
- eRPC: Introduced new Linux spidev-based transport.
- eRPC: Added formatting extension for VSC, GitHub PR #134.
- erpcgen: Introduce ustring type for unsigned char and force cast to char*, GitHub PR #125.

Fixed

- eRPC: Update makefile.
- eRPC: Fixed warnings and error with using MessageLoggers, GitHub PR #127.
- eRPC: Extend error msg for python server service handle function, GitHub PR #132.
- eRPC: Update CMSIS UART transport layer to avoid busy loops in rtos environments, introduce semaphores.
- eRPC: SPI transport update to allow usage without handshaking GPIO.
- eRPC: Native _WIN32 erpc serial transport and threading.
- eRPC: Arbitrator deadlock fix, TCP transport updated, TCP setup functions introduced, GitHub PR #121.
- eRPC: Update of matrix_multiply.py example: Add –serial and –baud argument, GitHub PR #137.
- eRPC: Update of .clang-format, GitHub PR #140.
- eRPC: Update of erpc_framed_transport.cpp: return error if received message has zero length, GitHub PR #141.
- eRPC, erpcgen: Fixed error messages produced by -Wall -Wextra -Wshadow -pedantic-errors compiler flags, GitHub PR #136, #139.
- eRPC, erpcgen: Core re-formatted using Clang version 10.
- erpcgen: Enable deallocation in server shim code when callback/function pointer used as out parameter in IDL.
- erpcgen: Removed '$' character from generated symbol name in '_$union' suffix, GitHub PR #103.
- erpcgen: Resolved mismatch between C++ and Python for callback index type, GitHub PR #111.
- erpcgen: Python generator improvements, GitHub PR #100, #118.
- erpcgen: Fixed error messages produced by -Wall -Wextra -Wshadow -pedantic-errors compiler flags, GitHub PR #136.

1.7.4

Added

- eRPC: Support MU transport unit testing.
- eRPC: Adding mbed os support.

Fixed

- eRPC: Unit test code updated to handle service add and remove operations.
- eRPC: Several MISRA issues in rpmsg-based transports addressed.
- eRPC: Fixed Linux/TCP acceptance tests in release target.
- eRPC: Minor documentation updates, code formatting.
- erpcgen: Whitespace removed from C common header template.

1.7.3

Fixed

- eRPC: Improved the test_callbacks logic to be more understandable and to allow requested callback execution on the server side.
- eRPC: TransportArbitrator::prepareClientReceive modified to avoid incorrect return value type.
- eRPC: The ClientManager and the ArbitratedClientManager updated to avoid performing client requests when the previous serialization phase fails.
- erpcgen: Generate the shim code for destroy of statically allocated services.

1.7.2

Added

- eRPC: Add missing doxygen comments for transports.

Fixed

- eRPC: Improved support of const types.
- eRPC: Fixed Mac build.
- eRPC: Fixed serializing python list.
- eRPC: Documentation update.

1.7.1

Fixed

- eRPC: Fixed semaphore in static message buffer factory.
- erpcgen: Fixed MU received error flag.
- erpcgen: Fixed tcp transport.

1.7.0

Added

- eRPC: List names are based on their types. Names are more deterministic.
- eRPC: Service objects are as a default created as global static objects.
- eRPC: Added missing doxygen comments.
- eRPC: Added support for 64bit numbers.
- eRPC: Added support of program language specific annotations.

Fixed

- eRPC: Improved code size of generated code.
- eRPC: Generating crc value is optional.
- eRPC: Fixed CMSIS Uart driver. Removed dependency on KSDK.
- eRPC: Forbid users use reserved words.
- eRPC: Removed outByref for function parameters.
- eRPC: Optimized code style of callback functions.

1.6.0

Added

- eRPC: Added @nullable support for scalar types.

Fixed

- eRPC: Improved code size of generated code.
- eRPC: Improved eRPC nested calls.
- eRPC: Improved eRPC list length variable serialization.

1.5.0

Added

- eRPC: Added support for unions type non-wrapped by structure.
- eRPC: Added callbacks support.
- eRPC: Added support @external annotation for functions.
- eRPC: Added support @name annotation.
- eRPC: Added Messaging Unit transport layer.
- eRPC: Added RPMSG Lite RTOS TTY transport layer.
- eRPC: Added version verification and IDL version verification between eRPC code and eRPC generated shim code.
- eRPC: Added support of shared memory pointer.

- eRPC: Added annotation to forbid generating const keyword for function parameters.
- eRPC: Added python matrix multiply example.
- eRPC: Added nested call support.
- eRPC: Added struct member “byref” option support.
- eRPC: Added support of forward declarations of structures
- eRPC: Added Python RPMsg Multiendpoint kernel module support
- eRPC: Added eRPC sniffer tool

1.4.0

Added

- eRPC: New RPMsg-Lite Zero Copy (RPMsgZC) transport layer.

Fixed

- eRPC: win_flex_bison.zip for windows updated.
- eRPC: Use one codec (instead of inCodec outCodec).

[1.3.0]

Added

- eRPC: New annotation types introduced (@length, @max_length, ...).
- eRPC: Support for running both erpc client and erpc server on one side.
- eRPC: New transport layers for (LP)UART, (D)SPI.
- eRPC: Error handling support.

[1.2.0]

Added

- eRPC source directory organization changed.
- Many eRPC improvements.

[1.1.0]

Added

- Multicore SDK 1.1.0 ported to KSDK 2.0.0.

[1.0.0]

Added

- Initial Release

2.8 Multimedia

2.8.1 Audio Voice

Audio Voice Components

MCUXpresso SDK : audio-voice-components

Overview This repository is for MCUXpresso SDK audio-voice-components 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 Audio Voice Components - 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.

Overview This repository allows users to add additional functionality to the Maestro Audio framework. This structure is designed for integration with Maestro and is not intended for standalone use. For information on the use of individual components, please refer to the Maestro programmer's guide.

This repository acts as Zephyr module, to be able to use these libraries in Zephyr build system.

Content

- *asrc* - Libraries and public files of Asynchronous Sample Rate Converter, version 1.0.0
- *ssrc* - Libraries and public files of Synchronous Sample Rate Converter, version 1.0.0
- *opus* - Source files of Opus decoder and encoder, version 1.3.1
- *opusfile* - Source files for Opus streams in the Ogg container, version 0.12
- *ogg* - Source files of Ogg container, version 1.3.5
- *decoders* - Libraries and public files of following audio decoders:
  - *aac* - AAC decoder, version 1.0.0
  - *flac* - FLAC decoder, version 1.0.0
  - *mp3* - MP3 decoder, version 1.0.0
  - *wav* - WAV decoder, version 1.0.0
- zephyr/ - Files allowing usage of the libraries in Zephyr build

Following table contains information about libraries and source files availability:

Asynchronous Sample Rate Converter The Asynchronous Sample Rate Converter (ASRC) software module compensates the drift between two mono audio signals. This is not a frequency converter and so the nominal signal frequency is the same before and after the ASRC. More details about ASRC are available in the User Guide, which is located in asrc\doc\.

Synchronous Sample Rate Converter The Synchronous Sample Rate Converter (SSRC) software module converts an audio signal (mono or stereo) with a certain sampling frequency to an audio signal with another sampling frequency. More details about SSRC are available in the *User Guide*.

Opus For Opus decoder and encoder documentation please see following link: opus.

Opus File The Opus File provides a API for decoding and basic manipulation of Opus streams in Ogg container and depends on *Opus* and *Ogg* libraries. For Opus File documentation please see following link: opusfile.

Ogg Container For Ogg container documentation please see following link: ogg.

Decoders Each decoder contains libraries for supported processor and toolchain (see table above), corresponding Public API file and documentation folder.

AAC For decoder features please see *aacdec*, for API Usage please see *aacd_ug*.

FLAC For decoder features please see *flacdec*, for API Usage please see *flacd_ug*.

MP3 For decoder features please see *mp3dec*, for API Usage please see *mp3d_ug*.

WAV For decoder features please see *wavdec*, for API Usage please see *wavd_ug*.

Zephyr build To add library into the Zephyr build, add CONFIG_NXP_AUDIO_VOICE_COMPONENTS_* for specific libraries into your prj.conf. For all configuration options, see *zephyr/Kconfig*.

List of supported libraries in Zephyr:

- Decoders:
  - AAC
  - FLAC
  - MP3
  - FLAC
  - OPUS
- Encoders
  - OPUS

AAC decoder

AAC decoder features

- The AAC decoder implementation supports the following:
- Supported profile : AAC-LC
- Sampling rate : 8 kHz, 11.025 kHz, 12 kHz, 16 kHz, 22.05 kHz, 24 kHz, 32 kHz, 44.1 kHz, 48 kHz
- Channel : stereo and mono
- Bits per samples : 16 bit
- Container format : (MPEG-2 Style)AAC transport format - ADTS and ADIF.

Specification and reference

Performance

Memory information The memory usage of the decoder in bytes is:

- Code/flash = 26332 + 19264 = 45596
- Data/RAM = 26832

Section	Size
.text	26332
.ro & .const	19264
.bss	26832

CPU usage

- CPU core clock in MHz: 20.97.

Track type	Duration of track in second	Frame size in bytes	Performance MIPS of codec (in MHz)
48 kHz, stereo	38 s	4096	12.2 MHz

API Usage of AAC Decoder

Overview

- This section describes the integration steps to call AAC decoder APIs by the application code. During each step, the used data structures and functions are explained. All CCI public APIs are defined in aac_cci.h header file. This file is located at \decoders\aac.

Configuration

Build Options AAC Decoder library is built with the following defined/enabled macros.

- There is no macro or define used to build the AAC decoder.

Buffer Allocation

- The AAC decoder does not perform dynamic memory allocation. The application calls the function AACDecoderGetMemorySize() to get the decoder memory requirements. This function must be called before all other decoder functions are invoked.
- The application first gets the required memory size for the decoder, then allocates memory for the decoder structures. Structures contain Main Decoder parameters and decoder information parameters.
- This function populates the required memory for the decoder and returns the required memory size in bytes.

Initialization

- AACDecoderInit() function must be called before decode API. This API allocates the memory to decoder main structure and also initializes the decoder main structure parameters.
- It also registers the call back functions to the decoder, which is used by the decoder to read or seek the input stream.

Decoding

- AACDecoderDecode() function is main decoding API of the decoder. This API decodes the encoded input stream and fills the PCM output samples into decoder output PCM buffer.
- This API gives the information about the number of samples produced by the decoder and also provides the pointer to the decoder output PCM samples buffer.

Seeking

- AACDecoderSeek() function calculates the actual frame boundary align offset from the unalign seek offset and returns the actual seek offset. It also resets the decoder internal states and variables.

Callback Usage All the callback functions are assigned to the respective pointers before the codec initialization is called. Callback APIs are described below.

Read Callback API AAC Decoder read call back API reads the bytes from the input stream and fills them into decoder internal bit stream buffer. It returns the number of bytes read from the input stream.

Seek Callback API This call back API is for the seek operation.

Get File Position Callback API This call back API gives the current file position.

FLAC decoder

FLAC decoder features

- The FLAC decoder implementation support the following:
- Sampling rate: 8 kHz, 11.05 kHz, 12 kHz, 16 kHz, 22.05 kHz, 32 kHz, 44.1 kHz, and 48 kHz.
- Channel : stereo and mono
- Bits per samples : 16 bits

Specification and reference

Official website

- FLAC lossless audio codec is at https://xiph.org/flac.

Inbound licensing

- For licensing information please refer to FLAC's official website: https://xiph.org/flac/license.html.

Performance

Memory information The memory usage of the decoder in bytes is:

- Code/flash = 15744 + 2080 = 17824
- Data/RAM = 27936

Section	Size
.text	15744
.ro & .const	2080
.bss	27936

CPU usage

- Output frame size: 16384 bytes.
- CPU core clock in MHz: 20.97.

Track type	Duration of track in second	Performance MIPS of codec (in MHz)
48 kHz, stereo	76 s	30.7 MHz
32 kHz, stereo	76 s	20.3 MHz
8 kHz, stereo	37 s	5.34 MHz

Following test cases are performed:

- Audio format listening test
- Audio quality test

For all above test cases, test tracks are played through the end without any distortion, glitching, hanging, or crashing.

API Usage of FLAC Decoder

Overview

- This section describes the integration steps to call FLAC decoder APIs by the application code. During each step the used data structures and functions are explained. All cci public APIs are defined in flac_cci.h header file. This file is located at \decoders\flac\include.

Configuration

Build Options

- SUPPORT_16_BITS_ONLY :- This macro is used to enable 16bits per sample flac decoder.
- *ASM* :- This macro is used to enable ARM assembly macros for 24bits per sample flac decoder.

Buffer Allocation

- The FLAC decoder does not perform dynamic memory allocation. The application calls the function FLACDecoderGetMemorySize() to get the decoder memory requirements. This function must be called before all other decoder functions are invoked.
- The application first gets the required memory size for the decoder and then allocates memory for the decoder structures. Structures contain Main Decoder parameters and decoder information parameters.
- This function populates the required memory for the decoder and returns the required memory size in bytes.

Initialization

- FLACDecoderInit() function must be called before decode API. This API allocates the memory to decoder main structure and also initializes the decoder main structure parameters.
- It also registers the call back functions to the decoder, which will be used by decoder to read or to seek the input stream.

Decoding

- FLACDecoderDecode() function is main decoding API of the decoder. This API decodes the encoded input stream and fills the PCM output samples into decoder output PCM buffer.
- This API gives the information about the number of samples produced by the decoder and also provides the pointer to the decoder output PCM samples buffer.

Seeking

- FLACDecoderSeek() function calculates the actual frame boundary align offset from the unalign seek offset and returns the actual seek offset. It also resets the decoder internal states and variables.

Callback Usage All the callback functions will be assigned to the respective pointers before the codec initialization is called. Callback APIs are described below.

Read Callback API FLAC Decoder read call back API reads the bytes from the input stream and fills them into decoder internal bit stream buffer. It returns the number of bytes read from the input stream.

Seek Callback API This call back API is for the seek operation.

Get File Position Callback API This call back API gives the current file position.

MP3 decoder

MP3 decoder features

- MP3 decoder supports mpeg-1, mpeg-2, mpeg-2.5.
- All MP3 features supported , including joint stereo, mid-side stereo, intensity stereo, and dual channel.
- Supported sampling rate: 8 kHz, 11.025 kHz, 12 kHz, 16 kHz, 22.05 kHz, 24 kHz, 32 kHz, 44.1 kHz and 48 kHz.
- Supported channel: stereo and mono
- Supported bits per samples: 16 bit
- Supported bit rate: 8, 16, 24, 32, 40, 48, 56, 64, 80, 96, 112, 128, 144, 160, 176, 192, 224, 256, 320, 384, 416, and 448.

Performance

Memory information The memory usage of the decoder (data obtained from IAR compiler) in bytes is:

- Code/flash = 26884 + 18372 = 45256
- RAM = 16200

Section	Size
.text	26884
.ro & .const	18372
.bss	16200

CPU usage The performance of the decoder was measured using the real hardware platform (RT1060).

- CPU core clock in MHz: 600.

Track type	Duration of track in second	Frame size in bytes	Performance MIPS of codec (in MHz)
320 Kbps, 44.1 kHz, stereo	358 s	2304	~24 MHz
192 Kbps, 48 kHz, stereo	10 s	2304	~18 MHz

API Usage of MP3 Decoder

Overview

- This section describes the integration steps to call MP3 decoder APIs by the application code. During each step the used data structures and functions are explained. All cci public APIs are defined in mp3_cci.h header file. This file is located at \decoders\mp3.

Configuration

Build Options MP3 Decoder library is built with the following defined/enabled macros.

- There is no macro or define used to build the MP3 decoder.

Buffer Allocation

- The MP3 decoder does not perform dynamic memory allocation. The application calls the function MP3DecoderGetMemorySize() to get the decoder memory requirements. This function must be called before all other decoder functions are invoked.
- The application first gets the required memory size for the decoder and then allocates memory for the decoder structures. Structures contain Main Decoder parameters and decoder information parameters.
- This function populates the required memory for the decoder and returns the required memory size in bytes.

Initialization

- MP3DecoderInit() function must be called before decode API. This API allocates the memory to decoder main structure and also initializes the decoder main structure parameters.
- It also registers the call back functions to the decoder, which will be used by decoder to read or to seek the input stream.

Decoding

- MP3DecoderDecode() function is main decoding API of the decoder. This API decodes the encoded input stream and fills the PCM output samples into decoder output PCM buffer.
- This API gives the information about the number of samples produced by the decoder and also provides the pointer to the decoder output PCM samples buffer.

Seeking

- MP3DecoderSeek() function calculates the actual frame boundary align offset from the unalign seek offset and returns the actual seek offset. It also resets the decoder internal states and variables.

Callback Usage All the callback functions will be assigned to the respective pointers before the codec initialization is called. Callback APIs are described below.

Read Callback API MP3 Decoder read call back API reads the bytes from the input stream and fills them into decoder internal bit stream buffer. It returns the number of bytes read from the input stream.

Seek Callback API This call back API is for the seek operation.

Get File Position Callback API This call back API gives the current file position.

WAV decoder

WAV decoder features

- The WAV decoder implementation support the following:
- Sampling rate: 8 kHz, 11.025kHz, 16 kHz, 22.05 kHz, 32 kHz, 44.1 kHz, and 48 kHz.
- Channel: stereo and mono
- PCM format with 8/16/24 bits per sample.

Performance

Memory information The memory usage of the decoder in bytes is:

- Code/flash = 6260 + 342 = 6602
- Data/RAM = 16 + 20696 = 20712

Section	Size
.text	6260
.ro & .const	342
.bss	20696
.data	16

CPU usage The performance of the decoder was measured using the decoder standalone unit test.

- CPU core clock in MHz: 20.97 MHz.

Track type	Duration of track in second	Frame size in bytes	Performance MIPS of codec (in MHz)
48 kHz, stereo, PCM	12 s	4096	9.68 MHz

Following test cases were performed:

- Audio format listening test
- Audio quality test

For all above test cases, test tracks are played through the end without any distortion, glitching, hanging, or crashing.

API Usage of WAV Decoder

Overview

- This section describes the integration steps to call MP3 decoder APIs by the application code. During each step the used data structures and functions are explained. All cci public APIs are defined in wav_cci.h header file. This file is located at \decoders\wav.

Configuration

Build Options WAV Decoder library is built with the following defined/enabled macros.

- There is no macro or define used to build the WAV decoder.

Buffer Allocation

- The WAV decoder does not perform dynamic memory allocation. The application calls the function WAVDecoderGetMemorySize() to get the decoder memory requirements. This function must be called before all other decoder functions are invoked.
- The application first gets the required memory size for the decoder and then allocates memory for the decoder structures. Structures contain Main Decoder parameters and decoder information parameters.
- This function populates the required memory for the decoder and returns the required memory size in bytes.

Initialization

- WAVDecoderInit() function must be called before decode API. This API allocates the memory to decoder main structure and also initializes the decoder main structure parameters.
- It also registers the call back functions to the decoder, which will be used by decoder to read or to seek the input stream.

Decoding

- WAVDecoderDecode() function is main decoding API of the decoder. This API decodes the encoded input stream and fills the PCM output samples into decoder output PCM buffer.
- This API gives the information about the number of samples produced by the decoder and also provides the pointer to the decoder output PCM samples buffer.

Seeking

- WAVDecoderSeek() function calculates the actual frame boundary align offset from the unalign seek offset and returns the actual seek offset. It also resets the decoder internal states and variables.

Callback Usage All the callback functions will be assigned to the respective pointers before the codec initialization is called. Callback APIs are described below.

Read Callback API WAV Decoder read call back API reads the bytes from the input stream and fills them into decoder internal bit stream buffer. It returns the number of bytes read from the input stream.

Seek Callback API This call back API is for the seek operation.

Get File Position Callback API This call back API gives the current file position.

Synchronous Sample Rate Converter

Introduction The Synchronous Sample Rate Converter (SSRC) software module converts a mono or stereo audio signal with a certain sampling frequency to an audio signal with a different sampling frequency. The sample rate converter works synchronously, meaning that input and output sampling rates are exactly known for a mutual clock reference.

To accomplish a professional sampling conversion quality and minimal system footprint, the SRC SW module contains highly optimized components.

The SSRC module supports the following features.

- Multiple instances of the sample rate converter can run at the same time.
- Supported sampling frequencies: 32 kHz, 44.1 kHz, and 48 kHz plus the halves and the quarters of these three sample rates. The input and output sample rates are freely selectable out of the supported sampling rates
- Selectable Mono/Stereo Input/Output.
- Selectable quality level: high quality/ very high quality.

Acronyms *Table 1* lists the acronyms used in this document.

Acror	Description
Fs	Sampling Frequency
Fs-LOW(	Lowest sample rate used for the conversion **Note:** Input sample rate for up sampling and the output sample rate for down sampling
FsIN	Input sample rate
FsOU	Output sample rate
MIPS	Million Instructions Per Second
SSRC	Synchronous sample rate converter
THD+	Total Harmonic Distortion plus Noise **Note:** The THD+N is defined as the total power of the unwanted signal divided by the power of the wanted signal. The wanted signal is defined as a full scale, 1 kHz sine wave.

Parent topic:*Introduction*

Performance figures The Total Harmonic Distortion Plus Noise (THD+N) of the converted signals is below - 76 (high-quality mode) and - 85 (very high-quality mode) for signal frequencies below 0.45*FsLOW (=90 % of the Nyquist range of the lowest sample clock)

Table 1 and *Table 2* give the THD+N performance (FsIN on the vertical axis and FsOUT on the horizontal axis) for the two supported quality levels. The numbers in the tables give the worst-case THD+N measured for signal frequencies below 0.45*FsLOW. For each conversion ratio, 100 THD+N measurements were executed with signal frequencies linearly spread over the complete Nyquist range.

FsIN/ FsOUT	8000	11025	12000	16000	22050	24000	32000	44100	48000
8000	-92.1	-79.7	-80.1	-80.1	-79.6	-80.2	-79.4	-79.1	-79.2
11025	-79	-92.9	-80	-79.9	-80.2	-79.8	-79.9	-79.5	-78.9
12000	-79	-79.2	-92.7	-80.1	-79.8	-80.3	-79.8	-79.8	-79.5
16000	-81.7	-78.8	-80.2	-93	-78.3	-77.7	-78.3	-78.3	-77.9
22050	-77.5	-81.8	-78.2	-79	-93	-79.9	-79.8	-80.3	-79.9
24000	-77.4	-77.9	-81.2	-79.1	-79.2	-92.5	-80.1	-79.8	-79.9
32000	-81	-77.5	-78.9	-81.2	-78.7	-80.1	-92.9	-79.7	-79.2
44100	-79.1	-81.2	-76.7	-77.8	-82	-78.2	-79.1	-93	-79.7
48000	-78.7	-78.8	-81.1	-77.6	-77.9	-81.8	-79.1	-79.3	-93

FsIN/ FsOUT	8000	11025	12000	16000	22050	24000	32000	44100	48000
8000	-92.1	-86.6	-88.6	-91.5	-86.4	-89	-89.7	-89.3	-89.3
11025	-89.1	-92.9	-86.3	-86.3	-91.6	-86.3	-86.5	-89.7	-89.3
12000	-91.4	-88.4	-92.7	-89.6	-86.6	-91.5	-86.8	-86.6	-89.7
16000	-93.1	-88.4	-90.4	-93	-86.6	-88.8	-91.5	-86.5	-89.4
22050	-90.7	-93.5	-89.7	-89.3	-93	-86.5	-86.3	-91.5	-86.6
24000	-93.8	-90.5	-93.5	-91.7	-88.4	-92.5	-89.7	-86.6	-91.5
32000	-93.8	-91	-91.2	-93.3	-88.4	-90.5	-92.9	-86.7	-89
44100	-93.7	-93.6	-91.5	-90.6	-93.8	-89.8	-89.3	-93	-86.5
48000	-94.1	-92.6	-94	-94	-90.1	-93.7	-91.8	-88.4	-93

Parent topic:*Introduction*

Resource usage This section lists the memory and processing requirements for the SSRC module.

Memory requirements The following are the memory requirements for the SSRC module.

Memory item	Size in bytes
Instance memory (persistent)	548
Scratch memory (non-persistent)	15.536 *1*
Program memory for Arm9E and XScale	14k
Program memory for Arm7	15k

Parent topic: Resource usage

1 Worst case number for I/O buffers of 40 ms. If smaller I/O buffers are used, this number is smaller. The required scratch memory is roughly equal to 2 times the buffer size on the highest sample rate.

Processing requirements The following tables give the MIPS performance of the SSRC module. The cycles are measured with zero wait state memory and for I/O buffers of 40 ms.

Note: The user processing 32-bit processing must refer to the very high-quality MIPS results.

On Arm7 and Arm9

FsIN / FsOUT	8000	11025	12000	16000	22050	24000	32000	44100	48000
8000	0.13	4.77	5.17	1.84	6.75	7.33	3.55	9.1	9.89
11025	5.42	0.18	5.58	6.84	2.53	7.75	9.71	4.89	10.31
12000	5.85	6.39	0.2	7.01	8.97	2.76	9.89	12.94	5.32
16000	1.69	7.74	7.99	0.26	9.54	10.33	3.68	13.5	14.65
22050	7.2	2.33	10.09	10.83	0.36	11.17	13.67	5.07	15.49
24000	7.79	8.33	2.53	11.7	12.78	0.39	14.03	17.94	5.51
32000	3.12	10.32	10.58	3.38	15.48	15.98	0.52	19.08	20.66
44100	9.96	4.3	13.65	14.4	4.65	20.18	21.67	0.72	22.34
48000	10.8	11.34	4.68	15.58	16.67	5.06	23.4	25.56	0.78

FsIN / FsOUT	8000	11025	12000	16000	22050	24000	32000	44100	48000
8000	0.07	7.71	8.24	2.28	10.5	11.28	4.41	13.44	14.48
11025	8.19	0.1	8.96	11.04	3.14	12	15.09	6.08	15.2
12000	8.76	9.52	0.1	11.3	14.48	3.41	15.36	20.07	6.61
16000	2.14	11.73	12.01	0.14	15.41	16.48	4.55	21	22.56
22050	10.78	2.94	15.39	16.38	0.19	17.92	22.08	6.27	24
24000	11.57	12.34	3.2	17.51	19.04	0.21	22.61	28.97	6.83
32000	4.19	15.48	15.77	4.27	23.46	24.01	0.28	30.83	32.96
44100	14.78	5.77	20.56	21.56	5.89	30.77	32.75	0.38	35.83
48000	15.92	16.7	6.28	23.15	24.69	6.41	35.02	38.08	0.42

FsIN / FsOUT	8000	11025	12000	16000	22050	24000	32000	44100	48000
8000	0.13	13.61	14.52	4.43	19.03	20.43	8.8	25.06	26.99
11025	14.85	0.18	15.91	19.47	6.1	21.82	27.35	12.13	28.38
12000	15.84	17.36	0.2	19.97	25.4	6.64	27.85	36.26	13.21
16000	4.25	21.24	21.79	0.26	27.22	29.03	8.86	38.07	40.85
22050	20.02	5.85	27.72	29.7	0.36	31.81	38.94	12.2	43.63
24000	21.45	22.98	6.37	31.68	34.71	0.39	39.94	50.8	13.28
32000	8.39	28.74	29.29	8.5	42.48	43.58	0.52	54.43	58.07
44100	28.11	11.57	38.05	40.03	11.71	55.43	59.4	0.72	63.62
48000	30.19	31.71	12.59	42.9	45.96	12.74	63.36	69.42	0.78

Parent topic: Processing requirements

On Arm9e and XScale

FsIN / FsOUT	8000	11025	12000	16000	22050	24000	32000	44100	48000
8000	0.03	1.14	1.25	0.54	1.95	2.14	1.04	3.85	4.23
11025	1.31	0.05	1.36	1.62	0.75	2.23	2.78	1.44	4.38
12000	1.43	1.57	0.05	1.68	2.13	0.82	2.84	3.72	1.57
16000	0.5	1.86	1.93	0.07	2.27	2.5	1.09	3.9	4.29
22050	2.19	0.69	2.42	2.61	0.1	2.72	3.24	1.5	4.46
24000	2.4	2.52	0.75	2.86	3.15	0.1	3.35	4.25	1.63
32000	0.92	3.12	3.18	1.01	3.72	3.86	0.14	4.55	4.99
44100	4.28	1.27	4.15	4.37	1.39	4.83	5.23	0.19	5.43
48000	4.7	4.9	1.39	4.8	5.03	1.51	5.72	6.3	0.21

FsIN / FsOUT	8000	11025	12000	16000	22050	24000	32000	44100	48000
8000	0.06	1.87	2.02	1.07	3.09	3.36	2.07	6.09	6.63
11025	2.27	0.09	2.25	2.66	1.47	3.56	4.4	2.85	7.01
12000	2.45	2.76	0.09	2.75	3.43	1.6	4.5	5.83	3.1
16000	0.99	3.23	3.36	0.13	3.73	4.05	2.14	6.17	6.72
22050	3.69	1.36	4.14	4.55	0.17	4.51	5.31	2.95	7.13
24000	4.01	4.28	1.48	4.9	5.51	0.19	5.51	6.85	3.21
32000	1.83	5.26	5.39	1.98	6.46	6.71	0.25	7.47	8.09
44100	7.22	2.52	6.94	7.38	2.72	8.27	9.1	0.35	9.02
48000	7.85	8.33	2.74	8.02	8.57	2.97	9.81	11.03	0.38

FsIN / FsOUT	8000	11025	12000	16000	22050	24000	32000	44100	48000
8000	0.03	1.21	1.33	0.61	2.08	2.29	1.17	4.1	4.51
11025	1.47	0.05	1.44	1.72	0.84	2.38	2.97	1.61	4.66
12000	1.62	1.76	0.05	1.78	2.26	0.91	3.03	3.98	1.75
16000	0.55	2.1	2.17	0.07	2.42	2.65	1.22	4.16	4.57
22050	2.49	0.76	2.73	2.95	0.1	2.88	3.45	1.68	4.75
24000	2.75	2.86	0.83	3.23	3.52	0.1	3.56	4.53	1.83
32000	1	3.56	3.63	1.11	4.2	4.34	0.14	4.84	5.3
44100	4.86	1.38	4.74	4.98	1.53	5.46	5.89	0.19	5.75
48000	5.38	5.55	1.5	5.5	5.71	1.66	6.47	7.05	0.21

FsIN / FsOUT	8000	11025	12000	16000	22050	24000	32000	44100	48000
8000	0.06	2.11	2.29	1.2	3.55	3.86	2.31	6.99	7.61
11025	2.62	0.09	2.52	3.01	1.66	4.07	5.07	3.19	8
12000	2.85	3.15	0.09	3.11	3.9	1.81	5.17	6.75	3.47
16000	1.09	3.73	3.85	0.13	4.22	4.57	2.41	7.1	7.72
22050	4.32	1.5	4.79	5.23	0.17	5.05	6.02	3.32	8.15
24000	4.74	4.99	1.64	5.69	6.3	0.19	6.22	7.8	3.61
32000	1.98	6.18	6.3	2.18	7.45	7.71	0.25	8.44	9.14
44100	8.43	2.72	8.18	8.64	3.01	9.59	10.47	0.35	10.1
48000	9.26	9.66	2.97	9.49	9.97	3.27	11.39	12.59	0.38

Parent topic: Processing requirements

On Cortex-A8 for worst case of 48000 Hz to 44100 Hz

Mode	MIPs
Mono at High Quality	3.13
Stereo at High Quality	3.61
Mono at Very High Quality	4.13
Stereo at Very High Quality	6.52

Parent topic: Processing requirements

Parent topic: Resource usage

Parent topic:*Introduction*

Application programmers interface (API) This section describes the application programming interface (API) libraries of the SSRC module.

Type definitions This section describes the type definitions of the SSRC module.

Types for allocation of instance and scratch memory The instance memory is the memory that contains the state of one instance of the SSRC module. Multiple instances of the SSRC module can exist, each with it is own instance memory. S memory is the memory that is only used temporarily by the process function of the SSRC module. This memory can be used as scratch memory by any other function running in the same thread as the SSRC module. Different threads cannot share the scratch memories.

The application must allocate both the instance and the scratch memory. The SSRC module does not allocate memory.

There is a data type available for both the instance and the scratch memory, namely SSRC_Instance_t and SSRC_Scratch_t. The instance type is defined as structures of the correct size in the SSRC header file. Both the instance and the scratch memory must be 4 bytes aligned.

Parent topic: Type definitions

LVM_Fs_en **Definition:**

```
typedef enum
{
    LVM_FS_8000    = 0,
    LVM_FS_11025   = 1,
    LVM_FS_12000   = 2,
    LVM_FS_16000   = 3,
    LVM_FS_22050   = 4,
    LVM_FS_24000   = 5,
    LVM_FS_32000   = 6,
    LVM_FS_44100   = 7,
    LVM_FS_48000   = 8
} LVM_Fs_en;
```

Description:

Used to pass the input and the output sample rate to the SSRC.

Parent topic: Type definitions

LVM_Format_en **Definition:**

```
typedef enum
{
    LVM_STEREO          = 0,
    LVM_MONOINSTEREO    = 1,
    LVM_MONO            = 2
} LVM_Format_en;
```

Description:

The LVM_Format_en enumerated type is used to set the value of the SSRC data format.

The SSRC supports input data in two formats Mono and Stereo. For an input buffer of NumSamples = N (meaning N sample pairs for Stereo and MonoInStereo or N samples for Mono), the format of data in the buffer is as listed in *Table 1*:

Sample Number	Stereo	MonoInStereo	Mono
0	Left(0)	Mono(0)	Mono(0)
1	Right(0)	Mono(0)	Mono(1)
2	Left(1)	Mono(1)	Mono(2)
3	Right(1)	Mono(1)	Mono(3)
4	Left(2)	Mono(2)	Mono(4)
“	“	“	“
“	“	“	“
N-2	Left(N/2-1)	Mono(N/2-1)	Mono(N-2)
N-1	Right(N/2-1)	Mono(N/2-1)	Mono(N-1)
N	Left(N/2)	Mono(N/2)	Not Used
N+1	Right(N/2)	Mono(N/2)	Not Used
N+2	Left(N/2+1)	Mono(N/2+1)	Not Used
N+3	Right(N/2+1)	Mono(N/2+1)	Not Used
“	“	“	Not Used
“	“	“	Not Used
2*N-2	Left(N-1)	Mono(N-1)	Not Used

Parent topic: Type definitions

SSRC_Quality_en **Definition:**

```
typedef enum
{
    SSRC_QUALITY_HIGH           = 0,
    SSRC_QUALITY_VERY_HIGH      = 1,
    SSRC_QUALITY_DUMMY          = LVM_MAXENUM
} SSRC_Quality_en;
```

Description:

Used to select the quality level of the SSRC. For details, see Performance figures. Selecting the highest-quality level, comes with a cost in the SSRC processing requirements. Therefore, it should only be done for critical applications.

Parent topic: Type definitions

Instance parameters **Definition:**

```
typedef struct
{
    SSRC_Quality_en     Quality;
    LVM_Fs_en           SSRC_Fs_In;
    LVM_Fs_en           SSRC_Fs_Out;
    LVM_Format_en       SSRC_NrOfChannels;
    short               NrSamplesIn;
    short               NrSamplesOut;
} SSRC_Params_t;
```

Description:

Used to pass the SSRC instance parameters to the SSRC module. It is a structure that contains the members for input sample rate, output sample rate, the number of channels, and the number of samples on the input and output audio stream.

Parent topic: Type definitions

Nr of samples mode **Definition:**

```
typedef enum
{
    SSRC_NR_SAMPLES_DEFAULT     = 0,
    SSRC_NR_SAMPLES_MIN         = 1,
    SSRC_NR_SAMPLES_DUMMY       = LVM_MAXENUM
} SSRC_NR_SAMPLES_MODE_en;
```

Description:

The SSRC_NR_SAMPLES_MODE_en enumerated type specifies the two different modes that can be used to retrieve the number of samples using the SSRC_GetNrSamples function.

Parent topic: Type definitions

Function return status **Definition:**

```
typedef enum
{
    SSRC_OK                     = 0,
    SSRC_INVALID_FS             = 1,
    SSRC_INVALID_NR_CHANNELS    = 2,
    SSRC_NULL_POINTER           = 3,
    SSRC_WRONG_NR_SAMPLES       = 4,
    SSRC_ALLINGMENT_ERROR       = 5,
    SSRC_INVALID_MODE           = 6,
    SSRC_INVALID_VALUE          = 7,
    SSRC_ALLINGMENT_ERROR       = 8,
    LVXXX_RETURNSTATUS_DUMMY = LVM_MAXENUM
} SSRC_ReturnStatus_en;
```

Description:

The SSRC_ReturnStatus_en enumerated type specifies the different error codes returned by the API functions. For the exact meaning, see the individual function descriptions.

Parent topic: Type definitions

Parent topic:*Application programmers interface (API)*

Functions This section lists all the API functions of the SSRC module and explains their parameters.

SSRC_GetNrSamples **Prototype:**

```
SSRC_ReturnStatus_en SSRC_GetNrSamples
  (SSRC_NR_SAMPLES_MODE_en  Mode,
   SSRC_Params_t*           pSSRC_Params );
```

Description:

This function retrieves the number of samples or sample pairs for stereo used as an input and as an output of the SSRC module.

Nam	Type	Description
Mod	SSRC_N	There are two modes: - SSRC_NR_SAMPLES_DEFAULT: In this mode, the function returns the number of samples for 40 ms blocks - SSRC_NR_SAMPLES_MIN: the function returns the minimal number of samples supported for this conversion ratio. The SSRC_Init function accepts each integer multiple of this ratio. Formula: $\text{blocksize (ms)} = 1/\text{gcd(Fs_In,Fs_Out)}$
pSSF	SSRC_Pa	Pointer to the instance parameters. The application fills in the values of the input sample rate, the output sample rate, and the number of channels. Based on this input, the SSRC_GetNrSamples fills in the values for the number of samples for the input and the output audio stream.

Returns:

SSRC_OK	When the function call succeeds.
SSRC_INVALID_FS	When the requested input or output sampling rates are invalid.
SSRC_INVALID_NR_CHANN	When the channel format is not equal to LVM_MONO or LVM_STEREO.
SSRC_NULL_POINTER	When pSSRC_Params is a NULL pointer.
SSRC_INVALID_MODE	When mode is not a valid setting.

Note: The SSRC_GetNrSamples function returns the values from the following tables. Instead of calling the SSRC_GetNrSamples function, use the values from these tables directly.

Sample rate	Nr of samples
8000	320
11025	441
12000	480
16000	640
22050	882
24000	960
32000	1280
44100	1764
48000	1920

In/Out	8000	11025	12000	16000	22050	24000	32000	44100	48000
8000	11	320441	23	12	160441	13	14	80441	16
11025	441320	11	147160	441640	12	147320	4411280	14	147640
12000	32	160147	11	34	80147	12	38	40147	14
16000	21	640441	43	11	320441	23	12	160441	13
22050	441160	21	14780	441320	11	147160	441640	12	147320
24000	31	320147	21	32	160147	11	34	80147	12
32000	41	1280441	83	21	640441	43	11	320441	23
44100	44180	41	14740	441160	21	14780	441320	11	147160
48000	61	640147	41	31	320147	21	32	160147	11

Parent topic: Functions

SSRC_GetScratchSize **Prototype:**

```
SSRC_ReturnStatus_en SSRC_GetScratchSize
  (SSRC_Params_t*     pSSRC_Params,
   LVM_INT32*         pScratchSize );
```

Description:

This function retrieves the scratch size for a given conversion ratio and for given buffer sizes at the input and at the output.

Name	Type	Description
pSSRC_Par	SSRC_Param	Pointer to the instance parameters. All members should have a valid value.
pScratch-Size	LVM_INT32*	Pointer to the scratch size. The SSRC_GetScratchSize function fills in the correct value (in bytes).

|

Returns:

SSRC_OK	When the function call succeeds.
SSRC_INVALID_FS	When the requested input or output sampling rates are invalid.
SSRC_INVALID_NR_CHANN	When the channel format is not equal to LVM_MONO or LVM_STEREO.
SSRC_NULL_POINTER	When pSSRC_Params or pScratchSize is a NULL pointer.
SSRC_WRONG_NR_SAMPLI	When the number of samples on the input or on the output are incorrect.

Parent topic: Functions

SSRC_Init **Prototype:**

```
SSRC_ReturnStatus_en SSRC_Init
  (SSRC_Instance_t* pSSRC_Instance,
   SSRC_Scratch_t*  pSSRC_Scratch,
   SSRC_Params_t*   pSSRC_Params,
   LVM_INT16**      ppInputInScratch,
   LVM_INT16**      ppOutputInScratch);
```

Description:

The SSRC_Init function initializes an instance of the SSRC module.

Name	Type	Description
pSSRC_	SSRC_	Pointer to the instance of the SSRC. This application must allocate the memory before calling the SSRC_Init function.
pSSRC_	SSRC_	Pointer to the scratch memory. The pointer is saved inside the instance and is used by the SSRC_Process function. The application must allocate the scratch memory before calling the SSRC_Init function.
pSSRC_	SSRC_	Pointer to the instance parameters.
ppInputInScratch	LVM_I	The SSRC module can be called with the input samples located in scratch. This pointer points to a location that holds the pointer to the location in the scratch memory that can be used to store the input samples. For example, to save memory.
ppOutputInScratch	LVM_I	The SSRC module can store the output samples in the scratch memory. This pointer points to a location that holds the pointer to the location in the scratch memory that can be used to store the output samples. For example, to save memory.

Returns:

SSRC_OK	When the function call succeeds.
SSRC_INVALID_FS	When the requested input or output sampling rates are invalid.
SSRC_INVALID_NR_CHANN	When the channel format is not equal to LVM_MONO or LVM_STEREO.
SSRC_NULL_POINTER	When pSSRC_Params or pScratchSize is a NULL pointer.
SSRC_WRONG_NR_SAMPLI	When the number of samples on the input or on the output are incorrect.
SSRC_ALIGNMENT_ERROR	When the instance memory or the scratch memory is not 4 bytes aligned.

Parent topic: Functions

SSRC_SetGains **Prototype:**

```
SSRC_ReturnStatus_en SSRC_SetGains
  (SSRC_Instance_t* pSSRC_Instance,
   LVM_Mode_en      bHeadroomGainEnabled,
   LVM_Mode_en      bOutputGainEnabled,
   LVM_INT16        OutputGain);
```

Description:

This function sets headroom gain and the post gain of the SSRC. The SSRC_SetGains function is an optional function that should be used only in rare cases. Preferably, use the default settings.

Name	Type	Description
pSSRC	SSRC_	Pointer to the instance of the SSRC.
bHeadroomGainEnabled	LVM_	Parameter to enable or disable the headroom gain of the SSRC. The default value is LVM_MODE_ON. LVM_MODE_OFF can be used if it can be guaranteed that the input level is below - 6 in all cases (the default headroom is -6 dB).
bOutputGainEnabled	LVM_	Parameter to enable or disable the output gain. The default value is LVM_MODE_ON.
OutputGain	LVM_	The value of the output gain. The output gain is a linear gain value. 0x7FFF is equal to +6 dB and 0x0000 corresponds to -inf dB. By default, a 3 dB gain is applied (OutputGain = 23197), resulting in an overall gain of -3 dB (-6 dB headroom +3 dB output gain). Unit Q format Data Range Default value Linear gain Q1.14 [0;32767] 23197

Returns:

SSRC_OK	When the function call succeeds
SSRC_NULL_POINT	When pSSRC_Instance is a NULL pointer
SSRC_INVALID_MO	Wrong value used for the bHeadroomGainEnabled or the OutputGainEnabled parameters.
SSRC_INVALID_VAI	When OutputGain is out of the range [0;32767].

Parent topic: Functions

SSRC_Process **Prototype:**

```
SSRC_ReturnStatus_en SSRC_Process
  (SSRC_Instance_t* pSSRC_Instance,
   LVM_INT16*       pSSRC_AudioIn,
   LVM_INT16*       pSSRC_AudioOut);
```

Description:

Process function for the SSRC module. The function takes pointers as input and output audio buffers.

The sample format used for the input and output buffers is 16-bit little-endian. Stereo buffers are interleaved (L1, R1, L2, R2, and so on), mono buffers are deinterleaved (L1, L2, and so on).

Name	Type	Description
pSSRC_Instance	SSRC_Instance_t*	Pointer to the instance of the SSRC.
pSSRC_AudioIn	LVM_INT16*	Pointer to the input samples.
pSSRC_AudioOut	LVM_INT16*	Pointer to the output samples.

Returns:

SSRC_OK	When the function call succeeds.
SSRC_NULL_POINTEI	When one of pSSRC_Instance, pSSRC_AudioIn, or pSSRC_AudioOut is NULL.

Parent topic: Functions

SSRC_Process_D32 **Prototype:**

```
SSRC_ReturnStatus_en SSRC_Process_D32
  (SSRC_Instance_t* pSSRC_Instance,
   LVM_INT32*       pSSRC_AudioIn,
   LVM_INT32*       pSSRC_AudioOut);
```

Description:

Process function for the SSRC module. The function takes pointers as input and output audio buffers.

The sample format used for the input and output buffers is 32-bit little-endian. Stereo buffers are interleaved (L1, R1, L2, R2, and so on), mono buffers are deinterleaved (L1, L2, and so on).

Name	Type	Description
pSSRC_Instance	SSRC_Instance_t*	Pointer to the instance of the SSRC.
pSSRC_AudioIn	LVM_INT32*	Pointer to the input samples.
pSSRC_AudioOut	LVM_INT32*	Pointer to the output samples.

Returns:

|SSRC_OK|When the function call succeeds.| |SSRC_NULL_POINTER|When one of pSSRC_Instance, pSSRC_AudioIn, or pSSRC_AudioOut is NULL.|

Parent topic: Functions

Parent topic:*Application programmers interface (API)*

Dynamic function usage

This chapter explains how and when the SSRC functions are or can be used.

Define the number of samples to be used on input and output

Call the function SSRC_GetNrSamples. Each integer multiple of the returned number of samples can be used.

Parent topic: Dynamic function usage

Allocate scratch memory

To calculate the required size of the scratch memory, call the SSRC_GetScratchSize function. Allocate memory for the returned size.

Parent topic: Dynamic function usage

Initialize the SSRC instance

Call the SSRC_Init function.

Parent topic: Dynamic function usage

Process samples The SSRC_Process function can now be called any number of times.

Parent topic: Dynamic function usage

Destroy the SSRC instance When the processing is completed, the allocated memory for the instance and the scratch can be freed.

Parent topic: Dynamic function usage

Parent topic:*Application programmers interface (API)*

Reentrancy None of the SSRC functions are re-entrant.

Parent topic:*Application programmers interface (API)*

Additional user information This section provides information on the Attenuation of the signal and Notes on integration.

Attenuation of the signal When a fully saturated or clipped input is applied to an SRC module, the aliases after the sample rate conversion, although sufficiently suppressed, can still result in a clipped output. To prevent clipped output, the output of the SSRC module is by default attenuated with 3 dB. Although not advised, this gain value can be changed using the SSRC_SetGains function.

Parent topic:*Additional user information*

Notes on integration Although the sample rate converter module works with audio signals on different sampling rates, it is a synchronous module. The module takes a block of input samples, consumes the input completely, and produces a full buffer with output samples. As a result, the SSRC only accepts a limited number of input and output block sizes. To flush last, incomplete, block of an audio stream, the block is padded with zeros until it is full before the SSRC processes it.

Parent topic:*Additional user information*

Example application The source code of the example application can be found in the .\EX_APP\APP_FileIO\SRC directory of the release package. The .\EX_APP\APP_FileIO\MAKE directory contains a make file that can be used to build the example application. When building the application, an executable is generated in the .\EX_APP\APP_FileIO\EXE directory.

The example application takes as command-line input parameters:

1. The path toward the input PCM file. It assumes raw 16 bit signed little-endian put. Stereo input samples should be interleaved (L1, L2 R1, R2,...), mono samples should be deinterleaved (L1, L2, and so on).
2. The path toward the output PCM file.
3. The input sample rate.
4. The output sample rate.
5. The channel format (mono or stereo).

Integration test A correct integration of the SSRC module can be verified in two ways.

- Bit accurate test
- THD+N measurement

Bit accurate test The TestFiles directory of the release package contains a test input (sampled at 44,100 Hz) and several expected output files (sample rates from 8000 Hz to 48,000 Hz). If the same test input file is applied to the SRC after integration in the target platform, the output is bit accurate with the expected output file that matches the output-sample rate

Parent topic:*Integration test*

THD+N measurement Produce a swept sine and feed it through the SSRC module. Do a THD+N measurement on the obtained output signal. The THD+N of the converted signals should be below - 77 in the interval [0 - 0.45] FsLOW.

Parent topic:*Integration test*

Maestro Audio Framework

MCUXpresso SDK : Maestro

Overview This repository is for MCUXpresso SDK maestro 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

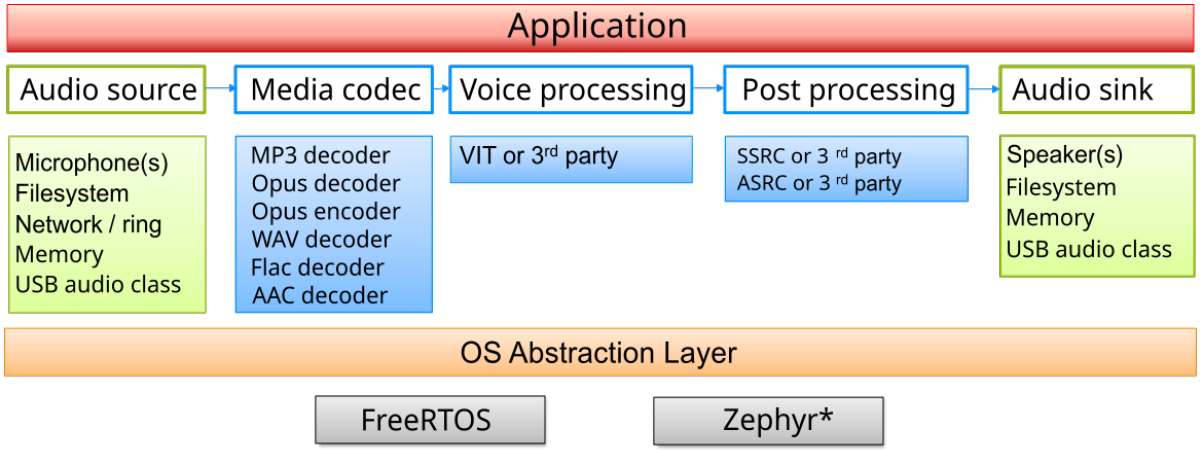
Visit Maestro - Documentation to review details on the contents in this sub-repo.

Setup Instructions on how to install the MCUXpresso SDK provided from GitHub via west manifest Getting Started with SDK - Detailed Installation Instructions

Contribution We welcome and encourage the community to submit patches directly to the Maestro project placed on github. Contributing can be managed via pull-requests.

Introduction Maestro audio framework intends to enable chaining of basic audio processing blocks, called *elements*. These blocks then form stream processing objects, called *pipeline*. This pipeline can be used for multiple audio processing use cases.

The processing blocks can include (but are not limited to) different audio sources (for example file or microphone), decoders or encoders, filters or effects, and audio sinks. Framework overview is depicted in the following picture:



*not all elements and libraries are supported in Zephyr port. For more information, see *Maestro on Zephyr*

The Maestro audio framework is an open-source component developed by NXP Semiconductors and released under the BSD-compatible license. It is running on RTOS (Zephyr or FreeRTOS), abstracted by OSA layer.

For detailed description of the audio Maestro framework, please refer to the *programmer's guide*.

To see what is new, see *changelog*.

Maestro on Zephyr Getting started guide and further information for Maestro on Zephyr may be found *here*.

Maestro on FreeRTOS Maestro on FreeRTOS is supported in NXP's SDK. To get started, see mcuxsdk doc.

Supported examples The current version of the Maestro audio framework supports several optional *features*, some of which are used in these examples:

- *maestro_playback*
- *maestro_record*
- *maestro_usb_mic*
- *maestro_usb_speaker*
- *maestro_sync*

The examples can be found in the **audio_examples** folder of the desired board. The demo applications are based on FreeRTOS and use multiple tasks to form the application functionality.

Example applications overview To set up the audio framework properly, it is necessary to create a streamer with streamer_create API. It is also essential to set up the desired hardware peripherals using the functions described in streamer_pcm.h. The Maestro example projects consist of several files regarding the audio framework. The initial file is main.c with code to create multiple tasks. For features including SD card (in the maestro_playback examples, reading a file from SD card is supported and in maestro_record writing to SD card is currently supported) the APP_SDCARD_Task is created. The command prompt and connected functionalities are handled by APP_Shell_Task.

One of the most important parts of the configuration is the streamer_pcm.c where the initialization of the hardware peripherals, input and output buffer management can be found. For further information please see also streamer_pcm.h

In the Maestro USB examples (maestro_usb_mic and maestro_usb_speaker), the USB configuration is located in the usb_device_descriptor.c, audio_microphone.c and audio_speaker.c files. For further information please see also usb_device_descriptor.h, audio_microphone.h and audio_speaker.h.

In order to be able to get the messages from the audio framework, it is necessary to create a thread for receiving the messages from the streamer, which is usually called a Message Task. The message thread is placed in the app_streamer.c file, reads the streamer message queue, and reacts to the following messages:

- STREAM_MSG_ERROR - stops the streamer and exits the message thread
- STREAM_MSG_EOS - stops the streamer and exits the message thread
- STREAM_MSG_UPDATE_DURATION - prints info about the stream duration
- STREAM_MSG_UPDATE_POSITION - prints info about current stream position
- STREAM_MSG_CLOSE_TASK - exits the message thread

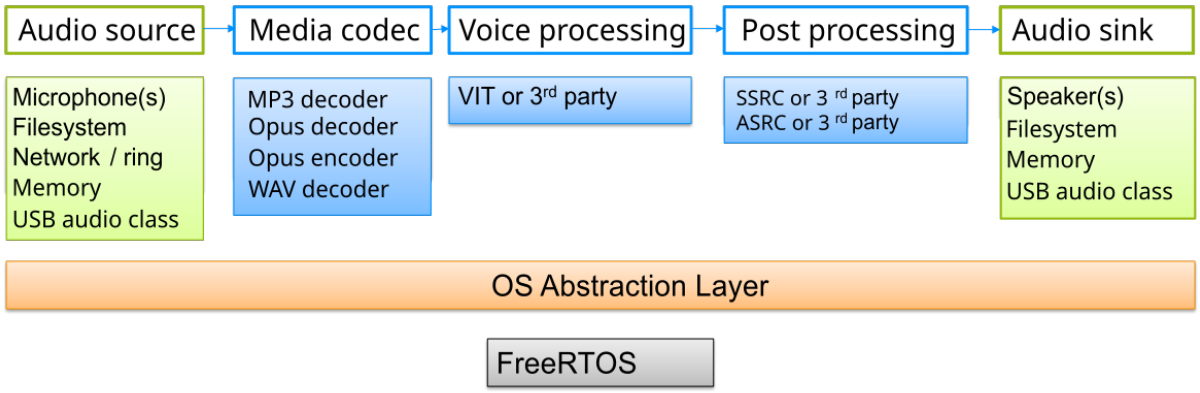
File structure

Folder	Description
src	Maestro audio framework sources
src/inc	Maestro include files
src/core	Maestro core sources
src/cci	Common decoder interface sources
src/cei	Common encoder interface sources
src/elements	Maestro elements sources
src/devices	External audio devices implementation (audio source & audio sink elements)
src/utils	Helper utilities utilized by Maestro
docs	Generated documentation
doxygen	Documentation sources
components	Glue for audio libraries, so they can be used in elements
tests	Maestro tests
zephyr/	Zephyr related files
zephyr/samples/	Zephyr samples
zephyr/tests/	Zephyr tests
zephyr/audioTracks/	Audio tracks for testing
zephyr/wrappers/	Zephyr NXP SDK Wrappers
zephyr/doc/	Zephyr documentation configuration for Sphinx
zephyr/scripts/	Zephyr helper scripts, mostly for testing

Maestro Audio Framework Programmer's Guide

Introduction Maestro audio framework provides instruments for playback and capture of different audio streams. In order to do that the framework uses API for creating various audio and voice pipelines with the support of media and track information. This document describes the framework in its detail, and the usage of API for pipeline creation using different elements. The framework needs an operating system in order to create different tasks for audio processing and communication with the application.

Architecture overview A high-level block diagram of the streamer used in Maestro is shown below. An element is the most important class of objects in the streamer (see streamer_element.c). A chain of elements will be created and linked together when a *pipeline* is created. Data flows through this chain of elements in form of data buffers. An element has one specific function, which can be the reading of data from a file, decoding of this data, or outputting this data to a sink device. By chaining together several such elements, a pipeline is created that can do a specific task, for example, the playback.



Pipeline

The pipeline is created within the streamer_create API using the streamer_create_pipeline call. In the example applications provided in the MCUXpresso SDK the pipeline is created in the app_streamer.c file. In order to create a pipeline user needs to provide a PipelineElements structure consisting of array of element indexes ElementIndex and the number of elements in the pipeline. Then the pipeline is built automatically and user can specify the properties of the elements using the streamer_set_property API. All the element properties can be found in the streamer_element_properties.h file.

The streamer can handle up to two pipelines within a single task. The first pipeline with index 0 can be created using the streamer_create function as described above. Then the streamer_create_pipeline function should be used to create the second pipeline (pipeline with index 1). An example creation can be found in the app_streamer.c file in the *maestro_sync_example*. Both pipelines are processed sequentially, so after the first pipeline is processed, the second pipeline is processed.

After the pipeline is sucessfully created, all elements and entire pipeline are in STATE_NULL state. A user can start the streamer by setting the pipeline state to STATE_PLAYING using the streamer_set_state function. The pipeline can also be paused or stopped using the same function. Use the STATE_PAUSED to pause and use STATE_NULL to stop. The function changes the state of each element that is in the pipeline in turn, and after all the elements have obtained the desired state, the state of entire pipeline is changed.

Elements The current version of the Maestro framework supports several types of elements (StreamElementType). In each pipeline should be used one source element (elements with the _SRC suffix) and one sink element (elements with the _SINK suffix). A decoder, encoder or audio_proc element can be connected between these two elements. The audio_proc element can be used more than once within the same pipeline.

Each element type (StreamElementType) has several functions that are determined by a unique element index (ElementIndex). These indexes are used to create a pipeline, and each element index can only be used once in the same pipeline. The type_lookup_table shows which StreamElementType supports which ElementIndex.

Each element index (ElementIndex) has its own properties and a list of these properties can be found in the streamer_element_properties.h file. These properties are divided into groups and each group is identified by a property mask (e.g. for speaker it is PROP_SPEAKER_MASK). Then the property_lookup_table in the streamer_msg.c file determines which property group relates to

which element index (ElementIndex). When an element is created and added to the pipeline, its properties are set to their defalut values. Default values can be seen in the initialization function of a particular element. The initialization functions are specified in the element_list array in the streamer_element.c file (e.g. for the audio_proc element it is the audio_proc_init_element function). The user can get the value of the property using the streamer_get_property function or change its value using the streamer_set_property function.

The source code of the elements can be found in the middleware\audio_voice\maestro\src\elements\ folder.

Add a new element type The user can add a new element type (StreamElementType) to the Maestro audio framework. For this, the following steps need to be done.

- Add a new element type to the StreamElementType enum type in the streamer_api.h.
- Create a new *.c and *.h files for the new element type in the middleware\audio_voice\maestro\src\elements\ folder. All necessary structures and functions (functions for src pads, sink pads and element itself) needs to be defined in these files. Inspiration can be found in other elements.
- Link the initialization function to the element type in the element_list array in the streamer_element.c file. To do this, a new definition that enables the element needs to be created (e.g. there is a STREAMER_ENABLE_AUDIO_PROC definition for the audio_proc element).
- Associate the newly created element type with an element index (ElementIndex) by adding a new pair to the type_lookup_table in the streamer.c file.
- If the user wants to use the newly created element in an application, the definiton that enables the element must be defined at the project level.

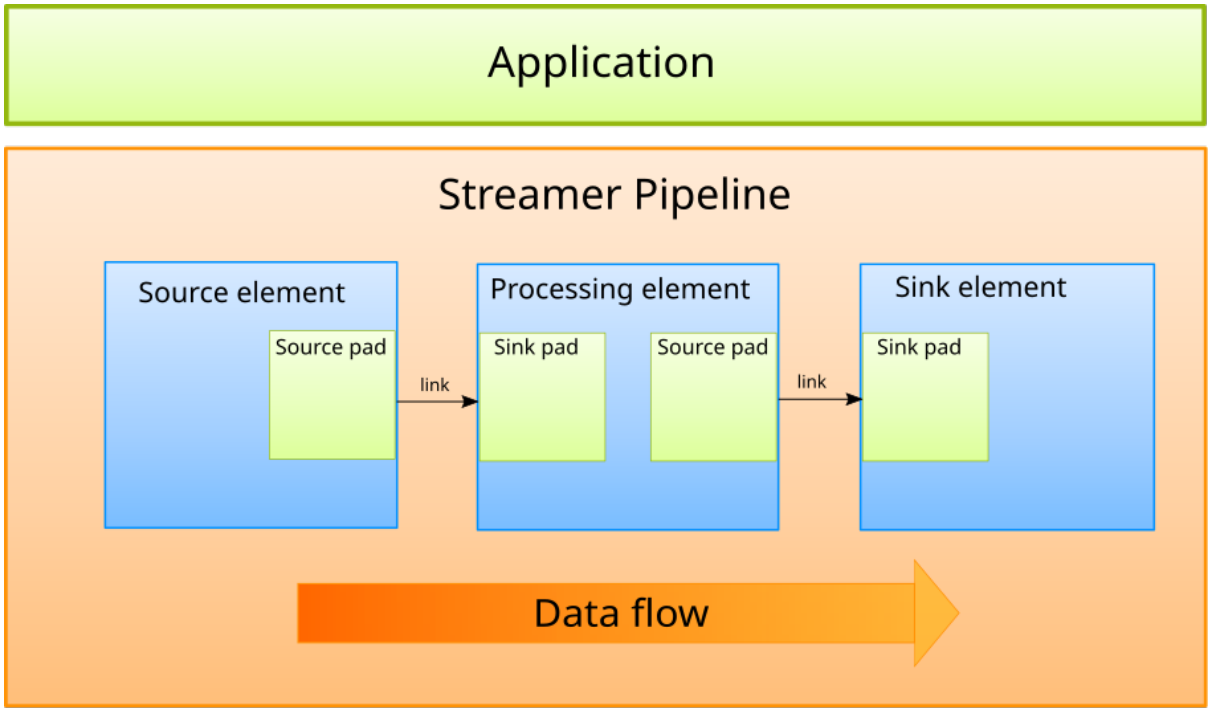
Mostly the user doesn't need to create a new element type, but just create an element index.

Add a new element index To create a new element index in the Maestro audio framework, follow these steps:

- Add a new element index to the ElementIndex enum type in the streamer_api.h.
- Create the required properties for the newly created element index in the streamer_element_properties.h file.
- Associate the newly created property group with newly created element index by adding a new pair to the property_lookup_table in the streamer_msg.c file.
- Associate the newly created element index with an element type (StreamElementType) by adding a new pair to the type_lookup_table in the streamer.c file.
- Add support for the created properties to functions of the associated element type. These functions are defined in files that correspond to a particular element type. The files are located in the middleware\audio_voice\maestro\src\elements\ folder.

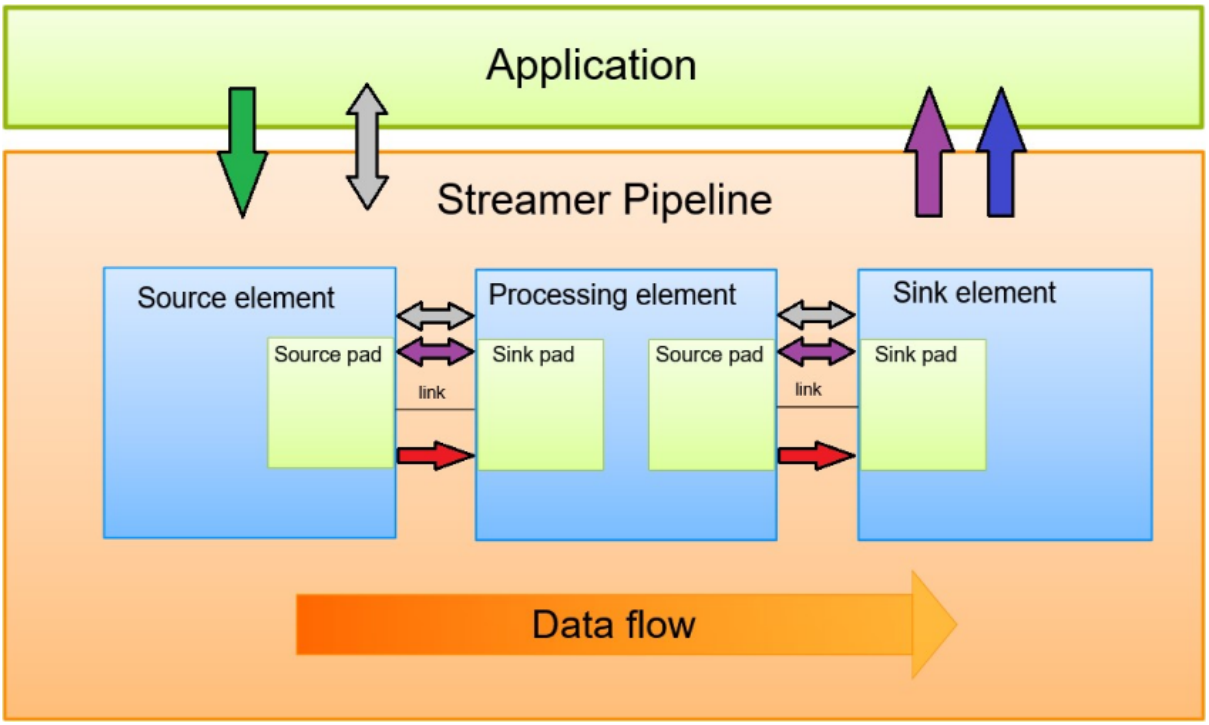
It is important to know that each element type (StreamElementType) can be associated with more than one element index (ElementIndex), but each element index (ElementIndex) can be associated with only one element type (StreamElementType).

Pads Pads are elements' inputs and outputs. A pad can be viewed as a "plug" or "port" on an element where links may be made with other elements, and through which data can flow to or from those elements. Data flows out of an element through a source pad, and elements accept incoming data through a sink pad. Source and sink elements have only source and sink pads, respectively. For detailed information about pads, please see the API reference from pad.c.



Internal communication The streamer (the core of the framework) provides several mechanisms for communication and data exchange between the application, a pipeline, and pipeline elements:

- Buffers are objects for passing streaming data between elements in the pipeline. Buffers always travel from sources to sinks (downstream).
- Messages are objects sent from the application to the streamer task to construct, configure, and control a streamer pipeline.
- Callbacks are used to transmit information such as errors, tags, state changes, etc. from the pipeline and elements to the application.
- Events are objects sent between elements. Events can travel upstream and downstream. Events may also be sent to the application
- Queries allow applications to request information such as duration or current playback position from the pipeline. Elements can also use queries to request information from their peer elements (such as the file size or duration). They can be used both ways within a pipeline, but upstream queries are more common



Decoders and encoders Maestro framework uses a common codec interface for decoding purposes and a common encoder interface for encoding. Those interfaces encapsulate the usage of specific codecs. Reference codecs are available in audio-voice-components repository which should be in \middleware\audio_voice\components\ folder.

Common codec interface The Common Codec Interface is the intended interface for all used **decoders**. The framework will integrate a CCI decoder element into the streamer to interface with all decoders.

Using the CCI to interface with Metadata

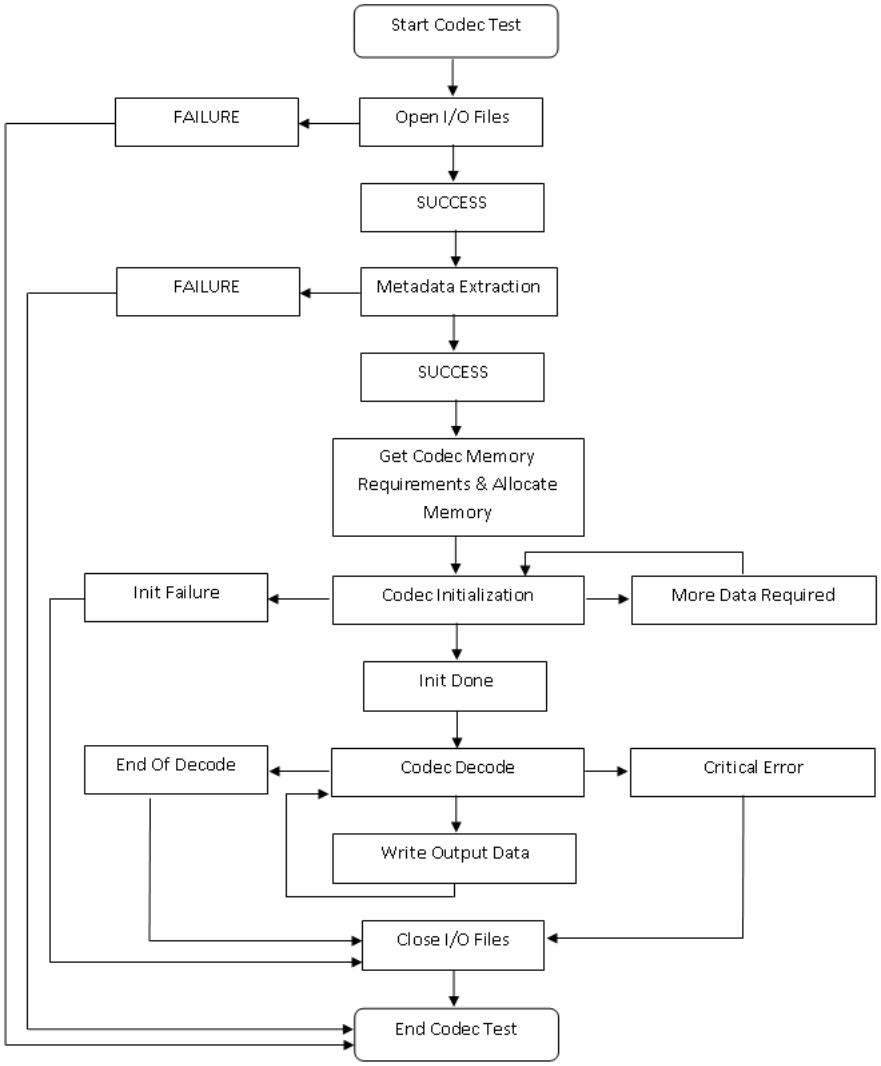
- cci_extract_meta_data must be called before any other Codec Interface APIs. This API extracts the metadata information of the codec and fills this information in the file_meta_data_t structure. The file_meta_data_t structure must be allocated by the application.
- This function first extracts the input file extension and based on that it calls the specific codec's metadata extraction function. If it finds an invalid extension or unsupported extension then it returns with META_DATA_FILE_NOT_SUPPORTED code for any unsupported file format.
- If this API finds the valid metadata then it returns with META_DATA_FOUND code. If this API does not find any metadata information then it returns with META_DATA_NOT_FOUND code. It also returns with META_DATA_FILE_NOT_SUPPORTED code for any unsupported file format.

Using the CCI to interface with Decoders

- codec_get_mem_info gets the memory requirement based on the specific decoder stream type. It returns the size in bytes of the specific codec. The user of the decoders must allocate memory of this size and this memory is used by the initialization API. The user or application must pass this allocated memory pointer to the init API.

- codec_init must be called before the codec's decode API. This API calls the codec-specific initialization function based on the codec stream type. This API allocates the memory to the codec main structure and also initializes the codec main structure parameters. It also registers the call back functions to the codec which will be used by the codec to read or seek the input stream.
- codec_decode is the main decoding API of the codec. This API calls the codec-specific decoding function based on the codec stream type. This API decodes the input raw stream and fills the PCM output samples into codec output PCM buffer. This API gives the information about the number of samples produced by the codec and also gives the pointer of the codec output PCM samples buffer.
- codec_get_pcm_samples must be called after the codec's decode API. This API calls the codec specific Get PCM Sample API based on the codec stream type. This API gets the PCM samples from the codec in constant block size and fills them into the output PCM buffer. It returns the number of samples get from the codec and also gives the pointer of the output PCM buffer.
- codec_reset calls the codec specific reset API base on stream type and resets the codec.
- codec_seek accepts the seek bytes offset converted from the time by application. This API calls the decoder's internal seek API to calculate the actual seek offset which frame boundary aligns. This API returns the actual seek offset.

The basic sequence to use a decoder with the CCI is shown below:



Adding new decoders to the CCI This section explains how to integrate a new decoder in the Common Codec Interface. The CCI assumes the decoder library to be used is in the \middleware\audio_voice\audiocomponents\decoders*decoder*\libs\ folder of the maestro framework. The CCI is just a wrapper around a specific implementation. The decoder is expected to be extended as needed to meet the APIs described above.

- Register Decoder Top level APIs in Common Codec Interface
  - Place the decoder lib in libs folder.
  - Add prototypes of the decoder top level APIs in codec_interface.h file (located at maestro\src\cci\inc\ folder).
  - In codec_interface.c file (located at maestro\src\cci\src\), add top level Decoder APIs in decoder function table.
  - Pseudo code for this is as described below.

```
const codec_interface_function_table_t g_codec_function_table[STREAM_TYPE_COUNT] = {
#ifdef VORBIS_CODEC
    {
        &VORBISDecoderGetMemorySize,
        &VORBISDecoderInit,
        &VORBISDecoderDecode,
        NULL,
        NULL,
        &VORBISDecoderSeek,
        &VORBISDecoderGetIOFrameSize,
    },
#else
    {
        NULL,
        NULL,
        NULL,
        NULL,
        NULL,
        NULL,
        NULL,
    }
#endif
};
```

- Enable or Disable Decoder
  - Define VORBIS_CODEC macro in audio_cfg.h file.
  - Comment this macro if you want to disable VORBIS Decoder otherwise keep it defined in order to enable the decoder.
- Add Extract Metadata API for the decoder
  - Add extract metadata API source file for the decoder at streamer/cci/metadata/src/vorbis folder.
  - Add this code in extract metadata lib project space.
  - Build the extract metadata lib and copy that lib to libs folder.
  - Add the desired stream type into ccidec_extract_meta_data API (in codecextractmetadata.c file) to call VORBIS Decoder extract metadata API.
- Add stream type of the new decoder in the stream type enum audio_stream_type_t in codec_interface_public_api.h
  - Stream type of the decoder in stream type enum and decoder APIs in decoder function table must be in the same sequence.

Common encoder interface Please see the following section about the *cei*.

Maestro performance

Memory information The memory usage of the framework components using reference codecs (data obtained from GNU ARM compiler) in bytes is:

text	data	bss	component
48790	2752	4	aac decoder
4348	16400	212	asrc
15512	0	4	flac decoder
76462	16	5013	maestro
34211	0	4	mp3 decoder
211974	0	0	opus
65446	0	4	ssrc
5850	16	12	wav decoder

Maestro framework uses dynamic allocation of audio buffers. The total amount of memory allocated for the pipeline depends on the following parameters:

- Number of elements in the pipeline
- Element types
- Audio stream properties
  - Sampling rate
  - Bit width
  - Channel number
  - Frame size

CPU usage The performance of the pipeline was measured using the real hardware platform (RT1060).

- CPU core clock in MHz: 600.

Pipeline type	Performance MIPS of pipeline (in MHz)
audio source -> audio sink	~10.26 MHz
audio source -> file sink	~9.84 MHz
file source (8-channel PCM) -> audio sink	~16.5 MHz

For performance details about the supported codecs please see audio-voice-components repository documentation.

CEI encoder The Maestro streamer contains an element adapting an extensible set of audio encoders in the form of functions conforming to the CEI (Common Encoder Interface). This element enables the user to choose and configure a suitable encoder at runtime.

Header files CEI itself and the CEI encoders are using following header files, in which you may be interested:

- cei.h - contains types used by the element itself and an encoder implementing the CEI
- cei_enctypes.h - contains a list of possible encoders and types used for interfacing with a CEI encoder
- cei_table.h - contains a table of functions implementing integrated CEI encoders

Instantiating the element This element's index is ELEMENT_ENCODER_INDEX and its type is TYPE_ELEMENT_ENCODER, as defined in streamer_api.h. It has one source pad (data input) and one sink pad (data output). It is initialized like any other element, meaning that it is instantiated and inserted into the pipeline using the create_element, add_element_pipeline and link_elements functions. Inversely, for destroying the element, the unlink_elements, remove_element_pipeline and destroy_element are used. This element alone does not depend on any additional software layers other than these required by the Maestro streamer itself, so no pre-initialization before this element instantiation is necessary.

Element properties Use Maestro streamer property API (streamer_set_property and streamer_get_property) for setting or getting these. The constants are defined in streamer_element_properties.h.

- PROP_ENCODER_CHUNK_SIZE
  - **Synopsis:** Determines the length of a chunk pulled from the sibling of the source pad and essentially influences the size of allocated buffers. If the actual amount of data pulled is smaller, the rest is zero-filled.
  - **Type:** unsigned 32-bit integer
  - **Default value:** 1920
  - **Constraints:**
    - Must be bigger than zero, otherwise STREAM_ERR_INVALID_ARGS is returned.
    - Cannot be changed if the actual encoder has been created. If done so, STREAM_ERR_ELEMENT_BAD_STATUS is returned.
- PROP_ENCODER_TYPE
  - **Synopsis:** Determines the exact encoder (CEI implementation) to be used.
  - **Type:** CeiEncoderType (cei_enctypes.h)
  - **Default value:** CEIENC_LAST
  - **Constraints:**
    - Must not be equal to CEIENC_LAST, otherwise STREAM_ERR_INVALID_ARGS will be returned.
    - Selected encoder must be implemented, otherwise STREAM_ERR_INVALID_ARGS will be returned.
    - Cannot be changed if the actual encoder has been created. If done so, STREAM_ERR_ELEMENT_BAD_STATUS will be returned.
  - **Behaviour influenced:** The encoder element process function will return FLOW_ERROR if this property isn't set.
- PROP_ENCODER_CONFIG
  - **Synopsis:** Determines encoder-specific configuration (application, bitrate, ...).
  - **Type:** Pointer to the encoder-specific configuration structure.

  - **Default value:** Determined by the encoder.
  - **Constraints**:
    * The encoder has to be configurable. If it is not, STREAM_ERR_ERR_GENERAL will be returned on any access.
    * The structure has to conform to the encoder requirements. If the encoder returns an error code, STREAM_ERR_GENERAL will be returned.

- PROP_ENCODER_BITSTREAMINFO
  - **Synopsis:** Specifies information about the incoming bitstream (sample rate, sample depth, ...).
  - **Type:** Pointer to CeiBitstreamInfo (cei_enctypes.h).
  - **Default value:**

```
(CeiBitstreamInfo) {
    .sample_rate = 0,
    .num_channels = 0,
    .endian = AF_LITTLE_ENDIAN,
    .sign = TRUE,
    .sample_size = 0,
    .interleaved = TRUE
}
```

  - **Constraints:**
    * Cannot be changed if the actual encoder has been created. If done so, STREAM_ERR_ELEMENT_BAD_STATUS will be returned.
    * As of now, only bitstreams containing 16-bit interleaved (if 2 or more channels will be encoded) samples are supported. If anything else was set to the sample_size and interleaved members, STREAM_ERR_INVALID_ARGS will be returned.
  - **Behaviour influenced:**
    * Given the characteristics of some elements available, different packets of data (header and payload, referred to as "chunk" above) may be pulled by this element. Each packet can contain a different header, which may or may not contain useful information about the bitstream. If a packet with the AudioPacketHeader (todofile.h) is pulled at first and any other iteration of the streamer pipeline, the bitstream parameters configured by this property are implicitly available and are not expected to be specified by the user. Other packet header types (such as RawPacketHeader) don't contain any bitstream parameters and require the user to specify the parameters manually using this property. Failure to do so will result in the element's process function returning FLOW_ERROR. Same situation will occur if a packet with the AudioPacketHeader is received and its contents differ from the already acquired bitstream parameters.
    * As of now, CEI is defined to work with 16-bit signed little-endian (s16le) samples, which are interleaved if the bitstream contains more than one channels. This element handles endianness and unsigned to signed conversion.

CEI definition - implementing your own encoder The CEI defines following function pointer types:

- CeiFnGetMemorySize: Returns number of bytes required for encoder state for a given number of channels.
- CeiFnEncoderInit: Initialize an encoder for a given sample rate and channel count.
- CeiFnEncoderGetConfig: Copy current or default configuration to a given structure pointer.

- CeiFnEncoderSetConfig: Configure the encoder from a given structure pointer.
- CeiFnEncode: Encode a given buffer to a given output buffer.

Detailed descriptions of function behaviour, parameters and expected return values are available as docblocks in the cei.h file.

Each encoder is implemented as a set of pointers pointing to functions conforming to these types, grouped in the CeiEncoderFunctions structure. Specifying the CeiEncoderGetConfig fnGetConfig and CeiFnEncoderSetConfig fnSetConfig members is optional, as an encoder does not have to be configurable. If so desired, specify NULL. Implementation of the remaining functions is mandatory, however. If at least one of these functions isn't implemented and NULL is specified instead, the encoder will be considered as not implemented.

To register an implemented encoder with the element, add a new entry to the CeiEncoderType enum and add the CeiEncoderFunctions struct value to the table CeiEncoderFunctions ceiEncTable[] located in the cei_table.h header file. Note and match the order of items in that table, as a CeiEncoderType value is used as an index. Same goes for the size_t ceiEncConfigSizeTable[]. If configuration is not applicable, specify 0 at the appropriate index. If configuration is applicable, describe the configuration structure in the cei_enctypes.h header file and add its size to that table.

Maestro playback example

Table of content

- *Overview*
- *Hardware requirements*
- *Hardware modifications*
- *Preparation*
- *Running the demo*
- *Example configuration*
- *Functionality*
- *States*
- *Commands in detail*
- *Processing Time*

Overview The Maestro playback example demonstrates audio processing on the ARM cortex core utilizing the Maestro Audio Framework library.

The application is controlled by commands from a shell interface using serial console and the audio files are read from the SD card.

Depending on target platform or development board there are different modes and features of the demo supported.

- **Standard** - The mode demonstrates playback of encoded files from an SD card with up to 2 channels, up to 48 kHz sample rate and up to 16 bit width. This mode is enabled by default.
- **Multi-channel** - The mode demonstrates playback of raw PCM files from an SD card with 2 or 8 channels, 96kHz sample rate and 32 bit width. The decoders and synchronous sample rate converter are not supported in this mode. The Multi-channel mode is only supported on selected platforms, see the table below. The *Example configuration* section contains information on how to enable it.

As shown in the table below, the application is supported on several development boards and each development board may have certain limitations, some development boards may also require hardware modifications or allow to use of an audio expansion board. Therefore, please check the supported features and *Hardware modifications* or *Example configuration* sections before running the demo.

Limitations:

- Note:
  - *LPCXPresso55s69* - MCUXpresso IDE project default debug console is semihost
- Decoder:
  - **AAC:**
    - The reference decoder is supported only in the MCUXpresso IDE and ARMGCC.
  - **FLAC:**
    - *LPCXpresso55s69* - When playing FLAC audio files with too small frame size (block size), the audio output may be distorted because the board is not fast enough.
  - **OPUS:**
    - *LPCXpresso55s69* - The decoder is disabled due to insufficient memory may be distorted because the board is not fast enough.
- Sample rate converter:
  - **SSRC:**
    - *LPCXpresso55s69* - When a memory allocation ERROR occurs, it is necessary to disable the SSRC element due to insufficient memory.

Known issues:

- Decoder:
  - **MP3:**
    - The reference decoder has issues with some of the files. One of the channels can be sometimes distorted or missing parts of the signal.
  - **OPUS:**
    - The decoder doesn't support all the combinations of frame sizes and sample rates. The application might crash when playing an unspupported file.

More information about supported features can be found on the *Supported features* page.

Hardware requirements

- Desired development board
- Micro USB cable
- Headphones with 3.5 mm stereo jack
- SD card with supported audio files
- Personal computer
- Optional:
  - Audio expansion board AUD-EXP-42448 (REV B)

Hardware modifications Some development boards need some hardware modifications to run the application. If the development board is not listed here, its default setting is required.

- *EVKB-MIMXRT1170:*
  1. Please remove below resistors if on board wifi chip is not DNP:
     - R228, R229, R232, R234
  2. Please make sure R136 is weld for GPIO card detect.

Preparation

1. Connect a micro USB cable between the PC host and the debug USB port on the development board.
2. Open a serial terminal with the following settings:
   - 115200 baud rate
   - 8 data bits
   - No parity
   - One stop bit
   - No flow control
3. Download the program to the target board.
4. Insert the headphones into the Line-Out connector (headphone jack) on the development board.
5. Either press the reset button on your development board or launch the debugger in your IDE to begin running the demo.

Running the demo When the example runs successfully, you should see similar output on the serial terminal as below:

```
**********************************
Maestro audio playback demo start
**********************************

[APP_Main_Task] started

Copyright  2022  NXP
[APP_SDCARD_Task] start
[APP_Shell_Task] start

>> [APP_SDCARD_Task] SD card drive mounted
```

Type help to see the command list. Similar description will be displayed on serial console (*If multi-channel playback mode is enabled, the description is slightly different*):

```
>> help

"help": List all the registered commands

"exit": Exit program

"version": Display component versions

"file": Perform audio file decode and playback

  USAGE: file [stop|pause|volume|seek|play|list|info]
```

(continues on next page)

(continued from previous page)

```
stop              Stops actual playback.
pause             Pause actual track or resume if already paused.
volume <volume>   Set volume. The volume can be set from 0 to 100.
seek <seek_time>  Seek currently paused track. Seek time is absolute time in milliseconds.
play <filename>   Select audio track to play.
list              List audio files available on mounted SD card.
info              Prints playback info.
```

Details of commands can be found *here*.

Example configuration The example can be configured by user. Before configuration, please check the *table* to see if the feature is supported on the development board.

- **Enable Multi-channel mode:**
  - Add the MULTICHANNEL_EXAMPLE symbol to preprocessor defines on project level.
  - Connect AUD-EXP-42448 (see the point below).
- **Connect AUD-EXP-42448:**
  - *EVKC-MIMXRT1060:*
    1. Disconnect the power supply for safety reasons.
    2. Insert AUD-EXP-42448 into J19 to be able to use the CS42448 codec for multichannel output.
    3. Uninstall J99.
    4. Set the DEMO_CODEC_WM8962 macro to 0 in the app_definitions.h file
    5. Set the DEMO_CODEC_CS42448 macro to 1 in the app_definitions.h file.

Functionality The file play <filename> command calls the STREAMER_file_Create or STREAMER_PCM_Create function from the app_streamer.c file depending on the selected mode.

- When the *Standard* mode is enabled, the command calls the STREAMER_file_Create function that creates a pipeline with the following elements:
  - ELEMENT_FILE_SRC_INDEX
  - ELEMENT_DECODER_INDEX
  - ELEMENT_SRC_INDEX (If SSRC_PROC is defined)
  - ELEMENT_SPEAKER_INDEX
- When the *Multi-channel* mode is enabled, the command calls STREAMER_PCM_Create function, which creates a pipeline with the following elements:
  - ELEMENT_FILE_SRC_INDEX (PCM format only)
  - ELEMENT_SPEAKER_INDEX
  - *Note:*
    * If the input file is an 8 channel PCM file, output to all 8 channels is available. The properties of the PCM file are set in the app_streamer.c file using file source properties sent to the streamer:
      - PROP_FILESRC_SET_SAMPLE_RATE - default value is 96000 [Hz]
      - PROP_FILESRC_SET_NUM_CHANNELS - default value is 8
      - PROP_FILESRC_SET_BIT_WIDTH - default value is 32

Playback itself can be started with the STREAMER_Start function.

Each of the elements has several properties that can be accessed using the streamer_get_property or streamer_set_property function. These properties allow a user to change the values of the appropriate elements. The list of properties can be found in streamer_element_properties.h. See the example of setting property value in the following piece of code from the app_streamer.c file:

```
ELEMENT_PROPERTY_T prop;

EXT_PROCESS_DESC_T ssrc_proc = {SSRC_Proc_Init, SSRC_Proc_Execute, SSRC_Proc_Deinit,␣
↪&get_app_data()->proc_args};

prop.prop = PROP_SRC_PROC_FUNCPTR;
prop.val  = (uintptr_t)&ssrc_proc;

if (streamer_set_property(streamer, 0, prop, true) != 0)
{
    return -1;
}

prop.prop = PROP_AUDIOSINK_SET_VOLUME;
prop.val  = volume;
streamer_set_property(streamer, 0, prop, true);
```

Some of the predefined values can be found in the streamer_api.h.

States The application can be in 3 different states:

- Idle
- Running
- Paused

In each state, each command can have a different behavior. For more information, see *Commands in detail* section.

Commands in detail The applicatin is controlled by commands from the shell interface and the available commands for the selected mode can be displayed using the help command. Commands are processed in the cmd.c file.

- *help, version*
- *file stop*
- *file pause*
- *file volume <volume>*
- *file seek <seek_time>*
- *file play <filename>*
- *file list*
- *file info*

Legend for diagrams:

```
flowchart TD
    classDef function fill:#69CA00
    classDef condition fill:#0EAFE0
    classDef state fill:#F9B500
```

```
    classDef error fill:#F54D4D

    A((State)):::state
    B{Condition}:::condition
    C[Error message]:::error
    D[Process function]:::function
```

help, version

```
flowchart TD
    classDef function fill:#69CA00
    classDef condition fill:#0EAFE0
    classDef state fill:#F9B500
    classDef error fill:#F54D4D

    A((Idle)):::state --> D[Write help or version]:::function
    B((Running)):::state --> D
    C((Paused)):::state --> D
    D-->E((No state
    change)):::state
```

file stop

```
flowchart TD
    classDef function fill:#69CA00
    classDef condition fill:#0EAFE0
    classDef state fill:#F9B500
    classDef error fill:#F54D4D

    B((Idle)):::state --> B
    C((Running)):::state -->E((Idle)):::state
    D((Paused)):::state -->E
```

file pause

```
flowchart TD
    classDef function fill:#69CA00
    classDef condition fill:#0EAFE0
    classDef state fill:#F9B500
    classDef error fill:#F54D4D

    B((Idle)):::state --> B
    C((Running)):::state -->E((Paused)):::state
    D((Paused)):::state -->F((Running)):::state
```

file volume <volume>

```
flowchart TD
    classDef function fill:#69CA00
    classDef condition fill:#0EAFE0
    classDef state fill:#F9B500
    classDef error fill:#F54D4D

    B((Idle)):::state --> M[Error: Play a track first]:::error
    C((Running)):::state --> G{Volume
    parameter
```

```
    empty?}:::condition
    D((Paused)):::state --> G
    G -- Yes -->H[Error: Enter volume parameter]:::error
    G -- No -->I{Volume
    in range?}:::condition
    I -- No -->J[Error: invalid value]:::error
    I -- Yes -->K[Set volume]:::function
    J --> L((No state
    change)):::state
    K --> L
    H--> L
```

file seek <seek_time> The seek argument is only supported in the Standard mode.

```
flowchart TD
    classDef function fill:#69CA00
    classDef condition fill:#0EAFE0
    classDef state fill:#F9B500
    classDef error fill:#F54D4D

    B((Idle)):::state --> E[Error: First select
    an audio track to play]:::error
    E-->B
    C((Running)):::state --> F[Error: First
    pause the track]:::error
    F --> C
    D((Paused)):::state --> G{Seek
    parameter
    empty?}:::condition
    G --No --> H{AAC file?}:::condition
    G --Yes --> I[Error: Enter
    a seek time value]:::error
    I-->N((Paused)):::state;
    H --Yes -->J[Error: The AAC decoder
    does not support
    the seek command]:::error
    J-->N
    H --No -->K{Seek
    parameter
    positive?}:::condition
    K --No -->L[Error: The seek
    time must be
    a positive value]:::error
    L-->N
    K --Yes -->M[Seek the file]:::function
    M-->N
```

file play <filename>

```
flowchart TD
    classDef function fill:#69CA00
    classDef condition fill:#0EAFE0
    classDef state fill:#F9B500
    classDef error fill:#F54D4D

    C((Running)):::state --> Z[Error: First stop
    current track]:::error
```

```
D((Paused)):::state --> Z
B((Idle)):::state --> E{SD Card
inserted?}:::condition
E -- No -->F[Error: Insert SD
card]:::error
E -- Yes -->G{File
name
empty?}:::condition
G -- Yes -->H[Error: Enter
file name]:::error
G -- No -->I{File exists?}:::condition
I -- No -->O[Error: File
doesn't exist]:::error
I -- Yes -->J{Supported
format?}:::condition
J -- Yes -->K[Play the track]:::function
J -- No -->L[Error: Unsupported
file]:::error
K -->M((Running)):::state
L --> W((No state
change)):::state
O --> W
H --> W
F --> W
Z --> W
```

file list

```
flowchart TD
    classDef function fill:#69CA00
    classDef condition fill:#0EAFE0
    classDef state fill:#F9B500
    classDef error fill:#F54D4D

    B((Idle)):::state --> G{SD Card
    inserted?}:::condition
    C((Running)):::state --> G
    D((Paused)):::state --> G
    G -- Yes -->H[List supported files]:::function
    G -- No -->I[Error: Insert SD card]:::error
    I --> J((No state
    change)):::state
    H --> J
```

file info

```
flowchart TD
    classDef function fill:#69CA00
    classDef condition fill:#0EAFE0
    classDef state fill:#F9B500
    classDef error fill:#F54D4D

    B((Idle)):::state -->E[Write file info]:::function
    C((Running)):::state -->E
    D((Paused)):::state -->E
    E --> F((No state
        change)):::state
```

Processing Time Typical streamer pipeline execution times and their individual elements for the EVKC-MIMXRT1060 development board are presented in the following tables. The time spent on output buffers is not included in the traversal measurements. However, file reading time is accounted for. In the case of the WAV codec, the audio file was accessed in every pipeline run. Therefore, during each run, the file was read from the SD card. However, for the MP3 codec, where data must be processed in complete MP3 frames, the file was not read in every run. Instead, it was read periodically only when the codec buffer did not contain a complete frame of data.

For further details, please refer to the *Processing Time* document.

WAV	streamer	file_src	codec	SSRC_proc	speaker
48kHz	1.1 ms	850 µs	150 µs	70 µs	40 µs
44kHz	1.75 ms	850 µs	180 µs	670 µs	40 µs

MP3	streamer	file_src	codec	SSRC_proc	speaker
48 kHz with file read	2.9 ms	2.3 µs	450 µs	60 µs	50 µs
48 kHz without file read	0.5 ms	x	400 µs	40 µs	40 µs
44 kHz with file read	3.2 ms	2.3 µs	440 µs	400 µs	50 µs
44 kHz without file read	0.9 ms	x	440 µs	390 µs	40 µs

Maestro record example

Table of content

- *Overview*
- *Hardware requirements*
- *Hardware modifications*
- *Preparation*
- *Running the demo*
- *Example configuration*
- *Functionality*
- *States*
- *Commands in detail*
- *Processing Time*

Overview The Maestro record example demonstrates audio processing on the ARM cortex core utilizing the Maestro Audio Framework library.

The application is controlled by commands from a shell interface using serial console.

Depending on target platform or development board there are different modes and features of the demo supported.

- **Loopback** - The application demonstrates a loopback from the microphone to the speaker without any audio processing. Mono, stereo or multichannel mode can be used, depending on the hardware, see *table* below.
- **File recording** - The application takes audio samples from the microphone inputs and stores them to an SD card as an PCM file. The PCM file has following parameters:

  - Mono and stereo : 2 channels, 16kHz, 16bit width
  - Multi-channel (AUD-EXP-42448): 6 channels, 16kHz, 32bit width
- **Voice control** - The application takes audio samples from the microphone input and uses the VIT library to recognize wake words and voice commands. If a wake word or a voice command is recognized, the application write it to the serial terminal.
- **Encoding** - The application takes PCM samples from memory and sends them to the Opus encoder. The encoded data is stored in memory and compared to a reference. The result of the comparison is finally written into the serial terminal.

As shown in the table below, the application is supported on several development boards, and each development board may have certain limitations, some development boards may also require hardware modifications or allow to use of an audio expansion board. Therefore, please check the supported features and *Hardware modifications* or *Example configuration* sections before running the demo.

Limitations:

- Note:
  - *LPCXPresso55s69* - MCUXpresso IDE project default debug console is semihost
- Addition labraries
  - **VIT:**
    - The VIT is supported only in the MCUXpresso IDE and ARMGCC.
    - *LPCXpresso55s69* - The VIT is disabled by default due to insufficient memory. To enable it, see the *Example configuration* section.
    - *EVK-MCXN5XX* - Some VIT models can't fit into memory. In order to free some space it is necessary to disable SD card handling and opus encoder. To disable it, see the *Example configuration* section.
  - **VoiceSeeker:**
    - The VoiceSeeker is supported only in the MCUXpresso IDE and ARMGCC.
- Encoder
  - **OPUS:**
    - *LPCXpresso55s69* - The encoder is not supported due to insufficient memory.
- The File recording mode is not supported on *RW612BGA* development board due to missing SD card slot.

Known issues:

- *EVKB-MIMXRT1170* - After several tens of runs (the number of runs is not deterministic), the development board restarts because a power-up sequence is detected on the RESET pin (due to a voltage drop).

More information about supported features can be found on the *Supported features* page.

Hardware requirements

- Desired development board
- Micro USB cable
- Headphones with 3.5 mm stereo jack
- Personal computer
- Optional:
  - SD card for file output

  - Audio expansion board AUD-EXP-42448 (REV B)
- *LPCXpresso55s69:*
  - Source of sound with 3.5 mm stereo jack connector

Hardware modifications Some development boards need some hardware modifications to run the application. If the development board is not listed here, its default setting is required.

- *EVKB-MIMXRT1170:*
  1. Please remove below resistors if on board wifi chip is not DNP:
     - R228, R229, R232, R234
  2. Please make sure R136 is weld for GPIO card detect.
- *EVK-MCXN5XX:*
  - Short: JP7 2-3, JP8 2-3, JP10 2-3, JP11 2-3
- *RW612BGA:*
  - Connect: JP50; Disconnect JP9, JP11

Preparation

1. Connect a micro USB cable between the PC host and the debug USB port on the development board
2. Open a serial terminal with the following settings:
   - 115200 baud rate
   - 8 data bits
   - No parity
   - One stop bit
   - No flow control
3. Download the program to the target board.
4. Insert the headphones into the Line-Out connector (headphone jack) on the development board.
5. *LPCXpresso55s69:*
   - Insert source of sound to audio Line-In connector (headphone jack) on the development board.
6. Either press the reset button on your development board or launch the debugger in your IDE to begin running the demo.

Running the demo When the example runs successfully, you should see similar output on the serial terminal as below:

```
******************************
Maestro audio record demo start
******************************

Copyright 2022 NXP
[APP_SDCARD_Task] start
[APP_Shell_Task] start

>> [APP_SDCARD_Task] SD card drive mounted
```

Type help to see the command list. Similar description will be displayed on serial console:

```
>> help

"help": List all the registered commands

"exit": Exit program

"version": Display component versions

"record_mic": Record MIC audio and perform one (or more) of following actions:
- playback on codec
- perform VoiceSeeker processing
- perform voice recognition (VIT)
- store samples to a file.

USAGE: record_mic [audio|file|<file_name>|vit] 20 [<language>]
The number defines length of recording in seconds.

Please see the project defined symbols for the languages supported.
Then specify one of: en/cn/de/es/fr/it/ja/ko/pt/tr as the language parameter.
For voice recognition say supported WakeWord and in 3s frame supported command.
Please note that this VIT demo is near-field and uses 1 on-board microphone.

NOTES: This command returns to shell after the recording is finished.
       To store samples to a file, the "file" option can be used to create a file
       with a predefined name, or any file name (without whitespaces) can be specified
       instead of the "file" option.

"opus_encode": Initializes the streamer with the Opus memory-to-memory pipeline and
encodes a hardcoded buffer.
```

Details of commands can be found *here*.

Example configuration The example can be configured by user. There are several options how to configure the example settings, depending on the environment. For configuration using west and Kconfig, please follow the instructions here. Before configuration, please check the *table* to see if the feature is supported on the development board.

- **Connect AUD-EXP-42448:**
  - *EVKC-MIMXRT1060:*
    1. Disconnect the power supply for safety reasons.
    2. Insert AUD-EXP-42448 into J19 to be able to use the CS42448 codec for multichannel output.
    3. Uninstall J99.
    4. Set the DEMO_CODEC_WM8962 macro to 0 in the app_definitions.h file
    5. Set the DEMO_CODEC_CS42448 macro to 1 in the app_definitions.h file.
    6. Enable VoiceSeeker, see point bellow.
  - *Note:*
    * The audio stream is as follows:
      - Stereo INPUT 1 (J12) -> LINE 1&2 OUTPUT (J6)
      - Stereo INPUT 2 (J15) -> LINE 3&4 OUTPUT (J7)
      - MIC1 & MIC2 (P1, P2) -> LINE 5&6 OUTPUT (J8)
      - Insert the headphones into the different line outputs to hear the inputs.

        - To use the Stereo INPUT 1, 2, connect an audio source LINE IN jack.

- **Enable VoiceSeeker:**
  - On some development boards the VoiceSeeker is enabled by default, see the *table* above.
  - If more than one channel is used and VIT is enabled, the VoiceSeeker that combines multiple channels into one must be used, as VIT can only work with mono signal.
  - Using MCUXPresso IDE:
    - It is necessary to add VOICE_SEEKER_PROC symbol to preprocessor defines on project level:
      - (Project -> Properties -> C/C++ Build -> Settings -> MCU C Compiler -> Preprocessor)
  - Using Kconfig:
    - Enable the VoiceSeeker in the guiconfig using MCUX_PRJSEG_middleware.audio_voice.components.voice_seeker
- **Enable VIT:**
  - *LPCXpresso55s69 and MCX-N5XX:*
    - In MCUXPresso IDE (SDK package):
      1. Remove SD_ENABLED and STREAMER_ENABLE_FILE_SINK symbols from preprocessor defines on project level.
      2. Add VIT_PROC symbol to preprocessor defines on project level:
         - (Project -> Properties -> C/C++ Build -> Settings -> MCU C Compiler -> Preprocessor)
    - In armgcc in SDK package:
      1. Remove SD_ENABLED and STREAMER_ENABLE_FILE_SINK symbols from preprocessor defines in flags.cmake file.
      2. Remove OPUS_ENCODE=1 and STREAMER_ENABLE_ENCODER preprocessor defines in flags.cmake file.
      3. Add VIT_PROC symbol to preprocessor defines in flags.cmake file.
      4. Remove sdmmc_config.c,.h files from CMakeLists.txt file.
    - In Kconfig:
      1. Disable File sink MCUX_COMPONENT_middleware.audio_voice.maestro.element.file_sink.enable
      2. Make sure SD card support is disabled MCUX_COMPONENT_middleware.sdmmc.sd and MCUX_COMPONENT_middleware.sdmmc.host.usdhc
      3. Make sure sdmmc_config files (.c, .h) is excluded from project build
         - remove mcux_add_source function that adds the sources in reconfig.cmake in maestro_record/cm33_core0 folder
      4. Disable fatfs MCUX_COMPONENT_middleware.fatfs and MCUX_COMPONENT_middleware.fatfs.sd
      5. Disable file utils MCUX_COMPONENT_middleware.audio_voice.maestro.file_utils.enable
      6. Make sure Opus encoder is disabled MCUX_COMPONENT_middleware.audio_voice.maestro.element.encoder.opus.enable
      7. Make sure VIT_PROC symbol is defined

        - remove mcux_remove_macro function that removes the VIT_PROC preprocessor definition in reconfig.cmake in maestro_record folder

    8. Make sure VIT processing is enabled MCUX_PRJSEG_middleware.audio_voice.components.vit

- **VIT model generation:**
  - For custom VIT model generation (defining own wake words and voice commands) please use https://vit.nxp.com/
- **Disable SD card handling:**
  - In MCUXPresso IDE:
    - Remove SD_ENABLED and STREAMER_ENABLE_FILE_SINK symbols from preprocessor defines on project level:
      - (Project -> Properties -> C/C++ Build -> Settings -> MCU C Compiler -> Preprocessor)
  - In armgcc in SDK package:
    - Remove SD_ENABLED and STREAMER_ENABLE_FILE_SINK symbols from preprocessor defines in flags.cmake file.
  - In Kconfig:
    1. Disable File sink MCUX_COMPONENT_middleware.audio_voice.maestro.element.file_sink.enable
    2. Make sure SD card support is disabled MCUX_COMPONENT_middleware.sdmmc.sd

Functionality The record_mic or opus_encode command calls the STREAMER_mic_Create or STREAMER_opusmem2mem_Create function from the app_streamer.c file depending on the selected mode.

- When the *Loopback* mode is selected, the command calls the STREAMER_mic_Create function that creates a pipeline with the following elements:
  - ELEMENT_MICROPHONE_INDEX
  - ELEMENT_SPEAKER_INDEX
- When the *File recording* mode is selected, the command calls the STREAMER_mic_Create function that creates a pipeline with the following elements: - ELEMENT_MICROPHONE_INDEX - ELEMENT_FILE_SINK_INDEX
- When the *Voice control* mode is selected, the command calls the STREAMER_mic_Create function that creates a pipeline with the following elements: - ELEMENT_MICROPHONE_INDEX - ELEMENT_VOICESEEKER_INDEX (If VOICE_SEEKER_PROC is defined) - ELEMENT_VIT_INDEX
- When the Encoding mode is selected, the command calls the STREAMER_opusmem2mem_Create function that creates a pipeline with the following elements: - ELEMENT_MEM_SRC_INDEX - ELEMENT_ENCODER_INDEX - ELEMENT_MEM_SINK_INDEX

Recording itself can be started with the STREAMER_Start function.

Each of the elements has several properties that can be accessed using the streamer_get_property or streamer_set_property function. These properties allow a user to change the values of the appropriate elements. The list of properties can be found in streamer_element_properties.h. See the example of setting property value in the following piece of code from the app_streamer.c file:

```
    ELEMENT_PROPERTY_T prop;

    prop.prop = PROP_MICROPHONE_SET_NUM_CHANNELS;
    prop.val  = DEMO_MIC_CHANNEL_NUM;
    streamer_set_property(handle->streamer, 0, prop, true);

    prop.prop = PROP_MICROPHONE_SET_BITS_PER_SAMPLE;
    prop.val  = DEMO_AUDIO_BIT_WIDTH;
    streamer_set_property(handle->streamer, 0, prop, true);

    prop.prop = PROP_MICROPHONE_SET_FRAME_MS;
    prop.val  = DEMO_MIC_FRAME_SIZE;
    streamer_set_property(handle->streamer, 0, prop, true);

    prop.prop = PROP_MICROPHONE_SET_SAMPLE_RATE;
    prop.val  = DEMO_AUDIO_SAMPLE_RATE;
    streamer_set_property(handle->streamer, 0, prop, true);
```

Some of the predefined values can be found in the streamer_api.h.

States The application can be in 2 different states:

- Idle
- Running

Commands in detail

- *help, version*
- *record_mic audio <time>*
- *record_mic file <time>*
- *record_mic <file_name> <time>*
- *record_mic vit <time> <language>*
- *opus_encode*

Legend for diagrams:

```
flowchart TD
    classDef function fill:#69CA00
    classDef condition fill:#0EAFE0
    classDef state fill:#F9B500
    classDef error fill:#F54D4D

    A((State)):::state
    B{Condition}:::condition
    C[Error message]:::error
    D[Process function]:::function
```

help, version

```
flowchart TD
    classDef function fill:#69CA00
    classDef condition fill:#0EAFE0
    classDef state fill:#F9B500
    classDef error fill:#F54D4D

    A((Idle)):::state --> C[Write help or version]:::function
```

```
B((Running)):::state --> C
C --> E((No state
change)):::state
```

record_mic audio <time>

```
flowchart TD
    classDef function fill:#69CA00
    classDef condition fill:#0EAFE0
    classDef state fill:#F9B500
    classDef error fill:#F54D4D

    B((Idle)):::state --> D{time
    > 0 ?}:::condition
    D -- Yes --> F[recording]:::function
    D -- No --> E[Error: Record length
    must be greater than 0]:::error
    E --> B
    F --> C((Running)):::state
    C -->G{time
    expired?}:::condition
    G -- No --> C
    G -- Yes --> B
```

record_mic file <time>/record_mic <file_name> <time>

```
flowchart TD
    classDef function fill:#69CA00
    classDef condition fill:#0EAFE0
    classDef state fill:#F9B500
    classDef error fill:#F54D4D

    B((Idle)):::state --> C{time
    > 0 ?}:::condition
    C -- Yes --> D{SD card
    inserted?}:::condition
    C -- No --> E[Error: Record length
    must be greater than 0]:::error
    E --> B
    D -- Yes --> G{Custom
    file name?}:::condition
    G -- Yes --> H[Create custom
    file name]:::function
    G -- No --> I[Create default
    file name]:::function
    H --> J[Recording]:::function
    I --> J
    J --> K((Running)):::state
    K -->L{time
    expired?}:::condition
    L -- No --> K
    L -- Yes --> B
    D -- No --> F[Error: Insert SD
    card first]:::error
    F --> B
```

record_mic vit <time> <language>

```
flowchart TD
    classDef function fill:#69CA00
    classDef condition fill:#0EAFE0
    classDef state fill:#F9B500
    classDef error fill:#F54D4D

    B((Idle)):::state --> C{time
    > 0 ?}:::condition
    C -- Yes --> E{Selected
    language?}:::condition
    C -- No --> D[Error: Record length
    must be greater than 0]:::error
    D --> B
    E -- Yes --> G{Supported
    language?}:::condition
    E -- No --> F[Error: Language
    not selected]:::error
    F -->B
    G -- Yes -->I[Recording with
    voice recognition]:::function
    G -- No -->H[Error: Language not supported]:::error
    H --> B
    I --> J((Running)):::state
    J -->K{time
    expired?}:::condition
    K -- No --> J
    K -- Yes --> B
```

opus_encode

```
flowchart TD
    classDef function fill:#69CA00
    classDef condition fill:#0EAFE0
    classDef state fill:#F9B500
    classDef error fill:#F54D4D

    B((Idle)):::state -->C[Encode file]:::function
    C -->D[Check result]:::function
    D -->B
```

Processing Time Typical execution times of the streamer pipeline for the EVKC-MIMXRT1060 development board are detailed in the following table. The duration spent on output buffers and reading from the microphone is excluded from traversal measurements. Three measured pipelines were considered. The first involves a loopback from microphone to speaker, supporting both mono and stereo configurations. The second pipeline is a mono voice control setup, comprising microphone and VIT blocks. The final pipeline is a stereo voice control setup, integrating microphone, voice seeker, and VIT blocks.

For further details of execution times on individual elements, please refer to the *Processing Time* document.

	streamer
microphone -> speaker 1 channel	40 µs
microphone -> speaker 2 channels	115 µs
microphone -> VIT	7.4 ms
microphone -> voice seeker -> VIT	9.9 ms

Maestro sync example

Table of content

- *Overview*
- *Hardware requirements*
- *Hardware modifications*
- *Preparation*
- *Running the demo*
- *Example configuration*
- *Functionality*
- *States*
- *Commands in detail*

Overview The Maestro sync example demonstrates the use of synchronous pipelines (Tx and Rx in this case) processing in a single streamer task on the ARM cortex core utilizing the Maestro Audio Framework library.

The application is controlled by commands from a shell interface using serial console.

The feature is useful for testing the latency of the pipeline or implementing algorithms requiring reference signals such as echo cancellation. The VoiceSeeker library available in this example is not featuring AEC (Acoustic Echo Cancellation), but NXP is offering it in the premium version of the library. More information about the premium version can be found at VoiceSeeker. page. The demo uses two pipelines running synchronously in a single streamer task:

1. Playback (Tx) pipeline:
    - Playback of audio data in PCM format stored in flash memory to the audio Line-Out connector (speaker).
2. Recording (Rx) pipline:
    - Record audio data using a microphone.
    - VoiceSeeker processing.
    - Wake words + voice commands recognition.
    - Save the VoiceSeeker output to the voiceseeker_output.pcm file on the SD card.

As shown in the table below, the application is supported on several development boards, and each development board may have certain limitations, some development boards may also require hardware modifications or allow to use of an audio expansion board. Therefore, please check the supported features and *Hardware modifications* or *Example configuration* sections before running the demo.

Limitations:

- Addition labraries
  - **VIT:**
    - The VIT is supported only in the MCUXpresso IDE.
  - **VoiceSeeker:**
    - The VoiceSeeker is supported only in the MCUXpresso IDE.

Known issues:

- No known issues.

More information about supported features can be found on the *Supported features* page.

Hardware requirements

- Desired development board
- Micro USB cable
- Speaker with 3.5 mm stereo jack
- Personal computer
- Optional:
  - SD card for file output

Hardware modifications Some development boards need some hardware modifications to run the application. If the development board is not listed here, its default setting is required.

- *EVKC-MIMXRT1060:*
  1. Please make sure resistors below are removed to be able to use SD-Card.
     - R368, R347, R349, R365, R363
  2. Please Make sure J99 is installed.

Preparation

1. Connect a micro USB cable between the PC host and the debug USB port on the development board
2. Open a serial terminal with the following settings:
   - 115200 baud rate
   - 8 data bits
   - No parity
   - One stop bit
   - No flow control
3. Download the program to the target board.
4. Insert the speaker into the Line-Out connector (headphone jack) on the development board.
5. *Optional:* Insert an SD card into the SD card slot to record to the VoiceSeeker output.
6. Either press the reset button on your development board or launch the debugger in your IDE to begin running the demo.

Running the demo When the example runs successfully, you should see similar output on the serial terminal as below:

```
****************************
Maestro audio sync demo start
****************************

Copyright 2022 NXP
[APP_SDCARD_Task] start
[APP_Shell_Task] start

>> [APP_SDCARD_Task] SD card drive mounted
```

Type help to see the command list. Similar description will be displayed on serial console:

```
>> help

"help": List all the registered commands

"exit": Exit program

"version": Display component versions

"start [nosdcard]": Starts a streamer task.
            - Initializes the streamer with the Memory->Speaker pipeline and with
              the Microphone->VoiceSeeker->VIT->SDcard pipeline.
            - Runs repeatedly until stop command.
    nosdcard - Doesn't use SD card to store data.

"stop": Stops a running streamer:

"debug [on|off]": Starts / stops debugging.
              - Starts / stops saving VoiceSeeker input data (reference and microphone data)
                to SDRAM.
              - After the stop command, this data is transferred to the SD card.
```

Details of commands can be found *here*.

Example configuration The example can be configured by user. Before configuration, please check the *table* to see if the feature is supported on the development board.

- **Enable the premium version of VoiceSeeker:**
  - The premium version of the VoiceSeeker library with AEC is API compatible with this example.
  - To get the premium version, please visit VoiceSeeker page.
  - The following steps are required to run this example with the VoiceSeeker&AEC library.
    - Link the voiceseeker.a library instead of voiceseeker_no_aec.a.
    - Set the RDSP_ENABLE_AEC definition to 1U in the voiceseeker.h file
- **VIT model generation:**
  - For custom VIT model generation (defining own wake words and voice commands) please use https://vit.nxp.com/

Functionality The start <nosdcard> command calls the STREAMER_Create function from the app_streamer.c file that creates pipelines with the following elements:

- Playback pipeline:

  - ELEMENT_MEM_SRC_INDEX
  - ELEMENT_SPEAKER_INDEX
- Record pipeline:
  - ELEMENT_MICROPHONE_INDEX
  - ELEMENT_VOICESEEKER_INDEX
  - ELEMENT_VIT_PROC_INDEX
  - ELEMENT_FILE_SINK_INDEX (If the nosdcard argument is not used)

Processing itself can be started with the STREAMER_Start function.

Each of the elements has several properties that can be accessed using the streamer_get_property or streamer_set_property function. These properties allow a user to change the values of the appropriate elements. The list of properties can be found in streamer_element_properties.h. See the example of setting property value in the following piece of code from the app_streamer.c file:

```
ELEMENT_PROPERTY_T prop;

MEMSRC_SET_BUFFER_T buf;
buf.location = (int8_t *)TESTAUDIO_DATA;
buf.size     = TESTAUDIO_LEN;

prop.prop = PROP_MEMSRC_SET_BUFF;
prop.val  = (uintptr_t)&buf;
if (STREAM_OK != streamer_set_property(handle->streamer, 0, prop, true))
{
    return kStatus_Fail;
}

prop.prop = PROP_MEMSRC_SET_MEM_TYPE;
prop.val  = AUDIO_DATA;
if (STREAM_OK != streamer_set_property(handle->streamer, 0, prop, true))
{
    return kStatus_Fail;
}

prop.prop = PROP_MEMSRC_SET_SAMPLE_RATE;
prop.val  = DEMO_SAMPLE_RATE;
if (STREAM_OK != streamer_set_property(handle->streamer, 0, prop, true))
{
    return kStatus_Fail;
}
```

Some of the predefined values can be found in the streamer_api.h.

States The application can be in 2 different states:

- Idle
- Running

Commands in detail

- *help, version*
- *start [nosdcard]*
- *stop*
- *debug [on|off]*

Legend for diagrams:

```
flowchart TD
    classDef function fill:#69CA00
    classDef condition fill:#0EAFE0
    classDef state fill:#F9B500
    classDef error fill:#F54D4D

    A((State)):::state
    B{Condition}:::condition
    C[Error message]:::error
    D[Process function]:::function
```

help, version

```
flowchart TD
    classDef function fill:#69CA00
    classDef condition fill:#0EAFE0
    classDef state fill:#F9B500
    classDef error fill:#F54D4D

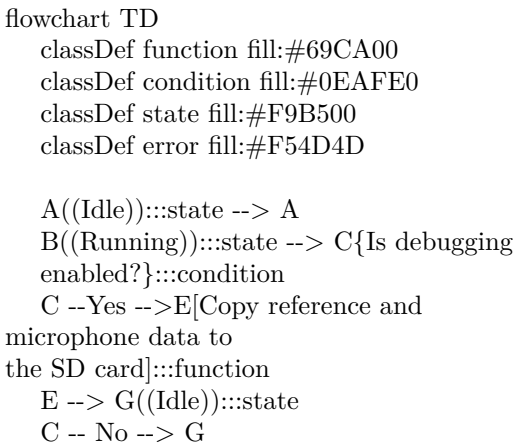
    A((Idle)):::state --> C[Write help or version]:::function
    B((Running)):::state --> C
    C --> E((No state
    change)):::state
```

start [nosdcard]

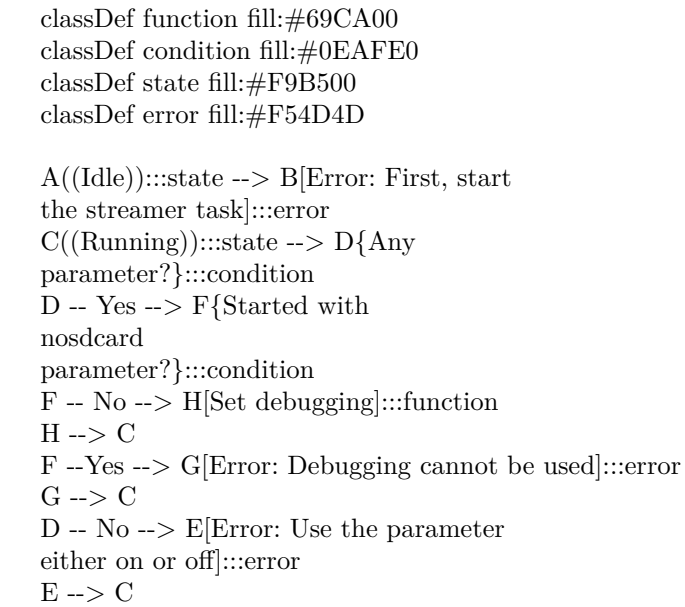
```
flowchart TD
    classDef function fill:#69CA00
    classDef condition fill:#0EAFE0
    classDef state fill:#F9B500
    classDef error fill:#F54D4D

    A((Idle)):::state --> B{nosdcard
    parameter?}:::condition
    B -- Yes --> CH[Playing to Line-out and
    recording]:::function
    CH --> L((Running)):::state
    B -- No --> C{Is SD card
    inserted?}:::condition
    C -- Yes --> E[Playing to Line-out and
    recording to SD card]:::function
    E --> F((Running)):::state
    F --> G{Debugging
    is enabled?}:::condition
    G -- No --> F
    G -- Yes --> H[Save reference and
    microphone data to SDRAM]:::function
    H --> F
    C -- No --> D[Error: Insert SD
card first]:::error
    D --> A
    J((Running)):::state --> K[Error: The streamer task is
    already running]:::error
    K --> J
```

stop



debug [on | off]



Maestro USB microphone example

Table of content

- *Overview*
- *Hardware requirements*
- *Hardware modifications*
- *Preparation*
- *Running the demo*
- *Example configuration*
- *Functionality*
- *States*
- *Commands in detail*

Overview The Maestro USB microphone example demonstrates audio processing on the ARM cortex core utilizing the Maestro Audio Framework library.

The application is controlled by commands from a shell interface using serial console.

The development board will be enumerated as a USB audio class 2.0 device on the USB host. The application takes audio samples from the microphone inputs and sends them to the USB host via the USB bus. User will see the volume levels obtained from the USB host but this is only an example application. To leverage the volume values, the demo has to be modified.

As shown in the table below, the application is supported on several development boards, and each development board may have certain limitations, some development boards may also require hardware modifications or allow to use of an audio expansion board. Therefore, please check the supported features and *Hardware modifications* or *Example configuration* sections before running the demo.

Limitations:

- *Note:*
  1. When connected to MacBook, change the PCM format from (0x02,0x00,) to (0x01,0x00,) in the g_config_descriptor[CONFIG_DESC_SIZE] in the usb_descriptor.c file. Otherwise, it can't be enumerated and noise is present when recording with the QuickTime player because the sampling frequency and bit resolution do not match.
  2. When device functionality is changed, please uninstall the previous PC driver to make sure the device with changed functionality can run normally.
  3. If you're having audio problems on Windows 10 for recorder, please disable signal enhancement as the following if it is enabled and have a try again.

Known issues:

- No known issues.

More information about supported features can be found on the *Supported features* page.

Hardware requirements

- Desired development board
- 2x Micro USB cable
- Personal Computer
- *LPCXpresso55s69:*
  - Source of sound with 3.5 mm stereo jack connector

Hardware modifications Some development boards need some hardware modifications to run the application. If the development board is not listed here, its default setting is required.

Preparation

1. Connect the first micro USB cable between the PC host and the debug USB port on the development board
2. Open a serial terminal with the following settings:
   - 115200 baud rate
   - 8 data bits
   - No parity
   - One stop bit

- No flow control

3. Download the program to the target board.
4. *LPCXpresso55s69:*
   - Insert source of sound to Audio Line-In connector (headphone jack) on the development board.
5. Connect the second micro USB cable between the PC host and the USB port on the development board.
6. Either press the reset button on your development board or launch the debugger in your IDE to begin running the demo.

Running the demo When the example runs successfully, you should see similar output on the serial terminal as below:

```
**************************************************
Maestro audio USB microphone solutions demo start
**************************************************

Copyright 2022 NXP
[APP_Shell_Task] start

>> usb_mic -1

Starting maestro usb microphone application
The application will run until the board restarts
[STREAMER] Message Task started
Starting recording
[STREAMER] start usb microphone
Set Cur Volume : 1f00
```

Type help to see the command list. Similar description will be displayed on serial console:

```
>> help

"help": List all the registered commands

"exit": Exit program

"version": Display component versions

"usb_mic": Record MIC audio and playback to the USB port as an audio 2.0
        microphone device.

  USAGE: usb_mic <seconds>
    <seconds> Time in seconds how long the application should run.
              When you enter a negative number the application will
              run until the board restarts.
  EXAMPLE:  The application will run for 20 seconds: usb_mic 20
```

Details of commands can be found *here*.

Example configuration The example only supports one mode and do not support any additional libraries, so the example can't be configured by user.

Functionality The usb_mic command calls the STREAMER_mic_Create function from the app_streamer.cfile that creates pipeline with the following elements: - ELEMENT_MICROPHONE_INDEX - ELEMENT_USB_SINK_INDEX

Recording itself can be started with the STREAMER_Start function.

Each of the elements has several properties that can be accessed using the streamer_get_property or streamer_set_property function. These properties allow a user to change the values of the appropriate elements. The list of properties can be found in streamer_element_properties.h. See the example of setting property value in the following piece of code from the app_streamer.c file:

```
ELEMENT_PROPERTY_T prop;

prop.prop = PROP_MICROPHONE_SET_SAMPLE_RATE;
prop.val  = AUDIO_SAMPLING_RATE;

streamer_set_property(handle->streamer, 0, prop, true);

prop.prop = PROP_MICROPHONE_SET_NUM_CHANNELS;
prop.val  = 1;

streamer_set_property(handle->streamer, 0, prop, true);

prop.prop = PROP_MICROPHONE_SET_FRAME_MS;
prop.val  = 1;

streamer_set_property(handle->streamer, 0, prop, true);
```

Some of the predefined values can be found in the streamer_api.h.

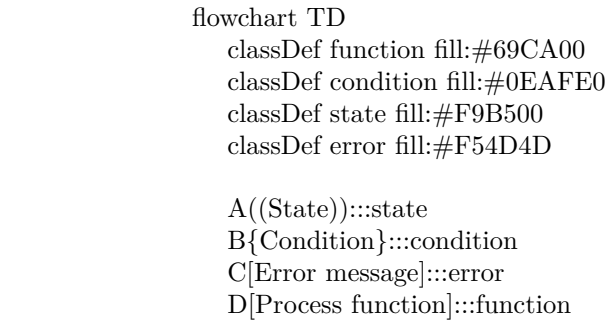
States The application can be in 2 different states:

- Idle
- Running

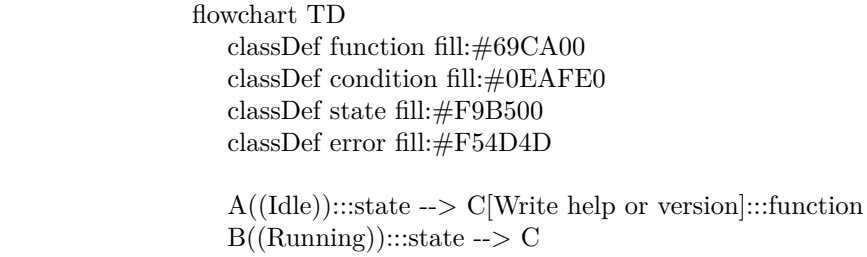
Commands in detail

- *help, version*
- *usb_mic <seconds>*

Legend for diagrams:



help, version



```
C --> E((No state
change)):::state
```

usb_mic <seconds>

```
flowchart TD
    classDef function fill:#c6d22c
    classDef condition fill:#7cb2de
    classDef state fill:#fcb415
    classDef error fill:#FF999C

    B((Idle)):::state -->C{seconds
    == 0?}:::condition
    C -- No --> E{seconds
    < 0?}:::condition
    C -- Yes --> D[Error: Incorrect
    command parameter]:::error
    D -->B
    E -- Yes --> G[recording]:::function
    G --> H((Running)):::state
    H --> H
    E -- No --> F[recording]:::function
    F --> I((Running)):::state
    I --> J{seconds
    expired?}:::condition
    J -- No -->I
    J -- Yes --> B
```

Maestro USB speaker example

Table of content

- *Overview*
- *Hardware requirements*
- *Hardware modifications*
- *Preparation*
- *Running the demo*
- *Example configuration*
- *Functionality*
- *States*
- *Commands in detail*

Overview The Maestro USB speaker example demonstrates audio processing on the ARM cortex core utilizing the Maestro Audio Framework library.

The application is controlled by commands from a shell interface using serial console.

The development board will be enumerated as a USB audio class 2.0 device on the USB host. The application takes audio samples from the USB host and sends them to the audio Line-Out port. User will see the volume levels obtained from the USB host but this is only an example application. To leverage the volume values, the demo has to be modified.

Depending on target platform or development board there are different modes and features of the demo supported.

- **Standard** - The mode demonstrates playback with up to 2 channels, up to 48 kHz sample rate and up to 16 bit width. This mode is enabled by default.
- **Multi-Channel** - In this mode the device is enumerated as a UAC 5.1. This mode is disabled by default. See the *Example configuration* section to see how to enable the mode.
  - When playing an 5.1 audio file, the example sends only the front-left and front-right channels to the audio Line-Out port (the other channels are ignored), since this example only supports on-board codecs with stereo audio output.

As shown in the table below, the application is supported on several development boards, and each development board may have certain limitations, some development boards may also require hardware modifications or allow to use of an audio expansion board. Therefore, please check the supported features and *Hardware modifications* or *Example configuration* sections before running the demo.

Limitations:

- *Note:*
  - If the USB device audio speaker example uses an ISO IN feedback endpoint, please attach the device to a host like PC which supports feedback function. Otherwise, there might be attachment issue or other problems.

Known issues:

- No known issues.

More information about supported features can be found on the *Supported features* page.

Hardware requirements

- Desired development board
- 2x Micro USB cable
- Personal Computer
- Headphones with 3.5 mm stereo jack

Hardware modifications Some development boards need some hardware modifications to run the application. If the development board is not listed here, its default setting is required.

Preparation

1. Connect the first micro USB cable between the PC host and the debug USB port on the development board
2. Open a serial terminal with the following settings:
   - 115200 baud rate
   - 8 data bits
   - No parity
   - One stop bit
   - No flow control
3. Download the program to the target board.
4. Connect the second micro USB cable between the PC host and the USB port on the development board.

5. Insert the headphones into Line-Out connector (headphone jack) on the development board.
6. Either press the reset button on your development board or launch the debugger in your IDE to begin running the demo.

Running the demo When the example runs successfully, you should see similar output on the serial terminal as below:

```
**********************************************
Maestro audio USB speaker solutions demo start
**********************************************

Copyright 2022 NXP
[APP_Shell_Task] start

>> usb_speaker -1

Starting maestro usb speaker application
The application will run until the board restarts
[STREAMER] Message Task started
Starting playing
[STREAMER] start usb speaker
Set Cur Volume : fbd5
```

Type help to see the command list. Similar description will be displayed on serial console:

```
>> help

"help": List all the registered commands

"exit": Exit program

"version": Display component versions

"usb_speaker": Play data from the USB port as an audio 2.0
               speaker device.

 USAGE:      usb_speaker <seconds>
  <seconds>  Time in seconds how long the application should run.
             When you enter a negative number the application will
             run until the board restarts.
  EXAMPLE:   The application will run for 20 seconds: usb_speaker 20
```

Details of commands can be found *here*.

Example configuration The example can be configured by user. Before configuration, please check the *table* to see if the feature is supported on the development board.

- **Enable Multi-channel mode:**
  - The feature can be enabled by set the USB_AUDIO_CHANNEL5_1 macro to 1U in the usb_device_descriptor.h file.
  - *Note:* When device functionality is changed, such as UAC 5.1, please uninstall the previous PC driver to make sure the device with changed functionality can run normally.

Functionality The Usb_speaker command calls the STREAMER_speaker_Create function from the app_streamer.cfile that creates pipeline with the following elements: - ELEMENT_USB_SRC_INDEX - ELEMENT_SPEAKER_INDEX

Playback itself can be started with the STREAMER_Start function.

Each of the elements has several properties that can be accessed using the streamer_get_property or streamer_set_property function. These properties allow a user to change the values of the appropriate elements. The list of properties can be found in streamer_element_properties.h. See the example of setting property value in the following piece of code from the app_streamer.c file:

```
ELEMENT_PROPERTY_T prop;

prop.prop = PROP_USB_SRC_SET_SAMPLE_RATE;
prop.val  = AUDIO_SAMPLING_RATE;

streamer_set_property(handle->streamer, 0, prop, true);

prop.prop = PROP_USB_SRC_SET_NUM_CHANNELS;
prop.val  = 2;

streamer_set_property(handle->streamer, 0, prop, true);

prop.prop = PROP_USB_SRC_SET_FRAME_MS;
prop.val  = 1;

streamer_set_property(handle->streamer, 0, prop, true);
```

Some of the predefined values can be found in the streamer_api.h.

States The application can be in 2 different states:

- Idle
- Running

Commands in detail

- *help, version*
- *usb_speaker <seconds>*

Legend for diagrams:

```
flowchart TD
    classDef function fill:#69CA00
    classDef condition fill:#0EAFE0
    classDef state fill:#F9B500
    classDef error fill:#F54D4D

    A((State)):::state
    B{Condition}:::condition
    C[Error message]:::error
    D[Process function]:::function
```

help, version

```
flowchart TD
    classDef function fill:#69CA00
    classDef condition fill:#0EAFE0
    classDef state fill:#F9B500
    classDef error fill:#F54D4D

    A((Idle)):::state --> C[Write help or version]:::function
    B((Running)):::state --> C
```

```
C --> E((No state
change)):::state
```

usb_speaker <seconds>

```
flowchart TD
    classDef function fill:#c6d22c
    classDef condition fill:#7cb2de
    classDef state fill:#fcb415
    classDef error fill:#FF999C

    B((Idle)):::state -->C{Duration
    == 0?}:::condition
    C -- No --> E{Duration
    < 0?}:::condition
    C -- Yes --> D[Error: Incorrect
    command parameter]:::error
    D -->B
    E -- Yes --> G[playing]:::function
    G --> H((Running)):::state
    H --> H
    E -- No --> F[playing]:::function
    F --> I((Running)):::state
    I --> J{Duration
    expired?}:::condition
    J -- No -->I
    J -- Yes --> B
```

Supported features The current version of the audio framework supports several optional features. These can be limited to some MCU cores or development boards variants. More information about support can be found on the specific example page:

- *maestro_playback*
- *maestro_record*
- *maestro_usb_mic*
- *maestro_usb_speaker*
- *maestro_sync*

Some features are delivered as prebuilt library and the binaries can be found in the \middleware\audio_voice\components*component*\libs folder. The source code of some features can be found in the \middleware\audio_voice\maestro\src folder.

Decoders Supported decoders and its options are:

Decoder	Sample rates [kHz]	Number of channels	Bit depth
AAC	8, 11.025, 12, 16, 22.05, 24, 32, 44.1, 48	1, 2 (mono/stereo)	16
FLAC	8, 11.025, 12, 16, 22.05, 32, 44.1, 48	1, 2 (mono/stereo)	16
MP3	8, 11.025, 12, 16, 22.05, 24, 32, 44.1, 48	1, 2 (mono/stereo)	16
OPUS	8, 16, 24, 48	1, 2 (mono/stereo)	16
WAV	8, 11.025, 16, 22.05, 32, 44.1, 48	1, 2 (mono/stereo)	8, 16, 24

For more details about the reference decoders please see audio-voice-components repository documentation \middleware\audio_voice\components\.

Encoders

- **OPUS encoder** - The current verion of the audio framework only supports a OPUS encoder. For more details about the encoder please see the following link.

Sample rate converters

- **SSRC** - Synchronous sample rate converter. More details about SSRC are available in the User Guide, which is located in middleware\audio_voice\components\ssrc\doc\.
- **ASRC** - Asynchronous sample rate converter is not used in our examples, but it is part of the maestro middleware and can be enabled. To enable ASRC, the maestro_framework_asrc and CMSIS_DSP_Library_Source components must be added to the project. Furthermore, it is necessary to switch from Redlib to Newlib (semihost) library and add a platform definition to the project (e.g. for RT1170: PLATFORM_RT1170_CORTEXM7). Supported platforms can be found in the PL_platformTypes.h file. More details about ASRC are available in the User Guide, which is located in middleware\audio_voice\components\asrc\doc\.

Additional libraries

- **VIT** - Voice Intelligent Technology (VIT) Wake Word and Voice Command Engines provide free, ready to use voice UI enablement for developers. It enables customer-defined wake words and commands using free online tools. More details about VIT are available in the VIT package, which is located in middleware\audio_voice\components\vit\{platform}\Doc\(depending on the platform) or via following link.
- **VoiceSeeker** - VoiceSeeker is a multi-microphone voice control audio front-end signal processing solution. More details about VoiceSeeker are available in the VoiceSeeker package, which is located in middleware\audio_voice\components\voice_seeker\{platform}\Doc\ (depending on the platform) or via following link.

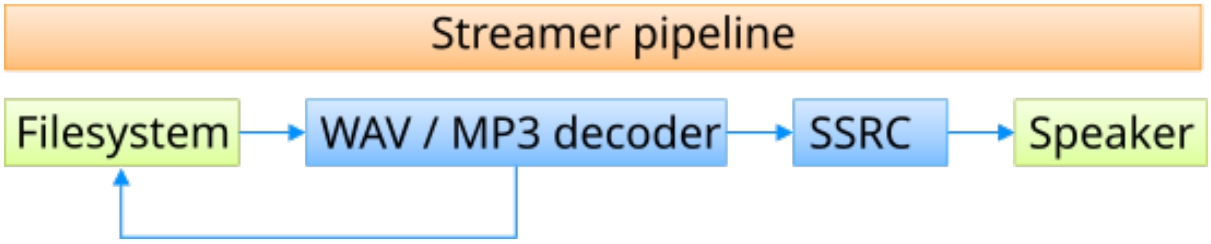
Processing Time

Table of content

- *Maestro playback example*
- *Maestro record example*

The individual time measurements were conducted using a logic analyzer by monitoring changes in the GPIO port levels on the EVKC-MIMXRT1060 development board. These measurements were executed for each individual pipeline run, capturing the timing at each corresponding element, and, when relevant, the interconnections between these elements.

Maestro playback example For the Maestro playback example the following reference audio file was used: test_48khz_16bit_2ch.wav. In this example, the pipeline depicted in the diagram was considered. Media codecs WAV and MP3 were taken into account. To compare the times spent on the SSRC block, sampling rates for both codecs were selected: 44.1 kHz and 48 kHz.



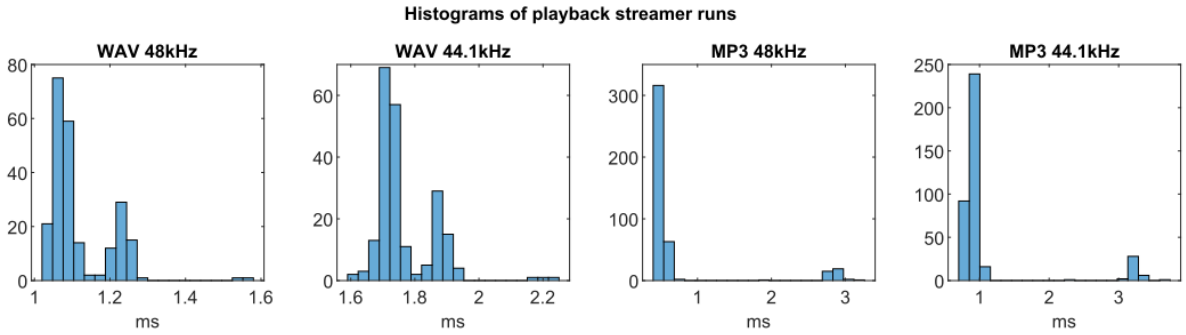
The measurement of streamer pipeline run started at the beginning of streamer_process_pipelines(): streamer.c and ended in the function streamer_pcm_write(): streamer_pcm.c just before the output buffer.

In the scenario involving the WAV codec, the audio file was accessed in every iteration of the streamer pipeline. Meaning, during each run, the file was read directly from the SD card. However, in the case of the MP3 codec, where data processing necessitates complete MP3 frames, the file wasn't read during every run. Rather, it was accessed periodically, triggered when the codec buffer lacked a complete MP3 frame of data. The total time spent on codec processing varies significantly depending on the type and implementation of the codec. For certain types of codecs, like FLAC, there may be multiple file accesses during a single pipeline run. The provided values are specific to the reference implementation. For details about the codecs please see see audio-voice-components documentation middleware\audio_voice\components\.

The duration of the streamer pipeline illustrates that with a sampling frequency of 48 kHz, there is no resampling occurring at the SSRC element. Consequently, the overall pipeline time is lower than in the case of 44.1 kHz audio, where resampling takes place.

To enhance comprehension of the system's behavior, histograms of the pipeline run times and its elements are included. The greater time variance with the MP3 codec is precisely due to the absence of file reads in every run. In clusters with shorter times, there are no file accesses, while in clusters with longer times, file reads occur. This indicates that the majority of runs do not involve file access.

	WAV 48 kHz	WAV 44 kHz	MP3 48 kHz file read	MP3 48 kHz w/o file read	MP3 44 kHz file read	MP3 44 kHz w/o file read
mean	1.11 ms	1.76 ms	2.87 ms	0.51 ms	3.22 ms	0.89 ms
min	1.03 ms	1.60 ms	2.74 ms	0.41 ms	2.33 ms	0.74 ms
max	1.29 ms	2.23 ms	3.24 ms	1.83 ms	3.73 ms	1.12 ms



Time on each element In the tables and histograms below, the timings for individual elements and their connections are provided. Given that the file reading function was invoked during the codec's operation, the tables for individual elements display the total time on the codec element, the time on the codec element before the file read, and the time on the codec element after the file read. The individual blocks in the tables are as follows:

- **streamer** - total time of one pipeline run without time on output buffers
- **codec start** - time on decoder before file read
- **codec end** - time on decoder after file read
- **codec total** - codec_start+codec_end
- **file_src** - file reading time
- **SSRC_proc** - time on SSRC element
- **audio_sink** - time on audio sink without ouput buffers

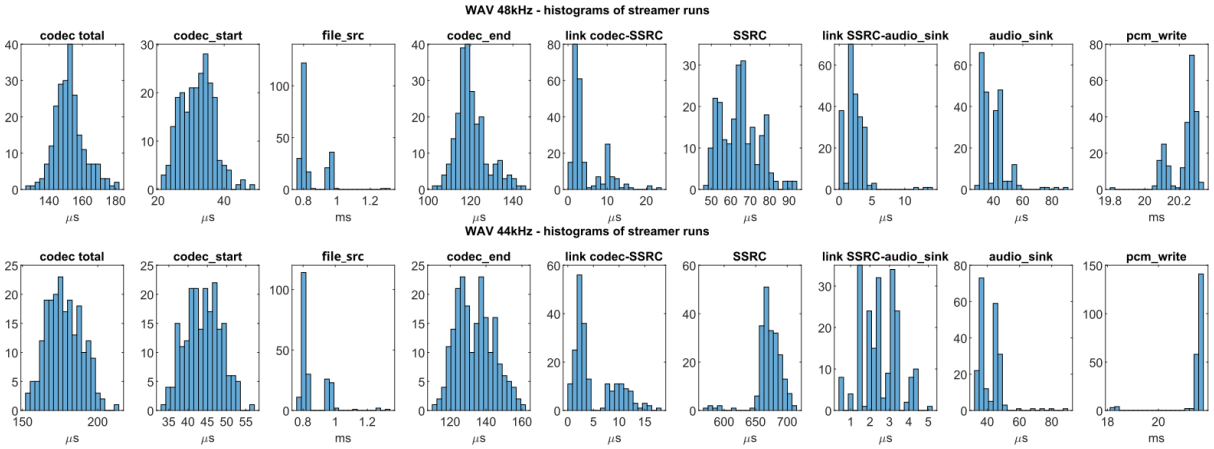
- **pcm_write** - time on output buffers
- **link** - time on element links

The start times of the time intervals for individual blocks and their respective links were measured by altering the GPIO pin level in the following functions:

- **streamer** - streamer_process_pipelines():streamer.c
- **codec** - decoder_sink_pad_process_handler():decoder_pads.c
- **file_src** - filesrc_read():file_src_rtos.c
- **SSRC_proc** - SSRC_Proc_Execute():ssrc_proc.c
- **audio_sink** - audiosink_sink_pad_chain_handler():audio_sink.c
- **pcm_write** - streamer_pcm_write():streamer_pcm.c
- **link** - pad_push():pad.c

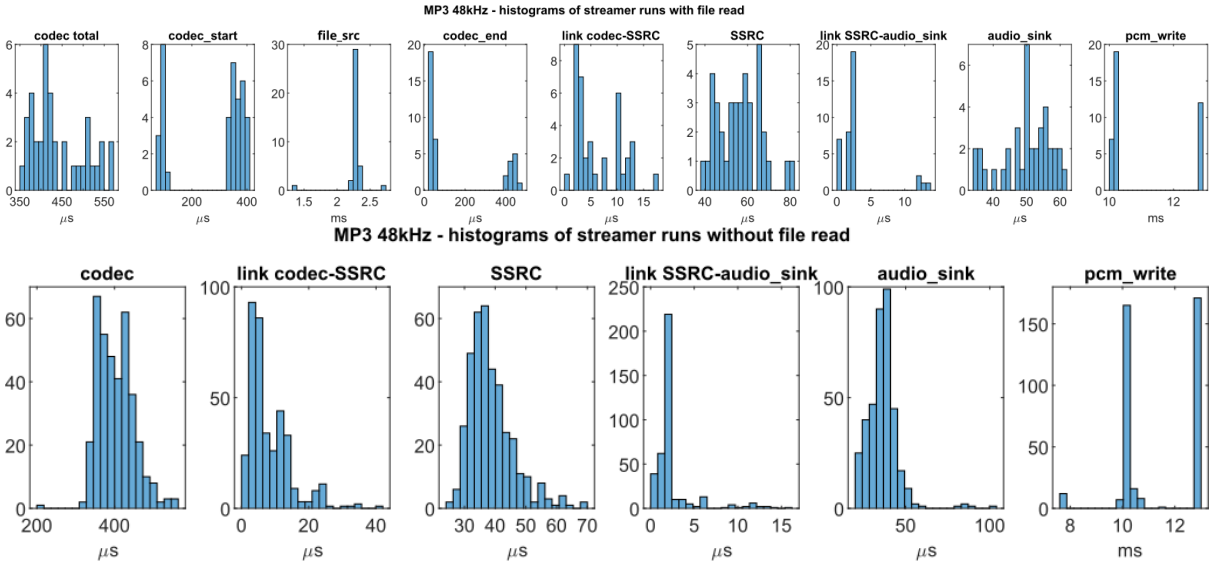
WAV 48kHz	strean	codec total	codec start	file_sr	codec end	link codec-SSRC	SSRC_p	link SSRC-audio_sink	au-dio_sin	pcm_write
mean	1.119 ms	152 µs	31 µs	0.843 ms	120 µs	5 µs	64 µs	2 µs	40 µs	20.228 ms
min	1.026 ms	125 µs	21 µs	0.773 ms	104 µs	<1 µs	47 µs	<1 µs	30 µs	19.805 ms
max	1.290 ms	193 µs	49 µs	1.311 ms	144 µs	23 µs	93 µs	14 µs	91 µs	20.324 ms

WAV 44kHz	strean	codec total	codec start	file_sr	codec end	link codec-SSRC	SSRC_p	link SSRC-audio_sink	au-dio_sin	pcm_write
mean	1.765 ms	178 µs	44 µs	0.853 ms	134 µs	5 µs	671 µs	3 µs	42 µs	21.472 ms
min	1.604 ms	145 µs	33 µs	0.770 ms	112 µs	<1 µs	574 µs	<1 µs	33 µs	18.163 ms
max	2.233 ms	218 µs	57 µs	1.335 ms	161 µs	18 µs	715 µs	5 µs	89 µs	21.746 ms



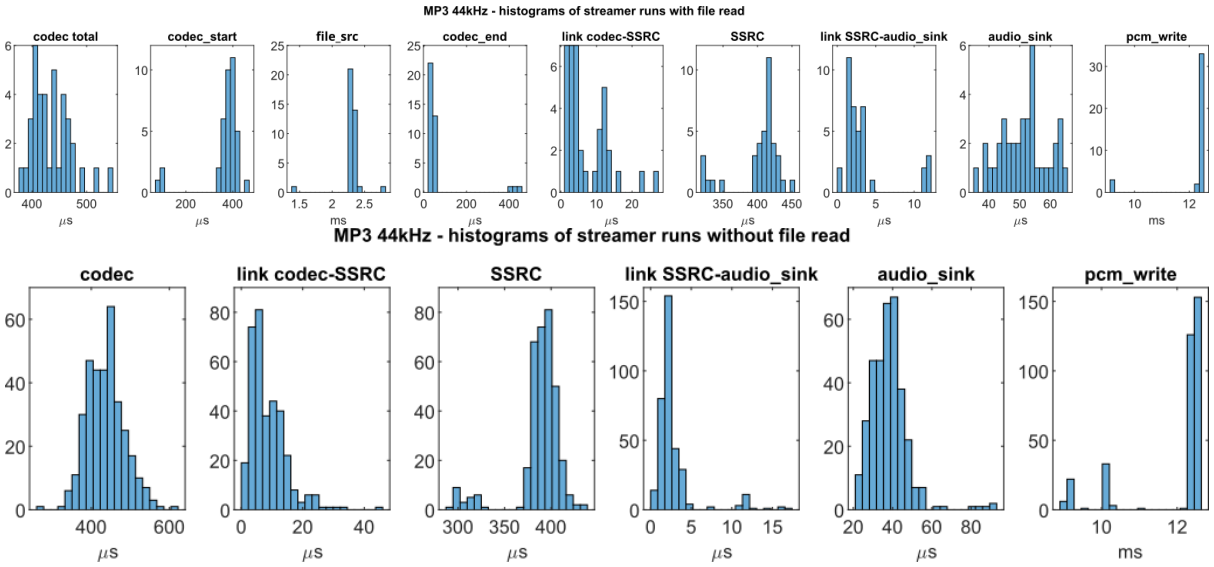
MP3 48 kHz w/ file read	strear	codec total	codec start	file_sr	codec end	link codec-SSRC	SSRC_l	link SSRC-audio_sink	au-dio_sir	pcm_write
mean	2.871 ms	441 µs	279 µs	2.271 ms	162 µs	6 µs	56 µs	3 µs	50 µs	11.019 ms
min	2.739 ms	353 µs	74 µs	1.353 ms	26 µs	<1 µs	40 µs	<1 µs	34 µs	10.091 ms
max	3.244 ms	570 µs	409 µs	2.728 ms	467 µs	18 µs	80 µs	14 µs	62 µs	12.910 ms

MP3 48 kHz w/o file read	strear	codec total	codec start	file_s	codec end	link codec-SSRC	SSRC_l	link SSRC-audio_sink	au-dio_sir	pcm_write
mean	0.508 ms	403 µs	x	x	x	8 µs	39 µs	3 µs	36 µs	11.326 ms
min	0.407 ms	208 µs	x	x	x	<1 µs	25 µs	<1 µs	21 µs	7.715 ms
max	1.834 ms	563 µs	x	x	x	41 µs	69 µs	16 µs	104 µs	12.941 ms

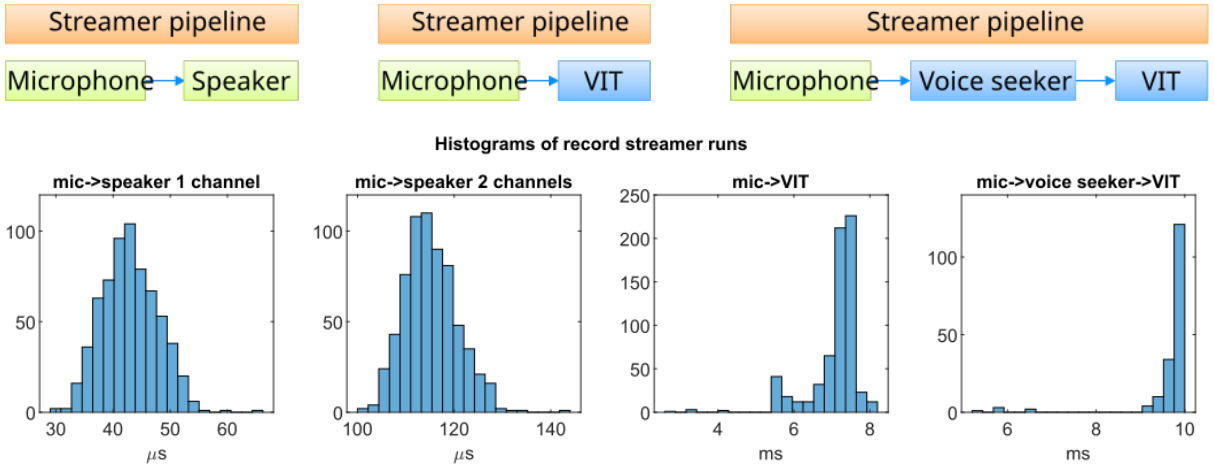


MP3 44 kHz w/ file read	strear	codec total	codec start	file_sr	codec end	link codec-SSRC	SSRC_l	link SSRC-audio_sink	au-dio_sir	pcm_write
mean	3.217 ms	436 µs	367 µs	2.300 ms	66 µs	7 µs	403 µs	3 µs	51 µs	12.188 ms
min	2.329 ms	383 µs	73 µs	1.411 ms	26 µs	2 µs	318 µs	<1 µs	35 µs	9.119 ms
max	3.726 ms	547 µs	464 µs	2.801 ms	441 µs	27 µs	454 µs	12 µs	65 µs	12.529 ms

MP3 44 kHz w/o file read	strear	codec total	codec start	file_s	codec end	link codec-SSRC	SSRC_l	link SSRC-audio_sink	au-dio_sir	pcm_write
mean	0.891 ms	437 µs	x	x	x	9 µs	388 µs	3 µs	38 µs	11.934 ms
min	0.738 ms	268 µs	x	x	x	<1 µs	290 µs	<1 µs	22 µs	8.964 ms
max	1.115 ms	620 µs	x	x	x	45 µs	438 µs	17 µs	92 µs	12.624 ms



Maestro record example Typical execution times of the streamer pipeline and its individual elements for the EVKC-MIMXRT1060 development board are detailed in the following tables. The duration spent on output buffers and reading from the microphone is excluded from traversal measurements. Three measured pipelines are depicted in the figure below. The first involves a loopback from microphone to speaker, supporting both mono and stereo configurations. The second pipeline is a mono voice control setup, comprising microphone and VIT blocks. The final pipeline is a stereo voice control setup, integrating microphone, voice seeker, and VIT blocks. The measurement of streamer pipeline run started at the beginning of streamer_process_pipelines():streamer.c and ended in the function streamer_pcm_write(): streamer_pcm.c just before the output buffer.



The individual blocks in the tables are as follows:

- **streamer** - total time of one pipeline run without time on output buffers and without time reading from the microphone
- **audio_src_start** - time on audio src before reading from the microphone
- **audio_src_end** - time on audio src after reading from the microphone
- **pcm_read** - reading from the microphone
- **voiceseeker** - time on voice seeker element
- **vit** - time on VIT element
- **audio_sink** - time on audio sink without ouput buffers
- **pcm_write** - time on output buffers
- **link** - time on element links

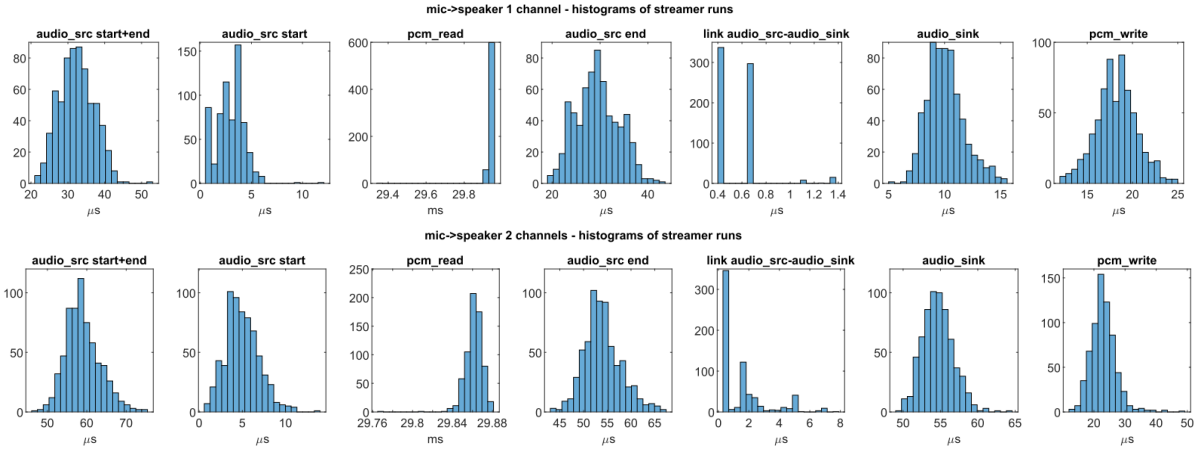
The start times of the time intervals for individual blocks and their respective links were measured by altering the GPIO pin level in the following functions:

- **streamer** - streamer_process_pipelines():streamer.c
- **audio_src** - audiosrc_src_process():audio_src.c
- **pcm_read** - streamer_pcm_read():streamer_pcm.c
- **voiceseeker** - audio_proc_sink_pad_chain_handler():audio_proc.c
- **vit** - vitsink_sink_pad_chain_handler():vit_sink.c
- **audio_sink** - audiosink_sink_pad_chain_handler():audio_sink.c
- **pcm_write** - streamer_pcm_write():streamer_pcm.c
- **link** - pad_push():pad.c

Pipeline Microphone -> Speaker

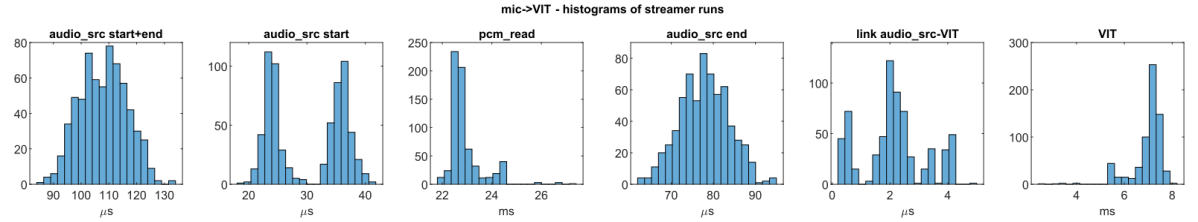
microphone -> speaker mono	stream	audio_src_sta	pcm_re	audio_src_er	link audio_src-audio_sink	audio_sink	pcm_write
mean	43 µs	3 µs	29.938 ms	29 µs	<1 µs	10 µs	18 µs
min	26 µs	<1 µs	29.350 ms	19 µs	<1 µs	5 µs	12 µs
max	72 µs	12 µs	29.957 ms	44 µs	1 µs	15 µs	25 µs

microphone -> speaker stereo	stream	audio_src_sta	pcm_re	audio_src_er	link audio_src-audio_sink	audio_sink	pcm_write
mean	115 µs	5 µs	29.861 ms	54 µs	2 µs	55 µs	23 µs
min	94 µs	<1 µs	29.768 ms	43 µs	<1 µs	50 µs	12 µs
max	154 µs	14 µs	29.880 ms	67 µs	8 µs	65 µs	49 µs



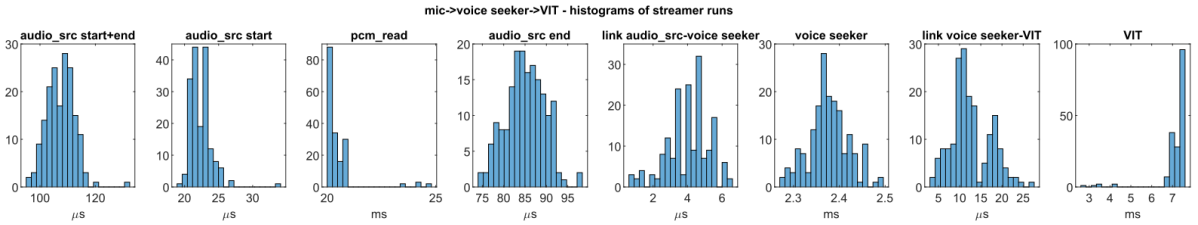
Pipeline Microphone -> VIT

microphone -> VIT	streamer	audio_src_start	pcm_read	audio_src_end	link audio_src-vit	vit
mean	7.380 ms	30 µs	22.624 ms	78 µs	2 µs	7.261 ms
min	2.641 ms	10 µs	2.2265 ms	58 µs	<1 µs	2.559 ms
max	7.780 ms	42 µs	2.7341 ms	94 µs	5 µs	7.624 ms



Pipeline Microphone -> Voice seeker -> VIT

microphone -> voice seeker -> VIT	stream	audio_src_s	pcm_r	audio_src_(	link audio_src-voiceseeker	voice-seeker	link voiceseeke vit	vit
mean	9.916 ms	22 µs	20.084 ms	84 µs	4 µs	2.386 ms	13 µs	7.407 ms
min	4.983 ms	19 µs	19.738 ms	72 µs	<1 µs	2.228 ms	2 µs	2.662 ms
max	10.423 ms	34 µs	24.777 ms	100 µs	7 µs	2.522 ms	31 µs	7.729 ms



Maestro on Zephyr

- Based on and tested with Zephyr version, given by tag v4.0.0
- Tested with Zephyr SDK version 16.4
- To see the pre-built documentation, see: README.html. Also see the *documentation section*.

Maestro sample for recording data from microphone to RAM

Description This sample records data from microphone (alias dmic0 in devicetree) and stores them to a buffer in RAM.

Currently one PDM channel with fixed 16 kHz sample rate and 16 bit sample width is supported.

For configuration options, see Kconfig and prj.conf.

User Input/Output

- Input:

  None.

- Output:

  UART Output:

  - Demo result: OK if everything went OK
  - Demo result: FAIL otherwise

Supported platforms Currently tested for:

- RD_RW612_BGA.

Maestro voice detection sample using VIT

Description Records data from microphone (alias dmic0 in devicetree) and detects voice commands from selected language model. Detected commands are printed via UART.

Language model may be changed via Kconfig using CONFIG_MAESTRO_EXAMPLE_VIT_LANGUAGE selection. For other configuration options, see example's Kconfig and prj.conf.

This project requires an NXP board supported by the VIT library.

The example has to be modified if a new board needs to be added. Please create an issue in that case.

User Input/Output

- Input:

  None.

- Output:

  UART Output:

  - List of voice commands the model can detect (printed immediately after start)
  - <Specific voice command> if voice command was detected
  - Demo result: FAIL otherwise

Dependencies

- VIT library: https://www.nxp.com/design/design-center/software/embedded-software/voice-intelligent-technology-wake-word-and-voice-command-engines:VOICE-INTELLIGENT-TECHNOLOGY

Supported platforms Currently tested for:

- RD_RW612_BGA.

Maestro decoder sample

Description Tests and demonstrates decoder functionality in Maestro pipeline.

Supported decoders:

- MP3
- WAV
- AAC
- FLAC
- OPUS with OGG envelop
- (RAW OPUS - TBD)

Data Input:

- Prepared encoded audio data (part of Maestro repository, folder `zephyr/audioTracks`)
- Prepared decoded audio data (RAW PCM format, part of Maestro repository, folder `zephyr/audioTracks`)

Function:

1. Loads encoded data into source buffer stored in RAM
2. Decodes audio data using selected decoder and stores data in RAM
3. Compares prepared data with decoded data to check if its the same
4. Prints `Demo result: OK` or `Demo result: FAIL` via UART

User Input/Output

- Input:

  None

- Output:

  UART Output

  - `Demo result: OK` if everything went OK
  - `Demo result: FAIL` otherwise

Dependencies

- Audio voice component library (pulled in by Maestro's west), containing Decoder libraries

Configuration

- See prj.conf for user input sections
  - Selecting decoder may be done by enabling CONFIG_MAESTRO_EXAMPLE_DECODER_SELECTED in prj.conf file. When no decoder is selected, default one (WAV) is used instead.
  - System settings should be modified (stack size, heap size) based on selected decoder and system capabilities/requirements in prj.conf.
- For other configuration options, see example's Kconfig and prj.conf.

Supported platforms Currently tested for:

- RD_RW612_BGA - Working decoders: FLAC, WAV, OPUS OGG

Maestro encoder sample

Description Tests and demonstrates encoder functionality in Maestro pipeline.

Supported encoders:

- OPUS with OGG envelop - TBD
- RAW OPUS - TBD

Input:

- Prepared decoded audio data (RAW PCM format, part of Maestro repository)
- Prepared encoded audio data (part of Maestro repository)

Function:

1. Loads RAW data into source buffer stored in RAM
2. Encodes audio data using selected encoder and stores data in RAM
3. Compares prepared data with decoded data if same
4. Prints Demo result: OK or Demo result: FAIL via UART

Dependencies

- Audio voice component library (pulled in by Maestro's west), containing Encoder libraries

User Input/Output Input:

- None

Output:

- UART Output
  - Demo result: OK if everything went OK
  - Demo result: FAIL otherwise

Configuration

- See prj.conf for user input sections
  - Selecting encoder may be done by enabling CONFIG_MAESTRO_EXAMPLE_ENCODER_SELECTED in prj.conf file. When no encoder is selected, default one (OPUS) is used instead.
  - System settings should be modified (stack size, heap size) based on selected encoder and system capabilities/requirements in prj.conf file.
- For other configuration options, see example's Kconfig and prj.conf.

Supported platforms Currently tested for:

- RD_RW612_BGA - Working encoders: None.

Maestro mem2mem sample

Description Tests basic memory to memory pipeline.

Function:

1. Moves generated data with fixed size of 256B from memory source to memory sink.
2. Compares copied data to check if they're the same.
3. Returns Demo result: OK or Demo result: FAIL via UART.

- *Maestro environment setup*
- *Build and run Maestro example*
  - *Using command line*
  - *Using MCUXpresso for VS Code*
- *Folder structure*
- *Supported elements and libraries*
- *Examples support*
- *Creating your own example*
- *Documentation*
- *FAQ*

Maestro environment setup Follow these steps to set up a Maestro development environment on your machine.

1. If you haven't already, please follow this guide to set up a Zephyr development environment and its dependencies first:
   - Cmake
   - Python
   - Devicetree compiler
   - West
   - Zephyr SDK bundle

2. Get Maestro. You can pick either of the options listed below. If you need help deciding which option is the best fit for your needs, please see the *FAQ*.

   - Freestanding Maestro - This option pulls in only Maestro's necessary dependencies.

     Run:

     ```
     1. west init -m <maestro repository url> --mr <revision> --mf west-freestanding.yml
     ↪<foldername>
     2. cd <foldername>
     3. west update
     ```

   - Maestro as a Zephyr module

     To include Maestro into Zephyr, update Zephyr's west.yml file:

     ```
     projects:
     name: maestro
     url: <maestro repository url>
     revision: <revision with Zephyr support>
     path: modules/audio/maestro
     import: west.yml
     ```

     Then run west update maestro command.

Build and run Maestro example These steps will guide you through building and running Maestro samples. You can use either the command line utilizing Zephyr's powerful west tool or you can use VS Code's GUI. Detailed steps for both options are listed below.

Using command line See Zephyr's Building, Flashing and Debugging guide if you aren't familiar with it yet.

1. To **build** a project, run:

   ```
   west build -b <board> -d <output build directory> <path to example> -p
   ```

   For example, this compiles VIT example for rd_rw612_bga board:

   ```
   1. cd maestro/zephyr
   2. west build -b rd_rw612_bga -d build samples/vit -p
   ```

2. To **run** a project, run:

   ```
   west flash -d <directory>
   ```

   e.g.:

   ```
   west flash -d build
   ```

3. To **debug** a project, run:

   ```
   west debug -d <directory>
   ```

   e.g.:

   ```
   west debug -d build
   ```

Using MCUXpresso for VS Code For this you have to have NXP's MCUXpresso for VS Code extension installed.

1. Import your topdir as a repository to MCUXPresso for VS Code:

- Open the MCUXpresso Extension. In the *Quickstart Panel* click *Import Repository*.
- In the displayed menu click *LOCAL* tab and select the folder location of your *topdir*.
- Click *Import*.
- The repository is successfully added to the *Installed Repositories* view once the import is successful.

2. To import any project from the imported repository:
   - In the *Quickstart Panel* click *Import Example from Repository*.
   - For **Repository** select *your imported* repository.
   - For **Zephyr SDK** the installed Zephyr SDK is selected automatically. If not, select one.
   - For **Board** select your board (*make sure* you've selected the *correct* revision).
   - For **Template** select the folder path to your project.
   - Click the *Create* button.
3. Build the project by clicking the *Build Selected* icon (displayed on hover) in the extension's *Projects* view. After the build, the debug console window displays the memory usage (or compiler errors if any).
4. Debug the project by clicking the *Debug* (play) icon (displayed on hover) in the extension's *Projects* view.
5. The execution will pause. To continue execution click *Continue* on the debug options.
6. In the *SERIAL MONITOR* tab of your console panel, the application prints the Zephyr boot banner during startup and then prints the test results.

Folder structure

```
maestro/
    ...
    zephyr/              All Zephyr related files
      samples/           Sample examples
      tests/             Tests
      audioTracks/       Audio tracks for testing
      doc/               Documentation configuration for Sphinx
      wrappers/          NXP SDK Wrappers
      scripts/           Helper scripts, mostly for testing
      module.yml         Defines module name, Cmake and Kconfig locations
      CMakeList.txt      Defines module's build process
      Kconfig            Defines module's configuration
      osa/               Deprecated. OSA port for Zephyr
      ...
```

Supported elements and libraries Here is the list of all features currently supported in Maestro on Zephyr. Our goal is to support all features in Maestro on Zephyr that are already supported in Maestro on NXP's SDK and to extend them further.

Supported elements:

- Memory source
- Memory sink
- Audio source
- Audio sink
- Process sink

- Decoder
- Encoder

Supported decoders:

- WAV
- MP3
- FLAC
- OPUS OGG
- AAC

Supported encoders:

- OPUS RAW

Supported libraries:

- VIT

Examples support All included examples use UART as output. Examples are located in zephyr/tests and zephyr/samples directories.

List of included examples:

- *Maestro sample for recording data from microphone to RAM*
- *Maestro voice detection sample using VIT*
- *Maestro encoder sample*
- *Maestro decoder sample*
- *Maestro mem2mem sample*

Examples support for specific boards:

Example	RDRW612BGA	LPCx-presso55s69	MIMXRT1060EVKE	MIMXRT1170EVKB
Record	YES	TO BE TESTED	TO BE TESTED	TO BE TESTED
VIT	YES	TO BE TESTED	TO BE TESTED	TO BE TESTED
Encoder	In progress: OPUS RAW	TO BE TESTED	TO BE TESTED	TO BE TESTED
Decoder	YES - WAV, FLAC, OPUS OGG	TO BE TESTED	TO BE TESTED	TO BE TESTED
Mem2mem	YES	TO BE TESTED	TO BE TESTED	TO BE TESTED

Creating your own example There are two ways to create your own example - you can either one of the included examples as a reference or you can create your own example from scratch by hand.

When creating your own example from scratch, set CONFIG_MAESTRO_AUDIO_FRAMEWORK=y in your prj.conf file. Then you can start enabling specific elements by setting CONFIG_MAESTRO_ELEMENT_<NAME>_ENABLE=y.

However, the recommended way to edit config options is to open gui-config (or menuconfig) by calling west build -t guiconfig. Then you can use the graphical interface to interactively turn on/off the features you need.

Documentation Please note, Maestro documentation is under reconstruction. It is currently mixing several tools and formats.

To see the pre-generated Maestro Zephyr documentation, see zephyr/doc/doc/README.html

To generate the Zephyr documentation, go under zephyr/doc folder and execute make html. Sphinx version sphinx-build 8.1.3 must be installed. Open doc/doc/html/README.hml afterwards.

To see Maestro core documentation, go to the Maestro top directory and see README.md.

FAQ

1. Should I choose the freestanding version of Maestro or should integrate it into my west instead?
    - Freestanding version of Maestro pulls in all the dependencies it needs including Zephyr itself.
    - Integrating it as a module is easier if you already have your Zephyr environment set up.

Maestro Audio Framework changelog

2.0.0 (newest)

- Added Zephyr port, see *Zephyr README*.
    - Possible to use standalone version, pulling its own Zephyr and dependencies
    - Possible to import it as a module in your Zephyr project
- Changed build system - newly uses Kconfig and Cmake
- Supports NXP MCUXSDK (previously 2.x)
- Changed folder structure and names to improve readability (description may be found in *README*)
- Removed audio libraries and placed into audio-voice-components repository
- Added libraries are pulled into the build via Kconfig and Cmake
- Changed Maestro library core - minor changes

1.8.0

- New platforms support: MCX-N5XX-EVK, FRDMMCXN236 and RD-RW612-BGA
- Fixed compilation warnings
- Documentation improvements and updates
    - Added section with processing time information
    - Added application state diagrams
- Various updates and fixes

1.7.0

- Removed EAP support for future SDK releases
- Created new API for audio_sink and audio_src to support USB source, sink
- ASRC library integrated
- License changed to BSD 3-Clause
- Improved pipeline creation API
- Fixed compilation warnings in Opus
- Various other improvements and bug fixes

1.6.0

- Up to 2 parallel pipelines supported
- Synchronous Sample Rate Converter support Added
- Various improvements and bug fixes

1.5.0

- Enabled switching from 2 to 4 channel output during processing
- PadReturn type has been replaced by FlowReturn
- Support of AAC, WAV, FLAC decoders
- Renamed eap element to audio_proc element
- Added audio_proc to VIT pipeline to support VoiceSeeker
- Minor bug fixes

1.4.0

- Use Opusfile lib for Ogg Opus decoder
- Refactor code, fix issues found in unit tests
- Various bug fixes

1.3.0

- Make Maestro framework open source (except mp3 and wav decoder)
- Refactor code, remove unused parts, add comments

1.2.0

- Unified buffering in audio source, audio sink
- Various improvements and bug fixes

1.0_rev0

- Initial version of framework with support for Cortex-M7 platforms

2.9 Wireless

2.9.1 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 25.06.00 has been tested for the wireless SoCs listed in Supported products.

Supported products

- 88W8987
- IW416
- IW611[1]
- IW612[2]
- AW611[3]
- RW610
- RW612

Parent topic:*About this document*

[1] The support of IW611 is enabled in i.MX RT1170 EVKB and i.MX RT1060 EVKC.
[2] The support of IW612 is enabled in i.MX RT1170 EVKB and i.MX RT1060 EVKC.
[3] AW611 module support is available only in i.MX RT1180 EVKA

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)
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

Table 1 – continued from p

Features	Sub features
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 Methods | WPA Supplicant
802.11i security	WPA2-Enterprise Mixed Mode | WPA Supplicant
802.11i security	WPA3-Enterprise (Suite-B) |National Security Algorithm (CS
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 Enterprise
Power save mode	Deep sleep
Power save mode	IEEE power save
Power save mode	Host sleep/WoWLAN (inband)
Power save mode	Host sleep/WoWLAN (outband)
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 connect[^3]
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	TKIP[^1]
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
General features	CSI (channel state information)
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)

Table 1 – continued from p

Features	Sub features
General features	802.11az

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)
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

Table 2 – continued from previ

Features	Sub features
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 Mode | Hostapd
802.11i security	WPA3-Enterprise (Suite-B) |National Security Algorithm (CS
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) mode[^2]
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

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 diversity[^1]	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/RW	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 optimizer	Y	Y	Y	Y	Y	Y
Generic	Firmware Compression[^2]	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

Wi-Fi direct/P2P

Features	Sub features	88W8987	IW416[^	IW611/IW61	RW610/RW61	IW610[^	AW611[^3]
P2P basic functionality[^1]	P2P Auto GO	Y	Y	Y	Y	Y	Y
P2P basic functionality[^1]	P2P GO	Y	Y	Y	Y	Y	Y
P2P basic functionality[^1]	P2P GC	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	IW4'	IW611/	RW610/	IW6'	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 support[^1] [^2]	Isochronous channel
Bluetooth LE audio support[^1] [^2]	Broadcast LE Audio BIS source
Bluetooth LE audio support[^1] [^2]	Broadcast LE Audio BIS sink
Bluetooth LE audio support[^1] [^2]	Broadcast LE Audio BIG Validation
Bluetooth LE audio support[^1] [^2]	Broadcast LE Audio Phy: 1M/2M/ coded
Bluetooth LE audio support[^1] [^2]	Broadcast LE Audio framed mode
Bluetooth LE audio support[^1] [^2]	Broadcast LE Audio unframed mode
Bluetooth LE audio support[^1] [^2]	Broadcast LE Audio sequential packing
Bluetooth LE audio support[^1] [^2]	Broadcast LE Audio: Mono and Stereo
Bluetooth LE audio support[^1] [^2]	Broadcast LE Audio BIS encrypted audio
Bluetooth LE audio support[^1] [^2]	Broadcast LE Audio BIS unencrypted audio
Bluetooth LE audio support[^1] [^2]	Unicast LE Audio CIS source
Bluetooth LE audio support[^1] [^2]	Unicast LE Audio CIS sink
Bluetooth LE audio support[^1] [^2]	Unicast LE Audio CIG validation
Bluetooth LE audio support[^1] [^2]	Unicast LE Audio CIS synchronization
Bluetooth LE audio support[^1] [^2]	Unicast LE Audio Phy: 1M/2M/ coded
Bluetooth LE audio support[^1] [^2]	Unicast LE Audio framed mode
Bluetooth LE audio support[^1] [^2]	Unicast LE Audio unframed mode
Bluetooth LE audio support[^1] [^2]	Unicast LE Audio sequential packing
Bluetooth LE audio support[^1] [^2]	Unicast LE Audio: mono and stereo
Bluetooth LE audio support[^1] [^2]	Unicast LE Audio CIS encrypted audio
Bluetooth LE audio support[^1] [^2]	Unicast LE Audio CIS unencrypted audio
Bluetooth LE audio support[^1] [^2]	Unicast LE Audio TX/RX and bidirectional traffic

Table 3 – continued from prev

Features	Sub features
Bluetooth LE audio support[^1] [^2]	ISO interval for LE Audio: 7.5ms 10ms 20ms 30ms
Bluetooth LE audio support[^1] [^2]	Sampling frequency for LE Audio: 8kHz 16kHz 24kHz, 32kH
Bluetooth LE audio support[^1] [^2]	LE Audio Auracast use cases: Auracast streaming 2 BISes
Bluetooth LE audio support[^1] [^2]	LE Audio Unicast use cases: Unicast streaming 2 CISes
Bluetooth LE audio support[^1] [^2]	LE Audio Unicast Use cases: Unicast streaming 4 CISes
Bluetooth LE audio support[^1] [^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 coexistence 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 antenna[^1] (lower and higher isolation) 1x1 Wi-Fi, (Bluetooth and 802.15.4 shared)	STA + Bluetooth	Y	N	N
BCA_TDM separate antenna[^1] (lower and higher isolation) 1x1 Wi-Fi, (Bluetooth and 802.15.4 shared)	Mobile AP + Bluetooth	Y	N	N
BCA_TDM separate antenna[^1] (lower and higher isolation) 1x1 Wi-Fi, (Bluetooth and 802.15.4 shared)	Bluetooth LE + Wi-Fi	Y	Y	Y
BCA_TDM separate antenna[^1] (lower and higher isolation) 1x1 Wi-Fi, (Bluetooth and 802.15.4 shared)	Bluetooth + Bluetooth LE + Wi-Fi	Y	N	N
BCA_TDM separate antenna[^1] (lower and higher isolation) 1x1 Wi-Fi, (Bluetooth and 802.15.4 shared)	OpenThread + Bluetooth	Y	N	N
BCA_TDM separate antenna[^1] (lower and higher isolation) 1x1 Wi-Fi, (Bluetooth and 802.15.4 shared)	OpenThread + Bluetooth LE[^2]	Y	Y	Y
BCA_TDM separate antenna[^1] (lower and higher isolation) 1x1 Wi-Fi, (Bluetooth and 802.15.4 shared)	OpenThread + Bluetooth + Bluetooth LE	Y	N	N
BCA_TDM separate antenna[^1] (lower and higher isolation) 1x1 Wi-Fi, (Bluetooth and 802.15.4 shared)	OpenThread + Wi-Fi	Y	Y	Y
BCA_TDM separate antenna[^1] (lower and higher isolation) 1x1 Wi-Fi, (Bluetooth and 802.15.4 shared)	Bluetooth + OpenThread + Wi-Fi	Y	N	N
BCA_TDM separate antenna[^1] (lower and higher isolation) 1x1 Wi-Fi, (Bluetooth and 802.15.4 shared)	Bluetooth LE + OpenThread + Wi-Fi	Y	Y	Y
BCA_TDM separate antenna[^1] (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 antenna[^1] (lower and higher isolation) 1x1 Wi-Fi, (Bluetooth and 802.15.4 shared)	Single antenna configuration	Y	Y	Y
BCA_TDM separate antenna[^1] (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
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_WMM[^1]	Flash - 10K, RAM - 57K
802.11mc	CONFIG_11MC CONFIG_CSI CONFIG_WLS_CSI_PROC[^2] CONFIG_11AZ	Flash: 52.78KB, RAM : 121.1KB
802.11az	CONFIG_11MC CONFIG_CSI[2] CONFIG_WLS_CSI_PROC[^2] 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.

Kconfig memory optimizer The MCUXpresso SDK provides options to reduce the host memory usage with build-time configuration parameters referred to as Kconfig memory optimizer. The configuration parameters are used to reduce the use of the flash memory and SRAM.

This section explains how to enable the host memory saving configurations within the Wi-Fi drivers of NXP wireless devices.

Memory impact of i.MX RT1060 EVKC + IW416 module (Murata 1XK):

- Maximum Flash usage: 889 KB
- Maximum SRAM usage: 418.77 KB

To reduce the use of the flash and SRAM, change the settings of the Kconfig macros listed in table in the file wifi_config.h located in <path-to-SDK_Wi-Fi_Example> directory.

Kconfig macros	Feature disabled
CONFIG_WIFI_SL	CONFIG_ROAMING CONFIG_11R
CONFIG_WIFI_SL	CONFIG_CLOUD_KEEP_ALIVE CONFIG_WIFI_EU_CRYPTO CONFIG_TX_AMPDU_PROT_MODE CONFIG_WNM_PS CONFIG_TURBO_MODE CONFIG_AUTO_RECONNECT CONFIG_DRIVER_OWE CONFIG_OWE CONFIG_WIFI_FORCE_RTS CONFIG_WIFI_FRAG_THRESHOLD CONFIG_COMBO_SCAN CONFIG_SCAN_CHANNEL_GAP
CONFIG_WIFI_SL	CONFIG_UAP_STA_MAC_ADDR_FILTER CONFIG_WIFI_MAX_CLIENTS_CNT
CONFIG_FREERTC	If the macro is enabled, the heap memory usage is reduced by 10 KB (from 70 KB to 60 KB).
CONFIG_LWIP_LC	Curtails LWIP stack parameters, reduces data throughput, disables data net-stats
Non-blocking firmware download mechanism	CONFIG_FW_DNLD_ASYNC

Parent topic:*Feature enable and memory impact*

[^1] The macro is not used for IW416.

[^2] Prerequisite macros for 802.11mc and 802.11az features

88W8987 release notes

Package information

- SDK version: 25.06.00

Parent topic:*88W8987 release notes*

Version information

- Wireless SoC: 88W8987
- Wi-Fi and Bluetooth/Bluetooth LE firmware version: 16.92.21.p151.7
  - 16 - Major revision
  - 91 - Feature pack
  - 21 - Release version
  - p151.7 - 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 | WPA3 SAE (R3)
- 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
OpenSecurity	46	51	60	60
WPA2-AES	45	42	60	54
WPA3-SAE	46	41	60	54

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
OpenSecurity	78	99	125	203
WPA2-AES	78	98	126	197
WPA3-SAE	82	98	125	197

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
OpenSecurity	43	51	56	62
WPA2-AES	42	50	54	61
WPA3-SAE	40	50	65	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
OpenSecurity	93	114	125	206
WPA2-AES	92	111	125	191
WPA3-SAE	92	111	125	173

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 {#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 {#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 {#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

Parent topic:*88W8987 release notes*

Known issues

Component	Description
-	NA

Parent topic:*88W8987 release notes*

IW416 release notes

Package information

- SDK version: 25.06.00

Parent topic:*IW416 release notes*

Version information

- Wireless SoC: IW416
- Wi-Fi and Bluetooth/Bluetooth LE firmware version: 16.92.21.p151.7
  - 16 - Major revision

  - 92 - Feature pack
  - 21 - Release version
  - p151.7 - 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 | WPA3 SAE (R3)
- 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
OpenSecurity	39	38	59	58
WPA2-AES	38	36	57	58
WPA3-SAE	41	35	57	57

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	58	58	93	91
WPA2-AES	56	49	94	74
WPA3-SAE	54	57	94	73

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
OpenSecurity	40	45	61	58
WPA2-AES	40	43	61	57
WPA3-SAE	40	44	61	57

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
OpenSecurity	60	66	94	100
WPA2-AES	58	61	94	98
WPA3-SAE	59	61	94	98

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
OpenSecurity	40	43	59	57
WPA2-AES	40	42	59	57
WPA3-SAE	39	42	59	57

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	62	74	121	118
WPA2-AES	60	64	116	91
WPA3-SAE	60	65	116	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
OpenSecurity	50	42	45	62
WPA2-AES	42	45	53	62
WPA3-SAE	42	62	53	45

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	66	76	126	96
WPA2-AES	63	68	121	95
WPA3-SAE	63	67	121	95

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 {#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 {#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 {#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 {#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

Parent topic:*IW416 release notes*

Known issues

Component	Description
Coex	Wi-Fi connection in 2.4GHz is not stable, observed deauthentication within 10sec.

Parent topic:*IW416 release notes*

IW611/IW612 release notes **Note:** The IW611/IW612 support is enabled in i.MX RT1170 EVKB and i.MX RT1060 EVKC.

Package information

- SDK version: 25.06.00

Parent topic:*IW611/IW612 release notes*

Version information

- Wireless SoC: IW611/IW612
- Wi-Fi and Bluetooth/Bluetooth LE firmware version: 18.99.3.p25.11
  - 18 - Major revision
  - 99 - Feature pack
  - 3 - Release version
  - p25.11 - Patch number

Parent topic:*IW611/IW612 release notes*

Host platform

- i.MX RT1170 EVKB and i.MX RT1060 EVKC Platforms running FreeRTOS
- 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 | WPA3 SAE (R3)
- 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 {#iperf_host_configuration_and_impact_on_throughpu
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_and_ap_throughput_captured_with_the_minimum_values_of_the_host_configuration_macr

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
OpenSecurity	43	47	45	60
WPA2-AES	43	46	48	59
WPA3-SAE	47	49	63	63

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	68	82	131	131
WPA2-AES	72	82	130	129
WPA3-SAE	68	81	129	130

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
OpenSecurity	45	51	63	65
WPA2-AES	45	51	63	65
WPA3-SAE	45	51	63	65

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
OpenSecurity	67	83	129	134
WPA2-AES	66	83	129	133
WPA3-SAE	65	83	129	133

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	49	53	72	71
WPA2-AES	48	52	73	70
WPA3-SAE	52	56	75	75

STA mode throughput - VHT Mode | 2.4 GHz Band | 40 MHz (HT)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	74	88	172	172
WPA2-AES	75	92	171	169
WPA3-SAE	77	92	172	171

STA mode throughput - VHT 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
OpenSecurity	50	58	76	78
WPA2-AES	49	57	76	77
WPA3-SAE	49	57	76	77

STA mode throughput - VHT 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
OpenSecurity	74	93	175	177
WPA2-AES	74	93	174	174
WPA3-SAE	73	93	173	175

STA mode throughput - VHT Mode | 5 GHz Band | 80 MHz (HT)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	88	94	221	196
WPA2-AES	87	95	219	194
WPA3-SAE	89	95	219	195

STA mode throughput - HE 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	60	60	98	115
WPA2-AES	62	61	94	113
WPA3-SAE	61	59	97	108

STA mode throughput - HE Mode | 2.4 GHz Band | 40 MHz (HT)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	77	71	215	190
WPA2-AES	77	72	212	187
WPA3-SAE	76	72	152	189

STA mode throughput - HE 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
OpenSecurity	63	65	127	128
WPA2-AES	63	67	125	128
WPA3-SAE	63	67	125	126

STA mode throughput - HE 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
OpenSecurity	79	64	212	199
WPA2-AES	78	68	218	199
WPA3-SAE	79	68	217	198

STA mode throughput - HE Mode | 5 GHz Band | 80 MHz (HT)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	88	70	219	192
WPA2-AES	87	72	219	193
WPA3-SAE	91	72	220	194

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
OpenSecurity	40	52	61	62
WPA2-AES	40	51	61	62
WPA3-SAE	40	51	61	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	63	83	116	130
WPA2-AES	67	82	115	131
WPA3-SAE	60	81	115	132

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	39	49	60	62
WPA2-AES	39	49	60	62
WPA3-SAE	41	51	63	62

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	66	89	128	133
WPA2-AES	64	87	128	133
WPA3-SAE	62	86	128	133

Mobile AP mode throughput - VHT 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
OpenSecurity	44	60	74	75
WPA2-AES	43	59	74	75
WPA3-SAE	43	59	74	75

Mobile AP mode throughput - VHT 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	64	53	112	75
WPA2-AES	65	51	111	75
WPA3-SAE	62	51	112	75

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
OpenSecurity	45	60	76	76
WPA2-AES	44	59	76	76
WPA3-SAE	44	59	76	76

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
OpenSecurity	73	100	155	178
WPA2-AES	72	99	152	176
WPA3-SAE	72	99	152	176

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
OpenSecurity	80	86	211	189
WPA2-AES	84	87	223	188
WPA3-SAE	83	94	224	192

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
OpenSecurity	53	66	85	123
WPA2-AES	52	65	83	122
WPA3-SAE	52	65	83	120

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
OpenSecurity	75	102	124	166
WPA2-AES	74	100	121	148
WPA3-SAE	73	101	121	154

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
OpenSecurity	54	68	84	124
WPA2-AES	53	66	83	122
WPA3-SAE	54	66	83	123

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
OpenSecurity	76	107	155	193
WPA2-AES	75	105	152	192
WPA3-SAE	76	106	151	191

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
OpenSecurity	86	118	220	187
WPA2-AES	86	119	221	185
WPA3-SAE	86	116	220	188

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 {#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 {#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 {#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 {#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

Parent topic:*IW611/IW612 release notes*

Known issues

Component	Description
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 budsWhile 4-CIS streaming, disconnection with connection timeout observed on first CIS SINK with Samsung Galaxy budsOnly two streams (CIS/BIS) with one channel is supported.

Parent topic:*IW611/IW612 release notes*

RW610/RW612 release notes

Package information

- SDK version: 25.06.00

Parent topic:*RW610/RW612 release notes*

Version information

- Wi-Fi firmware version: 18.99.6.p40
  - rw61x_sb_wifi_a2.bin for A2
  - 18 - Major revision
  - 99 - Feature pack
  - 6 - Release version
  - p40 - Patch number
- Bluetooth LE firmware version: 18.25.6.p40
  - rw61x_sb_ble_a2.bin for A2
  - 18 - Major revision
  - 25 - Feature pack
  - 6 - Release version
  - p40 - Patch number
- 802.15.4 and Bluetooth LE (up to core 4.1) firmware version: 18.34.6.p40
  - rw61x_sb_ble_15d4_combo_a2.bin for A2
  - 18 - Major revision
  - 34 - Feature pack
  - 6 - Release version
  - p40 - 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 | WPA3 SAE (R3)
- 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
OpenSecurity	38	38	62	62
WPA2-AES	37	37	61	63
WPA3-SAE	37	37	60	61

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	39	39	64	64
WPA2-AES	37	38	62	64
WPA3-SAE	39	38	62	64

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	41	75	74
WPA2-AES	41	41	73	74
WPA3-SAE	40	41	72	73

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
OpenSecurity	42	42	76	76
WPA2-AES	42	41	75	75
WPA3-SAE	42	41	75	74

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
OpenSecurity	44	45	97	99
WPA2-AES	43	44	96	98
WPA3-SAE	42	44	97	98

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
OpenSecurity	47	47	100	103
WPA2-AES	45	46	100	101
WPA3-SAE	47	46	100	101

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
OpenSecurity	39	39	62	62
WPA2-AES	39	39	61	61
WPA3-SAE	38	39	61	61

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
OpenSecurity	40	40	63	63
WPA2-AES	39	39	62	61
WPA3-SAE	39	39	62	61

Mobile AP throughput - VHT 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
OpenSecurity	43	43	73	73
WPA2-AES	43	42	72	72
WPA3-SAE	43	42	73	72

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
OpenSecurity	44	44	74	74
WPA2-AES	43	43	74	74
WPA3-SAE	43	43	74	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
OpenSecurity	48	48	95	96
WPA2-AES	47	47	98	95
WPA3-SAE	47	47	97	95

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
OpenSecurity	49	49	96	97
WPA2-AES	48	48	101	97
WPA3-SAE	48	48	101	97

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 {#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

Parent topic:*RW610/RW612 release notes*

Known issues

Component	Description
Wi-Fi	—
Bluetooth LE	—
Zigbee	-

Parent topic:*RW610/RW612 release notes*

IW610 release notes

Package information

- SDK version: 25.06.00

Parent topic:*IW610 release notes*

Version information

- Wireless SoC: IW610
- Wi-Fi and Bluetooth/Bluetooth LE firmware version: 18.99.5.p66
  - 18 - Major revision
  - 99 - Feature pack
  - 5 - Release version
  - p66 - Patch number

Parent topic:*IW610 release notes*

Host platform

- IW610 platform running FreeRTOS
- 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.

WFA certifications

- STA | 802.11n
- STA | PMF
- STA | FFD
- STA | SVD
- STA | WPA3 SAE (R3)
- 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:*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
OpenSecurity	40	42	63	61
WPA2-AES	40	47	60	62
WPA3-SAE	40	39	60	62

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	42	49	64	64
WPA2-AES	41	48	62	63
WPA3-SAE	41	48	62	63

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	45	54	75	74
WPA2-AES	45	53	73	73
WPA3-SAE	44	53	73	72

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
OpenSecurity	46	48	77	70
WPA2-AES	45	47	74	68
WPA3-SAE	46	47	74	68

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
OpenSecurity	51	62	98	98
WPA2-AES	50	60	96	91
WPA3-SAE	51	60	96	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
OpenSecurity	53	60	101	94
WPA2-AES	53	58	99	93
WPA3-SAE	52	58	99	93

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
OpenSecurity	39	39	63	62
WPA2-AES	39	38	60	60
WPA3-SAE	39	38	60	60

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
OpenSecurity	40	39	64	62
WPA2-AES	39	39	61	61
WPA3-SAE	39	38	61	61

Mobile AP throughput - VHT 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
OpenSecurity	43	43	74	73
WPA2-AES	42	42	73	71
WPA3-SAE	42	42	74	72

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
OpenSecurity	43	43	75	73
WPA2-AES	43	42	74	72
WPA3-SAE	43	43	74	72

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
OpenSecurity	46	46	95	94
WPA2-AES	45	45	96	91
WPA3-SAE	45	45	96	91

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
OpenSecurity	47	47	96	94
WPA2-AES	46	46	98	91
WPA3-SAE	46	46	99	91

Parent topic: Wi-Fi throughput

Parent topic:*IW610 release notes*

Known issues

Component	Description
Wi-Fi	—
Bluetooth LE	—

Parent topic:*IW610 release notes*

AW611 release notes

Note: The AW611 support is enabled in i.MX RT1180 EVKA.

Package information

- SDK version: 25.06.00

Parent topic:*AW611 release notes*

Version information

- Wireless SoC: AW611
- Wi-Fi and Bluetooth/Bluetooth LE firmware version: 18.99.3.p25.11
  - 18 - Major revision
  - 99 - Feature pack
  - 3 - Release version
  - p25.11 - Patch number

Parent topic:*AW611 release notes*

Host platform

- i.MX RT1180 EVKA Platform 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 | WPA3 SAE (R3)
- 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:*AW611 release notes*

Wi-Fi throughput

Throughput test setup

- Environment: Shield Room - Over the Air
- Access Point: Asus AX88u
- DUT: AW611 uBlox (Module: U-BLOX_Jody_W5 M.2) with EVK-MIMXRT1180 EVKA 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

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
OpenSecurity	43	47	45	60
WPA2-AES	43	46	48	59
WPA3-SAE	47	49	63	63

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	68	82	131	131
WPA2-AES	72	82	130	129
WPA3-SAE	68	81	129	130

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
OpenSecurity	45	51	63	65
WPA2-AES	45	51	63	65
WPA3-SAE	45	51	63	65

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
OpenSecurity	67	83	129	134
WPA2-AES	66	83	129	133
WPA3-SAE	65	83	129	133

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	49	53	72	71
WPA2-AES	48	52	73	70
WPA3-SAE	52	56	75	75

STA mode throughput - VHT Mode | 2.4 GHz Band | 40 MHz (HT)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	74	88	172	172
WPA2-AES	75	92	171	169
WPA3-SAE	77	92	172	171

STA mode throughput - VHT 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
OpenSecurity	50	58	76	78
WPA2-AES	49	57	76	77
WPA3-SAE	49	57	76	77

STA mode throughput - VHT 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
OpenSecurity	74	93	175	177
WPA2-AES	74	93	174	174
WPA3-SAE	73	93	173	175

STA mode throughput - VHT Mode | 5 GHz Band | 80 MHz (HT)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	88	94	221	196
WPA2-AES	87	95	219	194
WPA3-SAE	89	95	219	195

STA mode throughput - HE 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	60	60	98	115
WPA2-AES	62	61	94	113
WPA3-SAE	61	59	97	108

STA mode throughput - HE Mode | 2.4 GHz Band | 40 MHz (HT)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	77	71	215	190
WPA2-AES	77	72	212	187
WPA3-SAE	76	72	152	189

STA mode throughput - HE 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
OpenSecurity	63	65	127	128
WPA2-AES	63	67	125	128
WPA3-SAE	63	67	125	126

STA mode throughput - HE 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
OpenSecurity	79	64	212	199
WPA2-AES	78	68	218	199
WPA3-SAE	79	68	217	198

STA mode throughput - HE Mode | 5 GHz Band | 80 MHz (HT)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	88	70	219	192
WPA2-AES	87	72	219	193
WPA3-SAE	91	72	220	194

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
OpenSecurity	40	52	61	62
WPA2-AES	40	51	61	62
WPA3-SAE	40	51	61	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	63	83	116	130
WPA2-AES	67	82	115	131
WPA3-SAE	60	81	115	132

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	39	49	60	62
WPA2-AES	39	49	60	62
WPA3-SAE	41	51	63	62

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	66	89	128	133
WPA2-AES	64	87	128	133
WPA3-SAE	62	86	128	133

Mobile AP mode throughput - VHT 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
OpenSecurity	44	60	74	75
WPA2-AES	43	59	74	75
WPA3-SAE	43	59	74	75

Mobile AP mode throughput - VHT 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	64	53	112	75
WPA2-AES	65	51	111	75
WPA3-SAE	62	51	112	75

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
OpenSecurity	45	60	76	76
WPA2-AES	44	59	76	76
WPA3-SAE	44	59	76	76

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
OpenSecurity	73	100	155	178
WPA2-AES	72	99	152	176
WPA3-SAE	72	99	152	176

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
OpenSecurity	80	86	211	189
WPA2-AES	84	87	223	188
WPA3-SAE	83	94	224	192

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
OpenSecurity	53	66	85	123
WPA2-AES	52	65	83	122
WPA3-SAE	52	65	83	120

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
OpenSecurity	75	102	124	166
WPA2-AES	74	100	121	148
WPA3-SAE	73	101	121	154

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
OpenSecurity	54	68	84	124
WPA2-AES	53	66	83	122
WPA3-SAE	54	66	83	123

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
OpenSecurity	76	107	155	193
WPA2-AES	75	105	152	192
WPA3-SAE	76	106	151	191

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
OpenSecurity	86	118	220	187
WPA2-AES	86	119	221	185
WPA3-SAE	86	116	220	188

Parent topic: Wi-Fi throughput

Parent topic:*AW611 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:*AW611 release notes*

Bug fixes and/or feature enhancements {#bug_fixes_and_or_feature_enhancements_04}

Firmware version: 18.99.3.p10.5 to 18.99.3.p17.9 {#firmware_version_18_99_3_p10_5_to_18_99_3_p17_9_0}

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 {#firmware_version_18_99_3_p17_9_to_18_99_3_p21_154_0}

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

Parent topic:*AW611 release notes*

Known issues

Component	Description
Bluetootl	Packet lost would be observed in CIS case which causes audio noise.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 budsWhile 4-CIS streaming, disconnection with connection timeout observed on first CIS SINK with Samsung Galaxy budsOnly two streams (CIS/BIS) with one channel is supported.

Parent topic:*AW611 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
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

continues on next page

Table 4 – continued from previous page

Abbreviation	Definition
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)

2.9.2 EdgeFast Bluetooth

Currently we provide pdf version of those documentation, later release may convert the pdf documentation to markdown for better review and aligned format.

- EdgeFast BT PAL API Reference Manual pdf.

MCUXpressoSDK EdgeFast Bluetooth Protocol Abstraction

Introduction This document provides an overview of the EdgeFast Bluetooth Protocol Abstraction Layer stack software based on FreeRTOS OS on the NXP board with variant wireless module chipsets. This document covers hardware setup, build, and usage of the provided demo applications.

Stack API Reference EdgeFast Bluetooth Protocol Abstraction Layer is a wrapper layer on top of the bluetooth host stack. Zephyr Bluetooth host stack API is used as the basis of the EdgeFast Bluetooth Protocol Abstraction Layer with some enhancement on A2DP/SPP/HFP.

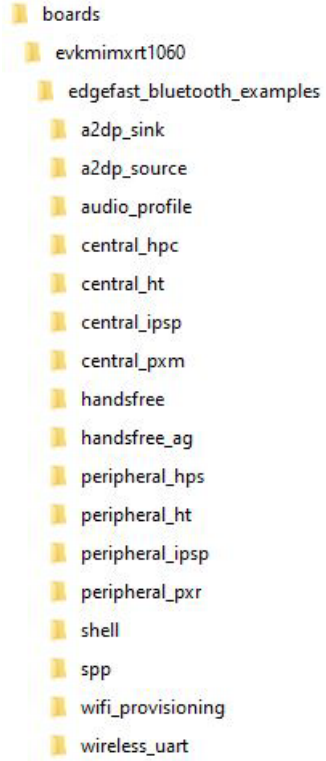
The APIs of the EdgeFast Bluetooth Protocol Abstraction Layer host stack are described in the EdgeFast Bluetooth Protocol Abstraction Layer RM document.

Note: The online document of the Zephyr Bluetooth Host stack is available here: https://docs.zephyrproject.org/latest/reference/bluetooth/index.html.

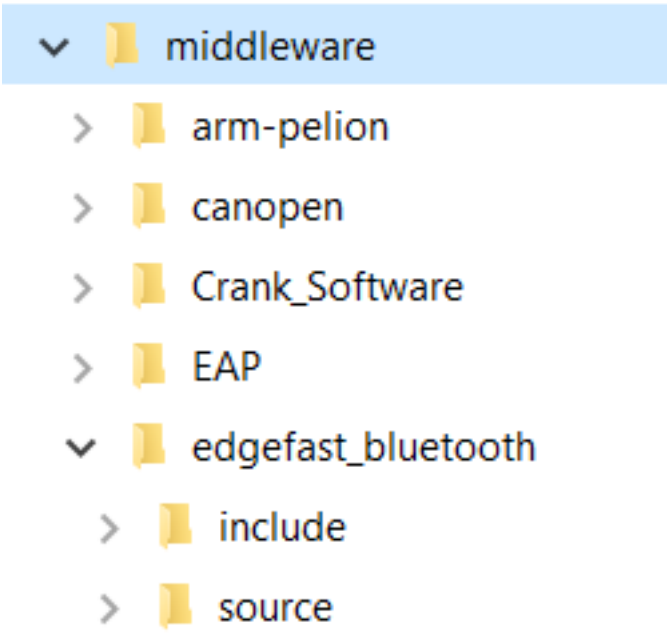
Parent topic:*Introduction*

Overview The EdgeFast Bluetooth Protocol Abstraction Layer host stack software is built based on MCUXpresso SDK. The following chapter uses RT1060 as an example, other boards have similar folder structure and corresponding document.

Folder structure The following figure shows the EdgeFast Bluetooth examples folder structure.



The following figure shows the EdgeFast Bluetooth Protocol Abstraction Layer host stack folder structure.



The following table provides information regarding the structure and description.

|**Folder**

|**Description**

| |————|————| |*boards/*

CMSIS/

devices/

docs/

middleware/

rtos/

tools/

|MCUXpresso SDK directory. Refer to Chapter 5

Release contents of MCUXpresso SDK Release Notes at *root/docs/ MCUXpresso SDK Release Notes for EVK-MIMXRT1060.pdf* to *know the details*

| |*boards/<board>/wireless/edgefast_bluetooth_examples*

|EdgeFast Bluetooth Protocol Abstraction Layer host stack example projects| |*middleware/wireless/edgefast_bluetooth*

|EdgeFast Bluetooth Protocol Abstraction Layer host stack source code

|

The EdgeFast Bluetooth folder includes two subfolders:

- **include**: This subfolder includes EdgeFast Bluetooth Protocol Abstraction Layer host stack headers.
- **source**: This subfolder includes EdgeFast Bluetooth Protocol Abstraction Layer host stack source code based on the Ethermind Bluetooth host stack APIs.

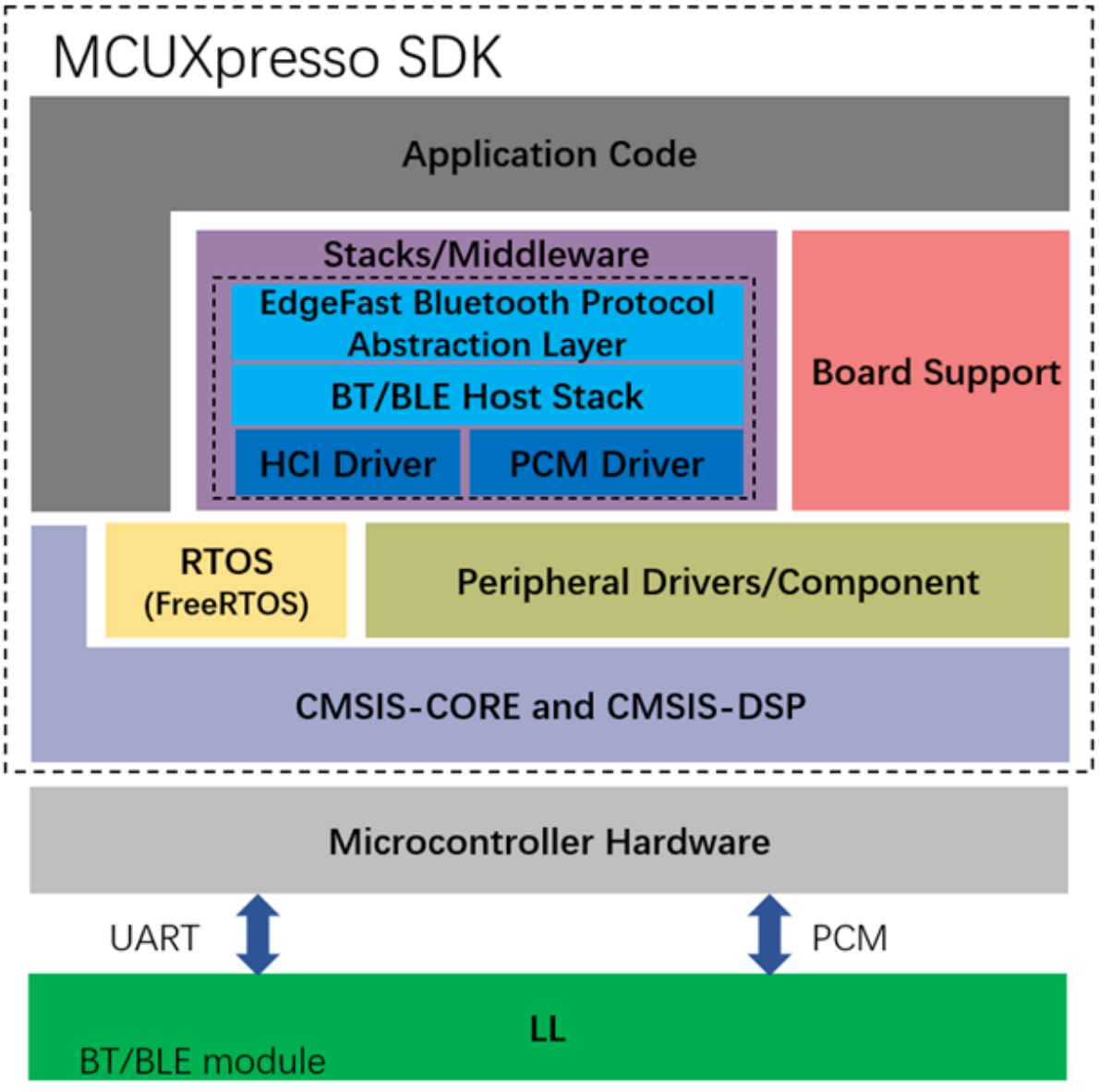
Parent topic:*Overview*

Architecture The figure Architecture of EdgeFast Bluetooth Protocol Abstraction Layer demo in MCUXpresso SDK below shows that the EdgeFast Bluetooth Protocol Abstraction Layer host stack is integrated into the MCUXpresso SDK as a middleware component. It leverages the RTOS, the board support, the peripheral driver/component, and other components in the MCUXpresso SDK. The Bluetooth application is built on top of the EdgeFast Bluetooth Protocol Abstraction Layer host stack and supports different peripheral features, Bluetooth features, and different RTOSes required by the user.

MCUXpresso SDK has the dual-chip architecture defined by EdgeFast Bluetooth Protocol Abstraction Layer project, the Bluetooth application code, and the EdgeFast Bluetooth Protocol Abstraction Layer host stack running on the reference board. For example, MIMXRT1060-EVK and the Linker Layer (LL) run on the Bluetooth modules like AW-AM457-USD, Murata Type 1XK, and Murata Type 1ZM and has single-chip architecture. Bluetooth Host stack and LL runs on the same chip, and communicate with Internal Communication Unit (IMU).

The communication between the host stack and the LL is implemented via the standard HCI UART interface and PCM interface for voice, or the IMU interface.

For details about the different components in MCUXpresso SDK, see *Getting Started with MCUXpresso SDK User's Guide* (document MCUXSDKGSUG) at *root/docs/Getting Started with MCUXpresso SDK.pdf*. For details on possible hardware rework requirements, see the hardware rework guide document of the relative board. For example, Hardware Rework Guide for EdgeFast BT PAL.



Parent topic:*Overview*

Features This section provides an overview of Bluetooth features, toolchain support, and RTOS support.

Bluetooth features

- Bluetooth 5.0 compliant
- Protocol support
  - L2CAP, GAP, GATT, RFCOMM, SDP, and SM

    Note: The Enhanced Attribute (EATT) protocol is not supported in the current version. However, the support will be available in a future version.
- Classic profile
  - SPP, A2DP, and HFP
- LE profile
  - HTP, PXP, IPSP, HPS
- Integrated the Fatfs based on USB Host MSD in SDK
- Digital Audio Interface including PCM interface for HFP

Parent topic: Features

Toolchain support

- IAR Embedded Workbench for ARM®
- MCUXpresso IDE
- Keil® MDK/µVision
- Makefiles support with GCC from Arm Embedded

Note: For details on IDE Development tools version details, see Section 3, Development tools in MCUXpresso SDK Release Notes (document MCUXSDKMIMXRT106XRN). The Release Notes document is available at *root/docs/ MCUXpresso SDK Release Notes for EVK-MIMXRT1060.pdf*.

Parent topic: Features

RTOS support

- FreeRTOSTMOS

Parent topic: Features

Parent topic:*Overview*

Examples list

- The following examples are provided. Not all the examples are implemented on all the boards. See the board package for a list of the implemented examples.
  - **central_hpc (central http proxy service client)**: Demonstrates a basic Bluetooth Low Energy Central role functionality. The application scans for other Bluetooth Low Energy devices and establishes a connection to the peripheral with the strongest signal. The application specifically looks for HPS Server and programs a set of characteristics that configures a Hyper Text Transfer Protocol (HTTP) request, initiates request, and reads the response once connected.

- **central_ht (central health thermometer)**: Demonstrates a basic Bluetooth Low Energy Central role functionality. The application scans for other Bluetooth Low Energy devices and establishes a connection to the peripheral with the strongest signal. The application specifically looks for health thermometer sensor and reports the die temperature readings once connected.
- **central_ipsp (central Internet protocol support profile)**: Demonstrates a basic Bluetooth Low Energy Central role functionality. The application scans for other Bluetooth Low Energy devices and establishes connection to the peripheral with the strongest signal. The application specifically looks for IPSP Service and communicates between the devices that support IPSP. Once connected, the communication is done using IPv6 packets over the Bluetooth Low Energy transport.
- **central_pxm (central proximity monitor)**: Demonstrates a basic Bluetooth Low Energy Central role functionality. The application scans for other Bluetooth Low Energy devices and establishes a connection to the peripheral with the strongest signal. The application specifically looks for Proximity Reporter.
- **peripheral beacon**: Demonstrates the Bluetooth Low Energy Peripheral role, This application implements types of beacon applications.
  - **beacon**: Demonstrates the Bluetooth Low Energy Broadcaster role functionality by advertising Company Identifier, Beacon Identifier, UUID, A, B, C, RSSI.
  - **Eddystone**: The Eddystone Configuration Service runs as a GATT service on the beacon while it is connectable and allows configuration of the advertised data, the broadcast power levels, and the advertising intervals.
  - **iBeacon**: Demonstrates the Bluetooth Low Energy Broadcaster role functionality by advertising an Apple iBeacon.
- **peripheral_hps (peripheral http proxy service)**: Demonstrates the Bluetooth Low Energy Peripheral role. The application specifically exposes the HTTP Proxy GATT Service.
- **peripheral_ht (peripheral health thermometer)**: Demonstrates the Bluetooth Low Energy Peripheral role. The application specifically exposes the HT (Health Thermometer) GATT Service. Once a device connects, it generates dummy temperature values.
- **peripheral_ipsp (peripheral Internet protocol support profile)**: Demonstrates the Bluetooth Low Energy Peripheral role. The application specifically exposes the Internet Protocol Support GATT Service.
- **peripheral_pxr (peripheral proximity reporter)**: Demonstrates the Bluetooth Low Energy Peripheral role. The application specifically exposes the Proximity Reporter (including LLS, IAS, and TPS) GATT Service.
- **wireless uart**: The application automatically starts advertising the wireless uart service and connects to the wireless uart service after the role switch. The wireless UART service is a custom service that implements a custom writable ASCII Char characteristic (UUID: 01ff0101-ba5e-f4ee-5ca1-eb1e5e4b1ce0) that holds the character written by the peer device.
- **spp (serial prot profile)**: Application demonstrates the use of the SPP feature.
- **handsfree**: Application demonstrating usage of the Hands-free Profile (HFP) feature.
- **handsfree_ag**: Application demonstrating usage of the Hands-free Profile Audio Gateway (HFP-AG) feature.
- **a2dp_sink**: Application demonstrating how to use the a2dp sink feature.
- **a2dp_source**: Application demonstrating how to use the a2dp source feature.
- **audio_profile**: Demonstrates the following functions.

    * There are five parts working in the demo: AWS cloud, Android app, audio demo (running on RT1060), U-disk, and Bluetooth headset.
    * With an app running on the smartphone (Android phone), the end user connects to the AWS cloud and controls the audio demo running on the RT1060 EVK board through AWS cloud. Some operations like play, play next, and pause are used to control the media play functionalities.
    * Audio demo running on the RT1060 EVK board connects to the AWS through WiFi. A connection establishes between the RT1060 EVK board and a Bluetooth headset. To get the media resource (mp3 files) from the U-disk, an HS USB host is enabled, and a U-disk with mp3 files is connected to RT1060 EVK board via the USB port. The audio demo searches the root directory of the U-disk for the music files (only mp3 files are supported) and uploads the song file list to AWS. The song list is shown in the app running on the smartphone. The music can then be played out via the Bluetooth headset once end user controls the app to play the mp3 file.

  – **wifi_provisioning**: Demonstrates the WiFi provisioning service that safely sends credential from phone to device over Bluetooth low energy. By default, AWS Wi-Fi provisioning demo starts advertising if the Wi-Fi access point (AP) is not configured and waits for the Wi-Fi AP configuration. After connecting to the Android APK, the demo executes the request from cellphone and sends the response. When the Wi-Fi AP is configured, the Shadow demo connects to the AWS via Wi-Fi and publishes the configured Wi-Fi AP information.
  – **shell**: Shell application demonstrating the shell mode of the simplified Adapter APIs.

Parent topic:*Overview*

Hardware For dual-chip implementation, the Bluetooth demo runs on a (reference board) along with the ported EdgeFast Bluetooth Protocol Abstraction Layer API host stack. The Linker Layer (LL) runs on a wireless module. A standard UART HCI and PCM is used to communicate between the two boards, the IMU is used to communicate in between. The Bluetooth host and controller stack run on different boards. The demo hardware requires two different boards; a development board for host stack and application and a wireless module adapter board for controller running. For example, the evkmimxrt1060 and uSD-15x15 Adapter Board for AW-AM457-uSD board, or any of the supported Murata modules with the Murata uSD-M.2 adapter. For details on the board hardware requirement and board setting, see the following documents. For one-chip implementation, the Bluetooth demo, EdgeFast Bluetooth Protocol Abstraction Layer API host stack, and LL run on one chip and they communicate with IMU.

- Hardware rework guide document of the relative board, Hardware Rework Guide for MIMXRT1060-EVK and AW-AM457-uSD, or Hardware Interconnection Guide for i.MX RT EVKs and Murata M.2 modules.
- Readme file of the examples.

Reference boards list

- MIMXRT1170: For details, see the quick start guide of this reference board (MIMXRT1170).
- MIMXRT685-EVK: For details, see the quick start guide of this reference board (MIMXRT685-EVK).
- MIMXRT595-EVK: For details, see the quick start guide of this reference board. (MIMXRT595-EVK).
- MIMXRT1050-EVKB: For details, see the quick start guide of this reference board (MIMXRT1050-EVKB).

Parent topic:*Hardware*

Dual-chip wireless module list

Module	HCI
uSD-15x15 Adapter Board for AW-AM457-uSD	UART
uSD-15x15 Adapter Board for AW-CM358-uSD	UART
uSD-15x15 Adapter Board for AW-AM510-uSD	UART
AW-CM358MA	UART
AW-CM510MA	UART
K32W061	UART
Murata uSD-M.2 Adapter (LBEE0ZZ1WE-uSD-M2) and Embedded Artists 1ZM M.2 Module (EAR00364)	UART
Murata uSD-M.2 Adapter (LBEE0ZZ1WE-uSD-M2) and Embedded Artists 1XK M.2 Module (EAR00385)	UART

For details on AzureWave module, see the quick start guide of this reference board AW-AM457-uSD, AW-CM358-uSD, AW-CM358MA, AW-AM510-uSD, AW-CM510MA, and K32W061.

For Murata documentation, refer to the Quick Start Guide and User Guide here.

Note: The boards and wireless module lists are not random combination. For the wireless module support list of specific board, see the readme.txt of each example.

Parent topic:*Hardware*

Demo This topic lists the steps to run a demo application using IAR, steps to run a demo application using MCUXpresso IDE, and steps to download LL firmware from the reference board. The following chapter uses RT1060 and peripheral_ht as an example.

Before you run the example, see the readme.txt in current the peripheral_ht directory and the Hardware Rework Guide for EdgeFast BT PAL document to set the jumper and connect the wireless module with development board.

The uSD type wireless module is similar to the Development board connector in the Run an IAR example section. If the module is M2 type, connect the module to the onboard M2 interface.

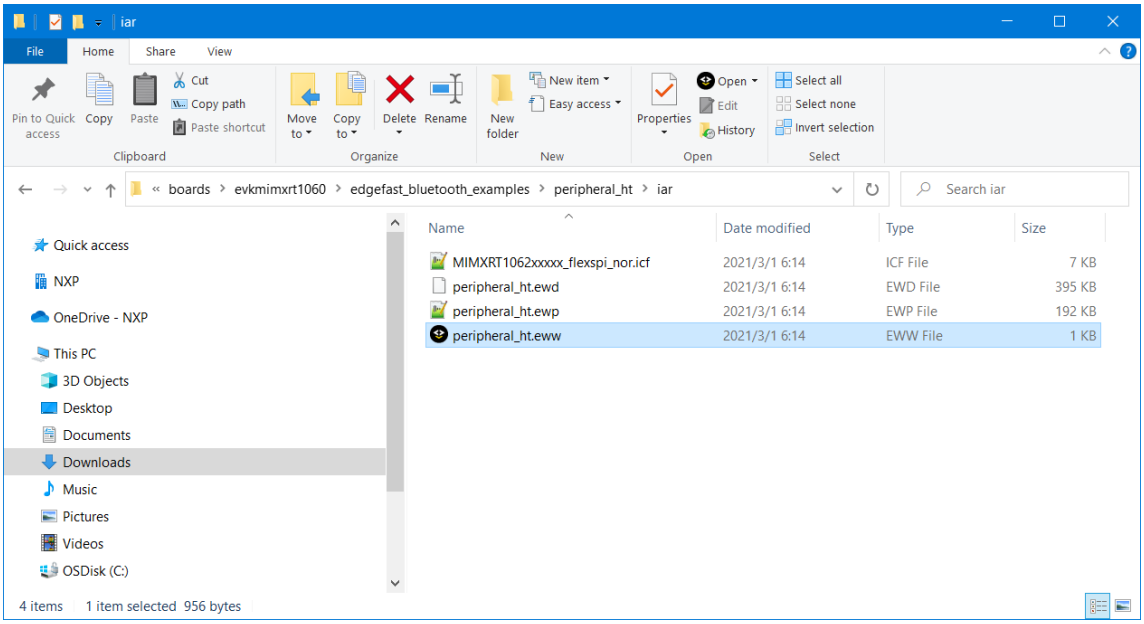
Run a demo application using IAR This document uses EVKRT1060 EdgeFast Bluetooth Protocol Abstraction Layer API example to describe the steps to open a project, build an example, and run a project. For details, see Section 3 in *Getting Started with MCUXpresso SDK User's Guide*(document MCUXSDKGSUG) at*root/docs/Getting Started with MCUXpresso SDK.pdf*.

Open an IAR example For the IAR Embedded Workbench, unpack the contents of the archive to a folder on a local drive.

1. The example projects are available at:

   <root>/boards/evkmimxrt1060/edgefast_bluetooth_examples/peripheral_ht/iar

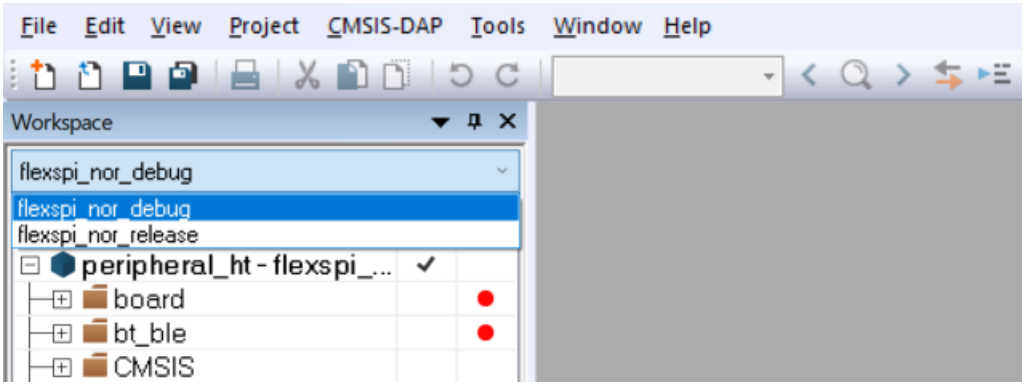
2. Open the IAR workspace file. For example, the highlighted *.eww format file



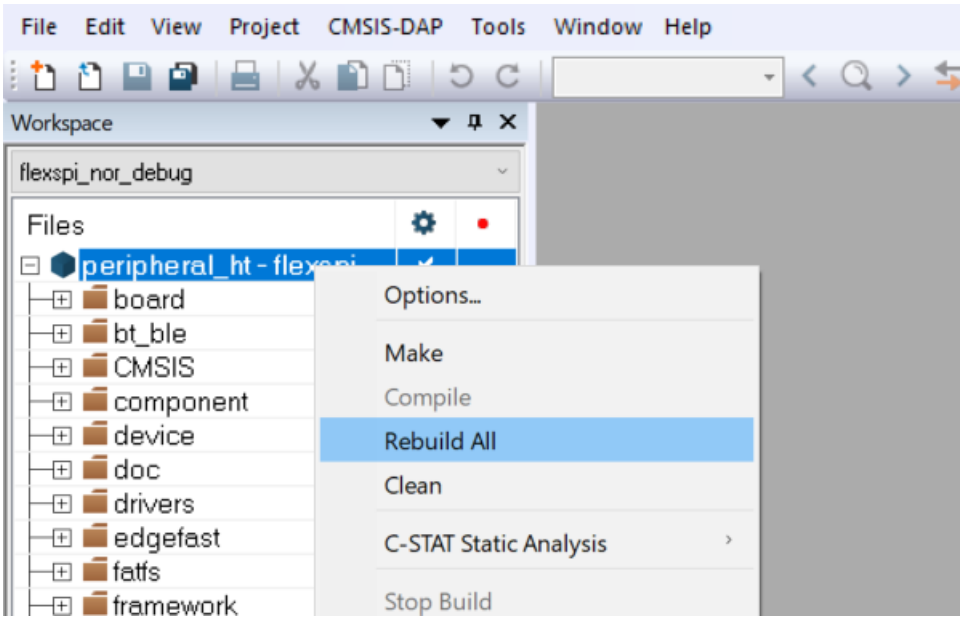
Parent topic: Run a demo application using IAR

Build an IAR example

1. Select flexspi_nor_debug or flexspi_nor_release configurations from the drop-down selector above the project tree in the workspace.



2. Build the EdgeFast Bluetooth Protocol Abstraction Layer project.

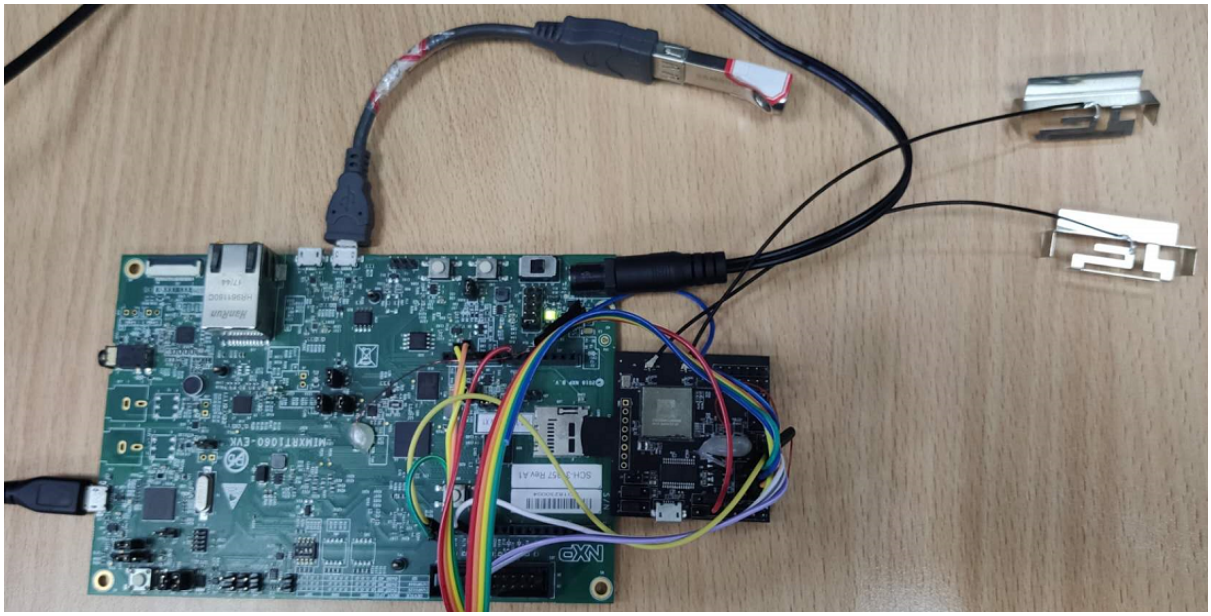


Note: Wireless module does not have flash hardware and requires 512 KB image loaded from board (such as RT1060) on system startup. The 512 KB image is kept on RT1060 side and only flexspi_nor target is supported for Bluetooth examples. Other targets are not supported because memory size limit.

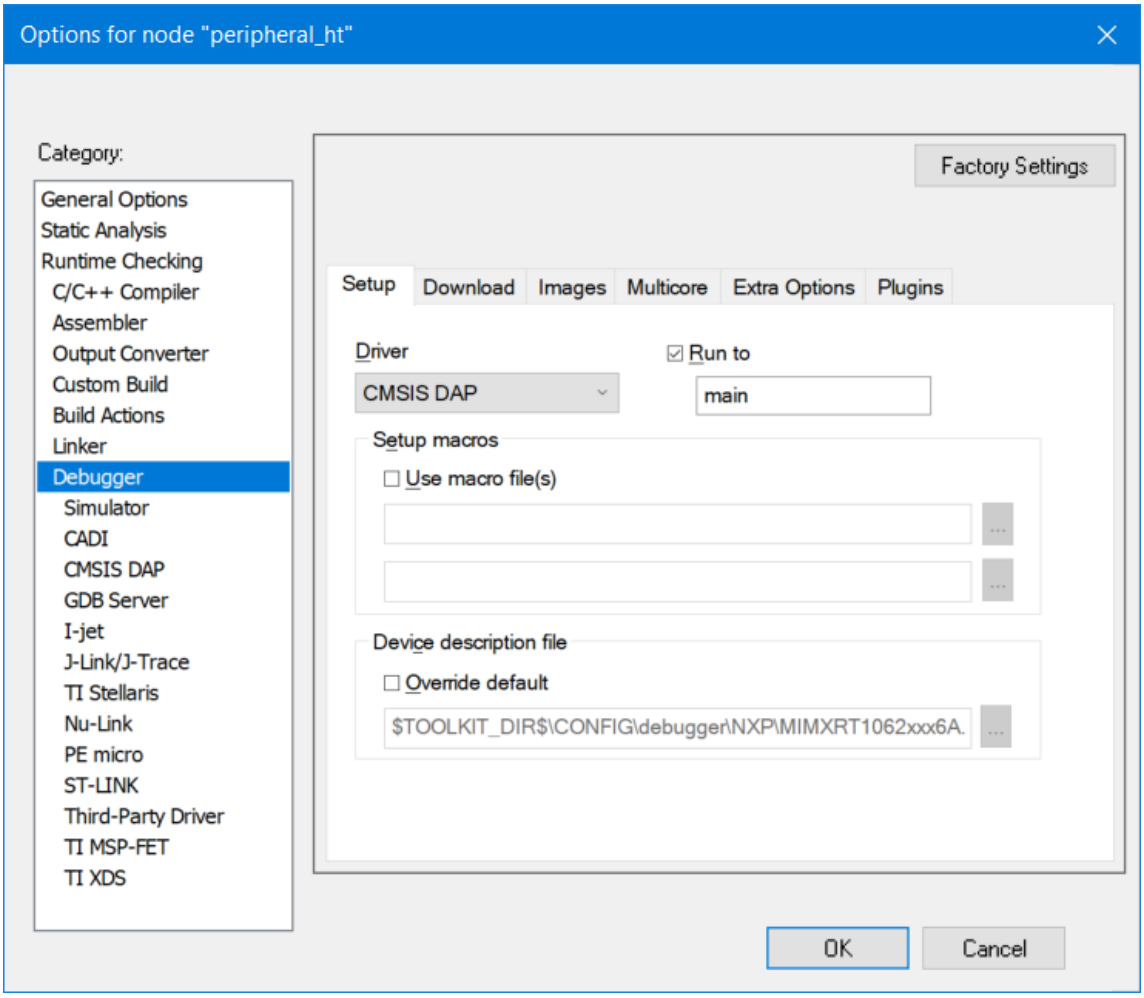
Parent topic: Run a demo application using IAR

Run an IAR example This document uses the peripheral_ht as an example to describe the steps to run an example. For details on other projects and compilers, see the readme file in the corresponding example directory.

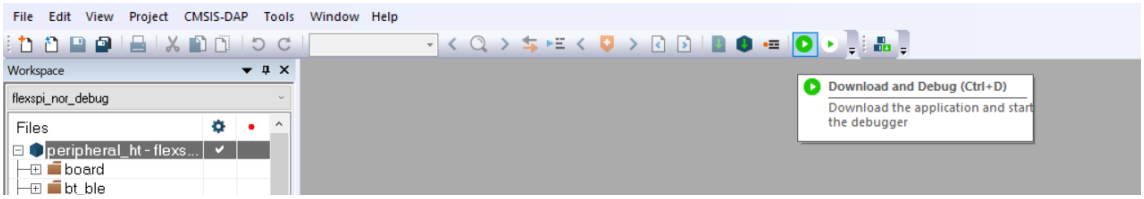
The following figure shows the connection of RT1060 and the uSD wireless module.



1. Connect the USB debug console port to PC. For example, connect J14 of EVKRT1060 to the PC.
2. Connect a 5 V power source to the J1 jack in the Wireless module board.
3. Make the appropriate debugger settings in the project options window, as shown in the figure below.



4. Click the **Download and Debug** button to flash the executable onto the board, as shown in the following figure. After the download is complete, if you must test the function of HFP, stop IAR debugging, and then connect the PCM interface. Reset the target board by manually.



5. Linker layer (LL) Firmware running in wireless module loads from EVKRT1060 by SDIO interface, so need take a bit time to download the LL firmware, “Initialize AW-AM457-uSD Driver” prints in the debug console. For example, it depends on the firmware. For details, see readme.txt.

Note: The projects are configured to use “CMSIS DAP” as the default debugger. Ensure that the OpenSDA chip of the board contains a CMSIS. DAP firmware or that the debugger selection corresponds to the physical interface used to interface to the board.

Parent topic: Run a demo application using IAR

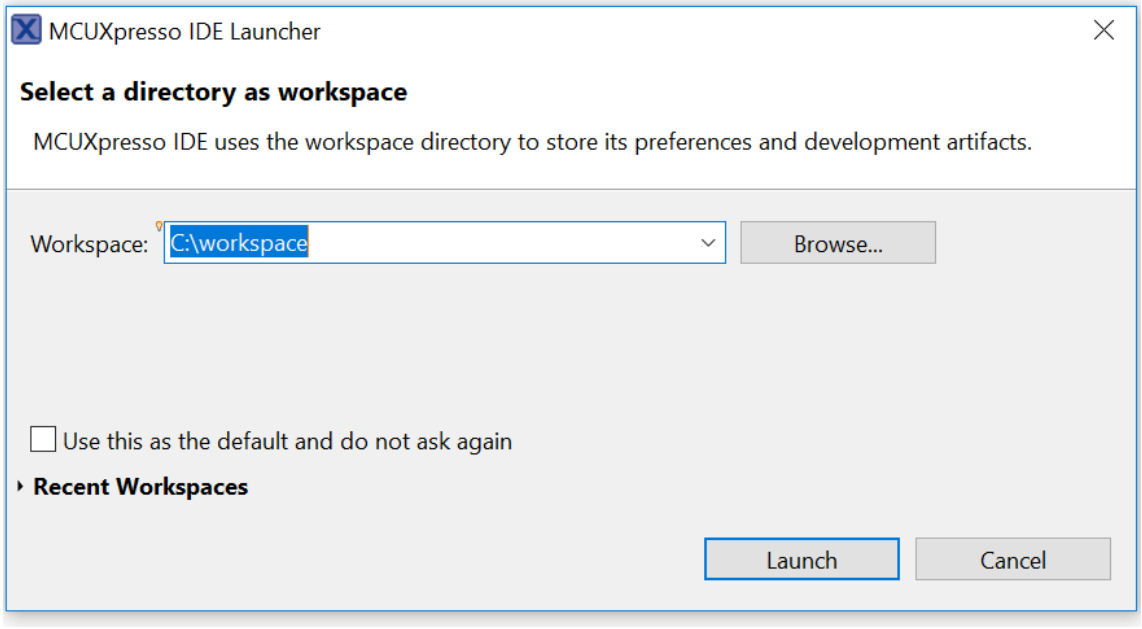
Parent topic:*Demo*

Run a demo application using MCUXpresso IDE This document uses peripheral_ht example to describe the steps to open a project, build an example, and run a project on MCUXpresso IDE.

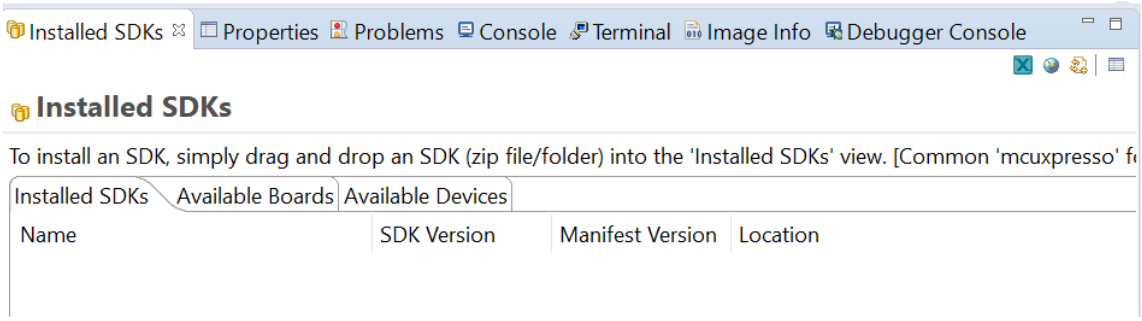
For details, see Section 3 in *Getting Started with MCUXpresso SDK User's Guide* (document MCUXSDKGSUG) at *root/docs/Getting Started with MCUXpresso SDK.pdf* and refer to the readme file in the corresponding demo's directory.

Open an MCUXpresso IDE example

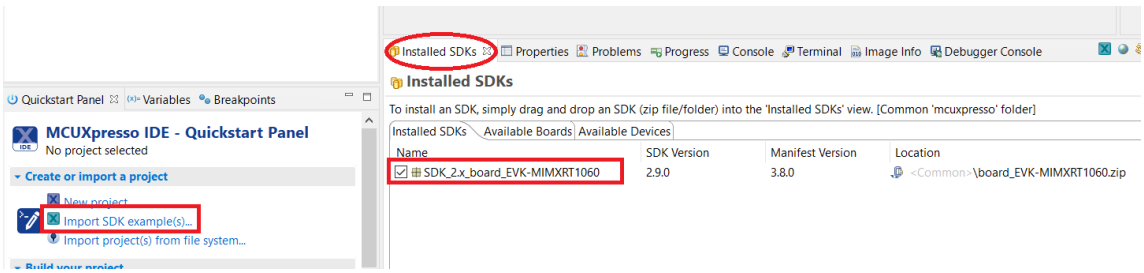
1. Open MCUXpresso IDE and open an existing or a new workspace location.



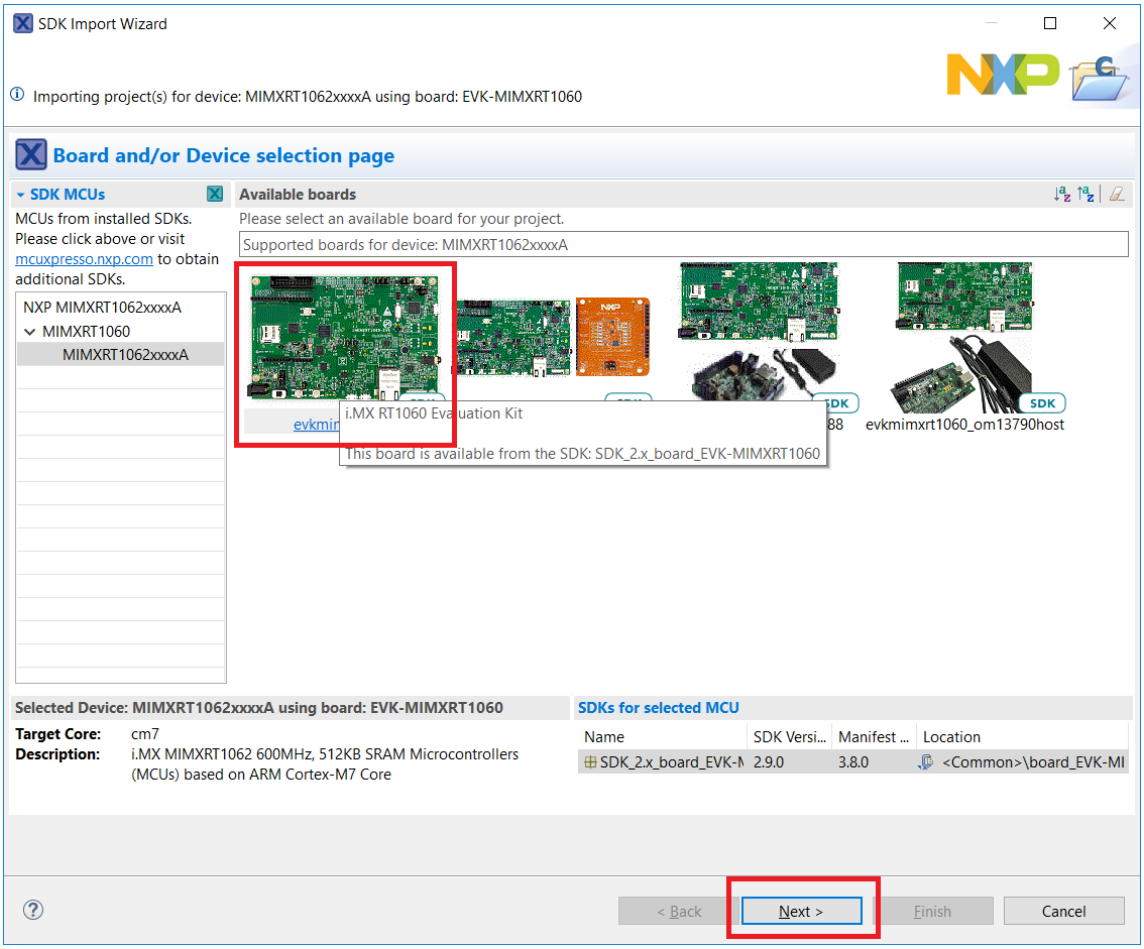
2. Drag and drop the package archive into the MCUXpresso Installed SDKs area in the lower right of the main window.



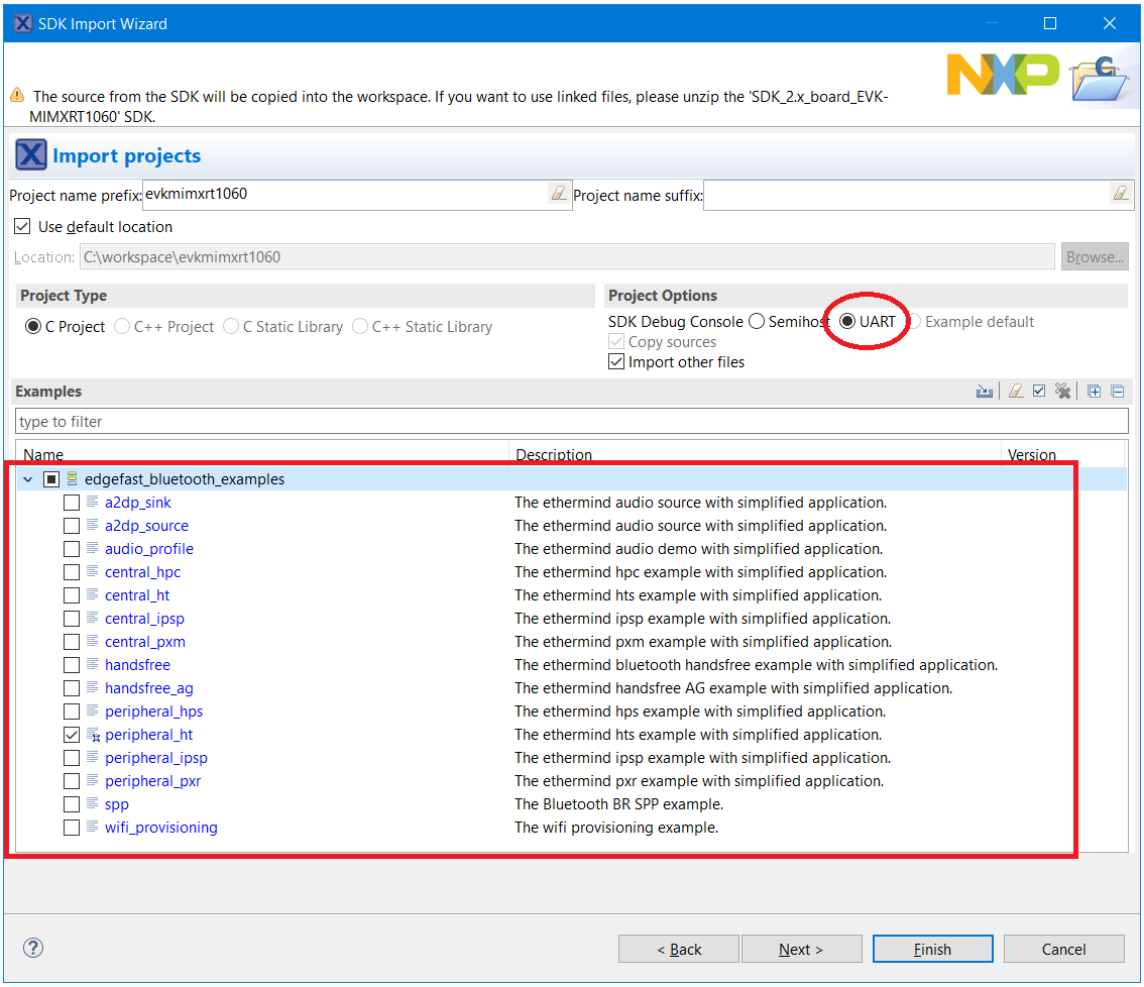
3. After the SDK is loaded successfully, select the **Import the SDK examples(s)...** to add examples to your workspace.



4. Select the evkmimxrt1060 board and click the **Next** button to select the desired example(s).



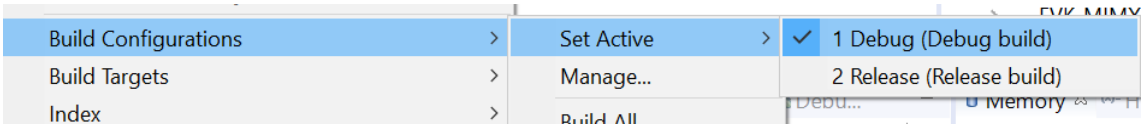
5. Select the evkmimxrt1060 board EdgeFast Bluetooth example. For example, peripheral_ht.
6. Ensure to change SDK debug console from **Semihost** to **UART**.
7. Click **Finish**.



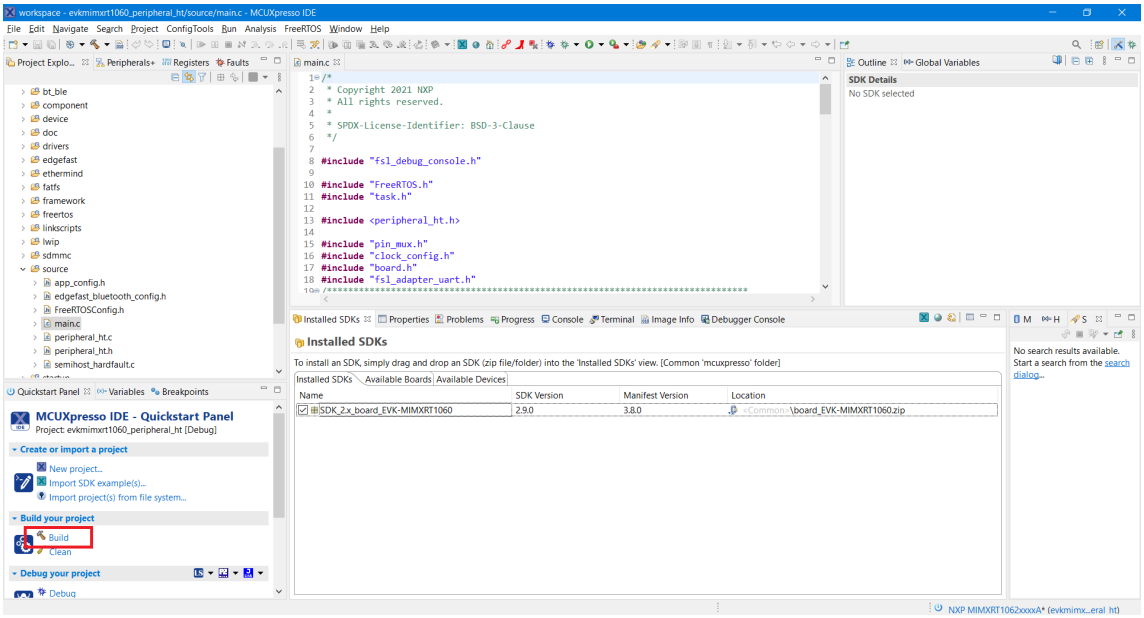
Parent topic: Run a demo application using MCUXpresso IDE

Build an MCUXpresso IDE example

1. Select desired target for your project.



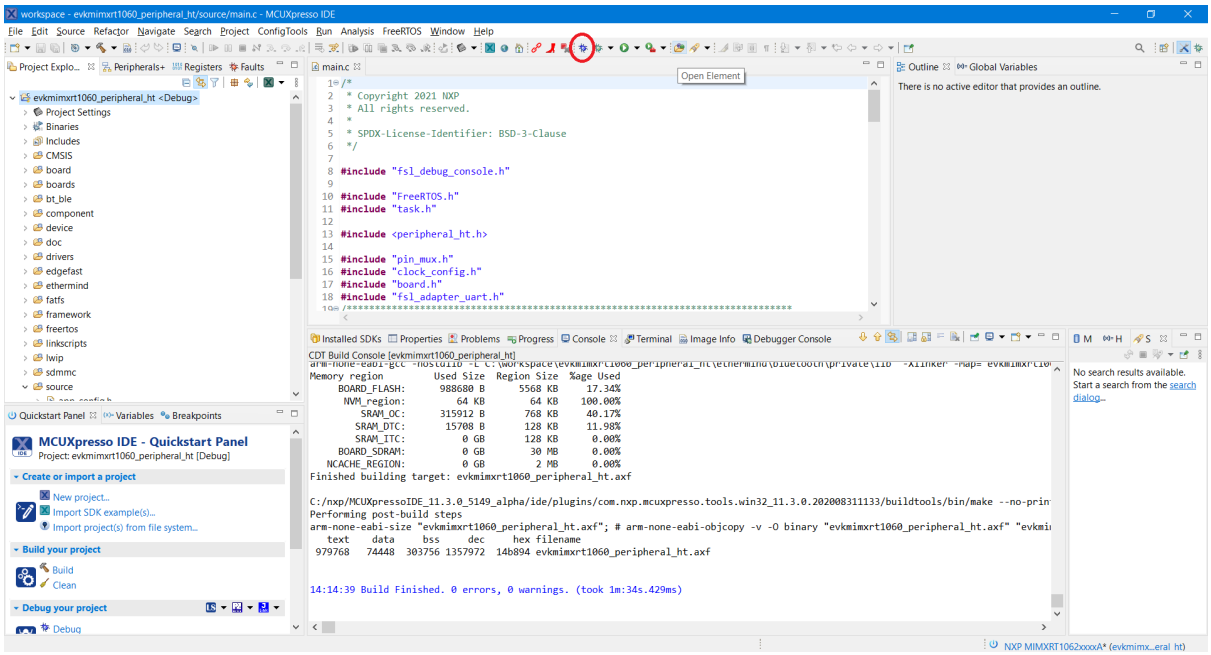
2. Build MCUXpresso IDE EdgeFast Bluetooth Protocol Abstraction Layer project.



Parent topic: Run a demo application using MCUXpresso IDE

Run an MCUXpresso IDE example For MCUXpresso IDE project running, all steps are similar to Run an IAR example except the steps of downloading image from compiler.

To download MCUXpresso IDE image to board, click the **Debug** button to download the executable file onto the board.



Parent topic: Run a demo application using MCUXpresso IDE

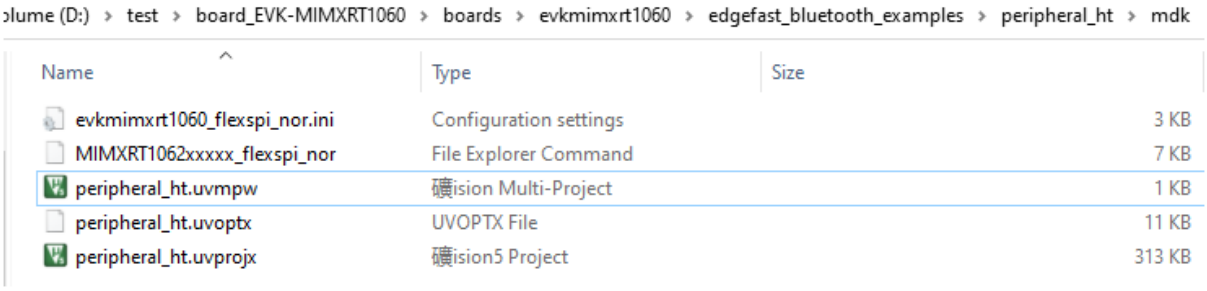
Parent topic:*Demo*

Run a demo application using MDK This document uses peripheral_ht example to describe the steps to open a project, build an example, and run a project on MDK.

For details, see the related section in the Getting Started with MCUXpresso SDK User's Guide (document: MCUXSDKGSUG) in the directory *root/docs/* and the readme file in the corresponding demo's directory.

Open an MDK project For the IAR Embedded Workbench, unpack the contents of the archive to a folder on a local drive.

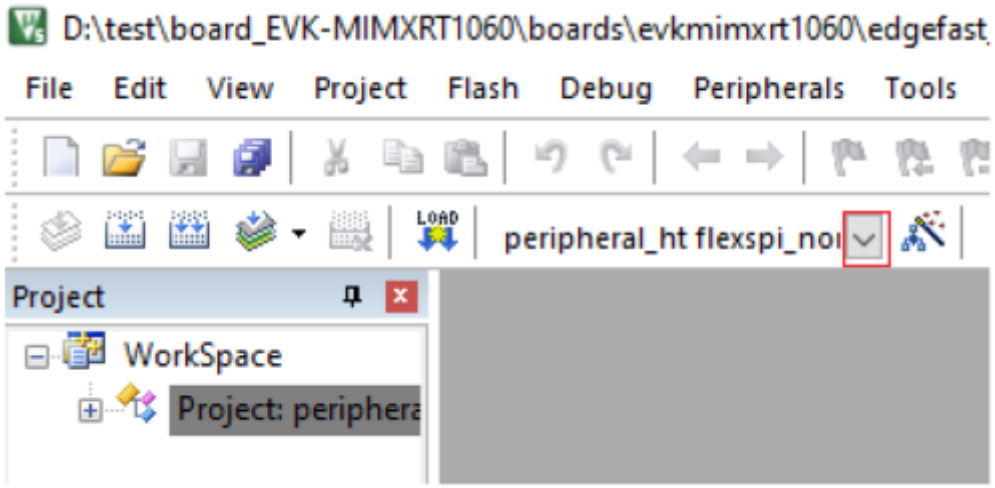
1. The example projects are available at: <root>/boards/evkmimxrt1060/edgefast_bluetooth_examples/peripheral_ht/mdk.
2. Open the mdk workspace file. For example, the highlighted *.uvmpw format file.



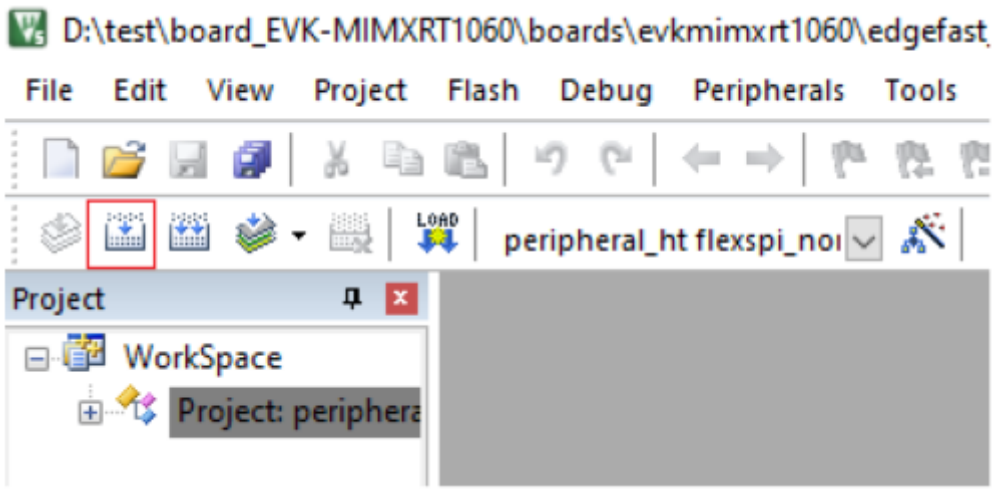
Parent topic: Run a demo application using MDK

Build an MDK example To build an MDK example:

1. Select *flexspi_nor_debug* or *flexspi_nor_release* configurations from the drop-down selector above the project tree in the workspace.



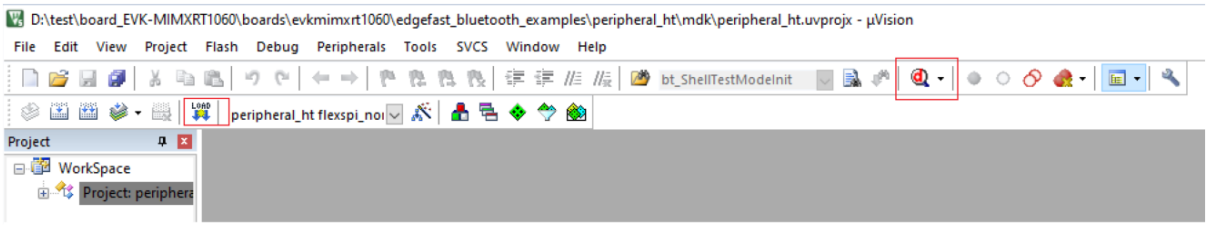
2. Click the highlighted icon to build the EdgeFast Bluetooth Protocol Abstraction Layer project.



Parent topic: Run a demo application using MDK

Run an MDK example For MDK project running, all steps are similar to Run an IAR example except the steps of downloading image from compiler.

To download the MDK image to the board, click the **Debug** button. The executable file downloads to the board.



Parent topic: Run a demo application using MDK

Parent topic:*Demo*

Run a demo application using Arm GCC This document uses peripheral_ht example to describe the steps to open a project, build an example, and run a project on MDK.

For details, see the related section in *Getting Started with MCUXpresso SDK User's Guide* (document: MCUXSDKGSUG) at *root/docs/* and the readme file in the corresponding demo's directory.

Setup tool chains See the section "Run a demo using Arm GCC" of getting start document. For example, *Getting Started with MCUXpresso SDK for MIMXRT1160-EVK*.

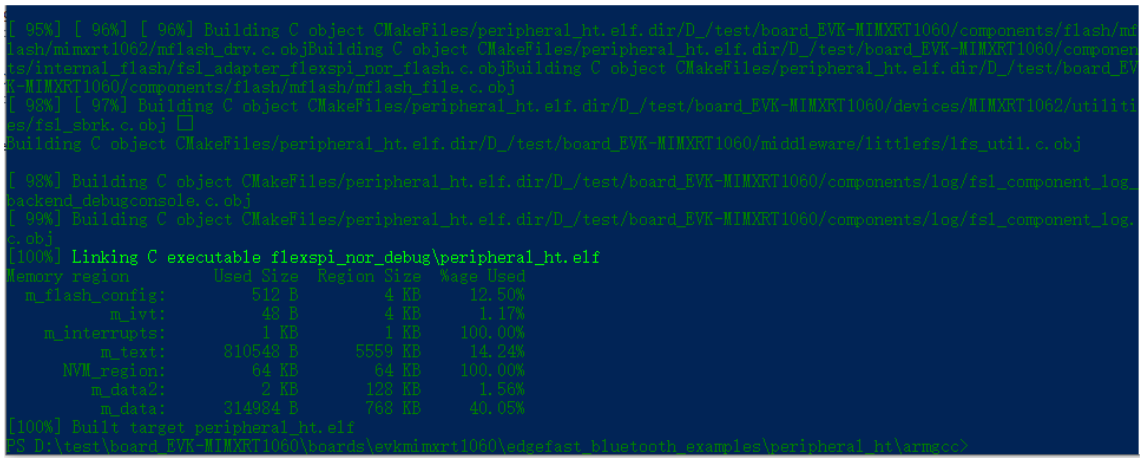
Parent topic: Run a demo application using Arm GCC

Build a GCC example To build a GCC example:

1. Change the directory to the project directory: <install_dir>\boards\evkmimxrt1060\edgefast_bluetooth_examples\peripheral_ht\armgcc.
2. Run the build script.

   For windows, the script is build_flexspi_nor_debug.bat/ build_flexspi_nor_release.bat.

   The build output is shown in the following figure.



Parent topic: Run a demo application using Arm GCC

Run a GCC example Refer to the section "Run a demo using Arm GCC" of the getting start document. For example, see Getting Started with MCUXpresso SDK for MIMXRT1060-EVK. The peripheral_ht.elf is the target to download.

Parent topic: Run a demo application using Arm GCC

Parent topic:*Demo*

Download Linker Layer firmware from the reference board Download the Linker Layer (LL) Firmware from Reference board EVKRT1060 by SDIO interface before running the Bluetooth Controller stack. The LL download is necessary because wireless module does not support flash.

Parent topic:*Demo*

Change board-specific parameters There are some board-specific parameters that can be changed in the application layer for EdgeFast BT PAL.

Change HCI UART parameters Since the controller can support different baud rates, the demo provides an interface with configurable baud rates. The function*controller_hci_uart_get_configuration* is used to get HCI UART parameters, including the instance, default baud rate, which depends on the controller, running baud rate which defined by macro BOARD_BT_UART_BAUDRATE and so on. If this function returns '0' and the running baud rate is inconsistent with the default baud rate, EdgeFast BT PAL switches the baud rate of the controller to the running baud rate.

Parent topic: Change board-specific parameters

Change USB Host stack parameters Since the board supports multiple USB ports, the demo provides a configurable interface for USB Host stack. The function*USB_HostGetConfiguration* received the instance of USB for EdgeFast BT PAL. For the case where there is a USBPHY, the demo configures the properties of the PHY through*USB_HostPhyGetConfiguration*.

Note: There are series of hex bytes printed on the console after the wireless module resets. However, it does not impact the EdgeFast BT PAL application running.

Parent topic: Change board-specific parameters

Parent topic:*Demo*

Known issues This section provides a list of known issues in the release package.

Notes This section provides a list of notes to use EdgeFast Bluetooth stack

- the follow configuration items related to resource needs more attention
  - CONFIG_BT_MAX_CONN The max connections that can be created.
  - CONFIG_BT_MAX_PAIRED The max supported paired devices.
  - CONFIG_BT_BUF_EVT_RX_COUNT The max received hci events and acl data packets at one time if the sys work queue task is blocked. One example is: when LE connection is created and HCI_LE_Enhanced_Connection_Complete is received, the sys work queue task is busy with processing the HCI_LE_Enhanced_Connection_Complete. If the received hci events exceed CONFIG_BT_BUF_EVT_RX_COUNT, it may leads potential issue, please increase value of the macro.
- All the EdgeFast Bluetooth API should be called only after EdgeFast Bluetooth is initialized.
- Don't send HCI cmd from the sys work queue task or any stack's callbacks.

CONFIG_BT_BUF_RESERVE

Buffer reserved length, suggested value is 8.

CONFIG_BT_SNOOP

Whether enable bt snoop feature, 0 - disable, 1 - enable.

CONFIG_BT_HCI_CMD_COUNT

Number of HCI command buffers, ranging from 2 to 64. Number of buffers available for HCI commands Range 2 to 64 is valid.

CONFIG_BT_RX_BUF_COUNT

Number of HCI RX buffers, ranging from 2 to 255. Number of buffers available for incoming ACL packets or HCI events from the controller Range 2 to 255 is valid.

CONFIG_BT_RX_BUF_LEN

Maximum supported HCI RX buffer length, ranging from 73 to 2000. Maximum data size for each HCI RX buffer. This size includes everything starting with the ACL or HCI event headers. Note that buffer sizes are always rounded up to the nearest multiple of 4, so if this Kconfig value is something else then there is some wasted space. The minimum of 73 has been taken for LE SC which has an L2CAP MTU of 65 bytes. On top of this, The L2CAP header (4 bytes) and the ACL header (also 4 bytes) which yields 73 bytes. Range is 73 to 2000.

CONFIG_BT_HCI_RESERVE

Reserve buffer size for user. Headroom that the driver needs for sending and receiving buffers. Add a new 'default' entry for each new driver.

CONFIG_BT_DISCARDABLE_BUF_COUNT

Number of discardable event buffers, if the macro is set to 0, disable this feature, if greater than 0, this feature is enabled. Number of buffers in a separate buffer pool for events which the HCI driver considers discardable. Examples of such events could be , for example, Advertising Reports. The benefit of having such a pool means that if there is a heavy inflow of such events it does not cause the allocation for other critical events to block and may even eliminate deadlocks in some cases.

CONFIG_BT_DISCARDABLE_BUF_SIZE

Size of discardable event buffers, ranging from 45 to 257. Size of buffers in the separate discardable event buffer pool. The minimum size is set based on the Advertising Report. Setting the buffer can save memory if with size set differently from that of the CONFIG_BT_RX_BUF_LEN. range is 45 to 257.

CONFIG_BT_HCI_TX_STACK_SIZE

HCI TX task stack size needed for executing bt_send with specified driver, should be no less than 512.

CONFIG_BT_HCI_TX_PRIO

HCI TX task priority.

CONFIG_BT_RX_STACK_SIZE

Size of the receiving thread stack. This is the context from which all event callbacks to the application occur. The default value is sufficient for basic operation, but if the application needs to do advanced things in its callbacks that require extra stack space, this value can be increased to accommodate for that.

CONFIG_BT_RX_PRIO

RX task priority.

CONFIG_BT_PERIPHERAL

Peripheral Role support, if the macro is set to 0, feature is disabled, if 1, feature is enabled. Select this for LE Peripheral role support.

CONFIG_BT_BROADCASTER

Broadcaster Role support, if the macro is set to 0, feature is disabled, if 1, feature is enabled. Select this for LE Broadcaster role support.

CONFIG_BT_EXT_ADV

Extended Advertising and Scanning support [EXPERIMENTAL], if the macro is set to 0, feature is disabled, if 1, feature is enabled. Select this to enable Extended Advertising API support. This enables support for advertising with multiple advertising sets, extended advertising data, and advertising on LE Coded PHY. It enables support for receiving extended advertising data as a scanner, including support for advertising data over the LE coded PHY. It enables establishing connections over LE Coded PHY.

CONFIG_BT_CENTRAL

Central Role support, if the macro is set to 0, feature is disabled, if 1, feature is enabled. Select this for LE Central role support.

CONFIG_BT_WHITELIST

Enable whitelist support. This option enables the whitelist API. This takes advantage of the whitelisting feature of a Bluetooth LE controller. The whitelist is a global list and the same whitelist is used by both scanner and advertiser. The whitelist cannot be modified while it is in use. An Advertiser can whitelist which peers can connect or request scan response data. A scanner can whitelist advertiser for which it generates advertising reports. Connections can be established automatically for whitelisted peers.

This option deprecates the bt_le_set_auto_conn API in favor of the bt_conn_create_aute_le API.

CONFIG_BT_DEVICE_NAME

Bluetooth device name. Name can be up to 248 bytes long (excluding NULL termination). Can be empty string.

CONFIG_BT_DEVICE_APPEARANCE

Bluetooth device appearance. For the list of possible values, see the link: www.bluetooth.com/specifications/assigned-numbers.

CONFIG_BT_DEVICE_NAME_DYNAMIC

Allow to set Bluetooth device name on runtime. Enabling this option allows for runtime configuration of Bluetooth device name.

CONFIG_BT_ID_MAX

Maximum number of local identities, range 1 to 10 is valid. Maximum number of supported local identity addresses. For most products, this is safe to leave as the default value (1). Range 1 to 10 is valid.

CONFIG_BT_CONN

Connection enablement, if the macro is set to 0, feature is disabled, if 1, feature is enabled.

CONFIG_BT_MAX_CONN

it is the max connection supported by host stack. Maximum number of simultaneous Bluetooth connections supported.

CONFIG_BT_HCI_ACL_FLOW_CONTROL

Controller to host ACL flow control support. Enable support for throttling ACL buffers from the controller to the host. This is useful when the host and controller are on separate cores, since it ensures that we do not run out of incoming ACL buffers.

CONFIG_BT_PHY_UPDATE

PHY Update, if the macro is set to 0, feature is disabled, if 1, feature is enabled. Enable support for Bluetooth 5.0 PHY Update Procedure.

CONFIG_BT_DATA_LEN_UPDATE

Data Length Update. If the macro is set to 0, feature is disabled, if 1, feature is enabled. Enable support for Bluetooth v4.2 LE Data Length Update procedure.

CONFIG_BT_CREATE_CONN_TIMEOUT

Timeout for pending LE Create Connection command in seconds.

CONFIG_BT_CONN_PARAM_UPDATE_TIMEOUT

Peripheral connection parameter update timeout in milliseconds, range 1 to 65535 is valid. The value is a timeout used by peripheral device to wait until it starts the connection parameters update procedure to change default connection parameters. The default value is set to 5s, to comply with BT protocol specification: Core 4.2 Vol 3, Part C, 9.3.12.2 Range 1 to 65535 is valid.

CONFIG_BT_CONN_TX_MAX

Maximum number of pending TX buffers. Maximum number of pending TX buffers that have not yet been acknowledged by the controller.

CONFIG_BT_REMOTE_INFO

Enable application access to remote information. Enable application access to the remote information available in the stack. The remote information is retrieved once a connection has been established and the application is notified when this information is available through the remote_version_available connection callback.

CONFIG_BT_REMOTE_VERSION

Enable fetching of remote version. Enable this to get access to the remote version in the Controller and in the host through bt_conn_get_info(). The fields in question can be then found in the bt_conn_info struct.

CONFIG_BT_SMP_SC_ONLY

Secure Connections Only Mode. This option enables support for Secure Connection Only Mode. In this mode device shall only use Security Mode 1 Level 4 with exception for services that only require Security Mode 1 Level 1 (no security). Security Mode 1 Level 4 stands for authenticated LE Secure Connections pairing with encryption. Enabling this option disables legacy pairing.

CONFIG_BT_SMP_OOB_LEGACY_PAIR_ONLY

Force Out of Band Legacy pairing. This option disables Legacy and LE SC pairing and forces legacy OOB.

CONFIG_BT_SMP_DISABLE_LEGACY_JW_PASSKEY

Forbid usage of insecure legacy pairing methods. This option disables Just Works and Passkey legacy pairing methods to increase security.

CONFIG_BT_PRIVACY

Privacy Feature, if the macro is set to 0, feature is disabled, if 1, feature is enabled. Enable local Privacy Feature support. This makes it possible to use Resolvable Private Addresses (RPAs).

CONFIG_BT_ECC

Enable ECDH key generation support. This option adds support for ECDH HCI commands.

CONFIG_BT_TINYCRYPT_ECC

Use TinyCrypt library for ECDH. If this option is used to set TinyCrypt library which is used for emulating the ECDH HCI commands and events needed by e.g. LE Secure Connections. In builds including the Bluetooth LE host, if don't set the controller crypto which is used for ECDH and if the controller doesn't support the required HCI commands the LE Secure Connections support will be disabled. In builds including the HCI Raw interface and the Bluetooth LE controller, this

option injects support for the 2 HCI commands required for LE Secure Connections so that hosts can make use of those. The option defaults to enabled for a combined build with Zephyr's own controller, since it does not have any special ECC support itself (at least not currently).

CONFIG_BT_TINYCRYPT_ECC_PRIORITY

Thread priority of ECC Task.

CONFIG_BT_HCI_ECC_STACK_SIZE

Thread stack size of ECC Task.

CONFIG_BT_RPA

Bluetooth Resolvable Private Address (RPA)

CONFIG_BT_RPA_TIMEOUT

Resolvable Private Address timeout, defaults to 900 seconds. This option defines how often resolvable private address is rotated. Value is provided in seconds and defaults to 900 seconds (15 minutes).

CONFIG_BT_SIGNING

Data signing support, if the macro is set to 0, feature is disabled, if 1, feature is enabled. This option enables data signing which is used for transferring authenticated data in an unencrypted connection.

CONFIG_BT_SMP_APP_PAIRING_ACCEPT

Accept or reject pairing initiative. When receiving pairing request or pairing response queries, the application shall either accept proceeding with pairing or not. This is for pairing over SMP and does not affect SSP, which will continue pairing without querying the application. The application can return an error code, which is translated into an SMP return value if the pairing is not allowed.

CONFIG_BT_SMP_ALLOW_UNAUTH_OVERWRITE

Allow unauthenticated pairing for paired device. This option allows all unauthenticated pairing attempts made by the peer where an unauthenticated bond already exists. This would enable cases where an attacker could copy the peer device address to connect and start an unauthenticated pairing procedure to replace the existing bond. When this option is disabled in order to create a new bond the old bond must be explicitly deleted with bt_unpair.

CONFIG_BT_FIXED_PASSKEY

Use a fixed passkey for pairing, set passkey to fixed or not. With this option enabled, the application will be able to call the bt_passkey_set() API to set a fixed passkey. If set, the pairing_confim() callback will be called for all incoming pairings.

CONFIG_BT_BONDABLE

Bondable Mode, if the macro is set to 0, feature is disabled, if 1, feature is enabled. This option enables support for Bondable Mode. In this mode, Bonding flag in AuthReq of SMP Pairing Request/Response is set indicating the support for this mode.

CONFIG_BT_BONDING_REQUIRED

Always require bonding. When this option is enabled remote devices are required to always set the bondable flag in their pairing request. Any other kind of requests will be rejected.

CONFIG_BT_SMP_ENFORCE_MITM

Enforce MITM protection, if the macro is set to 0, feature is disabled, if 1, feature is enabled. With this option enabled, the Security Manager is set MITM option in the Authentication Requirements Flags whenever local IO Capabilities allow the generated key to be authenticated.

CONFIG_BT_OOB_DATA_FIXED

Use a fixed random number for LESC OOB pairing. With this option enabled, the application will be able to perform LESC pairing with OOB data that consists of fixed random number and confirm value. This option should only be enabled for debugging and should never be used in production.

CONFIG_BT_KEYS_OVERWRITE_OLDEST

Overwrite oldest keys with new ones if key storage is full. With this option enabled, if a pairing attempt occurs and the key storage is full, then the oldest keys in storage will be removed to free space for the new pairing keys.

CONFIG_BT_HOST_CCM

Enable host side AES-CCM module. Enables the software-based AES-CCM engine in the host. Will use the controller's AES encryption functions if available, or BT_HOST_CRYPTO otherwise.

CONFIG_BT_L2CAP_RX_MTU

Maximum supported L2CAP MTU for incoming data, if CONFIG_BT_SMP is set, range is 65 to 1300, otherwise range is 23 to 1300. Maximum size of each incoming L2CAP PDU. Range is 23 to 1300 range is 65 to 1300 for CONFIG_BT_SMP.

CONFIG_BT_L2CAP_TX_BUF_COUNT

Number of buffers available for outgoing L2CAP packets, ranging from 2 to 255. Range is 2 to 255.

CONFIG_BT_L2CAP_TX_FRAG_COUNT

Number of L2CAP TX fragment buffers, ranging from 0 to 255. Number of buffers available for fragments of TX buffers.

Warning: Setting this to 0 means that the application must ensure that queued TX buffers never need to be fragmented, that is the controller's buffer size is large enough. If this is not ensured, and there are no dedicated fragment buffers, a deadlock may occur. In most cases the default value of 2 is a safe bet. Range is 0 to 255.

CONFIG_BT_L2CAP_TX_MTU

Maximum supported L2CAP MTU for L2CAP TX buffers, if CONFIG_BT_SMP is set, the range is 65 to 2000. Otherwise, range is 23 to 2000. Range is 23 to 2000. Range is 65 to 2000 for CONFIG_BT_SMP.

CONFIG_BT_L2CAP_DYNAMIC_CHANNEL

L2CAP Dynamic Channel support. This option enables support for LE Connection oriented Channels, allowing the creation of dynamic L2CAP Channels.

CONFIG_BT_L2CAP_DYNAMIC_CHANNEL

L2CAP Dynamic Channel support. This option enables support for LE Connection oriented Channels, allowing the creation of dynamic L2CAP Channels.

Bluetooth BR/EDR support [EXPERIMENTAL] This option enables Bluetooth BR/EDR support.

CONFIG_BT_ATT_PREPARE_COUNT

Number of ATT prepares write buffers, if the macro is set to 0, feature is disabled, if greater than 1, feature is enabled. Number of buffers available for ATT prepares write, setting this to 0 disables GATT long/reliable writes.

CONFIG_BT_ATT_TX_MAX

Maximum number of queued outgoing ATT PDUs. Number of ATT PDUs that can be at a single moment queued for transmission. If the application tries to send more than this amount the calls blocks until an existing queued PDU gets sent. Range is 1 to CONFIG_BT_L2CAP_TX_BUF_COUNT.

CONFIG_BT_GATT_SERVICE_CHANGED

GATT Service Changed support, if the macro is set to 0, feature is disabled, if 1, feature is enabled. This option enables support for the service changed characteristic.

CONFIG_BT_GATT_DYNAMIC_DB

GATT dynamic database support, if the macro is set to 0, feature is disabled, if 1, feature is enabled. This option enables registering/unregistering services at runtime.

CONFIG_BT_GATT_CACHING

GATT Caching support, if the macro is set to 0, feature is disabled, if 1, feature is enabled. This option enables support for GATT Caching. When enabled the stack registers Client Supported Features and Database Hash characteristics which is used by clients to detect if anything has changed on the GATT database.

CONFIG_BT_GATT_CLIENT

GATT client support, if the macro is set to 0, feature is disabled, if 1, feature is enabled. This option enables support for the GATT Client role.

CONFIG_BT_GATT_READ_MULTIPLE

GATT Read Multiple Characteristic. Values support, if the macro is set to 0, feature is disabled, if 1, feature is enabled. This option enables support for the GATT Read Multiple Characteristic Values procedure.

CONFIG_BT_GAP_AUTO_UPDATE_CONN_PARAMS

Automatic Update of Connection Parameters, if the macro is set to 0, feature is disabled, if 1, feature is enabled. This option, if enabled, allows automatically sending request for connection parameters update after GAP recommended 5 seconds of connection as peripheral.

CONFIG_BT_GAP_PERIPHERAL_PREF_PARAMS

Configure peripheral preferred connection parameters. This configures peripheral preferred connection parameters. Enabling this option results in adding PPCP characteristic in GAP. If disabled it is up to application to set expected connection parameters.

CONFIG_BT_MAX_PAIRED

Maximum number of paired devices. Maximum number of paired Bluetooth devices. The minimum (and default) number is 1.

CONFIG_BT_MAX_SCO_CONN

Maximum number of simultaneous SCO connections. Maximum number of simultaneous Bluetooth synchronous connections supported. The minimum (and default) number is 1. Range 1 to 3 is valid.

CONFIG_BT_RFCOMM

Bluetooth RFCOMM protocol support [EXPERIMENTAL], if the macro is set to 0, feature is disabled, if 1, feature is enabled. This option enables Bluetooth RFCOMM support.

CONFIG_BT_RFCOMM_L2CAP_MTU

L2CAP MTU for RFCOMM frames. Maximum size of L2CAP PDU for RFCOMM frames.

CONFIG_BT_HFP_HF

Bluetooth Handsfree profile HF Role support [EXPERIMENTAL], if the macro is set to 0, feature is disabled, if 1, feature is enabled. This option enables Bluetooth HF support.

CONFIG_BT_AVDTP

Bluetooth AVDTP protocol support [EXPERIMENTAL], if the macro is set to 0, feature is disabled, if 1, feature is enabled. This option enables Bluetooth AVDTP support.

CONFIG_BT_A2DP

Bluetooth A2DP Profile [EXPERIMENTAL]. This option enables the A2DP profile.

CONFIG_BT_A2DP_SOURCE

Bluetooth A2DP profile source function. This option enables the A2DP profile Source function.

CONFIG_BT_A2DP_SINK

Bluetooth A2DP profile sink function. This option enables the A2DP profile Sink function.

CONFIG_BT_A2DP_TASK_PRIORITY

Bluetooth A2DP profile task priority. This option sets the task priority. The task is used to process the streamer data and retry command.

CONFIG_BT_A2DP_TASK_STACK_SIZE

Bluetooth A2DP profile task stack size. This option sets the task stack size.

CONFIG_BT_PAGE_TIMEOUT

Bluetooth Page Timeout. This option sets the page timeout value. Value is selected as (N * 0.625) ms.

CONFIG_BT_DIS_MODEL

Model name. The device model inside Device Information Service.

CONFIG_BT_DIS_MANUF

Manufacturer name. The device manufacturer inside Device Information Service.

CONFIG_BT_DIS_PNP

Enable PnP_ID characteristic. Enable PnP_ID characteristic in Device Information Service.

CONFIG_BT_DIS_PNP_VID_SRC

Vendor ID source, range 1 - 2. The Vendor ID Source field designates which organization assigned the value used in the Vendor ID field value. The possible values are:

- 1 Bluetooth SIG, the Vendor ID was assigned by the Bluetooth SIG
- 2 USB IF, the Vendor ID was assigned by the USB IF

CONFIG_BT_DIS_PNP_VID

Vendor ID, range 0 - 0xFFFF. The Vendor ID field is intended to uniquely identify the vendor of the device. This field is used in conjunction with Vendor ID Source field, which determines which organization assigned the Vendor ID field value. Note: The Bluetooth Special Interest Group assigns Device ID Vendor ID, and the USB Implementers Forum assigns Vendor IDs, either of which can be used for the Vendor ID field value. Device providers should procure the Vendor ID from the USB Implementers Forum or the Company Identifier from the Bluetooth SIG.

CONFIG_BT_DIS_PNP_PID

Product ID, range 0 - 0xFFFF. The Product ID field is intended to distinguish between different products made by the vendor identified with the Vendor ID field. The vendors themselves manage Product ID field values.

CONFIG_BT_DIS_PNP_VER

Product Version, range 0 - 0xFFFF. The Product Version field is a numeric expression identifying the device release number in Binary-Coded Decimal. This is a vendor-assigned value, which defines the version of the product identified by the Vendor ID and Product ID fields. This field is intended to differentiate between versions of products with identical Vendor IDs and Product IDs. The value of the field value is 0xJJMN for version JJ.M.N (JJ - major version number, M - minor version number, N - subminor version number); For example, version 2.1.3 is represented with value 0x0213 and version 2.0.0 is represented with a value of 0x0200. When upward-compatible changes are made to the device, it is recommended that the minor version number be incremented. If incompatible changes are made to the device. It is recommended that the major version number is incremented. The subminor version is incremented for bug fixes.

CONFIG_BT_DIS_SERIAL_NUMBER

Enable DIS Serial number characteristic, 1 - enable, 0 - disable. Enable Serial Number characteristic in Device Information Service.

CONFIG_BT_DIS_SERIAL_NUMBER_STR

Serial Number. Serial Number characteristic string in Device Information Service.

CONFIG_BT_DIS_FW_REV

Enable DIS Firmware Revision characteristic, 1 - enable, 0 - disable. Enable Firmware Revision characteristic in Device Information Service.

CONFIG_BT_DIS_FW_REV_STR

Firmware revision. Firmware Revision characteristic String in Device Information Service.

CONFIG_BT_DIS_HW_REV

Enable DIS Hardware Revision characteristic, 1 - enable, 0 - disable. Enable Hardware Revision characteristic in Device Information Service.

CONFIG_BT_DIS_HW_REV_STR

Hardware revision. Hardware Revision characteristic String in Device Information Service.

CONFIG_BT_DIS_SW_REV

Enable DIS Software Revision characteristic, 1 - enable, 0 - disable. Enable Software Revision characteristic in Device Information Service.

CONFIG_BT_DIS_SW_REV_STR

Software revision Software revision characteristic String in Device Information Service.

CONFIG_SYSTEM_WORKQUEUE_STACK_SIZE

System work queue stack size.

CONFIG_SYSTEM_WORKQUEUE_PRIORITY

System work queue priority.

CONFIG_BT_HCI_TRANSPORT_INTERFACE_TYPE

HCI transport interface type.

CONFIG_BT_HCI_TRANSPORT_INTERFACE_INSTANCE

HCI transport interface instance number.

CONFIG_BT_HCI_TRANSPORT_INTERFACE_SPEED

HCI transport interface rate. Configures the interface speed, for example, the default interface is h4, the speed to 115200

CONFIG_BT_HCI_TRANSPORT_TX_THREAD

Whether enable HCI transport TX thread.

CONFIG_BT_HCI_TRANSPORT_RX_THREAD

Whether enable HCI transport RX thread.

CONFIG_BT_HCI_TRANSPORT_RX_STACK_SIZE

HCI transport RX thread stack size.

CONFIG_BT_HCI_TRANSPORT_TX_STACK_SIZE

HCI transport TX thread stack size.

CONFIG_BT_HCI_TRANSPORT_TX_PRIO

HCI transport TX thread priority.

CONFIG_BT_HCI_TRANSPORT_RX_PRIO

HCI transport RX thread priority.

CONFIG_BT_MSG_QUEUE_COUNT

Message number in message queue.

Rework Guide for EdgeFast Bluetooth Protocol Abstraction Layer

Hardware Rework Guide for MIMXRT1170-EVKB and Murata M.2 Module This section is a brief hardware rework guidance of the EdgeFast Bluetooth PAL on the NXP i.MX MIMXRT1170-EVKB and the Murata's 1XK, 1ZM, 2EL or 2LL solution - direct M.2 connection to Embedded Artists EAR00385 (1XK), EAR00364 (1ZM), Rev-A1 (2EL) or EAR00500 (2LL) M.2 modules.

The hardware rework has two parts:

- HCI UART rework
- PCM interface rework

Hardware rework

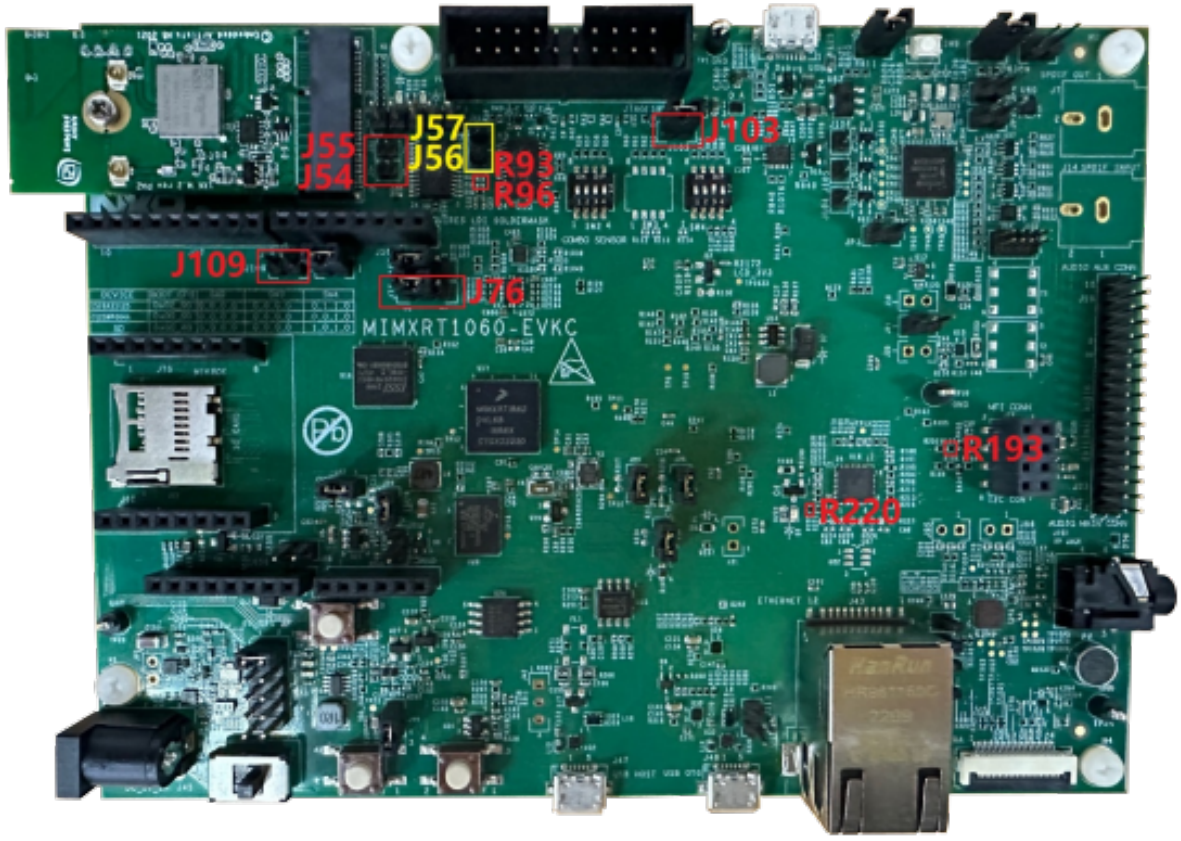
- **HCI UART rework**

1. Mount R93, R96.
2. Remove R193.
3. Connect J109, connect J76 2-3.

- **PCM interface rework**

1. Remove J54 and J55, connect J56 and J57.
2. Remove R220.
3. Connect J103.

Note: When J103 is connected, flash cannot be downloaded. So, remove the connection when downloading flash and reconnect it after downloading.



Parent topic:*Hardware Rework Guide for MIMXRT1060-EVKC and Murata M.2 Module*

Hardware Rework Guide for MIMXRT1170-EVKB and Murata 2EL M.2 Module Hardware Rework Guide for MIMXRT1170-EVKB and Murata

2EL M.2 Module

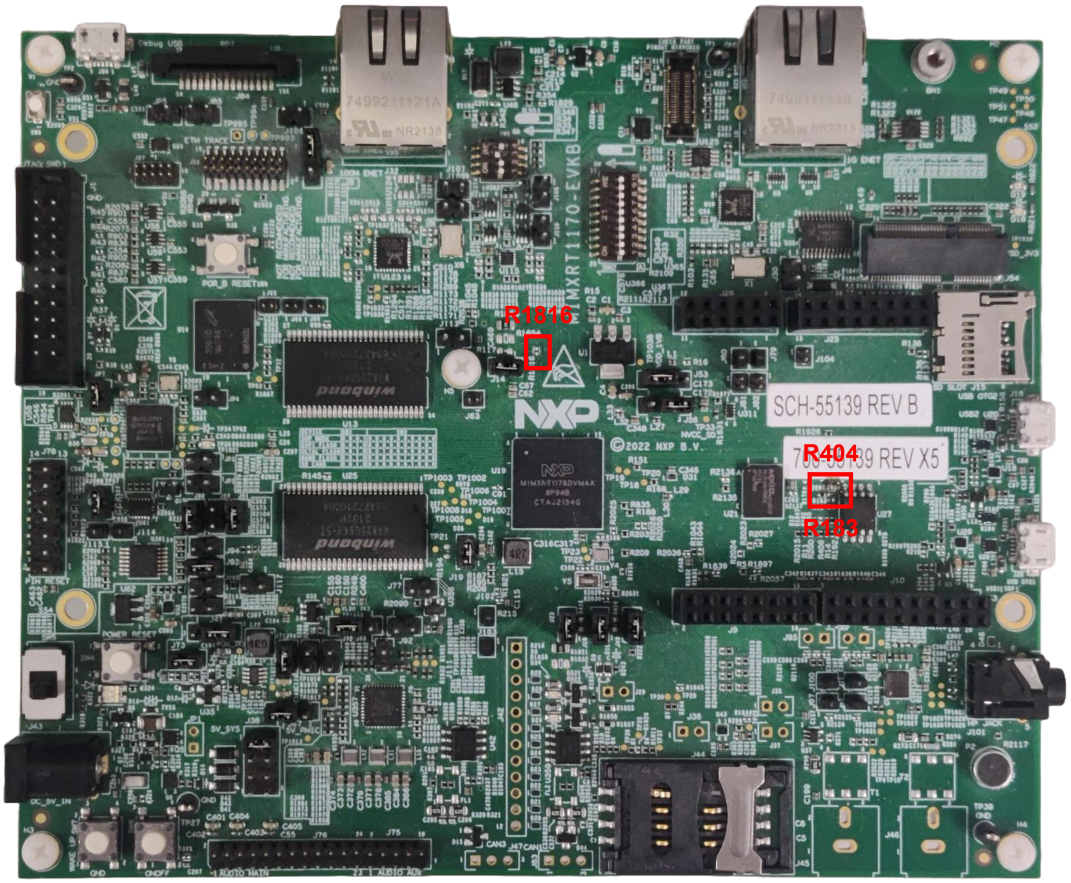
This section is a brief hardware rework guidance of the EdgeFast Bluetooth PAL on the NXP i.MX MIMXRT1170-EVKB and the Murata 2EL M.2 solution - direct M.2 connection to Embedded Artists' Rev-A1 (2EL) M.2 modules.

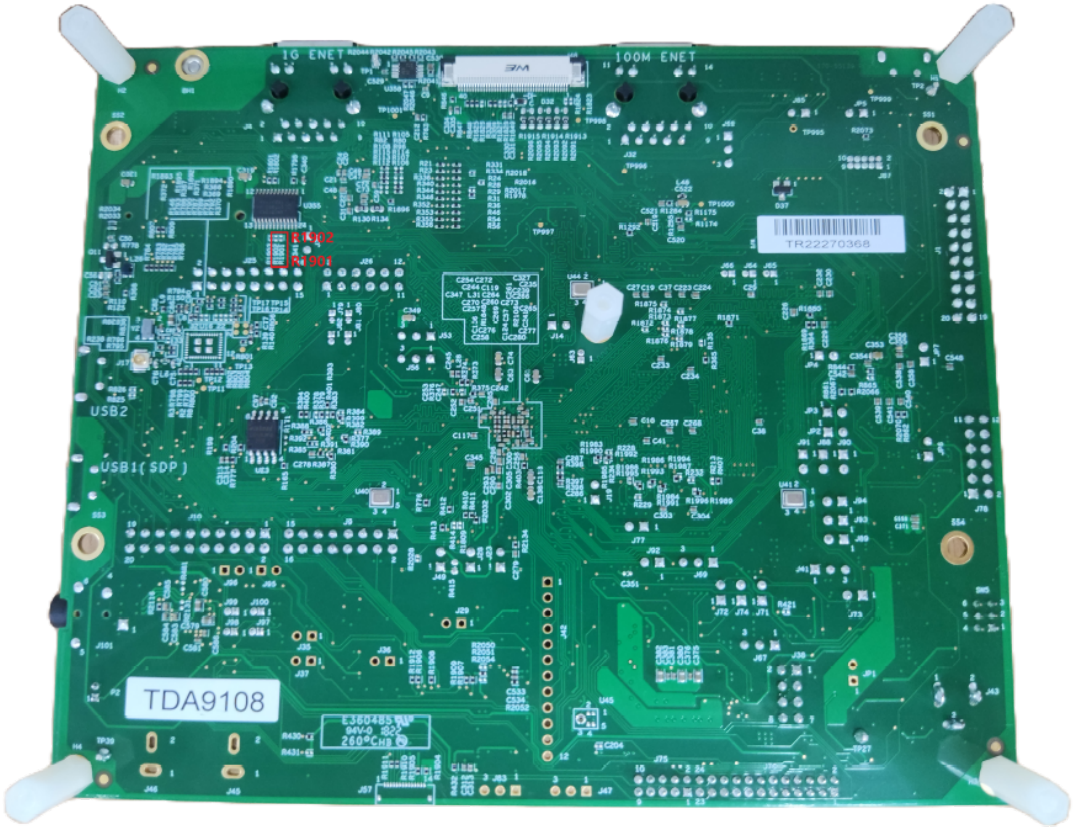
The hardware rework has three parts:

- HCI UART rework
- PCM interface rework
- LE Audio Synchronization interface rework (only used on sink side)

Hardware rework

- **HCI UART rework**
  1. Remove resistors R183 and R1816.
  2. Solder 0 ohm resistor to R404, R1901, and R1902.

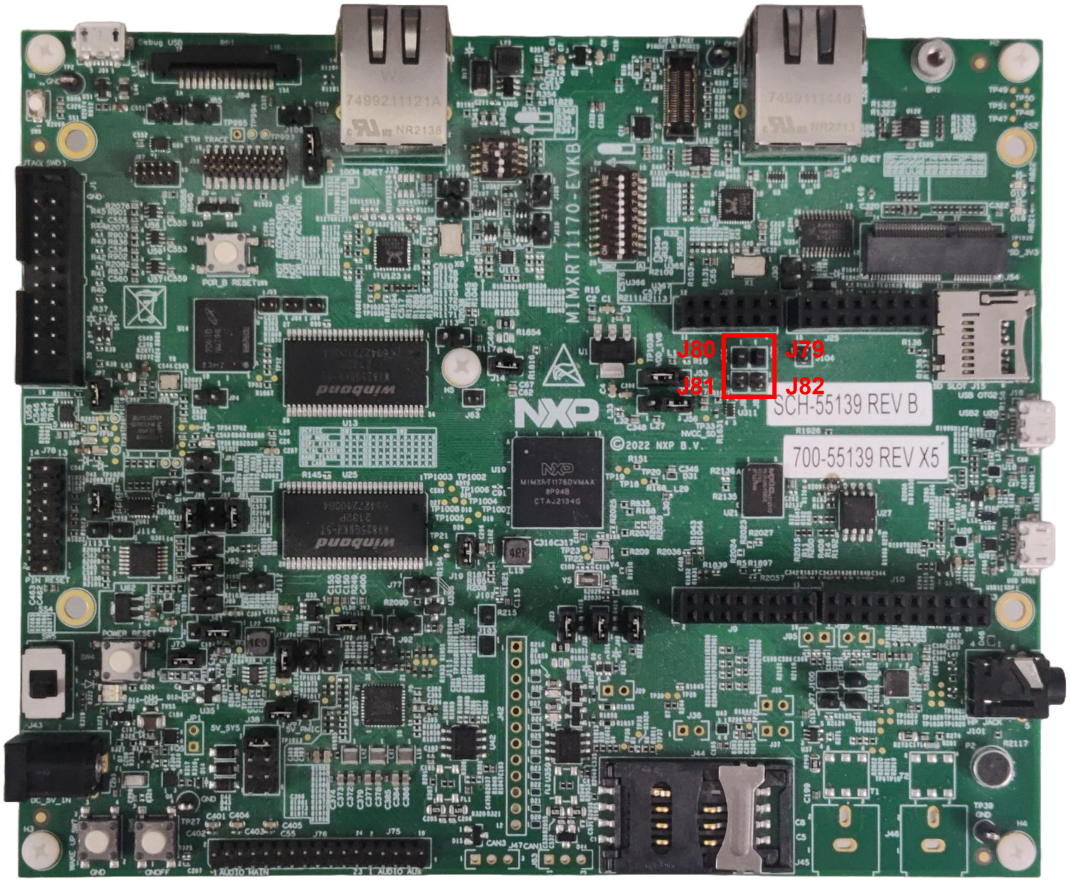


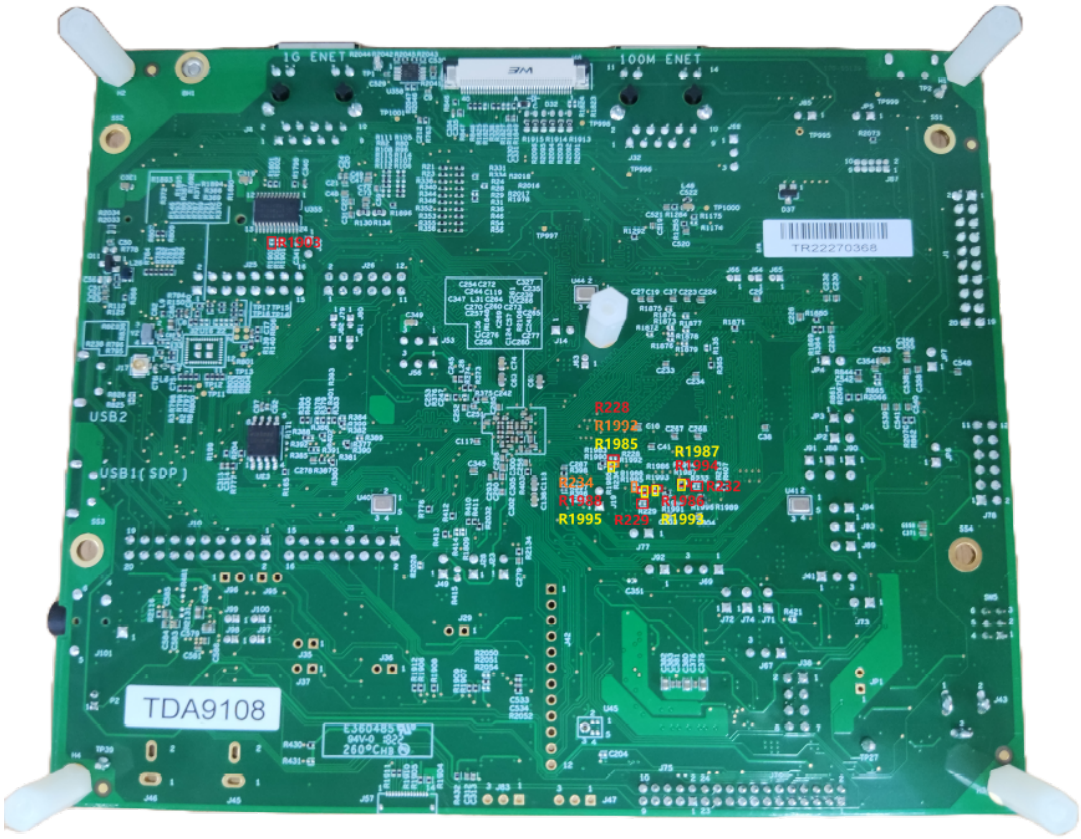


- **PCM interface rework**

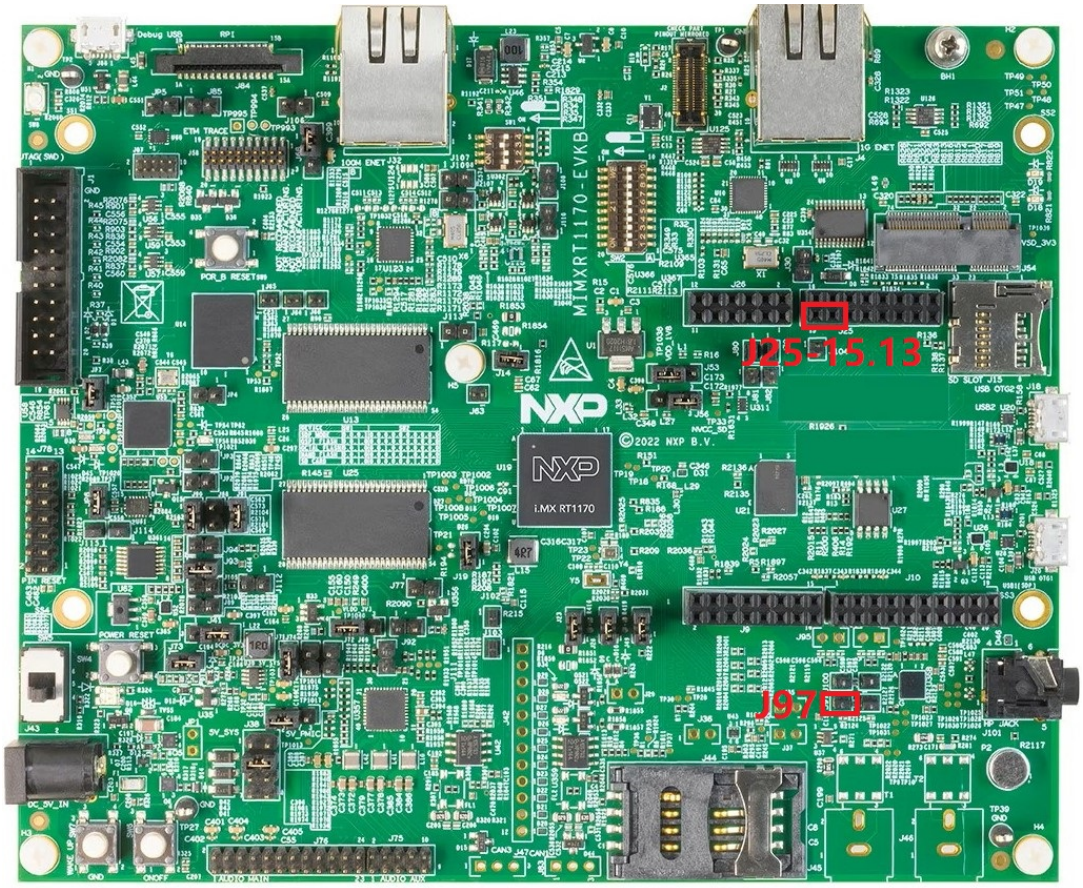
  1. Disconnect header J79 and J80.

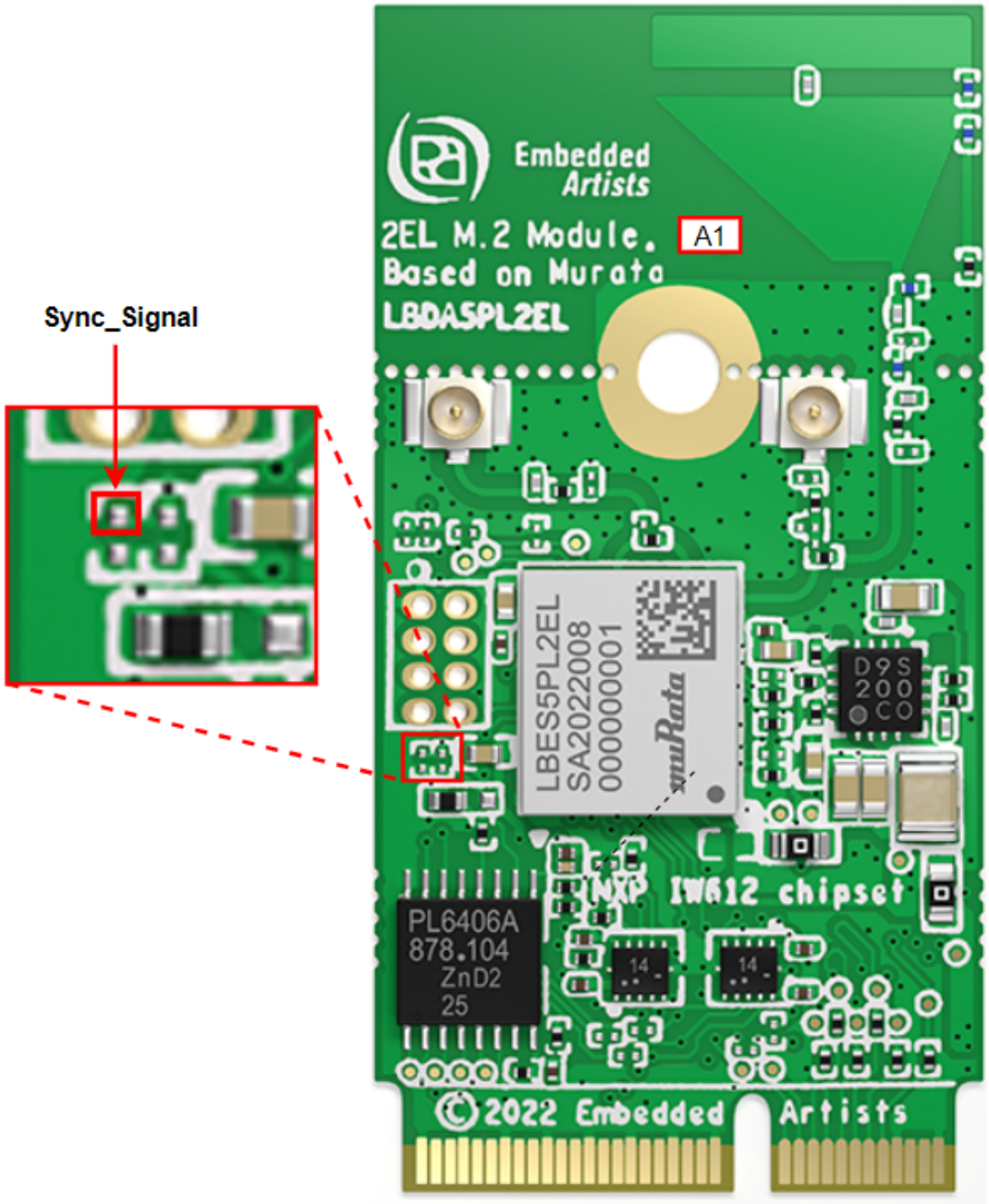
2. Connect header J81 and J82.
3. Remove resistors R1985, R1986, R1987, R1988, R1992, R1993, R1994, and R1995.
4. Solder 0 ohm resistor to R228, R229, R232, R234, and R1903.





- LE Audio Synchronization interface rework (only used on sink side)
  1. Connect J25-15 with J97.
  2. Connect J25-13 with 2EL's GPIO_27


J25 15.13
J97
MIMXRT1170-EVKB
i.MX RT1170



Parent topic:*Hardware Rework Guide for MIMXRT1170-EVKB and Murata 2EL M.2 Adapter*

Hardware Rework Guide for MIMXRT685-EVK and AW-AM457-uSD This section is a brief hardware rework guidance of the EdgeFast Bluetooth PAL on the NXP i.MX MIMXRT685-EVK board and AW-AM457-uSD. The AW-AM457-uSD user guide is available here.The hardware rework has one part:

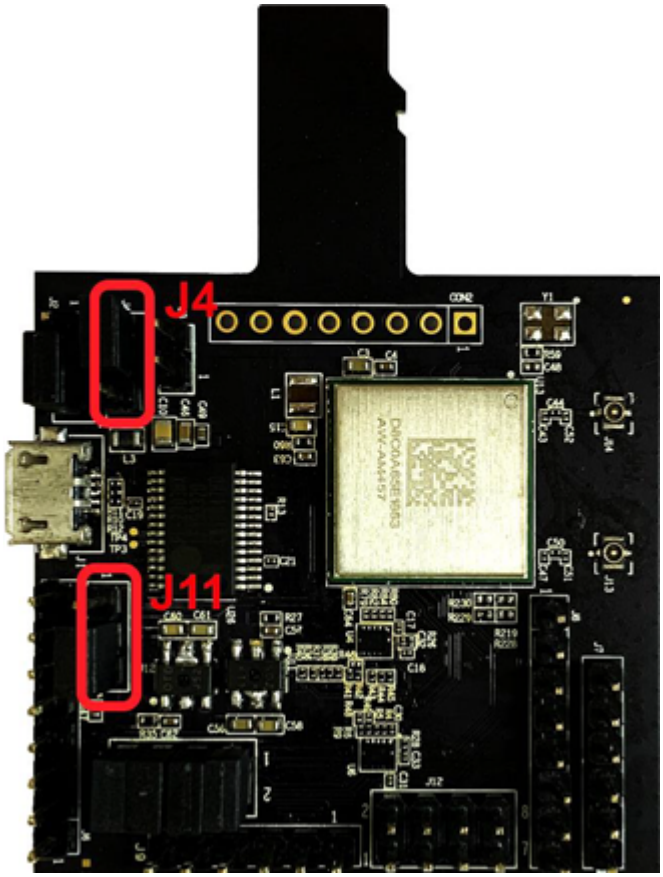
- HCI UART rework

Hardware rework **HCI UART rework**

- R398 move from 1-2 to 2-3
- JP12 2-3
- Connect the pins of two boards as the following table.

Pin Name	AW-AM457-uSD	i.MX RT685	PIN NAME	GPIONAME of i.MX RT685
UART_TXD	J10 (pin 4)	J27 (pin 1)	US-ART4_RXD	FC4_RXD_SDA_MOSI_DATA
UART_RXD	J10 (pin 2)	J27 (pin 2)	USART4_TXD	FC4_TXD_SCL_MISO_WS
UART_RTS	J10 (pin 6)	J47 (pin 9)	USART4_CTS	FC4_CTS_SDA_SSEL0
UART_CTS	J10 (pin 8)	J27 (pin 5)	USART4_RTS	FC4_RTS_SCL_SSEL1
GND	J6 (pin 7)	J29 (pin 6)	GND	GND





Jumper Settings:

- Connect J4[2-3] for VIO 3.3 V supply
- Connect J11[2-3] for VIO_SD 3.3 V supply

PCM interface rework

Connect the pins of two boards as the following table.

Pin Name	AW-AM457-uSD	i.MX RT685	PIN NAME of I.MX RT685	GPIONAME of I.MX RT685
PCM_IN	J9 (pin 1)	J47 (pin 7)	I2S2_TXD	FC2_RXD_SDA_MOSI_DATA
PCM_OUT	J9 (pin 2)	J28 (pin 4)	I2S5_RXD	FC5_RXD_SDA_MOSI_DATA
PCM_SYNC	J9 (pin 3)	J28 (pin 5)	I2S5_WS	FC5_TXD_SCL_MISO_WS
PCM_CLK	J9 (pin 4)	J28 (pin 6)	I2S5_SCK	FC5_SCK
GND	J9 (pin 6)	J29 (pin 7)	GND	GND

Parent topic:*Hardware Rework Guide for MIMXRT685-EVK and AW-AM457-uSD*

Hardware Rework Guide for MIMXRT685-EVK and AW-CM358-uSD This section is a brief hardware rework guidance of the EdgeFast Bluetooth PAL on the NXP i.MX MIMXRT685-EVK board and AW-CM358-uSD. The AW-CM358-uSD user guide is available here. The hardware rework has one part:

- HCI UART rework

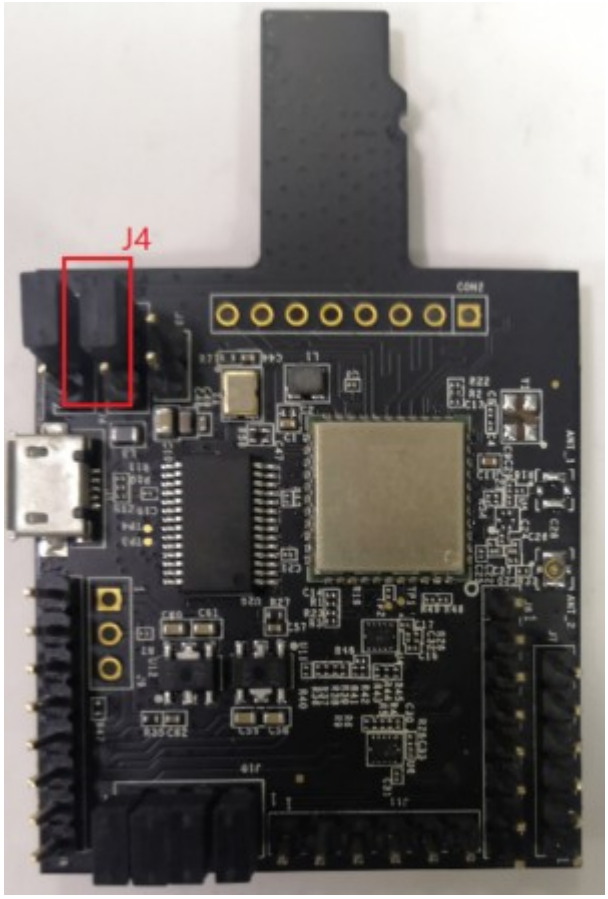
Hardware rework **HCI UART rework**

R398 move from 1-2 to 2-3.

Connect the pins of two boards as the following table.

Pin Name	AW-CM358-USD	i.MXRT685	PIN NAME	GPIONAME of RT685
UART_TXD	J10 (pin 4)	J27 (pin 1)	USART4_RXD	FC4_RXD_SDA_MOSI_DATA
UART_RXD	J10 (pin 2)	J27 (pin 2)	USART4_TXD	FC4_TXD_SCL_MISO_WS
UART_RTS	J10 (pin 6)	J47 (pin 9)	USART4_CTS	FC4_CTS_SDA_SSEL0
UART_CTS	J10 (pin 8)	J27 (pin 5)	USART4_RTS	FC4_RTS_SCL_SSEL1
GND	J6 (pin 7)	J29 (pin 6)	GND	GND





Jumper Setting:

Connect J4[1-2] for VIO 1.8 V supply.

PCM interface rework

Connect the pins of two boards as the following table.

Pin Name	AW-CM358-USD	i.MX RT685	PIN NAME of RT685	GPIONAME of RT685
PCM_IN	J11 (pin 1)	J47 (pin 7)	I2S2_TXD	FC2_RXD_SDA_MOSI_DATA
PCM_OUT	J11 (pin 2)	J28 (pin 4)	I2S5_RXD	FC5_RXD_SDA_MOSI_DATA
PCM_SYNC	J11 (pin 3)	J28 (pin 5)	I2S5_WS	FC5_TXD_SCL_MISO_WS
PCM_CLK	J11 (pin 4)	J28 (pin 6)	I2S5_SCK	FC5_SCK
GND	J11 (pin 5)	J29 (pin 7)	GND	GND

Parent topic:*Hardware Rework Guide for MIMXRT685-EVK and AW-CM358-uSD*

Hardware Rework Guide for MIMXRT685-EVK and AW-AM510-uSD This section is a brief hardware rework guidance of the EdgeFast Bluetooth PAL on the NXP i.MX MIMXRT685-EVK board and AW-AM510-uSD. The AW-AM510-uSD user guide is available here. The hardware rework has one part:

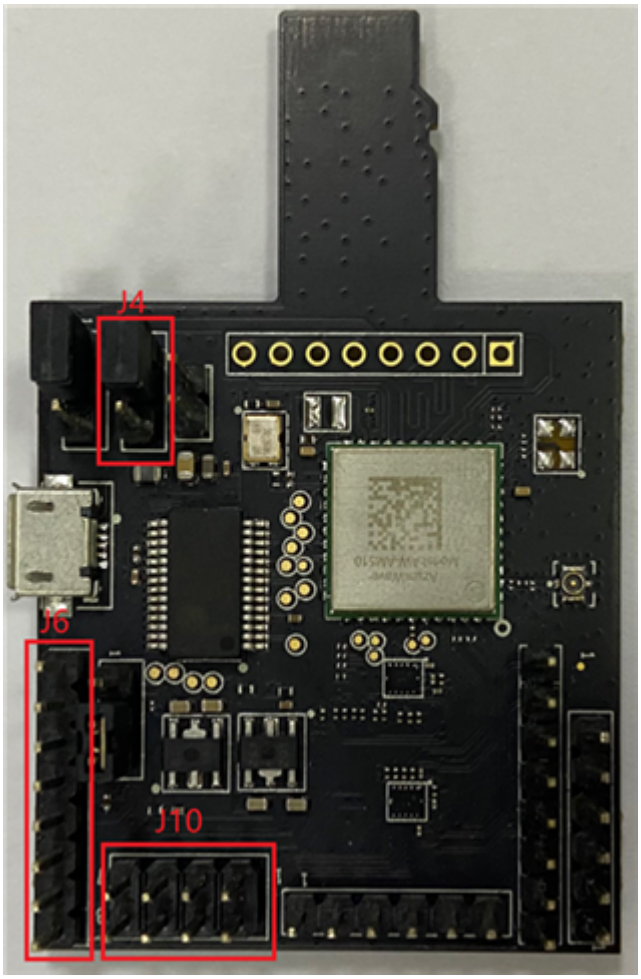
- HCI UART rework

Hardware rework

- **HCI UART rework**

Connect the pins of two boards as the following table.

Pin Name	AW-AM510-uSD	i.MXRT685	PIN NAME	GPIO NAME of RT685
UART_TXD	J10 (pin 4)	J27 (pin 1)	USART4_RXD	FC4_RXD_SDA_MOSI_DATA
UART_RXD	J10 (pin 2)	J27 (pin 2)	USART4_TXD	FC4_TXD_SCL_MISO_WS
UART_RTS	J10 (pin 6)	J47 (pin 9)	USART4_CTS	FC4_CTS_SDA_SSEL0
UART_CTS	J10 (pin 8)	J27 (pin 5)	USART4_RTS	FC4_RTS_SCL_SSEL1
GND	J6 (pin 7)	J29 (pin 6)	GND	GND



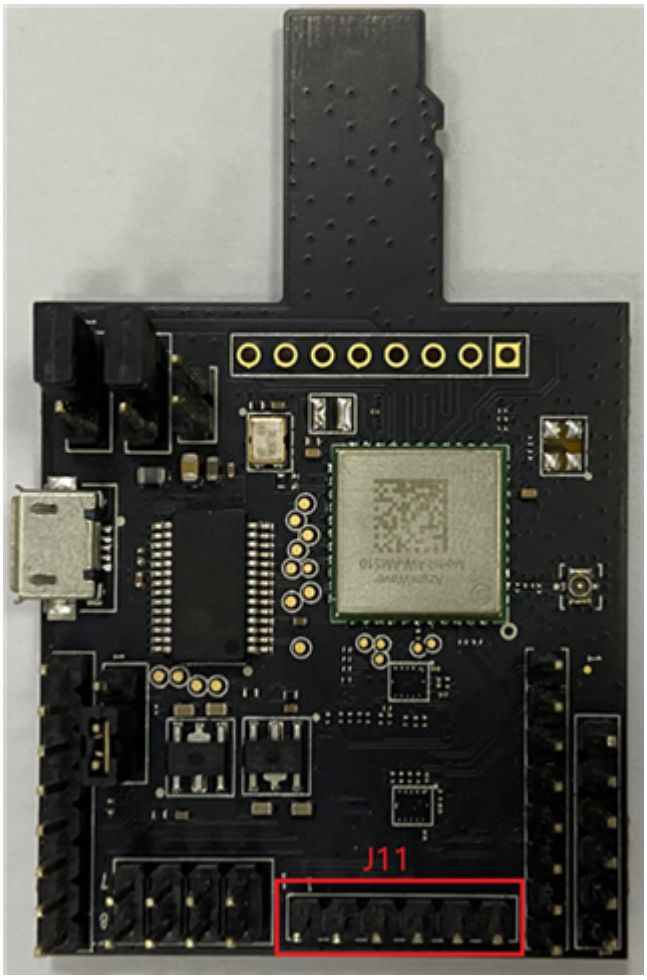
Jumper Setting:

– Connect J4[2-3] for VIO 3.3 V supply

- **PCM interface rework**

Connect the pins of two boards as the following table.

PIN NAME	AW-AM510-USD	i.MX RT685	PIN NAME of RT685	GPIONAME of RT685
PCM_IN	J11 (pin 1)	J47 (pin 7)	I2S2_TXD	FC2_RXD_SDA_MOSI_DATA
PCM_OUT	J11 (pin 2)	J28 (pin 4)	I2S5_RXD	FC5_RXD_SDA_MOSI_DATA
PCM_SYNC	J11 (pin 3)	J28 (pin 5)	I2S5_WS	FC5_TXD_SCL_MISO_WS
PCM_CLK	J11 (pin 4)	J28 (pin 6)	I2S5_SCK	FC5_SCK
GND	J11 (pin 6)	J29 (pin 7)	GND	GND



Parent topic:*Hardware Rework Guide for MIMXRT685-EVK and AW-AM510-uSD*

Hardware Rework Guide for MIMXRT685-EVK and Murata uSD-M.2 Adapter This section is a brief hardware rework guidance of the EdgeFast Bluetooth PAL on the NXP i.MX MIMXRT685-EVK board and the Murata uSD-M.2 adapter. For details on the Murata uSD-M.2 Adapter, see Murata's uSD-M.2 webpage.

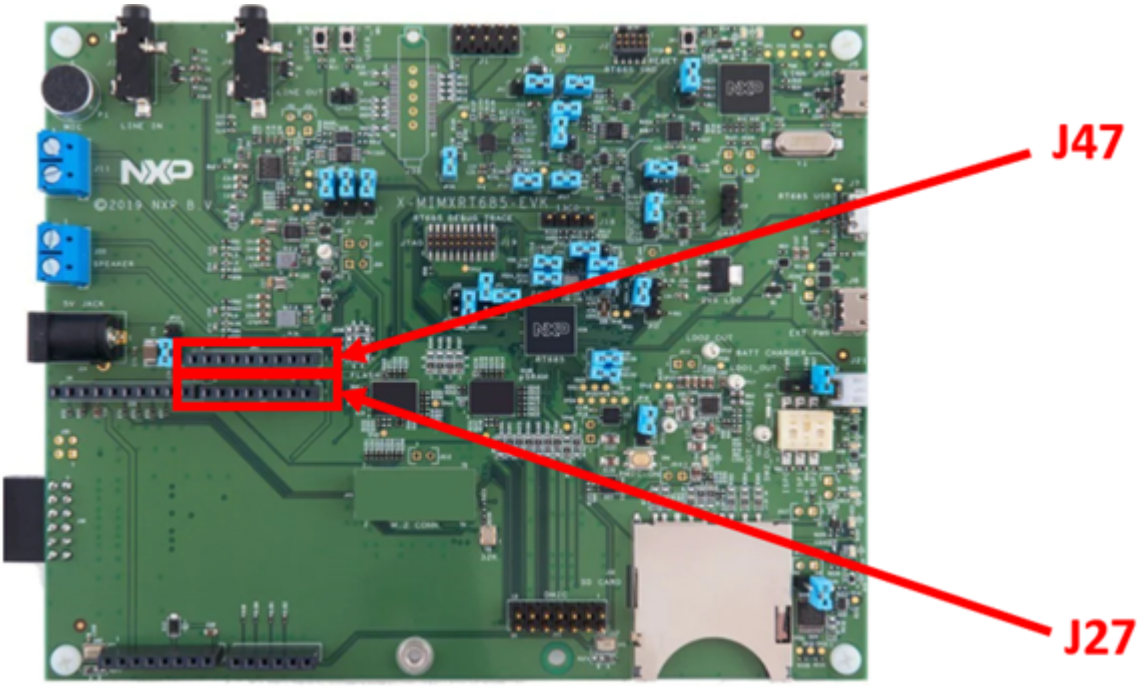
The hardware rework has one part:

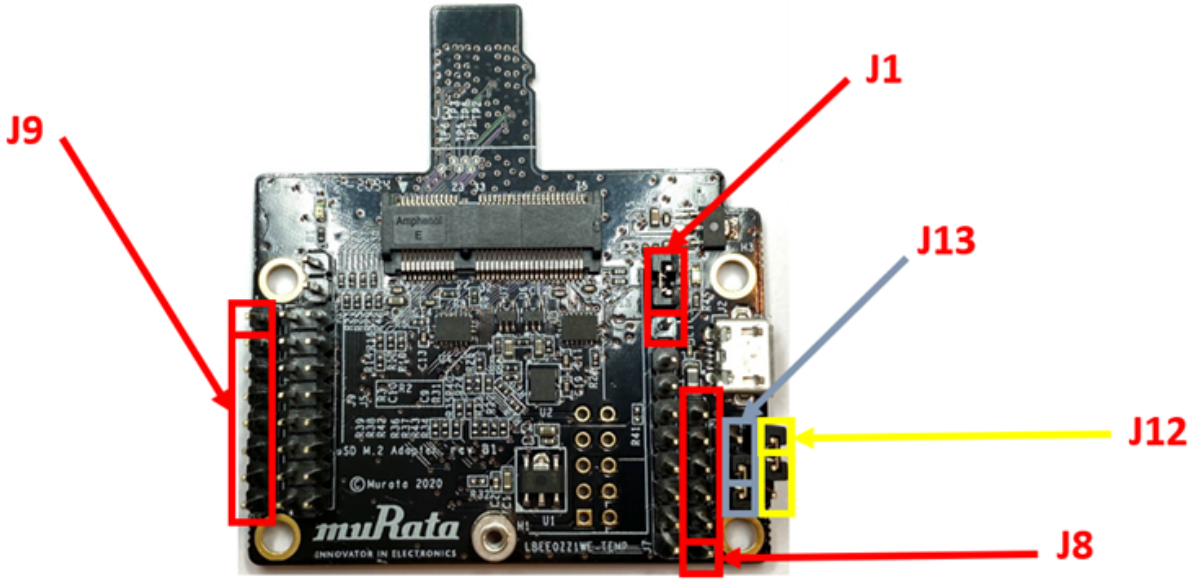
- HCI UART rework

Hardware rework **HCI UART rework :**

- JP12 2-3
- Connect the pins of two boards as the following table using jumper cables included in Murata's uSD-M.2 Adapter kit.

Pin name	uSD-M.2 adapter pin	i.MX RT685 pin	Pin name of RT685	GPIO name of RT685
BT_UART_TXD_HO	J9 (pin 1)	J27 (pin 1)	USART4_RXD	FC4_RXD_SDA_MOSI_DATA
BT_UART_RXD_HO	J9 (pin 2)	J27 (pin 2)	USART4_TXD	FC4_TXD_SCL_MISO_WS
BT_UART_RTS_HO	J8 (pin 3)	J47 (pin 9)	USART4_CTS	FC4_CTS_SDA_SSEL0
BT_UART_CTS_HO	J8 (pin 4)	J27 (pin 5)	USART4_RTS	FC4_RTS_SCL_SSEL1





Murata uSD-M.2 jumper settings:

- Both J12 and J13 = 1-2 (WLAN-SDIO = 1.8 V; and BT-UART and WLAN/BT-CTRL = 3.3 V)
- J1 = 2-3 (3.3 V from uSD connector)

Parent topic:*Hardware Rework Guide for MIMXRT685-EVK and Murata uSD-M.2 Adapter*

Hardware Rework Guide for MIMXRT685-AUD-EVK and Murata M.2 Module This section is a brief hardware rework guidance of the Edgefast Bluetooth PAL on the NXP i.MX MIMXRT685-AUD-EVK board and the Murata's 1XK, 1ZM, 2EL or 2LL solution - direct M.2 connection to Embedded Artists EAR00385 (1XK), EAR00364 (1ZM), Rev-A1 (2EL) or EAR00500 (2LL) M.2 modules.

The hardware rework has one part:

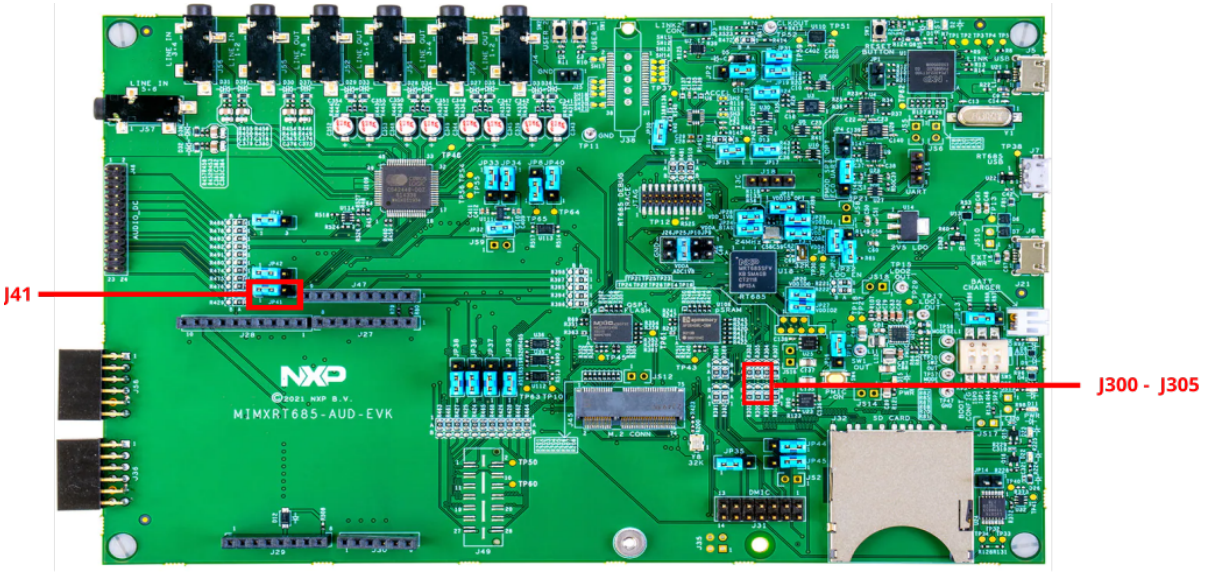
- HCI UART rework

Hardware rework **HCI UART rework:**

Mount R300~R305 A-B

Jumper Setting:

- Connect JP41[2-3]



Parent topic:*Hardware Rework Guide for MIMXRT685-AUD-EVK and Murata M.2 Module*

Hardware Rework Guide for Low Power Feature on MIMXRT595-EVK and Murata 2EL M.2 Module This section is a brief hardware rework guidance of the EdgeFast Bluetooth PAL for low power feature on the NXP i.MX MIMXRT595-EVK board and the Murata's 2EL - direct M.2 connection to Embedded Artists' Rev-A1 (2EL) M.2 modules.

The hardware rework has three parts:

- Debug console serial rework
- Host wake-up controller pin rework (H2C)
- Controller wake-up host pin rework (C2H)

Hardware rework

- **Debug console serial rework**

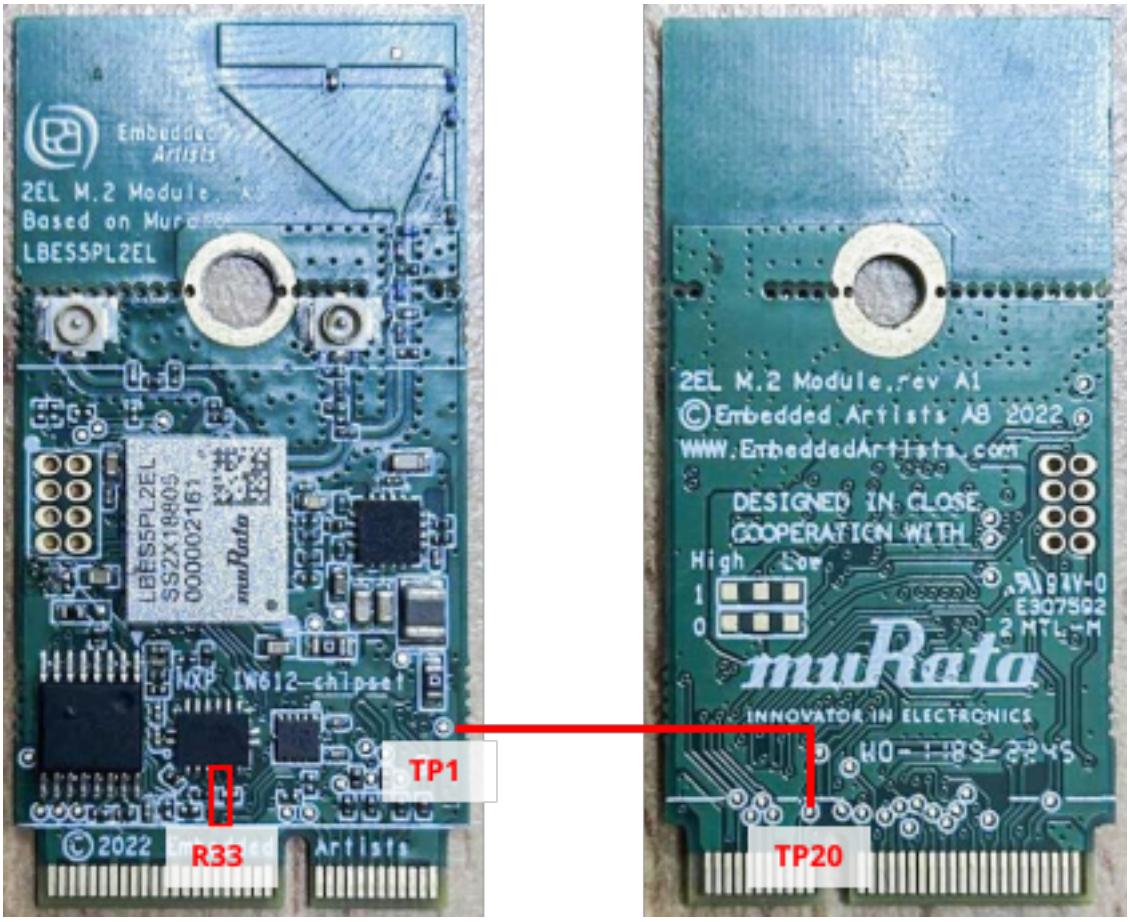
  For details, refer *Hardware Rework Guide for MIMXRT595-EVK and Murata M.2 Module*.

- **Host wake-up controller pin rework**:

  For details, refer Hardware Rework Guide for Low Power Feature on MIMXRT595-EVK and Murata 1XK M.2 Module.

- **Controller wake-up host pin rework:**

1. Remove resistors R709 on MIMXRT595-EVK,
2. Solder 0K ohm resistor on R33 of Murata 2EL M.2 Module
3. Solder 10K ohm resistor on the Murata 2EL M.2 Module between TP1 and TP20.



Parent topic:*Hardware Rework Guide for Low Power Feature on MIMXRT595-EVK and Murata 2EL M.2 Module*

Hardware Rework Guide for MIMXRT595-EVK and Murata M.2 Module This section is a brief hardware rework guidance of the EdgeFast Bluetooth PAL on the NXP i.MX MIMXRT595-EVK board and the Murata's 1XK, 1ZM, 2EL or 2LL solution - direct M.2 connection to Embedded Artists EAR00385 (1XK), EAR00364 (1ZM), Rev-A1 (2EL) or EAR00500 (2LL) M.2 modules.

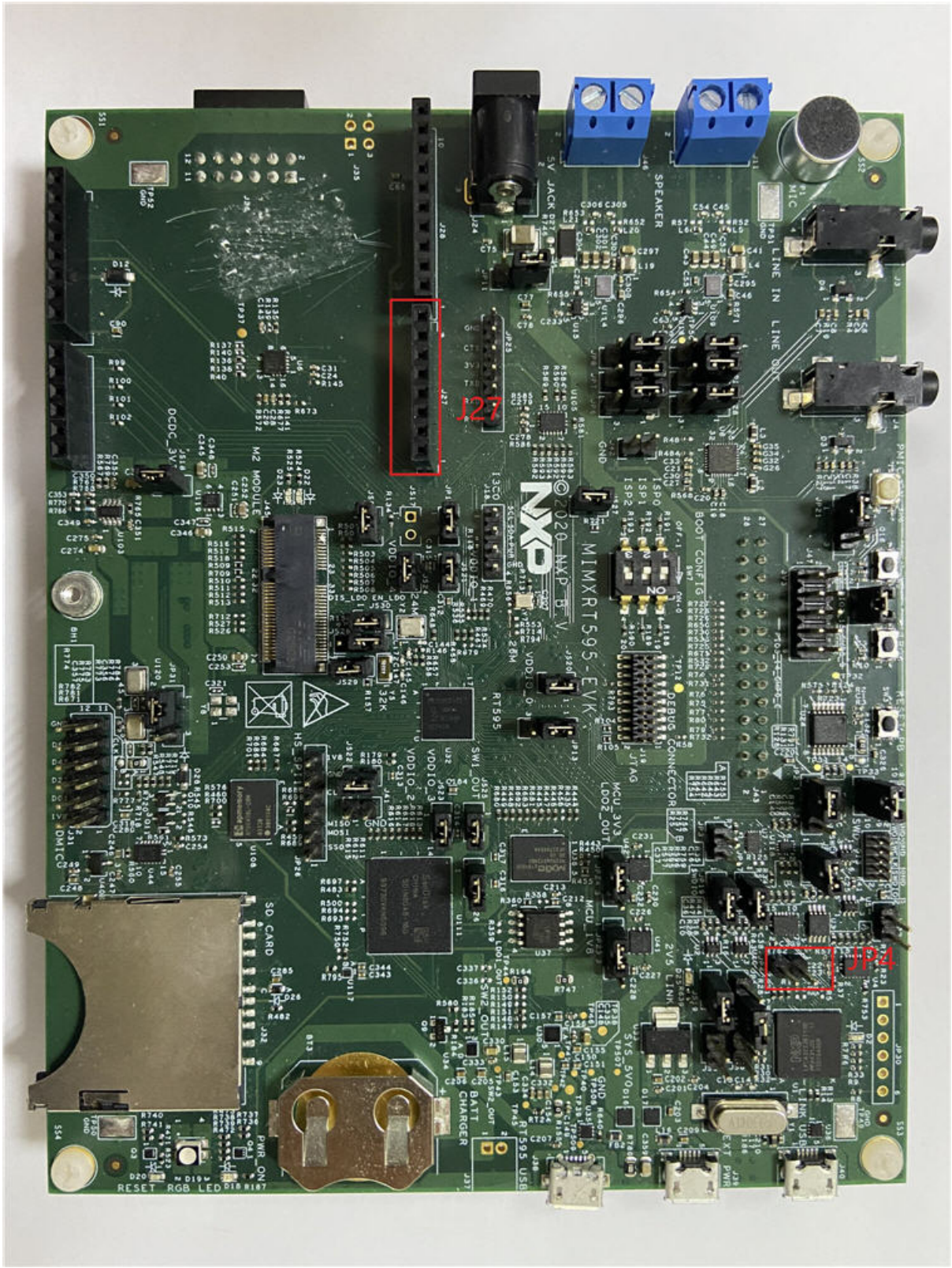
The hardware rework has one part:

- Debug console serial rework

Hardware rework **Debug console serial rework:**

No special rework is required, except the following to enable the debug port.

- JP4 1-2.
- J27 1 - TX of USB to serial converter
- J27 2 - RX of USB to serial converter



Parent topic:*Hardware Rework Guide for MIMXRT595-EVK and Murata M.2 Module*

Hardware Rework Guide for Low Power Feature on MIMXRT595-EVK and Murata 1XK M.2 Module This section is a brief hardware rework guidance of the EdgeFast Bluetooth PAL for low power feature on the NXP i.MX MIMXRT595-EVK board and the Murata's 1XK - direct M.2 connection to Embedded Artists EAR00385 (1XK) M.2 modules.

The hardware rework has three parts:

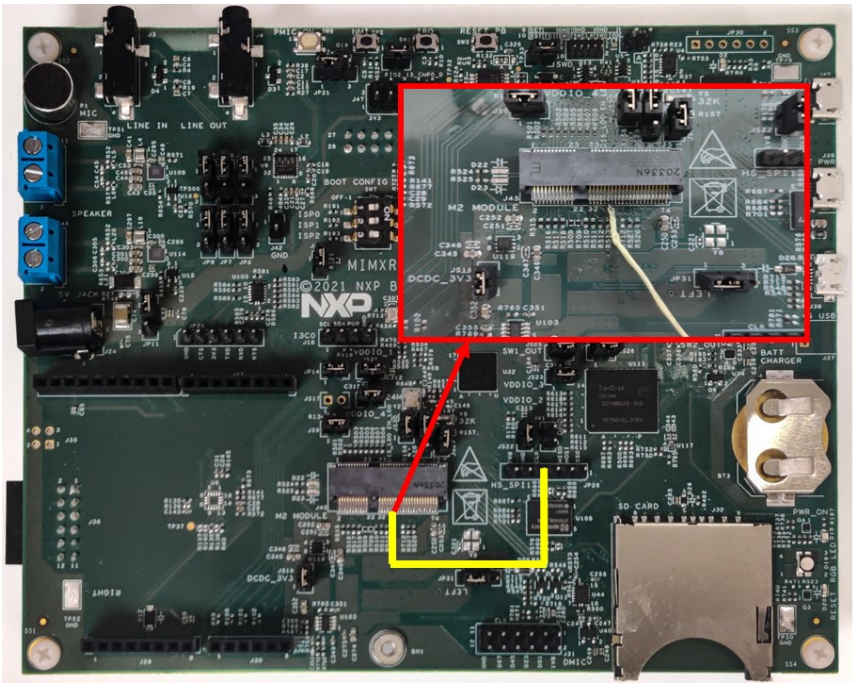
- Debug console serial rework
- Host wake-up controller pin rework (H2C)
- Controller wake-up host pin rework (C2H)

Hardware rework **Debug console serial rework:**

For details, refer *Hardware Rework Guide for MIMXRT595-EVK and Murata M.2 Module*.

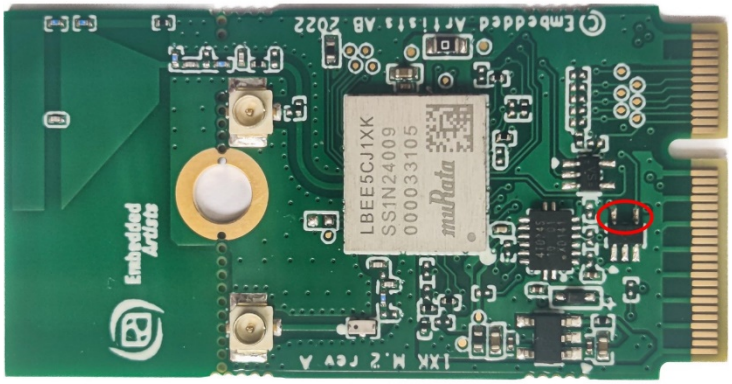
Host wake-up controller pin rework:

Connect M.2 (pin 42) to JP26 (pin 4) with a wire.



Controller wake-up host pin rework:

1. Remove resistors R709 on MIMXRT595-EVK.
2. Solder 10K ohm resistor on the Murata 1XK M.2 Module at the location shown in the following figure.



Parent topic:*Hardware Rework Guide for Low Power Feature on MIMXRT595-EVK and Murata 1XK M.2 Module*

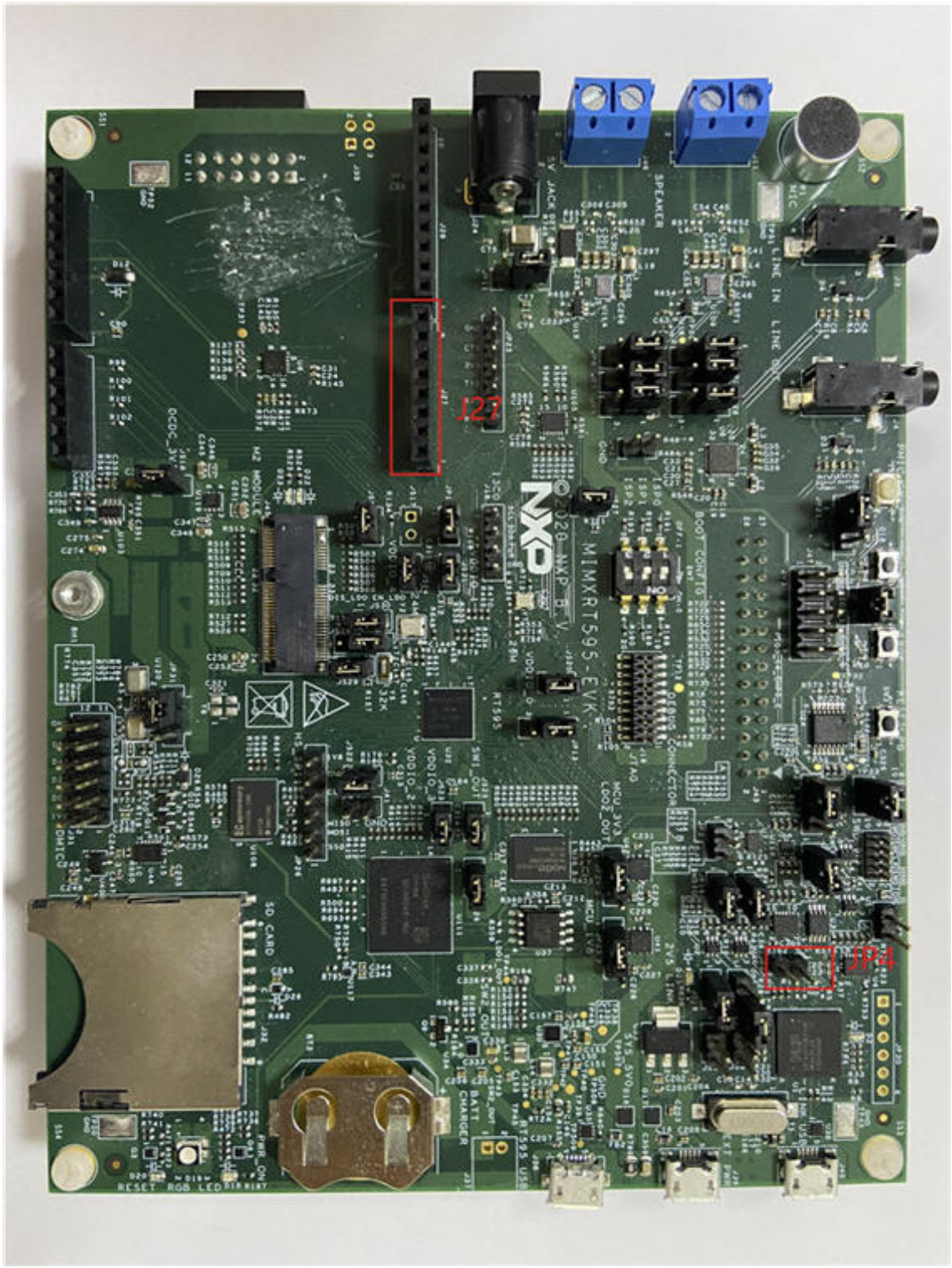
Hardware Rework Guide for MIMXRT595-EVK and AW-AM510MA This section is a brief hardware rework guidance of the EdgeFast Bluetooth PAL on the NXP i.MX MIMXRT595-EVK board and AW-AM510MA. The AW-AM510MA user guide is available here. The hardware rework has one part:

- Debug console serial rework

Hardware rework **Debug console serial rework:**

No special rework is required, except the following to enable the debug port.

- Connect J39 with external power.
- Connect JP4 1-2.
- J27 1 — TX of USB to serial converter.
- J27 2 — RX of USB to serial converter.



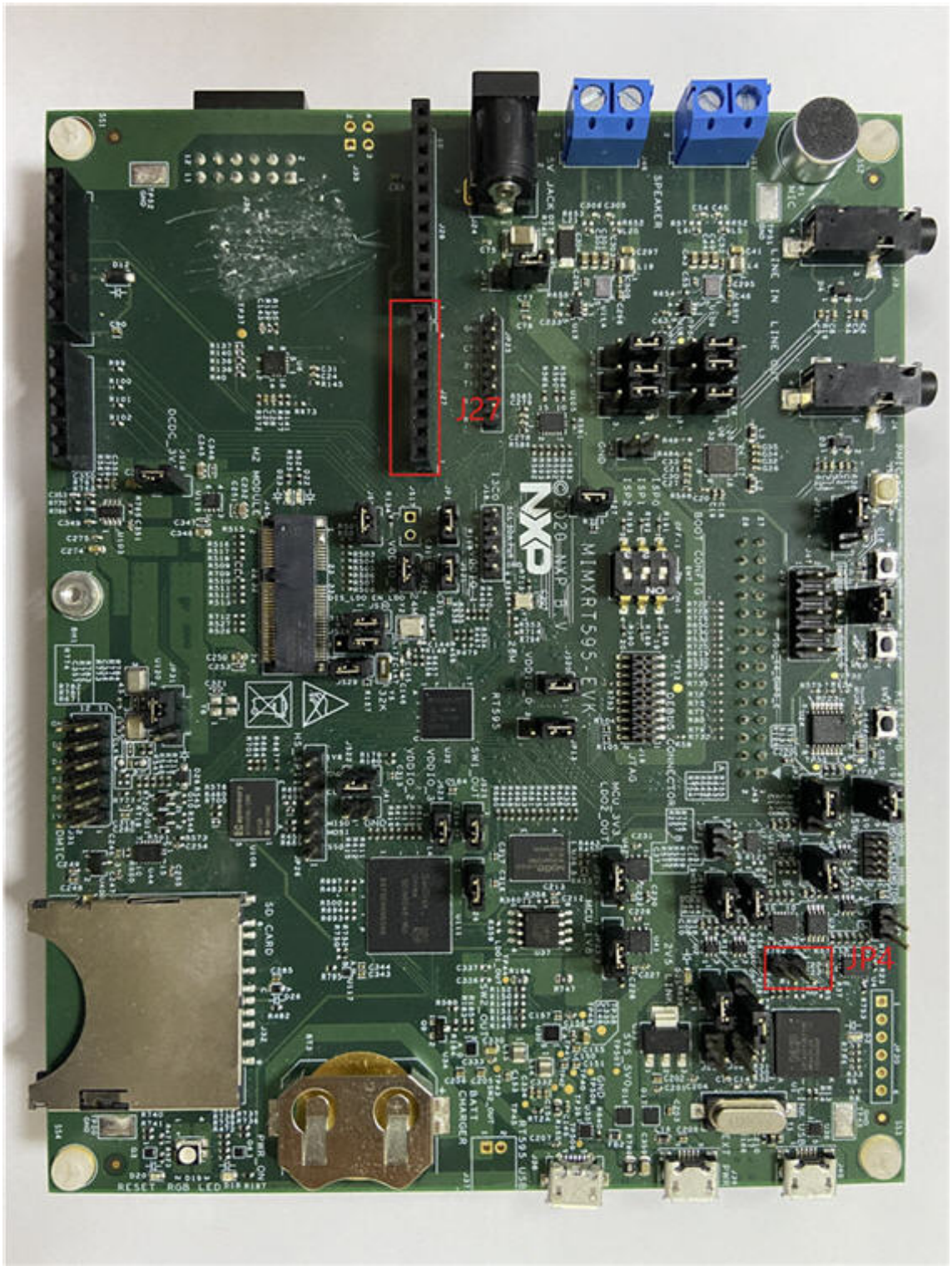
Parent topic:*Hardware Rework Guide for MIMXRT595-EVK and AW-AM510MA*

Hardware Rework Guide for MIMXRT595-EVK and AW-CM358MA This section is a brief hardware rework guidance of the Ethermind Bluetooth stack on the NXP i.MX MIMXRT595-EVK board and AW-CM358MA. The AW-CM358MA user guide is available here. The hardware rework has one part:

- Debug console serial rework

Hardware rework **Debug console serial rework:**

- Connect J39 with external power.
- JP4 1-2
- J27 1 - TX of USB to serial converter
- J27 2 - RX of USB to serial converter



Parent topic:*Hardware Rework Guide for MIMXRT595-EVK and AW-CM358MA*

Hardware Rework Guide for MIMXRT1040-EVK and Murata M.2 Module This section is a brief hardware rework guidance of the Edgefast Bluetooth PAL on the NXP i.MX MIMXRT1040-EVK board and the Murata's 1XK, 1ZM or 2LL solution - direct M.2 connection to Embedded Artists EAR00385 (1XK), EAR00364 (1ZM) or EAR00500 (2LL) M.2 modules.

The hardware rework has two parts:

- HCI UART rework
- PCM interface rework
- Wake pin rework

Hardware rework

1. HCI UART rework
    - Solder R93 and R96
2. PCM interface rework
    - Solder R70 and R79; remove R76 and R86; Connect J80.
3. Wake pin rework
    - When using 2LL M.2 module, remove R456 and R457 to avoid the module has an impact on boot configuration.

Note: Make sure to disconnect J80 when debugging. Otherwise, the debugger downloading fails.

Parent topic:*Hardware Rework Guide for MIMXRT1040-EVK and Murata M.2 Module*

Hardware Rework Guide for MIMXRT1060-EVKC and Murata M.2 Module This section is a brief hardware rework guidance of the EdgeFast Bluetooth PAL on the NXP i.MX MIMXRT1060-EVKC and the Murata's 1XK, 1ZM, 2EL or 2LL solution - direct M.2 connection to Embedded Artists EAR00385 (1XK), EAR00364 (1ZM), Rev-A1 (2EL) or EAR00500 (2LL) M.2 modules.

The hardware rework has two parts:

- HCI UART rework
- PCM interface rework

Hardware rework

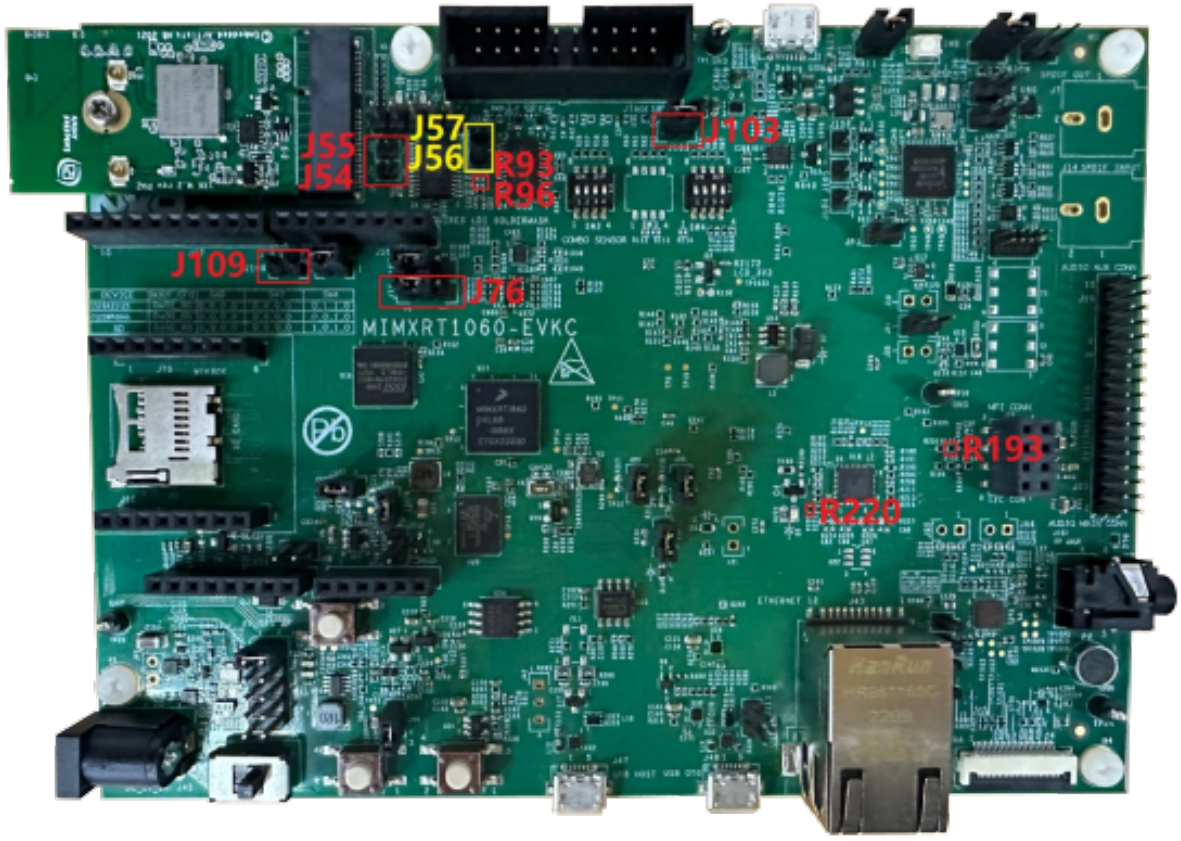
- **HCI UART rework**

1. Mount R93, R96.
2. Remove R193.
3. Connect J109, connect J76 2-3.

- **PCM interface rework**

1. Remove J54 and J55, connect J56 and J57.
2. Remove R220.
3. Connect J103.

Note: When J103 is connected, flash cannot be downloaded. So, remove the connection when downloading flash and reconnect it after downloading.



Parent topic:*Hardware Rework Guide for MIMXRT1060-EVKC and Murata M.2 Module*

Hardware Rework Guide for MIMXRT1060-EVKC and Murata 2EL M.2 Adapter This section is a brief hardware rework guidance of the EdgeFast Bluetooth PAL on the NXP i.MX MIMXRT1060-EVKC and the Murata 2EL M.2 solution - direct M.2 connection to Embedded Artists' Rev-A1 (2EL) M.2 modules.

The hardware rework has three parts:

- HCI UART rework
- PCM interface rework
- LE Audio Synchronization interface rework (only used on sink side)

Hardware rework

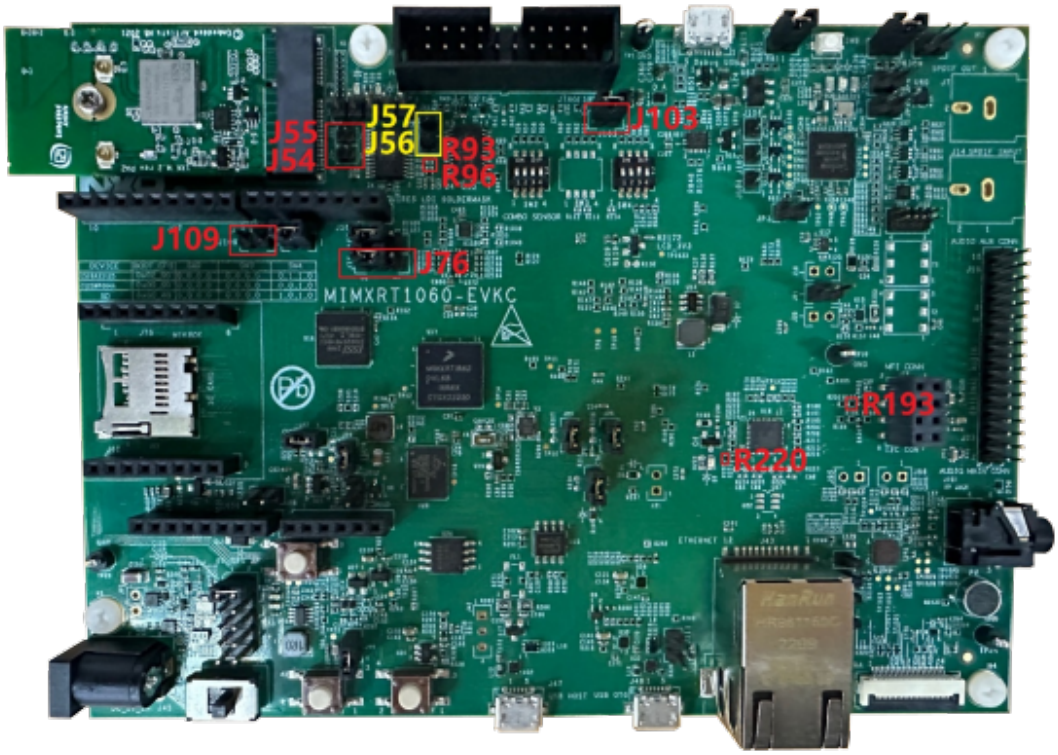
- HCI UART rework

1. Mount R93, R96.
2. Remove R193.
3. Connect J109, connect J76 2-3.

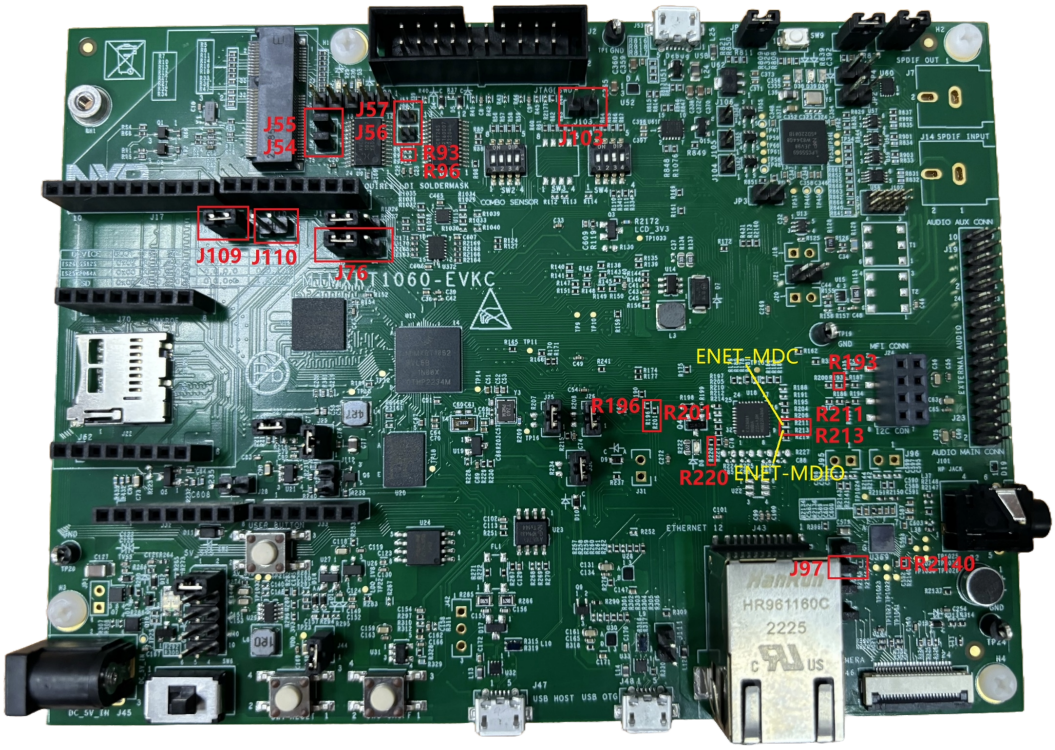
- PCM interface rework

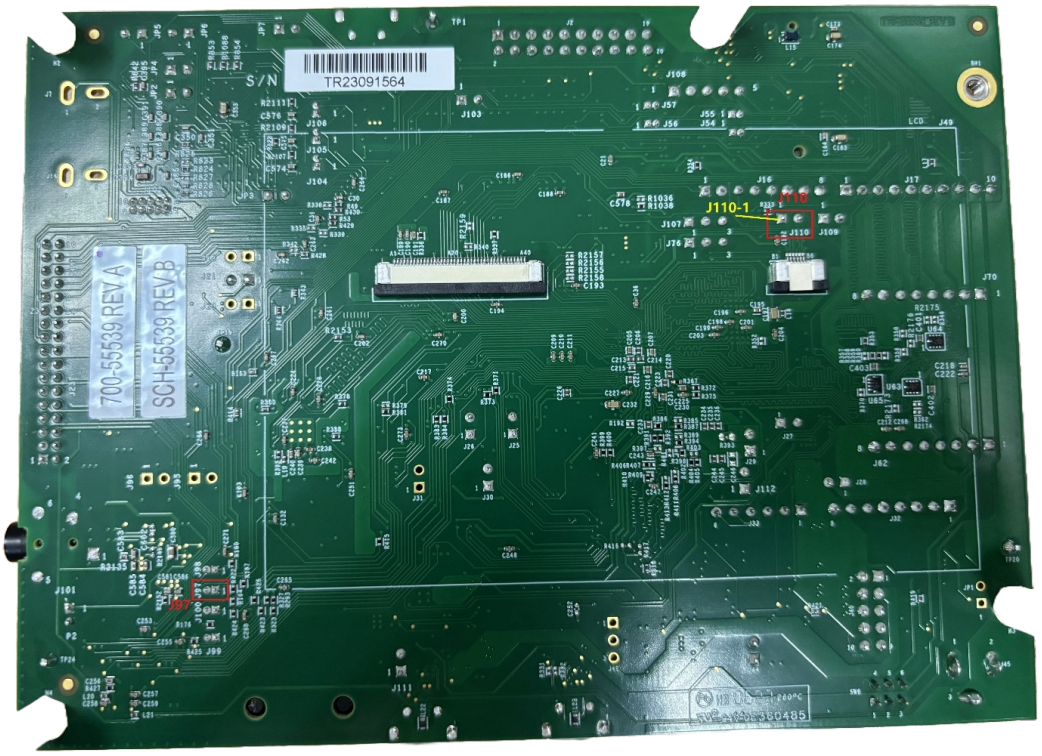
1. Remove J54 and J55, connect J56, and J57.
2. Remove R220.
3. Connect J103.

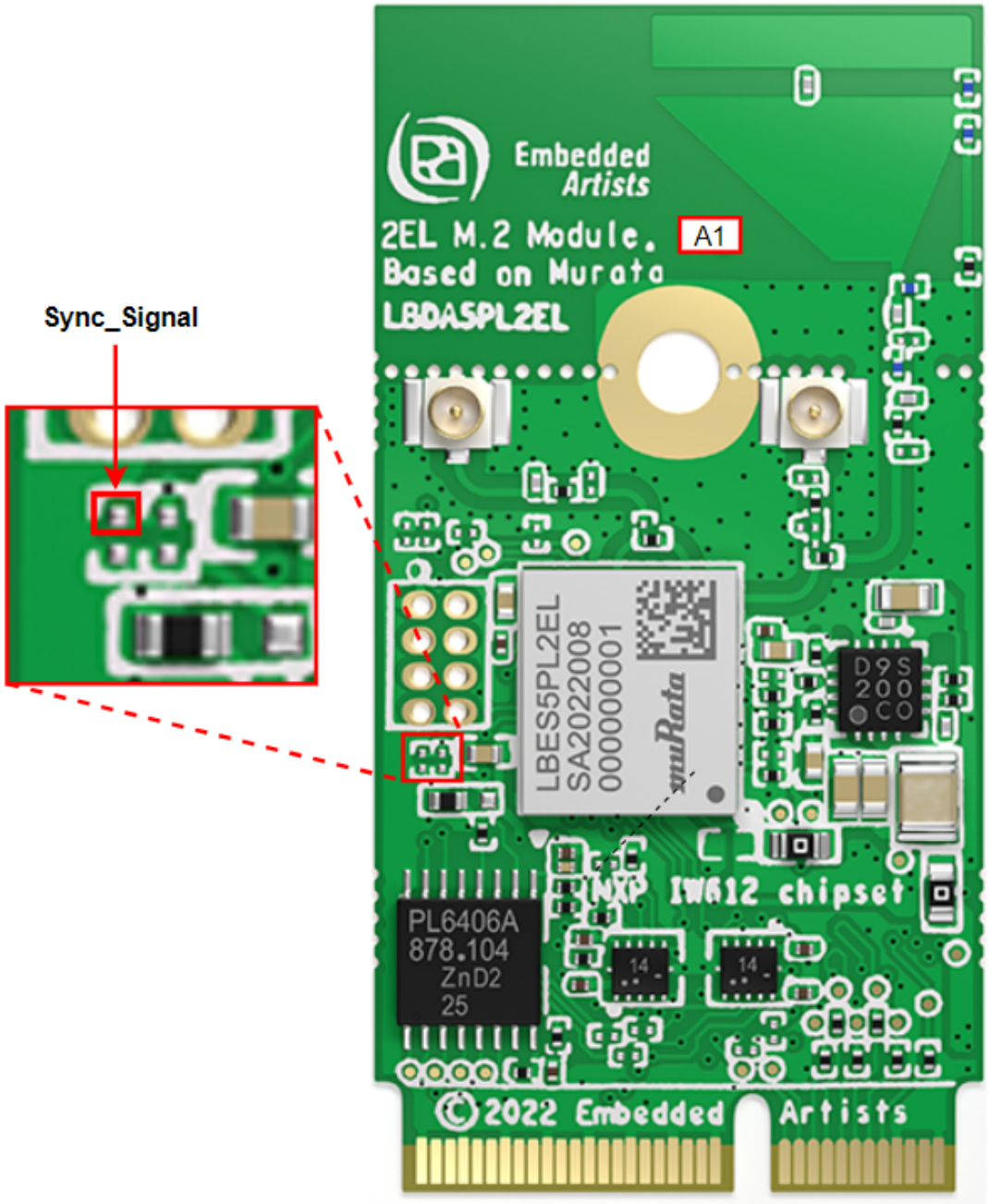
   Note: When J103 is connected, flash cannot be downloaded. So, remove the connection when downloading flash and reconnect it after downloading.



- LE Audio Synchronization interface rework (only used on sink side)
  1. Remove J110 jumper cap.
  2. Remove R196, R201, R213, and R211.
  3. Connect J110-1 (GPT2_CLK) to R2140 (SAI_MCLK).
  4. Connect ENET_MDIO (GPT2_CAP1) with J97 (SAI_SW).
  5. Connect ENET_MDC (GPT2_CAP2) with 2EL's GPIO_27 (Sync Signal).




S/N
TR23091564
700-55539 REV.A
SCH-55539 REV.B
J110-1
J110
J107
J101



Parent topic:*Hardware Rework Guide for MIMXRT1060-EVKC and Murata 2EL M.2 Adapter*

Hardware Rework Guide for MCXN547-EVK and Murata M.2 Module This section is a brief hardware rework guidance of the EdgeFast Bluetooth PAL on the NXP MCXN547-EVK board and the Murata's 1XK, 1ZM or 2LL solution - direct M.2 connection to Embedded Artists EAR00385 (1XK), EAR00364 (1ZM) or EAR00500 (2LL) M.2 modules.

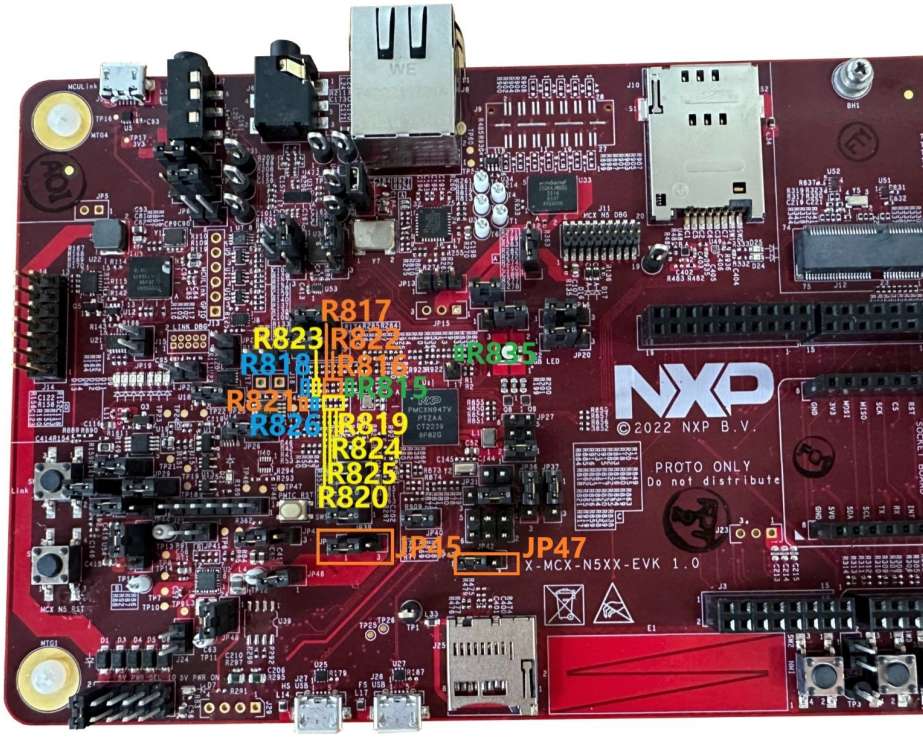
The hardware rework consists of two parts:

- M.2 UART interface
- M.2 SDIO interface

Hardware rework

- M.2 UART interface rework

  - Mount R835
  - Connect JP45 2-3 to supply 1.8V for GPIO4
- M.2 SDIO interface rework
  - Connect JP47 2-3 to supply 1.8V for GPIO2
  - Remove R818, connect R823
  - Remove R819, connect R824
  - Remove R817, connect R822
  - Remove R815, connect R816
  - Remove R820, connect R825
  - Remove R821, connect R826



Hardware Rework Guide for MCXN947-EVK and Murata M.2 Module This section is a brief hardware rework guidance of the EdgeFast Bluetooth PAL on the NXP MCXN947-EVK board and the Murata's 1XK, 1ZM or 2LL solution - direct M.2 connection to Embedded Artists EAR00385 (1XK), EAR00364 (1ZM) or EAR00500 (2LL) M.2 modules.

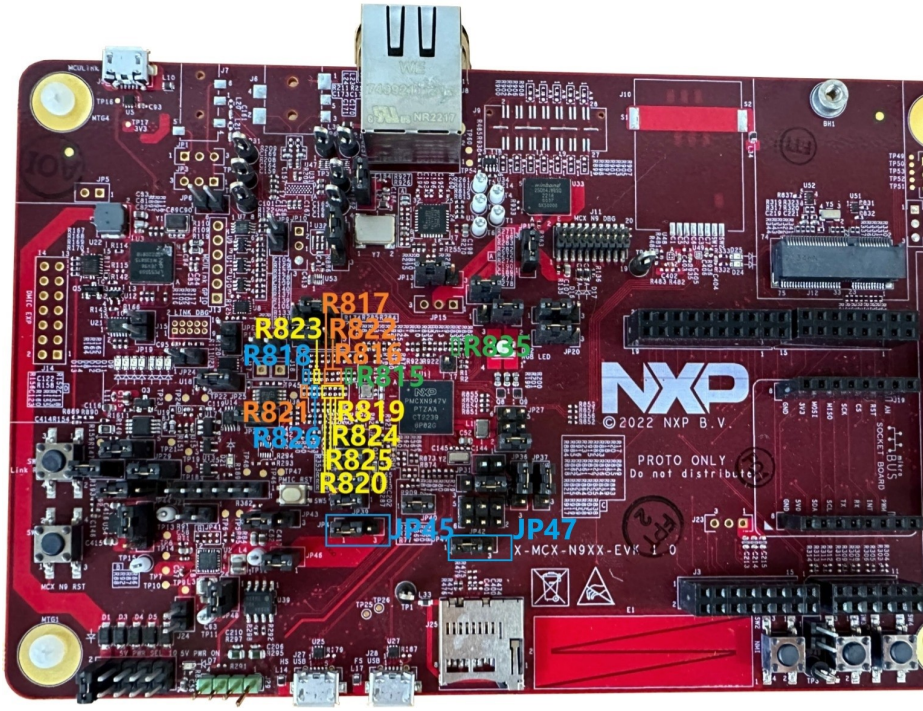
The hardware rework consists of two parts:

- M.2 UART interface
- M.2 SDIO interface

Hardware rework

- M.2 UART interface rework
  - Mount R835
  - Connect JP45 2-3 to supply 1.8V for GPIO4
- M.2 SDIO interface rework
  - Connect JP47 2-3 to supply 1.8V for GPIO2

- Remove R818, connect R823
- Remove R819, connect R824
- Remove R817, connect R822
- Remove R815, connect R816
- Remove R820, connect R825



- Remove R821, connect R826

Hardware Rework Guide for IMXRT1050-EVKB and Murata M.2 Module This section is a brief hardware rework guidance of the EdgeFast Bluetooth PAL on the NXP IMXRT1050-EVKB board and the Murata 1XK,1ZM and 2EL solution - direct M.2 connection to Embedded Artists' EAR00385 (1XK) , EAR00364 (1ZM) or EAR00409 (2EL)M.2 modules. The hardware rework consists of three parts:

- Murata uSDM
- HCI UART rework

Hardware rework

- Murata uSD-M.2 jumper settings
  - J12 = 1-2: WLAN-SDIO & BT-PCM = 1.8 V
  - J13 = 1-2: BT-UART & WLAN/BT-CTRL = 3.3 V
  - J1 = 2-3: 3.3 V from uSD connector
- HCI UART interface rework

  Connect the TX/RX/RTS/CTS pins of the two boards as show in Table 1 using the jumper cables included in the Murata's uSD-M.2 Adapter kit as shown in the following table.

Pin name	uSD-M.2 adapter pin	i.MX RT1050-EVKB pin	Pin name of RT1050-EVKB	GPIO name of RT1050-EVKB
BT_UART_TXD_	J9 (pin 1)	J22 (pin 1)	LPUART3_RXD	GPIO_AD_B1_07
BT_UART_RXD_	J9 (pin 2)	J22 (pin 2)	LPUART3_TXD	GPIO_AD_B1_06
BT_UART_RTS_]	J8 (pin 3)	J23 (pin 3)	LPUART3_CTS	GPIO_AD_B1_04
BT_UART_CTS_]	J8 (pin 4)	J23 (pin 4)	LPUART3_RTS	GPIO_AD_B1_05
GND	J7 (pin 7)	J25 (pin 7)	GND	GND

Parent topic:*Hardware Rework Guide for IMXRT1050-EVKB and Murata M.2 Module*

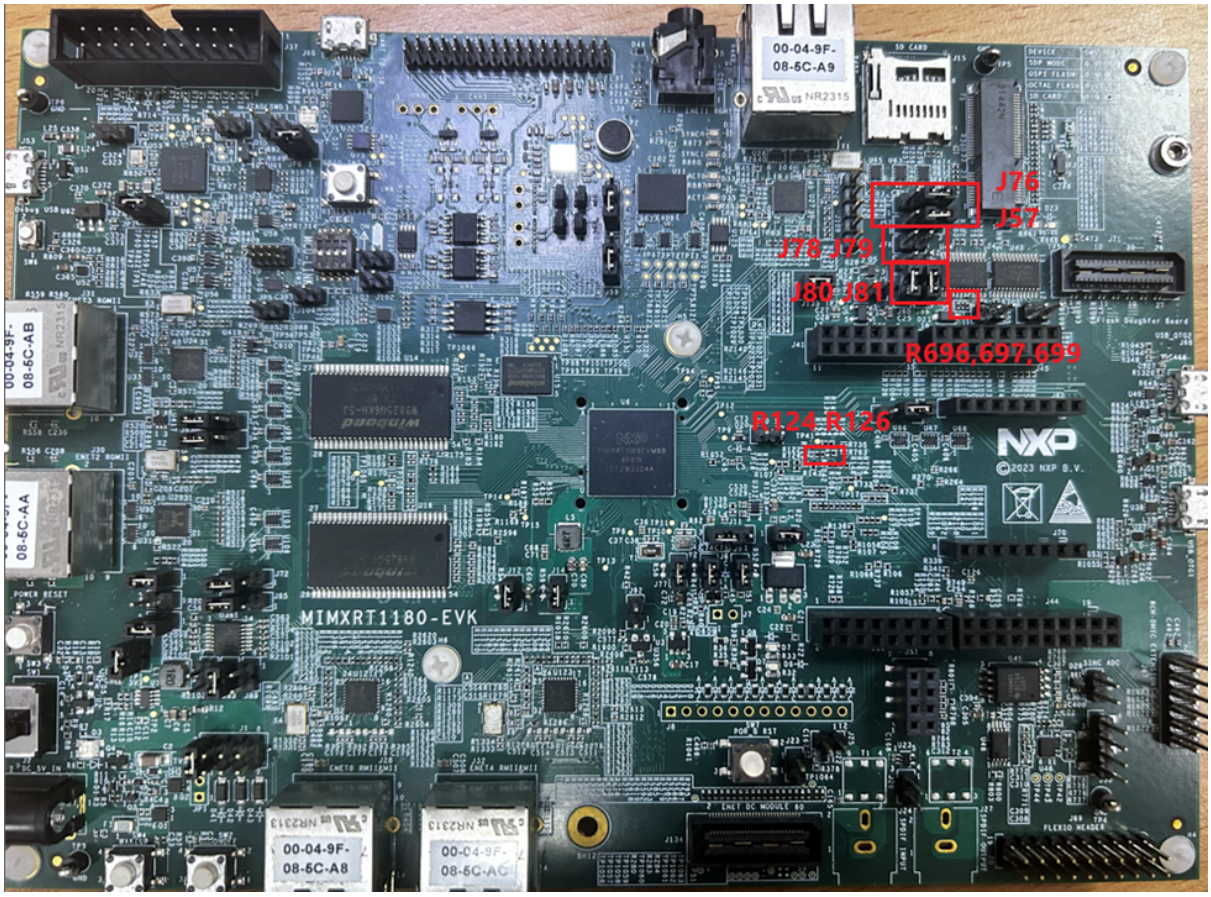
Hardware Rework Guide for MIMXRT1180 and Murata M.2 Module This section is a brief hardware rework guidance of the EdgeFast Bluetooth PAL on the NXP MIMXRT1180 board and the Murata's 1XK, 1ZM, 2EL or 2LL solution - direct M.2 connection to Embedded Artists EAR00385 (1XK), EAR00364 (1ZM), Rev-A1 (2EL) or EAR00500 (2LL) M.2 modules.

The hardware rework consists of two parts:

- HCI UART rework
- PCM interface rework

Hardware rework

- HCI UART rework:
  - Remove: R124,R126
  - Mount R696, R697
  - Connect J57 [2-3], J76 [2-3]
- PCM interface rework
  - Mount R699
  - Disconnect J78 J79
  - Connect J80 J81



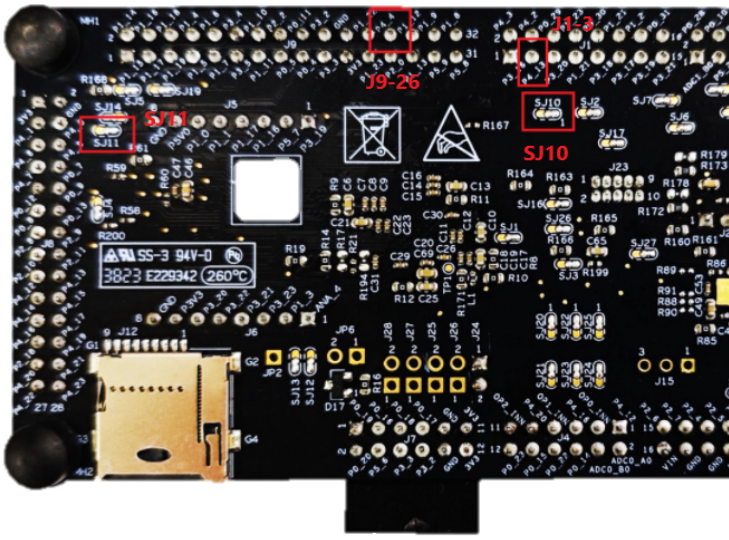
Hardware Rework Guide for FRDM-MCXN947 and X-FRDM-WIFI-M.2 Adapter This section is a brief hardware rework guidance of the EdgeFast Bluetooth PAL on the NXP FRDM-MCXN947 board and X-FRDM-WIFI-M.2 or the Murata's 2LL EAR00500 (2LL) M.2 modules solution.

The hardware rework consists of one part:

- UART interface rework

Hardware rework

- UART interface rework
  - Remove SJ11 1-2, connect SJ11 2-3
  - Remove SJ10 1-2, connect J1-3 to J9-26
- X-FRDM-WIFI-M.2 jumper setting
  - Connect J8(On X-FRDM-WIFI-M.2) for 1.8V
  - Connect J24(On X-FRDM-WIFI-M.2) for 3.3V
  - Connect J19(On X-FRDM-WIFI-M.2) for 1.8V
  - Connect J25(On X-FRDM-WIFI-M.2) for 3.3V
  - Connect J15(On X-FRDM-WIFI-M.2) for 1.8V
  - Connect J16(On X-FRDM-WIFI-M.2) for 3.3V
  - Connect J17(On X-FRDM-WIFI-M.2) for 1.8V



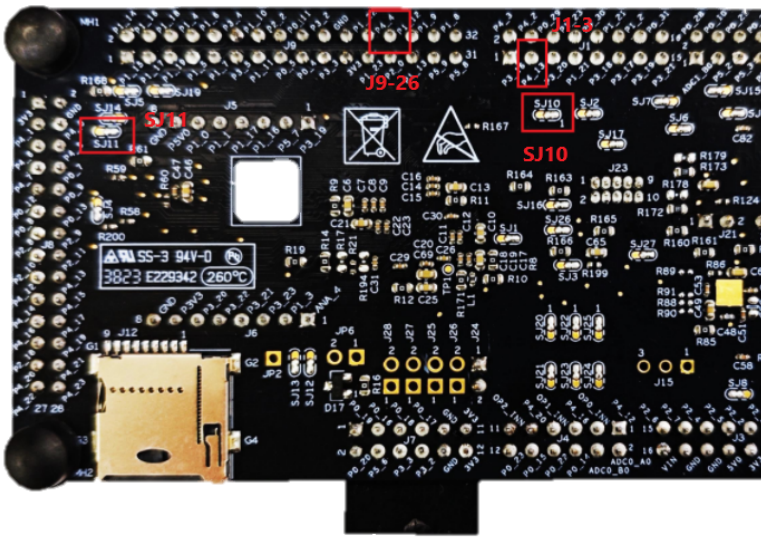
  - Connect J18(On X-FRDM-WIFI-M.2) for 3.3V

Hardware Rework Guide for FRDM-MCXN947 and FRDM-IW416-AW-AM510 This section is a brief hardware rework guidance of the EdgeFast Bluetooth PAL on the NXP FRDM-MCXN947 board and FRDM-IW416-AW-AM510 board. The hardware rework consists of two parts:

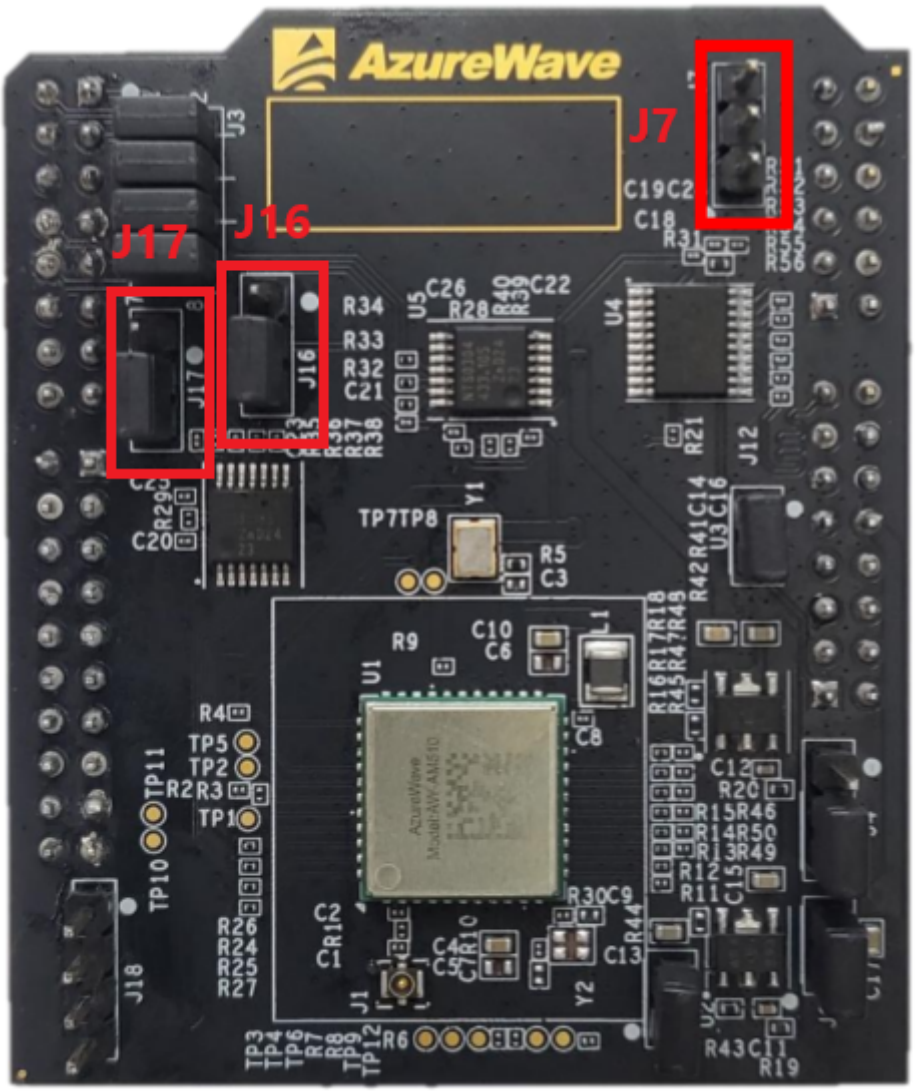
- UART interface rework
- FRDM-IW416-AW-AM510

Hardware rework

- UART interface rework
  - Remove SJ11 1-2, connect SJ11 2-3
  - Remove SJ10 1-2, connect J1-3 to J9-26
- FRDM-IW416-AW-AM510 jumper setting
  - Connect J16 2-3 for 3.3V supply
  - Connect J17 2-3 for 3.3V UART voltage level



  - Connect J7 2-3 for 3.3V SDIO voltage level



Enabling Additional EdgeFast Bluetooth Protocol Abstraction Layer Examples on RT1064

Introduction NXP supports Bluetooth/Bluetooth Low Energy on RT1060EVK and RT1060EVKC. RT1064 has the same MCU die with RT1060EVK and RT1060EVKC and therefore it is possible to migrate the examples.

This document takes *peripheral_ht* as an example and describes the steps to migrate EdgeFast examples from RT1060EVK to RT1064 (based on SDK 2.13.0) and from RT1060EVKC to RT1064 (based on SDK 2.14.0) with different toolchains including IAR, Arm GCC, and MDK.

Migrate examples from RT1060EVK to RT1064 This topic describes the Common steps and the steps to migrate with the IAR, Arm GCC, and MDKtoolchains.

Common steps

1. Download *SDK_2.13.0_EVK-MIMXRT1060* and *SDK_2.13.0_EVK-MIMXRT1064*.
2. Copy the following folders from RT1060EVK package to RT1064 package: <install_dir>/components/internal_flash/ <install_dir>/middleware/edgefast_bluetooth/ <install_dir>/middleware/wireless/.

3. Create a folder named edgefast_bluetooth_examples/ under <rt1064_install_dir>/boards/evkmimxrt1064/.
4. Copy the entire folder from <rt1060evk_install_dir>/boards/evkmimxrt1060/edgefast_bluetooth_examples/peripheral_ht/ to < rt1064_install_dir>/boards/evkmimxrt1064/edgefast_bluetooth_examples/.
5. Copy *clock_config.[c/h]* and *board.c* from <rt1064_install_dir>/boards/evkmimxrt1064/demo_apps/hello_world/ to <rt1064_installed>/boards/evkmimxrt1064/edgefast_bluetooth_examples/peripheral_ht/ to replace the previous files.
6. Add #define EDGEFAST_BT_LITTLEFS_MFLASH 1 in <rt1064_install_dir>/boards/evkmimxrt1064/edgefast_bluetooth_examples/peripheral_ht /app_config.c.
7. Make the following changes in <rt1064_installed>/boards/evkmimxrt1064/edgefast_bluetooth/peripheral_ht/board.h.

```
73 #define BOARD_FLASH_SIZE (0x800000U)
```

```
73 #define BOARD_FLASH_SIZE (0x400000U)
```

Parent topic:*Migrate examples from RT1060EVK to RT1064*

IAR

1. Navigate to <rt1064_install_dir>/boards/evkmimxrt1064/edgefast_bluetooth_examples/peripheral_ht/iar/.
2. Make the following changes.

File name	Previous item	New item
peripheral_ht.ewp	1060	1064
	1062	1064

3. Rename MIMXRT1062xxxxx_flexspi_nor.icf as MIMXRT1064xxxxx_flexspi_nor.icf and make the following changes.

```
47 define symbol m_interrupts_start       = 0x60002000;
48 define symbol m_interrupts_end         = 0x600023FF;
49
50 define symbol m_text_start             = 0x60002400;
51 define symbol _ROM_END_                = 0x6057FFFF;
```

```
47 define symbol m_interrupts_start       = 0x70002000;
48 define symbol m_interrupts_end         = 0x700023FF;
49
50 define symbol m_text_start             = 0x70002400;
51 define symbol _ROM_END_                = 0x7017FFFF;
```

```
66 define exported symbol m_boot_hdr_conf_start = 0x60000000;
67 define symbol m_boot_hdr_ivt_start           = 0x60001000;
68 define symbol m_boot_hdr_boot_data_start     = 0x60001020;
69 define symbol m_boot_hdr_dcd_data_start      = 0x60001030;
```

```
66 define exported symbol m_boot_hdr_conf_start = 0x70000000;
67 define symbol m_boot_hdr_ivt_start           = 0x70001000;
68 define symbol m_boot_hdr_boot_data_start     = 0x70001020;
69 define symbol m_boot_hdr_dcd_data_start      = 0x70001030;
```

```
95 BT_LITTLEFS_STORAGE_SECTOR_SIZE = 0x1000;  /* 4k flash secto
96 BT_LITTLEFS_STORAGE_MAX_SECTORS = (0x60800000 - EDGEFAST_BT_
97 ***/
```

```
95 BT_LITTLEFS_STORAGE_SECTOR_SIZE = 0x1000;  /* 4k flash sect
96 BT_LITTLEFS_STORAGE_MAX_SECTORS = (0x70400000 - EDGEFAST_BT
97 ***/
```

Parent topic:*Migrate examples from RT1060EVK to RT1064*

Arm GCC

1. Navigate to <rt1064_install_dir>/boards/evkmimxrt1064/edgefast_bluetooth_examples/peripheral_ht/armgcc/.
2. Rename the following files.

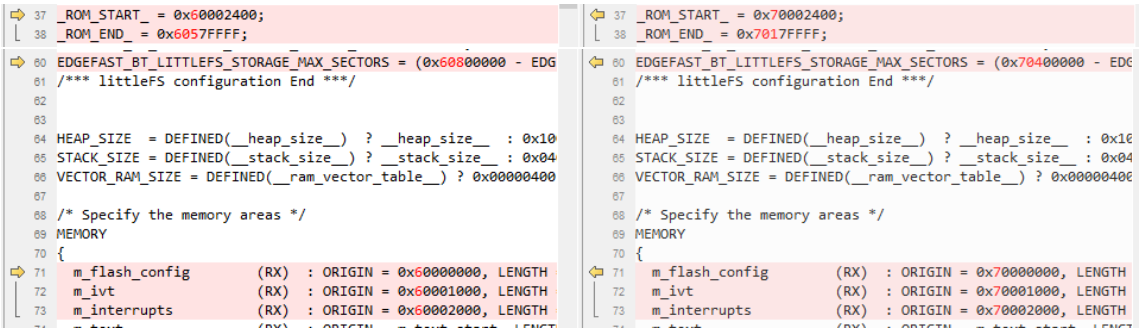
Path	Previous name	**New name**
<rt1064_install_dir>/middleware/wireless/ethermind/	middleware_edgefast_bluetooth_cmake	middleware_edgefast_bluetooth_k32w061_contro cmake

3. Make following changes.

File name	Previous item	New item
config.cmake	1060	1064
	1062	1064
flags.cmake	1062	1064
CMakeLists.txt	1060	1064
	1062	1064

4. *mflash* is used in RT1064 instead of flash_adapter,therefore, comment include(component_flexspi_nor_flash_adapter_rt1064_MIMXRT1064) in *CMakeLists.txt*.

5. Rename MIMXRT1062xxxxx_flexspi_nor.ld as MIMXRT1064xxxxx_flexspi_nor.ld and make the following changes.



Parent topic:*Migrate examples from RT1060EVK to RT1064*

MDK

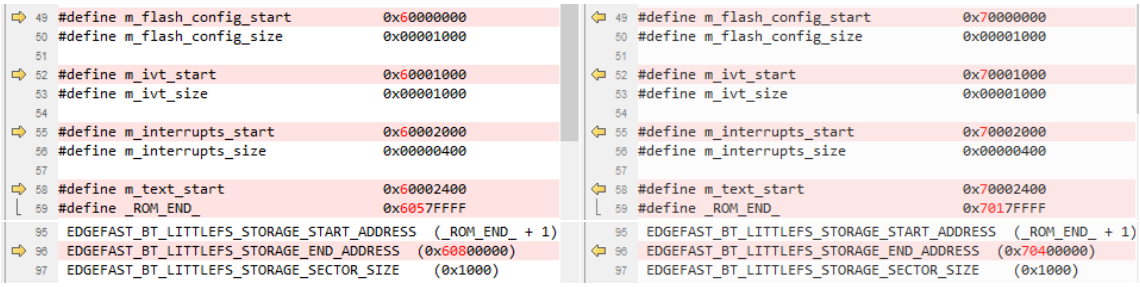
1. Navigate to <rt1064_install_dir>/boards/evkmimxrt1064/edgefast_bluetooth_examples/peripheral_ht/mdk/.

2. Make following changes.

File name	Previous item	New item
peripheral_ht.uvprojx	1060	1064
	1062	1064

3. Copy evkmimxrt1064_flexspi_nor.ini from <rt1064_install_dir>/boards/evkmimxrt1064/demo_apps/hello_world/mdk/ to <rt1064_install_dir>/boards/evkmimxrt1064/edgefast_bluetooth_examples/peripheral_ht/mdk/.

4. Rename MIMXRT1062xxxxx_flexspi_nor as MIMXRT1064xxxxx_flexspi_nor and make the following changes.



Parent topic:*Migrate examples from RT1060EVK to RT1064*

Migrate examples from RT1060EVKC to RT1064 This topic describes the Common steps and the steps to migrate with the IAR, Arm GCC, and MDK toolchains.

Common steps

1. Download SDK_2.14.0_EVKC-MIMXRT1060 and SDK_2.14.0_EVK-MIMXRT1064.
2. Copy the following folders from the RT1060EVKC package to the RT1064 package: <install_dir>/middleware/edgefast_bluetooth/ <install_dir>/middleware/wireless/ethermind.
3. Create a new folder named edgefast_bluetooth_examples/ under <rt1064_install_dir>/boards/evkmimxrt1064/.
4. Copy the entire folder from <rt1060evkc_install_dir>/boards/evkcmimxrt1060/edgefast_bluetooth_examples/peripheral_ht/ to <rt1064_install_dir>/boards/evkmimxrt1064/edgefast_bluetooth_examples/.
5. Copy clock_config.[c/h] and board.c from <rt1064_install_dir>/boards/evkmimxrt1064/demo_apps/hello_world/ to <rt1064_installed>/boards/evkmimxrt1064/edgefast_bluetooth_examples/peripheral_ht/ to replace the previous files.

Parent topic:*Migrate examples from RT1060EVKC to RT1064*

IAR

1. Navigate to <rt1064_install_dir>/boards/evkmimxrt1064/edgefast_bluetooth_examples/peripheral_ht/iar/.
2. Make the following changes in the listed order.

File name	Previous item	New item
peripheral_ht.ewp	1062	1064
mflash/evkcmimxrt1060	mflash/mimxrt1064	
evkcmimxrt1060	evkmimxrt1064	
6B	6A	

3. Rename MIMXRT1062xxxxx_flexspi_nor.icf as MIMXRT1064xxxxx_flexspi_nor.icf and make the following changes.

```
define symbol m_interrupts_start       = 0x60002000;
define symbol m_interrupts_end         = 0x600023FF;

define symbol m_text_start             = 0x60002400;
define symbol m_text_end               = 0x607FFFFF -

define exported symbol m_boot_hdr_conf_start = 0x60000000;
define symbol m_boot_hdr_ivt_start           = 0x60001000;
define symbol m_boot_hdr_boot_data_start     = 0x60001020;
define symbol m_boot_hdr_dcd_data_start      = 0x60001030;
```

```
define symbol m_interrupts_start       = 0x70002000;
define symbol m_interrupts_end         = 0x700023FF;

define symbol m_text_start             = 0x70002400;
define symbol m_text_end               = 0x703FFFFF -

define exported symbol m_boot_hdr_conf_start = 0x70000000;
define symbol m_boot_hdr_ivt_start           = 0x70001000;
define symbol m_boot_hdr_boot_data_start     = 0x70001020;
define symbol m_boot_hdr_dcd_data_start      = 0x70001030;
```

Parent topic:*Migrate examples from RT1060EVKC to RT1064*

Arm GCC

1. Navigate to <rt1064_install_dir>/boards/evkmimxrt1064/edgefast_bluetooth_examples/peripheral_ht/armgcc/.

2. Copy folder from <rt1060evkc_install_dir>/boards/evkcmimxrt1060/edgefast_bluetooth_examples/template/ to <rt1064_install_dir>/boards/evkmimxrt1064/edgefast_bluetooth_examples/ and rename the files.

Path	Previous name	New name
<rt1064_install_dir>/boards/evkmimxrt1064/edgefast_bluetooth_examples/	middleware_edgefast_bluetooth_mcux_linker_template_evkcmimxrt1060.cmake	middleware_edgefast_bluetooth_mcux_linker_template_evkmimxrt1064.cmake
	middleware_edgefast_bluetooth_sdio_template_evkcmimxrt1060.cmake	middleware_edgefast_bluetooth_sdio_template_evkmimxrt1064.cmake

3. Add the following content to<rt1064_install_dir>/devices/MIMXRT1064/all_lib_device.cmake at appropriate location.

```
...
  ${CMAKE_CURRENT_LIST_DIR}/../../boards
  ${CMAKE_CURRENT_LIST_DIR}/../../boards/evkmimxrt1064/edgefast_bluetooth_examples/
↪template
  ${CMAKE_CURRENT_LIST_DIR}/../../middleware/edgefast_bluetooth
  ${CMAKE_CURRENT_LIST_DIR}/../../middleware/wireless/ethermind
...
include_if_use(middleware_edgefast_bluetooth_ble_ethermind_cm7f)
include_if_use(middleware_edgefast_bluetooth_ble_ethermind_lib_cm7f)
include_if_use(middleware_edgefast_bluetooth_br_ethermind_cm7f)
include_if_use(middleware_edgefast_bluetooth_br_ethermind_lib_cm7f)
include_if_use(middleware_edgefast_bluetooth_btble_ethermind_cm7f)
include_if_use(middleware_edgefast_bluetooth_btble_ethermind_lib_cm7f)
include_if_use(middleware_edgefast_bluetooth_common_ethermind)
include_if_use(middleware_edgefast_bluetooth_common_ethermind_hci)
include_if_use(middleware_edgefast_bluetooth_common_ethermind_hci_uart)
include_if_use(middleware_edgefast_bluetooth_config_ethermind)
include_if_use(middleware_edgefast_bluetooth_config_template)
include_if_use(middleware_edgefast_bluetooth_extension_common_ethermind)
include_if_use(middleware_edgefast_bluetooth_k32w061_controller)
include_if_use(middleware_edgefast_bluetooth_mcux_linker_template_ evkmimxrt1064)
include_if_use(middleware_edgefast_bluetooth_pal)
include_if_use(middleware_edgefast_bluetooth_pal_db_gen_ethermind)
include_if_use(middleware_edgefast_bluetooth_pal_host_msd_fatfs_ethermind)
include_if_use(middleware_edgefast_bluetooth_pal_platform_ethermind)
include_if_use(middleware_edgefast_bluetooth_porting)
include_if_use(middleware_edgefast_bluetooth_porting_atomic)
include_if_use(middleware_edgefast_bluetooth_porting_list)
include_if_use(middleware_edgefast_bluetooth_porting_net)
include_if_use(middleware_edgefast_bluetooth_porting_toolchain)
include_if_use(middleware_edgefast_bluetooth_porting_work_queue)
include_if_use(middleware_edgefast_bluetooth_profile_bas)
include_if_use(middleware_edgefast_bluetooth_profile_dis)
include_if_use(middleware_edgefast_bluetooth_profile_fmp)
include_if_use(middleware_edgefast_bluetooth_profile_hps)
include_if_use(middleware_edgefast_bluetooth_profile_hrs)
include_if_use(middleware_edgefast_bluetooth_profile_hts)
include_if_use(middleware_edgefast_bluetooth_profile_ipsp)
include_if_use(middleware_edgefast_bluetooth_profile_pxr)
include_if_use(middleware_edgefast_bluetooth_profile_tip)
include_if_use(middleware_edgefast_bluetooth_profile_wu)
include_if_use(middleware_edgefast_bluetooth_sdio_template_evkmimxrt1064)
include_if_use(middleware_edgefast_bluetooth_shell)
include_if_use(middleware_edgefast_bluetooth_shell_ble)
include_if_use(middleware_edgefast_bluetooth_template)
include_if_use(middleware_edgefast_bluetooth_wifi_nxp_controller_base)...
```

4. Make the following changes in the listed order.

File name	Previous item	New item
config.cmake	MIMXRT1	MIMXRT1064xxxxA
mflash_evkcmimxrt1060	mflash_rt	
1062	1064	
evkcmimxrt1060	evk-mimxrt10	
flags.cmake	1062	1064
6B	6A	
CMakeLists.txt	1062	1064
<rt1064_install_dir>/middleware/edgefast_bluetooth/middleware_edgefast_bluetooth_template.cmake	evkcmimx	evk-mimxrt1064
<rt1064_install_dir>/middleware/wireless/ethermind/middleware_edgefast_bluetooth_common_ethermind_hci_uart.cmake	1062	1064
<rt1064_install_dir>/middleware/wireless/ethermind/middleware_edgefast_bluetooth_k32w061_controller.cmake	1062	1064
<rt1064_install_dir>/middleware/wireless/ethermind/middleware_edgefast_bluetooth_wifi_nxp_controller_base.cmake	evkcmimx	evk-mimxrt1064
<rt1064_install_dir>/boards/evkmimxrt1064/edgefast_bluetooth_examples/middleware_edgefast_bluetooth_mcux_cmake	1062	1064 mxrt1064.
<rt1064_install_dir>/boards/evkmimxrt1064/edgefast_bluetooth_examples/middleware_edgefast_bluetooth_sdio_t cmake	1062	1064 1.

5. Rename MIMXRT1062xxxxx_flexspi_nor.ld as MIMXRT1064xxxxx_flexspi_nor.ld and make the following changes.

```
m_text_start         = 0x60002400;
m_text_size          = 0x007FDC00 - LITTLEFS_REGION_SIZE;

HEAP_SIZE  = DEFINED(__heap_size__)  ? __heap_size__  : 0x1000;
STACK_SIZE = DEFINED(__stack_size__) ? __stack_size__ : 0x0400;
VECTOR_RAM_SIZE = DEFINED(__ram_vector_table__) ? 0x00000400 : 0;

/* Specify the memory areas */
MEMORY
{
  m_flash_config       (RX)  : ORIGIN = 0x60000000, LENGTH = 0x00001000
  m_ivt                (RX)  : ORIGIN = 0x60001000, LENGTH = 0x00001000
  m_interrupts         (RX)  : ORIGIN = 0x60002000, LENGTH = 0x00000400
```

```
m_text_start         = 0x70002400;
m_text_size          = 0x003FDC00 - LITTLEFS_REGION_SIZE;

HEAP_SIZE  = DEFINED(__heap_size__)  ? __heap_size__  : 0x1000;
STACK_SIZE = DEFINED(__stack_size__) ? __stack_size__ : 0x0400;
VECTOR_RAM_SIZE = DEFINED(__ram_vector_table__) ? 0x00000400 : 0;

/* Specify the memory areas */
MEMORY
{
  m_flash_config       (RX)  : ORIGIN = 0x70000000, LENGTH = 0x00001000
  m_ivt                (RX)  : ORIGIN = 0x70001000, LENGTH = 0x00001000
  m_interrupts         (RX)  : ORIGIN = 0x70002000, LENGTH = 0x00000400
```

Parent topic:*Migrate examples from RT1060EVKC to RT1064*

MDK

1. Navigate to <rt1064_install_dir>/boards/evkmimxrt1064/edgefast_bluetooth_examples/peripheral_ht/mdk/.
2. Make the following changes in the listed order.

File name	Previous item	New item
peripheral_ht.uvprojx	1062	1064
	mflash/evkcmimxrt1060	mflash/mimxrt1064
	evkcmimxrt1060	evkcmimxrt1064
	6B	6A

3. Copy evkmimxrt1064_flexspi_nor.ini from <rt1064_install_dir>/boards/evkmimxrt1064/demo_apps/hello_world/mdk/ to <rt1064_install_dir>/boards/evkmimxrt1064/edgefast_bluetooth_examples/peripheral_ht/mdk/.
4. Rename MIMXRT1062xxxxx_flexspi_nor as MIMXRT1064xxxxx_flexspi_nor and make the following changes.

```
43 #define m_flash_config_start          0x60000000
44 #define m_flash_config_size           0x00001000
45
46 #define m_ivt_start                   0x60001000
47 #define m_ivt_size                    0x00001000
48
49 #define m_interrupts_start            0x60002000
50 #define m_interrupts_size             0x00000400
51
52 #define m_text_start                  0x60002400
53 #define m_text_size                   0x007FDC00 - LITTLEFS
```

```
43 #define m_flash_config_start          0x70000000
44 #define m_flash_config_size           0x00001000
45
46 #define m_ivt_start                   0x70001000
47 #define m_ivt_size                    0x00001000
48
49 #define m_interrupts_start            0x70002000
50 #define m_interrupts_size             0x00000400
51
52 #define m_text_start                  0x70002400
53 #define m_text_size                   0x003FDC00 - LITTLEFS
```

Parent topic:*Migrate examples from RT1060EVKC to RT1064*

Note about the source code in the document Example code shown in this document has the following copyright and BSD-3-Clause license:

Copyright 2024 NXP Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Enabling Additional Edgefast BT PAL Examples on M4 core for RT1170

Introduction RT1170 works with two cores: M7 and M4, on which both all EdgeFast examples can run. However, all the EdgeFast examples in the release package are enabled on M7. Only the A2DP source example is enabled on M4.

EdgeFast projects for both the cores share the demo source files but with different project settings. Therefore, the examples can be migrated.

This document describes the steps to migrate EdgeFast examples from M7 to M4 with different toolchains. There are four main steps required. Additionally, you can also delete the function.

1. Create an M4 project
2. Rearrange source files
3. Rearrange project files

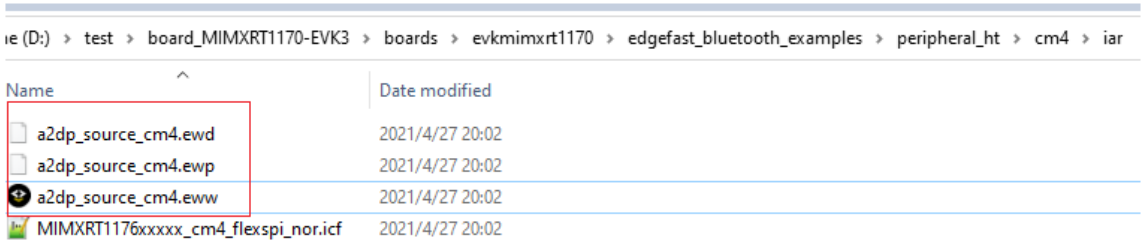
4. Adjust project settings
5. Delete function

In this document, the peripheral_ht example is used to demonstrate how to enable EdgeFast examples on M4 core with IAR and ARMGCC.

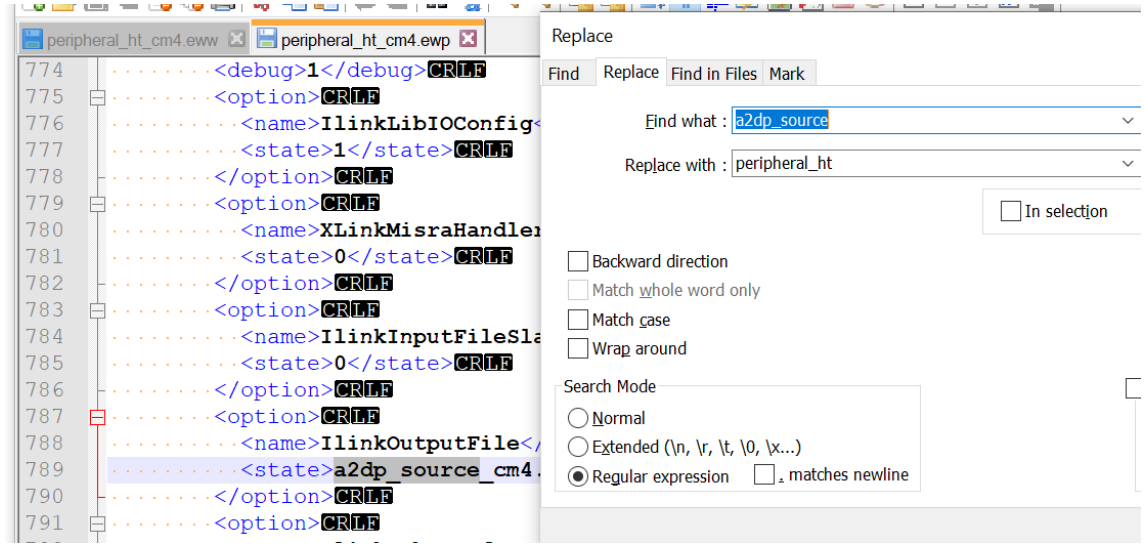
IAR This section describes the steps to create an M4 project with IAR, rearrange source and project files, adjust project settings, and delete function.

Create an M4 project To create an M4 project, perform the following steps:

1. Copy the folder *cm4* in the directory <*install_dir*>*boards*|*evkmimxrt1170*|*edgefast_bluetooth_examples*|*a2dp_* into the folder in which the example should be enabled. In this case, copy the folder *cm4* into the directory <*install_dir*>|*boards*|*evkmimxrt1170*|*edgefast_bluetooth_examples*|*peripheral_ht*.
2. Open the folder *iar* in the directory <*install_dir*>*boards*|*evkmimxrt1170*|*edgefast_bluetooth_examples*|*periph*
3. Rename the files. Change the file name name *a2dp_source_cm4* to *peripheral_ht_cm4* in all the respective files.



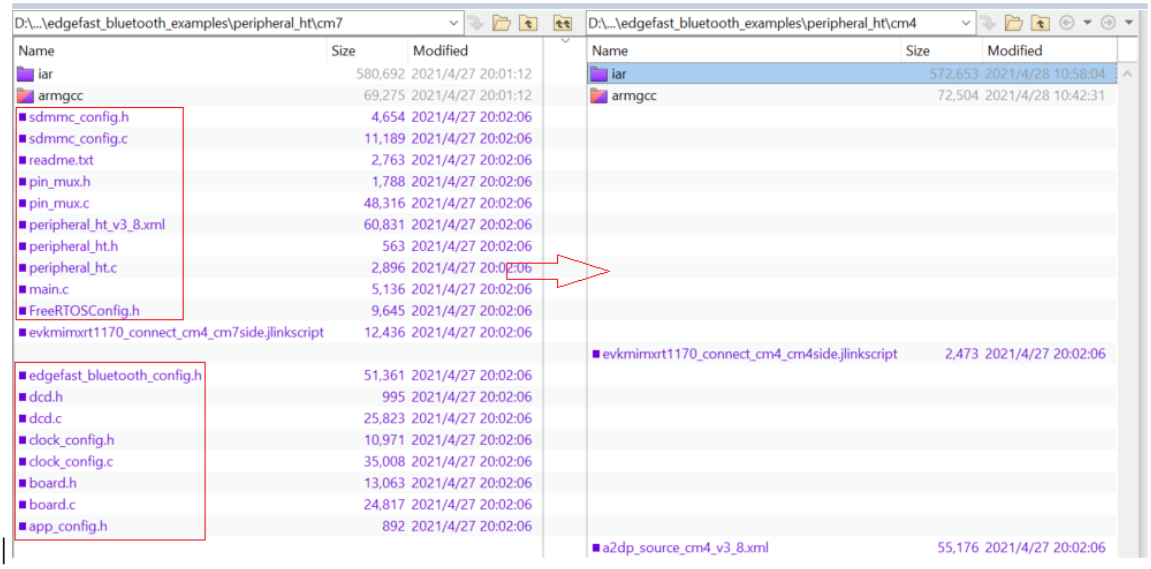
4. Open the files *peripheral_ht_cm4.eww* and *peripheral_ht_cm4.ewp* with a text editor, such as Notepad, Notepad++, Sublime, or Visual Studio Code.
5. Search and replace all *a2dp_source_cm4* with *peripheral_ht _cm4*, and then save the files.



Parent topic:*IAR*

Rearrange source files To rearrange source files, perform the following steps:

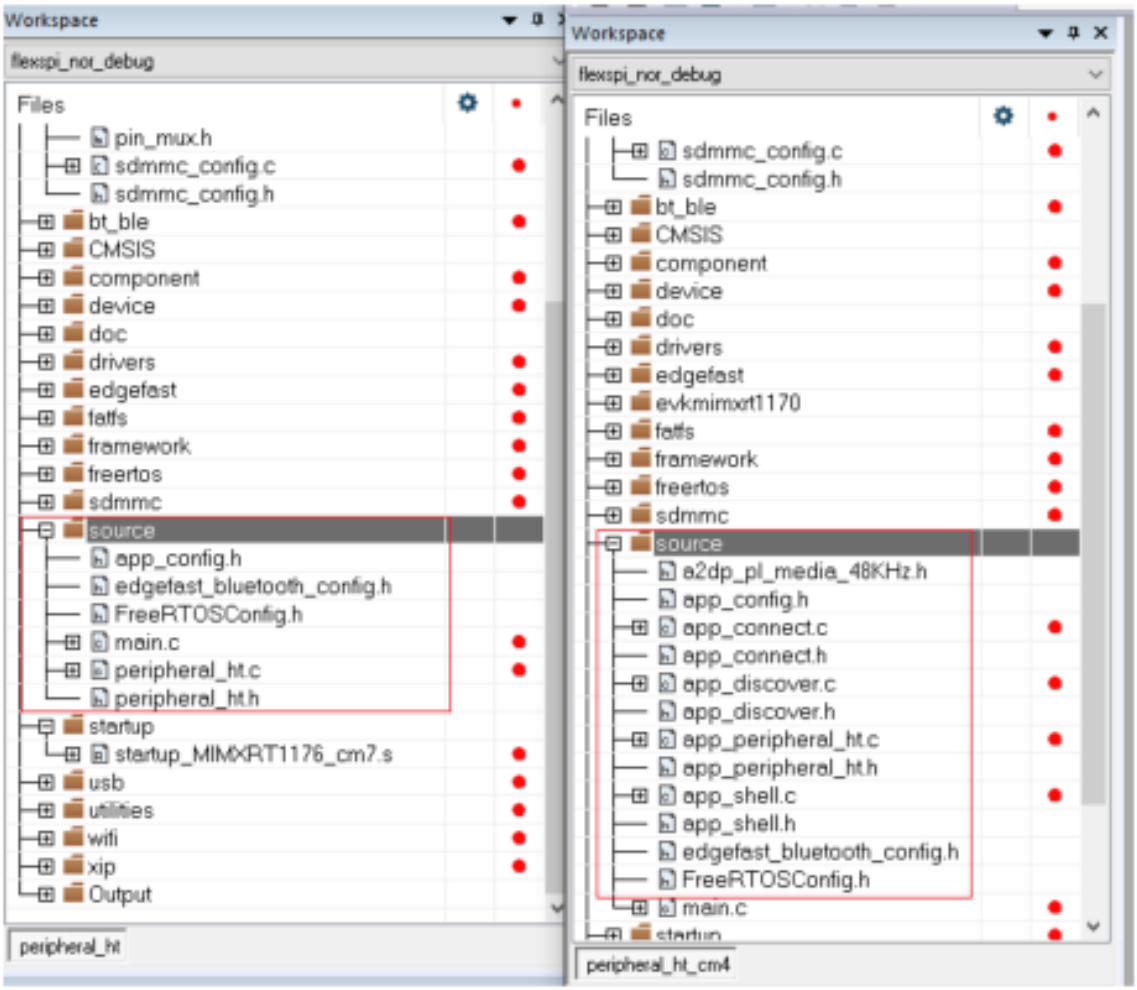
1. Open the folder *cm4* in the directory <*install_dir*> *boards*|*evkmimxrt1170*|*edgefast_bluetooth_examples*|*peri* and delete all files with the extensions *.c and *.h.
2. Copy the files with the extensions *.c and *.h from the folder *boards*|*evkmimxrt1170*|*edgefast_bluetooth_examples*|*peripheral_ht*|*cm7*| to the folder <*install_dir*> *boards*|*evkmimxrt1170*|*edgefast_bluetooth_examples*|*peripheral_ht*|*cm4*.



Parent topic:*IAR*

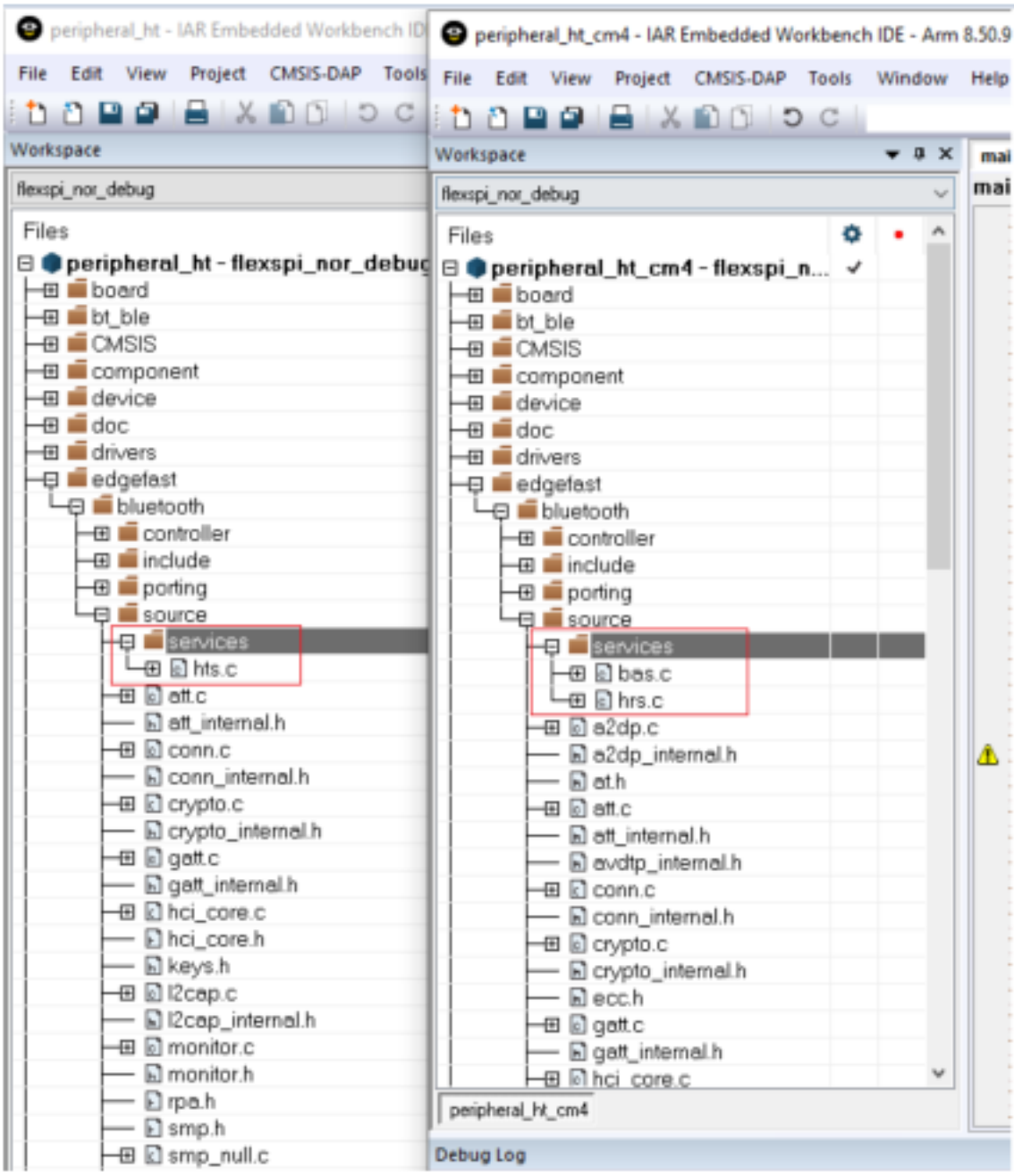
Rearrange project files To rearrange project files, perform the following steps:

1. Open the *peripheral_ht _cm7*and *peripheral_ht _cm4* IAR projects in the directories <*install_dir*> *boards*|*evkmimxrt1170*|*edgefast_bluetooth_examples*|*peripheral_ht* |*cm7*|*iar* and <*install_dir*> *boards*|*evkmimxrt1170*|*edgefast_bluetooth_examples*|*peripheral_ht* |*cm4*|*iar*.
    1. Compare the whole project directory, find file groups that the cm7 project has but are missing in the cm4 project. Add the missing file groups from the cm7 project into the cm4 project.
    2. Compare the difference between the two groups with the same name. Remove files that do not exist in the cm7 project but exist in the cm4 project. Find files that are available in the cm7 project but are missing in the cm4 project. Add the missing files from the cm7 project into the cm4 project.
2. For example, in the following figure, the files in the source group in the cm4 project must be removed, and the files in the path: <*install_dir*>|*boards*|*evkmimxrt1170*|*edgefast_bluetooth_examples*|*peripheral_ht* with the same name as the files in the cm7 project must be added into the *source* group.



3. Compare the *services* group.

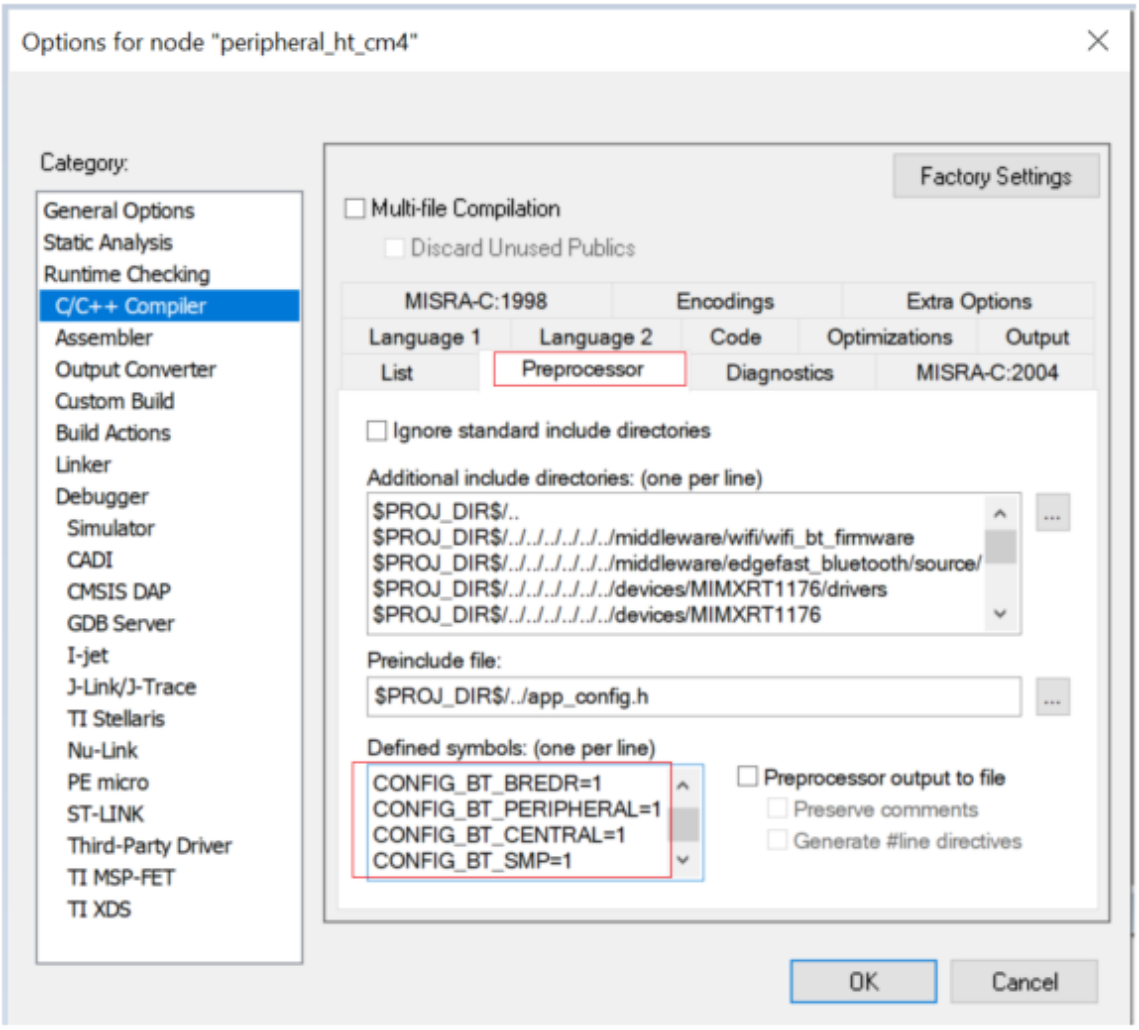
   The peripheral hts profile is in the *services* folder. Add the hts.c file to the *services* group of the *cm4* folder.



Parent topic:*IAR*

Adjust project settings To adjust the project settings, perform the following steps:

1. Compare the macro in the project settings: **Option** > **C/C++ compiler** > **Preprocessor**.
2. Find the macros that do not exist in the **cm4** project but are available in the **cm7** project. Delete these macro. The rule is that **m7** macro setting should be same with **m4**.



|

|

The macros are in the**peripheral_ht_cm4.ewp** file.

|

|

Parent topic:*IAR*

Delete function As a final step, remove the function *"SCB_DisableDCache()*; in main.c.

On the completion of the above steps, the M7 project successfully migrates to an M4 project. You can now download and debug the M4 example project.

Parent topic:*IAR*

Arm GCC This section describes the steps to create an M4 project with Arm GCC, rearrange source and project files, adjust project settings, and delete function.

Create an M4 project To create an M4 project, perform the following steps:

1. Copy the folder <*install_dir*> *boards\evkmimxrt1170\edgefast_bluetooth_examples\a2dp_source\cm4* into another folder in which the example should be enabled. In this case, copy the folder <*install_dir*>

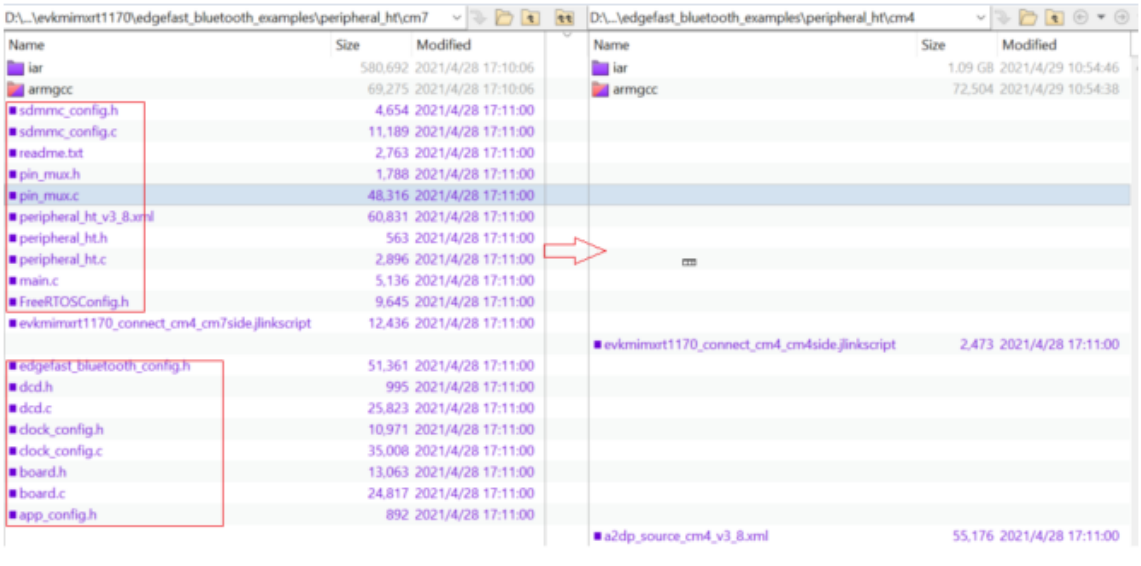
boards\evkmimxrt1170\edgefast_bluetooth_examples\a2dp_source \cm4 into*<install_dir>boards\evkmimxrt1170\edgefast_bluetooth_examples\peripheral_ht\cm4*.

2. Open the file *CMakeLists.txt* located in the path: *<install_dir>boards\evkmimxrt1170\edgefast_bluetooth_examples\peripheral_ht\cm4\ armgcc*.
3. Search and replace all *a2dp_source_cm4* with *peripheral_ht_cm4*, and then save the files.

Parent topic:*Arm GCC*

Rearrange source files To rearrange source files, perform the following steps:

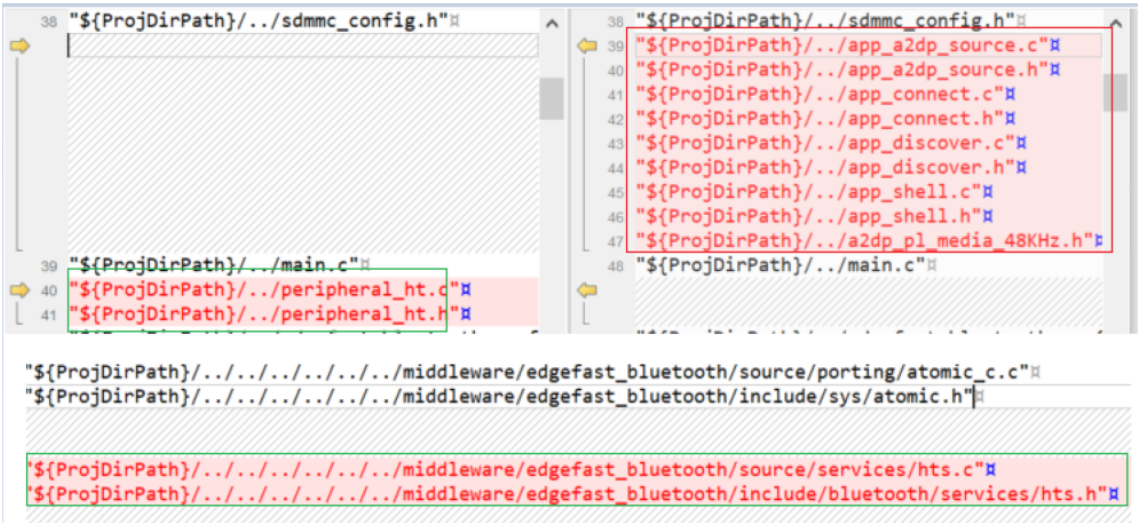
1. Open the folder *<install_dir>boards\evkmimxrt1170\edgefast_bluetooth_examples\peripheral_ht\cm4* and delete all files with the extensions *.c and *.h.
2. Copy the files with the extensions *.c and *.h in the folder *<install_dir>boards\evkmimxrt1170\edgefast_bluetooth_examples\peripheral_ht\cm7* to the folder *<install_dir>boards\evkmimxrt1170\edgefast_bluetooth_examples\peripheral_ht\cm4*.



Parent topic:*Arm GCC*

Rearrange project files To rearrange project files, perform the following steps:

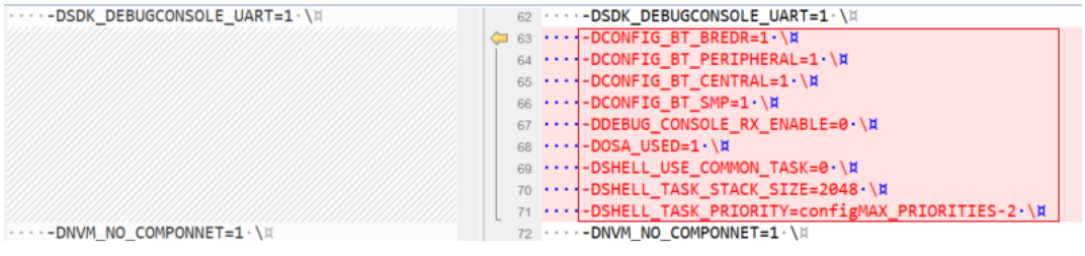
1. Open the *CMakeLists.txt* of the two examples respectively. The two files are in the *<install_dir>boards\evkmimxrt1170\edgefast_bluetooth_examples\peripheral_ht\cm7\armgcc* and *<install_dir>boards\evkmimxrt1170\edgefast_bluetooth_examples\peripheral_ht\cm4\armgcc* folders respectively.
2. Search the section *add_executable*. Compare the difference between the two sections. Remove files that do not exist in the *cm7* project but are available in the *cm4* project. Add the files that exist in the *cm7* project but are not available in the *cm4* project into the *cm4* project. For example, in the following figure, the files in the red box should be removed and the files in the green box must be added into the *cm4* project.



Parent topic:*Arm GCC*

Adjust project setting To adjust the project settings, perform the following steps:

1. Open the *flags.cmake* of the two examples respectively. The two files are in the <*install_dir*>*boards**evkmimxrt1170**edgefast_bluetooth_examples**peripheral_ht**cm7**armgcc* and <*install_dir*>*boards**evkmimxrt1170**edgefast_bluetooth_examples**peripheral_ht**cm4**armgcc* folders respectively.
2. Search the ***CMAKE_C_FLAGS_DEBUG*** section.
    1. Compare the macro between the two sections.
    2. Add the macros that do not exist in the **cm4** project but are available in the **cm7** project into the cm4 project. The rule is that macro setting should be same.
    3. Delete the macros highlighted in the red rectangle.



Parent topic:*Arm GCC*

Delete function As a final step, remove the function *"SCB_DisableDCache()* in main.c.

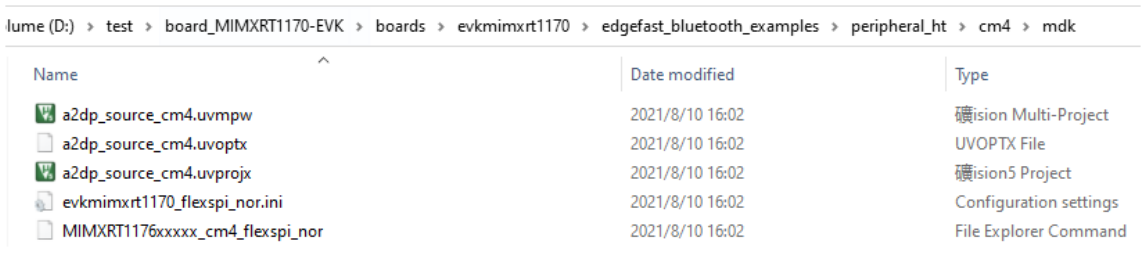
On the completion of the above steps, the M7 project successfully migrates to an M4 project. You can now download and debug the M4 example project.

Parent topic:*Arm GCC*

MDK This section describes the steps to create an M4 project with MDK, rearrange source and project files, adjust project settings, and delete function.

Create an M4 project

1. Copy folder *cm4* from <*install_dir*>*boards*|*evkmimxrt1170*|*edgefast_bluetooth_examples*|*a2dp_source*|*cm4* into the folder in where the example must be enabled. In this case, copy folder *cm4* into directory <*install_dir*>|*boards*|*evkmimxrt1170*|*edgefast_bluetooth_examples*|*peripheral_ht*.
2. Open folder mdk from <*install_dir*>*boards*|*evkmimxrt1170*|*edgefast_bluetooth_examples*|*peripheral_ht*|*cm4*

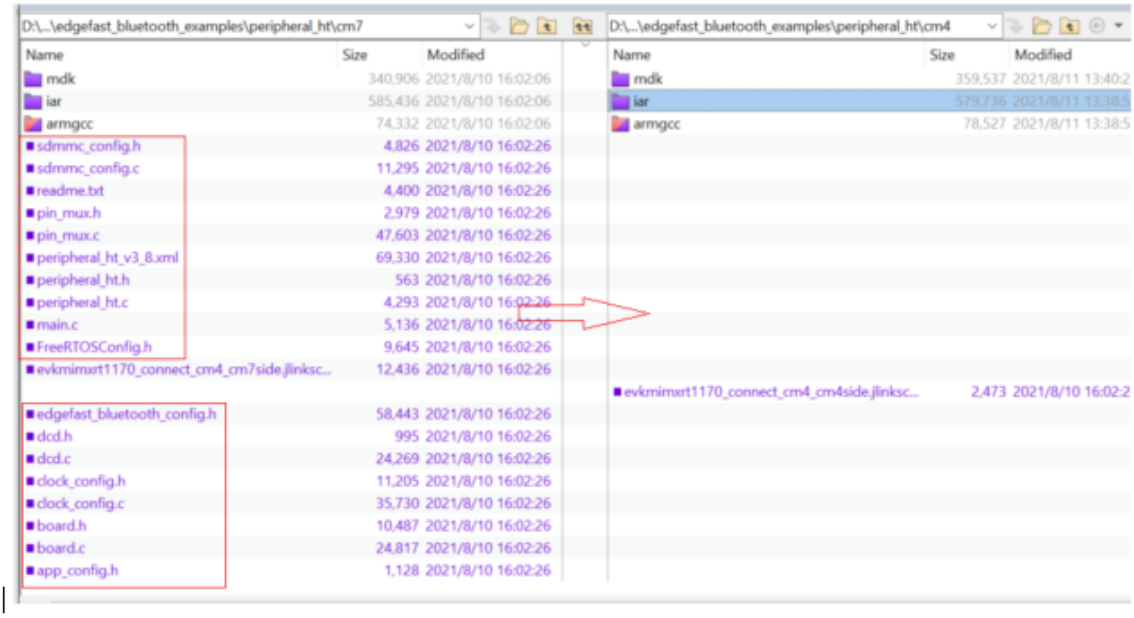


3. Change the filename *a2dp_source_cm4* to *peripheral_ht_cm4* respectively.
4. Open the files *peripheral_ht_cm4.**uvmpw* and *peripheral_ht_cm4.* *uvoptx*, *peripheral_ht_cm4.uvprojx*with a text editor, such as Notepad, Notepad++, Sublime, or Visual Studio code.
5. Search and replace *a2dp_source_cm4* with *peripheral_ht _cm4*, and then save the files.

Parent topic:*MDK*

Rearrange source files

1. Open folder *cm4* in *<install_dir>**boards*|*evkmimxrt1170*|*edgefast_bluetooth_examples*|*peripheral_ht*|*cm4*, and delete all files with the .c and .h file name extension.
2. Copy files with the .c and .h filename extension in folder *cm7* with directory <*install_dir*> *boards*|*evkmimxrt1170*|*edgefast_bluetooth_examples*|*peripheral_ht*|*cm7*to folder *cm4* with directory <*install_dir*> *boards*|*evkmimxrt1170*|*edgefast_bluetooth_examples*|*peripheral_ht*|*cm4*.

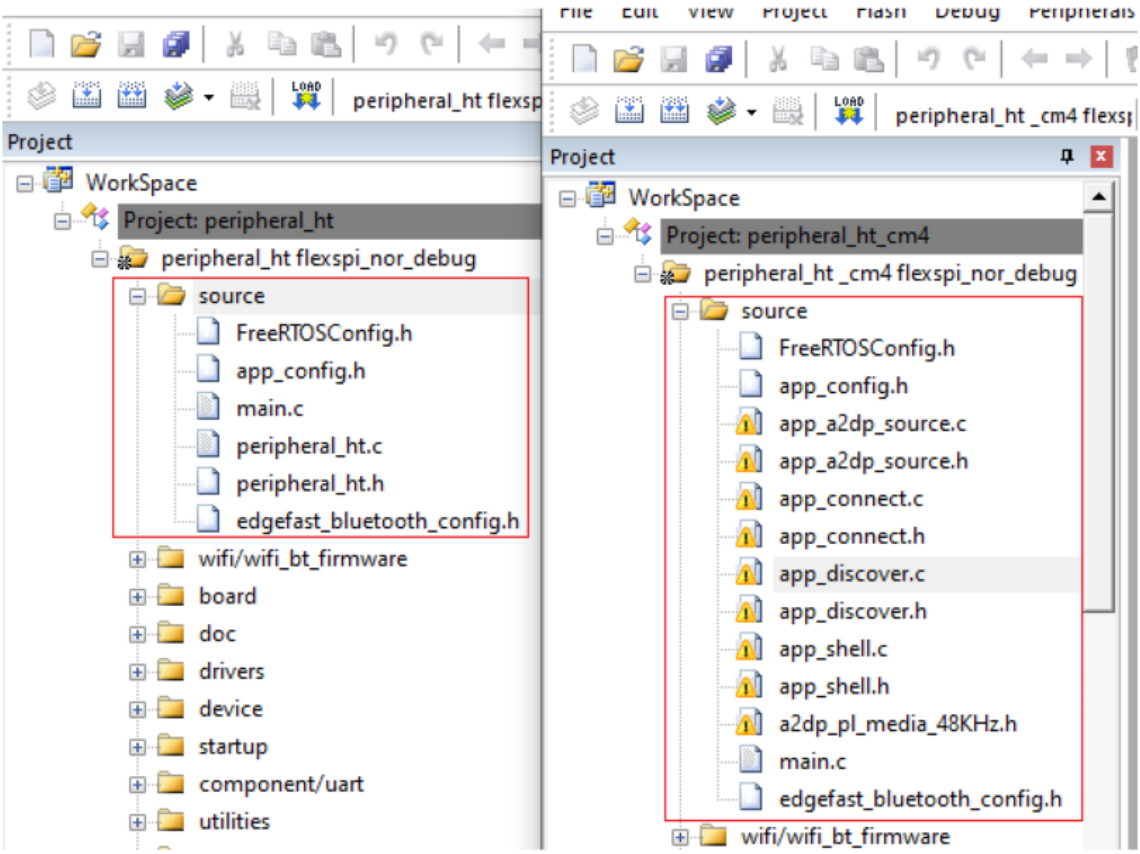


.

Parent topic:*MDK*

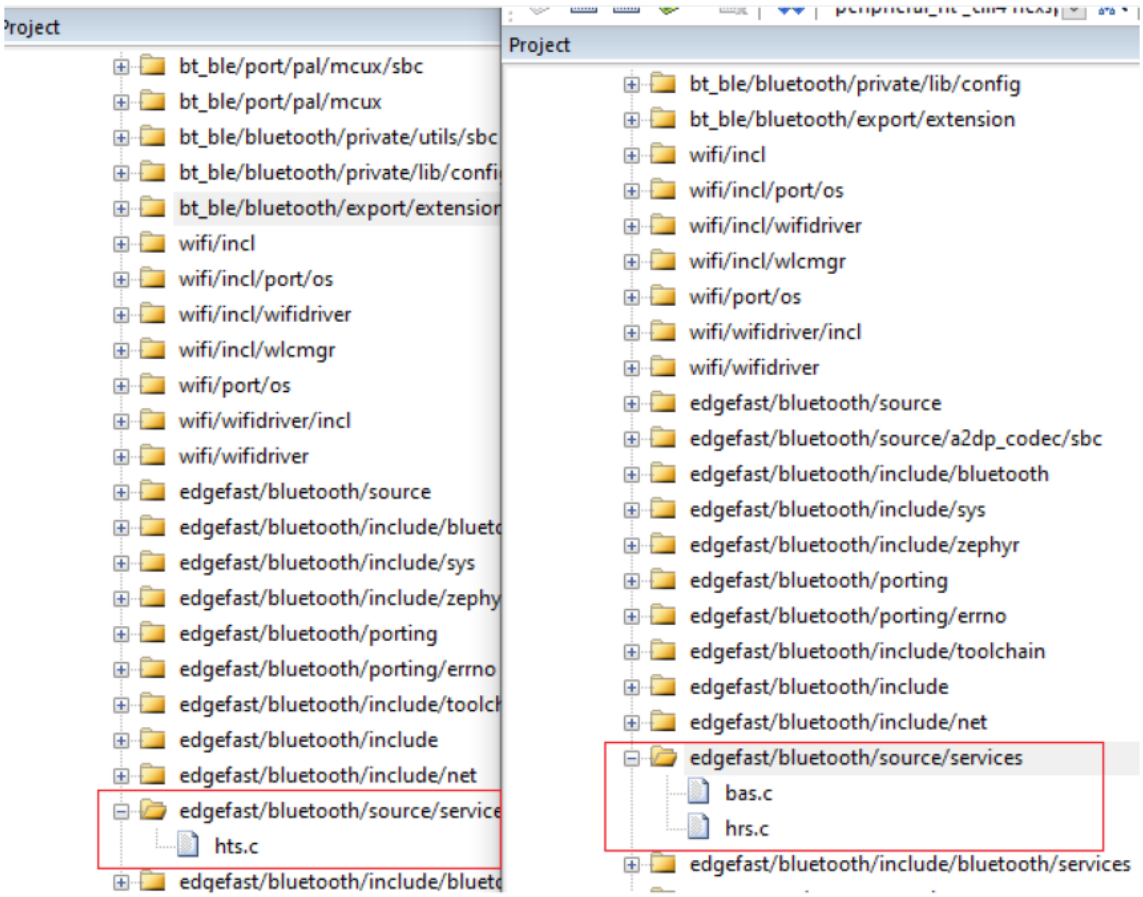
Rearrange project files

1. Open the *peripheral_ht _cm7* and *peripheral_ht _cm4* IAR projects. The two workspaces are located in *<install_dir>**boards**evkmimxrt1170**edgefast_bluetooth_examples**peripheral_ht* *cm7**mdk* and *<install_dir>**boards**evkmimxrt1170**edgefast_bluetooth_examples**peripheral_ht* *cm4**mdk* respectively.
   - Compare the whole project directory, find file groups that the cm7 project has but the cm4 project not and then add these groups into the cm4 project.
   - Compare the difference between the two groups with the same name, remove files that do not exist in the cm7 project but exist in the cm4 project; find files that the cm7 project has but the cm4 project not and then add these files into the cm4 project.
2. For the *source* group, in this case, the files in the source group in the cm4 project must be removed, and the files in the path *<install_dir>**boards**evkmimxrt1170*\ *boards**evkmimxrt1170**edgefast_bluetooth_examples**peripheral_ht**cm4* with the same name as the files in the cm7 project must be added into the *source* group.



3. Compare the **service: group**.

   Peripheral hts profile is located in "**service**" folder. Add the hts.c file to the services group of the cm4 folder.

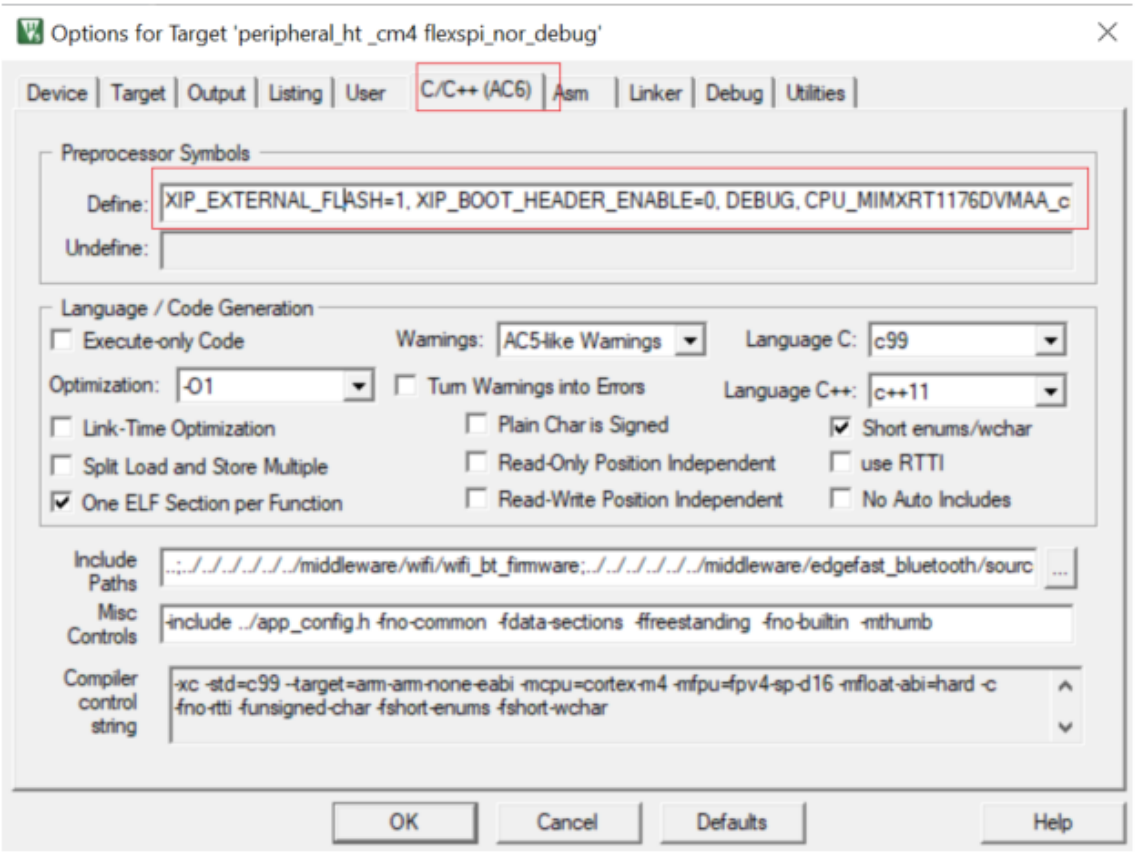


.

Parent topic:*MDK*

Adjust project settings

1. Compare the macro in the project settings: *preprocessor symbols*.
2. Compare the macro that does exist in the cm4 project but exists in the cm7 project.
3. Delete the following macro. The rule is that m7 macro setting should be same as m4 .
   The macro could also be found in be eripheral_ht_cm4.uvprojx.



|

|

|

Parent topic:*MDK*

Delete function Remove function *SCB_DisableDCache();* in *main.c*.

On successful completion of the above steps, the M7 project is changed to the M4 project. You can now download and debug the M4 example project.

Parent topic:*MDK*

Note The above steps are based on the **a2dp_source** example and help enable the **peripheral_ht** example on the **m4** core. You can use the same steps for other examples and migrate them from an **m7** project to an **m4** project.

Chapter 3

RTOS

3.1 FreeRTOS

3.1.1 FreeRTOS kernel

Open source RTOS kernel for small devices.

FreeRTOS kernel for MCUXpresso SDK Readme

FreeRTOS kernel for MCUXpresso SDK ChangeLog

FreeRTOS kernel Readme

3.1.2 FreeRTOS drivers

This is set of NXP provided FreeRTOS reentrant bus drivers.

3.1.3 backoffalgorithm

Algorithm for calculating exponential backoff with jitter for network retry attempts.

Readme

3.1.4 corehttp

C language HTTP client library designed for embedded platforms.

3.1.5 corejson

JSON parser.

Readme

3.1.6 coremqtt

MQTT publish/subscribe messaging library.

3.1.7 coremqtt-agent

The coreMQTT Agent library is a high level API that adds thread safety to the coreMQTT library.

Readme

3.1.8 corepkcs11

PKCS #11 key management library.

Readme

3.1.9 freertos-plus-tcp

Open source RTOS FreeRTOS Plus TCP.

Readme