FreeRTOS BSP i.MX 6SoloX API Reference Manual

NXP Semiconductors

Document Number: FRTOS6XAPIRM
Rev. 0
Jul 2016

Contents

Contents

Contents

Contents

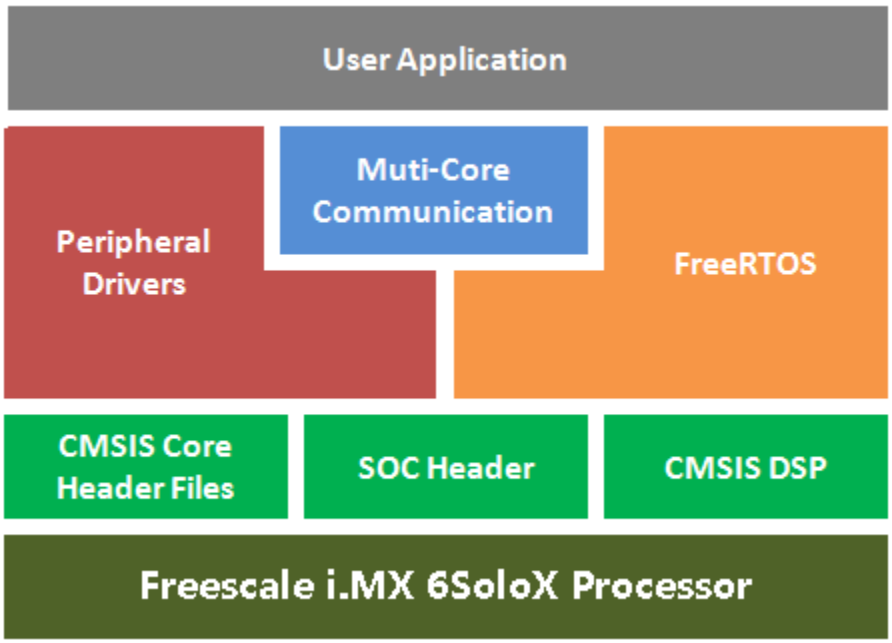
Contents

Chapter 1
Introduction

The FreeRTOS BSP for i.MX 6SoloX is a suite of robust peripheral drivers, FreeRTOS OS support, and multicore communication stack designed to simplify and accelerate application development on the i.MX 6SoloX Processor.

Included in the FreeRTOS BSP for i.MX 6SoloX is a full source code under a permissive open-source license for all demos, examples, RTOS, middleware, and peripheral driver software.

The FreeRTOS BSP for i.MX 6SoloX consists of the following runtime software components fully written in C:



- ARM® CMSIS Core, DSP standard libraries, and CMSIS-compliant device header files which provide direct access to the peripheral registers and bits.
- A set of peripheral drivers that provides a simple, stateless way to encapsulate register access of the peripherals.
- Multicore communication stack — RPMsg to facilitate the development of multicore communication.
- FreeRTOS operating system to provide an event driven preemptive scheduling RTOS.

The FreeRTOS BSP for i.MX 6SoloX comes complete with software examples demonstrating the usage of the peripheral drivers, middleware, and FreeRTOS operating system. All examples are provided with projects for the following toolchains:

- ARM DS-5

- GNU toolchain for ARM Cortex® -M with Cmake build system
- IAR Embedded Workbench®

The configurable items for each driver, at all levels, are encapsulated into C language data structures. Peripheral driver, board and FreeRTOS configuration is not fixed and can be changed according to the application.

The example applications demonstrate how to configure the drivers by passing configuration data to the APIs.

The organization of files in the FreeRTOS BSP for i.MX 6SoloX release package is focused on ease-of-use. The FreeRTOS BSP for i.MX 6SoloX folder hierarchy is organized at the top level with these folders:

Deliverable	Location
Examples	<install_dir>/examples/...
Demo applications	<install_dir>/examples/<board_name>/demo_apps/...
Driver examples	<install_dir>/examples/<board_name>/driver_examples/...
Docmumentations	<install_dir>/doc/...
Middleware	<install_dir>/middleware/...
Peripheral Driver, Startup Code and Utilities	<install_dir>/platform/...
Cortex Microcontroller Software Interface Standard (CMSIS) ARM Cortex®-M header files, DSP library source and lib files.	<install_dir>/platform/CMSIS/...
Processor header file	<install_dir>/platform/devices/<device_name>/include/...
Linker script for each supported toolchain	<install_dir>/platform/devices/<device_name>/linker/...
CMSIS compliant Startup Code	<install_dir>/platform/devices/<device_name>/startup/...
Peripheral Drivers	<install_dir>/platform/drivers/...
Utilities such as debug console	<install_dir>/platform/utilities/...
FreeRTOS Kernel Code	<install_dir>/rtos/FreeRTOS/...
External useful tools	<install_dir>/tools/...

The other sections of this document describes the API functions for Peripheral drivers.

Chapter 2
Architectural Overview

This chapter provides the architectural overview for the FreeRTOS BSP for i.MX 6SoloX. It describes each layer within the architecture and its associated components.

Overview

The FreeRTOS BSP for i.MX 6SoloX architecture consists of six key components listed below.

1. The ARM Cortex Microcontroller Software Interface Standard (CMSIS) core-compliant device-specific header files, SoC Header, and CMSIS DSP libraries.
2. Peripheral Drivers
3. Real-time Operating System (RTOS) — FreeRTOS OS
4. Board-specific configuration
5. Multicore communication stack integrated with FreeRTOS BSP for i.MX 6SoloX
6. Applications based on the FreeRTOS BSP architecture

i.MX Processor header files

The FreeRTOS BSP contains CMSIS-compliant device-specific header files which provide direct access to the i.MX Processor peripheral registers. Each supported i.MX device in FreeRTOS BSP has an overall System-on-Chip (SoC) memory-mapped header file. This header file contains the memory map and register base address for each peripheral and the IRQ vector table with associated vector numbers. The overall SoC header file provides an access to the peripheral registers through pointers and predefined masks.

Along with the SoC header files, the FreeRTOS BSP also includes common CMSIS header files for the ARM Cortex-M core and DSP library from the latest CMSIS release. The CMSIS DSP library source code is also included for reference. These files and the above mentioned header files can all be found in the FreeRTOS BSP platform/CMSIS directory.

Peripheral Drivers

The FreeRTOS BSP Peripheral Drivers consists of low-level drivers for the i.MX Series Processor on-chip peripherals. The main goal of this part is to abstract the hardware peripheral register accesses into a set of stateless basic functional operations. The Peripheral Driver itself can be used to build application-specific logic or as building blocks for use-case-driven high-level Drivers. It primarily focuses on the functional control, configuration, and realization of basic peripheral operations. The Peripheral Driver hides register access details and various processor peripheral instantiation differences so that, either an application or high-level Drivers, can be abstracted from the low-level hardware details. Therefore, the hardware peripheral must be accessed through a Peripheral Driver.

The Peripheral Drivers cover the most useful basic APIs for embedded system developers and provide easy-to-use initialization functions. The initialization functions initialization data structure consist of all the necessary parameters to bring Peripherals into use. For example, the UART Driver initialization data structure includes the baud rate, data bits number, number of stop bits, parity error check mode and data transfer direction. Essentially, the Peripheral Driver functional boundary is limited by the peripheral itself.

There is one Peripheral Driver for each peripheral and the Peripheral Driver only accesses the features available within the peripheral. In addition, the Peripheral Driver does not define interrupt service routine entries or support interrupt handling. These tasks must be handled by a high-level Driver or by the user application.

The Peripheral Drivers can be found in the platform/drivers directory.

Design Guidelines

This section summarizes the design guidelines that were used to develop the Peripheral Drivers. It is meant for information purposes and provides more details on the make-up of the Peripheral Drivers. As previously stated, the main goal of the Peripheral Driver is to abstract the hardware details and provide a set of easy-to-use low-level drivers. The Peripheral Driver itself can be used directly by the user application or as building blocks of high-level transaction drivers. The Peripheral Driver is mainly focused on individual functional primitives and makes no assumption of use cases. The Peripheral Driver APIs follow a naming convention that is consistent from one peripheral to the next. This is a summary of the design guidelines used when developing the Peripheral Driver drivers:

- There is a dedicated Peripheral Driver for each individual peripheral.
- Each Peripheral Driver has an initialization function to put the peripheral into a ready-to-use state.
- Each Peripheral Driver has an enable and disable function to enable or disable the peripheral module.
- The Peripheral Driver does not have an internal operation context and should not dynamically allocate memory.
- The Peripheral Driver must remain stateless.
- The Peripheral Driver does not depend on any other software entities or invoke any functions from other peripherals.

The CMSIS startup code contains the vector table with the ISR entry names which are defined as weak symbols. An application is able to redefine the ISR functions with same names to replace the weak references defined in the vector table.

Demo Applications

The Demo Applications in the FreeRTOS BSP provide examples, which show how to build user applications using the FreeRTOS BSP framework. The Demo Applications can be found in the FreeRTOS BSP top-level examples directory. The FreeRTOS BSP includes two types of demo applications:

- Driver Examples that demonstrate the usage of the Peripheral Drivers.
- Demo Applications that provide a reference design based on the features available on the target i.-MX Processor and its evaluation boards. This demo is targeted to highlight a certain feature of the SoC for its intended usage, and/or to provide turnkey references using the FreeRTOS BSP peripheral driver library with other integrated software components, such as RPMsg .

Board Configuration

The FreeRTOS BSP drivers make no assumption on board-specific configurations nor do the drivers configure pin muxing, which are part of the board-specific configuration. The FreeRTOS BSP provides board-configuration files that configure pin muxing for applications, clock gating for on-chip peripherals, Resource Domain Controller setting, FreeRTOS OS feature customize setting and functions that can be called before driver initialization. Board Configuration also includes a set of APIs which can help to get the current clock frequency of a peripheral in real-time.

- Pin Muxing configuration is used to set the IOMUX connection between dedicated pins and peripherals.
- Clock settings of the board configuration vary from demo to demo. The clock for peripheral is off by default and should be opened in the demo application's hardware_init file.
- Resource Domain Controller setting varies from demo to demo. The peripheral used by Cortex-M4 core can be set as monopolized or shared with ARM Cortex-A9 Core. The RDC configuration is also located in the hardware_init file of certain demo/application.
- FreeRTOS feature customize settings include settings to maximize kernel performance and functionality or to minimize code size.
- Peripheral clock frequency acquisition function can calculate peripheral clock frequency according to current clock configuration. It can hide the hardware complexity from the user.
- Board Configuration also includes some useful functions such as debug console initialization and so on.

These board-configuration files can be found in the examples/<board_name> directory.

FreeRTOS Operating System

The FreeRTOS BSP drivers are designed to work with or without an RTOS. The FreeRTOS OS provides a common set of service routines for integrated software solutions, and upper-level applications.

Middleware integration

FreeRTOS BSP also integrates multi-core communication stack RPMsg, to offer a complete, easy-to-use, software development kit for the i.MX Processor users who have a need to transfer data between Cortex--A9 an Cortex-M4 Cores.

Memory Division

This section discuss the FreeRTOS BSP demo/example build target location in the memory map and how to build an application/demo for other storage devices.

The demos/examples in FreeRTOS BSP are built for on-chip Tightly Coupled Memory (TCM) in the ARM Cortex-M4 core by default. Running at TCM gives the application the best performance. However, the TCML used to store application image has a 32 KB capacity limit. To overcome this drawback, the user should move the application to a different position in the memory map, for example OCRAM, DDR or QSPI flash.

To build a demo/example image for other memory space, the user doesn't need to change the source code of the demo/example. Only the linker script of the demo/example should be changed. The linker script for i.MX processor device is located in the <install_dir>/platform/devices/<device_name>/linker/<toolchain>. When making modifications, note the following:

- The modified memory space must be a legal space to boot the ARM Cortex-M4 Core.
- The modified memory space must not be occupied by the ARM Cortex-A9 Core.

Chapter 3
Analog-to-Digital Convert (ADC)

3.1 Overview

The FreeRTOS BSP provides a driver for the Analog-to-Digital Converter (ADC) block of i.MX devices.

Modules

- ADC driver

3.2 ADC driver

3.2.1 Overview

This chapter describes the programming interface of the ADC driver (platform/drivers/inc/adc_imx6sx.h).

The ADC has a Analog-to-Digital Converter with selectable conversion resolution from 8 bits to 12 bits and an interrupt of conversion complete is generated if interrupt of the ADC instance is enabled. The ADC driver provides a set of APIs to provide these services:

- Analog-to-Digital conversion property setting, include conversion speed, resolution and duration
- Analog-to-Digital Converter calibration
- Basic single conversion and continuous conversion
- Conversion result compare with programable condition and interrupt generation
- DMA management
- Interrupt management

3.2.2 ADC Driver model building

The ADC driver has two parts:

- Basic Converter - This part handles the mechanism that converts the external analog voltage to a digital value. API functions configure the converter.
- Advance Feature Group - The advanced feature group covers optional features for applications. Some of these features are already implemented by hardware, such as the hardware average, hardware compare, different power and speed mode. Although these features are optional, they are recommended to ensure that the ADC performs better, especially for calibration.

3.2.3 ADC Initialization

To initialize the ADC module, define an valid adc_init_config_t type variable and pass it to the ADC_Init() function. For example, to use the ADC1 module, pass the ADC1 base pointer and a pointer pointing to the adc_init_config_t structure.

1. Use the ADC_Init() function to set ADC module sample rate , sample resolution and hardware average times.

```
void ADC_Init(ADC_Type* base, const adc_init_config_t* initConfig)
```

2. Choose the ADC operation mode with ADC conversion control functions or ADC comparer control functions.

```
For SW trigger conversion:
User need disable continuous conversion by calling ADC_SetConvertCmd() and
calling ADC_TriggerSingleConvert() when ADC convert action is needed.

For continuous conversion:
User just need enable continuous conversion by calling ADC_SetConvertCmd().
The ADC module keeps convert action on selected channel with given property.
```

```
Conversion and Hardware Comparer control functions:
void ADC_SetConvertCmd(ADC_Type* base, uint8_t channel, bool enable)
void ADC_TriggerSingleConvert(ADC_Type* base, uint8_t channel)
void ADC_SetCmpMode(ADC_Type* base, uint8_t cmpMode, uint16_t cmpVal1, uint16_t cmpVal2)
```

3. Finally, set an interrupt mode with interrupt and flag control functions or DMA control functions.

```
void ADC_SetIntCmd(ADC_Type* base, bool enable)
uint32_t ADC_GetStatusFlag(ADC_Type* base, uint32_t flags)
void ADC_ClearStatusFlag(ADC_Type* base, uint32_t flags)
void ADC_SetDmaCmd(ADC_Type* base, bool enable)
```

4. For ADC self-calibration, use ADC calibration functions.

```
void ADC_SetCalibration(ADC_Type* base, bool enable)
uint8_t ADC_GetCalibrationResult(ADC_Type* base)
```

5. For a low-power performance, use ADC low-power control functions.

```
void ADC_SetPowerMode(ADC_Type* base, uint8_t powerMode)
```

3.2.4 ADC Get Result

Use ADC_GetConvertResult() to get conversion results.

```
uint16_t ADC_GetConvertResult(ADC_Type* base)
```

Data Structures

- struct adc_init_config_t
  ADC module initialize structure. More...

Enumerations

- enum _adc_average_number {
  adcAvgNum4 = 0U,
  adcAvgNum8 = 1U,
  adcAvgNum16 = 2U,
  adcAvgNum32 = 3U,
  adcAvgNumNone = 4U }
  ADC hardware average number.
- enum _adc_convert_trigger_mode {
  adcSoftwareTrigger = 0U,
  adcHardwareTrigger = 1U }
  ADC conversion trigger select.
- enum _adc_convert_speed_config {
  adcNormalSpeed = 0U,
  adcHighSpeed = 1U }

ADC conversion speed configure.

- enum _adc_sample_time_duration {
  adcSamplePeriodClock2,
  adcSamplePeriodClock4,
  adcSamplePeriodClock6,
  adcSamplePeriodClock8,
  adcSamplePeriodClock12,
  adcSamplePeriodClock16,
  adcSamplePeriodClock20,
  adcSamplePeriodClock24 }
  ADC sample time duration.
- enum _adc_power_mode {
  adcNormalPowerMode = 0U,
  adcLowPowerMode = 1U }
  ADC low power configure.
- enum _adc_resolution_mode {
  adcResolutionBit8 = 0U,
  adcResolutionBit10 = 1U,
  adcResolutionBit12 = 2U }
  ADC conversion resolution mode.
- enum _adc_clock_divide {
  adcInputClockDiv1 = 0U,
  adcInputClockDiv2 = 1U,
  adcInputClockDiv4 = 2U,
  adcInputClockDiv8 = 3U }
  ADC input clock divide.
- enum _adc_clock_source {
  adcIpgClock = 0U,
  adcIpgClockDivide2 = 1U,
  adcAsynClock = 3U }
  ADC clock source.
- enum _adc_compare_mode {
  adcCmpModeLessThanCmpVal1,
  adcCmpModeGreaterThanCmpVal1,
  adcCmpModeOutRangNotInclusive,
  adcCmpModeInRangNotInclusive,
  adcCmpModeInRangInclusive,
  adcCmpModeOutRangInclusive,
  adcCmpModeDisable }
  ADC comparer work mode configuration.
- enum _adc_general_status_flag {
  adcFlagAsynWakeUpInt = 1U << 0,
  adcFlagCalibrateFailed = 1U << 1,
  adcFlagConvertActive = 1U << 2 }
  ADC general status flag.

ADC Module Initialization and Configuration Functions.

- void ADC_Init (ADC_Type *base, const adc_init_config_t *initConfig)
  Initialize ADC to reset state and initialize with initialize structure.
- void ADC_Deinit (ADC_Type *base)
  This function reset ADC module register content to its default value.
- void ADC_SetConvertResultOverwrite (ADC_Type *base, bool enable)
  Enable or disable ADC module overwrite conversion result.
- void ADC_SetConvertTrigMode (ADC_Type *base, uint8_t mode)
  This function set ADC module conversion trigger mode.
- static uint8_t ADC_GetConvertTrigMode (ADC_Type *base)
  This function is used to get conversion trigger mode.
- void ADC_SetConvertSpeed (ADC_Type *base, uint8_t mode)
  This function set ADC module conversion speed mode.
- static uint8_t ADC_GetConvertSpeed (ADC_Type *base)
  This function get ADC module conversion speed mode.
- void ADC_SetSampleTimeDuration (ADC_Type *base, uint8_t duration)
  This function set ADC module sample time duration.
- void ADC_SetPowerMode (ADC_Type *base, uint8_t powerMode)
  This function set ADC module power mode.
- static uint8_t ADC_GetPowerMode (ADC_Type *base)
  This function get ADC module power mode.
- void ADC_SetClockSource (ADC_Type *base, uint8_t source, uint8_t div)
  This function set ADC module clock source.
- void ADC_SetAsynClockOutput (ADC_Type *base, bool enable)
  This function enable asynchronous clock source output regardless of the state of ADC and input clock select of ADC module.

ADC Calibration Control Functions.

- void ADC_SetCalibration (ADC_Type *base, bool enable)
  This function is used to enable or disable calibration function.
- static uint8_t ADC_GetCalibrationResult (ADC_Type *base)
  This function is used to get calibrate result value.

ADC Module Conversion Control Functions.

- void ADC_SetConvertCmd (ADC_Type *base, uint8_t channel, bool enable)
  Enable continuous conversion and start a conversion on target channel.
- void ADC_TriggerSingleConvert (ADC_Type *base, uint8_t channel)
  Enable single conversion and trigger single time conversion on target imput channel.If configured as hardware trigger mode, this function just set input channel and not start a conversion.
- void ADC_SetAverageNum (ADC_Type *base, uint8_t avgNum)
  Enable hardware aaverage function and set hardware average number.
- static void ADC_SetResolutionMode (ADC_Type *base, uint8_t mode)
  Set conversion resolution mode.
- static uint8_t ADC_GetResolutionMode (ADC_Type *base)
  Set conversion resolution mode.

- void ADC_StopConvert (ADC_Type *base)
  Set conversion disabled.
- uint16_t ADC_GetConvertResult (ADC_Type *base)
  Get right aligned conversion result.

ADC Comparer Control Functions.

- void ADC_SetCmpMode (ADC_Type *base, uint8_t cmpMode, uint16_t cmpVal1, uint16_t cmpVal2)
  Enable compare function and set the compare work mode of ADC module.

Offset Correction Control Functions.

- void ADC_SetCorrectionMode (ADC_Type *base, bool correctMode)
  Set ADC module offset correct mode.
- static void ADC_SetOffsetVal (ADC_Type *base, uint16_t val)
  Set ADC module offset value.

Interrupt and Flag Control Functions.

- void ADC_SetIntCmd (ADC_Type *base, bool enable)
  Enables or disables ADC conversion complete interrupt request.
- bool ADC_IsConvertComplete (ADC_Type *base)
  Gets the ADC module conversion complete status flag state.
- static uint32_t ADC_GetStatusFlag (ADC_Type *base, uint32_t flags)
  Gets the ADC module general status flag state.
- static void ADC_ClearStatusFlag (ADC_Type *base, uint32_t flags)
  Clear one or more ADC status flag state.

DMA Control Functions.

- void ADC_SetDmaCmd (ADC_Type *base, bool enable)
  Enable or Disable DMA request.

3.2.5 Data Structure Documentation

3.2.5.1 struct adc_init_config_t

Data Fields

- uint8_t clockSource
  Select input clock source to generate the internal conversion clock.
- uint8_t divideRatio

Selects divide ratio used to generate the internal conversion clock.

- uint8_t averageNumber
  The average number for hardware average function.
- uint8_t resolutionMode
  Set ADC resolution mode.

3.2.5.1.0.1 Field Documentation

3.2.5.1.0.1.1 uint8_t adc_init_config_t::clockSource

3.2.5.1.0.1.2 uint8_t adc_init_config_t::divideRatio

3.2.5.1.0.1.3 uint8_t adc_init_config_t::averageNumber

3.2.5.1.0.1.4 uint8_t adc_init_config_t::resolutionMode

3.2.6 Enumeration Type Documentation

3.2.6.1 enum _adc_average_number

Enumerator

adcAvgNum4 ADC Hardware Average Number is set to 4.
adcAvgNum8 ADC Hardware Average Number is set to 8.
adcAvgNum16 ADC Hardware Average Number is set to 16.
adcAvgNum32 ADC Hardware Average Number is set to 32.
adcAvgNumNone Disable ADC Hardware Average.

3.2.6.2 enum _adc_convert_trigger_mode

Enumerator

adcSoftwareTrigger ADC software trigger a conversion.
adcHardwareTrigger ADC hardware trigger a conversion.

3.2.6.3 enum _adc_convert_speed_config

Enumerator

adcNormalSpeed ADC set as normal conversion speed.
adcHighSpeed ADC set as high conversion speed.

3.2.6.4 enum _adc_sample_time_duration

Enumerator

adcSamplePeriodClock2 The sample time duration is set as 2 ADC clocks.

adcSamplePeriodClock4 The sample time duration is set as 4 ADC clocks.
adcSamplePeriodClock6 The sample time duration is set as 6 ADC clocks.
adcSamplePeriodClock8 The sample time duration is set as 8 ADC clocks.
adcSamplePeriodClock12 The sample time duration is set as 12 ADC clocks.
adcSamplePeriodClock16 The sample time duration is set as 16 ADC clocks.
adcSamplePeriodClock20 The sample time duration is set as 20 ADC clocks.
adcSamplePeriodClock24 The sample time duration is set as 24 ADC clocks.

3.2.6.5 enum _adc_power_mode

Enumerator

adcNormalPowerMode ADC hard block set as normal power mode.
adcLowPowerMode ADC hard block set as low power mode.

3.2.6.6 enum _adc_resolution_mode

Enumerator

adcResolutionBit8 ADC resolution set as 8 bit conversion mode.
adcResolutionBit10 ADC resolution set as 10 bit conversion mode.
adcResolutionBit12 ADC resolution set as 12 bit conversion mode.

3.2.6.7 enum _adc_clock_divide

Enumerator

adcInputClockDiv1 Input clock divide 1 to generate internal clock.
adcInputClockDiv2 Input clock divide 2 to generate internal clock.
adcInputClockDiv4 Input clock divide 4 to generate internal clock.
adcInputClockDiv8 Input clock divide 8 to generate internal clock.

3.2.6.8 enum _adc_clock_source

Enumerator

adcIpgClock Select ipg clock as input clock source.
adcIpgClockDivide2 Select ipg clock divide 2 as input clock source.
adcAsynClock Select asynchronous clock as input clock source.

3.2.6.9 enum _adc_compare_mode

Enumerator

adcCmpModeLessThanCmpVal1 Compare true if the result is less than compare value 1.
adcCmpModeGreaterThanCmpVal1 Compare true if the result is greater than or equal to compare value 1.
adcCmpModeOutRangNotInclusive Compare true if the result is less than compare value 1 or the result is Greater than compare value 2.
adcCmpModeInRangNotInclusive Compare true if the result is less than compare value 1 and the result is greater than compare value 2.
adcCmpModeInRangInclusive Compare true if the result is greater than or equal to compare value 1 and the result is less than or equal to compare value 2.
adcCmpModeOutRangInclusive Compare true if the result is greater than or equal to compare value 1 or the result is less than or equal to compare value 2.
adcCmpModeDisable ADC compare function disable.

3.2.6.10 enum _adc_general_status_flag

Enumerator

adcFlagAsynWakeUpInt Indicate asynnchronous wake up interrupt occured in stop mode.
adcFlagCalibrateFailed Indicate the result of the calibration sequence.
adcFlagConvertActive Indicate a conversion is in the process.

3.2.7 Function Documentation

3.2.7.1 void ADC_Init (ADC_Type * *base,* const adc_init_config_t * *initConfig*)

Parameters

base	ADC base pointer.
initConfig	ADC initialize structure.

3.2.7.2 void ADC_Deinit (ADC_Type * *base*)

Parameters

base	ADC base pointer.

3.2.7.3 void ADC_SetConvertResultOverwrite (ADC_Type * *base*, bool *enable*)

Parameters

base	ADC base pointer.
enable	Enable/Disable conversion result overwire function. • true: Enable conversion result overwire. • false: Disable conversion result overwrite.

3.2.7.4 void ADC_SetConvertTrigMode (ADC_Type * *base*, uint8_t *mode*)

Parameters

base	ADC base pointer.
mode	Conversion trigger (see _adc_convert_trigger_mode enumeration).

3.2.7.5 static uint8_t ADC_GetConvertTrigMode (ADC_Type * *base*) [inline], [static]

Parameters

base	ADC base pointer.

Returns

Conversion trigger mode (see _adc_convert_trigger_mode enumeration).

3.2.7.6 void ADC_SetConvertSpeed (ADC_Type * *base*, uint8_t *mode*)

Parameters

base	ADC base pointer.
mode	Conversion speed mode (see _adc_convert_speed_config enumeration).

3.2.7.7 static uint8_t ADC_GetConvertSpeed (ADC_Type * *base*) [inline], [static]

Parameters

base	ADC base pointer.

Returns

Conversion speed mode.

3.2.7.8 void ADC_SetSampleTimeDuration (ADC_Type * *base,* uint8_t *duration*)

Parameters

base	ADC base pointer.
duration	Sample time duration (see _adc_sample_time_duration enumeration).

3.2.7.9 void ADC_SetPowerMode (ADC_Type * *base,* uint8_t *powerMode*)

Parameters

base	ADC base pointer.
powerMode	power mode (see _adc_power_mode enumeration).

3.2.7.10 static uint8_t ADC_GetPowerMode (ADC_Type * *base*) [inline], [static]

Parameters

base	ADC base pointer.

Returns

Power mode.

3.2.7.11 void ADC_SetClockSource (ADC_Type * *base*, uint8_t *source*, uint8_t *div*)

Parameters

base	ADC base pointer.
source	Conversion clock source (see _adc_clock_source enumeration).
div	Input clock divide ratio (see _adc_clock_divide enumeration).

3.2.7.12 void ADC_SetAsynClockOutput (ADC_Type * *base*, bool *enable*)

Setting this bit allows the clock to be used even while the ADC is idle or operating from a different clock source.

Parameters

base	ADC base pointer.
enable	Asynchronous clock output enable. • true: Enable asynchronous clock output regardless of the state of ADC; • false: Only enable if selected as ADC input clock source and a ADC conversion is active.

3.2.7.13 void ADC_SetCalibration (ADC_Type * *base*, bool *enable*)

Parameters

base	ADC base pointer.
enable	Enable/Disable calibration function. • true: Enable calibration function. • false: Disable calibration function.

3.2.7.14 static uint8_t ADC_GetCalibrationResult (ADC_Type * *base*) [inline], [static]

Parameters

base	ADC base pointer.

Returns

Calibration result value.

3.2.7.15 void ADC_SetConvertCmd (ADC_Type * *base,* uint8_t *channel,* bool *enable*)

```
This function is only used for software trigger mode. If configured as
hardware trigger mode, this function just enable continuous conversion mode
and not start the conversion.
```

Parameters

base	ADC base pointer.
channel	Input channel selection.
enable	Enable/Disable continuous conversion. • true: Enable and start continuous conversion. • false: Disable continuous conversion.

3.2.7.16 void ADC_TriggerSingleConvert (ADC_Type * *base,* uint8_t *channel*)

Parameters

base	ADC base pointer.
channel	Input channel selection.

3.2.7.17 void ADC_SetAverageNum (ADC_Type * *base,* uint8_t *avgNum*)

Parameters

base	ADC base pointer.
avgNum	Hardware average number (see _adc_average_number enumeration). If avgNum is equal to adcAvgNumNone, it means disable hardware average function.

3.2.7.18 static void ADC_SetResolutionMode (ADC_Type * *base*, uint8_t *mode*) [inline], [static]

Parameters

base	ADC base pointer.
mode	resolution mode (see _adc_resolution_mode enumeration).

3.2.7.19 static uint8_t ADC_GetResolutionMode (ADC_Type * *base*) [inline], [static]

Parameters

base	ADC base pointer.

Returns

Resolution mode (see _adc_resolution_mode enumeration).

3.2.7.20 void ADC_StopConvert (ADC_Type * *base*)

Parameters

base	ADC base pointer.

3.2.7.21 uint16_t ADC_GetConvertResult (ADC_Type * *base*)

Parameters

base	ADC base pointer.

Returns

Conversion result.

3.2.7.22 void ADC_SetCmpMode (ADC_Type * *base*, uint8_t *cmpMode*, uint16_t *cmpVal1*, uint16_t *cmpVal2*)

```
If cmpMode is equal to adcCmpModeDisable, it means to disable the compare function.
```

Parameters

base	ADC base pointer.
cmpMode	Comparer work mode selected (see _adc_compare_mode enumeration). • adcCmpModeLessThanCmpVal1: only set compare value 1; • adcCmpModeGreaterThanCmpVal1: only set compare value 1; • adcCmpModeOutRangNotInclusive: set compare value 1 less than or eqeal to compare value 2; • adcCmpModeInRangNotInclusive: set compare value 1 greater than compare value 2; • adcCmpModeInRangInclusive: set compare value 1 less than or eqeal to compare value 2; • adcCmpModeOutRangInclusive: set compare value 1 greater than compare value 2; • adcCmpModeDisable: unnecessary to set compare value 1 and compare value 2.

cmpVal1	Compare threshold 1.
cmpVal2	Compare threshold 2.

3.2.7.23 void ADC_SetCorrectionMode (ADC_Type * *base,* bool *correctMode*)

Parameters

base	ADC base pointer.
correctMode	Offset correct mode. • true: The offset value is subtracted from the raw converted value; • false: The offset value is added with the raw result.

3.2.7.24 static void ADC_SetOffsetVal (ADC_Type * *base,* uint16_t *val*) [inline], [static]

Parameters

base	ADC base pointer.
val	Offset value.

3.2.7.25 void ADC_SetIntCmd (ADC_Type * *base,* bool *enable*)

Parameters

base	ADC base pointer.
enable	Enable/Disable ADC conversion complete interrupt. • true: Enable conversion complete interrupt. • false: Disable conversion complete interrupt.

3.2.7.26 bool ADC_IsConvertComplete (ADC_Type * *base*)

Parameters

base	ADC base pointer.

Return values

true,:	A conversion is completed.
false,:	A conversion is not completed.

3.2.7.27 static uint32_t ADC_GetStatusFlag (ADC_Type * *base,* uint32_t *flags*) [inline], [static]

Parameters

base	ADC base pointer.
flags	ADC status flag mask (see _adc_general_status_flag enumeration).

Returns

ADC status, each bit represents one status flag.

3.2.7.28 static void ADC_ClearStatusFlag (ADC_Type * *base,* uint32_t *flags*) [inline], [static]

Parameters

base	ADC base pointer.
flags	ADC status flag mask (see _adc_general_status_flag enumeration).

3.2.7.29 void ADC_SetDmaCmd (ADC_Type * *base,* bool *enable*)

Parameters

base	ADC base pointer.

enable	Enable/Disable ADC DMA request. • true: Enable DMA request. • false: Disable DMA request.

Chapter 4
Clock Control Module (CCM)

4.1 Overview

The FreeRTOS BSP provides a driver for the Clock Control Module (CCM) block of i.MX devices.

Modules

- CCM Analog driver
- CCM driver

4.2 CCM Analog driver

4.2.1 Overview

The chapter describes the programming interface of the CCM Analog driver (platform/drivers/inc/ccm_analog_imx6sx.h). The Clock Control Module (CCM) Analog part provides the PLL and PFD control. The CCM Analog driver provides a set of APIs to access the control registers, including these services:

- PLL power, gate, lock status, and output frequency
- PFD gate, stable, fraction, and output frequency

4.2.2 PLL power, gate, lock status and output frequency

PLL uses OSC as an external reference clock. To use CCM PLL, ensure that the reference clock is set correctly.

PLL can be powered up/down by the POWERDOWN bit in the PLL register. If no peripheral is running with the clock derived from this PLL, the PLL can be powered down to save power. Use the CCM_ANALOG_PowerUpPll() and CCM_ANALOG_PowerDownPll() functions for this purpose.

PLL can be bypassed and peripherals can use PLL as a clock source to get the OSC frequency in bypassed mode. This is a legacy method for i.MX series. However it's not recommended for the i.MX 6SoloX because most slow peripherals can directly select the OSC as a clock source and it doesn't make sense to force the PLL bypass. Use the CCM_ANALOG_SetPllBypass() and CCM_ANALOG_IsPllBypassed() functions to set and get the status of the PLL bypass mode.

After power up, the PLL clock is still not available for use. The PLL clock functions CCM_ANALOG_EnablePllClock() and CCM_ANALOG_DisablePllClock() can be used to control the clock output.

After enabling the PLL, check whether the PLL is locked before it is used by peripherals by using the CCM_ANALOG_IsPllLocked() function.

To help getting the current system PLL frequency easier, CCM_ANALOG_GetSysPllFreq() is provided to get the clock frequency in HZ.

4.2.3 PFD gate, stable, fraction and output frequency

The system PLL is equipped with the Phase Fractional Dividers(PFD) to generate the additional frequencies required by the different functional blocks.

CCM_ANALOG_EnablePfdClock() and CCM_ANALOG_DisablePfdClock() are used to allow clock outputting to functional blocks or not. In addition to the PFD clocks, some system PLL's clock output is also controlled by these functions.

After enabling the PFD clock, we need to make sure the PFD clock is stable before getting it used by peripherals. Call CCM_ANALOG_IsPfdStable() to get the status.

To get different frequencies, fractions are needed. Call CCM_ANALOG_SetPfdFrac() to set fraction to

get your required frequency. Call CCM_ANALOG_GetPfdFrac() to get current setting of fraction.

To help getting current PFD frequency easier, CCM_ANALOG_GetPfdFreq() is provided to get PFD clock frequency in HZ.

Enumerations

- enum _ccm_analog_pll_control {
  ccmAnalogPllArmControl = CCM_ANALOG_TUPLE(PLL_ARM, CCM_ANALOG_PLL_ARM_POWERDOWN_SHIFT),
  ccmAnalogPllUsb1Control = CCM_ANALOG_TUPLE(PLL_USB1, CCM_ANALOG_PLL_USB1_POWER_SHIFT),
  ccmAnalogPllUsb2Control = CCM_ANALOG_TUPLE(PLL_USB2, CCM_ANALOG_PLL_USB2_POWER_SHIFT),
  ccmAnalogPllSysControl = CCM_ANALOG_TUPLE(PLL_SYS, CCM_ANALOG_PLL_SYS_POWERDOWN_SHIFT),
  ccmAnalogPllAudioControl = CCM_ANALOG_TUPLE(PLL_AUDIO, CCM_ANALOG_PLL_AUDIO_POWERDOWN_SHIFT),
  ccmAnalogPllVideoControl = CCM_ANALOG_TUPLE(PLL_VIDEO, CCM_ANALOG_PLL_VIDEO_POWERDOWN_SHIFT),
  ccmAnalogPllEnetControl = CCM_ANALOG_TUPLE(PLL_ENET, CCM_ANALOG_PLL_ENET_POWERDOWN_SHIFT) }
  PLL control names for PLL power/bypass/lock/frequency operations.
- enum _ccm_analog_pll_clock {
  ccmAnalogPllArmClock = CCM_ANALOG_TUPLE(PLL_ARM, CCM_ANALOG_PLL_ARM_ENABLE_SHIFT),
  ccmAnalogPllUsb1Clock = CCM_ANALOG_TUPLE(PLL_USB1, CCM_ANALOG_PLL_USB1_ENABLE_SHIFT),
  ccmAnalogPllUsb2Clock = CCM_ANALOG_TUPLE(PLL_USB2, CCM_ANALOG_PLL_USB2_ENABLE_SHIFT),
  ccmAnalogPllSysClock = CCM_ANALOG_TUPLE(PLL_SYS, CCM_ANALOG_PLL_SYS_ENABLE_SHIFT),
  ccmAnalogPllAudioClock = CCM_ANALOG_TUPLE(PLL_AUDIO, CCM_ANALOG_PLL_AUDIO_ENABLE_SHIFT),
  ccmAnalogPllVideoClock = CCM_ANALOG_TUPLE(PLL_VIDEO, CCM_ANALOG_PLL_VIDEO_ENABLE_SHIFT),
  ccmAnalogPllEnetClock25Mhz = CCM_ANALOG_TUPLE(PLL_ENET, CCM_ANALOG_PLL_ENET_ENET_25M_REF_EN_SHIFT),
  ccmAnalogPllEnet2Clock125Mhz = CCM_ANALOG_TUPLE(PLL_ENET, CCM_ANALOG_PLL_ENET_ENET2_125M_EN_SHIFT),
  ccmAnalogPllEnet1Clock125Mhz = CCM_ANALOG_TUPLE(PLL_ENET, CCM_ANALOG_PLL_ENET_ENET1_125M_EN_SHIFT) }
  PLL clock names for clock enable/disable settings.
- enum _ccm_analog_pfd_clkgate {
  ccmAnalogPll2Pfd0ClkGate = CCM_ANALOG_TUPLE(PFD_528, CCM_ANALOG_PFD_528-

_PFD0_CLKGATE_SHIFT),
ccmAnalogPll2Pfd1ClkGate = CCM_ANALOG_TUPLE(PFD_528, CCM_ANALOG_PFD_528-_PFD1_CLKGATE_SHIFT),
ccmAnalogPll2Pfd2ClkGate = CCM_ANALOG_TUPLE(PFD_528, CCM_ANALOG_PFD_528-_PFD2_CLKGATE_SHIFT),
ccmAnalogPll2Pfd3ClkGate = CCM_ANALOG_TUPLE(PFD_528, CCM_ANALOG_PFD_528-_PFD3_CLKGATE_SHIFT),
ccmAnalogPll3Pfd0ClkGate = CCM_ANALOG_TUPLE(PFD_480, CCM_ANALOG_PFD_480-_PFD0_CLKGATE_SHIFT),
ccmAnalogPll3Pfd1ClkGate = CCM_ANALOG_TUPLE(PFD_480, CCM_ANALOG_PFD_480-_PFD1_CLKGATE_SHIFT),
ccmAnalogPll3Pfd2ClkGate = CCM_ANALOG_TUPLE(PFD_480, CCM_ANALOG_PFD_480-_PFD2_CLKGATE_SHIFT),
ccmAnalogPll3Pfd3ClkGate = CCM_ANALOG_TUPLE(PFD_480, CCM_ANALOG_PFD_480-_PFD3_CLKGATE_SHIFT) }

PFD gate names for clock gate settings, clock source is PLL2 and PLL3.

- enum _ccm_analog_pfd_frac {
  ccmAnalogPll2Pfd0Frac = CCM_ANALOG_TUPLE(PFD_528, CCM_ANALOG_PFD_528_PFD0_FRAC_SHIFT),
  ccmAnalogPll2Pfd1Frac = CCM_ANALOG_TUPLE(PFD_528, CCM_ANALOG_PFD_528_PFD1_FRAC_SHIFT),
  ccmAnalogPll2Pfd2Frac = CCM_ANALOG_TUPLE(PFD_528, CCM_ANALOG_PFD_528_PFD2_FRAC_SHIFT),
  ccmAnalogPll2Pfd3Frac = CCM_ANALOG_TUPLE(PFD_528, CCM_ANALOG_PFD_528_PFD3_FRAC_SHIFT),
  ccmAnalogPll3Pfd0Frac = CCM_ANALOG_TUPLE(PFD_480, CCM_ANALOG_PFD_480_PFD0_FRAC_SHIFT),
  ccmAnalogPll3Pfd1Frac = CCM_ANALOG_TUPLE(PFD_480, CCM_ANALOG_PFD_480_PFD1_FRAC_SHIFT),
  ccmAnalogPll3Pfd2Frac = CCM_ANALOG_TUPLE(PFD_480, CCM_ANALOG_PFD_480_PFD2_FRAC_SHIFT),
  ccmAnalogPll3Pfd3Frac = CCM_ANALOG_TUPLE(PFD_480, CCM_ANALOG_PFD_480_PFD3_FRAC_SHIFT) }

  PFD fraction names for clock fractional divider operations.
- enum _ccm_analog_pfd_stable {
  ccmAnalogPll2Pfd0Stable = CCM_ANALOG_TUPLE(PFD_528, CCM_ANALOG_PFD_528_PFD0_STABLE_SHIFT),
  ccmAnalogPll2Pfd1Stable = CCM_ANALOG_TUPLE(PFD_528, CCM_ANALOG_PFD_528_PFD1_STABLE_SHIFT),
  ccmAnalogPll2Pfd2Stable = CCM_ANALOG_TUPLE(PFD_528, CCM_ANALOG_PFD_528_PFD2_STABLE_SHIFT),
  ccmAnalogPll2Pfd3Stable = CCM_ANALOG_TUPLE(PFD_528, CCM_ANALOG_PFD_528_PFD3_STABLE_SHIFT),
  ccmAnalogPll3Pfd0Stable = CCM_ANALOG_TUPLE(PFD_480, CCM_ANALOG_PFD_480_P-

FD0_STABLE_SHIFT),
ccmAnalogPll3Pfd1Stable = CCM_ANALOG_TUPLE(PFD_480, CCM_ANALOG_PFD_480_PFD1_STABLE_SHIFT),
ccmAnalogPll3Pfd2Stable = CCM_ANALOG_TUPLE(PFD_480, CCM_ANALOG_PFD_480_PFD2_STABLE_SHIFT),
ccmAnalogPll3Pfd3Stable = CCM_ANALOG_TUPLE(PFD_480, CCM_ANALOG_PFD_480_PFD3_STABLE_SHIFT) }
PFD stable names for clock stable query.

CCM Analog PLL Operatoin Functions

- void CCM_ANALOG_PowerUpPll (CCM_ANALOG_Type *base, uint32_t pllControl)
  Power up PLL.
- void CCM_ANALOG_PowerDownPll (CCM_ANALOG_Type *base, uint32_t pllControl)
  Power down PLL.
- void CCM_ANALOG_SetPllBypass (CCM_ANALOG_Type *base, uint32_t pllControl, bool bypass)
  PLL bypass setting.
- static bool CCM_ANALOG_IsPllBypassed (CCM_ANALOG_Type *base, uint32_t pllControl)
  Check if PLL is bypassed.
- static bool CCM_ANALOG_IsPllLocked (CCM_ANALOG_Type *base, uint32_t pllControl)
  Check if PLL clock is locked.
- static void CCM_ANALOG_EnablePllClock (CCM_ANALOG_Type *base, uint32_t pllClock)
  Enable PLL clock.
- static void CCM_ANALOG_DisablePllClock (CCM_ANALOG_Type *base, uint32_t pllClock)
  Disable PLL clock.
- uint32_t CCM_ANALOG_GetPllFreq (CCM_ANALOG_Type *base, uint32_t pllControl)
  Get PLL(involved all PLLs) clock frequency.

CCM Analog PFD Operatoin Functions

- static void CCM_ANALOG_EnablePfdClock (CCM_ANALOG_Type *base, uint32_t pfdClkGate)
  Enable PFD clock.
- static void CCM_ANALOG_DisablePfdClock (CCM_ANALOG_Type *base, uint32_t pfdClkGate)
  Disable PFD clock.
- static bool CCM_ANALOG_IsPfdStable (CCM_ANALOG_Type *base, uint32_t pfdStable)
  Check if PFD clock is stable.
- static void CCM_ANALOG_SetPfdFrac (CCM_ANALOG_Type *base, uint32_t pfdFrac, uint32_t value)
  Set PFD clock fraction.
- static uint32_t CCM_ANALOG_GetPfdFrac (CCM_ANALOG_Type *base, uint32_t pfdFrac)
  Get PFD clock fraction.
- uint32_t CCM_ANALOG_GetPfdFreq (CCM_ANALOG_Type *base, uint32_t pfdFrac)
  Get PFD clock frequency.

4.2.4 Enumeration Type Documentation

4.2.4.1 enum _ccm_analog_pll_control

These constants define the PLL control names for PLL power/bypass/lock operations.

- 0:15: REG offset to CCM_ANALOG_BASE in bytes.
- 16:20: Powerdown/Power bit shift.

Enumerator

ccmAnalogPllArmControl CCM Analog ARM PLL Control.
ccmAnalogPllUsb1Control CCM Analog USB1 PLL Control.
ccmAnalogPllUsb2Control CCM Analog USB2 PLL Control.
ccmAnalogPllSysControl CCM Analog SYSTEM PLL Control.
ccmAnalogPllAudioControl CCM Analog AUDIO PLL Control.
ccmAnalogPllVideoControl CCM Analog VIDEO PLL Control.
ccmAnalogPllEnetControl CCM Analog ETHERNET PLL Control.

4.2.4.2 enum _ccm_analog_pll_clock

These constants define the PLL clock names for PLL clock enable/disable operations.

- 0:15: REG offset to CCM_ANALOG_BASE in bytes.
- 16:20: Clock enable bit shift.

Enumerator

ccmAnalogPllArmClock CCM Analog ARM PLL Clock.
ccmAnalogPllUsb1Clock CCM Analog USB1 PLL Clock.
ccmAnalogPllUsb2Clock CCM Analog USB2 PLL Clock.
ccmAnalogPllSysClock CCM Analog SYSTEM PLL Clock.
ccmAnalogPllAudioClock CCM Analog AUDIO PLL Clock.
ccmAnalogPllVideoClock CCM Analog VIDEO PLL Clock.
ccmAnalogPllEnetClock25Mhz CCM Analog ETHERNET 25MHz PLL Clock.
ccmAnalogPllEnet2Clock125Mhz CCM Analog ETHERNET2 125MHz PLL Clock.
ccmAnalogPllEnet1Clock125Mhz CCM Analog ETHERNET1 125MHz PLL Clock.

4.2.4.3 enum _ccm_analog_pfd_clkgate

These constants define the PFD gate names for PFD clock enable/disable operations.

- 0:15: REG offset to CCM_ANALOG_BASE in bytes.
- 16:20: Clock gate bit shift.

Enumerator

ccmAnalogPll2Pfd0ClkGate CCM Analog PLL2 PFD0 Clock Gate.
ccmAnalogPll2Pfd1ClkGate CCM Analog PLL2 PFD1 Clock Gate.
ccmAnalogPll2Pfd2ClkGate CCM Analog PLL2 PFD2 Clock Gate.
ccmAnalogPll2Pfd3ClkGate CCM Analog PLL2 PFD3 Clock Gate.
ccmAnalogPll3Pfd0ClkGate CCM Analog PLL3 PFD0 Clock Gate.
ccmAnalogPll3Pfd1ClkGate CCM Analog PLL3 PFD1 Clock Gate.
ccmAnalogPll3Pfd2ClkGate CCM Analog PLL3 PFD2 Clock Gate.
ccmAnalogPll3Pfd3ClkGate CCM Analog PLL3 PFD3 Clock Gate.

4.2.4.4 enum _ccm_analog_pfd_frac

These constants define the PFD fraction names for PFD fractional divider operations.

- 0:15: REG offset to CCM_ANALOG_BASE in bytes.
- 16:20: Fraction bits shift

Enumerator

ccmAnalogPll2Pfd0Frac CCM Analog PLL2 PFD0 fractional divider.
ccmAnalogPll2Pfd1Frac CCM Analog PLL2 PFD1 fractional divider.
ccmAnalogPll2Pfd2Frac CCM Analog PLL2 PFD2 fractional divider.
ccmAnalogPll2Pfd3Frac CCM Analog PLL2 PFD3 fractional divider.
ccmAnalogPll3Pfd0Frac CCM Analog PLL3 PFD0 fractional divider.
ccmAnalogPll3Pfd1Frac CCM Analog PLL3 PFD1 fractional divider.
ccmAnalogPll3Pfd2Frac CCM Analog PLL3 PFD2 fractional divider.
ccmAnalogPll3Pfd3Frac CCM Analog PLL3 PFD3 fractional divider.

4.2.4.5 enum _ccm_analog_pfd_stable

These constants define the PFD stable names for clock stable query.

- 0:15: REG offset to CCM_ANALOG_BASE in bytes.
- 16:20: Stable bit shift.

Enumerator

ccmAnalogPll2Pfd0Stable CCM Analog PLL2 PFD0 clock stable query.
ccmAnalogPll2Pfd1Stable CCM Analog PLL2 PFD1 clock stable query.
ccmAnalogPll2Pfd2Stable CCM Analog PLL2 PFD2 clock stable query.
ccmAnalogPll2Pfd3Stable CCM Analog PLL2 PFD3 clock stable query.
ccmAnalogPll3Pfd0Stable CCM Analog PLL3 PFD0 clock stable query.
ccmAnalogPll3Pfd1Stable CCM Analog PLL3 PFD1 clock stable query.
ccmAnalogPll3Pfd2Stable CCM Analog PLL3 PFD2 clock stable query.
ccmAnalogPll3Pfd3Stable CCM Analog PLL3 PFD3 clock stable query.

4.2.5 Function Documentation

4.2.5.1 void CCM_ANALOG_PowerUpPll (CCM_ANALOG_Type * *base,* uint32_t *pllControl*)

Parameters

base	CCM_ANALOG base pointer.
pllControl	PLL control name (see _ccm_analog_pll_control enumeration)

4.2.5.2 void CCM_ANALOG_PowerDownPll (CCM_ANALOG_Type * *base*, uint32_t *pllControl*)

Parameters

base	CCM_ANALOG base pointer.
pllControl	PLL control name (see _ccm_analog_pll_control enumeration)

4.2.5.3 void CCM_ANALOG_SetPllBypass (CCM_ANALOG_Type * *base*, uint32_t *pllControl*, bool *bypass*)

Parameters

base	CCM_ANALOG base pointer.
pllControl	PLL control name (see _ccm_analog_pll_control enumeration)
bypass	Bypass the PLL. • true: Bypass the PLL. • false: Do not bypass the PLL.

4.2.5.4 static bool CCM_ANALOG_IsPllBypassed (CCM_ANALOG_Type * *base*, uint32_t *pllControl*) [inline], [static]

Parameters

base	CCM_ANALOG base pointer.
pllControl	PLL control name (see _ccm_analog_pll_control enumeration)

Returns

PLL bypass status.
- true: The PLL is bypassed.
- false: The PLL is not bypassed.

4.2.5.5 static bool CCM_ANALOG_IsPllLocked (CCM_ANALOG_Type * *base,* uint32_t *pllControl*) [inline], [static]

Parameters

base	CCM_ANALOG base pointer.
pllControl	PLL control name (see _ccm_analog_pll_control enumeration)

Returns

PLL lock status.
- true: The PLL is locked.
- false: The PLL is not locked.

4.2.5.6 static void CCM_ANALOG_EnablePllClock (CCM_ANALOG_Type * *base*, uint32_t *pllClock*) [inline], [static]

Parameters

base	CCM_ANALOG base pointer.
pllClock	PLL clock name (see _ccm_analog_pll_clock enumeration)

4.2.5.7 static void CCM_ANALOG_DisablePllClock (CCM_ANALOG_Type * *base*, uint32_t *pllClock*) [inline], [static]

Parameters

base	CCM_ANALOG base pointer.
pllClock	PLL clock name (see _ccm_analog_pll_clock enumeration)

4.2.5.8 uint32_t CCM_ANALOG_GetPllFreq (CCM_ANALOG_Type * *base*, uint32_t *pllControl*)

Parameters

base	CCM_ANALOG base pointer.

pllControl	PLL control name (see _ccm_analog_pll_control enumeration)

Returns

PLL clock frequency in HZ.

4.2.5.9 static void CCM_ANALOG_EnablePfdClock (CCM_ANALOG_Type * *base*, uint32_t *pfdClkGate*) [inline], [static]

Parameters

base	CCM_ANALOG base pointer.
pfdClkGate	PFD clock gate (see _ccm_analog_pfd_clkgate enumeration)

4.2.5.10 static void CCM_ANALOG_DisablePfdClock (CCM_ANALOG_Type * *base*, uint32_t *pfdClkGate*) [inline], [static]

Parameters

base	CCM_ANALOG base pointer.
pfdClkGate	PFD clock gate (see _ccm_analog_pfd_clkgate enumeration)

4.2.5.11 static bool CCM_ANALOG_IsPfdStable (CCM_ANALOG_Type * *base*, uint32_t *pfdStable*) [inline], [static]

Parameters

base	CCM_ANALOG base pointer.
pfdStable	PFD stable identifier (see _ccm_analog_pfd_stable enumeration)

Returns

PFD clock stable status.
- true: The PFD clock is stable.
- false: The PFD clock is not stable.

4.2.5.12 static void CCM_ANALOG_SetPfdFrac (CCM_ANALOG_Type * *base*, uint32_t *pfdFrac*, uint32_t *value*) [inline], [static]

Parameters

base	CCM_ANALOG base pointer.
pfdFrac	PFD clock fraction (see _ccm_analog_pfd_frac enumeration)
value	PFD clock fraction value

4.2.5.13 static uint32_t CCM_ANALOG_GetPfdFrac (CCM_ANALOG_Type * *base*, uint32_t *pfdFrac*) [inline], [static]

Parameters

base	CCM_ANALOG base pointer.
pfdFrac	PFD clock fraction (see _ccm_analog_pfd_frac enumeration)

Returns

PFD clock fraction value

4.2.5.14 uint32_t CCM_ANALOG_GetPfdFreq (CCM_ANALOG_Type * *base*, uint32_t *pfdFrac*)

Parameters

base	CCM_ANALOG base pointer.
pfdFrac	PFD clock fraction (see _ccm_analog_pfd_frac enumeration)

Returns

PFD clock frequency in HZ

4.3 CCM driver

4.3.1 Overview

The chapter describes the programming interface of the CCM driver (platform/drivers/inc/ccm_imx6sx.-h). The Clock Control Module (CCM) provides clock routing, divider, and gate control. The CCM driver provides a set of APIs to access these control registers, including these services:

- Clock routing
- Clock divider
- Clock gate

4.3.2 Clock routing

There's a complicated clock tree in i.MX 6SoloX. To well configure the clock source to CPU, bus, and peripheral we have to know about the topology of clock tree. And each node on this tree consists of a MUX, a divider, or a clock gate. The root of MUX varies from OSC, PLL, PFD to external sources. Each MUX has different sources to select. Use CCM_SetRootMux() and CCM_GetRootMux() functions to set and get clock source. Before that, the clock node (which CPU, bus or peripheral or intermediate node) should be decided, and enumeration *ccm_root_clock_control is used for this purpose. The clock source is also enumerated by _ccm_rootmux* and every clock node has its own definition.

4.3.3 Clock divider

The clock root often has a high frequency, and, in many use cases, it needs to be divided before routing to the functional block. That's why the divider is introduced. There's quite a lot of dividers in i.MX 6SoloX clock tree and the first thing is to decide which divider to set. The enumeration _ccm_root_div_control is used for this purpose. Use CCM_SetRootDivider() function to set the dividers and CCM_GetRoot-Divider() function to get the current divider setting.

4.3.4 Clock gate

After clock routing and divider are properly set, the final step to output clock to functional block is opening the gate. CCM_ControlGate() function controls the gate states.

Clock gate has 3 states:

1. Domain clocks not needed
2. Domain clocks needed when in RUN
3. Domain clocks needed all the time
   In use cases 2, the gate closes automatically when the CPU runs into WAIT power mode.
   Choose the gate from _ccm_ccgr_gate enumeration and set the state defined by enum _ccm_gate_-value.

Enumerations

- enum _ccm_root_clock_control {
  ccmRootPll1SwClkSel = CCM_TUPLE(CCSR, CCM_CCSR_pll1_sw_clk_sel_SHIFT, CCM_CCSR_pll1_sw_clk_sel_MASK),
  ccmRootStepSel = CCM_TUPLE(CCSR, CCM_CCSR_step_sel_SHIFT, CCM_CCSR_step_sel_MASK),
  ccmRootPeriph2ClkSel = CCM_TUPLE(CBCDR, CCM_CBCDR_periph2_clk_sel_SHIFT, CCM_CBCDR_periph2_clk_sel_MASK),
  ccmRootPrePeriph2ClkSel = CCM_TUPLE(CBCMR, CCM_CBCMR_pre_periph2_clk_sel_SHIFT, CCM_CBCMR_pre_periph2_clk_sel_MASK),
  ccmRootPeriph2Clk2Sel = CCM_TUPLE(CBCMR, CCM_CBCMR_periph2_clk2_sel_SHIFT, CCM_CBCMR_periph2_clk2_sel_MASK),
  ccmRootPll3SwClkSel = CCM_TUPLE(CCSR, CCM_CCSR_pll3_sw_clk_sel_SHIFT, CCM_CCSR_pll3_sw_clk_sel_MASK),
  ccmRootOcramClkSel = CCM_TUPLE(CBCDR, CCM_CBCDR_ocram_clk_sel_SHIFT, CCM_CBCDR_ocram_clk_sel_MASK),
  ccmRootOcramAltClkSel = CCM_TUPLE(CBCDR, CCM_CBCDR_ocram_alt_clk_sel_SHIFT, CCM_CBCDR_ocram_alt_clk_sel_MASK),
  ccmRootPeriphClkSel = CCM_TUPLE(CBCDR, CCM_CBCDR_periph_clk_sel_SHIFT, CCM_CBCDR_periph_clk_sel_MASK),
  ccmRootPeriphClk2Sel = CCM_TUPLE(CBCMR, CCM_CBCMR_periph_clk2_sel_SHIFT, CCM_CBCMR_periph_clk2_sel_MASK),
  ccmRootPrePeriphClkSel = CCM_TUPLE(CBCMR, CCM_CBCMR_pre_periph_clk_sel_SHIFT, CCM_CBCMR_pre_periph_clk_sel_MASK),
  ccmRootPcieAxiClkSel = CCM_TUPLE(CBCMR, CCM_CBCMR_pcie_axi_clk_sel_SHIFT, CCM_CBCMR_pcie_axi_clk_sel_MASK),
  ccmRootPerclkClkSel = CCM_TUPLE(CSCMR1, CCM_CSCMR1_perclk_clk_sel_SHIFT, CCM_CSCMR1_perclk_clk_sel_MASK),
  ccmRootUsdhc1ClkSel = CCM_TUPLE(CSCMR1, CCM_CSCMR1_usdhc1_clk_sel_SHIFT, CCM_CSCMR1_usdhc1_clk_sel_MASK),
  ccmRootUsdhc2ClkSel = CCM_TUPLE(CSCMR1, CCM_CSCMR1_usdhc2_clk_sel_SHIFT, CCM_CSCMR1_usdhc2_clk_sel_MASK),
  ccmRootUsdhc3ClkSel = CCM_TUPLE(CSCMR1, CCM_CSCMR1_usdhc3_clk_sel_SHIFT, CCM_CSCMR1_usdhc3_clk_sel_MASK),
  ccmRootUsdhc4ClkSel = CCM_TUPLE(CSCMR1, CCM_CSCMR1_usdhc4_clk_sel_SHIFT, CCM_CSCMR1_usdhc4_clk_sel_MASK),
  ccmRootAclkEimSlowSel = CCM_TUPLE(CSCMR1, CCM_CSCMR1_aclk_eim_slow_sel_SHIFT, CCM_CSCMR1_aclk_eim_slow_sel_MASK),
  ccmRootGpuAxiSel = CCM_TUPLE(CBCMR, CCM_CBCMR_gpu_axi_sel_SHIFT, CCM_CBCMR_gpu_axi_sel_MASK),
  ccmRootGpuCoreSel = CCM_TUPLE(CBCMR, CCM_CBCMR_gpu_core_sel_SHIFT, CCM_CBCMR_gpu_core_sel_MASK),
  ccmRootVidClkSel = CCM_TUPLE(CSCMR2, CCM_CSCMR2_vid_clk_sel_SHIFT, CCM_CSC-

MR2_vid_clk_sel_MASK),
ccmRootEsaiClkSel = CCM_TUPLE(CSCMR2, CCM_CSCMR2_esai_clk_sel_SHIFT, CCM_CSCMR2_esai_clk_sel_MASK),
ccmRootAudioClkSel = CCM_TUPLE(CDCDR, CCM_CDCDR_audio_clk_sel_SHIFT, CCM_CDCDR_audio_clk_sel_MASK),
ccmRootSpdif0ClkSel = CCM_TUPLE(CDCDR, CCM_CDCDR_spdif0_clk_sel_SHIFT, CCM_CDCDR_spdif0_clk_sel_MASK),
ccmRootSsi1ClkSel = CCM_TUPLE(CSCMR1, CCM_CSCMR1_ssi1_clk_sel_SHIFT, CCM_CSCMR1_ssi1_clk_sel_MASK),
ccmRootSsi2ClkSel = CCM_TUPLE(CSCMR1, CCM_CSCMR1_ssi2_clk_sel_SHIFT, CCM_CSCMR1_ssi2_clk_sel_MASK),
ccmRootSsi3ClkSel = CCM_TUPLE(CSCMR1, CCM_CSCMR1_ssi3_clk_sel_SHIFT, CCM_CSCMR1_ssi3_clk_sel_MASK),
ccmRootLcdif2ClkSel = CCM_TUPLE(CSCDR2, CCM_CSCDR2_lcdif2_clk_sel_SHIFT, CCM_CSCDR2_lcdif2_clk_sel_MASK),
ccmRootLcdif2PreClkSel = CCM_TUPLE(CSCDR2, CCM_CSCDR2_lcdif2_pre_clk_sel_SHIFT, CCM_CSCDR2_lcdif2_pre_clk_sel_MASK),
ccmRootLdbDi1ClkSel = CCM_TUPLE(CS2CDR, CCM_CS2CDR_ldb_di1_clk_sel_SHIFT, CCM_CS2CDR_ldb_di1_clk_sel_MASK),
ccmRootLdbDi0ClkSel = CCM_TUPLE(CS2CDR, CCM_CS2CDR_ldb_di0_clk_sel_SHIFT, CCM_CS2CDR_ldb_di0_clk_sel_MASK),
ccmRootLcdif1ClkSel = CCM_TUPLE(CSCDR2, CCM_CSCDR2_lcdif1_clk_sel_SHIFT, CCM_CSCDR2_lcdif1_clk_sel_MASK),
ccmRootLcdif1PreClkSel = CCM_TUPLE(CSCDR2, CCM_CSCDR2_lcdif1_pre_clk_sel_SHIFT, CCM_CSCDR2_lcdif1_pre_clk_sel_MASK),
ccmRootM4ClkSel = CCM_TUPLE(CHSCCDR, CCM_CHSCCDR_m4_clk_sel_SHIFT, CCM_CHSCCDR_m4_clk_sel_MASK),
ccmRootM4PreClkSel = CCM_TUPLE(CHSCCDR, CCM_CHSCCDR_m4_pre_clk_sel_SHIFT, CCM_CHSCCDR_m4_pre_clk_sel_MASK),
ccmRootEnetClkSel = CCM_TUPLE(CHSCCDR, CCM_CHSCCDR_enet_clk_sel_SHIFT, CCM_CHSCCDR_enet_clk_sel_MASK),
ccmRootEnetPreClkSel = CCM_TUPLE(CHSCCDR, CCM_CHSCCDR_enet_pre_clk_sel_SHIFT, CCM_CHSCCDR_enet_pre_clk_sel_MASK),
ccmRootQspi2ClkSel = CCM_TUPLE(CS2CDR, CCM_CS2CDR_qspi2_clk_sel_SHIFT, CCM_CS2CDR_qspi2_clk_sel_MASK),
ccmRootDisplayClkSel = CCM_TUPLE(CSCDR3, CCM_CSCDR3_display_clk_sel_SHIFT, CCM_CSCDR3_display_clk_sel_MASK),
ccmRootCsiClkSel = CCM_TUPLE(CSCDR3, CCM_CSCDR3_csi_clk_sel_SHIFT, CCM_CSCDR3_csi_clk_sel_MASK),
ccmRootCanClkSel = CCM_TUPLE(CSCMR2, CCM_CSCMR2_can_clk_sel_SHIFT, CCM_CSCMR2_can_clk_sel_MASK),
ccmRootEcspiClkSel = CCM_TUPLE(CSCDR2, CCM_CSCDR2_ecspi_clk_sel_SHIFT, CCM_CSCDR2_ecspi_clk_sel_MASK),
ccmRootUartClkSel = CCM_TUPLE(CSCDR1, CCM_CSCDR1_uart_clk_sel_SHIFT, CCM_CSCDR1_uart_clk_sel_MASK) }

Root control names for root clock setting.

- enum _ccm_rootmux_pll1_sw_clk_sel {
  ccmRootmuxPll1SwClkPll1MainClk = 0U,
  ccmRootmuxPll1SwClkStepClk = 1U }
  Root clock select enumeration for pll1_sw_clk_sel.
- enum _ccm_rootmux_step_sel {
  ccmRootmuxStepOsc24m = 0U,
  ccmRootmuxStepPll2Pfd2 = 1U }
  Root clock select enumeration for step_sel.
- enum _ccm_rootmux_periph2_clk_sel {
  ccmRootmuxPeriph2ClkPrePeriph2Clk = 0U,
  ccmRootmuxPeriph2ClkPeriph2Clk2 = 1U }
  Root clock select enumeration for periph2_clk_sel.
- enum _ccm_rootmux_pre_periph2_clk_sel {
  ccmRootmuxPrePeriph2ClkPll2 = 0U,
  ccmRootmuxPrePeriph2ClkPll2Pfd2 = 1U,
  ccmRootmuxPrePeriph2ClkPll2Pfd0 = 2U,
  ccmRootmuxPrePeriph2ClkPll4 = 3U }
  Root clock select enumeration for pre_periph2_clk_sel.
- enum _ccm_rootmux_periph2_clk2_sel {
  ccmRootmuxPeriph2Clk2Pll3SwClk = 0U,
  ccmRootmuxPeriph2Clk2Osc24m = 1U }
  Root clock select enumeration for periph2_clk2_sel.
- enum _ccm_rootmux_pll3_sw_clk_sel {
  ccmRootmuxPll3SwClkPll3 = 0U,
  ccmRootmuxPll3SwClkPll3BypassClk = 1U }
  Root clock select enumeration for pll3_sw_clk_sel.
- enum _ccm_rootmux_ocram_clk_sel {
  ccmRootmuxOcramClkPeriphClk = 0U,
  ccmRootmuxOcramClkOcramAltClk = 1U }
  Root clock select enumeration for ocram_clk_sel.
- enum _ccm_rootmux_ocram_alt_clk_sel {
  ccmRootmuxOcramAltClkPll2Pfd2 = 0U,
  ccmRootmuxOcramAltClkPll3Pfd1 = 1U }
  Root clock select enumeration for ocram_alt_clk_sel.
- enum _ccm_rootmux_periph_clk_sel {
  ccmRootmuxPeriphClkPrePeriphClkSel = 0U,
  ccmRootmuxPeriphClkPeriphClk2Sel = 1U }
  Root clock select enumeration for periph_clk_sel.
- enum _ccm_rootmux_periph_clk2_sel {
  ccmRootmuxPeriphClk2Pll3SwClk = 0U,
  ccmRootmuxPeriphClk2OSC24m = 1U,
  ccmRootmuxPeriphClk2Pll2 = 2U }
  Root clock select enumeration for periph_clk2_sel.
- enum _ccm_rootmux_pre_periph_clk_sel {

ccmRootmuxPrePeriphClkPll2 = 0U,
ccmRootmuxPrePeriphClkPll2Pfd2 = 1U,
ccmRootmuxPrePeriphClkPll2Pfd0 = 2U,
ccmRootmuxPrePeriphClkPll2Pfd2div2 = 3U }
Root clock select enumeration for pre_periph_clk_sel.

- enum _ccm_rootmux_pcie_axi_clk_sel {
ccmRootmuxPcieAxiClkAxiClk = 0U,
ccmRootmuxPcieAxiClkAhbClk = 1U }
Root clock select enumeration for pcie_axi_clk_sel.
- enum _ccm_rootmux_perclk_clk_sel {
ccmRootmuxPerclkClkIpgClkRoot = 0U,
ccmRootmuxPerclkClkOsc24m = 1U }
Root clock select enumeration for perclk_clk_sel.
- enum _ccm_rootmux_usdhc1_clk_sel {
ccmRootmuxUsdhc1ClkPll2Pfd2 = 0U,
ccmRootmuxUsdhc1ClkPll2Pfd0 = 1U }
Root clock select enumeration for usdhc1_clk_sel.
- enum _ccm_rootmux_usdhc2_clk_sel {
ccmRootmuxUsdhc2ClkPll2Pfd2 = 0U,
ccmRootmuxUsdhc2ClkPll2Pfd0 = 1U }
Root clock select enumeration for usdhc2_clk_sel.
- enum _ccm_rootmux_usdhc3_clk_sel {
ccmRootmuxUsdhc3ClkPll2Pfd2 = 0U,
ccmRootmuxUsdhc3ClkPll2Pfd0 = 1U }
Root clock select enumeration for usdhc3_clk_sel.
- enum _ccm_rootmux_usdhc4_clk_sel {
ccmRootmuxUsdhc4ClkPll2Pfd2 = 0U,
ccmRootmuxUsdhc4ClkPll2Pfd0 = 1U }
Root clock select enumeration for usdhc4_clk_sel.
- enum _ccm_rootmux_aclk_eim_slow_sel {
ccmRootmuxAclkEimSlowAxiClk = 0U,
ccmRootmuxAclkEimSlowPll3SwClk = 1U,
ccmRootmuxAclkEimSlowPll2Pfd2 = 2U,
ccmRootmuxAclkEimSlowPll3Pfd0 = 3U }
Root clock select enumeration for aclk_eim_slow_sel.
- enum _ccm_rootmux_gpu_axi_sel {
ccmRootmuxGpuAxiPll2Pfd2 = 0U,
ccmRootmuxGpuAxiPll3Pfd0 = 1U,
ccmRootmuxGpuAxiPll2Pfd1 = 2U,
ccmRootmuxGpuAxiPll2 = 3U }
Root clock select enumeration for gpu_axi_sel.
- enum _ccm_rootmux_gpu_core_sel {
ccmRootmuxGpuCorePll3Pfd1 = 0U,
ccmRootmuxGpuCorePll3Pfd0 = 1U,
ccmRootmuxGpuCorePll2 = 2U,
ccmRootmuxGpuCorePll2Pfd2 = 3U }
Root clock select enumeration for gpu_core_sel.

- enum _ccm_rootmux_vid_clk_sel {
  ccmRootmuxVidClkPll3Pfd1 = 0U,
  ccmRootmuxVidClkPll3 = 1U,
  ccmRootmuxVidClkPll3Pfd3 = 2U,
  ccmRootmuxVidClkPll4 = 3U,
  ccmRootmuxVidClkPll5 = 4U }
  Root clock select enumeration for vid_clk_sel.
- enum _ccm_rootmux_esai_clk_sel {
  ccmRootmuxEsaiClkPll4 = 0U,
  ccmRootmuxEsaiClkPll3Pfd2 = 1U,
  ccmRootmuxEsaiClkPll5 = 2U,
  ccmRootmuxEsaiClkPll3SwClk = 3U }
  Root clock select enumeration for esai_clk_sel.
- enum _ccm_rootmux_audio_clk_sel {
  ccmRootmuxAudioClkPll4 = 0U,
  ccmRootmuxAudioClkPll3Pfd2 = 1U,
  ccmRootmuxAudioClkPll5 = 2U,
  ccmRootmuxAudioClkPll3SwClk = 3U }
  Root clock select enumeration for audio_clk_sel.
- enum _ccm_rootmux_spdif0_clk_sel {
  ccmRootmuxSpdif0ClkPll4 = 0U,
  ccmRootmuxSpdif0ClkPll3Pfd2 = 1U,
  ccmRootmuxSpdif0ClkPll5 = 2U,
  ccmRootmuxSpdif0ClkPll3SwClk = 3U }
  Root clock select enumeration for spdif0_clk_sel.
- enum _ccm_rootmux_ssi1_clk_sel {
  ccmRootmuxSsi1ClkPll3Pfd2 = 0U,
  ccmRootmuxSsi1ClkPll5 = 1U,
  ccmRootmuxSsi1ClkPll4 = 2U }
  Root clock select enumeration for ssi1_clk_sel.
- enum _ccm_rootmux_ssi2_clk_sel {
  ccmRootmuxSsi2ClkPll3Pfd2 = 0U,
  ccmRootmuxSsi2ClkPll5 = 1U,
  ccmRootmuxSsi2ClkPll4 = 2U }
  Root clock select enumeration for ssi2_clk_sel.
- enum _ccm_rootmux_ssi3_clk_sel {
  ccmRootmuxSsi3ClkPll3Pfd2 = 0U,
  ccmRootmuxSsi3ClkPll5 = 1U,
  ccmRootmuxSsi3ClkPll4 = 2U }
  Root clock select enumeration for ssi3_clk_sel.
- enum _ccm_rootmux_lcdif2_clk_sel {
  ccmRootmuxLcdif2ClkLcdif2PreClk = 0U,
  ccmRootmuxLcdif2ClkIppDi0Clk = 1U,
  ccmRootmuxLcdif2ClkIppDi1Clk = 2U,
  ccmRootmuxLcdif2ClkLdbDi0Clk = 3U,
  ccmRootmuxLcdif2ClkLdbDi1Clk = 4U }

Root clock select enumeration for lcdif2_clk_sel.

- enum _ccm_rootmux_lcdif2_pre_clk_sel {
  ccmRootmuxLcdif2ClkPrePll2 = 0U,
  ccmRootmuxLcdif2ClkPrePll3Pfd3 = 1U,
  ccmRootmuxLcdif2ClkPrePll5 = 2U,
  ccmRootmuxLcdif2ClkPrePll2Pfd0 = 3U,
  ccmRootmuxLcdif2ClkPrePll2Pfd3 = 4U,
  ccmRootmuxLcdif2ClkPrePll3Pfd1 = 5U }

  Root clock select enumeration for lcdif2_pre_clk_sel.
- enum _ccm_rootmux_ldb_di1_clk_sel {
  ccmRootmuxLdbDi1ClkPll3SwClk = 0U,
  ccmRootmuxLdbDi1ClkPll2Pfd0 = 1U,
  ccmRootmuxLdbDi1ClkPll2Pfd2 = 2U,
  ccmRootmuxLdbDi1ClkPll2 = 3U,
  ccmRootmuxLdbDi1ClkPll3Pfd3 = 4U,
  ccmRootmuxLdbDi1ClkPll3Pfd2 = 5U }

  Root clock select enumeration for ldb_di1_clk_sel.
- enum _ccm_rootmux_ldb_di0_clk_sel {
  ccmRootmuxLdbDi0ClkPll5 = 0U,
  ccmRootmuxLdbDi0ClkPll2Pfd0 = 1U,
  ccmRootmuxLdbDi0ClkPll2Pfd2 = 2U,
  ccmRootmuxLdbDi0ClkPll2Pfd3 = 3U,
  ccmRootmuxLdbDi0ClkPll3Pfd1 = 4U,
  ccmRootmuxLdbDi0ClkPll3Pfd3 = 5U }

  Root clock select enumeration for ldb_di0_clk_sel.
- enum _ccm_rootmux_lcdif1_clk_sel {
  ccmRootmuxLcdif1ClkLcdif1PreClk = 0U,
  ccmRootmuxLcdif1ClkIppDi0Clk = 1U,
  ccmRootmuxLcdif1ClkIppDi1Clk = 2U,
  ccmRootmuxLcdif1ClkLdbDi0Clk = 3U,
  ccmRootmuxLcdif1ClkLdbDi1Clk = 4U }

  Root clock select enumeration for lcdif1_clk_sel.
- enum _ccm_rootmux_lcdif1_pre_clk_sel {
  ccmRootmuxLcdif1PreClkPll2 = 0U,
  ccmRootmuxLcdif1PreClkPll3Pfd3 = 1U,
  ccmRootmuxLcdif1PreClkPll5 = 2U,
  ccmRootmuxLcdif1PreClkPll2Pfd0 = 3U,
  ccmRootmuxLcdif1PreClkPll2Pfd1 = 4U,
  ccmRootmuxLcdif1PreClkPll3Pfd1 = 5U }

  Root clock select enumeration for lcdif1_pre_clk_sel.
- enum _ccm_rootmux_m4_clk_sel {
  ccmRootmuxM4ClkM4PreClk = 0U,
  ccmRootmuxM4ClkPll3Pfd3 = 1U,
  ccmRootmuxM4ClkIppDi0Clk = 2U,
  ccmRootmuxM4ClkIppDi1Clk = 3U,
  ccmRootmuxM4ClkLdbDi0Clk = 4U,

ccmRootmuxM4ClkLdbDi1Clk = 5U }
Root clock select enumeration for m4_clk_sel.

- enum _ccm_rootmux_m4_pre_clk_sel {
  ccmRootmuxM4PreClkPll2 = 0U,
  ccmRootmuxM4PreClkPll3SwClk = 1U,
  ccmRootmuxM4PreClkOsc24m = 2U,
  ccmRootmuxM4PreClkPll2Pfd0 = 3U,
  ccmRootmuxM4PreClkPll2Pfd2 = 4U,
  ccmRootmuxM4PreClkPll3Pfd3 = 5U }
  Root clock select enumeration for m4_pre_clk_sel.
- enum _ccm_rootmux_enet_clk_sel {
  ccmRootmuxEnetClkEnetPreClk = 0U,
  ccmRootmuxEnetClkIppDi0Clk = 1U,
  ccmRootmuxEnetClkIppDi1Clk = 2U,
  ccmRootmuxEnetClkLdbDi0Clk = 3U,
  ccmRootmuxEnetClkLdbDi1Clk = 4U }
  Root clock select enumeration for nent_clk_sel.
- enum _ccm_rootmux_enet_pre_clk_sel {
  ccmRootmuxEnetPreClkPll2 = 0U,
  ccmRootmuxEnetPreClkPll3SwClk = 1U,
  ccmRootmuxEnetPreClkPll5 = 2U,
  ccmRootmuxEnetPreClkPll2Pfd0 = 3U,
  ccmRootmuxEnetPreClkPll2Pfd2 = 4U,
  ccmRootmuxEnetPreClkPll3Pfd2 = 5U }
  Root clock select enumeration for enet_pre_clk_sel.
- enum _ccm_rootmux_qspi2_clk_sel {
  ccmRootmuxQspi2ClkPll2Pfd0 = 0U,
  ccmRootmuxQspi2ClkPll2 = 1U,
  ccmRootmuxQspi2ClkPll3SwClk = 2U,
  ccmRootmuxQspi2ClkPll2Pfd2 = 3U,
  ccmRootmuxQspi2ClkPll3Pfd3 = 4U }
  Root clock select enumeration for qspi2_clk_sel.
- enum _ccm_rootmux_display_clk_sel {
  ccmRootmuxDisplayClkPll2 = 0U,
  ccmRootmuxDisplayClkPll2Pfd2 = 1U,
  ccmRootmuxDisplayClkPll3SwClk = 2U,
  ccmRootmuxDisplayClkPll3Pfd1 = 3U }
  Root clock select enumeration for display_clk_sel.
- enum _ccm_rootmux_csi_clk_sel {
  ccmRootmuxCsiClkOSC24m = 0U,
  ccmRootmuxCsiClkPll2Pfd2 = 1U,
  ccmRootmuxCsiClkPll3SwClkDiv2 = 2U,
  ccmRootmuxCsiClkPll3Pfd1 = 3U }
  Root clock select enumeration for csi_clk_sel.
- enum _ccm_rootmux_can_clk_sel {

ccmRootmuxCanClkPll3SwClkDiv8 = 0U,
ccmRootmuxCanClkOsc24m = 1U,
ccmRootmuxCanClkPll3SwClkDiv6 = 2U,
ccmRootmuxCanClkDisableFlexcanClk = 3U }
Root clock select enumeration for can_clk_sel.

- enum _ccm_rootmux_ecspi_clk_sel {
  ccmRootmuxEcspiClkPll3SwClkDiv8 = 0U,
  ccmRootmuxEcspiClkOsc24m = 1U }
  Root clock select enumeration for ecspi_clk_sel.
- enum _ccm_rootmux_uart_clk_sel {
  ccmRootmuxUartClkPll3SwClkDiv6 = 0U,
  ccmRootmuxUartClkOsc24m = 1U }
  Root clock select enumeration for uart_clk_sel.
- enum _ccm_root_div_control {
  ccmRootArmPodf = CCM_TUPLE(CACRR, CCM_CACRR_arm_podf_SHIFT, CCM_CACRR_-arm_podf_MASK),
  ccmRootFabricMmdcPodf = CCM_TUPLE(CBCDR, CCM_CBCDR_fabric_mmdc_podf_SHIFT, CCM_CBCDR_fabric_mmdc_podf_MASK),
  ccmRootPeriph2Clk2Podf = CCM_TUPLE(CBCDR, CCM_CBCDR_periph2_clk2_podf_SHIFT, CCM_CBCDR_periph2_clk2_podf_MASK),
  ccmRootOcramPodf = CCM_TUPLE(CBCDR, CCM_CBCDR_ocram_podf_SHIFT, CCM_CBC-DR_ocram_podf_MASK),
  ccmRootAhbPodf = CCM_TUPLE(CBCDR, CCM_CBCDR_ahb_podf_SHIFT, CCM_CBCDR_-ahb_podf_MASK),
  ccmRootPeriphClk2Podf = CCM_TUPLE(CBCDR, CCM_CBCDR_periph_clk2_podf_SHIFT, C-CM_CBCDR_periph_clk2_podf_MASK),
  ccmRootPerclkPodf = CCM_TUPLE(CSCMR1, CCM_CSCMR1_perclk_podf_SHIFT, CCM_CS-CMR1_perclk_podf_MASK),
  ccmRootIpgPodf = CCM_TUPLE(CBCDR, CCM_CBCDR_ipg_podf_SHIFT, CCM_CBCDR_-ipg_podf_MASK),
  ccmRootUsdhc1Podf = CCM_TUPLE(CSCDR1, CCM_CSCDR1_usdhc1_podf_SHIFT, CCM_C-SCDR1_usdhc1_podf_MASK),
  ccmRootUsdhc2Podf = CCM_TUPLE(CSCDR1, CCM_CSCDR1_usdhc2_podf_SHIFT, CCM_C-SCDR1_usdhc2_podf_MASK),
  ccmRootUsdhc3Podf = CCM_TUPLE(CSCDR1, CCM_CSCDR1_usdhc3_podf_SHIFT, CCM_C-SCDR1_usdhc3_podf_MASK),
  ccmRootUsdhc4Podf = CCM_TUPLE(CSCDR1, CCM_CSCDR1_usdhc4_podf_SHIFT, CCM_C-SCDR1_usdhc4_podf_MASK),
  ccmRootAclkEimSlowPodf = CCM_TUPLE(CSCMR1, CCM_CSCMR1_aclk_eim_slow_podf_S-HIFT, CCM_CSCMR1_aclk_eim_slow_podf_MASK),
  ccmRootGpuAxiPodf = CCM_TUPLE(CBCMR, CCM_CBCMR_gpu_axi_podf_SHIFT, CCM_C-BCMR_gpu_axi_podf_MASK),
  ccmRootGpuCorePodf = CCM_TUPLE(CBCMR, CCM_CBCMR_gpu_core_podf_SHIFT, CCM-_CBCMR_gpu_core_podf_MASK),
  ccmRootVidClkPodf = CCM_TUPLE(CSCMR2, CCM_CSCMR2_vid_clk_podf_SHIFT, CCM_-

CSCMR2_vid_clk_podf_MASK),
ccmRootEsaiClkPodf = CCM_TUPLE(CS1CDR, CCM_CS1CDR_esai_clk_podf_SHIFT, CCM_-CS1CDR_esai_clk_podf_MASK),
ccmRootEsaiClkPred = CCM_TUPLE(CS1CDR, CCM_CS1CDR_esai_clk_pred_SHIFT, CCM_-CS1CDR_esai_clk_pred_MASK),
ccmRootAudioClkPodf = CCM_TUPLE(CDCDR, CCM_CDCDR_audio_clk_podf_SHIFT, CCM-_CDCDR_audio_clk_podf_MASK),
ccmRootAudioClkPred = CCM_TUPLE(CDCDR, CCM_CDCDR_audio_clk_pred_SHIFT, CCM-_CDCDR_audio_clk_pred_MASK),
ccmRootSpdif0ClkPodf = CCM_TUPLE(CDCDR, CCM_CDCDR_spdif0_clk_podf_SHIFT, CC-M_CDCDR_spdif0_clk_podf_MASK),
ccmRootSpdif0ClkPred = CCM_TUPLE(CDCDR, CCM_CDCDR_spdif0_clk_pred_SHIFT, CC-M_CDCDR_spdif0_clk_pred_MASK),
ccmRootSsi1ClkPodf = CCM_TUPLE(CS1CDR, CCM_CS1CDR_ssi1_clk_podf_SHIFT, CCM_-CS1CDR_ssi1_clk_podf_MASK),
ccmRootSsi1ClkPred = CCM_TUPLE(CS1CDR, CCM_CS1CDR_ssi1_clk_pred_SHIFT, CCM_-CS1CDR_ssi1_clk_pred_MASK),
ccmRootSsi2ClkPodf = CCM_TUPLE(CS2CDR, CCM_CS2CDR_ssi2_clk_podf_SHIFT, CCM_-CS2CDR_ssi2_clk_podf_MASK),
ccmRootSsi2ClkPred = CCM_TUPLE(CS2CDR, CCM_CS2CDR_ssi2_clk_pred_SHIFT, CCM_-CS2CDR_ssi2_clk_pred_MASK),
ccmRootSsi3ClkPodf = CCM_TUPLE(CS1CDR, CCM_CS1CDR_ssi3_clk_podf_SHIFT, CCM_-CS1CDR_ssi3_clk_podf_MASK),
ccmRootSsi3ClkPred = CCM_TUPLE(CS1CDR, CCM_CS1CDR_ssi3_clk_pred_SHIFT, CCM_-CS1CDR_ssi3_clk_pred_MASK),
ccmRootLcdif2Podf = CCM_TUPLE(CSCMR1, CCM_CSCMR1_lcdif2_podf_SHIFT, CCM_CS-CMR1_lcdif2_podf_MASK),
ccmRootLcdif2Pred = CCM_TUPLE(CSCDR2, CCM_CSCDR2_lcdif2_pred_SHIFT, CCM_CSC-DR2_lcdif2_pred_MASK),
ccmRootLdbDi1Div = CCM_TUPLE(CSCMR2, CCM_CSCMR2_ldb_di1_div_SHIFT, CCM_CS-CMR2_ldb_di1_div_MASK),
ccmRootLdbDi0Div = CCM_TUPLE(CSCMR2, CCM_CSCMR2_ldb_di0_div_SHIFT, CCM_CS-CMR2_ldb_di0_div_MASK),
ccmRootLcdif1Podf = CCM_TUPLE(CBCMR, CCM_CBCMR_lcdif1_podf_SHIFT, CCM_CBC-MR_lcdif1_podf_MASK),
ccmRootLcdif1Pred = CCM_TUPLE(CSCDR2, CCM_CSCDR2_lcdif1_pred_SHIFT, CCM_CSC-DR2_lcdif1_pred_MASK),
ccmRootM4Podf = CCM_TUPLE(CHSCCDR, CCM_CHSCCDR_m4_podf_SHIFT, CCM_CHS-CCDR_m4_podf_MASK),
ccmRootEnetPodf = CCM_TUPLE(CHSCCDR, CCM_CHSCCDR_enet_podf_SHIFT, CCM_CH-SCCDR_enet_podf_MASK),
ccmRootQspi1Podf = CCM_TUPLE(CSCMR1, CCM_CSCMR1_qspi1_podf_SHIFT, CCM_CSC-MR1_qspi1_podf_MASK),
ccmRootQspi2ClkPodf = CCM_TUPLE(CS2CDR, CCM_CS2CDR_qspi2_clk_podf_SHIFT, CC-

M_CS2CDR_qspi2_clk_podf_MASK),
ccmRootQspi2ClkPred = CCM_TUPLE(CS2CDR, CCM_CS2CDR_qspi2_clk_pred_SHIFT, CCM_CS2CDR_qspi2_clk_pred_MASK),
ccmRootDisplayPodf = CCM_TUPLE(CSCDR3, CCM_CSCDR3_display_podf_SHIFT, CCM_CSCDR3_display_podf_MASK),
ccmRootCsiPodf = CCM_TUPLE(CSCDR3, CCM_CSCDR3_csi_podf_SHIFT, CCM_CSCDR3_csi_podf_MASK),
ccmRootCanClkPodf = CCM_TUPLE(CSCMR2, CCM_CSCMR2_can_clk_podf_SHIFT, CCM_CSCMR2_can_clk_podf_MASK),
ccmRootEcspiClkPodf = CCM_TUPLE(CSCDR2, CCM_CSCDR2_ecspi_clk_podf_SHIFT, CCM_CSCDR2_ecspi_clk_podf_MASK),
ccmRootUartClkPodf = CCM_TUPLE(CSCDR1, CCM_CSCDR1_uart_clk_podf_SHIFT, CCM_CSCDR1_uart_clk_podf_MASK) }

Root control names for root divider setting.

- enum _ccm_ccgr_gate {
ccmCcgrGateAipsTz1Clk = CCM_TUPLE(CCGR0, CCM_CCGR0_CG0_SHIFT, CCM_CCGR0_CG0_MASK),
ccmCcgrGateAipsTz2Clk = CCM_TUPLE(CCGR0, CCM_CCGR0_CG1_SHIFT, CCM_CCGR0_CG1_MASK),
ccmCcgrGateApbhdmaHclk = CCM_TUPLE(CCGR0, CCM_CCGR0_CG2_SHIFT, CCM_CCGR0_CG2_MASK),
ccmCcgrGateAsrcClk = CCM_TUPLE(CCGR0, CCM_CCGR0_CG3_SHIFT, CCM_CCGR0_CG3_MASK),
ccmCcgrGateCaamSecureMemClk = CCM_TUPLE(CCGR0, CCM_CCGR0_CG4_SHIFT, CCM_CCGR0_CG4_MASK),
ccmCcgrGateCaamWrapperAclk = CCM_TUPLE(CCGR0, CCM_CCGR0_CG5_SHIFT, CCM_CCGR0_CG5_MASK),
ccmCcgrGateCaamWrapperIpg = CCM_TUPLE(CCGR0, CCM_CCGR0_CG6_SHIFT, CCM_CCGR0_CG6_MASK),
ccmCcgrGateCan1Clk = CCM_TUPLE(CCGR0, CCM_CCGR0_CG7_SHIFT, CCM_CCGR0_CG7_MASK),
ccmCcgrGateCan1SerialClk = CCM_TUPLE(CCGR0, CCM_CCGR0_CG8_SHIFT, CCM_CCGR0_CG8_MASK),
ccmCcgrGateCan2Clk = CCM_TUPLE(CCGR0, CCM_CCGR0_CG9_SHIFT, CCM_CCGR0_CG9_MASK),
ccmCcgrGateCan2SerialClk = CCM_TUPLE(CCGR0, CCM_CCGR0_CG10_SHIFT, CCM_CCGR0_CG10_MASK),
ccmCcgrGateArmDbgClk = CCM_TUPLE(CCGR0, CCM_CCGR0_CG11_SHIFT, CCM_CCGR0_CG11_MASK),
ccmCcgrGateDcic1Clk = CCM_TUPLE(CCGR0, CCM_CCGR0_CG12_SHIFT, CCM_CCGR0_CG12_MASK),
ccmCcgrGateDcic2Clk = CCM_TUPLE(CCGR0, CCM_CCGR0_CG13_SHIFT, CCM_CCGR0_CG13_MASK),
ccmCcgrGateAipsTz3Clk = CCM_TUPLE(CCGR0, CCM_CCGR0_CG15_SHIFT, CCM_CCG-

R0_CG15_MASK),
ccmCcgrGateEcspi1Clk = CCM_TUPLE(CCGR1, CCM_CCGR1_CG0_SHIFT, CCM_CCGR1_-CG0_MASK),
ccmCcgrGateEcspi2Clk = CCM_TUPLE(CCGR1, CCM_CCGR1_CG1_SHIFT, CCM_CCGR1_-CG1_MASK),
ccmCcgrGateEcspi3Clk = CCM_TUPLE(CCGR1, CCM_CCGR1_CG2_SHIFT, CCM_CCGR1_-CG2_MASK),
ccmCcgrGateEcspi4Clk = CCM_TUPLE(CCGR1, CCM_CCGR1_CG3_SHIFT, CCM_CCGR1_-CG3_MASK),
ccmCcgrGateEcspi5Clk = CCM_TUPLE(CCGR1, CCM_CCGR1_CG4_SHIFT, CCM_CCGR1_-CG4_MASK),
ccmCcgrGateEpit1Clk = CCM_TUPLE(CCGR1, CCM_CCGR1_CG6_SHIFT, CCM_CCGR1_C-G6_MASK),
ccmCcgrGateEpit2Clk = CCM_TUPLE(CCGR1, CCM_CCGR1_CG7_SHIFT, CCM_CCGR1_C-G7_MASK),
ccmCcgrGateEsaiClk = CCM_TUPLE(CCGR1, CCM_CCGR1_CG8_SHIFT, CCM_CCGR1_C-G8_MASK),
ccmCcgrGateWakeupClk = CCM_TUPLE(CCGR1, CCM_CCGR1_CG9_SHIFT, CCM_CCGR1-_CG9_MASK),
ccmCcgrGateGptClk = CCM_TUPLE(CCGR1, CCM_CCGR1_CG10_SHIFT, CCM_CCGR1_C-G10_MASK),
ccmCcgrGateGptSerialClk = CCM_TUPLE(CCGR1, CCM_CCGR1_CG11_SHIFT, CCM_CCG-R1_CG11_MASK),
ccmCcgrGateGpuClk = CCM_TUPLE(CCGR1, CCM_CCGR1_CG13_SHIFT, CCM_CCGR1_C-G13_MASK),
ccmCcgrGateOcramSClk = CCM_TUPLE(CCGR1, CCM_CCGR1_CG14_SHIFT, CCM_CCG-R1_CG14_MASK),
ccmCcgrGateCanfdClk = CCM_TUPLE(CCGR1, CCM_CCGR1_CG15_SHIFT, CCM_CCGR1_-CG15_MASK),
ccmCcgrGateCsiClk = CCM_TUPLE(CCGR2, CCM_CCGR2_CG1_SHIFT, CCM_CCGR2_C-G1_MASK),
ccmCcgrGateI2c1Serialclk = CCM_TUPLE(CCGR2, CCM_CCGR2_CG3_SHIFT, CCM_CCG-R2_CG3_MASK),
ccmCcgrGateI2c2Serialclk = CCM_TUPLE(CCGR2, CCM_CCGR2_CG4_SHIFT, CCM_CCG-R2_CG4_MASK),
ccmCcgrGateI2c3Serialclk = CCM_TUPLE(CCGR2, CCM_CCGR2_CG5_SHIFT, CCM_CCG-R2_CG5_MASK),
ccmCcgrGateIimClk = CCM_TUPLE(CCGR2, CCM_CCGR2_CG6_SHIFT, CCM_CCGR2_C-G6_MASK),
ccmCcgrGateIomuxIptClkIo = CCM_TUPLE(CCGR2, CCM_CCGR2_CG7_SHIFT, CCM_CCG-R2_CG7_MASK),
ccmCcgrGateIpmux1Clk = CCM_TUPLE(CCGR2, CCM_CCGR2_CG8_SHIFT, CCM_CCGR2_-CG8_MASK),
ccmCcgrGateIpmux2Clk = CCM_TUPLE(CCGR2, CCM_CCGR2_CG9_SHIFT, CCM_CCGR2_-

CG9_MASK),
ccmCcgrGateIpmux3Clk = CCM_TUPLE(CCGR2, CCM_CCGR2_CG10_SHIFT, CCM_CCGR2-_CG10_MASK),
ccmCcgrGateIpsyncIp2apbtTasc1 = CCM_TUPLE(CCGR2, CCM_CCGR2_CG11_SHIFT, CCM-_CCGR2_CG11_MASK),
ccmCcgrGateLcdClk = CCM_TUPLE(CCGR2, CCM_CCGR2_CG14_SHIFT, CCM_CCGR2_C-G14_MASK),
ccmCcgrGatePxpClk = CCM_TUPLE(CCGR2, CCM_CCGR2_CG15_SHIFT, CCM_CCGR2_C-G15_MASK),
ccmCcgrGateM4Clk = CCM_TUPLE(CCGR3, CCM_CCGR3_CG1_SHIFT, CCM_CCGR3_C-G1_MASK),
ccmCcgrGateEnetClk = CCM_TUPLE(CCGR3, CCM_CCGR3_CG2_SHIFT, CCM_CCGR3_C-G2_MASK),
ccmCcgrGateDispAxiClk = CCM_TUPLE(CCGR3, CCM_CCGR3_CG3_SHIFT, CCM_CCGR3-_CG3_MASK),
ccmCcgrGateLcdif2PixClk = CCM_TUPLE(CCGR3, CCM_CCGR3_CG4_SHIFT, CCM_CCG-R3_CG4_MASK),
ccmCcgrGateLcdif1PixClk = CCM_TUPLE(CCGR3, CCM_CCGR3_CG5_SHIFT, CCM_CCG-R3_CG5_MASK),
ccmCcgrGateLdbDi0Clk = CCM_TUPLE(CCGR3, CCM_CCGR3_CG6_SHIFT, CCM_CCGR3_-CG6_MASK),
ccmCcgrGateQspi1Clk = CCM_TUPLE(CCGR3, CCM_CCGR3_CG7_SHIFT, CCM_CCGR3_C-G7_MASK),
ccmCcgrGateMlbClk = CCM_TUPLE(CCGR3, CCM_CCGR3_CG9_SHIFT, CCM_CCGR3_C-G9_MASK),
ccmCcgrGateMmdcCoreAclkFastP0 = CCM_TUPLE(CCGR3, CCM_CCGR3_CG10_SHIFT, CC-M_CCGR3_CG10_MASK),
ccmCcgrGateMmdcCoreIpgClkP0 = CCM_TUPLE(CCGR3, CCM_CCGR3_CG12_SHIFT, CCM-_CCGR3_CG12_MASK),
ccmCcgrGateMmdcCoreIpgClkP1 = CCM_TUPLE(CCGR3, CCM_CCGR3_CG13_SHIFT, CCM-_CCGR3_CG13_MASK),
ccmCcgrGateOcramClk = CCM_TUPLE(CCGR3, CCM_CCGR3_CG14_SHIFT, CCM_CCGR3-_CG14_MASK),
ccmCcgrGatePcieRoot = CCM_TUPLE(CCGR4, CCM_CCGR4_CG0_SHIFT, CCM_CCGR4_C-G0_MASK),
ccmCcgrGateQspi2Clk = CCM_TUPLE(CCGR4, CCM_CCGR4_CG5_SHIFT, CCM_CCGR4_C-G5_MASK),
ccmCcgrGatePl301Mx6qper1Bch = CCM_TUPLE(CCGR4, CCM_CCGR4_CG6_SHIFT, CCM_-CCGR4_CG6_MASK),
ccmCcgrGatePl301Mx6qper2Main = CCM_TUPLE(CCGR4, CCM_CCGR4_CG7_SHIFT, CCM-_CCGR4_CG7_MASK),
ccmCcgrGatePwm1Clk = CCM_TUPLE(CCGR4, CCM_CCGR4_CG8_SHIFT, CCM_CCGR4_-CG8_MASK),
ccmCcgrGatePwm2Clk = CCM_TUPLE(CCGR4, CCM_CCGR4_CG9_SHIFT, CCM_CCGR4_-

CG9_MASK),
ccmCcgrGatePwm3Clk = CCM_TUPLE(CCGR4, CCM_CCGR4_CG10_SHIFT, CCM_CCGR4_-CG10_MASK),
ccmCcgrGatePwm4Clk = CCM_TUPLE(CCGR4, CCM_CCGR4_CG11_SHIFT, CCM_CCGR4_-CG11_MASK),
ccmCcgrGateRawnandUBchInptApb = CCM_TUPLE(CCGR4, CCM_CCGR4_CG12_SHIFT, C-CM_CCGR4_CG12_MASK),
ccmCcgrGateRawnandUGpmiBch = CCM_TUPLE(CCGR4, CCM_CCGR4_CG13_SHIFT, CCM-_CCGR4_CG13_MASK),
ccmCcgrGateRawnandUGpmiGpmiIo = CCM_TUPLE(CCGR4, CCM_CCGR4_CG14_SHIFT, C-CM_CCGR4_CG14_MASK),
ccmCcgrGateRawnandUGpmiInpApb = CCM_TUPLE(CCGR4, CCM_CCGR4_CG15_SHIFT, C-CM_CCGR4_CG15_MASK),
ccmCcgrGateRomClk = CCM_TUPLE(CCGR5, CCM_CCGR5_CG0_SHIFT, CCM_CCGR5_C-G0_MASK),
ccmCcgrGateSdmaClk = CCM_TUPLE(CCGR5, CCM_CCGR5_CG3_SHIFT, CCM_CCGR5_C-G3_MASK),
ccmCcgrGateSpbaClk = CCM_TUPLE(CCGR5, CCM_CCGR5_CG6_SHIFT, CCM_CCGR5_C-G6_MASK),
ccmCcgrGateSpdifAudioClk = CCM_TUPLE(CCGR5, CCM_CCGR5_CG7_SHIFT, CCM_CCG-R5_CG7_MASK),
ccmCcgrGateSsi1Clk = CCM_TUPLE(CCGR5, CCM_CCGR5_CG9_SHIFT, CCM_CCGR5_C-G9_MASK),
ccmCcgrGateSsi2Clk = CCM_TUPLE(CCGR5, CCM_CCGR5_CG10_SHIFT, CCM_CCGR5_C-G10_MASK),
ccmCcgrGateSsi3Clk = CCM_TUPLE(CCGR5, CCM_CCGR5_CG11_SHIFT, CCM_CCGR5_C-G11_MASK),
ccmCcgrGateUartClk = CCM_TUPLE(CCGR5, CCM_CCGR5_CG12_SHIFT, CCM_CCGR5_C-G12_MASK),
ccmCcgrGateUartSerialClk = CCM_TUPLE(CCGR5, CCM_CCGR5_CG13_SHIFT, CCM_CCG-R5_CG13_MASK),
ccmCcgrGateSai1Clk = CCM_TUPLE(CCGR5, CCM_CCGR5_CG14_SHIFT, CCM_CCGR5_C-G14_MASK),
ccmCcgrGateSai2Clk = CCM_TUPLE(CCGR5, CCM_CCGR5_CG15_SHIFT, CCM_CCGR5_C-G15_MASK),
ccmCcgrGateUsboh3Clk = CCM_TUPLE(CCGR6, CCM_CCGR6_CG0_SHIFT, CCM_CCGR6_-CG0_MASK),
ccmCcgrGateUsdhc1Clk = CCM_TUPLE(CCGR6, CCM_CCGR6_CG1_SHIFT, CCM_CCGR6_-CG1_MASK),
ccmCcgrGateUsdhc2Clk = CCM_TUPLE(CCGR6, CCM_CCGR6_CG2_SHIFT, CCM_CCGR6_-CG2_MASK),
ccmCcgrGateUsdhc3Clk = CCM_TUPLE(CCGR6, CCM_CCGR6_CG3_SHIFT, CCM_CCGR6_-CG3_MASK),
ccmCcgrGateUsdhc4Clk = CCM_TUPLE(CCGR6, CCM_CCGR6_CG4_SHIFT, CCM_CCGR6_-

CG4_MASK),
ccmCcgrGateEimSlowClk = CCM_TUPLE(CCGR6, CCM_CCGR6_CG5_SHIFT, CCM_CCGR6_CG5_MASK),
ccmCcgrGatePwm8Clk = CCM_TUPLE(CCGR6, CCM_CCGR6_CG8_SHIFT, CCM_CCGR6_CG8_MASK),
ccmCcgrGateVadcClk = CCM_TUPLE(CCGR6, CCM_CCGR6_CG10_SHIFT, CCM_CCGR6_CG10_MASK),
ccmCcgrGateGisClk = CCM_TUPLE(CCGR6, CCM_CCGR6_CG11_SHIFT, CCM_CCGR6_CG11_MASK),
ccmCcgrGateI2c4SerialClk = CCM_TUPLE(CCGR6, CCM_CCGR6_CG12_SHIFT, CCM_CCGR6_CG12_MASK),
ccmCcgrGatePwm5Clk = CCM_TUPLE(CCGR6, CCM_CCGR6_CG13_SHIFT, CCM_CCGR6_CG13_MASK),
ccmCcgrGatePwm6Clk = CCM_TUPLE(CCGR6, CCM_CCGR6_CG14_SHIFT, CCM_CCGR6_CG14_MASK),
ccmCcgrGatePwm7Clk = CCM_TUPLE(CCGR6, CCM_CCGR6_CG15_SHIFT, CCM_CCGR6_CG15_MASK) }
CCM CCGR gate control for each module independently.

- enum _ccm_gate_value {
  ccmClockNotNeeded = 0U,
  ccmClockNeededRun = 1U,
  ccmClockNeededAll = 3U }
  CCM gate control value.
- enum _ccm_overrided_enable_signal {
  ccmOverridedSignalFromGpt = 1U << 5,
  ccmOverridedSignalFromEpit = 1U << 6,
  ccmOverridedSignalFromUsdhc = 1U << 7,
  ccmOverridedSignalFromGpu = 1U << 10,
  ccmOverridedSignalFromCan2Cpi = 1U << 28,
  ccmOverridedSignalFromCan1Cpi = 1U << 30 }
  CCM overrided clock enable signal from module.

CCM Root Clock Setting

- static void CCM_SetRootMux (CCM_Type *base, uint32_t ccmRootClk, uint32_t mux)
  Set clock root mux.
- static uint32_t CCM_GetRootMux (CCM_Type *base, uint32_t ccmRootClk)
  Get clock root mux.
- static void CCM_SetRootDivider (CCM_Type *base, uint32_t ccmRootDiv, uint32_t div)
  Set root clock divider.
- static uint32_t CCM_GetRootDivider (CCM_Type *base, uint32_t ccmRootDiv)
  Get root clock divider.
- void CCM_SetMmdcHandshakeMask (CCM_Type *base, bool mask)
  Set handshake mask of MMDC module.

CCM Gate Control

- static void CCM_ControlGate (CCM_Type *base, uint32_t ccmGate, uint32_t control)
  Set CCGR gate control for each module User should set specific gate for each module according to the description of the table of system clocks, gating and override in CCM chapter of reference manual.
- void CCM_SetClockEnableSignalOverrided (CCM_Type *base, uint32_t signal, bool control)
  Set override or do not override clock enable signal from module.

4.3.5 Enumeration Type Documentation

4.3.5.1 enum _ccm_root_clock_control

These constants define the root control names for root clock 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.

Enumerator

ccmRootPll1SwClkSel PLL1 SW Clock control name.
ccmRootStepSel Step SW Clock control name.
ccmRootPeriph2ClkSel Peripheral2 Clock control name.
ccmRootPrePeriph2ClkSel Pre Peripheral2 Clock control name.
ccmRootPeriph2Clk2Sel Peripheral2 Clock2 Clock control name.
ccmRootPll3SwClkSel PLL3 SW Clock control name.
ccmRootOcramClkSel OCRAM Clock control name.
ccmRootOcramAltClkSel OCRAM ALT Clock control name.
ccmRootPeriphClkSel Peripheral Clock control name.
ccmRootPeriphClk2Sel Peripheral Clock2 control name.
ccmRootPrePeriphClkSel Pre Peripheral Clock control name.
ccmRootPcieAxiClkSel PCIE AXI Clock control name.
ccmRootPerclkClkSel Pre Clock control name.
ccmRootUsdhc1ClkSel USDHC1 Clock control name.
ccmRootUsdhc2ClkSel USDHC2 Clock control name.
ccmRootUsdhc3ClkSel USDHC3 Clock control name.
ccmRootUsdhc4ClkSel USDHC4 Clock control name.
ccmRootAclkEimSlowSel ACLK EIM SLOW Clock control name.
ccmRootGpuAxiSel GPU AXI Clock control name.
ccmRootGpuCoreSel GPU Core Clock control name.
ccmRootVidClkSel VID Clock control name.
ccmRootEsaiClkSel ESAI Clock control name.
ccmRootAudioClkSel AUDIO Clock control name.
ccmRootSpdif0ClkSel SPDIF0 Clock control name.
ccmRootSsi1ClkSel SSI1 Clock control name.

ccmRootSsi2ClkSel SSI2 Clock control name.
ccmRootSsi3ClkSel SSI3 Clock control name.
ccmRootLcdif2ClkSel LCDIF2 Clock control name.
ccmRootLcdif2PreClkSel LCDIF2 Pre Clock control name.
ccmRootLdbDi1ClkSel LDB DI1 Clock control name.
ccmRootLdbDi0ClkSel LDB DI0 Clock control name.
ccmRootLcdif1ClkSel LCDIF1 Clock control name.
ccmRootLcdif1PreClkSel LCDIF1 Pre Clock control name.
ccmRootM4ClkSel M4 Clock control name.
ccmRootM4PreClkSel M4 Pre Clock control name.
ccmRootEnetClkSel Ethernet Clock control name.
ccmRootEnetPreClkSel Ethernet Pre Clock control name.
ccmRootQspi2ClkSel QSPI2 Clock control name.
ccmRootDisplayClkSel Display Clock control name.
ccmRootCsiClkSel CSI Clock control name.
ccmRootCanClkSel CAN Clock control name.
ccmRootEcspiClkSel ECSPI Clock control name.
ccmRootUartClkSel UART Clock control name.

4.3.5.2 enum _ccm_rootmux_pll1_sw_clk_sel

Enumerator

ccmRootmuxPll1SwClkPll1MainClk PLL1 SW Clock from PLL1 Main Clock.
ccmRootmuxPll1SwClkStepClk PLL1 SW Clock from Step Clock.

4.3.5.3 enum _ccm_rootmux_step_sel

Enumerator

ccmRootmuxStepOsc24m Step Clock from OSC 24M.
ccmRootmuxStepPll2Pfd2 Step Clock from PLL2 PFD2.

4.3.5.4 enum _ccm_rootmux_periph2_clk_sel

Enumerator

ccmRootmuxPeriph2ClkPrePeriph2Clk Peripheral2 Clock from Pre Peripheral2 Clock.
ccmRootmuxPeriph2ClkPeriph2Clk2 Peripheral2 Clock from Peripheral2.

4.3.5.5 enum _ccm_rootmux_pre_periph2_clk_sel

Enumerator

ccmRootmuxPrePeriph2ClkPll2 Pre Peripheral2 Clock from PLL2.
ccmRootmuxPrePeriph2ClkPll2Pfd2 Pre Peripheral2 Clock from PLL2 PFD2.
ccmRootmuxPrePeriph2ClkPll2Pfd0 Pre Peripheral2 Clock from PLL2 PFD0.
ccmRootmuxPrePeriph2ClkPll4 Pre Peripheral2 Clock from PLL4.

4.3.5.6 enum _ccm_rootmux_periph2_clk2_sel

Enumerator

ccmRootmuxPeriph2Clk2Pll3SwClk Peripheral2 Clock from PLL3 SW Clock.
ccmRootmuxPeriph2Clk2Osc24m Peripheral2 Clock from OSC 24M.

4.3.5.7 enum _ccm_rootmux_pll3_sw_clk_sel

Enumerator

ccmRootmuxPll3SwClkPll3 PLL3 SW Clock from PLL3.
ccmRootmuxPll3SwClkPll3BypassClk PLL3 SW Clock from PLL3 Bypass Clock.

4.3.5.8 enum _ccm_rootmux_ocram_clk_sel

Enumerator

ccmRootmuxOcramClkPeriphClk OCRAM Clock from Peripheral Clock.
ccmRootmuxOcramClkOcramAltClk OCRAM Clock from OCRAM ALT Clock.

4.3.5.9 enum _ccm_rootmux_ocram_alt_clk_sel

Enumerator

ccmRootmuxOcramAltClkPll2Pfd2 OCRAM ALT Clock from PLL2 PFD2.
ccmRootmuxOcramAltClkPll3Pfd1 OCRAM ALT Clock from PLL3 PFD1.

4.3.5.10 enum _ccm_rootmux_periph_clk_sel

Enumerator

ccmRootmuxPeriphClkPrePeriphClkSel Peripheral Clock from Pre Peripheral .
ccmRootmuxPeriphClkPeriphClk2Sel Peripheral Clock from Peripheral2.

4.3.5.11 enum _ccm_rootmux_periph_clk2_sel

Enumerator

ccmRootmuxPeriphClk2Pll3SwClk Peripheral Clock2 from from PLL3 SW Clock.
ccmRootmuxPeriphClk2OSC24m Peripheral Clock2 from OSC 24M.
ccmRootmuxPeriphClk2Pll2 Peripheral Clock2 from PLL2.

4.3.5.12 enum _ccm_rootmux_pre_periph_clk_sel

Enumerator

ccmRootmuxPrePeriphClkPll2 Pre Peripheral Clock from PLL2.
ccmRootmuxPrePeriphClkPll2Pfd2 Pre Peripheral Clock from PLL2 PFD2.
ccmRootmuxPrePeriphClkPll2Pfd0 Pre Peripheral Clock from PLL2 PFD0.
ccmRootmuxPrePeriphClkPll2Pfd2div2 Pre Peripheral Clock from PLL2 PFD2 divided by 2.

4.3.5.13 enum _ccm_rootmux_pcie_axi_clk_sel

Enumerator

ccmRootmuxPcieAxiClkAxiClk PCIE AXI Clock from AXI Clock.
ccmRootmuxPcieAxiClkAhbClk PCIE AXI Clock from AHB Clock.

4.3.5.14 enum _ccm_rootmux_perclk_clk_sel

Enumerator

ccmRootmuxPerclkClkIpgClkRoot Perclk from IPG Clock Root.
ccmRootmuxPerclkClkOsc24m Perclk from OSC 24M.

4.3.5.15 enum _ccm_rootmux_usdhc1_clk_sel

Enumerator

ccmRootmuxUsdhc1ClkPll2Pfd2 USDHC1 Clock from PLL2 PFD2.
ccmRootmuxUsdhc1ClkPll2Pfd0 USDHC1 Clock from PLL2 PFD0.

4.3.5.16 enum _ccm_rootmux_usdhc2_clk_sel

Enumerator

ccmRootmuxUsdhc2ClkPll2Pfd2 USDHC2 Clock from PLL2 PFD2.
ccmRootmuxUsdhc2ClkPll2Pfd0 USDHC2 Clock from PLL2 PFD0.

4.3.5.17 enum _ccm_rootmux_usdhc3_clk_sel

Enumerator

ccmRootmuxUsdhc3ClkPll2Pfd2 USDHC3 Clock from PLL2 PFD2.
ccmRootmuxUsdhc3ClkPll2Pfd0 USDHC3 Clock from PLL2 PFD0.

4.3.5.18 enum _ccm_rootmux_usdhc4_clk_sel

Enumerator

ccmRootmuxUsdhc4ClkPll2Pfd2 USDHC4 Clock from PLL2 PFD2.
ccmRootmuxUsdhc4ClkPll2Pfd0 USDHC4 Clock from PLL2 PFD0.

4.3.5.19 enum _ccm_rootmux_aclk_eim_slow_sel

Enumerator

ccmRootmuxAclkEimSlowAxiClk Aclk EimSlow Clock from AXI Clock.
ccmRootmuxAclkEimSlowPll3SwClk Aclk EimSlow Clock from PLL3 SW Clock.
ccmRootmuxAclkEimSlowPll2Pfd2 Aclk EimSlow Clock from PLL2 PFD2.
ccmRootmuxAclkEimSlowPll3Pfd0 Aclk EimSlow Clock from PLL3 PFD0.

4.3.5.20 enum _ccm_rootmux_gpu_axi_sel

Enumerator

ccmRootmuxGpuAxiPll2Pfd2 GPU AXI Clock from PLL2 PFD2.
ccmRootmuxGpuAxiPll3Pfd0 GPU AXI Clock from PLL3 PFD0.
ccmRootmuxGpuAxiPll2Pfd1 GPU AXI Clock from PLL2 PFD1.
ccmRootmuxGpuAxiPll2 GPU AXI Clock from PLL2.

4.3.5.21 enum _ccm_rootmux_gpu_core_sel

Enumerator

ccmRootmuxGpuCorePll3Pfd1 GPU Core Clock from PLL3 PFD1.
ccmRootmuxGpuCorePll3Pfd0 GPU Core Clock from PLL3 PFD0.
ccmRootmuxGpuCorePll2 GPU Core Clock from PLL2.
ccmRootmuxGpuCorePll2Pfd2 GPU Core Clock from PLL2 PFD2.

4.3.5.22 enum _ccm_rootmux_vid_clk_sel

Enumerator

ccmRootmuxVidClkPll3Pfd1 VID Clock from PLL3 PFD1.
ccmRootmuxVidClkPll3 VID Clock from PLL3.
ccmRootmuxVidClkPll3Pfd3 VID Clock from PLL3 PFD3.
ccmRootmuxVidClkPll4 VID Clock from PLL4.
ccmRootmuxVidClkPll5 VID Clock from PLL5.

4.3.5.23 enum _ccm_rootmux_esai_clk_sel

Enumerator

ccmRootmuxEsaiClkPll4 ESAI Clock from PLL4.
ccmRootmuxEsaiClkPll3Pfd2 ESAI Clock from PLL3 PFD2.
ccmRootmuxEsaiClkPll5 ESAI Clock from PLL5.
ccmRootmuxEsaiClkPll3SwClk ESAI Clock from PLL3 SW Clock.

4.3.5.24 enum _ccm_rootmux_audio_clk_sel

Enumerator

ccmRootmuxAudioClkPll4 Audio Clock from PLL4.
ccmRootmuxAudioClkPll3Pfd2 Audio Clock from PLL3 PFD2.
ccmRootmuxAudioClkPll5 Audio Clock from PLL5.
ccmRootmuxAudioClkPll3SwClk Audio Clock from PLL3 SW Clock.

4.3.5.25 enum _ccm_rootmux_spdif0_clk_sel

Enumerator

ccmRootmuxSpdif0ClkPll4 SPDIF0 Clock from PLL4.
ccmRootmuxSpdif0ClkPll3Pfd2 SPDIF0 Clock from PLL3 PFD2.
ccmRootmuxSpdif0ClkPll5 SPDIF0 Clock from PLL5.
ccmRootmuxSpdif0ClkPll3SwClk SPDIF0 Clock from PLL3 SW Clock.

4.3.5.26 enum _ccm_rootmux_ssi1_clk_sel

Enumerator

ccmRootmuxSsi1ClkPll3Pfd2 SSI1 Clock from PLL3 PFD2.
ccmRootmuxSsi1ClkPll5 SSI1 Clock from PLL5.
ccmRootmuxSsi1ClkPll4 SSI1 Clock from PLL4.

4.3.5.27 enum _ccm_rootmux_ssi2_clk_sel

Enumerator

ccmRootmuxSsi2ClkPll3Pfd2 SSI2 Clock from PLL3 PFD2.
ccmRootmuxSsi2ClkPll5 SSI2 Clock from PLL5.
ccmRootmuxSsi2ClkPll4 SSI2 Clock from PLL4.

4.3.5.28 enum _ccm_rootmux_ssi3_clk_sel

Enumerator

ccmRootmuxSsi3ClkPll3Pfd2 SSI3 Clock from PLL3 PFD2.
ccmRootmuxSsi3ClkPll5 SSI3 Clock from PLL5.
ccmRootmuxSsi3ClkPll4 SSI3 Clock from PLL4.

4.3.5.29 enum _ccm_rootmux_lcdif2_clk_sel

Enumerator

ccmRootmuxLcdif2ClkLcdif2PreClk LCDIF2 Clock from LCDIF2 Pre Clock.
ccmRootmuxLcdif2ClkIppDi0Clk LCDIF2 Clock from IPP DI0 Clock.
ccmRootmuxLcdif2ClkIppDi1Clk LCDIF2 Clock from IPP DI0 Clock.
ccmRootmuxLcdif2ClkLdbDi0Clk LCDIF2 Clock from LDB DI0 Clock.
ccmRootmuxLcdif2ClkLdbDi1Clk LCDIF2 Clock from LDB DI0 Clock.

4.3.5.30 enum _ccm_rootmux_lcdif2_pre_clk_sel

Enumerator

ccmRootmuxLcdif2ClkPrePll2 LCDIF2 Pre Clock from PLL2.
ccmRootmuxLcdif2ClkPrePll3Pfd3 LCDIF2 Pre Clock from PLL3 PFD3.
ccmRootmuxLcdif2ClkPrePll5 LCDIF2 Pre Clock from PLL3 PFD5.
ccmRootmuxLcdif2ClkPrePll2Pfd0 LCDIF2 Pre Clock from PLL2 PFD0.
ccmRootmuxLcdif2ClkPrePll2Pfd3 LCDIF2 Pre Clock from PLL2 PFD3.
ccmRootmuxLcdif2ClkPrePll3Pfd1 LCDIF2 Pre Clock from PLL3 PFD1.

4.3.5.31 enum _ccm_rootmux_ldb_di1_clk_sel

Enumerator

ccmRootmuxLdbDi1ClkPll3SwClk lDB DI1 Clock from PLL3 SW Clock.
ccmRootmuxLdbDi1ClkPll2Pfd0 lDB DI1 Clock from PLL2 PFD0.

ccmRootmuxLdbDi1ClkPll2Pfd2 lDB DI1 Clock from PLL2 PFD2.
ccmRootmuxLdbDi1ClkPll2 lDB DI1 Clock from PLL2.
ccmRootmuxLdbDi1ClkPll3Pfd3 lDB DI1 Clock from PLL3 PFD3.
ccmRootmuxLdbDi1ClkPll3Pfd2 lDB DI1 Clock from PLL3 PFD2.

4.3.5.32 enum _ccm_rootmux_ldb_di0_clk_sel

Enumerator

ccmRootmuxLdbDi0ClkPll5 lDB DI0 Clock from PLL5.
ccmRootmuxLdbDi0ClkPll2Pfd0 lDB DI0 Clock from PLL2 PFD0.
ccmRootmuxLdbDi0ClkPll2Pfd2 lDB DI0 Clock from PLL2 PFD2.
ccmRootmuxLdbDi0ClkPll2Pfd3 lDB DI0 Clock from PLL2 PFD3.
ccmRootmuxLdbDi0ClkPll3Pfd1 lDB DI0 Clock from PLL3 PFD1.
ccmRootmuxLdbDi0ClkPll3Pfd3 lDB DI0 Clock from PLL3 PFD3.

4.3.5.33 enum _ccm_rootmux_lcdif1_clk_sel

Enumerator

ccmRootmuxLcdif1ClkLcdif1PreClk LCDIF1 clock from LCDIF1 Pre Clock.
ccmRootmuxLcdif1ClkIppDi0Clk LCDIF1 clock from IPP DI0 Clock.
ccmRootmuxLcdif1ClkIppDi1Clk LCDIF1 clock from IPP DI1 Clock.
ccmRootmuxLcdif1ClkLdbDi0Clk LCDIF1 clock from LDB DI0 Clock.
ccmRootmuxLcdif1ClkLdbDi1Clk LCDIF1 clock from LDB DI1 Clock.

4.3.5.34 enum _ccm_rootmux_lcdif1_pre_clk_sel

Enumerator

ccmRootmuxLcdif1PreClkPll2 LCDIF1 pre clock from PLL2.
ccmRootmuxLcdif1PreClkPll3Pfd3 LCDIF1 pre clock from PLL3 PFD3.
ccmRootmuxLcdif1PreClkPll5 LCDIF1 pre clock from PLL5.
ccmRootmuxLcdif1PreClkPll2Pfd0 LCDIF1 pre clock from PLL2 PFD0.
ccmRootmuxLcdif1PreClkPll2Pfd1 LCDIF1 pre clock from PLL2 PFD1.
ccmRootmuxLcdif1PreClkPll3Pfd1 LCDIF1 pre clock from PLL3 PFD1.

4.3.5.35 enum _ccm_rootmux_m4_clk_sel

Enumerator

ccmRootmuxM4ClkM4PreClk M4 clock from M4 Pre Clock.

ccmRootmuxM4ClkPll3Pfd3 M4 clock from PLL3 PFD3.
ccmRootmuxM4ClkIppDi0Clk M4 clock from IPP DI0 Clock.
ccmRootmuxM4ClkIppDi1Clk M4 clock from IPP DI1 Clock.
ccmRootmuxM4ClkLdbDi0Clk M4 clock from LDB DI0 Clock.
ccmRootmuxM4ClkLdbDi1Clk M4 clock from LDB DI1 Clock.

4.3.5.36 enum _ccm_rootmux_m4_pre_clk_sel

Enumerator

ccmRootmuxM4PreClkPll2 M4 pre clock from PLL2.
ccmRootmuxM4PreClkPll3SwClk M4 pre clock from PLL3 SW Clock.
ccmRootmuxM4PreClkOsc24m M4 pre clock from OSC 24M.
ccmRootmuxM4PreClkPll2Pfd0 M4 pre clock from PLL2 PFD0.
ccmRootmuxM4PreClkPll2Pfd2 M4 pre clock from PLL2 PFD2.
ccmRootmuxM4PreClkPll3Pfd3 M4 pre clock from PLL3 PFD3.

4.3.5.37 enum _ccm_rootmux_enet_clk_sel

Enumerator

ccmRootmuxEnetClkEnetPreClk Ethernet clock from Ethernet Pre Clock.
ccmRootmuxEnetClkIppDi0Clk Ethernet clock from IPP DI0 Clock.
ccmRootmuxEnetClkIppDi1Clk Ethernet clock from IPP DI1 Clock.
ccmRootmuxEnetClkLdbDi0Clk Ethernet clock from LDB DI0 Clock.
ccmRootmuxEnetClkLdbDi1Clk Ethernet clock from LDB DI1 Clock.

4.3.5.38 enum _ccm_rootmux_enet_pre_clk_sel

Enumerator

ccmRootmuxEnetPreClkPll2 Ethernet Pre clock from PLL2.
ccmRootmuxEnetPreClkPll3SwClk Ethernet Pre clock from PLL3 SW Clock.
ccmRootmuxEnetPreClkPll5 Ethernet Pre clock from PLL5.
ccmRootmuxEnetPreClkPll2Pfd0 Ethernet Pre clock from PLL2 PFD0.
ccmRootmuxEnetPreClkPll2Pfd2 Ethernet Pre clock from PLL2 PFD2.
ccmRootmuxEnetPreClkPll3Pfd2 Ethernet Pre clock from PLL3 PFD2.

4.3.5.39 enum _ccm_rootmux_qspi2_clk_sel

Enumerator

ccmRootmuxQspi2ClkPll2Pfd0 QSPI2 Clock from PLL2 PFD0.

ccmRootmuxQspi2ClkPll2 QSPI2 Clock from PLL2.
ccmRootmuxQspi2ClkPll3SwClk QSPI2 Clock from PLL3 SW Clock.
ccmRootmuxQspi2ClkPll2Pfd2 QSPI2 Clock from PLL2 PFD2.
ccmRootmuxQspi2ClkPll3Pfd3 QSPI2 Clock from PLL3 PFD3.

4.3.5.40 enum _ccm_rootmux_display_clk_sel

Enumerator

ccmRootmuxDisplayClkPll2 Display Clock from PLL2.
ccmRootmuxDisplayClkPll2Pfd2 Display Clock from PLL2 PFD2.
ccmRootmuxDisplayClkPll3SwClk Display Clock from PLL3 SW Clock.
ccmRootmuxDisplayClkPll3Pfd1 Display Clock from PLL3 PFD1.

4.3.5.41 enum _ccm_rootmux_csi_clk_sel

Enumerator

ccmRootmuxCsiClkOSC24m CSI Clock from OSC 24M.
ccmRootmuxCsiClkPll2Pfd2 CSI Clock from PLL2 PFD2.
ccmRootmuxCsiClkPll3SwClkDiv2 CSI Clock from PLL3 SW clock divided by 2.
ccmRootmuxCsiClkPll3Pfd1 CSI Clock from PLL3 PFD1.

4.3.5.42 enum _ccm_rootmux_can_clk_sel

Enumerator

ccmRootmuxCanClkPll3SwClkDiv8 CAN Clock from PLL3 SW clock divided by 8.
ccmRootmuxCanClkOsc24m CAN Clock from OSC 24M.
ccmRootmuxCanClkPll3SwClkDiv6 CAN Clock from PLL3 SW clock divided by 6.
ccmRootmuxCanClkDisableFlexcanClk Disable FlexCAN clock.

4.3.5.43 enum _ccm_rootmux_ecspi_clk_sel

Enumerator

ccmRootmuxEcspiClkPll3SwClkDiv8 ecSPI Clock from PLL3 SW clock divided by 8.
ccmRootmuxEcspiClkOsc24m ecSPI Clock from OSC 24M.

4.3.5.44 enum _ccm_rootmux_uart_clk_sel

Enumerator

ccmRootmuxUartClkPll3SwClkDiv6 UART Clock from PLL3 SW clock divided by 6.
ccmRootmuxUartClkOsc24m UART Clock from OSC 24M.

4.3.5.45 enum _ccm_root_div_control

These constants define the root control names for root divider setting.

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

Enumerator

ccmRootArmPodf ARM Clock post divider control names.
ccmRootFabricMmdcPodf Fabric MMDC Clock post divider control names.
ccmRootPeriph2Clk2Podf Peripheral2 Clock2 post divider control names.
ccmRootOcramPodf OCRAM Clock post divider control names.
ccmRootAhbPodf AHB Clock post divider control names.
ccmRootPeriphClk2Podf Peripheral Clock2 post divider control names.
ccmRootPerclkPodf Pre Clock post divider control names.
ccmRootIpgPodf IPG Clock post divider control names.
ccmRootUsdhc1Podf USDHC1 Clock post divider control names.
ccmRootUsdhc2Podf USDHC2 Clock post divider control names.
ccmRootUsdhc3Podf USDHC3 Clock post divider control names.
ccmRootUsdhc4Podf USDHC4 Clock post divider control names.
ccmRootAclkEimSlowPodf ACLK EIM SLOW Clock post divider control names.
ccmRootGpuAxiPodf GPU AXI Clock post divider control names.
ccmRootGpuCorePodf GPU Core Clock post divider control names.
ccmRootVidClkPodf VID Clock post divider control names.
ccmRootEsaiClkPodf ESAI Clock pre divider control names.
ccmRootEsaiClkPred ESAI Clock post divider control names.
ccmRootAudioClkPodf AUDIO Clock post divider control names.
ccmRootAudioClkPred AUDIO Clock pre divider control names.
ccmRootSpdif0ClkPodf SPDIF0 Clock post divider control names.
ccmRootSpdif0ClkPred SPDIF0 Clock pre divider control names.
ccmRootSsi1ClkPodf SSI1 Clock post divider control names.
ccmRootSsi1ClkPred SSI1 Clock pre divider control names.
ccmRootSsi2ClkPodf SSI2 Clock post divider control names.
ccmRootSsi2ClkPred SSI2 Clock pre divider control names.
ccmRootSsi3ClkPodf SSI3 Clock post divider control names.
ccmRootSsi3ClkPred SSI3 Clock pre divider control names.

ccmRootLcdif2Podf LCDIF2 Clock post divider control names.
ccmRootLcdif2Pred LCDIF2 Clock pre divider control names.
ccmRootLdbDi1Div LDB DI1 Clock divider control names.
ccmRootLdbDi0Div LCDIDI0 Clock divider control names.
ccmRootLcdif1Podf LCDIF1 Clock post divider control names.
ccmRootLcdif1Pred LCDIF1 Clock pre divider control names.
ccmRootM4Podf M4 Clock post divider control names.
ccmRootEnetPodf Ethernet Clock post divider control names.
ccmRootQspi1Podf QSPI1 Clock post divider control names.
ccmRootQspi2ClkPodf QSPI2 Clock post divider control names.
ccmRootQspi2ClkPred QSPI2 Clock pre divider control names.
ccmRootDisplayPodf Display Clock post divider control names.
ccmRootCsiPodf CSI Clock post divider control names.
ccmRootCanClkPodf CAN Clock post divider control names.
ccmRootEcspiClkPodf ECSPI Clock post divider control names.
ccmRootUartClkPodf UART Clock post divider control names.

4.3.5.46 enum _ccm_ccgr_gate

These constants define the ccm ccgr clock gate for each module.

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

Enumerator

ccmCcgrGateAipsTz1Clk AipsTz1 Clock Gate.
ccmCcgrGateAipsTz2Clk AipsTz2 Clock Gate.
ccmCcgrGateApbhdmaHclk ApbhdmaH Clock Gate.
ccmCcgrGateAsrcClk Asrc Clock Gate.
ccmCcgrGateCaamSecureMemClk CaamSecureMem Clock Gate.
ccmCcgrGateCaamWrapperAclk CaamWrapperA Clock Gate.
ccmCcgrGateCaamWrapperIpg CaamWrapperIpg Clock Gate.
ccmCcgrGateCan1Clk Can1 Clock Gate.
ccmCcgrGateCan1SerialClk Can1 Serial Clock Gate.
ccmCcgrGateCan2Clk Can2 Clock Gate.
ccmCcgrGateCan2SerialClk Can2 Serial Clock Gate.
ccmCcgrGateArmDbgClk Arm Debug Clock Gate.
ccmCcgrGateDcic1Clk Dcic1 Clock Gate.
ccmCcgrGateDcic2Clk Dcic2 Clock Gate.
ccmCcgrGateAipsTz3Clk AipsTz3 Clock Gate.
ccmCcgrGateEcspi1Clk Ecspi1 Clock Gate.
ccmCcgrGateEcspi2Clk Ecspi2 Clock Gate.
ccmCcgrGateEcspi3Clk Ecspi3 Clock Gate.

ccmCcgrGateEcspi4Clk Ecspi4 Clock Gate.
ccmCcgrGateEcspi5Clk Ecspi5 Clock Gate.
ccmCcgrGateEpit1Clk Epit1 Clock Gate.
ccmCcgrGateEpit2Clk Epit2 Clock Gate.
ccmCcgrGateEsaiClk Esai Clock Gate.
ccmCcgrGateWakeupClk Wakeup Clock Gate.
ccmCcgrGateGptClk Gpt Clock Gate.
ccmCcgrGateGptSerialClk Gpt Serial Clock Gate.
ccmCcgrGateGpuClk Gpu Clock Gate.
ccmCcgrGateOcramSClk OcramS Clock Gate.
ccmCcgrGateCanfdClk Canfd Clock Gate.
ccmCcgrGateCsiClk Csi Clock Gate.
ccmCcgrGateI2c1Serialclk I2c1 Serial Clock Gate.
ccmCcgrGateI2c2Serialclk I2c2 Serial Clock Gate.
ccmCcgrGateI2c3Serialclk I2c3 Serial Clock Gate.
ccmCcgrGateIimClk Iim Clock Gate.
ccmCcgrGateIomuxIptClkIo Iomux Ipt Clock Gate.
ccmCcgrGateIpmux1Clk Ipmux1 Clock Gate.
ccmCcgrGateIpmux2Clk Ipmux2 Clock Gate.
ccmCcgrGateIpmux3Clk Ipmux3 Clock Gate.
ccmCcgrGateIpsyncIp2apbtTasc1 IpsyncIp2apbtTasc1 Clock Gate.
ccmCcgrGateLcdClk Lcd Clock Gate.
ccmCcgrGatePxpClk Pxp Clock Gate.
ccmCcgrGateM4Clk M4 Clock Gate.
ccmCcgrGateEnetClk Enet Clock Gate.
ccmCcgrGateDispAxiClk DispAxi Clock Gate.
ccmCcgrGateLcdif2PixClk Lcdif2Pix Clock Gate.
ccmCcgrGateLcdif1PixClk Lcdif1Pix Clock Gate.
ccmCcgrGateLdbDi0Clk LdbDi0 Clock Gate.
ccmCcgrGateQspi1Clk Qspi1 Clock Gate.
ccmCcgrGateMlbClk Mlb Clock Gate.
ccmCcgrGateMmdcCoreAclkFastP0 Mmdc Core Aclk FastP0 Clock Gate.
ccmCcgrGateMmdcCoreIpgClkP0 Mmdc Core Ipg Clk P0 Clock Gate.
ccmCcgrGateMmdcCoreIpgClkP1 Mmdc Core Ipg Clk P1 Clock Gate.
ccmCcgrGateOcramClk Ocram Clock Gate.
ccmCcgrGatePcieRoot Pcie Clock Gate.
ccmCcgrGateQspi2Clk Qspi2 Clock Gate.
ccmCcgrGatePl301Mx6qper1Bch Pl301Mx6qper1Bch Clock Gate.
ccmCcgrGatePl301Mx6qper2Main Pl301Mx6qper2Main Clock Gate.
ccmCcgrGatePwm1Clk Pwm1 Clock Gate.
ccmCcgrGatePwm2Clk Pwm2 Clock Gate.
ccmCcgrGatePwm3Clk Pwm3 Clock Gate.
ccmCcgrGatePwm4Clk Pwm4 Clock Gate.
ccmCcgrGateRawnandUBchInptApb RawnandUBchInptApb Clock Gate.
ccmCcgrGateRawnandUGpmiBch RawnandUGpmiBch Clock Gate.

ccmCcgrGateRawnandUGpmiGpmiIo RawnandUGpmiGpmiIo Clock Gate.
ccmCcgrGateRawnandUGpmiInpApb RawnandUGpmiInpApb Clock Gate.
ccmCcgrGateRomClk Rom Clock Gate.
ccmCcgrGateSdmaClk Sdma Clock Gate.
ccmCcgrGateSpbaClk Spba Clock Gate.
ccmCcgrGateSpdifAudioClk SpdifAudio Clock Gate.
ccmCcgrGateSsi1Clk Ssi1 Clock Gate.
ccmCcgrGateSsi2Clk Ssi2 Clock Gate.
ccmCcgrGateSsi3Clk Ssi3 Clock Gate.
ccmCcgrGateUartClk Uart Clock Gate.
ccmCcgrGateUartSerialClk Uart Serial Clock Gate.
ccmCcgrGateSai1Clk Sai1 Clock Gate.
ccmCcgrGateSai2Clk Sai2 Clock Gate.
ccmCcgrGateUsboh3Clk Usboh3 Clock Gate.
ccmCcgrGateUsdhc1Clk Usdhc1 Clock Gate.
ccmCcgrGateUsdhc2Clk Usdhc2 Clock Gate.
ccmCcgrGateUsdhc3Clk Usdhc3 Clock Gate.
ccmCcgrGateUsdhc4Clk Usdhc4 Clock Gate.
ccmCcgrGateEimSlowClk EimSlow Clock Gate.
ccmCcgrGatePwm8Clk Pwm8 Clock Gate.
ccmCcgrGateVadcClk Vadc Clock Gate.
ccmCcgrGateGisClk Gis Clock Gate.
ccmCcgrGateI2c4SerialClk I2c4 Serial Clock Gate.
ccmCcgrGatePwm5Clk Pwm5 Clock Gate.
ccmCcgrGatePwm6Clk Pwm6 Clock Gate.
ccmCcgrGatePwm7Clk Pwm7 Clock Gate.

4.3.5.47 enum _ccm_gate_value

Enumerator

ccmClockNotNeeded Clock always disabled.
ccmClockNeededRun Clock enabled when CPU is running.
ccmClockNeededAll Clock always enabled.

4.3.5.48 enum _ccm_overrided_enable_signal

Enumerator

ccmOverridedSignalFromGpt Overide clock enable signal from GPT.
ccmOverridedSignalFromEpit Overide clock enable signal from EPIT.
ccmOverridedSignalFromUsdhc Overide clock enable signal from USDHC.
ccmOverridedSignalFromGpu Overide clock enable signal from GPU.

ccmOverridedSignalFromCan2Cpi Overide clock enable signal from CAN2.
ccmOverridedSignalFromCan1Cpi Overide clock enable signal from CAN1.

4.3.6 Function Documentation

4.3.6.1 static void CCM_SetRootMux (CCM_Type * *base*, uint32_t *ccmRootClk*, uint32_t *mux*) [inline], [static]

User maybe need to set more than one mux node according to the clock tree description on the reference manual.

Parameters

base	CCM base pointer.
ccmRootClk	Root clock control (see _ccm_root_clock_control enumeration).
mux	Root mux value (see _ccm_rootmux_xxx enumeration).

4.3.6.2 static uint32_t CCM_GetRootMux (CCM_Type * *base*, uint32_t *ccmRootClk*) [inline], [static]

In order to get the clock source of root, user maybe need to get more than one node's mux value to obtain the final clock source of root.

Parameters

base	CCM base pointer.
ccmRootClk	Root clock control (see _ccm_root_clock_control enumeration).

Returns

Root mux value (see _ccm_rootmux_xxx enumeration).

4.3.6.3 static void CCM_SetRootDivider (CCM_Type * *base*, uint32_t *ccmRootDiv*, uint32_t *div*) [inline], [static]

User should set the dividers carefully according to the clock tree on the reference manual. Take care of that the setting of one divider value may affect several clock root.

Parameters

base	CCM base pointer.
ccmRootDiv	Root divider control (see _ccm_root_div_control enumeration)
div	Divider value (divider = div + 1).

4.3.6.4 static uint32_t CCM_GetRootDivider (CCM_Type * *base*, uint32_t *ccmRootDiv*) [inline], [static]

In order to get divider value of clock root, user shouold get specific divider value according to the clock tree description on reference manual. Then calculate the root clock with those divider value.

Parameters

base	CCM base pointer.
ccmRootDiv	Root control (see _ccm_root_div_control enumeration).
div	Pointer to divider value store address.

Returns

Root divider value.

4.3.6.5 void CCM_SetMmdcHandshakeMask (CCM_Type * *base*, bool *mask*)

During divider ratio mmdc_axi_podf change or sync mux periph2_clk_sel change (but not jtag) or SRC request during warm reset, mask handshake with mmdc module.

Parameters

base	CCM base pointer.
mask	True: mask handshake with MMDC; False: allow handshake with MMDC.

4.3.6.6 static void CCM_ControlGate (CCM_Type * *base*, uint32_t *ccmGate*, uint32_t *control*) [inline], [static]

Take care of that one module may need to set more than one clock gate.

Parameters

base	CCM base pointer.
ccmGate	Gate control for each module (see _ccm_ccgr_gate enumeration).
control	Gate control value (see _ccm_gate_value).

4.3.6.7 void CCM_SetClockEnableSignalOverrided (CCM_Type * *base*, uint32_t *signal*, bool *control*)

This is applicable only for modules whose clock enable signals are used.

Parameters

base	CCM base pointer.
signal	Overrided enable signal from module (see _ccm_overrided_enable_signal enumeration).
control	Override / Do not override clock enable signal from module. • true: override clock enable signal. • false: Do not override clock enable signal.

Chapter 5
Enhanced Configurable Serial Peripheral Interface (eCSPI)

5.1 Overview

The FreeRTOS BSP provides a driver for the Enhanced Configurable Serial Peripheral Interface (eCSPI) of i.MX devices.

Modules

- ECSPI driver

5.2 ECSPI driver

5.2.1 Overview

This chapter describes the programming interface of the eCSPI driver (platform/drivers/inc/ecspi.h). The eCSPI driver provides a set of APIs to achieve these features:

- Data send and receive
- DMA management
- Interrupt management

5.2.2 SPI initialization

To initialize the eCSPI module, call the ECSPI_Init() function and pass the instance of eCSPI and an initialization structure. For example, to use the eCSPI1 module, pass the eCSPI1 base pointer and a pointer pointing to the ecspi_init_config_t structure.

Call the eCSPI Initialization function ECSPI_Init() functions for initialization and configuration. First, configure the ecspi_init_config_t structure as needed. Then, call ECSPI_Init() function to complete the initialization.

The following is an example of the eCSPI module initialization:

```
#define BOARD_ECSPI_MASTER_BASEADDR     ECSPI2
#define ECSPI_MASTER_BURSTLENGTH        (7)
#define BOARD_ECSPI_MASTER_CHANNEL      ecspiSelectChannel0
#define ECSPI_MASTER_STARTMODE          (0)

    // Configure the initialization structure.
    // Include clockRate, baudRate, mode, burstLength, channelSelect, clockPhase, clockPolarity,
       ecspiAutoStart
    // user can configure master and slave as needed.
    ecspi_init_config_t ecspiMasterInitConfig = {
        .clockRate = get_ecspi_clock_freq(BOARD_ECSPI_MASTER_BASEADDR),
        .baudRate = 500000,
        .mode = ecspiMasterMode,
        .burstLength = ECSPI_MASTER_BURSTLENGTH,
        .channelSelect = BOARD_ECSPI_MASTER_CHANNEL,
        .clockPhase = ecspiClockPhaseSecondEdge,
        .clockPolarity = ecspiClockPolarityActiveHigh,
        .ecspiAutoStart = ECSPI_MASTER_STARTMODE
    };

    // Initialize eCSPI and parameter configuration
    ECSPI_Init(BOARD_ECSPI_MASTER_BASEADDR, &ecspiMasterInitConfig);
```

In addition, for some parameters not included in ecspi_init_config_t structure, the driver provides specific APIs to configure them, such as these functions.

```
static inline void ECSPI_InsertWaitState(ECSPI_Type* base, uint32_t number);
void ECSPI_SetSampClockSource(ECSPI_Type* base, uint32_t source);
static inline void ECSPI_SetDelay(ECSPI_Type* base, uint32_t delay);
static inline void ECSPI_SetSCLKInactiveState(ECSPI_Type* base, uint32_t channel,
       uint32_t state);
static inline void ECSPI_SetDataInactiveState(ECSPI_Type* base, uint32_t channel,
```

```
       uint32_t state);
static inline void ECSPI_SetBurstLength(ECSPI_Type* base, uint32_t length);
static inline void ECSPI_SetSSMultipleBurst(ECSPI_Type* base, uint32_t channel,
      bool ssMultiBurst);
static inline void ECSPI_SetSSPolarity(ECSPI_Type* base, uint32_t channel, uint32_t
      polarity);
static inline void ECSPI_SetSPIDataReady(ECSPI_Type* base, uint32_t spidataready);
uint32_t ECSPI_SetBaudRate(ECSPI_Type* base, uint32_t sourceClockInHz, uint32_t bitsPerSec
      );
```

Those APIs provide settings for the wait state number, sample clock source, delay, eCSPI clock inactive state, data line inactive state, burst length, SS wave form, SS polarity, data ready signal, and baudRate.

5.2.3 eCSPI transfers

The driver supports APIs to implement the write data to register and receive data from the register. The real transfer data functions with blocking mode are provided to the user in demos and examples.

Send data and receive data function APIs:

```
static inline void ECSPI_SendData(ECSPI_Type* base, uint32_t data);
static inline uint32_t ECSPI_ReceiveData(ECSPI_Type* base);
```

To get the number of words in FIFO, use the following APIs:

```
static inline uint32_t ECSPI_GetRxfifoCounter(ECSPI_Type* base);
static inline uint32_t ECSPI_GetTxfifoCounter(ECSPI_Type* base);
```

5.2.4 DMA management

For the DMA operations, use the following APIs:

```
void ECSPPI_SetDMACmd(ECSPI_Type* base, uint32_t source, bool enable);
static inline void ECSPI_SetDMABurstLength(ECSPI_Type* base, uint32_t length);
```

5.2.5 eCSPI interrupt

Enable a specific eCSPI interrupt according to the ECSPI_SetIntCmd function. The following API functions are used to manage the interrupt and status flags:

```
void ECSPI_SetIntCmd(ECSPI_Type* base, uint32_t flags, bool enable);
static inline uint32_t ECSPI_GetStatusFlag(ECSPI_Type* base, uint32_t flags);
static inline void ECSPI_ClearStatusFlag(ECSPI_Type* base, uint32_t flags);
```

ECSPI_SetIntCmd() function can enable or disable specific eCSPI interrupts. ECSPI_GetStatusFlag() can check whether the specific eCSPI flag is set or not. ECSPI_ClearStatusFlag() can clear one or more eCSPI status flags.

Data Structures

- struct ecspi_init_config_t
  Init structure. More...

Enumerations

- enum _ecspi_channel_select {
  ecspiSelectChannel0 = 0U,
  ecspiSelectChannel1 = 1U,
  ecspiSelectChannel2 = 2U,
  ecspiSelectChannel3 = 3U }
  Channel select.
- enum _ecspi_master_slave_mode {
  ecspiSlaveMode = 0U,
  ecspiMasterMode = 1U }
  Channel mode.
- enum _ecspi_clock_phase {
  ecspiClockPhaseFirstEdge = 0U,
  ecspiClockPhaseSecondEdge = 1U }
  Clock phase.
- enum _ecspi_clock_polarity {
  ecspiClockPolarityActiveHigh = 0U,
  ecspiClockPolarityActiveLow = 1U }
  Clock polarity.
- enum _ecspi_ss_polarity {
  ecspiSSPolarityActiveLow = 0U,
  ecspiSSPolarityActiveHigh = 1U }
  SS signal polarity.
- enum _ecspi_dataline_inactivestate {
  ecspiDataLineStayHigh = 0U,
  ecspiDataLineStayLow = 1U }
  Inactive state of data line.
- enum _ecspi_sclk_inactivestate {
  ecspiSclkStayLow = 0U,
  ecspiSclkStayHigh = 1U }
  Inactive state of SCLK.
- enum _ecspi_sampleperiod_clocksource {
  ecspiSclk = 0U,
  ecspiLowFreq32K = 1U }
  sample period counter clock source.
- enum _ecspi_dma_source {
  ecspiDmaTxfifoEmpty = 7U,
  ecspiDmaRxfifoRequest = 23U,
  ecspiDmaRxfifoTail = 31U }
  DMA Source definition.

- enum _ecspi_fifothreshold {
  ecspiTxfifoThreshold = 0U,
  ecspiRxfifoThreshold = 16U }
  RXFIFO and TXFIFO threshold.
- enum _ecspi_status_flag {
  ecspiFlagTxfifoEmpty = 1U << 0,
  ecspiFlagTxfifoDataRequest = 1U << 1,
  ecspiFlagTxfifoFull = 1U << 2,
  ecspiFlagRxfifoReady = 1U << 3,
  ecspiFlagRxfifoDataRequest = 1U << 4,
  ecspiFlagRxfifoFull = 1U << 5,
  ecspiFlagRxfifoOverflow = 1U << 6,
  ecspiFlagTxfifoTc = 1U << 7 }
  Status flag.
- enum _ecspi_data_ready {
  ecspiRdyNoCare = 0U,
  ecspiRdyFallEdgeTrig = 1U,
  ecspiRdyLowLevelTrig = 2U,
  ecspiRdyReserved = 3U }
  Data Ready Control.

eCSPI Initialization and Configuration functions

- void ECSPI_Init (ECSPI_Type *base, const ecspi_init_config_t *initConfig)
  Initializes the eCSPI module.
- static void ECSPI_Enable (ECSPI_Type *base)
  Enables the specified eCSPI module.
- static void ECSPI_Disable (ECSPI_Type *base)
  Disable the specified eCSPI module.
- static void ECSPI_InsertWaitState (ECSPI_Type *base, uint32_t number)
  Insert the number of wait states to be inserted in data transfers.
- void ECSPI_SetSampClockSource (ECSPI_Type *base, uint32_t source)
  Set the clock source for the sample period counter.
- static void ECSPI_SetDelay (ECSPI_Type *base, uint32_t delay)
  Set the eCSPI clocks insert between the chip select active edge and the first eCSPI clock edge.
- static void ECSPI_SetSCLKInactiveState (ECSPI_Type *base, uint32_t channel, uint32_t state)
  Set the inactive state of SCLK.
- static void ECSPI_SetDataInactiveState (ECSPI_Type *base, uint32_t channel, uint32_t state)
  Set the inactive state of data line.
- static void ECSPI_StartBurst (ECSPI_Type *base)
  Trigger a burst.
- static void ECSPI_SetBurstLength (ECSPI_Type *base, uint32_t length)
  Set the burst length.
- static void ECSPI_SetSSMultipleBurst (ECSPI_Type *base, uint32_t channel, bool ssMultiBurst)
  Set eCSPI SS Wave Form.
- static void ECSPI_SetSSPolarity (ECSPI_Type *base, uint32_t channel, uint32_t polarity)
  Set eCSPI SS Polarity.
- static void ECSPI_SetSPIDataReady (ECSPI_Type *base, uint32_t spidataready)

Set the Data Ready Control.
- uint32_t ECSPI_SetBaudRate (ECSPI_Type *base, uint32_t sourceClockInHz, uint32_t bitsPerSec)
  Calculated the eCSPI baud rate in bits per second.

Data transfers functions

- static void ECSPI_SendData (ECSPI_Type *base, uint32_t data)
  Transmits a data to TXFIFO.
- static uint32_t ECSPI_ReceiveData (ECSPI_Type *base)
  Receives a data from RXFIFO.
- static uint32_t ECSPI_GetRxfifoCounter (ECSPI_Type *base)
  Read the number of words in the RXFIFO.
- static uint32_t ECSPI_GetTxfifoCounter (ECSPI_Type *base)
  Read the number of words in the TXFIFO.

DMA management functions

- void ECSPI_SetDMACmd (ECSPI_Type *base, uint32_t source, bool enable)
  Enable or disable the specified DMA Source.
- static void ECSPI_SetDMABurstLength (ECSPI_Type *base, uint32_t length)
  Set the burst length of a DMA operation.
- void ECSPI_SetFIFOThreshold (ECSPI_Type *base, uint32_t fifo, uint32_t threshold)
  Set the RXFIFO or TXFIFO threshold.

Interrupts and flags management functions

- void ECSPI_SetIntCmd (ECSPI_Type *base, uint32_t flags, bool enable)
  Enable or disable the specified eCSPI interrupts.
- static uint32_t ECSPI_GetStatusFlag (ECSPI_Type *base, uint32_t flags)
  Checks whether the specified eCSPI flag is set or not.
- static void ECSPI_ClearStatusFlag (ECSPI_Type *base, uint32_t flags)
  Clear one or more eCSPI status flag.

5.2.6 Data Structure Documentation

5.2.6.1 struct ecspi_init_config_t

Data Fields

- uint32_t clockRate
  Specifies ECSPII module clock freq.
- uint32_t baudRate
  Specifies desired eCSPI baud rate.
- uint32_t channelSelect
  Specifies the channel select.

- uint32_t mode
  Specifies the mode.
- uint32_t burstLength
  Specifies the length of a burst to be transferred.
- uint32_t clockPhase
  Specifies the clock phase.
- uint32_t clockPolarity
  Specifies the clock polarity.
- bool ecspiAutoStart
  Specifies the start mode.

5.2.6.1.0.2 Field Documentation

5.2.6.1.0.2.1 uint32_t ecspi_init_config_t::clockRate

5.2.6.1.0.2.2 uint32_t ecspi_init_config_t::baudRate

5.2.6.1.0.2.3 uint32_t ecspi_init_config_t::channelSelect

5.2.6.1.0.2.4 uint32_t ecspi_init_config_t::mode

5.2.6.1.0.2.5 uint32_t ecspi_init_config_t::burstLength

5.2.6.1.0.2.6 uint32_t ecspi_init_config_t::clockPhase

5.2.6.1.0.2.7 uint32_t ecspi_init_config_t::clockPolarity

5.2.6.1.0.2.8 bool ecspi_init_config_t::ecspiAutoStart

5.2.7 Enumeration Type Documentation

5.2.7.1 enum _ecspi_channel_select

Enumerator

ecspiSelectChannel0 Select Channel 0. Chip Select 0 (SS0) is asserted.
ecspiSelectChannel1 Select Channel 1. Chip Select 1 (SS1) is asserted.
ecspiSelectChannel2 Select Channel 2. Chip Select 2 (SS2) is asserted.
ecspiSelectChannel3 Select Channel 3. Chip Select 3 (SS3) is asserted.

5.2.7.2 enum _ecspi_master_slave_mode

Enumerator

ecspiSlaveMode Set Slave Mode.
ecspiMasterMode Set Master Mode.

5.2.7.3 enum _ecspi_clock_phase

Enumerator

ecspiClockPhaseFirstEdge Data is captured on the leading edge of the SCK and changed on the following edge.

ecspiClockPhaseSecondEdge Data is changed on the leading edge of the SCK and captured on the following edge.

5.2.7.4 enum _ecspi_clock_polarity

Enumerator

ecspiClockPolarityActiveHigh Active-high eCSPI clock (idles low).

ecspiClockPolarityActiveLow Active-low eCSPI clock (idles high).

5.2.7.5 enum _ecspi_ss_polarity

Enumerator

ecspiSSPolarityActiveLow Active-low, eCSPI SS signal.

ecspiSSPolarityActiveHigh Active-high, eCSPI SS signal.

5.2.7.6 enum _ecspi_dataline_inactivestate

Enumerator

ecspiDataLineStayHigh Data line inactive state stay high.

ecspiDataLineStayLow Data line inactive state stay low.

5.2.7.7 enum _ecspi_sclk_inactivestate

Enumerator

ecspiSclkStayLow SCLK inactive state stay low.

ecspiSclkStayHigh SCLK line inactive state stay high.

5.2.7.8 enum _ecspi_sampleperiod_clocksource

Enumerator

ecspiSclk Sample period counter clock from SCLK.

ecspiLowFreq32K Sample period counter clock from from LFRC (32.768 KHz).

5.2.7.9 enum _ecspi_dma_source

Enumerator

ecspiDmaTxfifoEmpty TXFIFO Empty DMA Request.
ecspiDmaRxfifoRequest RXFIFO DMA Request.
ecspiDmaRxfifoTail RXFIFO TAIL DMA Request.

5.2.7.10 enum _ecspi_fifothreshold

Enumerator

ecspiTxfifoThreshold Defines the FIFO threshold that triggers a TX DMA/INT request.
ecspiRxfifoThreshold defines the FIFO threshold that triggers a RX DMA/INT request.

5.2.7.11 enum _ecspi_status_flag

Enumerator

ecspiFlagTxfifoEmpty TXFIFO Empty Flag.
ecspiFlagTxfifoDataRequest TXFIFO Data Request Flag.
ecspiFlagTxfifoFull TXFIFO Full Flag.
ecspiFlagRxfifoReady RXFIFO Ready Flag.
ecspiFlagRxfifoDataRequest RXFIFO Data Request Flag.
ecspiFlagRxfifoFull RXFIFO Full Flag.
ecspiFlagRxfifoOverflow RXFIFO Overflow Flag.
ecspiFlagTxfifoTc TXFIFO Transform Completed Flag.

5.2.7.12 enum _ecspi_data_ready

Enumerator

ecspiRdyNoCare The SPI_RDY signal is ignored.
ecspiRdyFallEdgeTrig Burst is triggered by the falling edge of the SPI_RDY signal (edge-triggered).

ecspiRdyLowLevelTrig Burst is triggered by a low level of the SPI_RDY signal (level-triggered).
ecspiRdyReserved Reserved.

5.2.8 Function Documentation

5.2.8.1 void ECSPI_Init (ECSPI_Type * *base*, const ecspi_init_config_t * *initConfig*)

Parameters

base	eCSPI base pointer.
initConfig	eCSPI initialization structure.

5.2.8.2 static void ECSPI_Enable (ECSPI_Type * *base*) [inline], [static]

Parameters

base	eCSPI base pointer.

5.2.8.3 static void ECSPI_Disable (ECSPI_Type * *base*) [inline], [static]

Parameters

base	eCSPI base pointer.

5.2.8.4 static void ECSPI_InsertWaitState (ECSPI_Type * *base,* uint32_t *number*) [inline], [static]

Parameters

base	eCSPI base pointer.
number	the number of wait states.

5.2.8.5 void ECSPI_SetSampClockSource (ECSPI_Type * *base,* uint32_t *source*)

Parameters

base	eCSPI base pointer.
source	The clock source (see _ecspi_sampleperiod_clocksource enumeration).

5.2.8.6 static void ECSPI_SetDelay (ECSPI_Type * *base,* uint32_t *delay*) [inline], [static]

Parameters

base	eCSPI base pointer.
delay	The number of wait states.

5.2.8.7 static void ECSPI_SetSCLKInactiveState (ECSPI_Type * *base*, uint32_t *channel*, uint32_t *state*) [inline], [static]

Parameters

base	eCSPI base pointer.
channel	eCSPI channel select (see _ecspi_channel_select enumeration).
state	SCLK inactive state (see _ecspi_sclk_inactivestate enumeration).

5.2.8.8 static void ECSPI_SetDataInactiveState (ECSPI_Type * *base*, uint32_t *channel*, uint32_t *state*) [inline], [static]

Parameters

base	eCSPI base pointer.
channel	eCSPI channel select (see _ecspi_channel_select enumeration).
state	Data line inactive state (see _ecspi_dataline_inactivestate enumeration).

5.2.8.9 static void ECSPI_StartBurst (ECSPI_Type * *base*) [inline], [static]

Parameters

base	eCSPI base pointer.

5.2.8.10 static void ECSPI_SetBurstLength (ECSPI_Type * *base*, uint32_t *length*) [inline], [static]

Parameters

base	eCSPI base pointer.
length	The value of burst length.

5.2.8.11 static void ECSPI_SetSSMultipleBurst (ECSPI_Type * *base*, uint32_t *channel*, bool *ssMultiBurst*) [inline], [static]

Parameters

base	eCSPI base pointer.
channel	eCSPI channel selected (see _ecspi_channel_select enumeration).
ssMultiBurst	For master mode, set true for multiple burst and false for one burst. For slave mode, set true to complete burst by SS signal edges and false to complete burst by number of bits received.

5.2.8.12 static void ECSPI_SetSSPolarity (ECSPI_Type * *base*, uint32_t *channel*, uint32_t *polarity*) [inline], [static]

Parameters

base	eCSPI base pointer.
channel	eCSPI channel selected (see _ecspi_channel_select enumeration).
polarity	Set SS signal active logic (see _ecspi_ss_polarity enumeration).

5.2.8.13 static void ECSPI_SetSPIDataReady (ECSPI_Type * *base*, uint32_t *spidataready*) [inline], [static]

Parameters

base	eCSPI base pointer.
spidataready	eCSPI data ready control (see _ecspi_data_ready enumeration).

5.2.8.14 uint32_t ECSPI_SetBaudRate (ECSPI_Type * *base*, uint32_t *sourceClockInHz*, uint32_t *bitsPerSec*)

```
The calculated baud rate must not exceed the desired baud rate.
```

Parameters

base	eCSPI base pointer.
sourceClockIn-Hz	eCSPI Clock(SCLK) (in Hz).
bitsPerSec	the value of Baud Rate.

Returns

The calculated baud rate in bits-per-second, the nearest possible baud rate without exceeding the desired baud rate.

5.2.8.15 static void ECSPI_SendData (ECSPI_Type * *base,* uint32_t *data*) [inline], [static]

Parameters

base	eCSPI base pointer.
data	Data to be transmitted.

5.2.8.16 static uint32_t ECSPI_ReceiveData (ECSPI_Type * *base*) [inline], [static]

Parameters

base	eCSPI base pointer.

Returns

The value of received data.

5.2.8.17 static uint32_t ECSPI_GetRxfifoCounter (ECSPI_Type * *base*) [inline], [static]

Parameters

base	eCSPI base pointer.

Returns

The number of words in the RXFIFO.

5.2.8.18 static uint32_t ECSPI_GetTxfifoCounter (ECSPI_Type * *base*) [inline], [static]

Parameters

base	eCSPI base pointer.

Returns

The number of words in the TXFIFO.

5.2.8.19 void ECSPI_SetDMACmd (ECSPI_Type * *base,* uint32_t *source,* bool *enable*)

Parameters

base	eCSPI base pointer.
source	specifies DMA source (see _ecspi_dma_source enumeration).
enable	Enable/Disable specified DMA Source. • true: Enable specified DMA Source. • false: Disable specified DMA Source.

5.2.8.20 static void ECSPI_SetDMABurstLength (ECSPI_Type * *base,* uint32_t *length*) [inline], [static]

Parameters

base	eCSPI base pointer.
length	Specifies the burst length of a DMA operation.

5.2.8.21 void ECSPI_SetFIFOThreshold (ECSPI_Type * *base*, uint32_t *fifo*, uint32_t *threshold*)

Parameters

base	eCSPI base pointer.
fifo	Data transfer FIFO (see _ecspi_fifothreshold enumeration).
threshold	Threshold value.

5.2.8.22 void ECSPI_SetIntCmd (ECSPI_Type * *base*, uint32_t *flags*, bool *enable*)

Parameters

base	eCSPI base pointer.
flags	eCSPI status flag mask (see _ecspi_status_flag for bit definition).
enable	Interrupt enable. • true: Enable specified eCSPI interrupts. • false: Disable specified eCSPI interrupts.

5.2.8.23 static uint32_t ECSPI_GetStatusFlag (ECSPI_Type * *base*, uint32_t *flags*) [inline], [static]

Parameters

base	eCSPI base pointer.
flags	eCSPI status flag mask (see _ecspi_status_flag for bit definition).

Returns

eCSPI status, each bit represents one status flag.

5.2.8.24 static void ECSPI_ClearStatusFlag (ECSPI_Type * *base*, uint32_t *flags*) [inline], [static]

Parameters

base	eCSPI base pointer.
flags	eCSPI status flag mask (see _ecspi_status_flag for bit definition).

Chapter 6
Enhanced Periodic Interrupt Timer (EPIT)

6.1 Overview

The FreeRTOS BSP provides a driver for the Enhanced Periodic Interrupt Timer (EPIT) block of i.MX devices.

Modules

- EPIT driver

6.2 EPIT driver

6.2.1 Overview

The chapter describes the programming interface of the EPIT driver (platform/drivers/inc/epit.h). The EPIT has a 32-bit down-counter and can generate an interrupt and an event on the output pin when the timer reaches a programmed value. The EPIT driver provides a set of APIs to provide these services:

- EPIT general setting;
- EPIT output signal control;
- EPIT data load control;
- EPIT interrupt control;

6.2.2 EPIT general setting

Before any other function is called, EPIT_Init() must be invoked. EPIT_Init() initializes the module to reset state and configure the EPIT behavior in different CPU modes.

To keep the EPIT clock source, mode setting and reset all other configurations, EPIT_SoftReset() is used. And after the function return, the reset operation is finished.

EPIT counter has several source to select, including OSC(24M), low reference clock(32K), or peripheral clock(EPIT module clock). Use EPIT_SetClockSource() to set clock source for the counter. EPIT_GetClockSource() can help getting current counter clock source setting.

EPIT also provide divider to make the counter clock source fit into appropriate frequency range. The user can use EPIT_SetPrescaler() to set the divider, or EPIT_GetPrescaler() to get current divider setting.

When above EPIT setting is done, EPIT_Enable() can be used to start the counter, and then EPIT_Disable() to stop the counter. To get current counter value, EPIT_ReadCounter() can be used.

6.2.3 EPIT output signal control

Each EPIT instance has 1 output signal and EPIT_SetOutputCompareValue() can be used to set the compare value. Once the counter reaches the compare value, an event is triggered and some kind of operation on external pin occurs. The user can use EPIT_SetOutputOperationMode() to set the operation when the event happen:

1. Nothing
2. Toggle the value
3. Set to low (clear)
4. Set to high (set)
   Similarly, EPIT_GetOutputOperationMode() and EPIT_GetOutputCompareValue() can be used to get current setting for certain channel.

6.2.4 EPIT data load control

EPIT has a down-counter and some initial value need to be set before start counting. Use EPIT_SetCounterLoadValue() to set the load value as the counter initial value and EPIT_GetCounterLoadValue() to get the load value. When counter reaches zero, the load value will be reloaded an continue counting down.

There's a special case that the load value can be overwritten to the counter immediately, when using EPIT_SetOverwriteCounter() to enable the overwrite mode. If the behavior is not intended, use EPIT_SetOverwriteCounter() to disable the overwrite mode.

6.2.5 EPIT interrupt control

EPIT module provide only one interrupt event: output compare event. The user can use EPIT_SetIntCmd() to enable or disable this interrupt, and use EPIT_GetStatusFlag() to get current event status. Once some event occurs, EPIT_ClearStatusFlag() can be used to clear the event.

Data Structures

- struct epit_init_config_t
  Structure to configure the running mode. More...

Enumerations

- enum _epit_clock_source {
  epitClockSourceOff = 0U,
  epitClockSourcePeriph = 1U,
  epitClockSourceHighFreq = 2U,
  epitClockSourceLowFreq = 3U }
  Clock source.
- enum _epit_output_operation_mode {
  epitOutputOperationDisconnected = 0U,
  epitOutputOperationToggle = 1U,
  epitOutputOperationClear = 2U,
  epitOutputOperationSet = 3U }
  Output compare operation mode.

EPIT State Control

- void EPIT_Init (EPIT_Type *base, const epit_init_config_t *initConfig)
  Initialize EPIT to reset state and initialize running mode.
- static void EPIT_SoftReset (EPIT_Type *base)
  Software reset of EPIT module.
- static void EPIT_SetClockSource (EPIT_Type *base, uint32_t source)

Set clock source of EPIT.

- static uint32_t EPIT_GetClockSource (EPIT_Type *base)
  Get clock source of EPIT.
- static void EPIT_SetPrescaler (EPIT_Type *base, uint32_t prescaler)
  Set pre scaler of EPIT.
- static uint32_t EPIT_GetPrescaler (EPIT_Type *base)
  Get pre scaler of EPIT.
- static void EPIT_Enable (EPIT_Type *base)
  Enable EPIT module.
- static void EPIT_Disable (EPIT_Type *base)
  Disable EPIT module.
- static uint32_t EPIT_ReadCounter (EPIT_Type *base)
  Get EPIT counter value.

EPIT Output Signal Control

- static void EPIT_SetOutputOperationMode (EPIT_Type *base, uint32_t mode)
  Set EPIT output compare operation mode.
- static uint32_t EPIT_GetOutputOperationMode (EPIT_Type *base)
  Get EPIT output compare operation mode.
- static void EPIT_SetOutputCompareValue (EPIT_Type *base, uint32_t value)
  Set EPIT output compare value.
- static uint32_t EPIT_GetOutputCompareValue (EPIT_Type *base)
  Get EPIT output compare value.

EPIT Data Load Control

- static void EPIT_SetCounterLoadValue (EPIT_Type *base, uint32_t value)
  Set the value that is to be loaded into counter register.
- static uint32_t EPIT_GetCounterLoadValue (EPIT_Type *base)
  Get the value that loaded into counter register.
- void EPIT_SetOverwriteCounter (EPIT_Type *base, bool enable)
  Enable or disable EPIT overwrite counter immediately.

EPIT Interrupt and Status Control

- static uint32_t EPIT_GetStatusFlag (EPIT_Type *base)
  Get EPIT status of output compare interrupt flag.
- static void EPIT_ClearStatusFlag (EPIT_Type *base)
  Clear EPIT Output compare interrupt flag.
- void EPIT_SetIntCmd (EPIT_Type *base, bool enable)
  Enable or disable EPIT interrupt.

6.2.6 Data Structure Documentation

6.2.6.1 struct epit_init_config_t

Data Fields

- bool freeRun
  true: set-and-forget mode, false: free-running mode.
- bool waitEnable
  EPIT enabled in wait mode.
- bool stopEnable
  EPIT enabled in stop mode.
- bool dbgEnable
  EPIT enabled in debug mode.
- bool enableMode
  true: counter starts counting from load value (RLD=1) or 0xFFFF_FFFF (If RLD=0) when enabled, false: counter restores the value that it was disabled when enabled.

6.2.6.1.0.3 Field Documentation

6.2.6.1.0.3.1 bool epit_init_config_t::freeRun

6.2.6.1.0.3.2 bool epit_init_config_t::waitEnable

6.2.6.1.0.3.3 bool epit_init_config_t::stopEnable

6.2.6.1.0.3.4 bool epit_init_config_t::dbgEnable

6.2.6.1.0.3.5 bool epit_init_config_t::enableMode

6.2.7 Enumeration Type Documentation

6.2.7.1 enum _epit_clock_source

Enumerator

epitClockSourceOff EPIT Clock Source Off.
epitClockSourcePeriph EPIT Clock Source from Peripheral Clock.
epitClockSourceHighFreq EPIT Clock Source from High Frequency Reference Clock.
epitClockSourceLowFreq EPIT Clock Source from Low Frequency Reference Clock.

6.2.7.2 enum _epit_output_operation_mode

Enumerator

epitOutputOperationDisconnected EPIT Output Operation: Disconnected from pad.
epitOutputOperationToggle EPIT Output Operation: Toggle output pin.
epitOutputOperationClear EPIT Output Operation: Clear output pin.

epitOutputOperationSet EPIT Output Operation: Set putput pin.

6.2.8 Function Documentation

6.2.8.1 void EPIT_Init (EPIT_Type * *base*, const epit_init_config_t * *initConfig*)

Parameters

base	EPIT base pointer.
initConfig	EPIT initialize structure.

6.2.8.2 static void EPIT_SoftReset (EPIT_Type * *base*) [inline], [static]

Parameters

base	EPIT base pointer.

6.2.8.3 static void EPIT_SetClockSource (EPIT_Type * *base*, uint32_t *source*) [inline], [static]

Parameters

base	EPIT base pointer.
source	clock source (see _epit_clock_source enumeration).

6.2.8.4 static uint32_t EPIT_GetClockSource (EPIT_Type * *base*) [inline], [static]

Parameters

base	EPIT base pointer.

Returns

clock source (see _epit_clock_source enumeration).

6.2.8.5 static void EPIT_SetPrescaler (EPIT_Type * *base*, uint32_t *prescaler*) [inline], [static]

Parameters

base	EPIT base pointer.
prescaler	Pre-scaler of EPIT (0-4095, divider = prescaler + 1).

6.2.8.6 static uint32_t EPIT_GetPrescaler (EPIT_Type * *base*) [inline], [static]

Parameters

base	EPIT base pointer.

Returns

Pre-scaler of EPIT (0-4095).

6.2.8.7 static void EPIT_Enable (EPIT_Type * *base*) [inline], [static]

Parameters

base	EPIT base pointer.

6.2.8.8 static void EPIT_Disable (EPIT_Type * *base*) [inline], [static]

Parameters

base	EPIT base pointer.

6.2.8.9 static uint32_t EPIT_ReadCounter (EPIT_Type * *base*) [inline], [static]

Parameters

base	EPIT base pointer.

Returns

EPIT counter value.

6.2.8.10 static void EPIT_SetOutputOperationMode (EPIT_Type * *base,* uint32_t *mode*) [inline], [static]

Parameters

base	EPIT base pointer.
mode	EPIT output compare operation mode (see _epit_output_operation_mode enumeration).

6.2.8.11 static uint32_t EPIT_GetOutputOperationMode (EPIT_Type * *base*) [inline], [static]

Parameters

base	EPIT base pointer.

Returns

EPIT output operation mode (see _epit_output_operation_mode enumeration).

6.2.8.12 static void EPIT_SetOutputCompareValue (EPIT_Type * *base,* uint32_t *value*) [inline], [static]

Parameters

base	EPIT base pointer.
value	EPIT output compare value.

6.2.8.13 static uint32_t EPIT_GetOutputCompareValue (EPIT_Type * *base*) [inline], [static]

Parameters

base	EPIT base pointer.

Returns

EPIT output compare value.

6.2.8.14 static void EPIT_SetCounterLoadValue (EPIT_Type * *base,* uint32_t *value*) [inline], [static]

Parameters

base	EPIT base pointer.
value	Counter load value.

6.2.8.15 static uint32_t EPIT_GetCounterLoadValue (EPIT_Type * *base*) [inline], [static]

Parameters

base	EPIT base pointer.

Returns

The counter load value.

6.2.8.16 void EPIT_SetOverwriteCounter (EPIT_Type * *base,* bool *enable*)

Parameters

base	EPIT base pointer.
enable	Enable/Disable EPIT overwrite counter immediately. • true: Enable overwrite counter immediately. • false: Disable overwrite counter immediately.

6.2.8.17 static uint32_t EPIT_GetStatusFlag (EPIT_Type * *base*) [inline], [static]

Parameters

base	EPIT base pointer.

Returns

EPIT status of output compare interrupt flag.

6.2.8.18 static void EPIT_ClearStatusFlag (EPIT_Type * *base*) [inline], [static]

Parameters

base	EPIT base pointer.

6.2.8.19 void EPIT_SetIntCmd (EPIT_Type * *base*, bool *enable*)

Parameters

base	EPIT base pointer.
enable	Enable/Disable EPIT interrupt. • true: Enable interrupt. • false: Disable interrupt.

Chapter 7
Flex Controller Area Network (FlexCAN)

7.1 Overview

The FreeRTOS BSP provides a driver for the Flex Controller Area Network (FlexCAN) block of i.MX devices.

Modules

- FlexCAN driver

7.2 FlexCAN driver

7.2.1 Overview

The section describes the programming interface of the FlexCAN driver(platform/drivers/inc/flexcan.h).

7.2.2 FlexCAN initialization

To initialize the FlexCAN module, define a flexcan_init_config_t type variable and pass it to the FLEXCAN_Init() function. These are the members of the structure definition:

1. timing: The timing characteristic of CAN Bus communication defined in CAN 2.0B spec;
2. operatingMode: The operating mode of FlexCAN module with 3 modes defined in enum _flexcan_operating_modes;
3. maxMsgBufNum: The maximum number of message buffers used for CAN communication; the unused message buffer area can be used as a normal SRAM.

The user should also set the Rx Mask Mode using the FLEXCAN_SetRxMaskMode() and the global/individual mask using FLEXCAN_SetRxGlobalMask() / FLEXCAN_SetRxIndividualMask() functions. After that, the user can send/receive messages though the FlexCAN message buffers.

7.2.3 FlexCAN Data Transactions

The FlexCAN driver provides API to acquire the Message Buffer(MB) for data transfers. All data transfers are controlled by setting the MB internal fields.

FlexCAN data send

To send data through the UART port, follow these steps:

1. Acquire the message buffer for data sending by calling the FLEXCAN_GetMsgBufPtr() function.
2. Fill the local priority, identifier, data length, remote frame type, identifier format, and substitute the remote request according to the application requirements.
3. Load data to the message buffer data field and write flexcanTxDataOrRemte to the message buffer code field to start the transition.
4. Call the FLEXCAN_GetMsgBufStatusFlag() function to see if the transition is finished.
5. Repeat the above process to send more data to the CAN bus.

FlexCAN data receive

To receive data through the UART bus, follow these steps:

1. Acquire the message buffer for data receiving by calling the FLEXCAN_GetMsgBufPtr() function.
2. Fill the message buffer with the identifier of the message you want to receive.

3. Write the flexcanRxEmpty to the message buffer code field to start the transition.
4. Call the FLEXCAN_GetMsgBufStatusFlag() function to see whether the transition is finished.
5. Call the FLEXCAN_LockRxMsgBuf() before copying the received data from Rx MB and call the FLEXCAN_UnlockAllRxMsgBuf() function to unlock all MB to guarantee the data consistency.
6. Repeat step 4 and 5 to read more data from the CAN bus.

FlexCAN status and interrupt

This driver provides APIs to handle the FlexCAN module error status and interrupt:

```
FLEXCAN_SetErrIntCmd()
FLEXCAN_GetErrStatusFlag()
FLEXCAN_ClearErrStatusFlag()
```

This driver provides APIs to handle the FlexCAN Message Buffer status and interrupt:

```
FLEXCAN_SetMsgBufIntCmd()
FLEXCAN_GetMsgBufStatusFlag()
FLEXCAN_ClearMsgBufStatusFlag()
```

Specific FlexCAN functions

In addition to the functions mentioned above, the FlexCAN driver also provides a set of functions for a specialized purpose, such as the Rx FIFO and FIFO mask control and functions for optimizing the communication system reliability. See the *i.MX 6SoloX Applications Processor Reference Manual* (IMX6SXRM) and function descriptions below for more information about these functions.

Example

For more information about how to use this driver, see the FlexCAN demo/example under examples/<board_name>/.

Data Structures

- struct flexcan_id_table_t
  FlexCAN RX FIFO ID filter table structure. *More...*
- struct flexcan_msgbuf_t
  FlexCAN message buffer structure. *More...*
- struct flexcan_timing_t
  FlexCAN timing-related structures. *More...*
- struct flexcan_init_config_t
  FlexCAN module initialization structure. *More...*

Enumerations

- enum _flexcan_msgbuf_code_rx {
  flexcanRxInactive = 0x0,
  flexcanRxFull = 0x2,
  flexcanRxEmpty = 0x4,
  flexcanRxOverrun = 0x6,
  flexcanRxBusy = 0x8,
  flexcanRxRanswer = 0xA,
  flexcanRxNotUsed = 0xF }
  FlexCAN message buffer CODE for Rx buffers.
- enum _flexcan_msgbuf_code_tx {
  flexcanTxInactive = 0x8,
  flexcanTxAbort = 0x9,
  flexcanTxDataOrRemte = 0xC,
  flexcanTxTanswer = 0xE,
  flexcanTxNotUsed = 0xF }
  FlexCAN message buffer CODE FOR Tx buffers.
- enum _flexcan_operatining_modes {
  flexcanNormalMode = 0x1,
  flexcanListenOnlyMode = 0x2,
  flexcanLoopBackMode = 0x4 }
  FlexCAN operation modes.
- enum _flexcan_rx_mask_mode {
  flexcanRxMaskGlobal = 0x0,
  flexcanRxMaskIndividual = 0x1 }
  FlexCAN RX mask mode.
- enum _flexcan_rx_mask_id_type {
  flexcanRxMaskIdStd = 0x0,
  flexcanRxMaskIdExt = 0x1 }
  The ID type used in rx matching process.
- enum _flexcan_interrutpt {
  flexcanIntRxWarning = 0x01,
  flexcanIntTxWarning = 0x02,
  flexcanIntWakeUp = 0x04,
  flexcanIntBusOff = 0x08,
  flexcanIntError = 0x10 }
  FlexCAN error interrupt source enumeration.
- enum _flexcan_status_flag {

flexcanStatusSynch = CAN_ESR1_SYNCH_MASK,
flexcanStatusTxWarningInt = CAN_ESR1_TWRN_INT_MASK,
flexcanStatusRxWarningInt = CAN_ESR1_RWRN_INT_MASK,
flexcanStatusBit1Err = CAN_ESR1_BIT1_ERR_MASK,
flexcanStatusBit0Err = CAN_ESR1_BIT0_ERR_MASK,
flexcanStatusAckErr = CAN_ESR1_ACK_ERR_MASK,
flexcanStatusCrcErr = CAN_ESR1_CRC_ERR_MASK,
flexcanStatusFrameErr = CAN_ESR1_FRM_ERR_MASK,
flexcanStatusStuffingErr = CAN_ESR1_STF_ERR_MASK,
flexcanStatusTxWarning = CAN_ESR1_TX_WRN_MASK,
flexcanStatusRxWarning = CAN_ESR1_RX_WRN_MASK,
flexcanStatusIdle = CAN_ESR1_IDLE_MASK,
flexcanStatusTransmitting = CAN_ESR1_TX_MASK,
flexcanStatusFltConf = CAN_ESR1_FLT_CONF_MASK,
flexcanStatusReceiving = CAN_ESR1_RX_MASK,
flexcanStatusBusOff = CAN_ESR1_BOFF_INT_MASK,
flexcanStatusError = CAN_ESR1_ERR_INT_MASK,
flexcanStatusWake = CAN_ESR1_WAK_INT_MASK }

FlexCAN error interrupt flags.

- enum _flexcan_rx_fifo_id_element_format {
  flexcanRxFifoIdElementFormatA = 0x0,
  flexcanRxFifoIdElementFormatB = 0x1,
  flexcanRxFifoIdElementFormatC = 0x2,
  flexcanRxFifoIdElementFormatD = 0x3 }

  The id filter element type selection.
- enum _flexcan_rx_fifo_filter_id_number {
  flexcanRxFifoIdFilterNum8 = 0x0,
  flexcanRxFifoIdFilterNum16 = 0x1,
  flexcanRxFifoIdFilterNum24 = 0x2,
  flexcanRxFifoIdFilterNum32 = 0x3,
  flexcanRxFifoIdFilterNum40 = 0x4,
  flexcanRxFifoIdFilterNum48 = 0x5,
  flexcanRxFifoIdFilterNum56 = 0x6,
  flexcanRxFifoIdFilterNum64 = 0x7,
  flexcanRxFifoIdFilterNum72 = 0x8,
  flexcanRxFifoIdFilterNum80 = 0x9,
  flexcanRxFifoIdFilterNum88 = 0xA,
  flexcanRxFifoIdFilterNum96 = 0xB,
  flexcanRxFifoIdFilterNum104 = 0xC,
  flexcanRxFifoIdFilterNum112 = 0xD,
  flexcanRxFifoIdFilterNum120 = 0xE,
  flexcanRxFifoIdFilterNum128 = 0xF }

  FlexCAN Rx FIFO filters number.

FlexCAN Initialization and Configuration functions

- void FLEXCAN_Init (CAN_Type *base, const flexcan_init_config_t *initConfig)
  Initialize FlexCAN module with given initialization structure.
- void FLEXCAN_Deinit (CAN_Type *base)
  This function reset FlexCAN module register content to its default value.
- void FLEXCAN_Enable (CAN_Type *base)
  This function is used to Enable the FlexCAN Module.
- void FLEXCAN_Disable (CAN_Type *base)
  This function is used to Disable the FlexCAN Module.
- void FLEXCAN_SetTiming (CAN_Type *base, const flexcan_timing_t *timing)
  Sets the FlexCAN time segments for setting up bit rate.
- void FLEXCAN_SetOperatingMode (CAN_Type *base, uint8_t mode)
  Set operation mode.
- void FLEXCAN_SetMaxMsgBufNum (CAN_Type *base, uint32_t bufNum)
  Set the maximum number of Message Buffers.
- static bool FLEXCAN_IsModuleReady (CAN_Type *base)
  Get the working status of FlexCAN module.
- void FLEXCAN_SetAbortCmd (CAN_Type *base, bool enable)
  Set the Transmit Abort feature enablement.
- void FLEXCAN_SetLocalPrioCmd (CAN_Type *base, bool enable)
  Set the local transmit priority enablement.
- void FLEXCAN_SetMatchPrioCmd (CAN_Type *base, bool priority)
  Set the Rx matching process priority.

FlexCAN Message buffer control functions

- flexcan_msgbuf_t * FLEXCAN_GetMsgBufPtr (CAN_Type *base, uint8_t msgBufIdx)
  Get message buffer pointer for transition.
- bool FLEXCAN_LockRxMsgBuf (CAN_Type *base, uint8_t msgBufIdx)
  Locks the FlexCAN Rx message buffer.
- uint16_t FLEXCAN_UnlockAllRxMsgBuf (CAN_Type *base)
  Unlocks the FlexCAN Rx message buffer.

FlexCAN Interrupts and flags management functions

- void FLEXCAN_SetMsgBufIntCmd (CAN_Type *base, uint8_t msgBufIdx, bool enable)
  Enables/Disables the FlexCAN Message Buffer interrupt.
- bool FLEXCAN_GetMsgBufStatusFlag (CAN_Type *base, uint8_t msgBufIdx)
  Gets the individual FlexCAN MB interrupt flag.
- void FLEXCAN_ClearMsgBufStatusFlag (CAN_Type *base, uint32_t msgBufIdx)
  Clears the interrupt flag of the message buffers.
- void FLEXCAN_SetErrIntCmd (CAN_Type *base, uint32_t errorSrc, bool enable)
  Enables error interrupt of the FlexCAN module.
- uint32_t FLEXCAN_GetErrStatusFlag (CAN_Type *base, uint32_t errFlags)
  Gets the FlexCAN module interrupt flag.
- void FLEXCAN_ClearErrStatusFlag (CAN_Type *base, uint32_t errFlags)
  Clears the interrupt flag of the FlexCAN module.

- void FLEXCAN_GetErrCounter (CAN_Type *base, uint8_t *txError, uint8_t *rxError)
  Get the error counter of FlexCAN module.

Rx FIFO management functions

- void FLEXCAN_EnableRxFifo (CAN_Type *base, uint8_t numOfFilters)
  Enables the Rx FIFO.
- void FLEXCAN_DisableRxFifo (CAN_Type *base)
  Disables the Rx FIFO.
- void FLEXCAN_SetRxFifoFilterNum (CAN_Type *base, uint32_t numOfFilters)
  Set the number of the Rx FIFO filters.
- void FLEXCAN_SetRxFifoFilter (CAN_Type *base, uint32_t idFormat, flexcan_id_table_t *idFilterTable)
  Set the FlexCAN Rx FIFO fields.
- flexcan_msgbuf_t * FLEXCAN_GetRxFifoPtr (CAN_Type *base)
  Gets the FlexCAN Rx FIFO data pointer.
- uint16_t FLEXCAN_GetRxFifoInfo (CAN_Type *base)
  Gets the FlexCAN Rx FIFO information.

Rx Mask Setting functions

- void FLEXCAN_SetRxMaskMode (CAN_Type *base, uint32_t mode)
  Set the Rx masking mode.
- void FLEXCAN_SetRxMaskRtrCmd (CAN_Type *base, bool enable)
  Set the remote trasmit request mask enablement.
- void FLEXCAN_SetRxGlobalMask (CAN_Type *base, uint32_t mask)
  Set the FlexCAN RX global mask.
- void FLEXCAN_SetRxIndividualMask (CAN_Type *base, uint32_t msgBufIdx, uint32_t mask)
  Set the FlexCAN Rx individual mask for ID filtering in the Rx MBs and the Rx FIFO.
- void FLEXCAN_SetRxMsgBuff14Mask (CAN_Type *base, uint32_t mask)
  Set the FlexCAN RX Message Buffer BUF14 mask.
- void FLEXCAN_SetRxMsgBuff15Mask (CAN_Type *base, uint32_t mask)
  Set the FlexCAN RX Message Buffer BUF15 mask.
- void FLEXCAN_SetRxFifoGlobalMask (CAN_Type *base, uint32_t mask)
  Set the FlexCAN RX Fifo global mask.

Misc. Functions

- void FLEXCAN_SetSelfWakeUpCmd (CAN_Type *base, bool lpfEnable, bool enable)
  Enable/disable the FlexCAN self wakeup feature.
- void FLEXCAN_SetSelfReceptionCmd (CAN_Type *base, bool enable)
  Enable/Disable the FlexCAN self reception feature.
- void FLEXCAN_SetRxVoteCmd (CAN_Type *base, bool enable)
  Enable/disable the enhance FlexCAN Rx vote.
- void FLEXCAN_SetAutoBusOffRecoverCmd (CAN_Type *base, bool enable)
  Enable/disable the Auto Busoff recover feature.
- void FLEXCAN_SetTimeSyncCmd (CAN_Type *base, bool enable)

Enable/disable the Time Sync feature.
- void FLEXCAN_SetAutoRemoteResponseCmd (CAN_Type *base, bool enable)
  Enable/disable the Auto Remote Response feature.
- static void FLEXCAN_SetGlitchFilterWidth (CAN_Type *base, uint8_t filterWidth)
  Enable/disable the Glitch Filter Width when FLEXCAN enters the STOP mode.
- static uint32_t FLEXCAN_GetLowestInactiveMsgBuf (CAN_Type *base)
  Get the lowest inactive message buffer number.
- static void FLEXCAN_SetTxArbitrationStartDelay (CAN_Type *base, uint8_t tasd)
  Set the Tx Arbitration Start Delay number.

7.2.4 Data Structure Documentation

7.2.4.1 struct flexcan_id_table_t

Data Fields

- uint32_t * idFilter
  Rx FIFO ID filter elements.
- bool isRemoteFrame
  Remote frame.
- bool isExtendedFrame
  Extended frame.

7.2.4.1.0.4 Field Documentation

7.2.4.1.0.4.1 uint32_t* flexcan_id_table_t::idFilter

7.2.4.1.0.4.2 bool flexcan_id_table_t::isRemoteFrame

7.2.4.1.0.4.3 bool flexcan_id_table_t::isExtendedFrame

7.2.4.2 struct flexcan_msgbuf_t

7.2.4.2.0.5 Field Documentation

7.2.4.2.0.5.1 uint32_t flexcan_msgbuf_t::cs

7.2.4.2.0.5.2 uint32_t flexcan_msgbuf_t::id

7.2.4.2.0.5.3 uint32_t flexcan_msgbuf_t::word0

7.2.4.2.0.5.4 uint32_t flexcan_msgbuf_t::word1

7.2.4.3 struct flexcan_timing_t

Data Fields

- uint32_t preDiv
  Clock pre divider.

- uint32_t rJumpwidth
  Resync jump width.
- uint32_t phaseSeg1
  Phase segment 1.
- uint32_t phaseSeg2
  Phase segment 2.
- uint32_t propSeg
  Propagation segment.

7.2.4.3.0.6 Field Documentation

7.2.4.3.0.6.1 uint32_t flexcan_timing_t::preDiv

7.2.4.3.0.6.2 uint32_t flexcan_timing_t::rJumpwidth

7.2.4.3.0.6.3 uint32_t flexcan_timing_t::phaseSeg1

7.2.4.3.0.6.4 uint32_t flexcan_timing_t::phaseSeg2

7.2.4.3.0.6.5 uint32_t flexcan_timing_t::propSeg

7.2.4.4 struct flexcan_init_config_t

Data Fields

- flexcan_timing_t timing
  Desired FlexCAN module timing configuration.
- uint32_t operatingMode
  Desired FlexCAN module operating mode.
- uint8_t maxMsgBufNum
  The maximal number of available message buffer.

7.2.4.4.0.7 Field Documentation

7.2.4.4.0.7.1 flexcan_timing_t flexcan_init_config_t::timing

7.2.4.4.0.7.2 uint32_t flexcan_init_config_t::operatingMode

7.2.4.4.0.7.3 uint8_t flexcan_init_config_t::maxMsgBufNum

7.2.5 Enumeration Type Documentation

7.2.5.1 enum _flexcan_msgbuf_code_rx

Enumerator

flexcanRxInactive MB is not active.
flexcanRxFull MB is full.
flexcanRxEmpty MB is active and empty.
flexcanRxOverrun MB is overwritten into a full buffer.

flexcanRxBusy FlexCAN is updating the contents of the MB.
flexcanRxRanswer The CPU must not access the MB. A frame was configured to recognize a Remote Request Frame
flexcanRxNotUsed and transmit a Response Frame in return. Not used.

7.2.5.2 enum _flexcan_msgbuf_code_tx

Enumerator

flexcanTxInactive MB is not active.
flexcanTxAbort MB is aborted.
flexcanTxDataOrRemte MB is a TX Data Frame(when MB RTR = 0) or. MB is a TX Remote Request Frame (when MB RTR = 1).
flexcanTxTanswer MB is a TX Response Request Frame from.
flexcanTxNotUsed an incoming Remote Request Frame. Not used.

7.2.5.3 enum _flexcan_operatining_modes

Enumerator

flexcanNormalMode Normal mode or user mode.
flexcanListenOnlyMode Listen-only mode.
flexcanLoopBackMode Loop-back mode.

7.2.5.4 enum _flexcan_rx_mask_mode

Enumerator

flexcanRxMaskGlobal Rx global mask.
flexcanRxMaskIndividual Rx individual mask.

7.2.5.5 enum _flexcan_rx_mask_id_type

Enumerator

flexcanRxMaskIdStd Standard ID.
flexcanRxMaskIdExt Extended ID.

7.2.5.6 enum _flexcan_interrutpt

Enumerator

flexcanIntRxWarning Tx Warning interrupt source.
flexcanIntTxWarning Tx Warning interrupt source.
flexcanIntWakeUp Wake Up interrupt source.
flexcanIntBusOff Bus Off interrupt source.
flexcanIntError Error interrupt source.

7.2.5.7 enum _flexcan_status_flag

Enumerator

flexcanStatusSynch Bus Synchronized flag.
flexcanStatusTxWarningInt Tx Warning initerrupt flag.
flexcanStatusRxWarningInt Tx Warning initerrupt flag.
flexcanStatusBit1Err Bit0 Error flag.
flexcanStatusBit0Err Bit1 Error flag.
flexcanStatusAckErr Ack Error flag.
flexcanStatusCrcErr CRC Error flag.
flexcanStatusFrameErr Frame Error flag.
flexcanStatusStuffingErr Stuffing Error flag.
flexcanStatusTxWarning Tx Warning flag.
flexcanStatusRxWarning Rx Warning flag.
flexcanStatusIdle FlexCAN Idle flag.
flexcanStatusTransmitting Trasmitting flag.
flexcanStatusFltConf Fault Config flag.
flexcanStatusReceiving Receiving flag.
flexcanStatusBusOff Bus Off interrupt flag.
flexcanStatusError Error interrupt flag.
flexcanStatusWake Wake Up interrupt flag.

7.2.5.8 enum _flexcan_rx_fifo_id_element_format

Enumerator

flexcanRxFifoIdElementFormatA One full ID (standard and extended) per ID Filter Table element.

flexcanRxFifoIdElementFormatB Two full standard IDs or two partial 14-bit (standard and extended) IDs per ID Filter Table element.
flexcanRxFifoIdElementFormatC Four partial 8-bit Standard IDs per ID Filter Table element.
flexcanRxFifoIdElementFormatD All frames rejected.

7.2.5.9 enum _flexcan_rx_fifo_filter_id_number

Enumerator

flexcanRxFifoIdFilterNum8 8 Rx FIFO Filters.
flexcanRxFifoIdFilterNum16 16 Rx FIFO Filters.
flexcanRxFifoIdFilterNum24 24 Rx FIFO Filters.
flexcanRxFifoIdFilterNum32 32 Rx FIFO Filters.
flexcanRxFifoIdFilterNum40 40 Rx FIFO Filters.
flexcanRxFifoIdFilterNum48 48 Rx FIFO Filters.
flexcanRxFifoIdFilterNum56 56 Rx FIFO Filters.
flexcanRxFifoIdFilterNum64 64 Rx FIFO Filters.
flexcanRxFifoIdFilterNum72 72 Rx FIFO Filters.
flexcanRxFifoIdFilterNum80 80 Rx FIFO Filters.
flexcanRxFifoIdFilterNum88 88 Rx FIFO Filters.
flexcanRxFifoIdFilterNum96 96 Rx FIFO Filters.
flexcanRxFifoIdFilterNum104 104 Rx FIFO Filters.
flexcanRxFifoIdFilterNum112 112 Rx FIFO Filters.
flexcanRxFifoIdFilterNum120 120 Rx FIFO Filters.
flexcanRxFifoIdFilterNum128 128 Rx FIFO Filters.

7.2.6 Function Documentation

7.2.6.1 void FLEXCAN_Init (CAN_Type * *base*, const flexcan_init_config_t * *initConfig*)

Parameters

base	CAN base pointer.
initConfig	CAN initialization structure (see flexcan_init_config_t structure).

7.2.6.2 void FLEXCAN_Deinit (CAN_Type * *base*)

Parameters

base	FlexCAN base pointer.

7.2.6.3 void FLEXCAN_Enable (CAN_Type * *base*)

Parameters

base	FlexCAN base pointer.

7.2.6.4 void FLEXCAN_Disable (CAN_Type * *base*)

Parameters

base	FlexCAN base pointer.

7.2.6.5 void FLEXCAN_SetTiming (CAN_Type * *base*, const flexcan_timing_t * *timing*)

Parameters

base	FlexCAN base pointer.
timing	FlexCAN time segments, which need to be set for the bit rate (See flexcan_timing_t structure).

7.2.6.6 void FLEXCAN_SetOperatingMode (CAN_Type * *base*, uint8_t *mode*)

Parameters

base	FlexCAN base pointer.
mode	Set an operation mode.

7.2.6.7 void FLEXCAN_SetMaxMsgBufNum (CAN_Type * *base*, uint32_t *bufNum*)

Parameters

base	FlexCAN base pointer.
bufNum	Maximum number of message buffers.

7.2.6.8 static bool FLEXCAN_IsModuleReady (CAN_Type * *base*) [inline], [static]

Parameters

base	FlexCAN base pointer.

Returns

- true: FLEXCAN module is either in Normal Mode, Listen-Only Mode or Loop-Back Mode.
  - false: FLEXCAN module is either in Disable Mode, Stop Mode or Freeze Mode.

7.2.6.9 void FLEXCAN_SetAbortCmd (CAN_Type * *base,* bool *enable*)

Parameters

base	FlexCAN base pointer.
enable	Enable/Disable Transmit Abort feature. • true: Enable Transmit Abort feature. • false: Disable Transmit Abort feature.

7.2.6.10 void FLEXCAN_SetLocalPrioCmd (CAN_Type * *base,* bool *enable*)

Parameters

base	FlexCAN base pointer.
enable	Enable/Disable local transmit periority. • true: Transmit MB with highest local priority. • false: Transmit MB with lowest MB number.

7.2.6.11 void FLEXCAN_SetMatchPrioCmd (CAN_Type * *base,* bool *priority*)

Parameters

base	FlexCAN base pointer.

priority	Set Rx matching process priority. • true: Matching starts from Mailboxes and continues on Rx FIFO. • false: Matching starts from Rx FIFO and continues on Mailboxes.

7.2.6.12 flexcan_msgbuf_t* FLEXCAN_GetMsgBufPtr (CAN_Type * *base,* uint8_t *msgBufIdx*)

Parameters

base	FlexCAN base pointer.
msgBufIdx	message buffer index.

Returns

message buffer pointer.

7.2.6.13 bool FLEXCAN_LockRxMsgBuf (CAN_Type * *base,* uint8_t *msgBufIdx*)

Parameters

base	FlexCAN base pointer.
msgBufIdx	Index of the message buffer

Returns

- true: Lock Rx Message Buffer successful.
  • false: Lock Rx Message Buffer failed.

7.2.6.14 uint16_t FLEXCAN_UnlockAllRxMsgBuf (CAN_Type * *base*)

Parameters

base	FlexCAN base pointer.

Returns

current free run timer counter value.

7.2.6.15 void FLEXCAN_SetMsgBufIntCmd (CAN_Type * *base,* uint8_t *msgBufIdx,* bool *enable*)

Parameters

base	FlexCAN base pointer.
msgBufIdx	Index of the message buffer.
enable	Enables/Disables interrupt. • true: Enable Message Buffer interrupt. • disable: Disable Message Buffer interrupt.

7.2.6.16 bool FLEXCAN_GetMsgBufStatusFlag (CAN_Type * *base,* uint8_t *msgBufIdx*)

Parameters

base	FlexCAN base pointer.
msgBufIdx	Index of the message buffer.

Return values

true,:	Message Buffer Interrupt is pending.
false,:	There is no Message Buffer Interrupt.

7.2.6.17 void FLEXCAN_ClearMsgBufStatusFlag (CAN_Type * *base,* uint32_t *msgBufIdx*)

Parameters

base	FlexCAN base pointer.
msgBufIdx	Index of the message buffer.

7.2.6.18 void FLEXCAN_SetErrIntCmd (CAN_Type * *base,* uint32_t *errorSrc,* bool *enable*)

Parameters

base	FlexCAN base pointer.
errorSrc	The interrupt source (see _flexcan_interrutpt enumeration).
enable	Choose enable or disable.

7.2.6.19 uint32_t FLEXCAN_GetErrStatusFlag (CAN_Type * *base*, uint32_t *errFlags*)

Parameters

base	FlexCAN base pointer.
errFlags	FlexCAN error flags (see _flexcan_status_flag enumeration).

Returns

The individual Message Buffer interrupt flag (0 and 1 are the flag value)

7.2.6.20 void FLEXCAN_ClearErrStatusFlag (CAN_Type * *base*, uint32_t *errFlags*)

Parameters

base	FlexCAN base pointer.
errFlags	The value to be written to the interrupt flag1 register (see _flexcan_status_flag enumeration).

7.2.6.21 void FLEXCAN_GetErrCounter (CAN_Type * *base*, uint8_t * *txError*, uint8_t * *rxError*)

Parameters

base	FlexCAN base pointer.
txError	Tx_Err_Counter pointer.
rxError	Rx_Err_Counter pointer.

7.2.6.22 void FLEXCAN_EnableRxFifo (CAN_Type * *base*, uint8_t *numOfFilters*)

Parameters

base	FlexCAN base pointer.
numOfFilters	The number of Rx FIFO filters

7.2.6.23 void FLEXCAN_DisableRxFifo (CAN_Type * *base*)

Parameters

base	FlexCAN base pointer.

7.2.6.24 void FLEXCAN_SetRxFifoFilterNum (CAN_Type * *base,* uint32_t *numOfFilters*)

Parameters

base	FlexCAN base pointer.
numOfFilters	The number of Rx FIFO filters.

7.2.6.25 void FLEXCAN_SetRxFifoFilter (CAN_Type * *base,* uint32_t *idFormat,* flexcan_id_table_t * *idFilterTable*)

Parameters

base	FlexCAN base pointer.
idFormat	The format of the Rx FIFO ID Filter Table Elements
idFilterTable	The ID filter table elements which contain RTR bit, IDE bit and RX message ID.

7.2.6.26 flexcan_msgbuf_t* FLEXCAN_GetRxFifoPtr (CAN_Type * *base*)

Parameters

base	FlexCAN base pointer.

Returns

Rx FIFO data pointer.

7.2.6.27 uint16_t FLEXCAN_GetRxFifoInfo (CAN_Type * *base*)

```
The return value indicates which Identifier Acceptance Filter
(see Rx FIFO Structure) was hit by the received message.
```

Parameters

base	FlexCAN base pointer.

Returns

Rx FIFO filter number.

7.2.6.28 void FLEXCAN_SetRxMaskMode (CAN_Type * *base*, uint32_t *mode*)

Parameters

base	FlexCAN base pointer.
mode	The FlexCAN Rx mask mode (see _flexcan_rx_mask_mode enumeration).

7.2.6.29 void FLEXCAN_SetRxMaskRtrCmd (CAN_Type * *base*, bool *enable*)

Parameters

base	FlexCAN base pointer.
enable	Enable/Disable remote trasmit request mask. • true: Enable RTR matching judgement. • false: Disable RTR matching judgement.

7.2.6.30 void FLEXCAN_SetRxGlobalMask (CAN_Type * *base*, uint32_t *mask*)

Parameters

base	FlexCAN base pointer.
mask	Rx Global mask.

7.2.6.31 void FLEXCAN_SetRxIndividualMask (CAN_Type * *base*, uint32_t *msgBufIdx*, uint32_t *mask*)

Parameters

base	FlexCAN base pointer.
msgBufIdx	Index of the message buffer.
mask	Individual mask

7.2.6.32 void FLEXCAN_SetRxMsgBuff14Mask (CAN_Type * *base*, uint32_t *mask*)

Parameters

base	FlexCAN base pointer.
mask	Message Buffer BUF14 mask.

7.2.6.33 void FLEXCAN_SetRxMsgBuff15Mask (CAN_Type * *base*, uint32_t *mask*)

Parameters

base	FlexCAN base pointer.
mask	Message Buffer BUF15 mask.

7.2.6.34 void FLEXCAN_SetRxFifoGlobalMask (CAN_Type * *base*, uint32_t *mask*)

Parameters

base	FlexCAN base pointer.
mask	Rx Fifo Global mask.

7.2.6.35 void FLEXCAN_SetSelfWakeUpCmd (CAN_Type * *base*, bool *lpfEnable*, bool *enable*)

Parameters

base	FlexCAN base pointer.
lpfEnable	The low pass filter for Rx self wakeup feature enablement.
enable	The self wakeup feature enablement.

7.2.6.36 void FLEXCAN_SetSelfReceptionCmd (CAN_Type * *base*, bool *enable*)

Parameters

base	FlexCAN base pointer.
enable	Enable/Disable self reception feature. • true: Enable self reception feature. • false: Disable self reception feature.

7.2.6.37 void FLEXCAN_SetRxVoteCmd (CAN_Type * *base*, bool *enable*)

Parameters

base	FlexCAN base pointer.
enable	Enable/Disable FlexCAN Rx vote mechanism • true: Three samples are used to determine the value of the received bit. • false: Just one sample is used to determine the bit value.

7.2.6.38 void FLEXCAN_SetAutoBusOffRecoverCmd (CAN_Type * *base*, bool *enable*)

Parameters

base	FlexCAN base pointer.
enable	Enable/Disable Auto Busoff Recover • true: Enable Auto Bus Off recover feature. • false: Disable Auto Bus Off recover feature.

7.2.6.39 void FLEXCAN_SetTimeSyncCmd (CAN_Type * *base*, bool *enable*)

Parameters

base	FlexCAN base pointer.
enable	Enable/Disable the Time Sync • true: Enable Time Sync feature. • false: Disable Time Sync feature.

7.2.6.40 void FLEXCAN_SetAutoRemoteResponseCmd (CAN_Type * *base*, bool *enable*)

Parameters

base	FlexCAN base pointer.
enable	Enable/Disable the Auto Remote Response feature • true: Enable Auto Remote Response feature. • false: Disable Auto Remote Response feature.

7.2.6.41 static void FLEXCAN_SetGlitchFilterWidth (CAN_Type * *base*, uint8_t *filterWidth*) [inline], [static]

Parameters

base	FlexCAN base pointer.
filterWidth	The Glitch Filter Width.

7.2.6.42 static uint32_t FLEXCAN_GetLowestInactiveMsgBuf (CAN_Type * *base*) [inline], [static]

Parameters

base	FlexCAN base pointer.

Returns

bit 22-16 : The lowest number inactive Mailbox. bit 14 : Indicates whether the number content is valid or not. bit 13 : This bit indicates whether there is any inactive Mailbox.

7.2.6.43 static void FLEXCAN_SetTxArbitrationStartDelay (CAN_Type * *base*, uint8_t *tasd*) [inline], [static]

```
This function is used to optimize the transmit performance.
For more information about to set this value, see the Chip Reference Manual.
```

Parameters

base	FlexCAN base pointer.
tasd	The lowest number inactive Mailbox.

Chapter 8
General Purpose Input/Output (GPIO)

8.1 Overview

The FreeRTOS BSP provides a driver for the General Purpose Input/Output (GPIO) block of i.MX devices.

Modules

- GPIO driver

8.2 GPIO driver

8.2.1 Overview

This chapter describes the programming interface of the GPIO driver (platform/drivers/inc/gpio_imx.h). The GPIO driver configures pins to digital input/output or interrupt mode and provides a set of APIs to access these registers, including these services:

- GPIO pin configuration;
- GPIO pin input/output operation;
- GPIO pin interrupt management;

8.2.2 GPIO pin configuration

Configure GPIO pins according to the target board and ensure that the configurations are correct. Define gpio pins configuration file based on a specific board to store the GPIO pin configurations.

GPIO pin configuration file example:

```
// Feel free to change the pin name, base, pin number, muxReg and padReg as what you want.
gpio_config_t gpioLed = {
    "USER LED",                          // name
    &IOMUXC_SW_MUX_CTL_PAD_GPIO1_IO09,  // muxReg
    0,                                   // muxConfig
    &IOMUXC_SW_PAD_CTL_PAD_GPIO1_IO09,  // padReg
    0,                                   // padConfig
    GPIO1,                               // base
    9                                    // pin
};

// Configure a specific GPIO pin.
configure_gpio_pin(&gpioLed);
```

8.2.3 GPIO initialization

To initialize the GPIO module, define a structure gpio_init_config_t. First, configure the structure. Then, call the GPIO_Init() function and pass the initialization structure.

This is an example of the GPIO module Initialization:

```
#include "gpio_imx.h"

#define BOARD_GPIO_LED_CONFIG   &gpioLed

    // Configure "USER LED" as a digital output and no interrupt mode.
    gpio_init_config_t ledInitConfig = {
        .pin = BOARD_GPIO_LED_CONFIG->pin,
        .direction = gpioDigitalOutput,
        .interruptMode = gpioNoIntmode
    };

    //Initializes GPIO module.
    GPIO_Init(BOARD_GPIO_LED_CONFIG->base, &ledInitConfig);
```

Note: interruptMode can also be configured as a value of gpio_interrupt_mode_t.

8.2.4 Output operations

To use the output operation, configure the target GPIO pin as a digital output in gpio_init_config_t structure. The output operation is provided to configure the output logic level according to passed parameters:

```
void GPIO_WritePinOutput(GPIO_Type* base, uint32_t pin,
      gpio_pin_action_t pinVal);
static inline void GPIO_WritePortOutput(GPIO_Type* base, uint32_t portVal);
```

GPIO_WritePinOutput() function is used for single pin. And GPIO_WritePortOutput() function is used for all 32 pins of a GPIO instance.

8.2.5 Input operations

To use the input operation, configure the target GPIO pin as a digital input in the gpio_init_config_t structure. For the input operation, this is the most commonly used API function:

```
static inline uint8_t GPIO_ReadPinInput(GPIO_Type* base, uint32_t pin);
```

8.2.6 Read pad status

To use the read pad status operation, no care configuring GPIO pin as a input or output. This operation can read a specific GPIO pin logic level according to passed parameters:

```
static inline uint8_t GPIO_ReadPadStatus(GPIO_Type* base, uint32_t pin);
```

8.2.7 GPIO interrupt

Enable a specific pin interrupt in GPIO initialization structures according to configure the interrupt mode. The following API functions are used to manage the interrupt and status flags:

```
void GPIO_SetPinIntMode(GPIO_Type* base, uint32_t pin, bool enable);
static inline bool GPIO_IsIntPending(GPIO_Type* base, uint32_t pin);
static inline void GPIO_ClearStatusFlag(GPIO_Type* base, uint32_t pin);
```

GPIO_SetPinIntMode() function can enable or disable a specific GPIO pin. GPIO_IsIntPending() can check individual pin interrupt status. GPIO_ClearStatusFlag() can clear pin interrupt flag by writing a 1 to the corresponding bit position.

Data Structures

- struct gpio_init_config_t
  GPIO Init structure definition. More...

Enumerations

- enum gpio_pin_direction_t {
  gpioDigitalInput = 0U,
  gpioDigitalOutput = 1U }
  GPIO direction definition.
- enum gpio_interrupt_mode_t {
  gpioIntLowLevel = 0U,
  gpioIntHighLevel = 1U,
  gpioIntRisingEdge = 2U,
  gpioIntFallingEdge = 3U,
  gpioNoIntmode = 4U }
  GPIO interrupt mode definition.
- enum gpio_pin_action_t {
  gpioPinClear = 0U,
  gpioPinSet = 1U }
  GPIO pin(bit) value definition.

GPIO Initialization and Configuration functions

- void GPIO_Init (GPIO_Type *base, const gpio_init_config_t *initConfig)
  Initializes the GPIO peripheral according to the specified parameters in the initConfig.

GPIO Read and Write Functions

- static uint8_t GPIO_ReadPinInput (GPIO_Type *base, uint32_t pin)
  Reads the current input value of the pin when pin's direction is configured as input.
- static uint32_t GPIO_ReadPortInput (GPIO_Type *base)
  Reads the current input value of a specific GPIO port when port's direction are all configured as input.
- static uint8_t GPIO_ReadPinOutput (GPIO_Type *base, uint32_t pin)
  Reads the current pin output.
- static uint32_t GPIO_ReadPortOutput (GPIO_Type *base)
  Reads out all pin output status of the current port.
- void GPIO_WritePinOutput (GPIO_Type *base, uint32_t pin, gpio_pin_action_t pinVal)
  Sets the output level of the individual GPIO pin to logic 1 or 0.
- static void GPIO_WritePortOutput (GPIO_Type *base, uint32_t portVal)
  Sets the output of the GPIO port pins to a specific logic value.

GPIO Read Pad Status Functions

- static uint8_t GPIO_ReadPadStatus (GPIO_Type *base, uint32_t pin)
  Reads the current GPIO pin pad status.

Interrupts and flags management functions

- void GPIO_SetPinIntMode (GPIO_Type *base, uint32_t pin, bool enable)
  Enable or Disable the specific pin interrupt.
- static bool GPIO_IsIntPending (GPIO_Type *base, uint32_t pin)
  Check individual pin interrupt status.
- static void GPIO_ClearStatusFlag (GPIO_Type *base, uint32_t pin)
  Clear pin interrupt flag.
- void GPIO_SetIntEdgeSelect (GPIO_Type *base, uint32_t pin, bool enable)
  Enable or disable the edge select bit to override the ICR register's configuration.

8.2.8 Data Structure Documentation

8.2.8.1 struct gpio_init_config_t

Data Fields

- uint32_t pin
  Specifies the pin number.
- gpio_pin_direction_t direction
  Specifies the pin direction.
- gpio_interrupt_mode_t interruptMode
  Specifies the pin interrupt mode, a value of gpio_interrupt_mode_t.

8.2.8.1.0.8 Field Documentation

8.2.8.1.0.8.1 uint32_t gpio_init_config_t::pin

8.2.8.1.0.8.2 gpio_pin_direction_t gpio_init_config_t::direction

8.2.8.1.0.8.3 gpio_interrupt_mode_t gpio_init_config_t::interruptMode

8.2.9 Enumeration Type Documentation

8.2.9.1 enum gpio_pin_direction_t

Enumerator

gpioDigitalInput Set current pin as digital input.
gpioDigitalOutput Set current pin as digital output.

8.2.9.2 enum gpio_interrupt_mode_t

Enumerator

gpioIntLowLevel Set current pin interrupt is low-level sensitive.
gpioIntHighLevel Set current pin interrupt is high-level sensitive.

gpioIntRisingEdge Set current pin interrupt is rising-edge sensitive.
gpioIntFallingEdge Set current pin interrupt is falling-edge sensitive.
gpioNoIntmode Set current pin general IO functionality.

8.2.9.3 enum gpio_pin_action_t

Enumerator

gpioPinClear Clear GPIO Pin.
gpioPinSet Set GPIO Pin.

8.2.10 Function Documentation

8.2.10.1 void GPIO_Init (GPIO_Type * *base*, const gpio_init_config_t * *initConfig*)

Parameters

base	GPIO base pointer.
initConfig	pointer to a gpio_init_config_t structure that contains the configuration information.

8.2.10.2 static uint8_t GPIO_ReadPinInput (GPIO_Type * *base*, uint32_t *pin*) [inline], [static]

Parameters

base	GPIO base pointer.
pin	GPIO port pin number.

Returns

GPIO pin input value.

8.2.10.3 static uint32_t GPIO_ReadPortInput (GPIO_Type * *base*) [inline], [static]

```
This function  gets all 32-pin input as a 32-bit integer.
```

Parameters

base	GPIO base pointer.

Returns

GPIO port input data.

8.2.10.4 static uint8_t GPIO_ReadPinOutput (GPIO_Type * *base,* uint32_t *pin*) [inline], [static]

Parameters

base	GPIO base pointer.
pin	GPIO port pin number.

Returns

Current pin output value.

8.2.10.5 static uint32_t GPIO_ReadPortOutput (GPIO_Type * *base*) [inline], [static]

```
This function operates all 32 port pins.
```

Parameters

base	GPIO base pointer.

Returns

Current port output status.

8.2.10.6 void GPIO_WritePinOutput (GPIO_Type * *base,* uint32_t *pin,* gpio_pin_action_t *pinVal*)

Parameters

base	GPIO base pointer.
pin	GPIO port pin number.
pinVal	pin output value (See gpio_pin_action_t structure).

8.2.10.7 static void GPIO_WritePortOutput (GPIO_Type * *base*, uint32_t *portVal*) [inline], [static]

```
This function  operates all 32 port pins.
```

Parameters

base	GPIO base pointer.
portVal	data to configure the GPIO output.

8.2.10.8 static uint8_t GPIO_ReadPadStatus (GPIO_Type * *base*, uint32_t *pin*) [inline], [static]

Parameters

base	GPIO base pointer.
pin	GPIO port pin number.

Returns

GPIO pin pad status value.

8.2.10.9 void GPIO_SetPinIntMode (GPIO_Type * *base*, uint32_t *pin*, bool *enable*)

Parameters

base	GPIO base pointer.

pin	GPIO pin number.
enable	Enable or disable interrupt. • true: Enable GPIO interrupt. • false: Disable GPIO interrupt.

8.2.10.10 static bool GPIO_IsIntPending (GPIO_Type * *base,* uint32_t *pin*) [inline], [static]

Parameters

base	GPIO base pointer.
pin	GPIO port pin number.

Returns

current pin interrupt status flag.

8.2.10.11 static void GPIO_ClearStatusFlag (GPIO_Type * *base,* uint32_t *pin*) [inline], [static]

Status flags are cleared by writing a 1 to the corresponding bit position.

Parameters

base	GPIO base pointer.
pin	GPIO port pin number.

8.2.10.12 void GPIO_SetIntEdgeSelect (GPIO_Type * *base,* uint32_t *pin,* bool *enable*)

Parameters

base	GPIO base pointer.
pin	GPIO port pin number.

enable	Enable or disable edge select bit.

Chapter 9
InterIntegrated Circuit (I2C)

9.1 Overview

The FreeRTOS BSP provides a driver for the InterIntegrated Circuit (I2C) block of i.MX devices.

Modules

- I2C driver

9.2 I2C driver

9.2.1 Overview

The section describes the programming interface of the I2C driver (platform/drivers/inc/i2c_imx.h).

9.2.2 I2C initialization

To initialize the I2C module, define an i2c_init_config_t type variable and pass it to the I2C_Init() function. Here is the members of the structure definition:

1. clockRate: Current I2C module clock frequency. This variable can be obtained by calling get_i2c_-clock_freq() function.
2. baudRate: Desired I2C baud rate. The legal baud rate should not exceed 400 kHz, which is the highest baud rate supported by this module.
3. slaveAddress: I2C module's own address when addressed as the slave device. Note that this value is the I2C module's own address, not the I2C slave's address that you want to communicate with.

After the I2C module is initialized, call I2C_Enable() to enable the I2C module before any data transaction.

9.2.3 I2C data transactions

I2C driver provides these APIs for data transactions:

```
I2C_WriteByte
I2C_ReadByte
I2C_SendRepeatStart
I2C_SetWorkMode
I2C_SetDirMode
I2C_SetAckBit
```

I2C data send

To send data through the I2C bus, follow these steps:

1. Set the I2C module to work under Tx mode by calling I2C_SetDirMode().
2. Switch to Master Mode and Send Start Signal by calling I2C_SetWorkMode().
3. Send the data to I2C bus by calling I2C_WriteByte().
4. Wait for I2C interrupt or poll the I2C status bit to see if the data is sent successfully.

I2C data receive

To receive data through the I2C bus, follow these steps:

1. Set the I2C module to work under Rx mode by calling I2C_SetDirMode().
2. Switch to Master Mode and Send Start Signal by calling I2C_SetWorkMode().

3. Call I2C_ReadByte() to trigger a I2C bus Read.
4. Wait for I2C interrupt or poll I2C status bit to see if the data is received successfully.
5. Read data by calling I2C_ReadByte() function.

I2C status and interrupt

This driver also provides APIs to handle I2C module Status and Interrupt:

1. Call I2C_SetIntCmd() to enable/disable I2C module interrupt.
2. Call I2C_GetStatusFlag() to get the I2C status flags (described in enum _i2c_status_flag) condition.
3. Call I2C_ClearStatusFlag() to clear specified status flags.

Example

For more information about how to use this driver, see I2C demo/example under examples/<board_name>/.

Data Structures

- struct i2c_init_config_t
  I2C module initialization structure. More...

Enumerations

- enum _i2c_status_flag {
  i2cStatusTransferComplete = I2C_I2SR_ICF_MASK,
  i2cStatusAddressedAsSlave = I2C_I2SR_IAAS_MASK,
  i2cStatusBusBusy = I2C_I2SR_IBB_MASK,
  i2cStatusArbitrationLost = I2C_I2SR_IAL_MASK,
  i2cStatusSlaveReadWrite = I2C_I2SR_SRW_MASK,
  i2cStatusInterrupt = I2C_I2SR_IIF_MASK,
  i2cStatusReceivedAck = I2C_I2SR_RXAK_MASK }
  Flag for I2C interrupt status check or polling status.
- enum _i2c_work_mode {
  i2cModeSlave = 0x0,
  i2cModeMaster = I2C_I2CR_MSTA_MASK }
  I2C Bus role of this module.
- enum _i2c_direction_mode {
  i2cDirectionReceive = 0x0,
  i2cDirectionTransmit = I2C_I2CR_MTX_MASK }
  Data transfer direction.

Variables

- uint32_t i2c_init_config_t::clockRate
  Current I2C module clock freq.
- uint32_t i2c_init_config_t::baudRate
  Desired I2C baud rate.
- uint8_t i2c_init_config_t::slaveAddress
  I2C module's own address when addressed as slave device.

I2C Initialization and Configuration functions

- void I2C_Init (I2C_Type *base, const i2c_init_config_t *initConfig)
  Initialize I2C module with given initialization structure.
- void I2C_Deinit (I2C_Type *base)
  This function reset I2C module register content to its default value.
- static void I2C_Enable (I2C_Type *base)
  This function is used to Enable the I2C Module.
- static void I2C_Disable (I2C_Type *base)
  This function is used to Disable the I2C Module.
- void I2C_SetBaudRate (I2C_Type *base, uint32_t clockRate, uint32_t baudRate)
  This function is used to set the baud rate of I2C Module.
- static void I2C_SetSlaveAddress (I2C_Type *base, uint8_t slaveAddress)
  This function is used to set the own I2C bus address when addressed as a slave.

I2C Bus Control functions

- static void I2C_SendRepeatStart (I2C_Type *base)
  This function is used to Generate a Repeat Start Signal on I2C Bus.
- static void I2C_SetWorkMode (I2C_Type *base, uint32_t mode)
  This function is used to select the I2C bus role of this module, both I2C Bus Master and Slave can be select.
- static void I2C_SetDirMode (I2C_Type *base, uint32_t direction)
  This function is used to select the data transfer direction of this module, both Transmit and Receive can be select.
- void I2C_SetAckBit (I2C_Type *base, bool ack)
  This function is used to set the Transmit Acknowledge action when receive data from other device.

Data transfers functions

- static void I2C_WriteByte (I2C_Type *base, uint8_t byte)
  Writes one byte of data to the I2C bus.
- static uint8_t I2C_ReadByte (I2C_Type *base)
  Returns the last byte of data read from the bus and initiate another read.

Interrupts and flags management functions

- void I2C_SetIntCmd (I2C_Type *base, bool enable)
  Enable or disable I2C interrupt requests.
- static uint32_t I2C_GetStatusFlag (I2C_Type *base, uint32_t flags)
  Gets the I2C status flag state.
- static void I2C_ClearStatusFlag (I2C_Type *base, uint32_t flags)
  Clear one or more I2C status flag state.

9.2.4 Data Structure Documentation

9.2.4.1 struct i2c_init_config_t

Data Fields

- uint32_t clockRate
  Current I2C module clock freq.
- uint32_t baudRate
  Desired I2C baud rate.
- uint8_t slaveAddress
  I2C module's own address when addressed as slave device.

9.2.5 Enumeration Type Documentation

9.2.5.1 enum _i2c_status_flag

Enumerator

i2cStatusTransferComplete Data Transfer complete flag.
i2cStatusAddressedAsSlave Addressed as a slave flag.
i2cStatusBusBusy Bus is busy flag.
i2cStatusArbitrationLost Arbitration is lost flag.
i2cStatusSlaveReadWrite Master reading from slave flag(De-assert if master writing to slave).
i2cStatusInterrupt An interrupt is pending flag.
i2cStatusReceivedAck No acknowledge detected flag.

9.2.5.2 enum _i2c_work_mode

Enumerator

i2cModeSlave This module works as I2C Slave.
i2cModeMaster This module works as I2C Master.

9.2.5.3 enum _i2c_direction_mode

Enumerator

i2cDirectionReceive This module works at receive mode.
i2cDirectionTransmit This module works at transmit mode.

9.2.6 Function Documentation

9.2.6.1 void I2C_Init (I2C_Type * *base*, const i2c_init_config_t * *initConfig*)

Parameters

base	I2C base pointer.
initConfig	I2C initialization structure (see i2c_init_config_t).

9.2.6.2 void I2C_Deinit (I2C_Type * *base*)

Parameters

base	I2C base pointer.

9.2.6.3 static void I2C_Enable (I2C_Type * *base*) [inline], [static]

Parameters

base	I2C base pointer.

9.2.6.4 static void I2C_Disable (I2C_Type * *base*) [inline], [static]

Parameters

base	I2C base pointer.

9.2.6.5 void I2C_SetBaudRate (I2C_Type * *base*, uint32_t *clockRate*, uint32_t *baudRate*)

Parameters

base	I2C base pointer.
clockRate	I2C module clock frequency.
baudRate	Desired I2C module baud rate.

9.2.6.6 static void I2C_SetSlaveAddress (I2C_Type * *base*, uint8_t *slaveAddress*) [inline], [static]

Parameters

base	I2C base pointer.
slaveAddress	Own I2C Bus address.

9.2.6.7 static void I2C_SendRepeatStart (I2C_Type * *base*) [inline], [static]

Parameters

base	I2C base pointer.

9.2.6.8 static void I2C_SetWorkMode (I2C_Type * *base*, uint32_t *mode*) [inline], [static]

Parameters

base	I2C base pointer.
mode	I2C Bus role to set (see _i2c_work_mode enumeration).

9.2.6.9 static void I2C_SetDirMode (I2C_Type * *base*, uint32_t *direction*) [inline], [static]

Parameters

base	I2C base pointer.
direction	I2C Bus data transfer direction (see _i2c_direction_mode enumeration).

9.2.6.10 void I2C_SetAckBit (I2C_Type * *base*, bool *ack*)

Parameters

base	I2C base pointer.
ack	The ACK value answerback to remote I2C device. • true: An acknowledge signal is sent to the bus at the ninth clock bit. • false: No acknowledge signal response is sent.

9.2.6.11 static void I2C_WriteByte (I2C_Type * *base*, uint8_t *byte*) [inline], [static]

Parameters

base	I2C base pointer.
byte	The byte of data to transmit.

9.2.6.12 static uint8_t I2C_ReadByte (I2C_Type * *base*) [inline], [static]

In a master receive mode, calling this function initiates receiving the next byte of data.

Parameters

base	I2C base pointer.

Returns

This function returns the last byte received while the I2C module is configured in master receive or slave receive mode.

9.2.6.13 void I2C_SetIntCmd (I2C_Type * *base*, bool *enable*)

Parameters

base	I2C base pointer.
enable	Enable/Disbale I2C interrupt. • true: Enable I2C interrupt. • false: Disable I2C interrupt.

9.2.6.14 static uint32_t I2C_GetStatusFlag (I2C_Type * *base*, uint32_t *flags*) [inline], [static]

Parameters

base	I2C base pointer.
flags	I2C status flag mask (see _i2c_status_flag enumeration.)

Returns

I2C status, each bit represents one status flag

9.2.6.15 static void I2C_ClearStatusFlag (I2C_Type * *base*, uint32_t *flags*) [inline], [static]

Parameters

base	I2C base pointer.
flags	I2C status flag mask (see _i2c_status_flag enumeration.)

9.2.7 Variable Documentation

9.2.7.1 uint32_t i2c_init_config_t::clockRate

9.2.7.2 uint32_t i2c_init_config_t::baudRate

9.2.7.3 uint8_t i2c_init_config_t::slaveAddress

Chapter 10
Local Memory Controller (LMEM)

10.1 Overview

The FreeRTOS BSP provides a driver for the Local Memory Controller block of i.MX devices Cortex-M4 Core.

Modules

- LMEM driver

10.2 LMEM driver

10.2.1 Overview

The chapter describes the programming interface of the LMEM driver.

Cache Enable/Disable

Use LMEM_EnableSystemCache() / LMEM_DisableSystemCache() function to enable/disable System Cache or use LMEM_EnableCodeCache() / LMEM_DisableCodeCache() function to enable/disable Code Cache.

Cache Flush

Use a set of functions to flush the entire System/Code Cache or just flush several lines.

For System Cache flush, use LMEM_FlushSystemCache() and LMEM_FlushSystemCacheLines(). Similar functions are also provided for Code Cache flush.

Cache Invalidation

Use a set of functions to invalidate the entire System/Code Cache or just invalidate several lines.

For System Cache invalidation, use LMEM_InvalidateSystemCache() and LMEM_InvalidateSystemCacheLines(). Similar functions are also provided for Code Cache invalidation.

Processor System Cache control functions

- void LMEM_EnableSystemCache (LMEM_Type ∗base)
  This function enable the System Cache.
- void LMEM_DisableSystemCache (LMEM_Type ∗base)
  This function disable the System Cache.
- void LMEM_FlushSystemCache (LMEM_Type ∗base)
  This function flush the System Cache.
- void LMEM_FlushSystemCacheLines (LMEM_Type ∗base, void ∗address, uint32_t length)
  This function is called to flush the System Cache by performing cache copy-backs.
- void LMEM_InvalidateSystemCache (LMEM_Type ∗base)
  This function invalidate the System Cache.
- void LMEM_InvalidateSystemCacheLines (LMEM_Type ∗base, void ∗address, uint32_t length)
  This function is responsible for performing an System Cache invalidate.

Processor Code Cache control functions

- void LMEM_EnableCodeCache (LMEM_Type *base)
  This function enable the Code Cache.
- void LMEM_DisableCodeCache (LMEM_Type *base)
  This function disable the Code Cache.
- void LMEM_FlushCodeCache (LMEM_Type *base)
  This function flush the Code Cache.
- void LMEM_FlushCodeCacheLines (LMEM_Type *base, void *address, uint32_t length)
  This function is called to flush the Code Cache by performing cache copy-backs.
- void LMEM_InvalidateCodeCache (LMEM_Type *base)
  This function invalidate the Code Cache.
- void LMEM_InvalidateCodeCacheLines (LMEM_Type *base, void *address, uint32_t length)
  This function is responsible for performing an Code Cache invalidate.

10.2.2 Function Documentation

10.2.2.1 void LMEM_EnableSystemCache (LMEM_Type * *base*)

Parameters

base	LMEM base pointer.

10.2.2.2 void LMEM_DisableSystemCache (LMEM_Type * *base*)

Parameters

base	LMEM base pointer.

10.2.2.3 void LMEM_FlushSystemCache (LMEM_Type * *base*)

Parameters

base	LMEM base pointer.

10.2.2.4 void LMEM_FlushSystemCacheLines (LMEM_Type * *base,* void * *address,* uint32_t *length*)

```
It must determine how many cache lines need to be copied back and then
perform the copy-backs.
```

Parameters

base	LMEM base pointer.
address	The start address of cache line.
length	The length of flush address space.

10.2.2.5 void LMEM_InvalidateSystemCache (LMEM_Type * *base*)

Parameters

base	LMEM base pointer.

10.2.2.6 void LMEM_InvalidateSystemCacheLines (LMEM_Type * *base,* void * *address,* uint32_t *length*)

```
It must determine how many cache lines need to be invalidated and then
perform the invalidation.
```

Parameters

base	LMEM base pointer.
address	The start address of cache line.
length	The length of invalidate address space.

10.2.2.7 void LMEM_EnableCodeCache (LMEM_Type * *base*)

Parameters

base	LMEM base pointer.

10.2.2.8 void LMEM_DisableCodeCache (LMEM_Type * *base*)

Parameters

base	LMEM base pointer.

10.2.2.9 void LMEM_FlushCodeCache (LMEM_Type * *base*)

Parameters

base	LMEM base pointer.

10.2.2.10 void LMEM_FlushCodeCacheLines (LMEM_Type * *base*, void * *address*, uint32_t *length*)

```
It must determine how many cache lines need to be copied back and then
perform the copy-backs.
```

Parameters

base	LMEM base pointer.
address	The start address of cache line.
length	The length of flush address space.

10.2.2.11 void LMEM_InvalidateCodeCache (LMEM_Type * *base*)

Parameters

base	LMEM base pointer.

10.2.2.12 void LMEM_InvalidateCodeCacheLines (LMEM_Type * *base*, void * *address*, uint32_t *length*)

```
It must determine how many cache lines need to be invalidated and then
perform the invalidation.
```

Parameters

base	LMEM base pointer.
address	The start address of cache line.
length	The length of invalidate address space.

Chapter 11
Messaging Unit (MU)

11.1 Overview

The FreeRTOS BSP provides a driver for the Messaging Unit (MU) block of i.MX devices.

Modules

- MU driver

11.2 MU driver

11.2.1 Overview

This chapter describes the programming interface of the MU driver (platform/drivers/inc/mu_imx.h).

The MU driver provides these kinds of functions:

- Functions for send/receive message between processor A and B.
- Functions for general purpose interrupt between processor A and B.
- Functions for the flags between processor A and B.

11.2.2 Message send and receive functions

MU driver provides similar functions for message send and message receive. They are:

- Function to check whether the send/receive register is ready.
- Non-blocking function. Send/receive if the register is ready, otherwise return error status immediately.
- Blocking function. Wait until the send/receive register is ready, and send/receive the message.
- Function to enable/disable the RX/TX interrupt.

There are the functions:

```
// Functions for send message.
mu_status_t MU_TrySendMsg(MU_Type * base, uint32_t regIndex, uint32_t msg);
void MU_SendMsg(MU_Type * base, uint32_t regIndex, uint32_t msg);
bool MU_IsTxEmpty(MU_Type * base, uint32_t index);
void MU_EnableTxEmptyInt(MU_Type * base, uint32_t index);
void MU_DisableTxEmptyInt(MU_Type * base, uint32_t index);

// Functions for receive message.
mu_status_t MU_TryReceiveMsg(MU_Type * base, uint32_t regIndex, uint32_t *msg);
void MU_ReceiveMsg(MU_Type * base, uint32_t regIndex, uint32_t *msg);
bool MU_IsRxFull(MU_Type * base, uint32_t index);
void MU_EnableRxFullInt(MU_Type * base, uint32_t index);
void MU_DisableRxFullInt(MU_Type * base, uint32_t index);
```

11.2.3 General purpose interrupt functions

MU driver provides such functions for general purpose interrupt:

- Function to enable/disable the general purpose interrupt.
- Function to check and clear the general purpose interrupt pending status.
- Function to trigger general purpose interrupt to the other core.
- Function to check whether the general purpose interrupt has been processed by the other core.

The functions are:

```
void MU_EnableGeneralInt(MU_Type * base, uint32_t index);
void MU_DisableGeneralInt(MU_Type * base, uint32_t index);
bool MU_IsGeneralIntPending(MU_Type * base, uint32_t index);
```

```
void MU_ClearGeneralIntPending(MU_Type * base, uint32_t index);
mu_status_t MU_TriggerGeneralInt(MU_Type * base, uint32_t index);
bool MU_IsGeneralIntAccepted(MU_Type * base, uint32_t index);
```

Note that the enable/disable functions only control interrupt is issued or not. It means, if core B disables general purpose interrupt, and core A triggers the general purpose interrupt, then core B general purpose interrupt state is still pending, but it does not issue an interrupt.

11.2.4 Flag functions

By setting the flags on one side, the flags reflect on the other side, during the internal synchronization, it is not allowed to set flags again. Therefore, MU driver provides such functions for the MU flag:

```
mu_status_t MU_TrySetFlags(MU_Type * base, uint32_t flags);
void MU_SetFlags(MU_Type * base, uint32_t flags);
bool MU_IsFlagPending(MU_Type * base);
static inline uint32_t MU_GetFlags(MU_Type * base);
```

They are used for the non-blocking set, blocking set, pending status checking and flag check.

11.2.5 Other MU functions

MU driver provides functions for dual core boot up, clock, and power settings. Check the functions for details.

Macros

- #define MU_SR_GIP0_MASK (1U<<31U)
  Bit mask for general purpose interrupt 0 pending.
- #define MU_SR_RF0_MASK (1U<<27U)
  Bit mask for RX full interrupt 0 pending.
- #define MU_SR_TE0_MASK (1U<<23U)
  Bit mask for TX empty interrupt 0 pending.
- #define MU_CR_GIE0_MASK (1U<<31U)
  Bit mask for general purpose interrupt 0 enable.
- #define MU_CR_RIE0_MASK (1U<<27U)
  Bit mask for RX full interrupt 0 enable.
- #define MU_CR_TIE0_MASK (1U<<23U)
  Bit mask for TX empty interrupt 0 enable.
- #define MU_CR_GIR0_MASK (1U<<19U)
  Bit mask to trigger general purpose interrupt 0.
- #define MU_GPn_COUNT (4U)
  Number of general purpose interrupt.

Enumerations

- enum mu_status_t {
  kStatus_MU_Success = 0U,
  kStatus_MU_TxNotEmpty = 1U,
  kStatus_MU_RxNotFull = 2U,
  kStatus_MU_FlagPending = 3U,
  kStatus_MU_EventPending = 4U,
  kStatus_MU_Initialized = 5U,
  kStatus_MU_IntPending = 6U,
  kStatus_MU_Failed = 7U }
  MU status return codes.
- enum mu_msg_status_t {
  kMuTxEmpty0 = MU_SR_TE0_MASK,
  kMuTxEmpty1 = MU_SR_TE0_MASK >> 1U,
  kMuTxEmpty2 = MU_SR_TE0_MASK >> 2U,
  kMuTxEmpty3 = MU_SR_TE0_MASK >> 3U,
  kMuTxEmpty,
  kMuRxFull0 = MU_SR_RF0_MASK,
  kMuRxFull1 = MU_SR_RF0_MASK >> 1U,
  kMuRxFull2 = MU_SR_RF0_MASK >> 2U,
  kMuRxFull3 = MU_SR_RF0_MASK >> 3U,
  kMuRxFull,
  kMuGenInt0 = MU_SR_GIP0_MASK,
  kMuGenInt1 = MU_SR_GIP0_MASK >> 1U,
  kMuGenInt2 = MU_SR_GIP0_MASK >> 2U,
  kMuGenInt3 = MU_SR_GIP0_MASK >> 3U,
  kMuGenInt,
  kMuStatusAll }
  MU message status.
- enum mu_power_mode_t {
  kMuPowerModeRun = 0x00U,
  kMuPowerModeWait = 0x01U,
  kMuPowerModeStop = 0x02U,
  kMuPowerModeDsm = 0x03U }
  Power mode definition.

Initialization.

- static void MU_Init (MU_Type *base)
  Initializes the MU module to reset state.

Send Messages.

- mu_status_t MU_TrySendMsg (MU_Type *base, uint32_t regIndex, uint32_t msg)
  Try to send a message.
- void MU_SendMsg (MU_Type *base, uint32_t regIndex, uint32_t msg)
  Block to send a message.
- static bool MU_IsTxEmpty (MU_Type *base, uint32_t index)
  Check TX empty status.
- static void MU_EnableTxEmptyInt (MU_Type *base, uint32_t index)
  Enable TX empty interrupt.
- static void MU_DisableTxEmptyInt (MU_Type *base, uint32_t index)
  Disable TX empty interrupt.

Receive Messages.

- mu_status_t MU_TryReceiveMsg (MU_Type *base, uint32_t regIndex, uint32_t *msg)
  Try to receive a message.
- void MU_ReceiveMsg (MU_Type *base, uint32_t regIndex, uint32_t *msg)
  Block to receive a message.
- static bool MU_IsRxFull (MU_Type *base, uint32_t index)
  Check RX full status.
- static void MU_EnableRxFullInt (MU_Type *base, uint32_t index)
  Enable RX full interrupt.
- static void MU_DisableRxFullInt (MU_Type *base, uint32_t index)
  Disable RX full interrupt.

General Purpose Interrupt.

- static void MU_EnableGeneralInt (MU_Type *base, uint32_t index)
  Enable general purpose interrupt.
- static void MU_DisableGeneralInt (MU_Type *base, uint32_t index)
  Disable general purpose interrupt.
- static bool MU_IsGeneralIntPending (MU_Type *base, uint32_t index)
  Check specific general purpose interrupt pending flag.
- static void MU_ClearGeneralIntPending (MU_Type *base, uint32_t index)
  Clear specific general purpose interrupt pending flag.
- mu_status_t MU_TriggerGeneralInt (MU_Type *base, uint32_t index)
  Trigger specific general purpose interrupt.
- static bool MU_IsGeneralIntAccepted (MU_Type *base, uint32_t index)
  Check specific general purpose interrupt is accepted or not.

Flags

- mu_status_t MU_TrySetFlags (MU_Type *base, uint32_t flags)
  Try to set some bits of the 3-bit flag reflect on the other MU side.
- void MU_SetFlags (MU_Type *base, uint32_t flags)
  Set some bits of the 3-bit flag reflect on the other MU side.

- static bool MU_IsFlagPending (MU_Type *base)
  Checks whether the previous flag update is pending.
- static uint32_t MU_GetFlags (MU_Type *base)
  Get the current value of the 3-bit flag set by other side.

Misc.

- static mu_power_mode_t MU_GetOtherCorePowerMode (MU_Type *base)
  Get the power mode of the other core.
- static bool MU_IsEventPending (MU_Type *base)
  Get the event pending status.
- static uint32_t MU_GetMsgStatus (MU_Type *base, uint32_t statusToCheck)
  Get the the MU message status.

11.2.6 Macro Definition Documentation

11.2.6.1 #define MU_SR_GIP0_MASK (1U<<31U)

11.2.6.2 #define MU_SR_RF0_MASK (1U<<27U)

11.2.6.3 #define MU_SR_TE0_MASK (1U<<23U)

11.2.6.4 #define MU_CR_GIE0_MASK (1U<<31U)

11.2.6.5 #define MU_CR_RIE0_MASK (1U<<27U)

11.2.6.6 #define MU_CR_TIE0_MASK (1U<<23U)

11.2.6.7 #define MU_CR_GIR0_MASK (1U<<19U)

11.2.6.8 #define MU_GPn_COUNT (4U)

11.2.7 Enumeration Type Documentation

11.2.7.1 enum mu_status_t

Enumerator

kStatus_MU_Success Success.
kStatus_MU_TxNotEmpty TX register is not empty.
kStatus_MU_RxNotFull RX register is not full.
kStatus_MU_FlagPending Previous flags update pending.
kStatus_MU_EventPending MU event is pending.
kStatus_MU_Initialized MU driver has initialized previously.

kStatus_MU_IntPending Previous general interrupt still pending.
kStatus_MU_Failed Execution failed.

11.2.7.2 enum mu_msg_status_t

Enumerator

kMuTxEmpty0 TX0 empty status.
kMuTxEmpty1 TX1 empty status.
kMuTxEmpty2 TX2 empty status.
kMuTxEmpty3 TX3 empty status.
kMuTxEmpty TX empty status.
kMuRxFull0 RX0 full status.
kMuRxFull1 RX1 full status.
kMuRxFull2 RX2 full status.
kMuRxFull3 RX3 full status.
kMuRxFull RX empty status.
kMuGenInt0 General purpose interrupt 0 pending status.
kMuGenInt1 General purpose interrupt 2 pending status.
kMuGenInt2 General purpose interrupt 2 pending status.
kMuGenInt3 General purpose interrupt 3 pending status.
kMuGenInt General purpose interrupt pending status.
kMuStatusAll All MU status.

11.2.7.3 enum mu_power_mode_t

Enumerator

kMuPowerModeRun Run mode.
kMuPowerModeWait WAIT mode.
kMuPowerModeStop STOP mode.
kMuPowerModeDsm DSM mode.

11.2.8 Function Documentation

11.2.8.1 static void MU_Init (MU_Type * *base*) [inline], [static]

This function sets the MU module control register to its default reset value.

Parameters

base	Register base address for the module.

11.2.8.2 mu_status_t MU_TrySendMsg (MU_Type * *base*, uint32_t *regIndex*, uint32_t *msg*)

This function tries to send a message, if the TX register is not empty, this function returns kStatus_MU_-TxNotEmpty.

Parameters

base	Register base address for the module.
regIdex	Tx register index.
msg	Message to send.

Return values

kStatus_MU_Success	Message send successfully.
kStatus_MU_TxNotEmpty	Message not send because TX is not empty.

11.2.8.3 void MU_SendMsg (MU_Type * *base*, uint32_t *regIndex*, uint32_t *msg*)

This function waits until TX register is empty and send the message.

Parameters

base	Register base address for the module.
regIdex	Tx register index.
msg	Message to send.

11.2.8.4 static bool MU_IsTxEmpty (MU_Type * *base*, uint32_t *index*) [inline], [static]

This function checks the specific transmit register empty status.

Parameters

base	Register base address for the module.
index	TX register index to check.

Return values

true	TX register is empty.
false	TX register is not empty.

11.2.8.5 static void MU_EnableTxEmptyInt (MU_Type * *base*, uint32_t *index*) [inline], [static]

This function enables specific TX empty interrupt.

Parameters

base	Register base address for the module.
index	TX interrupt index to enable.

Example:

```
// To enable TX0 empty interrupts.
MU_EnableTxEmptyInt(MU0_BASE, 0U);
```

11.2.8.6 static void MU_DisableTxEmptyInt (MU_Type * *base*, uint32_t *index*) [inline], [static]

This function disables specific TX empty interrupt.

Parameters

base	Register base address for the module.
disableMask	Bitmap of the interrupts to disable.

Example:

```
// To disable TX0 empty interrupts.
MU_DisableTxEmptyInt(MU0_BASE, 0U);
```

11.2.8.7 mu_status_t MU_TryReceiveMsg (MU_Type * *base*, uint32_t *regIndex*, uint32_t * *msg*)

This function tries to receive a message, if the RX register is not full, this function returns kStatus_MU_-RxNotFull.

Parameters

base	Register base address for the module.
regIdex	Rx register index.
msg	Message to receive.

Return values

kStatus_MU_Success	Message receive successfully.
kStatus_MU_RxNotFull	Message not received because RX is not full.

11.2.8.8 void MU_ReceiveMsg (MU_Type * *base,* uint32_t *regIndex,* uint32_t * *msg*)

This function waits until RX register is full and receive the message.

Parameters

base	Register base address for the module.
regIdex	Rx register index.
msg	Message to receive.

11.2.8.9 static bool MU_IsRxFull (MU_Type * *base,* uint32_t *index*) [inline], [static]

This function checks the specific receive register full status.

Parameters

base	Register base address for the module.
index	RX register index to check.

Return values

true	RX register is full.
false	RX register is not full.

11.2.8.10 static void MU_EnableRxFullInt (MU_Type * *base,* uint32_t *index*) [inline], [static]

This function enables specific RX full interrupt.

Parameters

base	Register base address for the module.
index	RX interrupt index to enable.

Example:

```
// To enable RX0 full interrupts.
MU_EnableRxFullInt(MU0_BASE, 0U);
```

11.2.8.11 static void MU_DisableRxFullInt (MU_Type * *base*, uint32_t *index*) [inline], [static]

This function disables specific RX full interrupt.

Parameters

base	Register base address for the module.
disableMask	Bitmap of the interrupts to disable.

Example:

```
// To disable RX0 full interrupts.
MU_DisableRxFullInt(MU0_BASE, 0U);
```

11.2.8.12 static void MU_EnableGeneralInt (MU_Type * *base*, uint32_t *index*) [inline], [static]

This function enables specific general purpose interrupt.

Parameters

base	Register base address for the module.
index	General purpose interrupt index to enable.

Example:

```
// To enable general purpose interrupts 0.
MU_EnableGeneralInt(MU0_BASE, 0U);
```

11.2.8.13 static void MU_DisableGeneralInt (MU_Type * *base*, uint32_t *index*) [inline], [static]

This function disables specific general purpose interrupt.

Parameters

base	Register base address for the module.
index	General purpose interrupt index to disable.

Example:

```
// To disable general purpose interrupts 0.
MU_DisableGeneralInt(MU0_BASE, 0U);
```

11.2.8.14 static bool MU_IsGeneralIntPending (MU_Type * *base*, uint32_t *index*) [inline], [static]

This function checks the specific general purpose interrupt pending status.

Parameters

base	Register base address for the module.
index	Index of the general purpose interrupt flag to check.

Return values

true	General purpose interrupt is pending.
false	General purpose interrupt is not pending.

11.2.8.15 static void MU_ClearGeneralIntPending (MU_Type * *base*, uint32_t *index*) [inline], [static]

This function clears the specific general purpose interrupt pending status.

Parameters

base	Register base address for the module.
index	Index of the general purpose interrupt flag to clear.

11.2.8.16 mu_status_t MU_TriggerGeneralInt (MU_Type * *base*, uint32_t *index*)

This function triggers specific general purpose interrupt to other core.

To ensure proper operations, make sure the correspond general purpose interrupt triggered previously has been accepted by the other core. The function MU_IsGeneralIntAccepted can be used for this check. If the previous general interrupt has not been accepted by the other core, this function does not trigger interrupt actually and returns an error.

Parameters

base	Register base address for the module.
index	Index of general purpose interrupt to trigger.

Return values

kStatus_MU_Success	Interrupt has been triggered successfully.
kStatus_MU_IntPending	Previous interrupt has not been accepted.

11.2.8.17 static bool MU_IsGeneralIntAccepted (MU_Type * *base*, uint32_t *index*) [inline], [static]

This function checks whether the specific general purpose interrupt has been accepted by the other core or not.

Parameters

base	Register base address for the module.
index	Index of the general purpose interrupt to check.

Return values

true	General purpose interrupt is accepted.
false	General purpose interrupt is not accepted.

11.2.8.18 mu_status_t MU_TrySetFlags (MU_Type * *base*, uint32_t *flags*)

This functions tries to set some bits of the 3-bit flag. If previous flags update is still pending, this function returns kStatus_MU_FlagPending.

Parameters

base	Register base address for the module.

Return values

kStatus_MU_Success	Flag set successfully.
kStatus_MU_Flag-Pending	Previous flag update is pending.

11.2.8.19 void MU_SetFlags (MU_Type * *base*, uint32_t *flags*)

This functions set some bits of the 3-bit flag. If previous flags update is still pending, this function blocks and polls to set the flag.

Parameters

base	Register base address for the module.

11.2.8.20 static bool MU_IsFlagPending (MU_Type * *base*) [inline], [static]

After setting flags, the flags update request is pending until internally acknowledged. During the pending period, it is not allowed to set flags again. This function is used to check the pending status, it can be used together with function MU_TrySetFlags.

Parameters

base	Register base address for the module.

Returns

True if pending, false if not.

11.2.8.21 static uint32_t MU_GetFlags (MU_Type * *base*) [inline], [static]

This functions gets the current value of the 3-bit flag.

Parameters

base	Register base address for the module.

Returns

flags Current value of the 3-bit flag.

11.2.8.22 static mu_power_mode_t MU_GetOtherCorePowerMode (MU_Type * *base*) [inline], [static]

This functions gets the power mode of the other core.

Parameters

base	Register base address for the module.

Returns

powermode Power mode of the other core.

11.2.8.23 static bool MU_IsEventPending (MU_Type * *base*) [inline], [static]

This functions gets the event pending status. To ensure events have been posted to the other side before entering STOP mode, verify the event pending status using this function.

Parameters

base	Register base address for the module.

Return values

true	Event is pending.
false	Event is not pending.

11.2.8.24 static uint32_t MU_GetMsgStatus (MU_Type * *base*, uint32_t *statusToCheck*) [inline], [static]

This functions gets TX/RX and general purpose interrupt pending status. The parameter is passed in as bitmask of the status to check.

Parameters

base	Register base address for the module.
statusToCheck	The status to check, see mu_msg_status_t.

Returns

Status checked.

Example:

```
// To check TX0 empty status.
MU_GetMsgStatus(MU0_BASE, kMuTxEmpty0);

// To check all RX full status.
MU_GetMsgStatus(MU0_BASE, kMuRxFull);
```

```
// To check general purpose interrupt 0 and 3 pending status.
MU_GetMsgStatus(MU0_BASE, kMuGenInt0 | kMuGenInt3);

// To check all status.
MU_GetMsgStatus(MU0_BASE, kMuStatusAll);
```

Chapter 12
Resource Domain Controller (RDC)

12.1 Overview

The FreeRTOS BSP provides a driver for the Resource Domain Controller (RDC) block of i.MX devices.

Modules

- RDC Semaphore driver
- RDC definitions on i.MX 6SoloX
- RDC driver

12.2 RDC driver

12.2.1 Overview

The chapter describes the programming interface of the RDC driver (platform/drivers/inc/rdc.h). The RDC provides robust support for isolation of peripherals and memory among different bus masters. The RDC driver provides a set of APIs to provide these services:

- RDC domain control
- RDC status control

12.2.2 RDC domain control

RDC defines "domain", which is the unit of the access control. One or more bus masters can be put into one domain to get the permission from all the domains to access peripherals or memory. Each bus master is allocated a unique RDC master ID called "MDA", and the user can find the MDA enumeration *rdc_mda in rdc_defs*<device>.h, where <device> specifies the i.MX device name. Normally i.MX devices have 4 domains with ID from 0 to 3, and RDC_SetDomainID() can be used to put some MDA into one of those domains. To avoid malfunction, the setting can be locked with "lock" parameter. RDC_GetDomainID() is used to check which domain is some MDA associated.

There are some functions related to peripheral access permission for certain domain. Just like MDA, each peripheral has a RDC peripheral ID called "PDAP", which is also defined in rdc_defs_<device>.h. RDC_SetPdapAccess() is to set PDAP permission for each domain, and "lock" parameter is also available to avoid further change of this setting. If some peripheral need to be accessed by multiple MDA, access conflict must be resolved. RDC provides RDC SEMAPHORE to allow each MDA to access the peripheral exclusively. And when using RDC_SetPdapAccess(), the user can also force RDC SEMAPHORE being acquired before peripheral access. Parameter "sreq" is for this purpose. RDC_GetPdapAccess() and RDC_IsPdapSemaphoreRequired() can be used to check current setting of some PDAP.

Besides peripheral access, memory can also be protected by RDC. Here memory can be QSPI, DDR, OCRAM, PCIE, and so on. Each type of memory can have several regions in RDC, with different access permission for each region. RDC memory region setting must be enabled before it take effects. RDC_SetMrAccess() is used for memory region access permission setting, and RDC memory region (defined in rdc_defs_<device>.h) type has to be matched with the [startAddr, endAddr) area. RDC_GetMrAccess() and RDC_IsMrEnabled() are used to check current setting of some memory region. In additional, the user can also use RDC_GetViolationStatus() to get the memory region's violation address and which domain causes this violation. Once violation occurs and gets properly handled, RDC_ClearViolationStatus() is used to clear the violation status.

12.2.3 RDC status control

RDC provides a interrupt that indicates the memory region setting restoration has completed. This is useful when some memory region as well as the RDC memory region configuration is powered off in low power mode. The interrupt can guarantee the memory and memory access configuration work before

the bus master accesses this region. RDC_IsMemPowered(), RDC_IsIntPending() and RDC_ClearStatusFlag() are functions for low power recovery. RDC_GetSelfDomainID() is used to return the domain ID of running CPU.

RDC State Control

- static uint32_t RDC_GetSelfDomainID (RDC_Type *base)
  Get domain ID of core that is reading this.
- static bool RDC_IsMemPowered (RDC_Type *base)
  Check whether memory region controlled by RDC is accessible after low power recovery.
- static bool RDC_IsIntPending (RDC_Type *base)
  Check whether there's pending RDC memory region restoration interrupt.
- static void RDC_ClearStatusFlag (RDC_Type *base)
  Clear interrupt status.
- static void RDC_SetIntCmd (RDC_Type *base, bool enable)
  Set RDC interrupt mode.

RDC Domain Control

- static void RDC_SetDomainID (RDC_Type *base, uint32_t mda, uint32_t domainId, bool lock)
  Set RDC domain ID for RDC master.
- static uint32_t RDC_GetDomainID (RDC_Type *base, uint32_t mda)
  Get RDC domain ID for RDC master.
- static void RDC_SetPdapAccess (RDC_Type *base, uint32_t pdap, uint8_t perm, bool sreq, bool lock)
  Set RDC peripheral access permission for RDC domains.
- static uint8_t RDC_GetPdapAccess (RDC_Type *base, uint32_t pdap)
  Get RDC peripheral access permission for RDC domains.
- static bool RDC_IsPdapSemaphoreRequired (RDC_Type *base, uint32_t pdap)
  Check whether RDC semaphore is required to access the peripheral.
- void RDC_SetMrAccess (RDC_Type *base, uint32_t mr, uint32_t startAddr, uint32_t endAddr, uint8_t perm, bool enable, bool lock)
  Set RDC memory region access permission for RDC domains.
- uint8_t RDC_GetMrAccess (RDC_Type *base, uint32_t mr, uint32_t *startAddr, uint32_t *endAddr)
  Get RDC memory region access permission for RDC domains.
- static bool RDC_IsMrEnabled (RDC_Type *base, uint32_t mr)
  Check whether the memory region is enabled.
- bool RDC_GetViolationStatus (RDC_Type *base, uint32_t mr, uint32_t *violationAddr, uint32_t *violationDomain)
  Get memory violation status.
- static void RDC_ClearViolationStatus (RDC_Type *base, uint32_t mr)
  Clear RDC violation status.

12.2.4 Function Documentation

12.2.4.1 static uint32_t RDC_GetSelfDomainID (RDC_Type * *base*) [inline], [static]

Parameters

base	RDC base pointer.

Returns

Domain ID of self core

12.2.4.2 static bool RDC_IsMemPowered (RDC_Type * *base*) [inline], [static]

Parameters

base	RDC base pointer.

Returns

Memory region power status.
- true: on and accessible.
- false: off.

12.2.4.3 static bool RDC_IsIntPending (RDC_Type * *base*) [inline], [static]

Parameters

base	RDC base pointer.

Returns

RDC interrupt status
- true: Interrupt pending.
- false: No interrupt pending.

12.2.4.4 static void RDC_ClearStatusFlag (RDC_Type * *base*) [inline], [static]

Parameters

base	RDC base pointer.

12.2.4.5 static void RDC_SetIntCmd (RDC_Type * *base,* bool *enable*) [inline], [static]

Parameters

base	RDC base pointer
enable	RDC interrupt control. • true: enable interrupt. • false: disable interrupt.

12.2.4.6 static void RDC_SetDomainID (RDC_Type * *base,* uint32_t *mda,* uint32_t *domainId,* bool *lock*) [inline], [static]

Parameters

base	RDC base pointer
mda	RDC master assignment (see *rdc_mda in rdc_defs*<device>.h)
domainId	RDC domain ID (0-3)
lock	Whether to lock this setting? Once locked, no one can change the domain assignment until reset

12.2.4.7 static uint32_t RDC_GetDomainID (RDC_Type * *base,* uint32_t *mda*) [inline], [static]

Parameters

base	RDC base pointer
mda	RDC master assignment (see *rdc_mda in rdc_defs*<device>.h)

Returns

RDC domain ID (0-3)

12.2.4.8 static void RDC_SetPdapAccess (RDC_Type * *base,* uint32_t *pdap,* uint8_t *perm,* bool *sreq,* bool *lock*) [inline], [static]

Parameters

base	RDC base pointer
pdap	RDC peripheral assignment (see *rdc_pdap in rdc_defs*<device>.h)
perm	RDC access permission from RDC domain to peripheral (byte: D3R D3W D2R D2W D1R D1W D0R D0W)
sreq	Force acquiring SEMA42 to access this peripheral or not
lock	Whether to lock this setting or not. Once locked, no one can change the RDC setting until reset

12.2.4.9 static uint8_t RDC_GetPdapAccess (RDC_Type * *base*, uint32_t *pdap*) [inline], [static]

Parameters

base	RDC base pointer
pdap	RDC peripheral assignment (see *rdc_pdap in rdc_defs*<device>.h)

Returns

RDC access permission from RDC domain to peripheral (byte: D3R D3W D2R D2W D1R D1W D0R D0W)

12.2.4.10 static bool RDC_IsPdapSemaphoreRequired (RDC_Type * *base*, uint32_t *pdap*) [inline], [static]

Parameters

base	RDC base pointer
pdap	RDC peripheral assignment (see *rdc_pdap in rdc_defs*<device>.h)

Returns

RDC semaphore required or not.
- true: RDC semaphore is required.
- false: RDC semaphore is not required.

12.2.4.11 void RDC_SetMrAccess (RDC_Type * *base*, uint32_t *mr*, uint32_t *startAddr*, uint32_t *endAddr*, uint8_t *perm*, bool *enable*, bool *lock*)

Parameters

base	RDC base pointer
mr	RDC memory region assignment (see *rdc_mr in rdc_defs*<device>.h)
startAddr	memory region start address (inclusive)
endAddr	memory region end address (exclusive)
perm	RDC access permission from RDC domain to peripheral (byte: D3R D3W D2R D2W D1R D1W D0R D0W)
enable	Enable this memory region for RDC control or not
lock	Whether to lock this setting or not. Once locked, no one can change the RDC setting until reset

12.2.4.12 uint8_t RDC_GetMrAccess (RDC_Type * *base*, uint32_t *mr*, uint32_t * *startAddr*, uint32_t * *endAddr*)

Parameters

base	RDC base pointer
mr	RDC memory region assignment (see *rdc_mr in rdc_defs*<device>.h)
startAddr	pointer to get memory region start address (inclusive), NULL is allowed.
endAddr	pointer to get memory region end address (exclusive), NULL is allowed.

Returns

RDC access permission from RDC domain to peripheral (byte: D3R D3W D2R D2W D1R D1W D0R D0W)

12.2.4.13 static bool RDC_IsMrEnabled (RDC_Type * *base*, uint32_t *mr*) [inline], [static]

Parameters

base	RDC base pointer

mr	RDC memory region assignment (see *rdc_mr in rdc_defs*<device>.h)

Returns

Memory region enabled or not.
- true: Memory region is enabled.
- false: Memory region is not enabled.

12.2.4.14 bool RDC_GetViolationStatus (RDC_Type * *base,* uint32_t *mr,* uint32_t * *violationAddr,* uint32_t * *violationDomain*)

Parameters

base	RDC base pointer
mr	RDC memory region assignment (see *rdc_mr in rdc_defs*<device>.h)
violationAddr	Pointer to store violation address, NULL allowed
violation-Domain	Pointer to store domain ID causing violation, NULL allowed

Returns

Memory violation occurred or not.
- true: violation happened.
- false: No violation happened.

12.2.4.15 static void RDC_ClearViolationStatus (RDC_Type * *base,* uint32_t *mr*) [inline], [static]

Parameters

base	RDC base pointer
mr	RDC memory region assignment (see *rdc_mr in rdc_defs*<device>.h)

12.3 RDC definitions on i.MX 6SoloX

12.3.1 Overview

The chapter describes the RDC MDA, PDAP and Memory Region definitions (platform/drivers/inc/rdc_-defs_imx6sx.h).

Enumerations

- enum _rdc_mda {
  rdcMdaA9L2Cache = 0U,
  rdcMdaM4 = 1U,
  rdcMdaGpu = 2U,
  rdcMdaCsi1 = 3U,
  rdcMdaCsi2 = 4U,
  rdcMdaLcdif1 = 5U,
  rdcMdaLcdif2 = 6U,
  rdcMdaPxp = 7U,
  rdcMdaPcieCtrl = 8U,
  rdcMdaDap = 9U,
  rdcMdaCaam = 10U,
  rdcMdaSdmaPeriph = 11U,
  rdcMdaSdmaBurst = 12U,
  rdcMdaApbhdma = 13U,
  rdcMdaRawnand = 14U,
  rdcMdaUsdhc1 = 15U,
  rdcMdaUsdhc2 = 16U,
  rdcMdaUsdhc3 = 17U,
  rdcMdaUsdhc4 = 18U,
  rdcMdaUsb = 19U,
  rdcMdaMlb = 20U,
  rdcMdaTestPort = 21U,
  rdcMdaEnet1Tx = 22U,
  rdcMdaEnet1Rx = 23U,
  rdcMdaEnet2Tx = 24U,
  rdcMdaEnet2Rx = 25U,
  rdcMdaSdmaPort = 26U }
  RDC master assignment.
- enum _rdc_pdap {

rdcPdapPwm1 = 0U,
rdcPdapPwm2 = 1U,
rdcPdapPwm3 = 2U,
rdcPdapPwm4 = 3U,
rdcPdapCan1 = 4U,
rdcPdapCan2 = 5U,
rdcPdapGpt = 6U,
rdcPdapGpio1 = 7U,
rdcPdapGpio2 = 8U,
rdcPdapGpio3 = 9U,
rdcPdapGpio4 = 10U,
rdcPdapGpio5 = 11U,
rdcPdapGpio6 = 12U,
rdcPdapGpio7 = 13U,
rdcPdapKpp = 14U,
rdcPdapWdog1 = 15U,
rdcPdapWdog2 = 16U,
rdcPdapCcm = 17U,
rdcPdapAnatopDig = 18U,
rdcPdapSnvsHp = 19U,
rdcPdapEpit1 = 20U,
rdcPdapEpit2 = 21U,
rdcPdapSrc = 22U,
rdcPdapGpc = 23U,
rdcPdapIomuxc = 24U,
rdcPdapIomuxcGpr = 25U,
rdcPdapCanfdCan1 = 26U,
rdcPdapSdma = 27U,
rdcPdapCanfdCan2 = 28U,
rdcPdapRdcSema421 = 29U,
rdcPdapRdcSema422 = 30U,
rdcPdapRdc = 31U,
rdcPdapAipsTz1GlobalEnable1 = 32U,
rdcPdapAipsTz1GlobalEnable2 = 33U,
rdcPdapUsb02hPl301 = 34U,
rdcPdapUsb02hUsb = 35U,
rdcPdapEnet1 = 36U,
rdcPdapMlb2550 = 37U,
rdcPdapUsdhc1 = 38U,
rdcPdapUsdhc2 = 39U,
rdcPdapUsdhc3 = 40U,
rdcPdapUsdhc4 = 41U,
rdcPdapI2c1 = 42U,
rdcPdapI2c2 = 43U,
rdcPdapI2c3 = 44U,
rdcPdapRomcp = 45U,
rdcPdapMmdc = 46U,
rdcPdapEnet2 = 47U,
rdcPdapEim = 48U,

rdcPdapSpDisplaymix = 109U }
RDC peripheral assignment.

- enum _rdc_mr {
rdcMrMmdc = 0U,
rdcMrMmdcLast = 7U,
rdcMrQspi1 = 8U,
rdcMrQspi1Last = 15U,
rdcMrQspi2 = 16U,
rdcMrQspi2Last = 23U,
rdcMrWeim = 24U,
rdcMrWeimLast = 31U,
rdcMrPcie = 32U,
rdcMrPcieLast = 39U,
rdcMrOcram = 40U,
rdcMrOcramLast = 44U,
rdcMrOcramS = 45U,
rdcMrOcramSLast = 49U,
rdcMrOcramL2 = 50U,
rdcMrOcramL2Last = 54U }
RDC memory region.

12.3.2 Enumeration Type Documentation

12.3.2.1 enum _rdc_mda

Enumerator

rdcMdaA9L2Cache A9 L2 Cache RDC Master.
rdcMdaM4 M4 RDC Master.
rdcMdaGpu GPU RDC Master.
rdcMdaCsi1 Csi1 RDC Master.
rdcMdaCsi2 Csi2 RDC Master.
rdcMdaLcdif1 Lcdif1 RDC Master.
rdcMdaLcdif2 Lcdif2 RDC Master.
rdcMdaPxp Pxp RDC Master.
rdcMdaPcieCtrl Pcie Ctrl RDC Master.
rdcMdaDap Dap RDC Master.
rdcMdaCaam Caam RDC Master.
rdcMdaSdmaPeriph Sdma Periph RDC Master.
rdcMdaSdmaBurst Sdma Burst RDC Master.
rdcMdaApbhdma Apbhdma RDC Master.
rdcMdaRawnand Rawnand RDC Master.
rdcMdaUsdhc1 Usdhc1 RDC Master.
rdcMdaUsdhc2 Usdhc2 RDC Master.
rdcMdaUsdhc3 Usdhc3 RDC Master.

rdcMdaUsdhc4 Usdhc4 RDC Master.
rdcMdaUsb USB RDC Master.
rdcMdaMlb MLB RDC Master.
rdcMdaTestPort Test Port RDC Master.
rdcMdaEnet1Tx Enet1 Tx RDC Master.
rdcMdaEnet1Rx Enet1 Rx Master.
rdcMdaEnet2Tx Enet2 Tx RDC Master.
rdcMdaEnet2Rx Enet2 Rx RDC Master.
rdcMdaSdmaPort Sdma Port RDC Master.

12.3.2.2 enum _rdc_pdap

Enumerator

rdcPdapPwm1 Pwm1 RDC Peripheral.
rdcPdapPwm2 Pwm2 RDC Peripheral.
rdcPdapPwm3 Pwm3 RDC Peripheral.
rdcPdapPwm4 Pwm4 RDC Peripheral.
rdcPdapCan1 Can1 RDC Peripheral.
rdcPdapCan2 Can2 RDC Peripheral.
rdcPdapGpt Gpt RDC Peripheral.
rdcPdapGpio1 Gpio1 RDC Peripheral.
rdcPdapGpio2 Gpio2 RDC Peripheral.
rdcPdapGpio3 Gpio3 RDC Peripheral.
rdcPdapGpio4 Gpio4 RDC Peripheral.
rdcPdapGpio5 Gpio5 RDC Peripheral.
rdcPdapGpio6 Gpio6 RDC Peripheral.
rdcPdapGpio7 Gpio7 RDC Peripheral.
rdcPdapKpp Kpp RDC Peripheral.
rdcPdapWdog1 Wdog1 RDC Peripheral.
rdcPdapWdog2 Wdog2 RDC Peripheral.
rdcPdapCcm Ccm RDC Peripheral.
rdcPdapAnatopDig AnatopDig RDC Peripheral.
rdcPdapSnvsHp SnvsHp RDC Peripheral.
rdcPdapEpit1 Epit1 RDC Peripheral.
rdcPdapEpit2 Epit2 RDC Peripheral.
rdcPdapSrc Src RDC Peripheral.
rdcPdapGpc Gpc RDC Peripheral.
rdcPdapIomuxc Iomuxc RDC Peripheral.
rdcPdapIomuxcGpr IomuxcGpr RDC Peripheral.
rdcPdapCanfdCan1 Canfd Can1 RDC Peripheral.
rdcPdapSdma Sdma RDC Peripheral.
rdcPdapCanfdCan2 Canfd Can2 RDC Peripheral.
rdcPdapRdcSema421 Rdc Sema421 RDC Peripheral.

rdcPdapRdcSema422 Rdc Sema422 RDC Peripheral.
rdcPdapRdc Rdc RDC Peripheral.
rdcPdapAipsTz1GlobalEnable1 AipsTz1GlobalEnable1 RDC Peripheral.
rdcPdapAipsTz1GlobalEnable2 AipsTz1GlobalEnable2 RDC Peripheral.
rdcPdapUsb02hPl301 Usb02hPl301 RDC Peripheral.
rdcPdapUsb02hUsb Usb02hUsb RDC Peripheral.
rdcPdapEnet1 Enet1 RDC Peripheral.
rdcPdapMlb2550 Mlb2550 RDC Peripheral.
rdcPdapUsdhc1 Usdhc1 RDC Peripheral.
rdcPdapUsdhc2 Usdhc2 RDC Peripheral.
rdcPdapUsdhc3 Usdhc3 RDC Peripheral.
rdcPdapUsdhc4 Usdhc4 RDC Peripheral.
rdcPdapI2c1 I2c1 RDC Peripheral.
rdcPdapI2c2 I2c2 RDC Peripheral.
rdcPdapI2c3 I2c3 RDC Peripheral.
rdcPdapRomcp Romcp RDC Peripheral.
rdcPdapMmdc Mmdc RDC Peripheral.
rdcPdapEnet2 Enet2 RDC Peripheral.
rdcPdapEim Eim RDC Peripheral.
rdcPdapOcotpCtrlWrapper OcotpCtrlWrapper RDC Peripheral.
rdcPdapCsu Csu RDC Peripheral.
rdcPdapPerfmon1 Perfmon1 RDC Peripheral.
rdcPdapPerfmon2 Perfmon2 RDC Peripheral.
rdcPdapAxiMon AxiMon RDC Peripheral.
rdcPdapTzasc1 Tzasc1 RDC Peripheral.
rdcPdapSai1 Sai1 RDC Peripheral.
rdcPdapAudmux Audmux RDC Peripheral.
rdcPdapSai2 Sai2 RDC Peripheral.
rdcPdapQspi1 Qspi1 RDC Peripheral.
rdcPdapQspi2 Qspi2 RDC Peripheral.
rdcPdapUart2 Uart2 RDC Peripheral.
rdcPdapUart3 Uart3 RDC Peripheral.
rdcPdapUart4 Uart4 RDC Peripheral.
rdcPdapUart5 Uart5 RDC Peripheral.
rdcPdapI2c4 I2c4 RDC Peripheral.
rdcPdapQosc Qosc RDC Peripheral.
rdcPdapCaam Caam RDC Peripheral.
rdcPdapDap Dap RDC Peripheral.
rdcPdapAdc1 Adc1 RDC Peripheral.
rdcPdapAdc2 Adc2 RDC Peripheral.
rdcPdapWdog3 Wdog3 RDC Peripheral.
rdcPdapEcspi5 Ecspi5 RDC Peripheral.
rdcPdapSema4 Sema4 RDC Peripheral.
rdcPdapMuA MuA RDC Peripheral.
rdcPdapCanfdCpu Canfd Cpu RDC Peripheral.

rdcPdapMuB MuB RDC Peripheral.
rdcPdapUart6 Uart6 RDC Peripheral.
rdcPdapPwm5 Pwm5 RDC Peripheral.
rdcPdapPwm6 Pwm6 RDC Peripheral.
rdcPdapPwm7 Pwm7 RDC Peripheral.
rdcPdapPwm8 Pwm8 RDC Peripheral.
rdcPdapAipsTz3GlobalEnable0 AipsTz3GlobalEnable0 RDC Peripheral.
rdcPdapAipsTz3GlobalEnable1 AipsTz3GlobalEnable1 RDC Peripheral.
rdcPdapSpdif Spdif RDC Peripheral.
rdcPdapEcspi1 Ecspi1 RDC Peripheral.
rdcPdapEcspi2 Ecspi2 RDC Peripheral.
rdcPdapEcspi3 Ecspi3 RDC Peripheral.
rdcPdapEcspi4 Ecspi4 RDC Peripheral.
rdcPdapUart1 Uart1 RDC Peripheral.
rdcPdapEsai Esai RDC Peripheral.
rdcPdapSsi1 Ssi1 RDC Peripheral.
rdcPdapSsi2 Ssi2 RDC Peripheral.
rdcPdapSsi3 Ssi3 RDC Peripheral.
rdcPdapAsrc Asrc RDC Peripheral.
rdcPdapSpbaMaMegamix SpbaMaMegamix RDC Peripheral.
rdcPdapGis Gis RDC Peripheral.
rdcPdapDcic1 Dcic1 RDC Peripheral.
rdcPdapDcic2 Dcic2 RDC Peripheral.
rdcPdapCsi1 Csi1 RDC Peripheral.
rdcPdapPxp Pxp RDC Peripheral.
rdcPdapCsi2 Csi2 RDC Peripheral.
rdcPdapLcdif1 Lcdif1 RDC Peripheral.
rdcPdapLcdif2 Lcdif2 RDC Peripheral.
rdcPdapVadc Vadc RDC Peripheral.
rdcPdapVdec Vdec RDC Peripheral.
rdcPdapSpDisplaymix SpDisplaymix RDC Peripheral.

12.3.2.3 enum _rdc_mr

Enumerator

rdcMrMmdc alignment 4096
rdcMrMmdcLast alignment 4096
rdcMrQspi1 alignment 4096
rdcMrQspi1Last alignment 4096
rdcMrQspi2 alignment 4096
rdcMrQspi2Last alignment 4096
rdcMrWeim alignment 4096
rdcMrWeimLast alignment 4096

rdcMrPcie alignment 4096
rdcMrPcieLast alignment 4096
rdcMrOcram alignment 128
rdcMrOcramLast alignment 128
rdcMrOcramS alignment 128
rdcMrOcramSLast alignment 128
rdcMrOcramL2 alignment 128
rdcMrOcramL2Last alignment 128

12.4 RDC Semaphore driver

12.4.1 Overview

The chapter describes the programming interface of the RDC Semaphore driver (platform/drivers/inc/rdc_-semaphore.h). The RDC SEMAPHORE provides hardware semaphores for peripheral exclusively access. The RDC SEMAPHORE driver provides a set of APIs to provide these services:

- RDC SEMAPHORE lock/unlock control
- RDC SEMAPHORE reset control

12.4.2 RDC SEMAPHORE lock/unlock control

Peripheral can be configured in RDC to access with hardware semaphore. In this mode, accessing it without acquiring the semaphore first causes violation, even if the peripheral is accessible with this master.

RDC_SEMAPHORE_TryLock(), RDC_SEMAPHORE_Lock(), RDC_SEMAPHORE_Unlock() are the operations for the hardware semaphore.

If the hardware semaphore is locked, the user can use RDC_SEMAPHORE_GetLockDomainID() to get the domain ID who locks the semaphore and RDC_SEMAPHORE_GetLockMaster() to get the master index on the bus who locks the semaphore.

12.4.3 RDC SEMAPHORE reset control

In some use cases, the user might need to recover from error status and the hardware semaphore need to be reset to free status. Hereby RDC_SEMAPHORE_Reset() is introduced to reset single peripheral semaphore and RDC_SEMAPHORE_ResetAll() to reset all peripheral semaphores.

Enumerations

- enum rdc_semaphore_status_t {
  statusRdcSemaphoreSuccess = 0U,
  statusRdcSemaphoreBusy = 1U }
  RDC Semaphore status return codes.

RDC_SEMAPHORE State Control

- rdc_semaphore_status_t RDC_SEMAPHORE_TryLock (uint32_t pdap)
  Lock RDC semaphore for shared peripheral access.
- void RDC_SEMAPHORE_Lock (uint32_t pdap)
  Lock RDC semaphore for shared peripheral access, polling until success.
- void RDC_SEMAPHORE_Unlock (uint32_t pdap)
  Unlock RDC semaphore.

- uint32_t RDC_SEMAPHORE_GetLockDomainID (uint32_t pdap)
  Get domain ID which locks the semaphore.
- uint32_t RDC_SEMAPHORE_GetLockMaster (uint32_t pdap)
  Get master index which locks the semaphore.

RDC_SEMAPHORE Reset Control

- void RDC_SEMAPHORE_Reset (uint32_t pdap)
  Reset RDC semaphore to unlocked status.
- void RDC_SEMAPHORE_ResetAll (RDC_SEMAPHORE_Type *base)
  Reset all RDC semaphore to unlocked status for certain RDC_SEMAPHORE instance.

12.4.4 Enumeration Type Documentation

12.4.4.1 enum rdc_semaphore_status_t

Enumerator

statusRdcSemaphoreSuccess Success.
statusRdcSemaphoreBusy RDC semaphore has been locked by other processor.

12.4.5 Function Documentation

12.4.5.1 rdc_semaphore_status_t RDC_SEMAPHORE_TryLock (uint32_t *pdap*)

Parameters

pdap	RDC peripheral assignment (see *rdc_pdap in rdc_defs*<device>.h)

Return values

statusRdcSemaphore-Success	Lock the semaphore successfully.
statusRdcSemaphoreBusy	Semaphore has been locked by other processor.

12.4.5.2 void RDC_SEMAPHORE_Lock (uint32_t *pdap*)

Parameters

pdap	RDC peripheral assignment (see *rdc_pdap in rdc_defs*<device>.h)

12.4.5.3 void RDC_SEMAPHORE_Unlock (uint32_t *pdap*)

Parameters

pdap	RDC peripheral assignment (see *rdc_pdap in rdc_defs*<device>.h)

12.4.5.4 uint32_t RDC_SEMAPHORE_GetLockDomainID (uint32_t *pdap*)

Parameters

pdap	RDC peripheral assignment (see *rdc_pdap in rdc_defs*<device>.h)

Returns

domain ID which locks the RDC semaphore

12.4.5.5 uint32_t RDC_SEMAPHORE_GetLockMaster (uint32_t *pdap*)

Parameters

pdap	RDC peripheral assignment (see *rdc_pdap in rdc_defs*<device>.h)

Returns

master index which locks the RDC semaphore, or RDC_SEMAPHORE_MASTER_NONE to indicate it is not locked.

12.4.5.6 void RDC_SEMAPHORE_Reset (uint32_t *pdap*)

Parameters

pdap	RDC peripheral assignment (see *rdc_pdap in rdc_defs*<device>.h)

12.4.5.7 void RDC_SEMAPHORE_ResetAll (RDC_SEMAPHORE_Type * *base*)

Parameters

base	RDC semaphore base pointer.

Chapter 13
Hardware Semaphores (SEMA4)

13.1 Overview

The FreeRTOS BSP provides a driver for the hardware semaphore (SEMA4) block of i.MX devices.

Modules

- SEMA4 driver

13.2 SEMA4 driver

13.2.1 Overview

This chapter describes the programming interface of the SEMA4 driver (platform/drivers/inc/sema4.h).

SEMA4 driver provides three kinds of APIs:

- APIs to lock/unlock SEMA4 gate.
- APIs to reset SEMA4.
- APIs to control SEMA4 interrupt.

13.2.2 SEMA4 lock and unlock

To lock some SEMA4 gate, there are two functions to use. SEMA4_TryLock() is a non-block function, and it only tries to lock the gate. If not locked, this function returns error. SEMA4_Lock() is a blocking function, and it spins to lock the gate until it is locked.

SEMA4_Unlock() can be used to unlock the gate. To get the SEMA4 gate lock status, use the function SEMA4_GetLockProcessor(), which returns the processor number that locks the gate, or SEMA4_PROCESSOR_NONE if the SEMA4 is free to use.

13.2.3 SEMA4 reset

SEMA4 driver provides the functions to reset specific gate or all gates. The functions are SEMA4_ResetGate() and SEMA4_ResetAllGates(). To check the reset status or which bus master resets the SEMA4, use the functions SEMA4_GetGateResetState() and SEMA4_GetGateResetBus().

SEMA driver can also reset specific gates or all gates' notifications. The functions are SEMA4_ResetNotification() and SEMA4_ResetAllNotifications(). To check the reset status or which bus master resets the SEMA4 notification, use the functions SEMA4_GetNotificationResetState() and SEMA4_GetNotificationResetBus().

13.2.4 SEMA4 interrupt control

SEMA4 driver can also provider interrupt control to help the user to implement event driven hardware semaphore and free CPU from spinning, because when the semaphore is locked by the other processor, the user can get unlock interrupt if the gate's interrupt is enabled. SEMA4_SetIntCmd() is used to enable or disable specific gate's interrupt.

SEMA4_GetStatusFlag() and SEMA4_GetIntEnabled() can be used to get current interrupt status and check whether the specific gate's interrupt is enabled.

Enumerations

- enum _sema4_status_flag {
  sema4StatusFlagGate0 = 1U << 7,
  sema4StatusFlagGate1 = 1U << 6,
  sema4StatusFlagGate2 = 1U << 5,
  sema4StatusFlagGate3 = 1U << 4,
  sema4StatusFlagGate4 = 1U << 3,
  sema4StatusFlagGate5 = 1U << 2,
  sema4StatusFlagGate6 = 1U << 1,
  sema4StatusFlagGate7 = 1U << 0,
  sema4StatusFlagGate8 = 1U << 15,
  sema4StatusFlagGate9 = 1U << 14,
  sema4StatusFlagGate10 = 1U << 13,
  sema4StatusFlagGate11 = 1U << 12,
  sema4StatusFlagGate12 = 1U << 11,
  sema4StatusFlagGate13 = 1U << 10,
  sema4StatusFlagGate14 = 1U << 9,
  sema4StatusFlagGate15 = 1U << 8 }
  Status flag.
- enum _sema4_reset_state {
  sema4ResetIdle = 0U,
  sema4ResetMid = 1U,
  sema4ResetFinished = 2U }
  SEMA4 reset finite state machine.
- enum sema4_status_t {
  statusSema4Success = 0U,
  statusSema4Busy = 1U }
  SEMA4 status return codes.

SEMA4 State Control

- sema4_status_t SEMA4_TryLock (SEMA4_Type *base, uint32_t gateIndex)
  Lock SEMA4 gate for exclusive access between multicore.
- void SEMA4_Lock (SEMA4_Type *base, uint32_t gateIndex)
  Lock SEMA4 gate for exclusive access between multicore, polling until success.
- void SEMA4_Unlock (SEMA4_Type *base, uint32_t gateIndex)
  Unlock SEMA4 gate.
- uint32_t SEMA4_GetLockProcessor (SEMA4_Type *base, uint32_t gateIndex)
  Get processor number which locks the SEMA4 gate.

SEMA4 Reset Control

- void SEMA4_ResetGate (SEMA4_Type *base, uint32_t gateIndex)
  Reset SEMA4 gate to unlocked status.

- void SEMA4_ResetAllGates (SEMA4_Type *base)
  Reset all SEMA4 gates to unlocked status.
- static uint8_t SEMA4_GetGateResetBus (SEMA4_Type *base)
  Get bus master number which performing the gate reset function.
- static uint8_t SEMA4_GetGateResetState (SEMA4_Type *base)
  Get sema4 gate reset state.
- void SEMA4_ResetNotification (SEMA4_Type *base, uint32_t gateIndex)
  Reset SEMA4 IRQ notification.
- void SEMA4_ResetAllNotifications (SEMA4_Type *base)
  Reset all IRQ notifications.
- static uint8_t SEMA4_GetNotificationResetBus (SEMA4_Type *base)
  Get bus master number which performing the notification reset function.
- static uint8_t SEMA4_GetNotificationResetState (SEMA4_Type *base)
  Get sema4 notification reset state.

SEMA4 Interrupt and Status Control

- static uint16_t SEMA4_GetStatusFlag (SEMA4_Type *base, uint16_t flags)
  Get SEMA4 notification status.
- void SEMA4_SetIntCmd (SEMA4_Type *base, uint16_t intMask, bool enable)
  Enable or disable SEMA4 IRQ notification.
- static uint16_t SEMA4_GetIntEnabled (SEMA4_Type *base, uint16_t flags)
  check whether SEMA4 IRQ notification enabled.

13.2.5 Enumeration Type Documentation

13.2.5.1 enum _sema4_status_flag

Enumerator

sema4StatusFlagGate0 Sema4 Gate 0 flag.
sema4StatusFlagGate1 Sema4 Gate 1 flag.
sema4StatusFlagGate2 Sema4 Gate 2 flag.
sema4StatusFlagGate3 Sema4 Gate 3 flag.
sema4StatusFlagGate4 Sema4 Gate 4 flag.
sema4StatusFlagGate5 Sema4 Gate 5 flag.
sema4StatusFlagGate6 Sema4 Gate 6 flag.
sema4StatusFlagGate7 Sema4 Gate 7 flag.
sema4StatusFlagGate8 Sema4 Gate 8 flag.
sema4StatusFlagGate9 Sema4 Gate 9 flag.
sema4StatusFlagGate10 Sema4 Gate 10 flag.
sema4StatusFlagGate11 Sema4 Gate 11 flag.
sema4StatusFlagGate12 Sema4 Gate 12 flag.
sema4StatusFlagGate13 Sema4 Gate 13 flag.
sema4StatusFlagGate14 Sema4 Gate 14 flag.
sema4StatusFlagGate15 Sema4 Gate 15 flag.

13.2.5.2 enum _sema4_reset_state

Enumerator

sema4ResetIdle Idle, waiting for the first data pattern write.
sema4ResetMid Waiting for the second data pattern write.
sema4ResetFinished Reset completed. Software can't get this state.

13.2.5.3 enum sema4_status_t

Enumerator

statusSema4Success Success.
statusSema4Busy SEMA4 gate has been locked by other processor.

13.2.6 Function Documentation

13.2.6.1 sema4_status_t SEMA4_TryLock (SEMA4_Type * *base*, uint32_t *gateIndex*)

Parameters

base	SEMA4 base pointer.
gateIndex	SEMA4 gate index.

Return values

statusSema4Success	Lock the gate successfully.
statusSema4Busy	SEMA4 gate has been locked by other processor.

13.2.6.2 void SEMA4_Lock (SEMA4_Type * *base*, uint32_t *gateIndex*)

Parameters

base	SEMA4 base pointer.
gateIndex	SEMA4 gate index.

13.2.6.3 void SEMA4_Unlock (SEMA4_Type * *base*, uint32_t *gateIndex*)

Parameters

base	SEMA4 base pointer.
gateIndex	SEMA4 gate index.

13.2.6.4 uint32_t SEMA4_GetLockProcessor (SEMA4_Type * *base,* uint32_t *gateIndex*)

Parameters

base	SEMA4 base pointer.
gateIndex	SEMA4 gate index.

Returns

processor number which locks the SEMA4 gate, or SEMA4_PROCESSOR_NONE to indicate the gate is not locked.

13.2.6.5 void SEMA4_ResetGate (SEMA4_Type * *base,* uint32_t *gateIndex*)

Parameters

base	SEMA4 base pointer.
gateIndex	SEMA4 gate index.

13.2.6.6 void SEMA4_ResetAllGates (SEMA4_Type * *base*)

Parameters

base	SEMA4 base pointer.

13.2.6.7 static uint8_t SEMA4_GetGateResetBus (SEMA4_Type * *base*) [inline], [static]

```
This function gets the bus master number which performing the
gate reset function.
```

Parameters

base	SEMA4 base pointer.

Returns

Bus master number.

13.2.6.8 static uint8_t SEMA4_GetGateResetState (SEMA4_Type * *base*) [inline], [static]

```
This function gets current state of the sema4 reset gate finite
state machine.
```

Parameters

base	SEMA4 base pointer.

Returns

Current state (see _sema4_reset_state).

13.2.6.9 void SEMA4_ResetNotification (SEMA4_Type * *base,* uint32_t *gateIndex*)

Parameters

base	SEMA4 base pointer.
gateIndex	SEMA4 gate index.

13.2.6.10 void SEMA4_ResetAllNotifications (SEMA4_Type * *base*)

Parameters

base	SEMA4 base pointer.

13.2.6.11 static uint8_t SEMA4_GetNotificationResetBus (SEMA4_Type * *base*) [inline], [static]

```
This function gets the bus master number which performing the notification
reset function.
```

Parameters

base	SEMA4 base pointer.

Returns

Bus master number.

13.2.6.12 static uint8_t SEMA4_GetNotificationResetState (SEMA4_Type * *base*) [inline], [static]

This function gets current state of the sema4 reset notification finite state machine.

Parameters

base	SEMA4 base pointer.

Returns

Current state (See _sema4_reset_state).

13.2.6.13 static uint16_t SEMA4_GetStatusFlag (SEMA4_Type * *base,* uint16_t *flags*) [inline], [static]

Parameters

base	SEMA4 base pointer.
flags	SEMA4 gate status mask (See _sema4_status_flag).

Returns

SEMA4 notification status bits. If bit value is set, the corresponding gate's notification is available.

13.2.6.14 void SEMA4_SetIntCmd (SEMA4_Type * *base,* uint16_t *intMask,* bool *enable*)

Parameters

base	SEMA4 base pointer.
intMask	SEMA4 gate status mask (see _sema4_status_flag).
enable	Enable/Disable Sema4 interrupt, only those gates whose intMask is set are affected. • true: Enable Sema4 interrupt. • false: Disable Sema4 interrupt.

13.2.6.15 static uint16_t SEMA4_GetIntEnabled (SEMA4_Type * *base*, uint16_t *flags*) [inline], [static]

Parameters

base	SEMA4 base pointer.
flags	SEMA4 gate status mask (see _sema4_status_flag).

Returns

SEMA4 notification interrupt enable status bits. If bit value is set, the corresponding gate's notification is enabled

Chapter 14
Universal Asynchronous Receiver/Transmitter (UART)

14.1 Overview

The FreeRTOS BSP provides a driver for the Universal Asynchronous Receiver/Transmitter (UART) block of i.MX devices.

Modules

- UART driver

14.2 UART driver

14.2.1 Overview

The section describes the programming interface of the UART driver (platform/drivers/inc/uart_imx.h).

14.2.2 UART initialization

To initialize the UART module, define an uart_init_config_t type variable and pass it to the UART_Init() function. Here is the Members of the structure definition:

1. clockRate: Current UART module clock frequency. This variable can be obtained by calling get_-uart_clock_freq() function.
2. baudRate: Desired UART baud rate. If the desired baud rate exceeds UART module's limitation, the most nearest legal value is chosen.n
3. wordLength: Data bits in one frame.
4. stopBitNum: Number of stop bits in one frame.
5. parity: Parity error check mode of this module.
6. direction: Data transfer direction of this module. This field is used to select the transfer direction. Choose the direction that can be used only to save system' s power.

Call UART_SetTxFifoWatermark() and UART_SetRxFifoWatermark() to set the watermark of TX/RX FIFO. Then, call UART_Enable() to enable UART module and transfer data through the UART port.

14.2.3 FlexCAN Data Transactions

UART driver provides these APIs for data transactions:

```
UART_Putchar()
UART_Getchar()
```

UART data send

To send data through UART port, follow these steps:

1. Call UART_GetStatusFlag() to check if UART Tx FIFO has available space.
2. If Tx FIFO is not full, call UART_Putchar() to add data to Tx FIFO.
3. Call UART_GetStatusFlag() to check if UART Transmit is finished.
4. Repeat the above process to send more data.

UART data receive

To receive data through UART bus, follow these steps:

1. Call UART_GetStatusFlag() to check if UART Rx FIFO has received data.

2. If Rx FIFO is not empty, call UART_Getchar() to read data from Rx FIFO.
3. Repeat the above process to read more data until Rx FIFO empty.

UART status and interrupt

This driver also provide APIs to handle UART module Status and Interrupt:

1. Call UART_SetIntCmd() to enable/disable UART module interrupt.
2. Call UART_GetStatusFlag() to get the UART status flags (described in enum _uart_status_flag) condition.
3. Call UART_ClearStatusFlag() to clear specified status flags.

Specific UART functions

Besides the functions mentioned above, the UART driver also provides a set of functions for special purpose, like Auto Baud Detection, RS-485 multidrop communication, and IrDA compatible low-speed optical communication. For more information about how to use these driver, see the *i.MX 6SoloX Applications Processor Reference Manual* (IMX6SXRM) and the function description below.

Example

For more information about how to use this driver, see the UART demo/example under examples/<board_name>/.

Data Structures

- struct uart_init_config_t
  Uart module initialization structure. *More...*

Enumerations

- enum _uart_word_length {
  uartWordLength7Bits = 0x0,
  uartWordLength8Bits = UART_UCR2_WS_MASK }
  UART number of data bits in a character.
- enum _uart_stop_bit_num {
  uartStopBitNumOne = 0x0,
  uartStopBitNumTwo = UART_UCR2_STPB_MASK }
  UART number of stop bits.
- enum _uart_partity_mode {
  uartParityDisable = 0x0,
  uartParityEven = UART_UCR2_PREN_MASK,
  uartParityOdd = UART_UCR2_PREN_MASK | UART_UCR2_PROE_MASK }

UART parity mode.

- enum _uart_direction_mode {
  uartDirectionDisable = 0x0,
  uartDirectionTx = UART_UCR2_TXEN_MASK,
  uartDirectionRx = UART_UCR2_RXEN_MASK,
  uartDirectionTxRx = UART_UCR2_TXEN_MASK | UART_UCR2_RXEN_MASK }

  Data transfer direction.
- enum _uart_interrupt {
  uartIntAutoBaud = 0x0080000F,
  uartIntTxReady = 0x0080000D,
  uartIntIdle = 0x0080000C,
  uartIntRxReady = 0x00800009,
  uartIntTxEmpty = 0x00800006,
  uartIntRtsDelta = 0x00800005,
  uartIntEscape = 0x0084000F,
  uartIntRts = 0x00840004,
  uartIntAgingTimer = 0x00840003,
  uartIntDtr = 0x0088000D,
  uartIntParityError = 0x0088000C,
  uartIntFrameError = 0x0088000B,
  uartIntDcd = 0x00880009,
  uartIntRi = 0x00880008,
  uartIntRxDs = 0x00880006,
  uartInttAirWake = 0x00880005,
  uartIntAwake = 0x00880004,
  uartIntDtrDelta = 0x00880003,
  uartIntAutoBaudCnt = 0x00880000,
  uartIntIr = 0x008C0008,
  uartIntWake = 0x008C0007,
  uartIntTxComplete = 0x008C0003,
  uartIntBreakDetect = 0x008C0002,
  uartIntRxOverrun = 0x008C0001,
  uartIntRxDataReady = 0x008C0000,
  uartIntRs485SlaveAddrMatch = 0x00B80003 }

  This enumeration contains the settings for all of the UART interrupt configurations.
- enum _uart_status_flag {

uartStatusRxCharReady = 0x0000000F,
uartStatusRxError = 0x0000000E,
uartStatusRxOverrunError = 0x0000000D,
uartStatusRxFrameError = 0x0000000C,
uartStatusRxBreakDetect = 0x0000000B,
uartStatusRxParityError = 0x0000000A,
uartStatusParityError = 0x0094000F,
uartStatusRtsStatus = 0x0094000E,
uartStatusTxReady = 0x0094000D,
uartStatusRtsDelta = 0x0094000C,
uartStatusEscape = 0x0094000B,
uartStatusFrameError = 0x0094000A,
uartStatusRxReady = 0x00940009,
uartStatusAgingTimer = 0x00940008,
uartStatusDtrDelta = 0x00940007,
uartStatusRxDs = 0x00940006,
uartStatustAirWake = 0x00940005,
uartStatusAwake = 0x00940004,
uartStatusRs485SlaveAddrMatch = 0x00940003,
uartStatusAutoBaud = 0x0098000F,
uartStatusTxEmpty = 0x0098000E,
uartStatusDtr = 0x0098000D,
uartStatusIdle = 0x0098000C,
uartStatusAutoBaudCntStop = 0x0098000B,
uartStatusRiDelta = 0x0098000A,
uartStatusRi = 0x00980009,
uartStatusIr = 0x00980008,
uartStatusWake = 0x00980007,
uartStatusDcdDelta = 0x00980006,
uartStatusDcd = 0x00980005,
uartStatusRts = 0x00980004,
uartStatusTxComplete = 0x00980003,
uartStatusBreakDetect = 0x00980002,
uartStatusRxOverrun = 0x00980001,
uartStatusRxDataReady = 0x00980000 }

Flag for UART interrupt/DMA status check or polling status.

- enum _uart_dma {
  uartDmaRxReady = 0x00800008,
  uartDmaTxReady = 0x00800003,
  uartDmaAgingTimer = 0x00800002,
  uartDmaIdle = 0x008C0006 }

  The events generate the DMA Request.
- enum _uart_rts_int_trigger_edge {
  uartRtsTriggerEdgeRising = UART_UCR2_RTEC(0),
  uartRtsTriggerEdgeFalling = UART_UCR2_RTEC(1),

uartRtsTriggerEdgeBoth = UART_UCR2_RTEC(2) }
RTS pin interrupt trigger edge.

- enum _uart_modem_mode {
  uartModemModeDce = 0,
  uartModemModeDte = UART_UFCR_DCEDTE_MASK }
  UART module modem role selections.
- enum _uart_dtr_int_trigger_edge {
  uartDtrTriggerEdgeRising = UART_UCR3_DPEC(0),
  uartDtrTriggerEdgeFalling = UART_UCR3_DPEC(1),
  uartDtrTriggerEdgeBoth = UART_UCR3_DPEC(2) }
  DTR pin interrupt trigger edge.
- enum _uart_irda_vote_clock {
  uartIrdaVoteClockSampling = 0x0,
  uartIrdaVoteClockReference = UART_UCR4_IRSC_MASK }
  IrDA vote clock selections.
- enum _uart_rx_idle_condition {
  uartRxIdleMoreThan4Frames = UART_UCR1_ICD(0),
  uartRxIdleMoreThan8Frames = UART_UCR1_ICD(1),
  uartRxIdleMoreThan16Frames = UART_UCR1_ICD(2),
  uartRxIdleMoreThan32Frames = UART_UCR1_ICD(3) }
  UART module Rx Idle condition selections.

UART Initialization and Configuration functions

- void UART_Init (UART_Type *base, const uart_init_config_t *initConfig)
  Initialize UART module with given initialization structure.
- void UART_Deinit (UART_Type *base)
  This function reset UART module register content to its default value.
- static void UART_Enable (UART_Type *base)
  This function is used to Enable the UART Module.
- static void UART_Disable (UART_Type *base)
  This function is used to Disable the UART Module.
- void UART_SetBaudRate (UART_Type *base, uint32_t clockRate, uint32_t baudRate)
  This function is used to set the baud rate of UART Module.
- static void UART_SetDirMode (UART_Type *base, uint32_t direction)
  This function is used to set the transform direction of UART Module.
- static void UART_SetRxIdleCondition (UART_Type *base, uint32_t idleCondition)
  This function is used to set the number of frames RXD is allowed to be idle before an idle condition is reported.
- void UART_SetInvertCmd (UART_Type *base, uint32_t direction, bool invert)
  This function is used to set the polarity of UART signal.

Low Power Mode functions.

- void UART_SetDozeMode (UART_Type *base, bool enable)
  This function is used to set UART enable condition in the DOZE state.
- void UART_SetLowPowerMode (UART_Type *base, bool enable)

This function is used to set UART enable condition of the UART low power feature.

Data transfer functions.

- static void UART_Putchar (UART_Type *base, uint8_t data)
  This function is used to send data in RS-232 and IrDA Mode.
- static uint8_t UART_Getchar (UART_Type *base)
  This function is used to receive data in RS-232 and IrDA Mode.

Interrupt and Flag control functions.

- void UART_SetIntCmd (UART_Type *base, uint32_t intSource, bool enable)
  This function is used to set the enable condition of specific UART interrupt source.
- bool UART_GetStatusFlag (UART_Type *base, uint32_t flag)
  This function is used to get the current status of specific UART status flag(including interrupt flag).
- void UART_ClearStatusFlag (UART_Type *base, uint32_t flag)
  This function is used to get the current status of specific UART status flag.

DMA control functions.

- void UART_SetDmaCmd (UART_Type *base, uint32_t dmaSource, bool enable)
  This function is used to set the enable condition of specific UART DMA source.

FIFO control functions.

- static void UART_SetTxFifoWatermark (UART_Type *base, uint8_t watermark)
  This function is used to set the watermark of UART Tx FIFO.
- static void UART_SetRxFifoWatermark (UART_Type *base, uint8_t watermark)
  This function is used to set the watermark of UART Rx FIFO.

Hardware Flow control and Modem Signal functions.

- void UART_SetRtsFlowCtrlCmd (UART_Type *base, bool enable)
  This function is used to set the enable condition of RTS Hardware flow control.
- static void UART_SetRtsIntTriggerEdge (UART_Type *base, uint32_t triggerEdge)
  This function is used to set the RTS interrupt trigger edge.
- void UART_SetCtsFlowCtrlCmd (UART_Type *base, bool enable)
  This function is used to set the enable condition of CTS auto control.
- void UART_SetCtsPinLevel (UART_Type *base, bool active)
  This function is used to control the CTS_B pin state when auto CTS control is disabled.
- static void UART_SetCtsTriggerLevel (UART_Type *base, uint8_t triggerLevel)
  This function is used to set the auto CTS_B pin control trigger level.
- void UART_SetModemMode (UART_Type *base, uint32_t mode)
  This function is used to set the role (DTE/DCE) of UART module in RS-232 communication.

- static void UART_SetDtrIntTriggerEdge (UART_Type *base, uint32_t triggerEdge)
  This function is used to set the edge of DTR_B (DCE) or DSR_B (DTE) on which an interrupt is generated.
- void UART_SetDtrPinLevel (UART_Type *base, bool active)
  This function is used to set the pin state of DSR pin(for DCE mode) or DTR pin(for DTE mode) for the modem interface.
- void UART_SetDcdPinLevel (UART_Type *base, bool active)
  This function is used to set the pin state of DCD pin.
- void UART_SetRiPinLevel (UART_Type *base, bool active)
  This function is used to set the pin state of RI pin.

Multiprocessor and RS-485 functions.

- void UART_Putchar9 (UART_Type *base, uint16_t data)
  This function is used to send 9 Bits length data in RS-485 Multidrop mode.
- uint16_t UART_Getchar9 (UART_Type *base)
  This functions is used to receive 9 Bits length data in RS-485 Multidrop mode.
- void UART_SetMultidropMode (UART_Type *base, bool enable)
  This function is used to set the enable condition of 9-Bits data or Multidrop mode.
- void UART_SetSlaveAddressDetectCmd (UART_Type *base, bool enable)
  This function is used to set the enable condition of Automatic Address Detect Mode.
- static void UART_SetSlaveAddress (UART_Type *base, uint8_t slaveAddress)
  This function is used to set the slave address char that the receiver tries to detect.

IrDA control functions.

- void UART_SetIrDACmd (UART_Type *base, bool enable)
  This function is used to set the enable condition of IrDA Mode.
- void UART_SetIrDAVoteClock (UART_Type *base, uint32_t voteClock)
  This function is used to set the clock for the IR pulsed vote logic.

Misc. functions.

- void UART_SetAutoBaudRateCmd (UART_Type *base, bool enable)
  This function is used to set the enable condition of Automatic Baud Rate Detection feature.
- static uint16_t UART_ReadBaudRateCount (UART_Type *base)
  This function is used to read the current value of Baud Rate Count Register value.
- void UART_SendBreakChar (UART_Type *base, bool active)
  This function is used to send BREAK character.It is important that SNDBRK is asserted high for a sufficient period of time to generate a valid BREAK.
- void UART_SetEscapeDecectCmd (UART_Type *base, bool enable)
  This function is used to Enable/Disable the Escape Sequence Decection feature.
- static void UART_SetEscapeChar (UART_Type *base, uint8_t escapeChar)
  This function is used to set the enable condition of Escape Sequence Detection feature.
- static void UART_SetEscapeTimerInterval (UART_Type *base, uint16_t timerInterval)
  This function is used to set the maximum time interval (in ms) allowed between escape characters.

14.2.4 Data Structure Documentation

14.2.4.1 struct uart_init_config_t

Data Fields

- uint32_t clockRate
  Current UART module clock freq.
- uint32_t baudRate
  Desired UART baud rate.
- uint32_t wordLength
  Data bits in one frame.
- uint32_t stopBitNum
  Number of stop bits in one frame.
- uint32_t parity
  Parity error check mode of this module.
- uint32_t direction
  Data transfer direction of this module.

14.2.4.1.0.9 Field Documentation

14.2.4.1.0.9.1 uint32_t uart_init_config_t::clockRate

14.2.4.1.0.9.2 uint32_t uart_init_config_t::baudRate

14.2.4.1.0.9.3 uint32_t uart_init_config_t::wordLength

14.2.4.1.0.9.4 uint32_t uart_init_config_t::stopBitNum

14.2.4.1.0.9.5 uint32_t uart_init_config_t::parity

14.2.4.1.0.9.6 uint32_t uart_init_config_t::direction

14.2.5 Enumeration Type Documentation

14.2.5.1 enum _uart_word_length

Enumerator

uartWordLength7Bits One character has 7 bits.
uartWordLength8Bits One character has 8 bits.

14.2.5.2 enum _uart_stop_bit_num

Enumerator

uartStopBitNumOne One bit Stop.
uartStopBitNumTwo Two bits Stop.

14.2.5.3 enum _uart_partity_mode

Enumerator

uartParityDisable Parity error check disabled.
uartParityEven Even error check is selected.
uartParityOdd Odd error check is selected.

14.2.5.4 enum _uart_direction_mode

Enumerator

uartDirectionDisable Both Tx and Rx are disabled.
uartDirectionTx Tx is enabled.
uartDirectionRx Rx is enabled.
uartDirectionTxRx Both Tx and Rx are enabled.

14.2.5.5 enum _uart_interrupt

Enumerator

uartIntAutoBaud Automatic baud rate detection Interrupt Enable.
uartIntTxReady transmitter ready Interrupt Enable.
uartIntIdle IDLE Interrupt Enable.
uartIntRxReady Receiver Ready Interrupt Enable.
uartIntTxEmpty Transmitter Empty Interrupt Enable.
uartIntRtsDelta RTS Delta Interrupt Enable.
uartIntEscape Escape Sequence Interrupt Enable.
uartIntRts Request to Send Interrupt Enable.
uartIntAgingTimer Aging Timer Interrupt Enable.
uartIntDtr Data Terminal Ready Interrupt Enable.
uartIntParityError Parity Error Interrupt Enable.
uartIntFrameError Frame Error Interrupt Enable.
uartIntDcd Data Carrier Detect Interrupt Enable.
uartIntRi Ring Indicator Interrupt Enable.
uartIntRxDs Receive Status Interrupt Enable.
uartInttAirWake Asynchronous IR WAKE Interrupt Enable.
uartIntAwake Asynchronous WAKE Interrupt Enable.
uartIntDtrDelta Data Terminal Ready Delta Interrupt Enable.
uartIntAutoBaudCnt Autobaud Counter Interrupt Enable.
uartIntIr Serial Infrared Interrupt Enable.
uartIntWake WAKE Interrupt Enable.
uartIntTxComplete TransmitComplete Interrupt Enable.
uartIntBreakDetect BREAK Condition Detected Interrupt Enable.

uartIntRxOverrun Receiver Overrun Interrupt Enable.
uartIntRxDataReady Receive Data Ready Interrupt Enable.
uartIntRs485SlaveAddrMatch RS-485 Slave Address Detected Interrupt Enable.

14.2.5.6 enum _uart_status_flag

Enumerator

uartStatusRxCharReady Rx Character Ready Flag.
uartStatusRxError Rx Error Detect Flag.
uartStatusRxOverrunError Rx Overrun Flag.
uartStatusRxFrameError Rx Frame Error Flag.
uartStatusRxBreakDetect Rx Break Detect Flag.
uartStatusRxParityError Rx Parity Error Flag.
uartStatusParityError Parity Error Interrupt Flag.
uartStatusRtsStatus RTS_B Pin Status Flag.
uartStatusTxReady Transmitter Ready Interrupt/DMA Flag.
uartStatusRtsDelta RTS Delta Flag.
uartStatusEscape Escape Sequence Interrupt Flag.
uartStatusFrameError Frame Error Interrupt Flag.
uartStatusRxReady Receiver Ready Interrupt/DMA Flag.
uartStatusAgingTimer Ageing Timer Interrupt Flag.
uartStatusDtrDelta DTR Delta Flag.
uartStatusRxDs Receiver IDLE Interrupt Flag.
uartStatustAirWake Asynchronous IR WAKE Interrupt Flag.
uartStatusAwake Asynchronous WAKE Interrupt Flag.
uartStatusRs485SlaveAddrMatch RS-485 Slave Address Detected Interrupt Flag.
uartStatusAutoBaud Automatic Baud Rate Detect Complete Flag.
uartStatusTxEmpty Transmit Buffer FIFO Empty.
uartStatusDtr DTR edge triggered interrupt flag.
uartStatusIdle Idle Condition Flag.
uartStatusAutoBaudCntStop Autobaud Counter Stopped Flag.
uartStatusRiDelta Ring Indicator Delta Flag.
uartStatusRi Ring Indicator Input Flag.
uartStatusIr Serial Infrared Interrupt Flag.
uartStatusWake Wake Flag.
uartStatusDcdDelta Data Carrier Detect Delta Flag.
uartStatusDcd Data Carrier Detect Input Flag.
uartStatusRts RTS Edge Triggered Interrupt Flag.
uartStatusTxComplete Transmitter Complete Flag.
uartStatusBreakDetect BREAK Condition Detected Flag.
uartStatusRxOverrun Overrun Error Flag.
uartStatusRxDataReady Receive Data Ready Flag.

14.2.5.7 enum _uart_dma

Enumerator

uartDmaRxReady Receive Ready DMA Enable.
uartDmaTxReady Transmitter Ready DMA Enable.
uartDmaAgingTimer Aging DMA Timer Enable.
uartDmaIdle DMA IDLE Condition Detected Interrupt Enable.

14.2.5.8 enum _uart_rts_int_trigger_edge

Enumerator

uartRtsTriggerEdgeRising RTS pin interrupt triggered on rising edge.
uartRtsTriggerEdgeFalling RTS pin interrupt triggered on falling edge.
uartRtsTriggerEdgeBoth RTS pin interrupt triggered on both edge.

14.2.5.9 enum _uart_modem_mode

Enumerator

uartModemModeDce UART module works as DCE.
uartModemModeDte UART module works as DTE.

14.2.5.10 enum _uart_dtr_int_trigger_edge

Enumerator

uartDtrTriggerEdgeRising DTR pin interrupt triggered on rising edge.
uartDtrTriggerEdgeFalling DTR pin interrupt triggered on falling edge.
uartDtrTriggerEdgeBoth DTR pin interrupt triggered on both edge.

14.2.5.11 enum _uart_irda_vote_clock

Enumerator

uartIrdaVoteClockSampling The vote logic uses the sampling clock (16x baud rate) for normal operation.
uartIrdaVoteClockReference The vote logic uses the UART reference clock.

14.2.5.12 enum _uart_rx_idle_condition

Enumerator

uartRxIdleMoreThan4Frames Idle for more than 4 frames.
uartRxIdleMoreThan8Frames Idle for more than 8 frames.
uartRxIdleMoreThan16Frames Idle for more than 16 frames.
uartRxIdleMoreThan32Frames Idle for more than 32 frames.

14.2.6 Function Documentation

14.2.6.1 void UART_Init (UART_Type * *base*, const uart_init_config_t * *initConfig*)

Parameters

base	UART base pointer.
initConfig	UART initialization structure (see uart_init_config_t structure above).

14.2.6.2 void UART_Deinit (UART_Type * *base*)

Parameters

base	UART base pointer.

14.2.6.3 static void UART_Enable (UART_Type * *base*) [inline], [static]

Parameters

base	UART base pointer.

14.2.6.4 static void UART_Disable (UART_Type * *base*) [inline], [static]

Parameters

base	UART base pointer.

14.2.6.5 void UART_SetBaudRate (UART_Type * *base*, uint32_t *clockRate*, uint32_t *baudRate*)

Parameters

base	UART base pointer.
clockRate	UART module clock frequency.
baudRate	Desired UART module baud rate.

14.2.6.6 static void UART_SetDirMode (UART_Type * *base*, uint32_t *direction*) [inline], [static]

Parameters

base	UART base pointer.
direction	UART transfer direction (see _uart_direction_mode enumeration).

14.2.6.7 static void UART_SetRxIdleCondition (UART_Type * *base*, uint32_t *idleCondition*) [inline], [static]

The available condition can be select from _uart_idle_condition enumeration.

Parameters

base	UART base pointer.
idleCondition	The condition that an idle condition is reported (see _uart_idle_condition enumeration).

14.2.6.8 void UART_SetInvertCmd (UART_Type * *base*, uint32_t *direction*, bool *invert*)

The polarity of Tx and Rx can be set separately.

Parameters

base	UART base pointer.
direction	UART transfer direction (see _uart_direction_mode enumeration).
invert	Set true to invert the polarity of UART signal.

14.2.6.9 void UART_SetDozeMode (UART_Type * *base*, bool *enable*)

Parameters

base	UART base pointer.
enable	Enable/Disable UART module in doze mode. • true: Enable UART module in doze mode. • false: Disable UART module in doze mode.

14.2.6.10 void UART_SetLowPowerMode (UART_Type * *base,* bool *enable*)

Parameters

base	UART base pointer.
enable	Enable/Disable UART module low power feature. • true: Enable UART module low power feature. • false: Disable UART module low power feature.

14.2.6.11 static void UART_Putchar (UART_Type * *base,* uint8_t *data*) [inline], [static]

A independent 9 Bits RS-485 send data function is provided.

Parameters

base	UART base pointer.
data	Data to be set through UART module.

14.2.6.12 static uint8_t UART_Getchar (UART_Type * *base*) [inline], [static]

A independent 9 Bits RS-485 receive data function is provided.

Parameters

base	UART base pointer.

Returns

The data received from UART module.

14.2.6.13 void UART_SetIntCmd (UART_Type * *base*, uint32_t *intSource*, bool *enable*)

The available interrupt source can be select from _uart_interrupt enumeration.

Parameters

base	UART base pointer.
intSource	Available interrupt source for this module.
enable	Enable/Disable corresponding interrupt. • true: Enable corresponding interrupt. • false: Disable corresponding interrupt.

14.2.6.14 bool UART_GetStatusFlag (UART_Type * *base*, uint32_t *flag*)

The available status flag can be select from _uart_status_flag enumeration.

Parameters

base	UART base pointer.
flag	Status flag to check.

Returns

current state of corresponding status flag.

14.2.6.15 void UART_ClearStatusFlag (UART_Type * *base*, uint32_t *flag*)

The available status flag can be select from _uart_status_flag enumeration.

Parameters

base	UART base pointer.
flag	Status flag to clear.

14.2.6.16 void UART_SetDmaCmd (UART_Type * *base*, uint32_t *dmaSource*, bool *enable*)

The available DMA source can be select from _uart_dma enumeration.

Parameters

base	UART base pointer.
dmaSource	The Event that can generate DMA request.
enable	Enable/Disable corresponding DMA source. • true: Enable corresponding DMA source. • false: Disable corresponding DMA source.

14.2.6.17 static void UART_SetTxFifoWatermark (UART_Type * *base*, uint8_t *watermark*) [inline], [static]

```
A maskable interrupt is generated whenever the data level in
the TxFIFO falls below the Tx FIFO watermark.
```

Parameters

base	UART base pointer.
watermark	The Tx FIFO watermark.

14.2.6.18 static void UART_SetRxFifoWatermark (UART_Type * *base*, uint8_t *watermark*) [inline], [static]

```
A maskable interrupt is generated whenever the data level in
the RxFIFO reaches the Rx FIFO watermark.
```

Parameters

base	UART base pointer.
watermark	The Rx FIFO watermark.

14.2.6.19 void UART_SetRtsFlowCtrlCmd (UART_Type * *base*, bool *enable*)

Parameters

base	UART base pointer.
enable	Enable/Disbale RTS hardware flow control. • true: Enable RTS hardware flow control. • false: Disbale RTS hardware flow control.

14.2.6.20 static void UART_SetRtsIntTriggerEdge (UART_Type * *base*, uint32_t *triggerEdge*) [inline], [static]

```
The available trigger edge can be select from
@ref _uart_rts_trigger_edge enumeration.
```

Parameters

base	UART base pointer.
triggerEdge	Available RTS pin interrupt trigger edge.

14.2.6.21 void UART_SetCtsFlowCtrlCmd (UART_Type * *base*, bool *enable*)

if CTS control is enabled, the CTS_B pin is controlled by the receiver, otherwise the CTS_B pin is controlled by UART_CTSPinCtrl function.

Parameters

base	UART base pointer.
enable	Enable/Disable CTS auto control. • true: Enable CTS auto control. • false: Disable CTS auto control.

14.2.6.22 void UART_SetCtsPinLevel (UART_Type * *base*, bool *active*)

```
The CTS_B pin is low(active)
The CTS_B pin is high(inactive)
```

Parameters

base	UART base pointer.
active	The CTS_B pin state to set. • true: the CTS_B pin active; • false: the CTS_B pin inactive.

14.2.6.23 static void UART_SetCtsTriggerLevel (UART_Type * *base*, uint8_t *triggerLevel*) [inline], [static]

The CTS_B pin is de-asserted when Rx FIFO reach CTS trigger level.

Parameters

base	UART base pointer.
triggerLevel	Auto CTS_B pin control trigger level.

14.2.6.24 void UART_SetModemMode (UART_Type * *base*, uint32_t *mode*)

Parameters

base	UART base pointer.
mode	The role(DTE/DCE) of UART module (see _uart_modem_mode enumeration).

14.2.6.25 static void UART_SetDtrIntTriggerEdge (UART_Type * *base*, uint32_t *triggerEdge*) [inline], [static]

Parameters

base	UART base pointer.
triggerEdge	The trigger edge on which an interrupt is generated (see _uart_dtr_trigger_edge enumeration above).

14.2.6.26 void UART_SetDtrPinLevel (UART_Type * *base,* bool *active*)

Parameters

base	UART base pointer.
active	The state of DSR pin. • true: DSR/DTR pin is logic one. • false: DSR/DTR pin is logic zero.

14.2.6.27 void UART_SetDcdPinLevel (UART_Type * *base*, bool *active*)

THIS FUNCTION IS FOR DCE MODE ONLY.

Parameters

base	UART base pointer.
active	The state of DCD pin. • true: DCD_B pin is logic one (DCE mode) • false: DCD_B pin is logic zero (DCE mode)

14.2.6.28 void UART_SetRiPinLevel (UART_Type * *base*, bool *active*)

THIS FUNCTION IS FOR DCE MODE ONLY.

Parameters

base	UART base pointer.
active	The state of RI pin. • true: RI_B pin is logic one (DCE mode) • false: RI_B pin is logic zero (DCE mode)

14.2.6.29 void UART_Putchar9 (UART_Type * *base*, uint16_t *data*)

Parameters

base	UART base pointer.
data	Data(9 bits) to be set through UART module.

14.2.6.30 uint16_t UART_Getchar9 (UART_Type * *base*)

Parameters

base	UART base pointer.

Returns

The data(9 bits) received from UART module.

14.2.6.31 void UART_SetMultidropMode (UART_Type * *base*, bool *enable*)

Parameters

base	UART base pointer.
enable	Enable/Disable Multidrop mode. • true: Enable Multidrop mode. • false: Disable Multidrop mode.

14.2.6.32 void UART_SetSlaveAddressDetectCmd (UART_Type * *base*, bool *enable*)

Parameters

base	UART base pointer.
enable	Enable/Disable Automatic Address Detect mode. • true: Enable Automatic Address Detect mode. • false: Disable Automatic Address Detect mode.

14.2.6.33 static void UART_SetSlaveAddress (UART_Type * *base*, uint8_t *slaveAddress*) [inline], [static]

Parameters

base	UART base pointer.
slaveAddress	The slave to detect.

14.2.6.34 void UART_SetIrDACmd (UART_Type * *base*, bool *enable*)

Parameters

base	UART base pointer.
enable	Enable/Disable IrDA mode. • true: Enable IrDA mode. • false: Disable IrDA mode.

14.2.6.35 void UART_SetIrDAVoteClock (UART_Type * *base*, uint32_t *voteClock*)

The available clock can be select from _uart_irda_vote_clock enumeration.

Parameters

base	UART base pointer.
voteClock	The available IrDA vote clock selection.

14.2.6.36 void UART_SetAutoBaudRateCmd (UART_Type * *base*, bool *enable*)

Parameters

base	UART base pointer.
enable	Enable/Disable Automatic Baud Rate Detection feature. • true: Enable Automatic Baud Rate Detection feature. • false: Disable Automatic Baud Rate Detection feature.

14.2.6.37 static uint16_t UART_ReadBaudRateCount (UART_Type * *base*) [inline], [static]

this counter is used by Auto Baud Rate Detect feature.

Parameters

base	UART base pointer.

Returns

Current Baud Rate Count Register value.

14.2.6.38 void UART_SendBreakChar (UART_Type * *base,* bool *active*)

Parameters

base	UART base pointer.
active	Asserted high to generate BREAK. • true: Generate BREAK character. • false: Stop generate BREAK character.

14.2.6.39 void UART_SetEscapeDecectCmd (UART_Type * *base*, bool *enable*)

Parameters

base	UART base pointer.
enable	Enable/Disable Escape Sequence Decection. • true: Enable Escape Sequence Decection. • false: Disable Escape Sequence Decection.

14.2.6.40 static void UART_SetEscapeChar (UART_Type * *base*, uint8_t *escapeChar*) [inline], [static]

Parameters

base	UART base pointer.
escapeChar	The Escape Character to detect.

14.2.6.41 static void UART_SetEscapeTimerInterval (UART_Type * *base*, uint16_t *timerInterval*) [inline], [static]

Parameters

base	UART base pointer.
timerInterval	Maximum time interval allowed between escape characters.

Chapter 15
Watchdog Timer (WDOG)

15.1 Overview

The FreeRTOS BSP provides a driver for the Watchdog Timer (WDOG) block of i.MX devices.

Modules

- WDOG driver on i.MX

15.2 WDOG driver on i.MX

15.2.1 Overview

The chapter describes the programming interface of the WDOG driver on i.MX (platform/drivers/inc/wdog_imx.h). The i.MX watchdog protects against system failures by providing a method by which to escape from unexpected events or programming errors. The WDOG driver on i.MX provides a set of APIs to provide these services:

- Watchdog general control
- Watchdog interrupt control

15.2.2 Watchdog general control

After reset, WDOG_DisablePowerdown() must be called to avoid the power down timeout. It is a one-shot function and cannot be called more than once.

Before enabling the watchdog, WDOG_Init() is needed to initialize the watchdog driver and specify the watchdog behavior in different modes. This function is also a one-shot function.

Then WDOG_Enable() can be used to enable the watchdog with specified timeout, and when timeout, CPU is reset and external pin WDOG_B might be asserted based on the behavior setting. Once enabled, the watchdog cannot be disabled so it is also a one-shot function.

To avoid timeout, the program must WDOG_Refresh() the counter periodically.

The user can also use WDOG_Reset() to reset the CPU and assert some reset signal specified by parameters immediately.

15.2.3 Watchdog interrupt control

i.MX watchdog also provides interrupt before timeout reset occurs. The user can use WDOG_EnableInt() with proper time to make sure the interrupt would happen some time before timeout reset.

When the interrupt occurs, WDOG_ClearStatusFlag() is used to clear the status. WDOG_IsIntPending() can be used to check whether there's any interrupt pending.

Data Structures

- struct wdog_init_config_t
  Structure to configure the running mode. More...

Enumerations

- enum _wdog_reset_source {
  wdogResetSourcePor = WDOG_WRSR_POR_MASK,
  wdogResetSourceTimeout = WDOG_WRSR_TOUT_MASK,
  wdogResetSourceSwRst = WDOG_WRSR_SFTW_MASK }
  The reset source of latest reset.

WDOG State Control

- static void WDOG_Init (WDOG_Type *base, const wdog_init_config_t *initConfig)
  Configure WDOG functions, call once only.
- void WDOG_Enable (WDOG_Type *base, uint8_t timeout)
  Enable WDOG with timeout, call once only.
- void WDOG_Reset (WDOG_Type *base, bool wda, bool srs)
  Assert WDOG software reset signal.
- static uint32_t WDOG_GetResetSource (WDOG_Type *base)
  Get the latest reset source generated due to WatchDog Timer.
- void WDOG_Refresh (WDOG_Type *base)
  Refresh the WDOG to prevent timeout.
- static void WDOG_DisablePowerdown (WDOG_Type *base)
  Disable WDOG power down counter.

WDOG Interrupt Control

- static void WDOG_EnableInt (WDOG_Type *base, uint8_t time)
  Enable WDOG interrupt.
- static bool WDOG_IsIntPending (WDOG_Type *base)
  Check whether WDOG interrupt is pending.
- static void WDOG_ClearStatusFlag (WDOG_Type *base)
  Clear WDOG interrupt status.

15.2.4 Data Structure Documentation

15.2.4.1 struct wdog_init_config_t

Data Fields

- bool wdw
  true: suspend in low power wait, false: not suspend
- bool wdt
  true: assert WDOG_B when timeout, false: not assert WDOG_B
- bool wdbg
  true: suspend in debug mode, false: not suspend
- bool wdzst
  true: suspend in doze and stop mode, false: not suspend

15.2.5 Enumeration Type Documentation

15.2.5.1 enum _wdog_reset_source

Enumerator

wdogResetSourcePor Indicates the reset is the result of a power on reset.
wdogResetSourceTimeout Indicates the reset is the result of a WDOG timeout.
wdogResetSourceSwRst Indicates the reset is the result of a software reset.

15.2.6 Function Documentation

15.2.6.1 static void WDOG_Init (WDOG_Type * *base,* const wdog_init_config_t * *initConfig*) [inline], [static]

Parameters

base	WDOG base pointer.
initConfig	WDOG mode configuration

15.2.6.2 void WDOG_Enable (WDOG_Type * *base,* uint8_t *timeout*)

Parameters

base	WDOG base pointer.
timeout	WDOG timeout ((n+1)/2 second)

15.2.6.3 void WDOG_Reset (WDOG_Type * *base,* bool *wda,* bool *srs*)

Parameters

base	WDOG base pointer.
wda	WDOG reset. • true: Assert WDOG_B. • false: No impact on WDOG_B.

srs	System reset. • true: Assert system reset WDOG_RESET_B_DEB. • false: No impact on system reset.

15.2.6.4 static uint32_t WDOG_GetResetSource (WDOG_Type * *base*) [inline], [static]

Parameters

base	WDOG base pointer.

Returns

The latest reset source (see _wdog_reset_source enumeration).

15.2.6.5 void WDOG_Refresh (WDOG_Type * *base*)

Parameters

base	WDOG base pointer.

15.2.6.6 static void WDOG_DisablePowerdown (WDOG_Type * *base*) [inline], [static]

Parameters

base	WDOG base pointer.

15.2.6.7 static void WDOG_EnableInt (WDOG_Type * *base,* uint8_t *time*) [inline], [static]

Parameters

base	WDOG base pointer.
time	how long before the timeout must the interrupt occur (n/2 seconds).

15.2.6.8 static bool WDOG_IsIntPending (WDOG_Type * *base*) [inline], [static]

Parameters

base	WDOG base pointer.

Returns

WDOG interrupt status.
- true: Pending.
- false: Not pending.

15.2.6.9 static void WDOG_ClearStatusFlag (WDOG_Type * *base*) [inline], [static]

Parameters

base	WDOG base pointer.

Chapter 16
Utilities for the FreeRTOS BSP

16.1 Overview

The FreeRTOS BSP provides debug console to help user with their development.

Modules

- Debug Console

16.2 Debug Console

16.2.1 Overview

This section describes the programming interface of the debug console driver.

16.2.2 Debug Console Initialization

To initialize the DbgConsole module, call the DbgConsole_Init() function and pass in the parameters needed by this function. This function automatically enables the module and clock. After the DbgConsole_Init() function is called and returned, stdout and stdin are connected to the selected UART.

The parameters needed by this function are shown here:

```
1. UART_Type* base      : The base address of the UART module used as debug console;
2. uint32_t   clockRate : The clock source frequency of UART module, this value can be obtained by calling
      get_uart_clock_freq() function;
3. uint32_t   baudRate  : The desired baud rate frequency.
```

Debug console state is stored in debug_console_state_t structure:

```
typedef struct DebugConsoleState {
    bool  inited;                   /*<! Identify debug console initialized or not. */
    void* base;                     /*<! Base of the IP register. */
    debug_console_ops_t ops;        /*<! Operation function pointers for debug UART operations. */
} debug_console_state_t;
```

This example shows how to call the DbgConsole_Init() given the user configuration parameters.

```
DbgConsole_Init(BOARD_DEBUG_UART_BASEADDR, get_uart_clock_freq(BOARD_DEBUG_UART_BASEADDR), 1
      15200);
```

Debug Console formatted IO

Debug console has its own printf/scanf/putchar/getchar functions which are defined in the header:

```
int debug_printf(const char  *fmt_s, ...);
int debug_putchar(int ch);
int debug_scanf(const char  *fmt_ptr, ...);
int debug_getchar(void);
```

Choose toolchain's printf/scanf or FreeRTOS BSP version printf/scanf:

```
/*Configuration for toolchain's printf/scanf or FreeRTOS BSP version printf/scanf */
#define PRINTF          debug_printf
//#define PRINTF          printf
#define SCANF           debug_scanf
//#define SCANF           scanf
#define PUTCHAR         debug_putchar
//#define PUTCHAR         putchar
#define GETCHAR         debug_getchar
//#define GETCHAR         getchar
```

Function _doprint outputs its parameters according to a formatted string. I/O is performed by calling given function pointer using (∗func_ptr)(c,farg).

```
int _doprint(void *farg, PUTCHAR_FUNC func_ptr, int max_count, char *fmt, va_list ap)
```

Function scan_prv converts an input line of ASCII characters based upon a provided string format.

```
int scan_prv(const char *line_ptr, char *format, va_list args_ptr)
```

Function mknumstr converts a radix number to a string and return its length.

```
static int32_t mknumstr (char *numstr, void *nump, int32_t neg, int32_t radix, bool use_caps);
```

Function mkfloatnumstr converts a floating radix number to a string and return its length.

```
static int32_t mkfloatnumstr (char *numstr, void *nump, int32_t radix, uint32_t precision_width);
```

Macros

- #define PRINTF debug_printf
  Configuration for toolchain's printf/scanf or NXP version printf/scanf.

Enumerations

- enum debug_console_status_t
  Error code for the debug console driver.

Initialization

- debug_console_status_t DbgConsole_Init (UART_Type ∗base, uint32_t clockRate, uint32_t baudRate)
  Initialize the UART_IMX used for debug messages.
- debug_console_status_t DbgConsole_DeInit (void)
  Deinitialize the UART/LPUART used for debug messages.
- int debug_printf (const char ∗fmt_s,...)
  Prints formatted output to the standard output stream.
- int debug_putchar (int ch)
  Writes a character to stdout.
- int debug_scanf (const char ∗fmt_ptr,...)
  Reads formatted data from the standard input stream.
- int debug_getchar (void)
  Reads a character from standard input.

16.2.3 Enumeration Type Documentation

16.2.3.1 enum debug_console_status_t

16.2.4 Function Documentation

16.2.4.1 debug_console_status_t DbgConsole_Init (UART_Type * *base*, uint32_t *clockRate*, uint32_t *baudRate*)

Call this function to enable debug log messages to be output via the specified UART_IMX base address and at the specified baud rate. Just initializes the UART_IMX to the given baud rate and 8N1. After this function has returned, stdout and stdin are connected to the selected UART_IMX. The debug_printf() function also uses this UART_IMX.

Parameters

base	Which UART_IMX instance is used to send debug messages.
clockRate	The input clock of UART_IMX module.
baudRate	The desired baud rate in bits per second.

Returns

Whether initialization was successful or not.

16.2.4.2 debug_console_status_t DbgConsole_DeInit (void)

Call this function to disable debug log messages to be output via the specified UART/LPUART base address and at the specified baud rate.

Returns

Whether de-initialization was successful or not.

16.2.4.3 int debug_printf (const char * *fmt_s*, ...)

Call this function to print formatted output to the standard output stream.

Parameters

fmt_s	Format control string.

Returns

Returns the number of characters printed, or a negative value if an error occurs.

16.2.4.4 int debug_putchar (int *ch*)

Call this function to write a character to stdout.

Parameters

ch	Character to be written.

Returns

Returns the character written.

16.2.4.5 int debug_scanf (const char * *fmt_ptr*, ...)

Call this function to read formatted data from the standard input stream.

Parameters

fmt_ptr	Format control string.

Returns

Returns the number of fields successfully converted and assigned.

16.2.4.6 int debug_getchar (void)

Call this function to read a character from standard input.

Returns

Returns the character read.

How to Reach Us:

Home Page:
nxp.com

Web Support:
nxp.com/support

Information in this document is provided solely to enable system and software implementers to use Freescale products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document.

Freescale reserves the right to make changes without further notice to any products herein. Freescale makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in Freescale data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including “typicals,” must be validated for each customer application by customer’s technical experts. Freescale does not convey any license under its patent rights nor the rights of others. Freescale sells products pursuant to standard terms and conditions of sale, which can be found at the following address: nxp.com/SalesTermsandConditions.

Document Number: FRTOS6XAPIRM
Rev. 0
07/2016