

Renesas Flexible Software Package (FSP) v0.8.0

User's Manual

Renesas RA Family

All information contained in these materials, including products and product specifications, represents information on the product at the time of publication and is subject to change by Renesas Electronics Corp. without notice. Please review the latest information published by Renesas Electronics Corp. through various means, including the Renesas Electronics Corp. website (<http://www.renesas.com>).

Table of Contents

Chapter 1 Introduction	6
1.1 Overview	6
1.2 How to Read this Manual	6
1.3 Documentation Standard	6
Chapter 2 Starting Development	8
2.1 Starting Development Introduction	8
2.1.1 Getting Started with the e2 studio ISDE and FSP	8
2.2 e2 studio ISDE User Guide	9
2.2.1 What is e2 studio ISDE?	9
2.2.2 e2 studio ISDE Prerequisites	11
2.2.2.1 Obtaining an RA MCU Kit	11
2.2.2.2 PC Requirements	11
2.2.2.3 Installing e2 studio, platform installer and the FSP package	11
2.2.2.4 Choosing a Toolchain	11
2.2.2.5 Licensing	12
2.2.3 What is a Project?	12
2.2.4 Creating a Project	13
2.2.4.1 Creating a New Project	14
2.2.4.2 Selecting a Board and Toolchain	15
2.2.4.3 Selecting a Project Template	16
2.2.5 Configuring a Project	17
2.2.5.1 Configuring the BSP with the ISDE	18
2.2.5.2 Configuring Clocks	19
2.2.5.3 Configuring Pins	19
2.2.5.4 Configuring Interrupts	22
2.2.5.5 Viewing Event Links	23
2.2.6 Adding Threads and Drivers	24
2.2.6.1 Adding and Configuring HAL Drivers	25
2.2.6.2 Adding Drivers to a Thread and Configuring the Drivers	26
2.2.6.3 Configuring Threads	29
2.2.7 Reviewing and Adding Components	30
2.2.8 Writing the Application	30
2.2.8.1 Coding Features	30
2.2.8.2 RTOS-independent Applications	36
2.2.8.3 RTOS Applications	37
2.2.9 Debugging the Project	38
2.2.10 Modifying Toolchain Settings	39
2.2.11 Importing an Existing Project into e2 studio ISDE	40
2.3 Tutorial: Your First RA MCU Project - Blinky	44
2.3.1 Tutorial Blinky	44
2.3.2 What Does Blinky Do?	44
2.3.3 Prerequisites	44
2.3.4 Create a New Project for Blinky	44
2.3.4.1 Details about the Blinky Configuration	47
2.3.4.2 Configuring the Blinky Clocks	47
2.3.4.3 Configuring the Blinky Pins	47
2.3.4.4 Configuring the Parameters for Blinky Components	47
2.3.4.5 Where is main()?	47
2.3.4.6 Blinky Example Code	47
2.3.5 Build the Blinky Project	48
2.3.6 Debug the Blinky Project	48

2.3.6.1 Debug prerequisites	49
2.3.6.2 Debug steps	49
2.3.6.3 Details about the Debug Process	50
2.3.7 Run the Blinky Project	51
2.4 Tutorial: Using HAL Drivers - Programming the WDT	51
2.4.1 Application WDT	51
2.4.2 Creating a WDT Application Using the RA MCU FSP and ISDE	51
2.4.2.1 Using the FSP and the e2 studio ISDE	51
2.4.2.2 The WDT Application	51
2.4.2.3 WDT Application flow	52
2.4.3 Creating the Project with the ISDE	52
2.4.4 Configuring the Project with the ISDE	55
2.4.4.1 BSP Tab	56
2.4.4.2 Clocks Tab	56
2.4.4.3 Pins Tab	57
2.4.4.4 Stacks Tab	57
2.4.4.5 Components Tab	59
2.4.5 WDT Generated Project Files	60
2.4.5.1 WDT hal_data.h	61
2.4.5.2 WDT hal_data.c	62
2.4.5.3 WDT main.c	63
2.4.5.4 WDT hal_entry.c	64
2.4.6 Building and Testing the Project	66
Chapter 3 FSP Architecture	69
3.1 FSP Architecture Overview	69
3.1.1 C99 Use	69
3.1.2 Doxygen	69
3.1.3 Weak Symbols	69
3.1.4 Memory Allocation	69
3.1.5 FSP Terms	69
3.2 FSP Modules	71
3.3 FSP Stacks	72
3.4 FSP Interfaces	72
3.4.1 FSP Interface Enumerations	73
3.4.2 FSP Interface Callback Functions	73
3.4.3 FSP Interface Data Structures	75
3.4.3.1 FSP Interface Configuration Structure	75
3.4.3.2 FSP Interface API Structure	76
3.4.3.3 FSP Interface Instance Structure	79
3.5 FSP Instances	79
3.5.1 FSP Instance Control Structure	80
3.5.2 FSP Interface Extensions	80
3.5.2.1 FSP Extended Configuration Structure	80
3.5.3 FSP Instance API	81
3.6 FSP API Standards	81
3.6.1 FSP Function Names	81
3.6.2 Use of const in API parameters	81
3.6.3 FSP Version Information	81
3.7 FSP Build Time Configurations	82
3.8 FSP File Structure	83
3.9 FSP Architecture in Practice	83
3.9.1 FSP Connecting Layers	83
3.9.2 Using FSP Modules in an Application	84
3.9.2.1 Create a Module Instance in the RA Configuration Tool	84
3.9.2.2 Use the Instance API in the Application	84

Chapter 4 API Reference	86
4.1 BSP	90
4.1.1 Common Error Codes	90
4.1.2 MCU Board Support Package	101
4.1.2.1 RA2A1	108
4.1.2.2 RA4M1	111
4.1.2.3 RA6M1	114
4.1.2.4 RA6M2	117
4.1.2.5 RA6M3	120
4.1.3 BSP I/O access	123
4.2 Modules	134
4.2.1 High-Speed Analog Comparator (r_acmphs)	139
4.2.2 Low-Power Analog Comparator (r_acmplp)	140
4.2.3 Analog to Digital Converter (r_adc)	142
4.2.4 Asynchronous General Purpose Timer (r_agt)	143
4.2.5 Clock Frequency Accuracy Measurement Circuit (r_cac)	145
4.2.6 Clock Generation Circuit (r_cgc)	146
4.2.7 Cyclic Redundancy Check (CRC) Calculator (r_crc)	148
4.2.8 Capacitive Touch Sensing Unit (r_cts)	150
4.2.9 Digital to Analog Converter (r_dac)	151
4.2.10 Direct Memory Access Controller (r_dmac)	152
4.2.11 Data Operation Circuit (r_doc)	155
4.2.12 D/AVE 2D Port Interface (r_drw)	156
4.2.13 Data Transfer Controller (r_dtc)	157
4.2.14 Event Link Controller (r_elc)	162
4.2.15 Ethernet (r_ether)	164
4.2.16 Ethernet PHY (r_ether_phy)	167
4.2.17 High-Performance Flash Driver (r_flash_hp)	170
4.2.18 Low-Power Flash Driver (r_flash_lp)	172
4.2.19 Graphics LCD Controller (r_glcdc)	174
4.2.20 General PWM Timer (r_gpt)	176
4.2.21 Interrupt Controller Unit (r_icu)	178
4.2.22 I2C Master on IIC (r_iic_master)	179
4.2.23 I2C Slave on IIC (r_iic_slave)	181
4.2.24 I/O Ports (r_ioport)	182
4.2.25 Independent Watchdog Timer (r_iwdt)	184
4.2.26 JPEG Codec (r_jpeg)	185
4.2.27 Key Interrupt (r_kint)	188
4.2.28 Low Power Modes (r_lpm)	189
4.2.29 Low Voltage Detection (r_lvd)	190
4.2.30 Realtime Clock (r_rtc)	191
4.2.31 Serial Communications Interface (SCI) I2C (r_sci_i2c)	193
4.2.32 Serial Communications Interface (SCI) SPI (r_sci_spi)	195
4.2.33 Serial Communications Interface (SCI) UART (r_sci_uart)	196
4.2.34 SD/MMC Host Interface (r_sdhi)	198
4.2.35 Serial Peripheral Interface (r_spi)	199
4.2.36 Serial Sound Interface (r_ssi)	201
4.2.37 Universal Serial Bus (r_usb_basic)	202
4.2.38 Host Mass Storage Class Driver (r_usb_hmsc)	208
4.2.39 Universal Serial Bus Peripheral Communication Device Class (r_usb_pcdc)	213
4.2.40 Watchdog Timer (r_wdt)	218
4.2.41 SEGGER emWin Port (rm_emwin_port)	220
4.2.42 FreeRTOS Plus FAT (rm_freertos_plus_fat)	222
4.2.43 Amazon FreeRTOS Port (rm_freertos_port)	224

4.2.44 Crypto Middleware (rm_psa_crypto)	243
4.2.45 Capacitive Touch Middleware (rm_touch)	258
4.3 Interfaces	259
4.3.1 ADC Interface	262
4.3.2 CAC Interface	281
4.3.3 CGC Interface	290
4.3.4 Comparator Interface	303
4.3.5 CRC Interface	312
4.3.6 CTSU Interface	317
4.3.7 DAC Interface	330
4.3.8 Display Interface	335
4.3.9 DOC Interface	353
4.3.10 ELC Interface	358
4.3.11 Ethernet Interface	363
4.3.12 Ethernet PHY Interface	372
4.3.13 External IRQ Interface	378
4.3.14 Flash Interface	384
4.3.15 I2C Master Interface	400
4.3.16 I2C Slave Interface	408
4.3.17 I2S Interface	415
4.3.18 I/O Port Interface	427
4.3.19 JPEG Codec Interface	441
4.3.20 Key Matrix Interface	453
4.3.21 Low Power Modes Interface	459
4.3.22 Low Voltage Detection Interface	473
4.3.23 RTC Interface	483
4.3.24 SD/MMC Interface	494
4.3.25 SPI Interface	510
4.3.26 Timer Interface	521
4.3.27 Transfer Interface	533
4.3.28 UART Interface	545
4.3.29 USB Interface	555
4.3.30 USB HMSC Interface	580
4.3.31 USB PCDC Interface	584
4.3.32 WDT Interface	585
4.3.33 Touch Middleware Interface	594

Chapter 1 Introduction

1.1 Overview

This manual describes how to use the Renesas Flexible Software Package (FSP) for writing applications for the RA microcontroller series.

1.2 How to Read this Manual

For help getting started with the FSP, see:

- [Starting Development](#)

To learn about the FSP architecture and about board and chip-level support included in the FSP, see:

- [FSP Architecture](#)
- [MCU Board Support Package](#)

For user guides describing the FSP modules, see:

- [Modules](#)

For shared interface API documentation, see:

- [Interfaces](#)

1.3 Documentation Standard

Each module user guide outlines the following:

- **Features:** A bullet list of high level features provided by the module.
- **Configuration:** A description of module specific configurations available in the configuration tool.
- **Usage Notes:** Module specific documentation and limitations.
- **Examples:** Example code provided to help the user get started.
- **API Reference:** Usage notes for each API in the module, including the function prototype and hyperlinks to the interface documentation for parameter definitions.

Interface documentation includes typed enumerations and structures—including a structure of function pointers that defines the API—that are shared by all modules that implement the interface.

Introduction to FSP

Purpose

The Renesas Flexible Software Package (FSP) is an optimized software package designed to provide easy to use, scalable, high quality software for embedded system design. The primary goal is to

provide lightweight, efficient drivers that meet common use cases in embedded systems.

Quality

FSP code quality is enforced by peer reviews, automated requirements-based testing, and automated static analysis.

Ease of Use

The FSP provides uniform and intuitive APIs that are well documented. Each module is supported with detailed user documentation including example code.

Scalability

FSP modules can be used on any MCU in the RA family, provided the MCU has any peripherals required by the module.

FSP modules also have build time configurations that can be used to optimize the size of the module for the feature set required by the application.

Chapter 2 Starting Development

2.1 Starting Development Introduction

The Renesas Flexible Software Package (FSP) provides a host of efficiency enhancing tools for developing projects targeting the Renesas RA series of MCU devices. The e² studio Integrated System Development Environment (ISDE) provides a familiar development cockpit from which the key steps of project creation, module selection and configuration, code development, code generation, and debugging are all managed. FSP runs within e² studio and enables the module selection, configuration, and code generation steps. FSP uses a Graphical User Interface (GUI) to simplify the selection, configuration, code generation and code development of high level modules and their associated Application Program Interfaces (APIs) to dramatically accelerate the development process.

The wealth of resources available to learn about and use e² studio and FSP can be overwhelming on first inspection, so the following section provides a Getting Started Guide with a list of the most important first steps. Following these highly recommended first 10 steps will bring you up to speed on the development environment in record time. Even experienced developers can benefit from the use of this guide, to learn the terminology that might be unfamiliar or different from previous environments.

2.1.1 Getting Started with the e² studio ISDE and FSP

This section describes how to use the Renesas e² Integrated Solutions Development Environment (ISDE) to develop applications with the Renesas Flexible Software Package (FSP). Here is the recommended sequence for quickly Getting Started with using e² when developing with the RA MCU Family:

1. Read over the section [What is e² studio ISDE?](#), up to but not including [e² studio ISDE Prerequisites](#). This will provide a description of the various windows and views to use e² to create a project, add modules and threads, configure module properties, add code, and debug a project. It also describes how to use key coding 'accelerators' like Developer Assist (to drag and drop parameter populated API function calls right into your code), a context aware Autocomplete (to easily find and select from suggested enumerations, functions, types, and many other coding elements), and many other similar productivity enhancers.
2. Read over the FSP Architecture sections [FSP Architecture](#), [FSP Modules](#) and [FSP Stacks](#). These provide the basic background on how FSP modules and stacks are used to construct your application. Understanding their definitions and the theory behind how they combine will make it easier to develop with FSP.
3. Read over a few "Module User Guide" sections to see how to use API function calls, structures, enumerations, types and callbacks. These user guides provide the information you will use to implement your project code. (Much of the details are provided with Developer Assistance, covered in step 5, below.
4. If you don't have a kit, you can order one using the link included in the [e² studio ISDE Prerequisites](#) section. Then, if you haven't yet downloaded and installed e² studio and FSP, use the link included in the [e² studio ISDE Prerequisites](#) section to download the tools. Then you can build and debug a simple project to prove out you installation, tool flow, and the kit. The simple "Blinky" project, that blinks an LED on and off, is located in the [Tutorial: Your First RA MCU Project - Blinky](#) section. Follow the instructions for importing and running this

project. It will use some of the key steps for managing projects within e² and is a good way to learn the basics.

5. Once you have successfully run Blinky you have a good starting point for using FSP for more complex projects. The Watchdog Timer hands-on lab, available in the [Tutorial: Using HAL Drivers - Programming the WDT](#) section, shows how to create a project from scratch and use FSP API functions, and demonstrates the use of some of the coding efficiency tools like Developer Assistance and Autocomplete. Run through this lab to establish a good starting point for developing custom projects.
6. The balance of the [FSP Architecture](#) sections, those not called out in step 2 above, contain additional reference material that may be helpful in the future. Scan them over so you know what they contain, in case you need them.
7. The balance of the e² ISDE User Guide, starting with the [What is a Project?](#) section up to [Writing the Application](#) section, provides a detailed description of each of the key steps, windows, and entries used to create, manage, configure, build and debug a project. Most of this will be familiar after doing the Blinky and WDT exercises from steps 4 and 5 above. Skim over these references so you know to come back to them when questions come up. Make sure you have a good grasp of what each of the configuration tabs are used for since that is where the bulk of the project preparation work takes place prior to writing code.
8. Read over the [Writing the Application](#) section to get a short introduction to the steps used when creating application code with FSP. It covers both RTOS-independent and RTOS-dependent applications. The [Tutorial: Using HAL Drivers - Programming the WDT](#) section is a good introduction to the key steps for an RTOS-independent application. Make sure you have run through it at least once before doing a custom project.
9. Scan the [Debugging the Project](#) section to see the steps required to download and start a debug session.
10. Explore the additional material available on the following web pages and bookmark the resources that look most valuable to you:
 - a. RA Landing Page: <https://www.renesas.com/ra>
 - b. FSP Landing Page: <https://www.renesas.com/fsp>

2.2 e2 studio ISDE User Guide

2.2.1 What is e2 studio ISDE?

The Renesas e² studio ISDE, or Integrated Solution Development Environment, is a development tool encompassing code development, build, and debug. The ISDE is based on the open-source Eclipse IDE and the associated C/C++ Development Tooling (CDT).

When developing for RA MCUs, the ISDE hosts the Renesas Flexible Software Package (FSP). FSP provides a wide range of time saving tools to simplify the selection, configuration, and management of modules and threads, to easily implement complex applications. The time saving tools available in e² studio and FSP include the following:

- A Graphical User Interface (GUI) (see [Adding Threads and Drivers](#)) with numerous wizards for configuring and auto-generating code
- A context sensitive Autocomplete (see [Tutorial: Using HAL Drivers - Programming the WDT](#)) feature that provides intelligent options for completing a programming element
- A [Developer Assistance](#) tool for selection of and drag and drop placement of API functions directly in application code
- A [Smart Manual](#) provides driver and device documentation in the form of tooltips right in the code
- An [Edit Hover](#) feature to show detailed descriptions of code elements while editing
- A [Welcome Window](#) with links to example projects, application notes and a variety of other self-help support resources

- An **Information Icon**, from each module, is provided in the graphic configuration viewer that links to specific design resources, including code 'cheat sheets' that provide useful starting points for common application implementations.



Figure 1: e2 studio Splash Screen

The e² studio ISDE organizes project work based on Perspectives, Views, Windows, Panes, and Pages (sometimes called Tabs). Windows are a section of the ISDE GUI that presents information on a key topic. Windows often use tabs to select sub-topics. For example, an editor window might have a tab available for each open file, so it is easy to switch back and forth between them. A window Pane is a section of a window. Within a window, multiple Panes can be opened and viewed simultaneously, as opposed to a tabbed window, where only individual content is displayed. A memory-display Window, for example, might have multiple Panes that allow the data to be displayed in different formats, simultaneously. A Perspective is a collection of Views and Windows typical for a specific stage of development. The default perspectives are a C/C++ Perspective, an FSP Configuration Perspective and a Debug Perspective. These provide specific Views, Windows, Tabs, and Panes tailored for the common tasks needed during the specific development stage. These three default perspectives are each illustrated in the below screen shots, along with graphic indicators helpful in identifying example Views, Windows, Tabs and Panes.

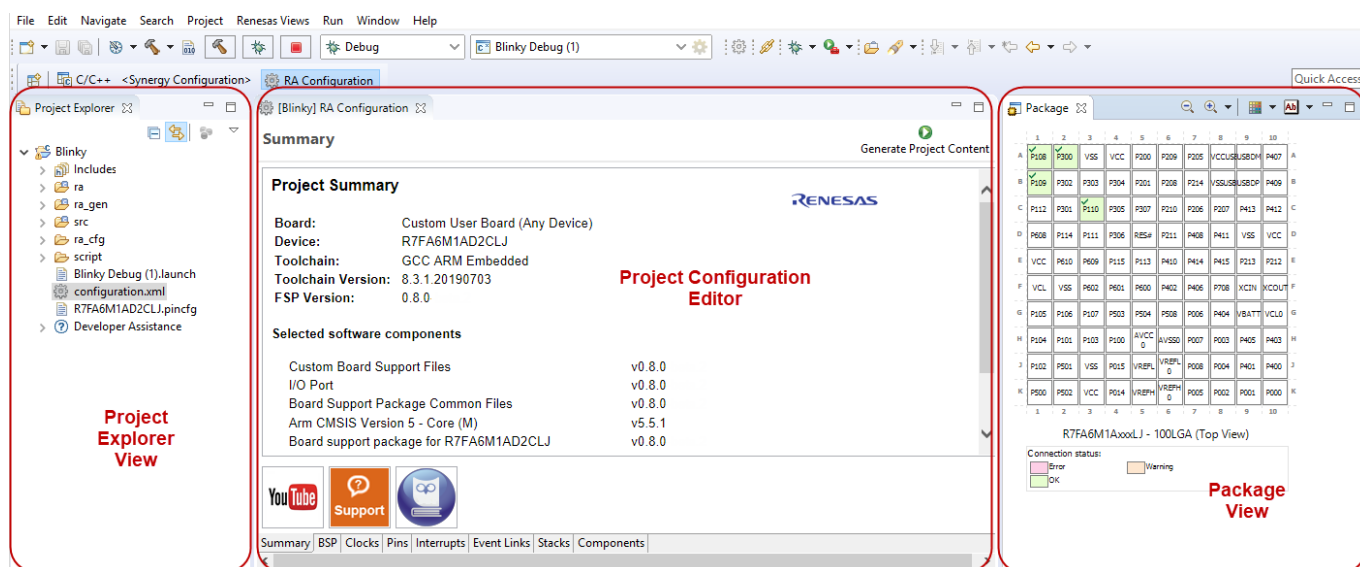


Figure 2: Default Perspective

In addition to managing project development, selecting modules, configuring them and simplifying code development, e² studio also hosts the engine for automatically generating code based on module selections and configurations. The engine continually checks for dependencies and automatically adds any needed lower level modules to the module stack. It also identifies any lower level modules that require configuration (for example, an interrupt that needs to have a priority assigned). It also provides a guide for selecting between multiple choices or options to make it easy to complete a fully functional module stack.

The Generate Project Content function takes the selected and configured modules and automatically generates the complete and correct configuration code. The code is added to the folders visible in the **Project Explorer** window in e² studio. The configuration.xml file in the project folder holds all the configuration settings generated by the ISDE. This file can be opened in the GUI-based configuration editor to make further edits and changes. Once a project has been generated, you can go back and reconfigure any of the modules and settings if required using this editor.

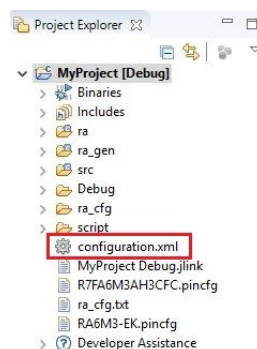


Figure 3: Project Explorer Window showing generated folders and configuration.xml file

2.2.2 e2 studio ISDE Prerequisites

2.2.2.1 Obtaining an RA MCU Kit

To develop applications with FSP, start with one of the Renesas RA MCU Evaluation Kits. The Renesas RA MCU Evaluation Kits are designed to seamlessly integrate with the e² studio ISDE.

Ordering information, Quick Start Guides, User Manuals, and other related documents for all RA MCU Evaluation Kits are available at <https://www.renesas.com/ra>.

2.2.2.2 PC Requirements

The following are the minimum PC requirements to use the e² studio ISDE:

- Windows 10 with Intel i5 or i7, or AMD A10-7850K or FX
- Memory: 8-GB DDR3 or DDR4 DRAM (16-GB DDR4/2400-MHz RAM is preferred)
- Minimum 250-GB hard disk

2.2.2.3 Installing e2 studio, platform installer and the FSP package

Detailed installation instructions for the e² studio ISDE and the FSP are available on the Renesas website <https://www.renesas.com/fsp>. Review the release notes for e² studio to ensure that the e² studio version supports the selected FSP version. The starting version of the installer includes all features of the RA MCUs.

2.2.2.4 Choosing a Toolchain

The e² studio ISDE can work with several toolchains and toolchain versions such as the GNU ARM compiler, AC6. A version of the GNU ARM compiler is included in the e² studio installer and has been verified to run with the FSP version.

2.2.2.5 Licensing

FSP licensing includes full source code, limited to Renesas hardware only.

2.2.3 What is a Project?

In e² studio, all FSP applications are organized in RA MCU projects. Setting up an RA MCU project involves:

1. Creating a Project
2. Configuring a Project

These steps are described in detail in the next two sections. When you have existing projects already, after you launch e² studio and select a workspace, all projects previously saved in the selected workspace are loaded and displayed in the **Project Explorer** window. Each project has an associated configuration file named `configuration.xml`, which is located in the project's root directory.

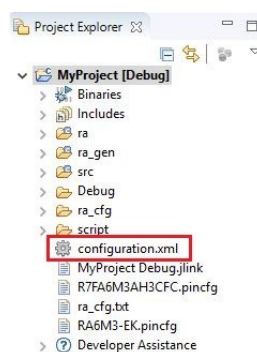


Figure 4: e2 studio Project Configuration file

Double-click on the `configuration.xml` file to open the RA MCU Project Editor. To edit the project configuration, make sure that the **RA Configuration** perspective is selected in the upper right hand corner of the e² studio window. Once selected, you can use the editor to view or modify the configuration settings associated with this project.



Figure 5: e2 studio RA Configuration Perspective

Note

Whenever the RA project configuration (that is, the `configuration.xml` file) is saved, a verbose RA Project Report file (`ra_cfg.txt`) with all the project settings is generated. The format allows differences to be easily viewed using a text comparison tool. The generated file is located in the project root directory.

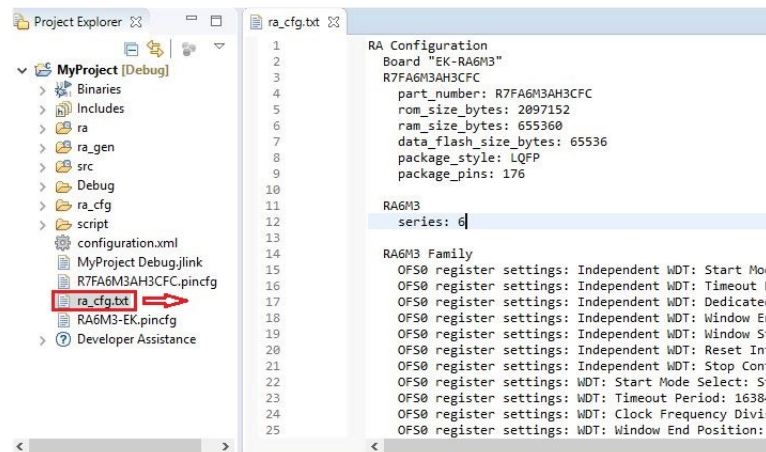


Figure 6: RA Project Report

The RA Project Editor has a number of tabs. The configuration steps and options for individual tabs are discussed in the following sections.

Note

Which tabs are available with the RA Project Editor depends on the e² studio version.



Figure 7: RA Project Summary tabs

- Click on the YouTube icon to visit the Renesas FSP playlist on YouTube
- Click on the Support icon to visit RA support pages at Renesas.com
- Click on the user manual (owl) icon to open the RA software package User's Manual

2.2.4 Creating a Project

During project creation, you specify the type of project, give it a project name and location, and configure the project settings for version, target board, whether an RTOS is included, the toolchain version, and the beginning template. This section includes easy-to-follow step-by-step instructions for all of the project creation tasks. Once you have created the project, you can move to configuring the project hardware (clocks, pins, interrupts) and the parameters of all the modules that are part of

your application.

2.2.4.1 Creating a New Project

For RA MCU applications, generate a new project using the following steps:

1. Click on **File > New > RA C/C++ Project**.

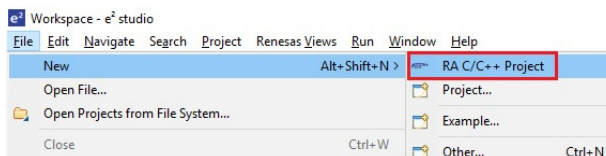


Figure 8: New RA MCU Project

Then click on the type of template for the type of project you are creating.

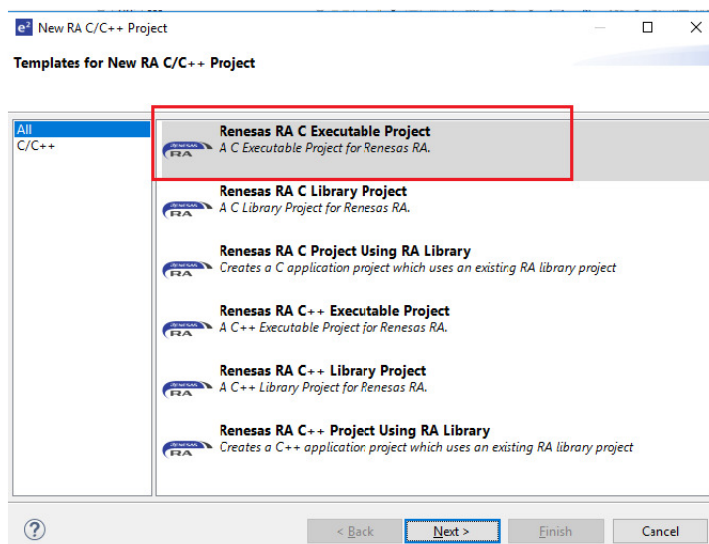


Figure 9: New Project Templates

2. Select a project name and location.

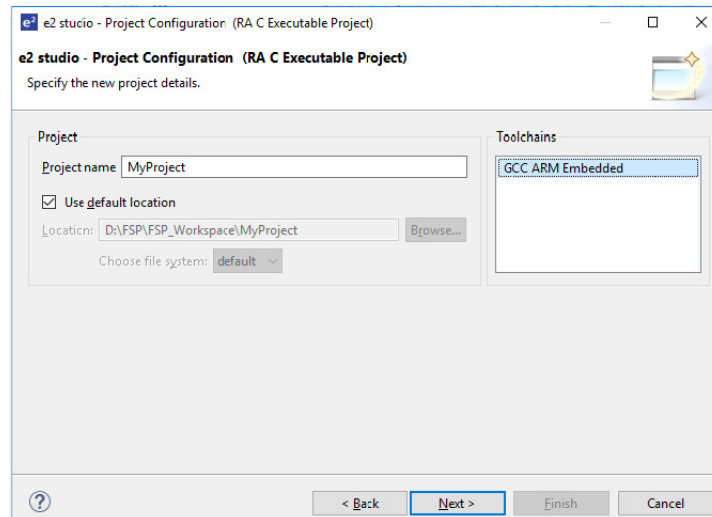


Figure 10: RA MCU Project Generator (Screen 1)

3. Click **Next**.

2.2.4.2 Selecting a Board and Toolchain

In the **Project Configuration** window select the hardware and software environment:

1. Select the **FSP version**.
2. Select the **Board** for your application. You can select an existing RA MCU Evaluation Kit or select **Custom User Board** for any of the RA MCU devices with your own BSP definition.
3. Select the **Device**. The **Device** is automatically populated based on the **Board** selection. Only change the **Device** when using the **Custom User Board (Any Device)** board selection.
4. To add threads, select **RTOS**, or **No RTOS** if an RTOS is not being used.
5. The **Toolchain** selection defaults to **GCC ARM Embedded**.
6. Select the **Toolchain version**. This should default to the installed toolchain version.
7. Select the **Debugger**. The J-Link ARM Debugger is preselected.

8. Click **Next**.

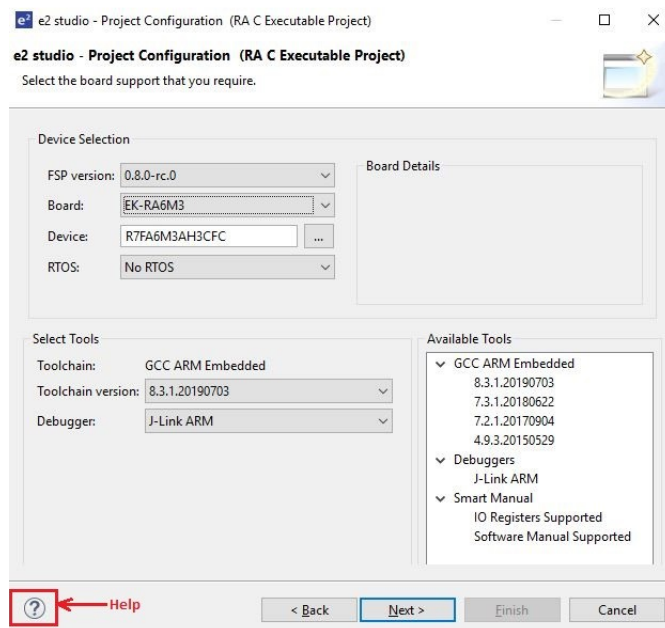


Figure 11: RA MCU Project Generator (Screen 2)

Click on the **Help** icon (?) for user guides, RA contents, and other documents.

2.2.4.3 Selecting a Project Template

In the next window, select a project template from the list of available templates. By default, this screen shows the templates that are included in your current RA MCU pack. Once you have selected the appropriate template, click **Finish**.

Note

*If you want to develop your own application, select the basic template for your board, **Bare Metal - Minimal**.*

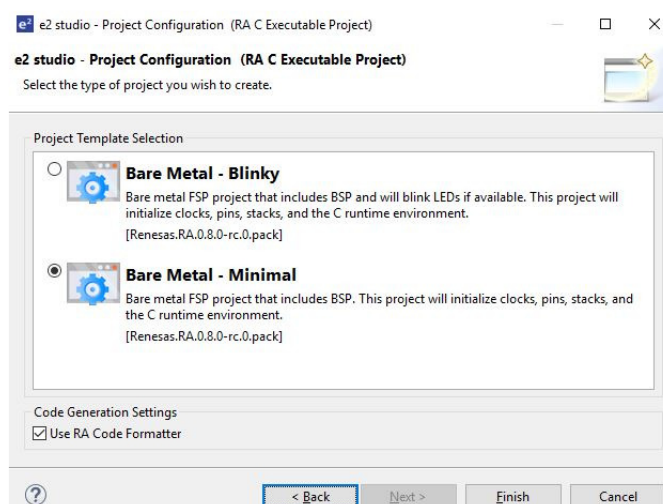


Figure 12: RA MCU Project Generator (Screen 3)

When the project is created, the ISDE displays a summary of the current project configuration in the

RA MCU Project Editor.



Figure 13: RA MCU Project Editor and available editor tabs

On the bottom of the RA MCU Project Editor view, you can find the tabs for configuring multiple aspects of your project:

- With the **BSP** tab, you can change board specific parameters from the initial project selection.
- With the **Clocks** tab, you can configure the MCU clock settings for your project.
- With the **Pins** tab, you can configure the electrical characteristics and functions of each port pin.
- With the **Stacks** tab, you can add FSP modules for non-RTOS applications and configure the modules. For each module selected in this tab, the **Properties** window provides access to the configuration parameters, interrupt priorities, and pin selections.
- With the **Interrupt** tab, you can add new user events/interrupts.
- With the **Event Links** tab, you can configure events used by the Event Link Controller.
- The **Components** tab provides an overview of the selected modules. You can also add drivers for specific FSP releases and application sample code here.

The functions and use of each of these tabs is explained in detail in the next section.

2.2.5 Configuring a Project

Each of the configurable elements in an FSP project can be edited using the appropriate tab in the configuration editor window. Importantly, the initial configuration of the MCU after reset and before any user code is executed is set by the configuration settings in the **BSP**, **Clocks** and **Pins** tabs. When you select a project template during project creation, the ISDE configures default values that are appropriate for the associated board. You can change those default values as needed. The following sections detail the process of configuring each of the project elements for each of the associated tabs.



Figure 14: RA MCU Project Editor and available editor tabs

2.2.5.1 Configuring the BSP with the ISDE

The **BSP** tab shows the currently selected board (if any) and device. The Properties view is located in the lower left of the Project Configurations view as shown below.

Note

*If the Properties view is not visible, click **Window > Show View > Properties** in the top menu bar.*

EK-RA6M3		
Settings	Property	Value
	▼ R7FA6M3AH3CFC	
	part_number	R7FA6M3AH3CFC
	rom_size_bytes	2097152
	ram_size_bytes	655360
	data_flash_size_bytes	65536
	package_style	LQFP
	package_pins	176
	▼ RA6M3	
	series	6
	▼ RA6M3 Family	
	> OFS0 register settings	
	> OFS1 register settings	
	> MPU	
	▼ RA Common	
	Main stack size (bytes)	0x400
	Heap size (bytes) - A minimum of 4K	0
	MCU Vcc (mV)	3300
	Parameter checking	Disabled
	Assert Failures	Return FSP_ERR_ASSERTION
	Error Log	No Error Log
	ID Code Mode	Unlocked (Ignore ID)
	ID Code (32 Hex Characters)	FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
	Soft Reset	Disabled
	PFS Protect	Enabled
	Main Oscillator Wait Time	32768 us
	Main Oscillator Clock Source	Crystal or Resonator
	Subclock Populated	Populated

Figure 15: ISDE BSP tab

The **Properties** view shows the configurable options available for the BSP. These can be changed as required. The BSP is the FSP layer above the MCU hardware. The ISDE checks the entry fields to flag invalid entries. For example, only valid numeric values can be entered for the stack size.

When you click the **Generate Project Content** button, the BSP configuration contents are written to ra_cfg/fsp_cfg/bsp/bsp_cfg.h

This file is created if it does not already exist.

Warning

Do not edit this file as it is overwritten whenever the **Generate Project Content** button is clicked.

2.2.5.2 Configuring Clocks

The **Clocks** tab presents a graphical view of the MCU's clock tree, allowing the various clock dividers and sources to be modified. If a clock setting is invalid, the offending clock value is highlighted in red. It is still possible to generate code with this setting, but correct operation cannot be guaranteed. In the figure below, the USB clock HOCO has been changed so the resulting clock frequency is 24 MHz instead of the required 48 MHz. This parameter is colored red.

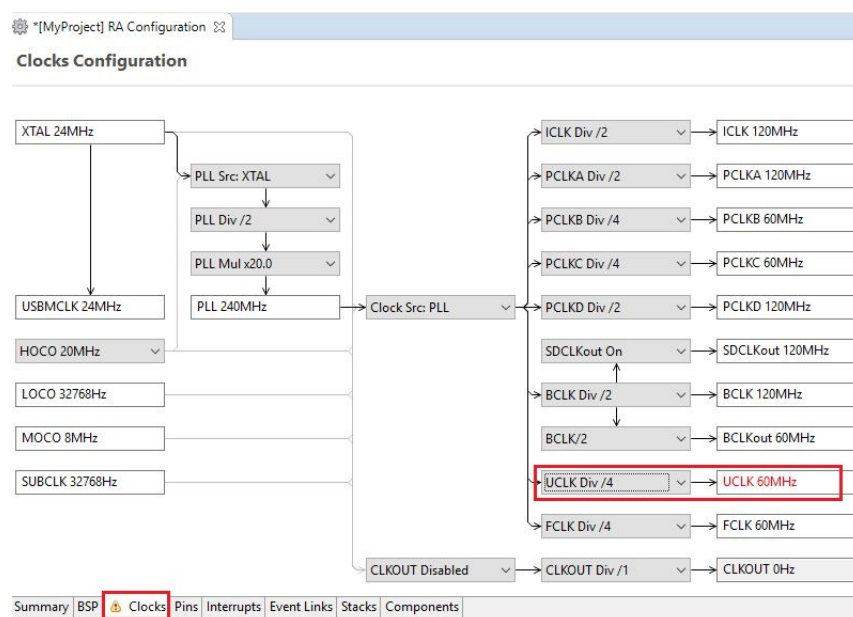


Figure 16: ISDE Clocks tab

When you click the **Generate Project Content** button, the clock configuration contents are written to: `ra_gen/bsp_clock_cfg.h`

This file will be created if it does not already exist.

Warning

Do not edit this file as it is overwritten whenever the **Generate Project Content** button is clicked.

2.2.5.3 Configuring Pins

The **Pins** tab provides flexible configuration of the MCU's pins. As many pins are able to provide multiple functions, they can be configured on a peripheral basis. For example, selecting a serial channel via the SCI peripheral offers multiple options for the location of the receive and transmit pins for that module and channel. Once a pin is configured, it is shown as green in the **Package** view.

Note

If the **Package** view window is not open in the ISDE, select **Window > Show View > Pin Configurator > Package**

from the top menu bar to open it.

The **Pins** tab simplifies the configuration of large packages with highly multiplexed pins by highlighting errors and presenting the options for each pin or for each peripheral. If you selected a project template for a specific board such as the RA6M3, some peripherals connected on the board are preselected.

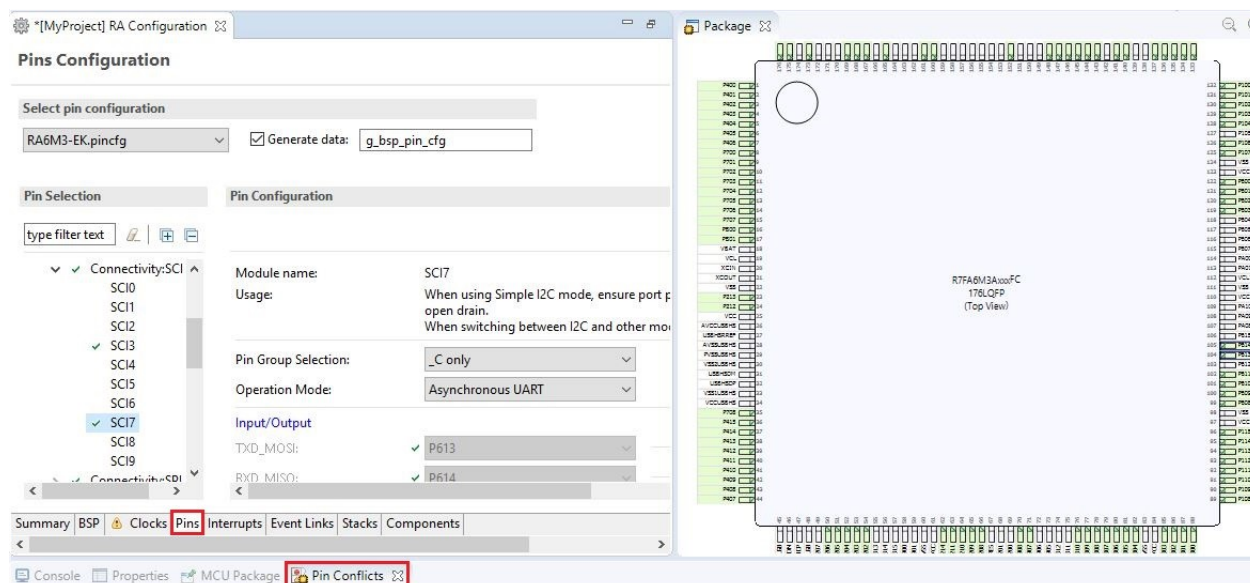


Figure 17: Pins Configuration

The pin configurator includes a built-in conflict checker, so if the same pin is allocated to another peripheral or I/O function the pin will be shown as red in the package view and also with white cross in a red square in the **Pin Selection** pane and **Pin Configuration** pane in the main **Pins** tab. The **Pin Conflicts** view provides a list of conflicts, so conflicts can be quickly identified and fixed.

In the example shown below, port P611 is already used by the CAC, and the attempt to connect this port to the Serial Communications Interface (SCI) results in a dangling connection error. To fix this error, select another port from the pin drop-down list or disable the CAC in the **Pin Selection** pane on the left side of the tab.

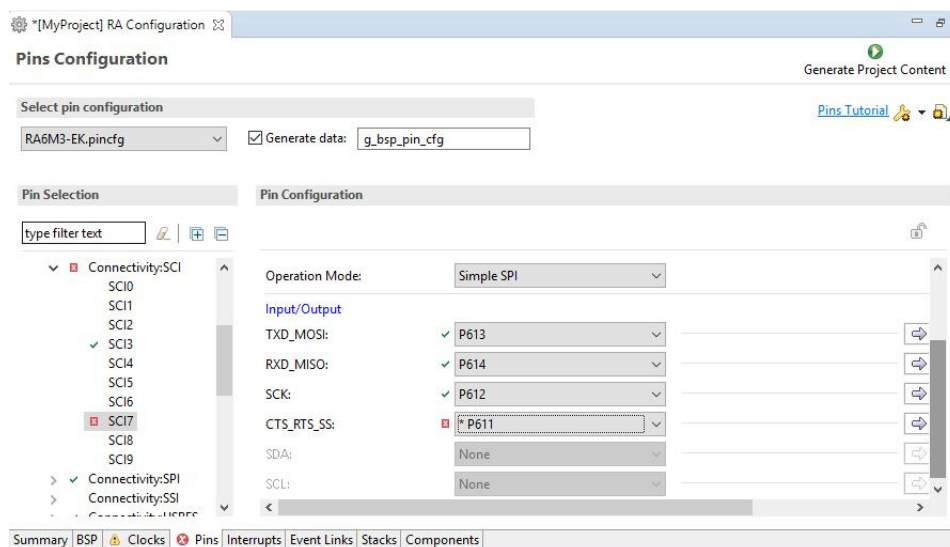


Figure 18: ISDE Pin configurator

The pin configurator also shows a package view and the selected electrical or functional characteristics of each pin.

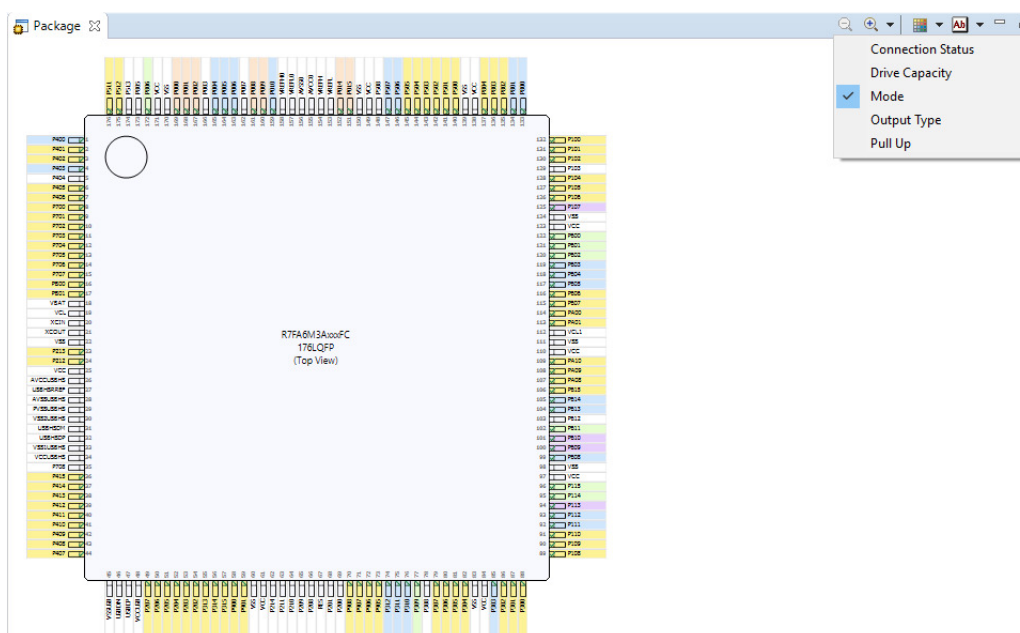


Figure 19: ISDE Pin configurator package view

When you click the **Generate Project Content** button, the pin configuration contents are written to: ra_gen\bsp_pin_cfg.h

This file will be created if it does not already exist.

Warning

Do not edit this file as it is overwritten whenever the **Generate Project Content** button is clicked.

To make it easy to share pinning information for your project, the ISDE exports your pin configuration settings to a csv format and copies the csv file to `ra_gen/<MCU package>.csv`.

2.2.5.4 Configuring Interrupts

You can use the **Properties** view in the **Stacks** tab to enable interrupts by setting the interrupt priority. Select the driver in the **Stacks** pane to view and edit its properties.

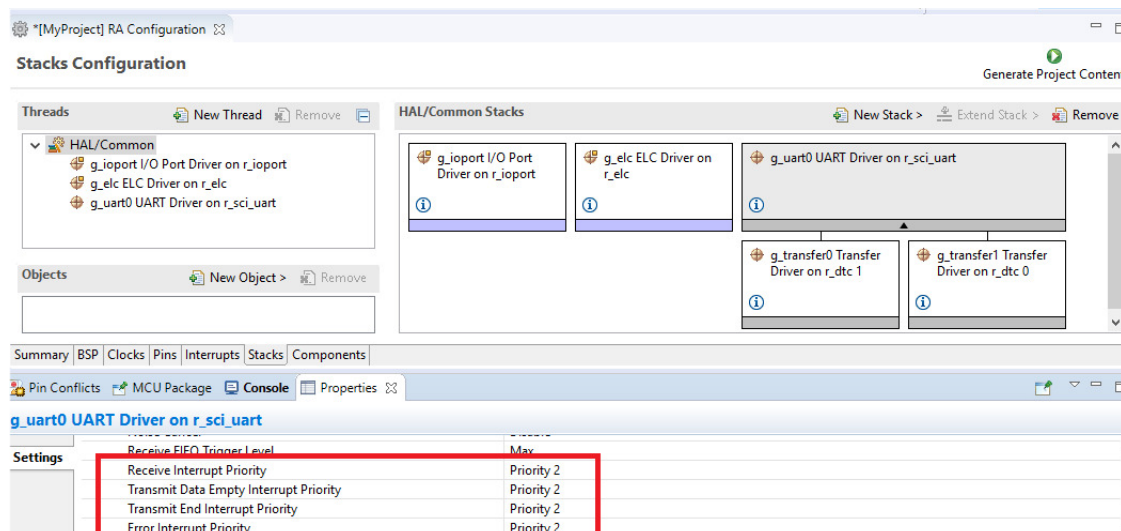


Figure 20: Configuring Interrupt on the Stacks tab

Interrupts

In the **Interrupt** tab, the user can bypass a peripheral interrupt and have user-defined ISRs for the peripheral interrupt. This can be done by adding a new event with the user define tab (**New User Event**).

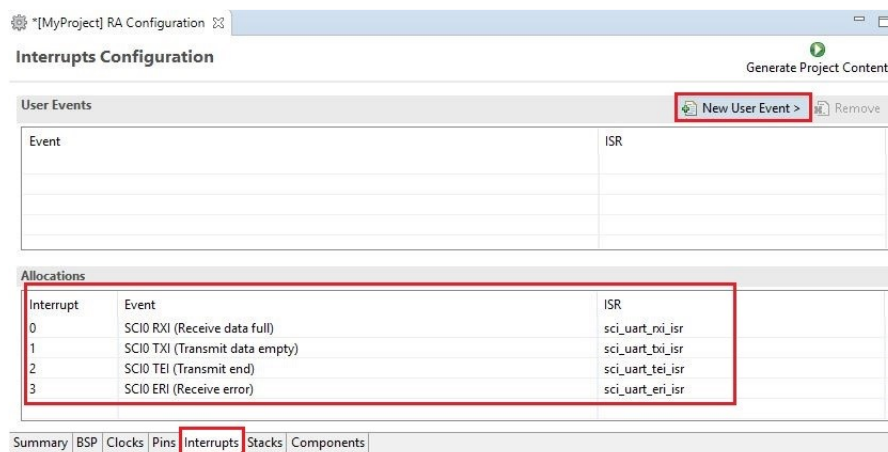


Figure 21: Configuring interrupt in Interrupt Tab

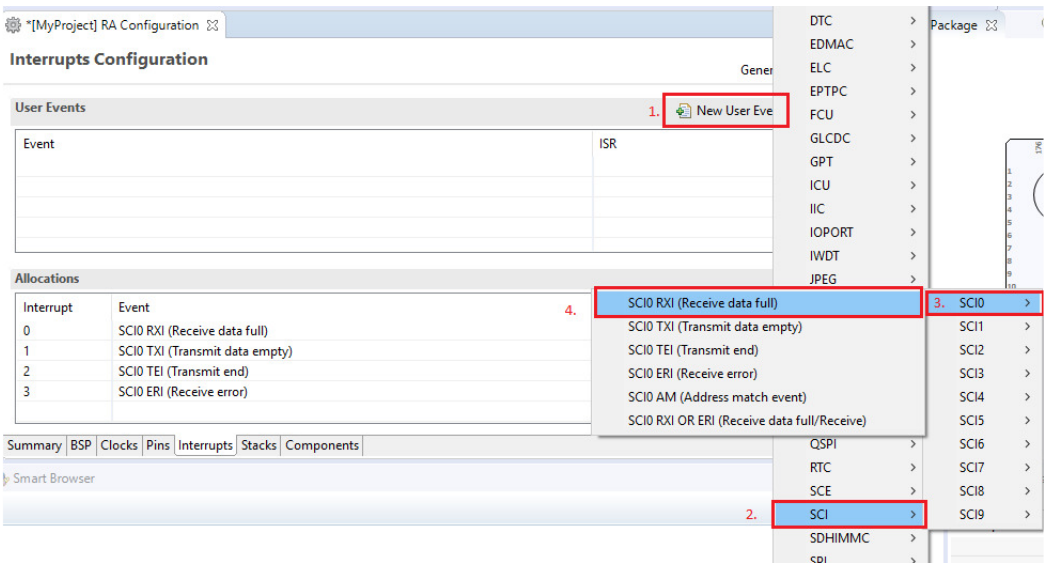


Figure 22: Adding user-defined event

Enter the name of ISR for new user event.

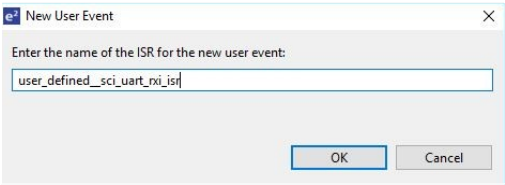


Figure 23: User-defined event ISR

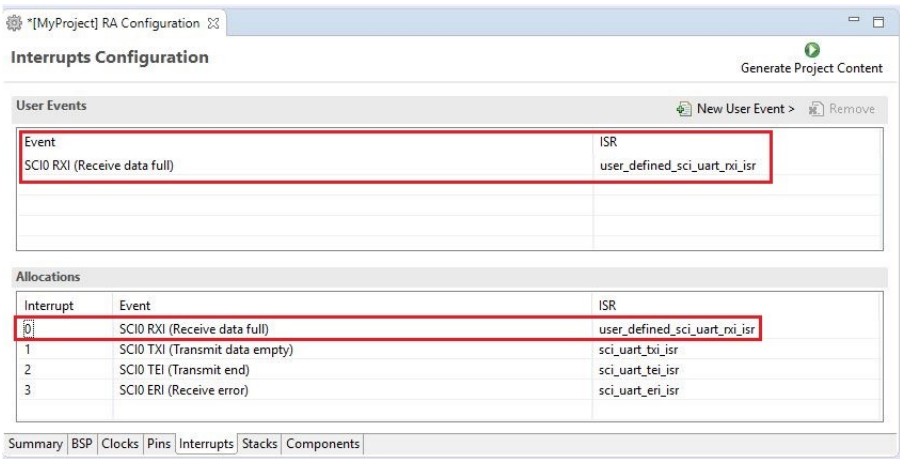


Figure 24: Using a user-defined event

2.2.5.5 Viewing Event Links

The Event Links tab can be used to view the Event Link Controller events. The events are sorted by peripheral to make it easy to find and verify them.

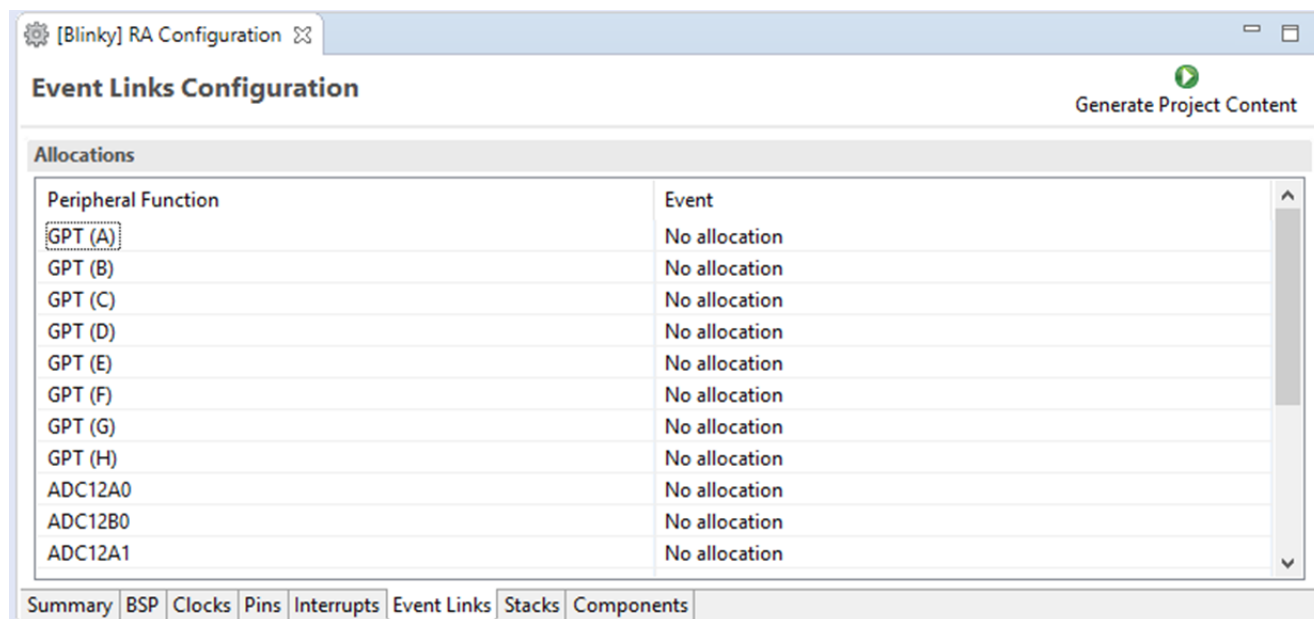


Figure 25: Viewing Event Links

2.2.6 Adding Threads and Drivers

Every FreeRTOS-based RA Project includes at least one RTOS Thread and a stack of FSP modules running in that thread. The **Stacks** tab is a graphical user interface which helps you to add the right modules to a thread and configure the properties of both the threads and the modules associated with each thread. Once you have configured the thread, the ISDE automatically generates the code reflecting your configuration choices.

For any driver, or, more generally, any module that you add to a thread, the ISDE automatically resolves all dependencies with other modules and creates the appropriate stack. This stack is displayed in the Stacks pane, which the ISDE populates with the selected modules and module options for the selected thread.

The default view of the **Stacks** tab includes a Common Thread called **HAL/Common**. This thread includes the driver for I/O control (IOPORT). The default stack is shown in the **HAL/Common Stacks** pane. The default modules added to the HAL/Common driver are special in that the FSP only requires a single instance of each, which the ISDE then includes in every user-defined thread by default.

In applications that do not use an RTOS or run outside of the RTOS, the HAL/Common thread becomes the default location where you can add additional drivers to your application.

For a detailed description on how to add and configure modules and stacks, see the following sections:

- [Adding and Configuring HAL Drivers](#)
- [Adding Drivers to a Thread and Configuring the Drivers](#)

Once you have added a module either to HAL/Common or to a new thread, you can access the driver's configuration options in the **Properties** view. If you added thread objects, you can access the objects configuration options in the **Properties** view in the same way.

You can find details about how to configure threads here: [Configuring Threads](#)

Note

Driver and module selections and configuration options are defined in the FSP pack and can therefore change when the FSP version changes.

2.2.6.1 Adding and Configuring HAL Drivers

For applications that run outside or without the RTOS, you can add additional HAL drivers to your application using the HAL/Common thread. To add drivers, follow these steps:

1. Click on the HAL/Common icon in the **Stacks** pane. The Modules pane changes to **HAL/Common Stacks**.

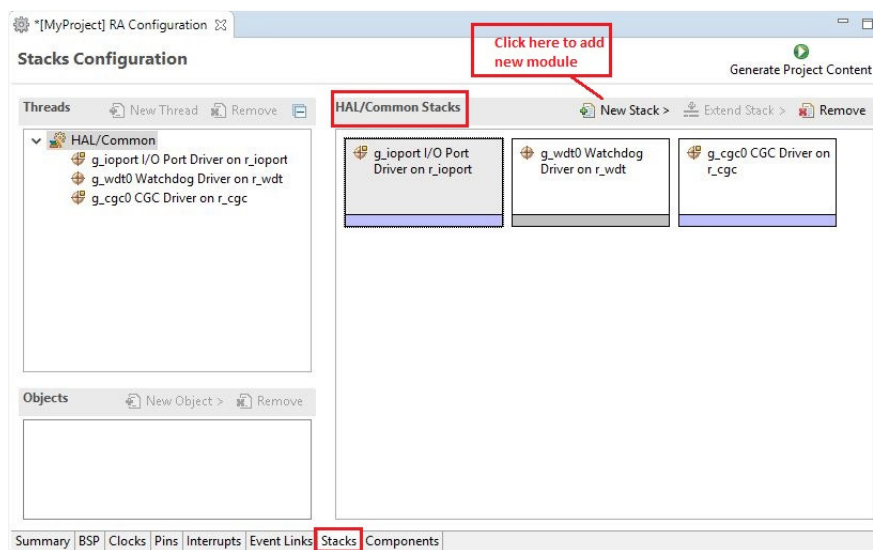


Figure 26: ISDE Project configurator - Adding drivers

2. Click **New Stack** to see a drop-down list of HAL level drivers available in the FSP.
3. Select a driver from the menu **New Stack > Driver**.

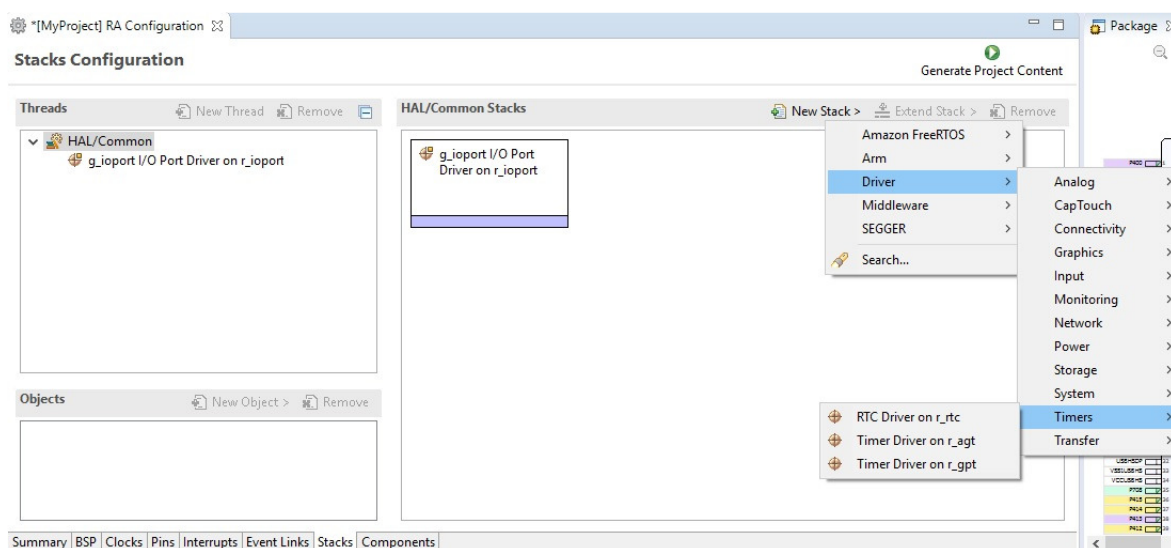


Figure 27: Select a driver

4. Select the driver module in the **HAL/Common Modules** pane and configure the driver properties in the **Properties** view.

The ISDE adds the following files when you click the **Generate Project Content** button:

- The selected driver module and its files to the ra/fsp directory
- The main() function and configuration structures and header files for your application as shown in the table below.

File	Contents	Overwritten by Generate Project Content?
ra_gen/main.c	Contains main() calling generated and user code. When called, the BSP already has Initialized the MCU.	Yes
ra_gen/hal_data.c	Configuration structures for HAL Driver only modules.	Yes
ra_gen/hal_data.h	Header file for HAL driver only modules.	Yes
src/hal_entry.c	User entry point for HAL Driver only code. Add your code here.	No

The configuration header files for all included modules are created or overwritten in this folder:
ra_cfg/fsp_cfg

2.2.6.2 Adding Drivers to a Thread and Configuring the Drivers

For an application that uses the RTOS, you can add one or more threads, and for each thread at least one module that runs in the thread. You can select modules from the Driver dropdown menu. To add modules to a thread, follow these steps:

1. In the **Threads** pane, click **New Thread** to add a Thread.

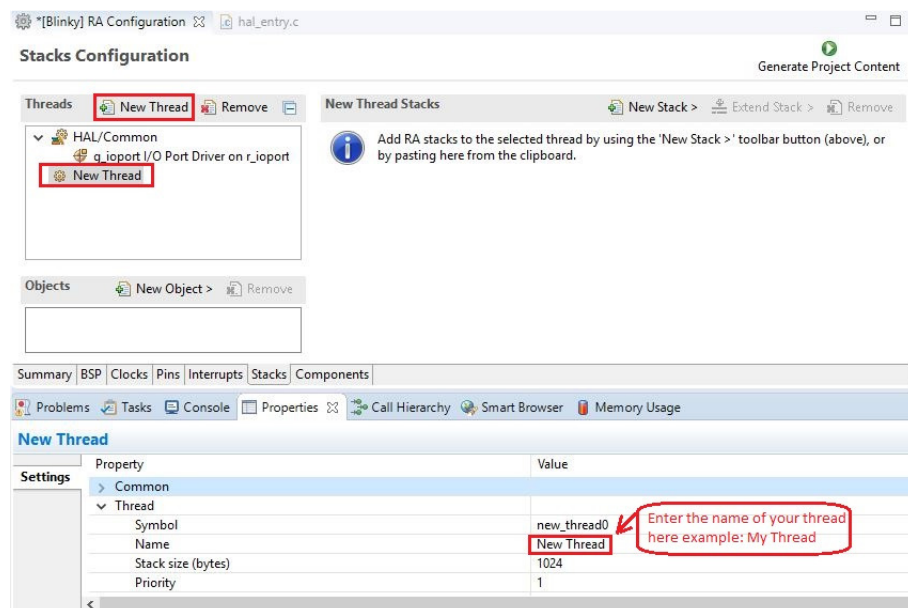


Figure 28: Adding a new RTOS Thread on the Stacks tab

2. In the **Properties** view, click on the **Name** and **Symbol** entries and enter a distinctive name and symbol for the new thread.

Note

*The ISDE updates the name of the thread stacks pane to **My Thread Stacks**.*

3. In the **My Thread Stacks** pane, click on **New Stack** to see a list of modules and drivers. HAL-level drivers can be added here.

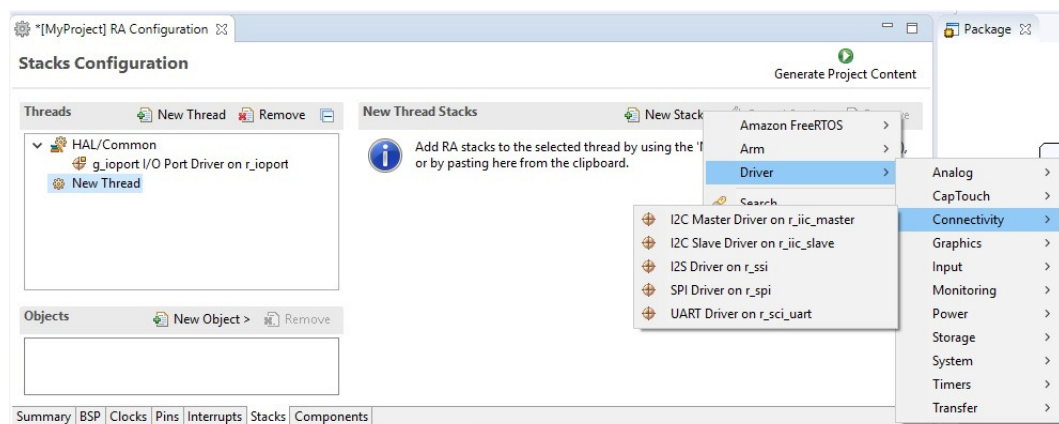


Figure 29: Adding Modules and Drivers to a thread

4. Select a module or driver from the list.
5. Click on the added driver and configure the driver as required by the application by updating the configuration parameters in the **Properties** view. To see the selected module or driver and be able to edit its properties, make sure the Thread containing the driver is

highlighted in the **Threads** pane.

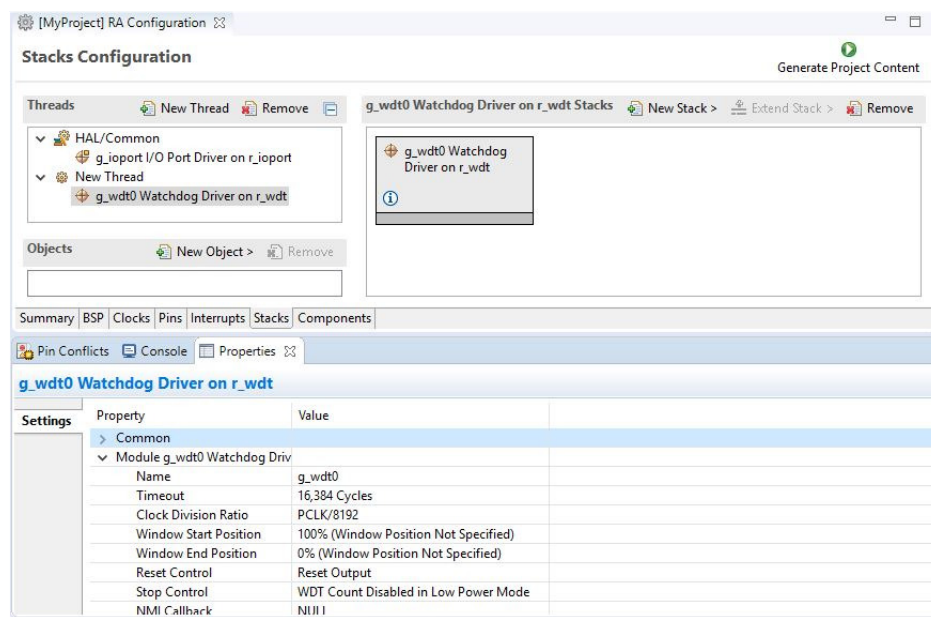


Figure 30: Configuring Module or Driver properties

6. If needed, add another thread by clicking **New Thread** in the **Threads** pane.

When you press the **Generate Project Content** button for the example above, the ISDE creates the files as shown in the following table:

File	Contents	Overwritten by Generate Project Content?
ra_gen/main.c	Contains main() calling generated and user code. When called the BSP will have initialized the MCU.	Yes
ra_gen/my_thread.c	Generated thread "my_thread" and configuration structures for modules added to this thread.	Yes
ra_gen/my_thread.h	Header file for thread "my_thread"	Yes
ra_gen/hal_data.c	Configuration structures for HAL Driver only modules.	Yes
ra_gen/hal_data.h	Header file for HAL Driver only modules.	Yes
src/hal_entry.c	User entry point for HAL Driver only code. Add your code here.	No
src/my_thread_entry.c	User entry point for thread "my_thread". Add your code here.	No

The configuration header files for all included modules and drivers are created or overwritten in the following folders: ra_cfg/fsp_cfg/<header files>

2.2.6.3 Configuring Threads

If the application uses the FreeRTOS, the **Stacks** tab can be used to simplify the creation of FreeRTOS threads, semaphores, mutexes, and event flags.

The components of each thread can be configured from the **Properties** view as shown below.

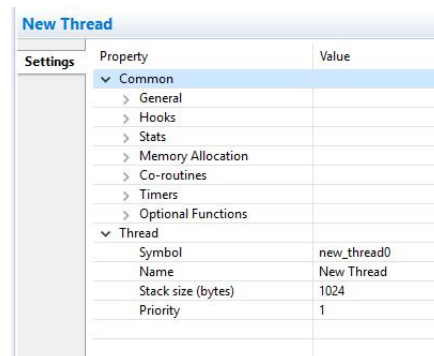


Figure 31: New Thread Properties

The **Properties** view contains settings common for all Threads (**Common**) and settings for this particular thread (**Thread**).

For this thread instance, the thread's name and properties (such as priority level or stack size) can be easily configured. The ISDE checks that the entries in the property field are valid. For example, the ISDE ensures that the field **Priority**, which requires an integer value, only contains numeric values between 0 and 9.

To add FreeRTOS resources to a Thread, select a thread and click on **New Object** in the Thread Objects pane. The pane takes on the name of the selected thread, in this case **My Thread Objects**.

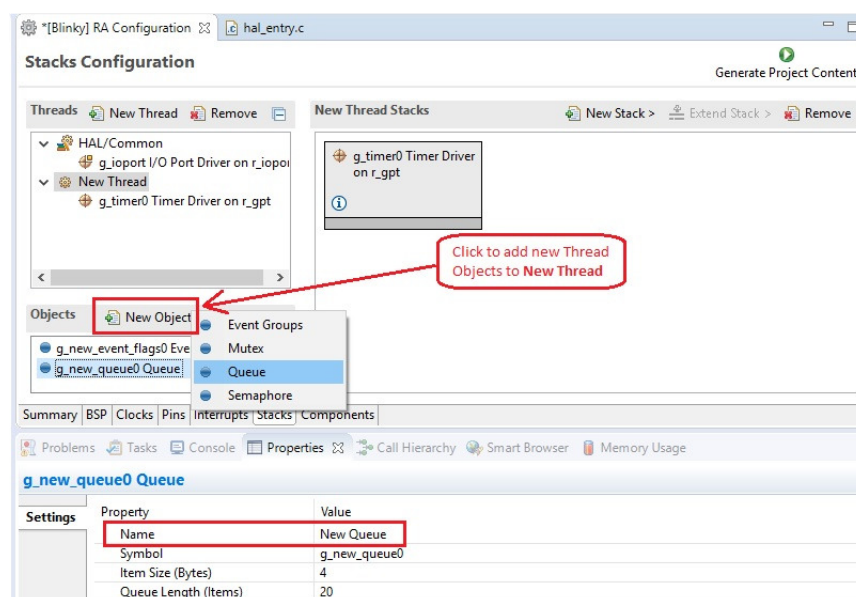


Figure 32: Configuring Thread Object Properties

Make sure to give each thread object a unique name and symbol by updating the **Name** and **Symbol** entries in the **Properties** view.

2.2.7 Reviewing and Adding Components

The **Components** tab enables the individual modules required by the application to be included or excluded. Modules common to all RA MCU projects are preselected (for example: **BSP > BSP > Board-specific BSP** and **HAL Drivers > all > r_cgc**). All modules that are necessary for the modules selected in the **Stacks** tab are included automatically. You can include or exclude additional modules by ticking the box next to the required component.

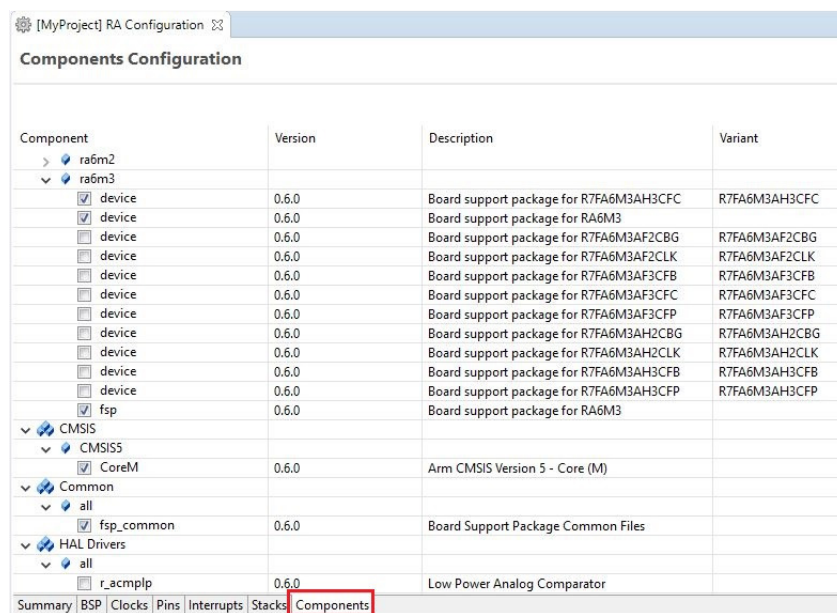


Figure 33: Components Tab

While the components tab selects modules for a project, you must configure the modules themselves in the other tabs. clicking the **Generate Project Content** button copies the .c and .h files for each component for a Pack file into the following folders:

- ra/fsp/inc/api
- ra/fsp/inc/instances
- ra/fsp/src/bsp
- ra/fsp/src/<Driver_Name>

The ISDE also creates configuration files in the ra_cfg/fsp_cfg folder with configuration options included from the remaining **Stacks** tabs.

2.2.8 Writing the Application

Once you have added Modules and drivers and set their configuration parameters in the **Stacks** tab, you can add the application code that calls the Modules and drivers.

Note

To check your configuration, build the project once without errors before adding any of your own application code.

2.2.8.1 Coding Features

The ISDE provides several efficiency improving features that help write code. Review these features prior to digging into the code development step-by-step sections that follow.

Edit Hover

e² studio supports hovers in the textual editor. This function can be enabled or disabled via **Window > Preferences > C/C++ > Editor > Hovers**.

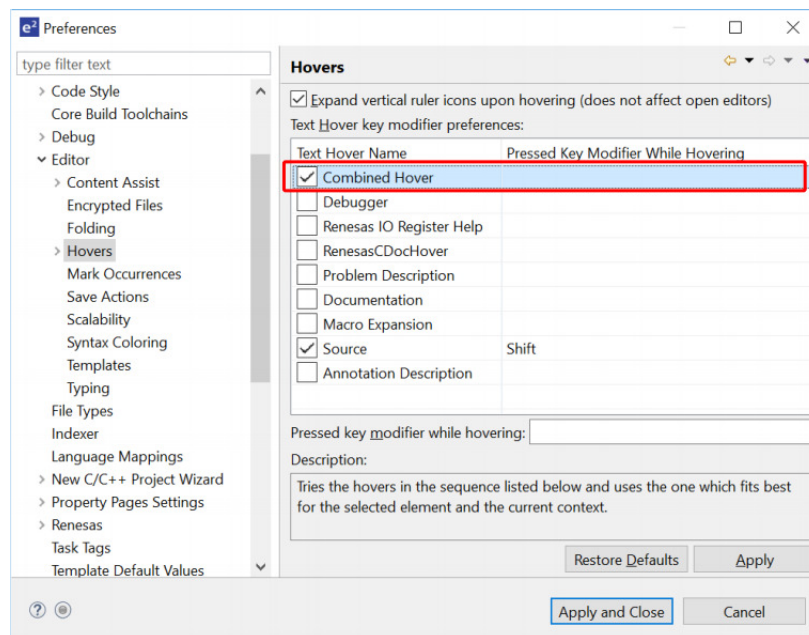


Figure 34: Hover preference

To enable hover, check **Combined Hover** box. To disable it, uncheck this box. By default, it is enabled. The Hover function allows a user to view detailed information about any identifiers in the source code by hovering the mouse over an identifier and checking the pop-up.

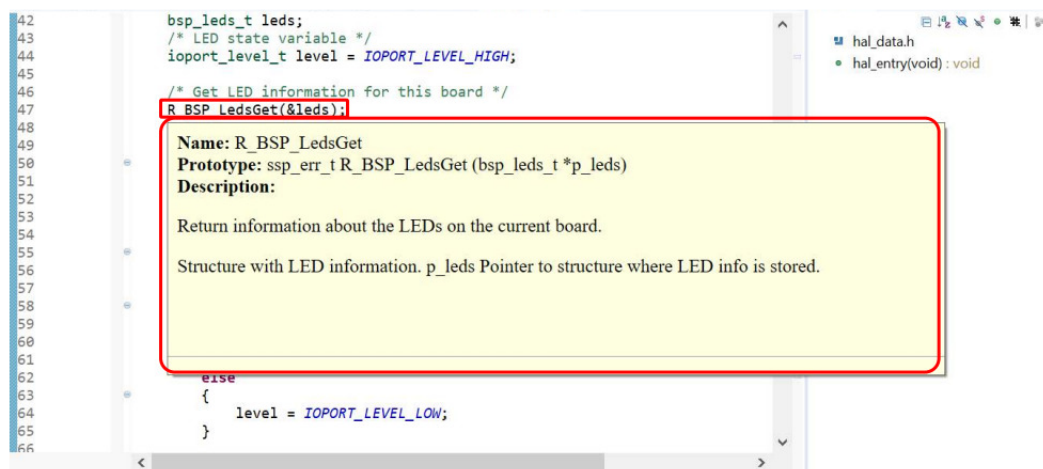


Figure 35: Hover Example

Welcome Window

The e² studio Welcome window displays useful information and common links to assist in development. Check out these resources to see what is available. They are updated with each release, so check back to see what has been added after a new release.

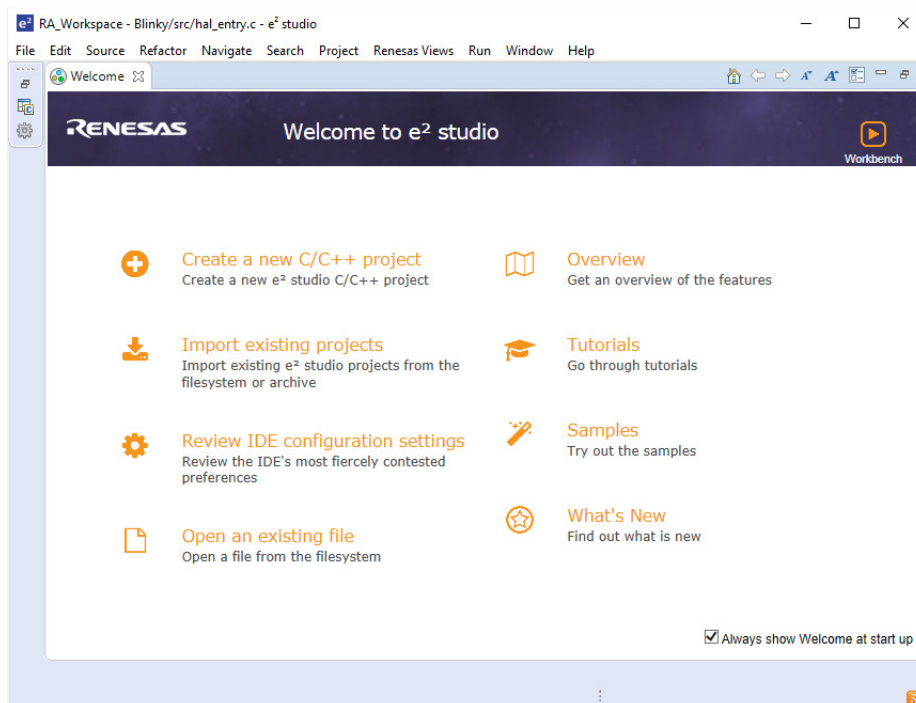


Figure 36: Welcome window

Cheat Sheets

Cheat sheets are macro driven illustrations of some common tasks. They show, step-by-step, what commands and menus are used. These will be populated with more examples on each release. Cheat Sheets are available from the **Help** menu.

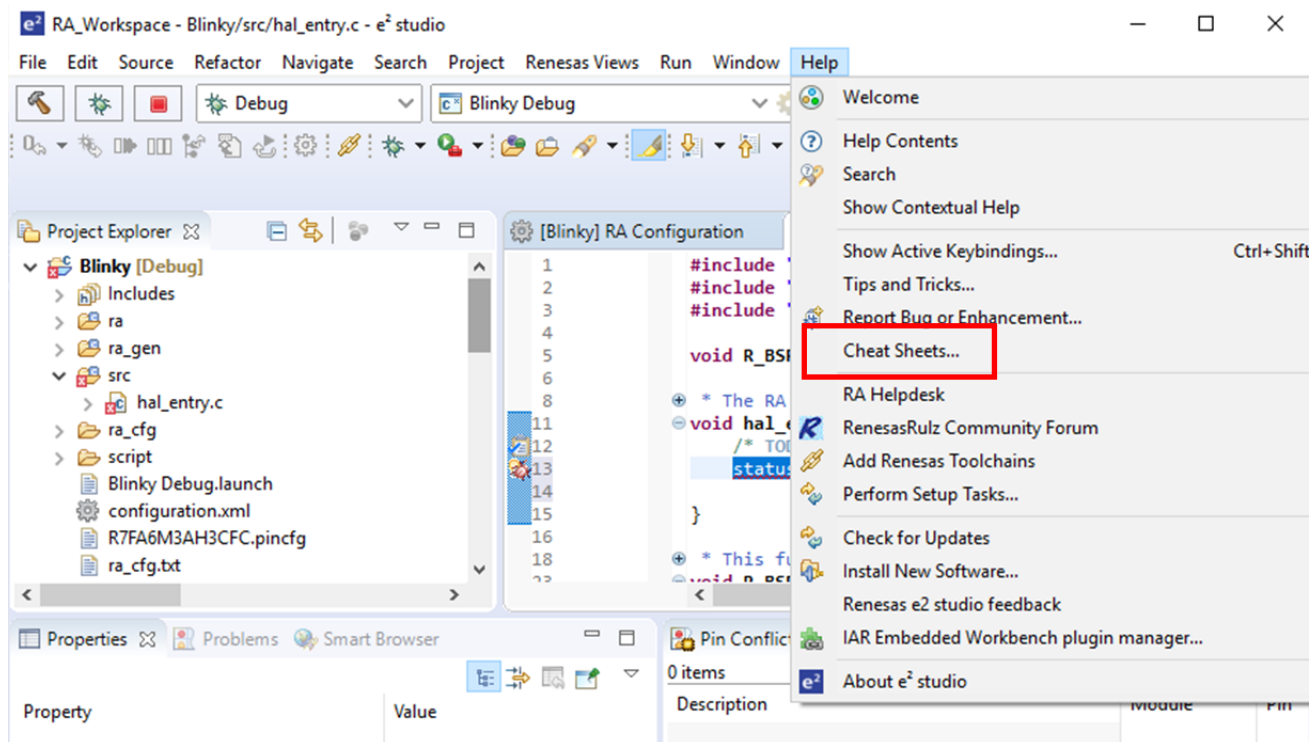


Figure 37: Cheat Sheets

Developer Assistance

FSP Developer Assistance provides developers with module and Application Programming Interface (API) reference documentation in e² studio. After configuring the threads and software stacks for an FSP project with the Configuration Editor, Developer Assistance quickly helps you get started writing C/C++ application code for the project using the configured stack modules.

1. Expand the project explorer to view Developer Assistance

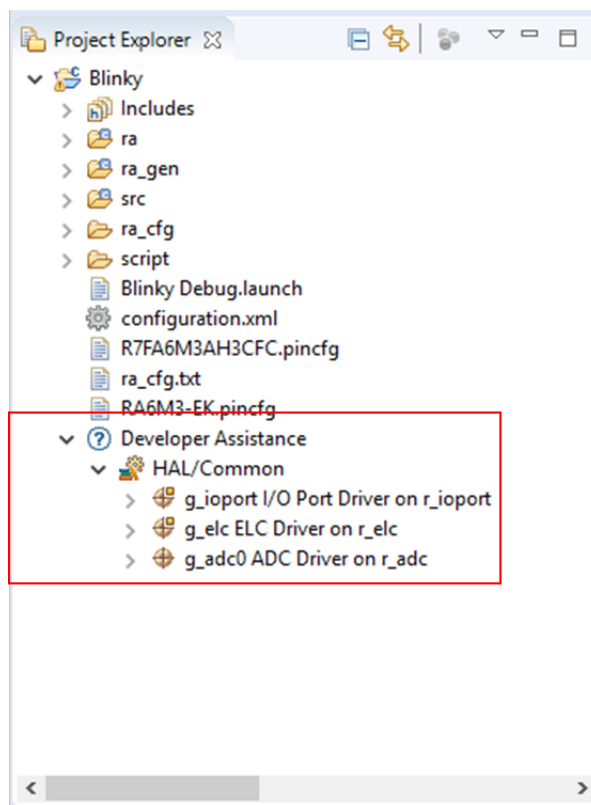


Figure 38: Developer Assistance

2. Expand a stack module to show its APIs

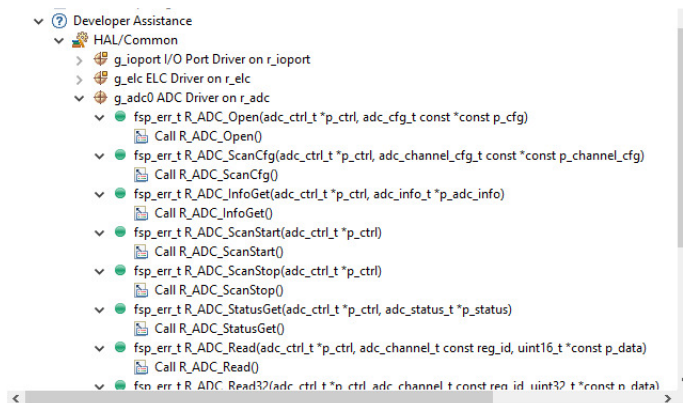


Figure 39: Developer Assistance APIs

3. Dragging and dropping an API from Develop Assistance to a source file helps to write source code quickly.

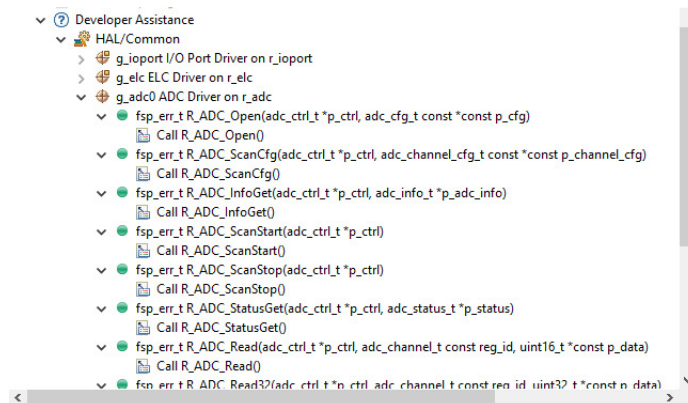


Figure 40: Dragging and Dropping an API in Developer Assistance

Information Icon

Information icons are available on each module in the thread stack. Clicking on these icons opens a module folder on GitHub that contains additional information on the module. An example information icon is shown below:

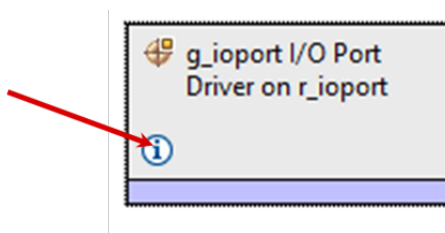


Figure 41: Information icon

Smart Manual

Smart Manual is the view that displays information (register information/search results by keyword) extracted from the hardware user's manual. Smart Manual provides search capability of hardware manual information (register information search and keyword search result) and provides a view displaying result.

You can open Smart Manual view by selecting the menu: **Renesas Views > Solution Toolkit > Smart Manual**. Register search and Keyword search are both available by selecting the appropriate tab.

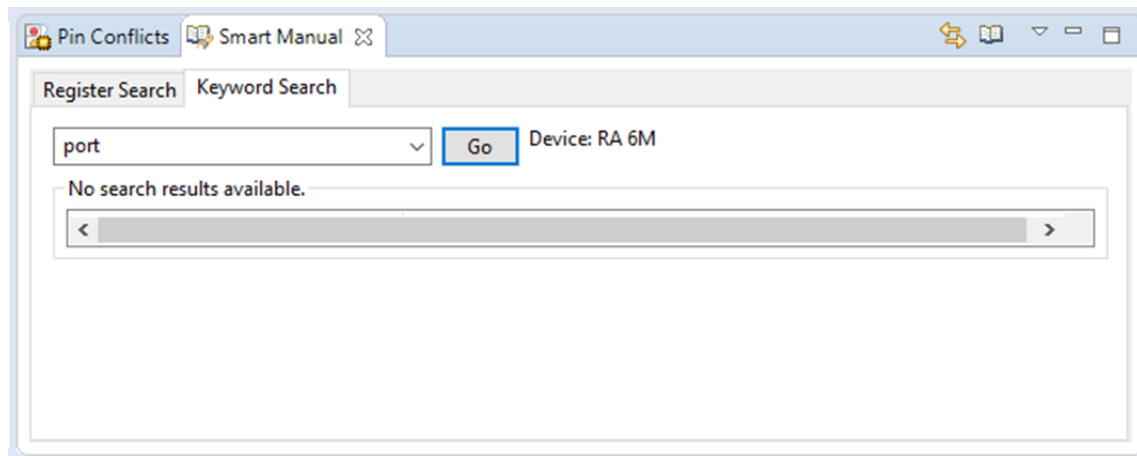
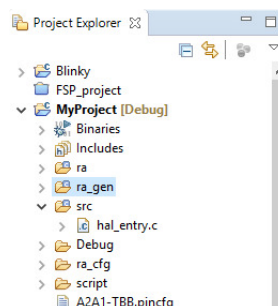


Figure 42: Smart Manual

2.2.8.2 RTOS-independent Applications

To write application code:

1. Add all drivers and modules in the **Stacks** tab and resolve all dependencies flagged by the ISDE such as missing interrupts or drivers.
2. Configure the drivers in the **Properties** view.
3. In the Project Configuration view, click the **Generate Project Content** button.
4. In the **Project Explorer** view, double-click on the src/hal_entry.c file to edit the source file.



Note

All configuration structures necessary for the driver to be called in the application are initialized in `ra_gen/hal_data.c`.

Warning

Do not modify the files in the directory `ra_gen`. These files are overwritten every time you push the **Generate Project Content** button.

5. Add your application code here:

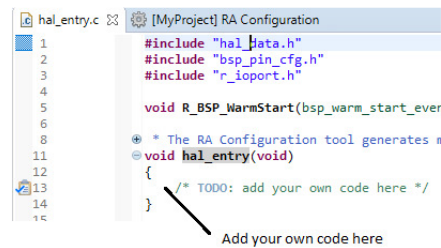


Figure 43: Adding user code to hal_entry.c

6. Build the project without errors by clicking on **Project > Build Project**.

The following tutorial shows how execute the steps above and add application code: [Tutorial: Using HAL Drivers - Programming the WDT](#).

The WDT example is a HAL level application which does not use an RTOS. The user guides for each module also include basic application code that you can add to hal_entry.c.

2.2.8.3 RTOS Applications

To write RTOS-aware application code using FreeRTOS, follow these steps:

1. Add a thread using the **Stacks** tab.
2. Provide a unique name for the thread in the **Properties** view for this thread.
3. Configure all drivers and resources for this thread and resolve all dependencies flagged by the ISDE such as missing interrupts or drivers.
4. Configure the thread objects.
5. Provide unique names for each thread object in the **Properties** view for each object.
6. Add more threads if needed and repeat steps 1 to 5.
7. In the **RA Project Editor**, click the **Generate Project Content** button.
8. In the **Project Explorer** view, double-click on the src/my_thread_1_entry.c file to edit the source file.

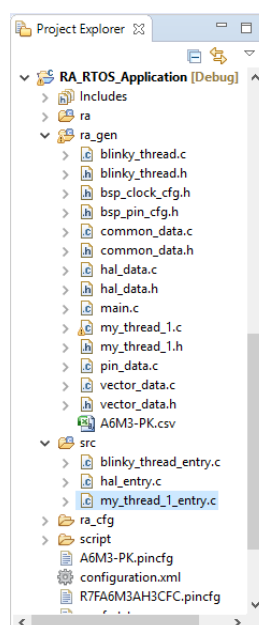


Figure 44: ISDE generated files for an RTOS application

Note

All configuration structures necessary for the driver to be called in the application are initialized in `ra_gen/my_thread_1.c` and `my_thread_2.c`

Warning

Do not modify the files in the directory `ra_gen`. These files are overwritten every time you push the **Generate Project Content** button.

9. Add your application code here:

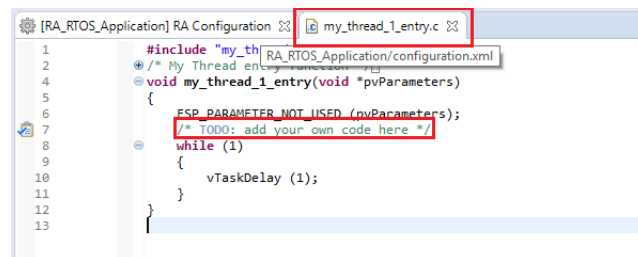


Figure 45: Adding user code to `my_thread_1.entry`

10. Repeat steps 1 to 9 for the next thread.

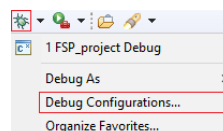
11. Build your project without errors by clicking on **Project > Build Project**.

2.2.9 Debugging the Project

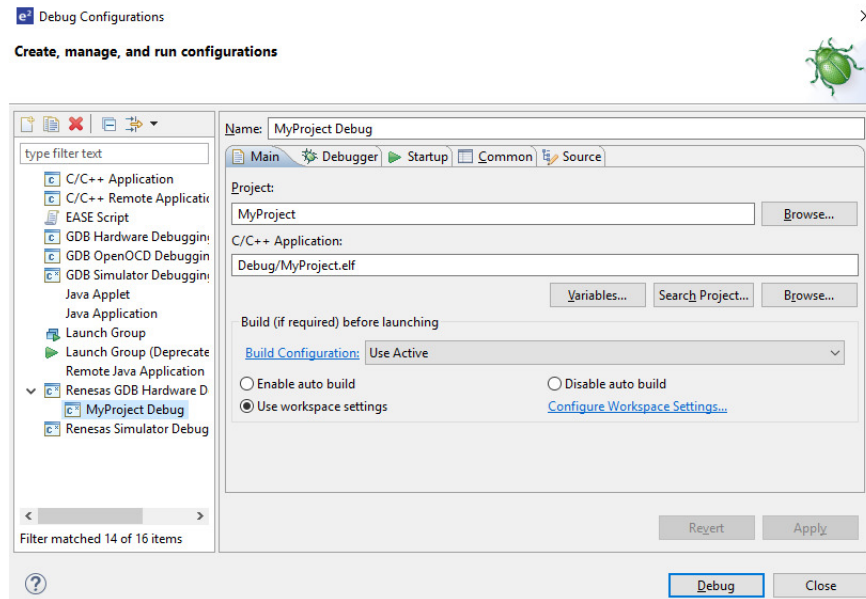
Once your project builds without errors, you can use the Debugger to download your application to the board and execute it.

To debug an application follow these steps:

1. On the drop-down list next to the debug icon, select **Debug Configurations**.



2. In the **Debug Configurations** view, click on your project listed as **MyProject Debug**.



3. Connect the board to your PC via either a standalone Segger J-Link debugger or a Segger J-Link On-Board (included on all RA EKs) and click **Debug**.

Note

For details on using J-Link and connecting the board to the PC, see the Quick Start Guide included in the RA MCU Kit.

2.2.10 Modifying Toolchain Settings

There are instances where it may be necessary to make changes to the toolchain being used (for example, to change optimization level of the compiler or add a library to the linker). Such modifications can be made from within the ISDE through the menu **Project > Properties > Settings** when the project is selected. The following screenshot shows the settings dialog for the GNU ARM toolchain. This dialog will look slightly different depending upon the toolchain being used.

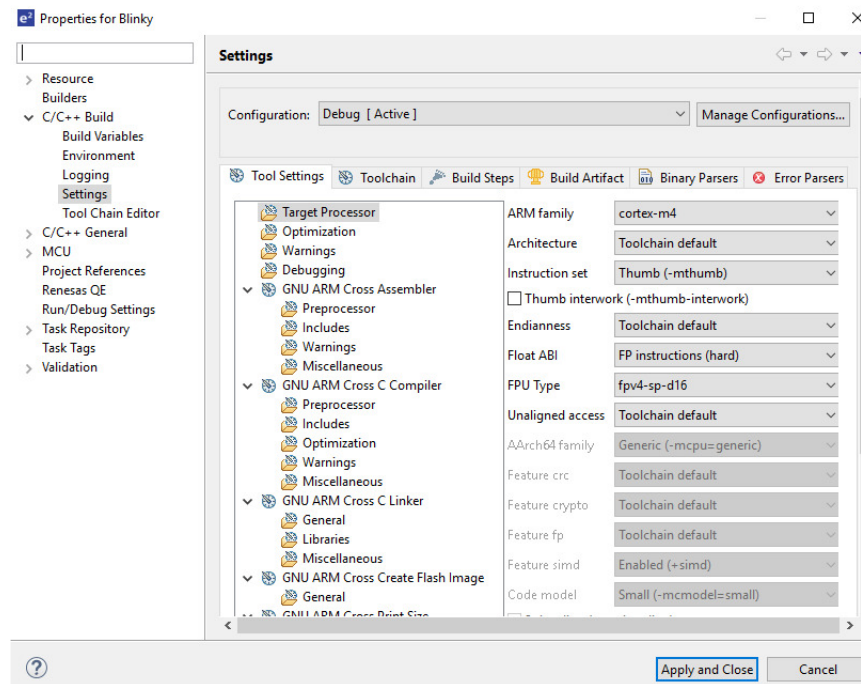


Figure 46: ISDE Project toolchain settings

The scope for the settings is project scope which means that the settings are valid only for the project being modified.

The settings for the linker which control the location of the various memory sections are contained in a script file specific for the device being used. This script file is included in the project when it is created and is found in the script folder (for example, /script/a6m3.ld).

2.2.11 Importing an Existing Project into e2 studio ISDE

1. Start by opening e² studio.
2. Open an existing Workspace to import the project and skip to step d. If the workspace doesn't exist, proceed with the following steps:
 - a. At the end of e² studio startup, you will see the Workspace Launcher Dialog box as shown in the following figure.

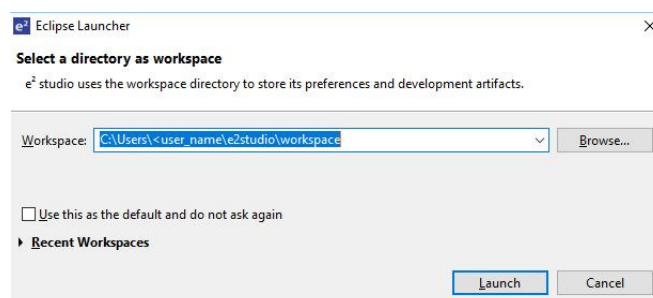


Figure 47: Workspace Launcher dialog

- b. Enter a new workspace name in the Workspace Launcher Dialog as shown in the following figure. e² studio creates a new workspace with this name.

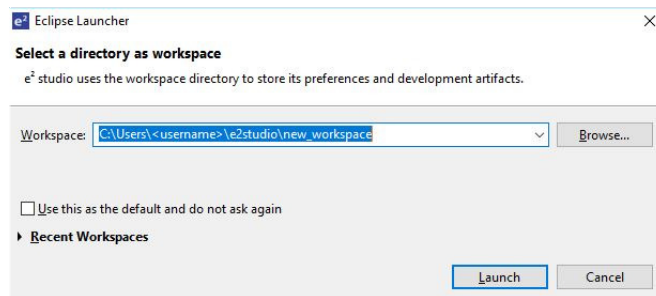


Figure 48: Workspace Launcher dialog - Select Workspace

- c. Click **Launch**.
- d. When the workspace is opened, you may see the Welcome Window. Click on the **Workbench** arrow button to proceed past the Welcome Screen as seen in the following figure.

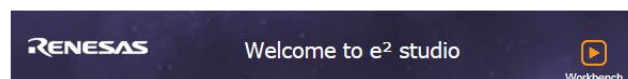


Figure 49: Workbench arrow button

3. You are now in the workspace that you want to import the project into. Click the **File** menu in the menu bar, as shown in the following figure.



Figure 50: Menu and tool bar

4. Click **Import** on the **File** menu or in the menu bar, as shown in the following figure.

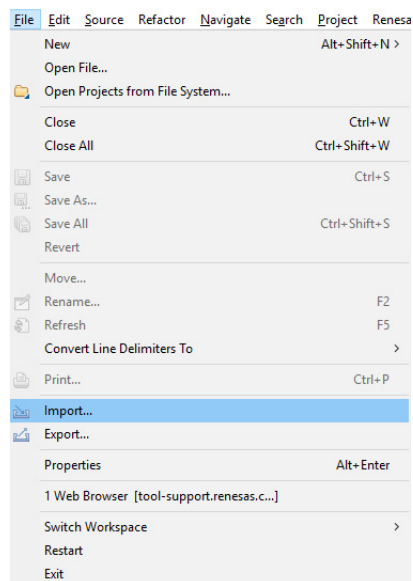


Figure 51: File drop-down menu

5. In the **Import** dialog box, as shown in the following figure, choose the **General** option, then **Existing Projects into Workspace**, to import the project into the current workspace.

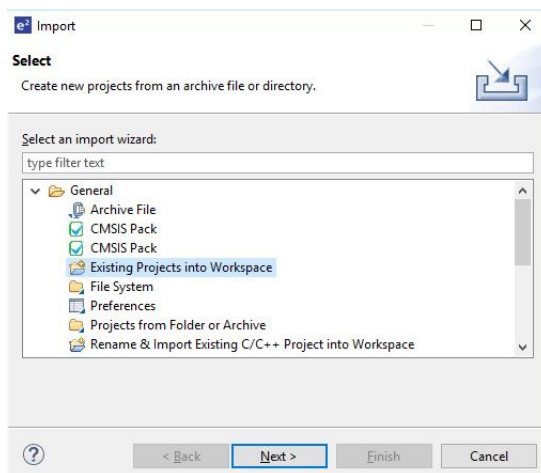


Figure 52: Project Import dialog with

Existing Projects into Workspace" option selected"

6. Click **Next**.
7. To import the project, use either **Select archive file** or **Select root directory**.
- a. Click **Select archive file** as shown in the following figure.

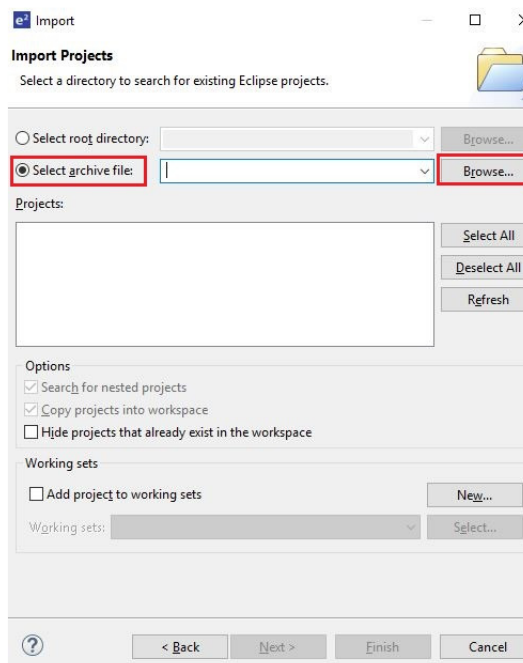


Figure 53: Import Existing Project dialog 1 - Select archive file

b. Click **Select root directory** as shown in the following figure.

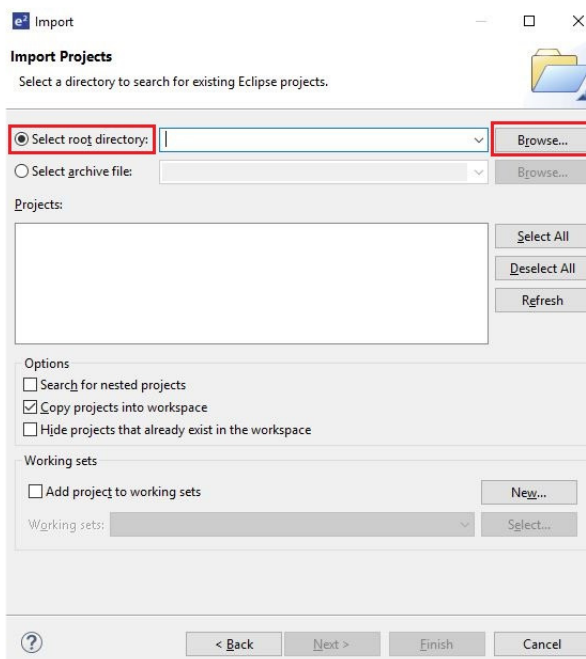


Figure 54: Import Existing Project dialog 1 - Select root directory

8. Click **Browse**.
9. For **Select archive file**, browse to the folder where the zip file for the project you want to import is located. For **Select root directory**, browse to the project folder that you want to import.
10. Select the file for import. In our example, it is CAN_HAL_MG_AP.zip or CAN_HAL_MG_AP.

11. Click **Open**.

12. Select the project to import from the list of **Projects**, as shown in the following figure.

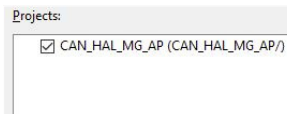


Figure 55: Import Existing Project dialog 2

13. Click **Finish** to import the project.

2.3 Tutorial: Your First RA MCU Project - Blinky

2.3.1 Tutorial Blinky

The goal of this tutorial is to quickly get acquainted with the Flexible Platform by moving through the steps of creating a simple application using e² studio and running that application on an RA MCU board.

2.3.2 What Does Blinky Do?

The application used in this tutorial is Blinky, traditionally the first program run in a new embedded development environment.

Blinky is the "Hello World" of microcontrollers. If the LED blinks you know that:

- The toolchain is setup correctly and builds a working executable image for your chip.
- The debugger has installed with working drivers and is properly connected to the board.
- The board is powered up and its jumper and switch settings are probably correct.
- The microcontroller is alive, the clocks are running, and the memory is initialized.

The Blinky example application used in this tutorial is designed to run the same way on all boards offered by Renesas that hold the RA microcontroller. The code in Blinky is completely board independent. It does the work by calling into the BSP (board support package) for the particular board it is running on. This works because:

- Every board has at least one LED connected to a GPIO pin.
- That one LED is always labeled LED1 on the silk screen.
- Every BSP supports an API that returns a list of LEDs on a board, and their port and pin assignments.

2.3.3 Prerequisites

To follow this tutorial, you need:

- Windows based PC
- e² studio
- Flexible Software Package
- An RA MCU board kit

2.3.4 Create a New Project for Blinky

The creation and configuration of an RA MCU project is the first step in the creation of an application. The base RA MCU pack includes a pre-written Blinky example application that is simple and works on all Renesas RA MCU boards.

Follow these steps to create an RA MCU project:

1. In e² studio ISDE, click **File > New > RA Project** and select **Renesas RA C Executable Project**.
2. Assign a name to this new project. Blinky is a good name to use for this tutorial.
3. Click **Next**. The **Project Configuration** window shows your selection.

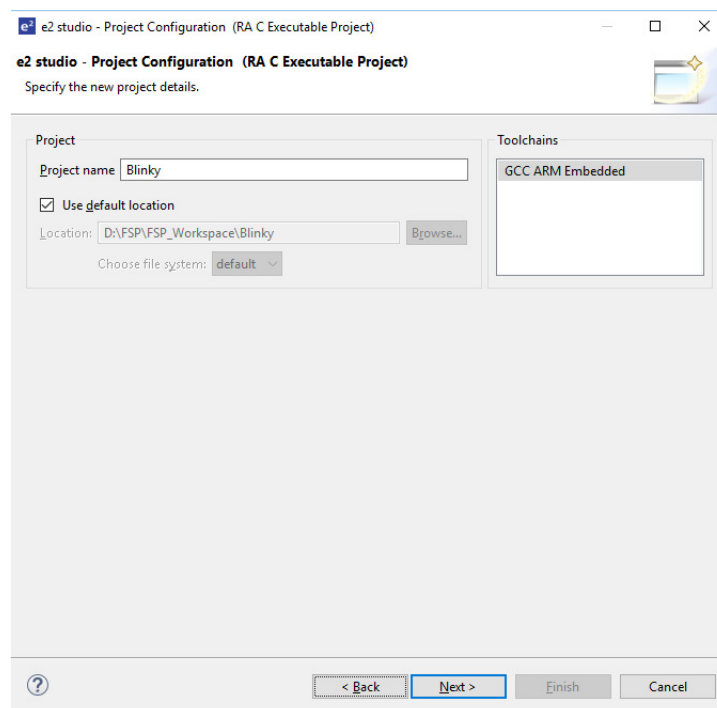


Figure 56: e2 studio ISDE Project Configuration window (part 1)

4. Select the board support package by selecting the name of your board from the **Device Selection** drop-down list and click **Next**.

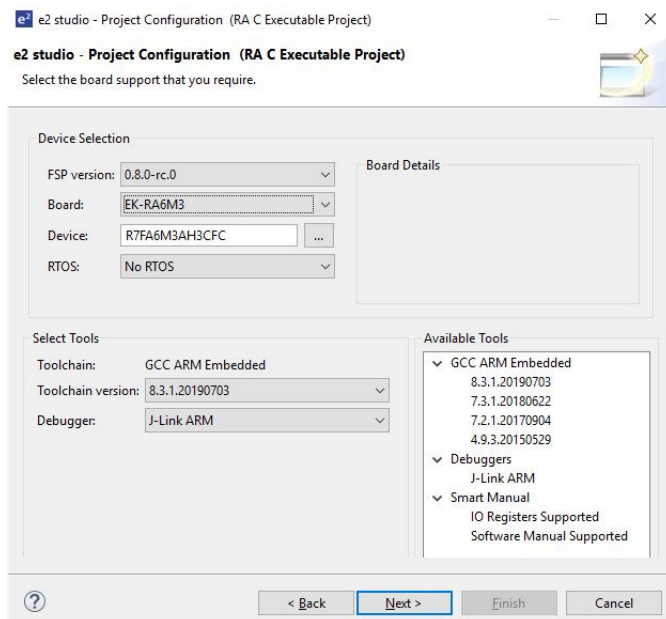


Figure 57: e2 studio ISDE Project Configuration window (part 2)

5. Select the Blinky template for your board and click **Finish**.

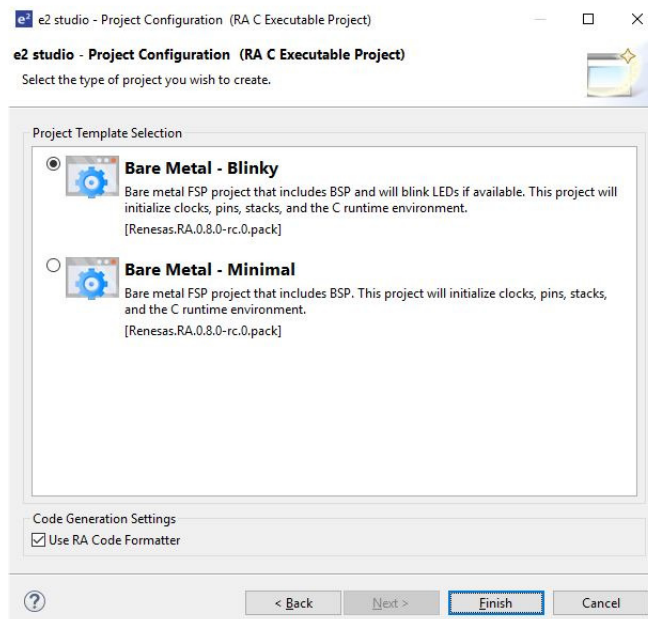


Figure 58: e2 studio ISDE Project Configuration window (part 3)

Once the project has been created, the name of the project will show up in the **Project Explorer** window of the ISDE. Now click the **Generate Project Content** button in the top right corner of the **Project Configuration** window to generate your board specific files.

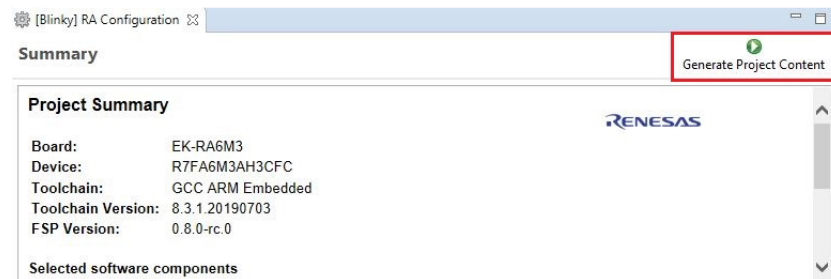


Figure 59: e2 studio ISDE Project Configuration tab

Your new project is now created, configured, and ready to build.

2.3.4.1 Details about the Blinky Configuration

The **Generate Project Content** button creates configuration header files, copies source files from templates, and generally configures the project based on the state of the **Project Configuration** screen.

For example, if you check a box next to a module in the **Components** tab and click the **Generate Project Content** button, all the files necessary for the inclusion of that module into the project will be copied or created. If that same check box is then unchecked those files will be deleted.

2.3.4.2 Configuring the Blinky Clocks

By selecting the Blinky template, the clocks are configured by the ISDE for the Blinky application. The ISDE clock configuration tab (see [Configuring Clocks](#)) shows the Blinky clock configuration. The Blinky clock configuration is stored in the BSP clock configuration file (see [BSP Clock Configuration](#)).

2.3.4.3 Configuring the Blinky Pins

By selecting the Blinky template, the GPIO pins used to toggle the LED1 are configured by the ISDE for the Blinky application. The ISDE pin configuration tab shows the pin configuration for the Blinky application (see [Configuring Pins](#)). The Blinky pin configuration is stored in the BSP configuration file (see [BSP Pin Configuration](#)).

2.3.4.4 Configuring the Parameters for Blinky Components

The Blinky project automatically selects the following HAL components in the ISDE Component:

- r_ioport

To see the configuration parameters for any of the components, check the **Properties** tab in the HAL window for the respective driver (see [Adding and Configuring HAL Drivers](#)).

2.3.4.5 Where is main()?

The main function is located in < project >/ra_gen/main.c. It is one of the files that are generated during the project creation stage and only contains a call to hal_entry(). For more information on generated files, see [Adding and Configuring HAL Drivers](#).

2.3.4.6 Blinky Example Code

The blinky application is stored in the `hal_entry.c` file. This file is generated by the ISDE when you select the Blinky Project template and is located in the project's `src/` folder.

The application performs the following steps:

1. Get the LED information for the selected board by `bsp_leds_t` structure.
2. Define the output level HIGH for the GPIO pins controlling the LEDs for the selected board.
3. Get the selected system clock speed and scale down the clock, so the LED toggling can be observed.
4. Toggle the LED by writing to the GPIO pin with `R_BSP_PinWrite((bsp_io_port_pin_t) pin, pin_level);`

2.3.5 Build the Blinky Project

Highlight the new project in the **Project Explorer** window by clicking on it and build it.

There are three ways to build a project:

- a. Click on **Project** in the menu bar and select **Build Project**.
- b. Click on the hammer icon.
- c. Right-click on the project and select **Build Project**.

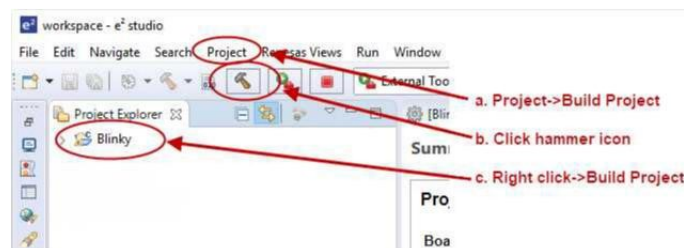


Figure 60: e2 studio ISDE Project Explorer window

Once the build is complete a message is displayed in the build **Console** window that displays the final image file name and section sizes in that image.

```

Pin Conflicts  MCU Package  Console
CDT Build Console [Blinky]
'Finished building: ../ra/board/ra6m3_ek/board_leds.c'
'Finished building: ../ra/board/ra6m3_ek/board_init.c'
'
'
'Finished building: ../ra/board/ra6m3_ek/board_qspi.c'
'
'Building target: Blinky.elf'
'Invoking: GNU ARM Cross C Linker'
arm-none-eabi-gcc @"Blinky.elf.in"
'Finished building target: Blinky.elf'
'
'Invoking: GNU ARM Cross Create Flash Image'
arm-none-eabi-objcopy -O srec "Blinky.elf" "Blinky.srec"
'Invoking: GNU ARM Cross Print Size'
arm-none-eabi-size --format=berkeley "Blinky.elf"
  text  data  bss   dec   hex filename
 4240    8   1152  5400  1518 Blinky.elf
'Finished building: Blinky.srec'
'Finished building: Blinky.siz'
'
'
11:50:45 Build Finished. 0 errors, 0 warnings. (took 19s.208ms)

```

Figure 61: e2 studio ISDE Project Build console

2.3.6 Debug the Blinky Project

2.3.6.1 Debug prerequisites

To debug the project on a board, you need

- The board to be connected to the ISDE
- The debugger to be configured to talk to the board
- The application to be programmed to the microcontroller

Applications run from the internal flash of your microcontroller. To run or debug the application, the application must first be programmed to the microcontroller's flash. There are two ways to do this:

- JTAG debugger
- Built-in boot-loader via UART or USB

Some boards have an on-board JTAG debugger and others require an external JTAG debugger connected to a header on the board.

Refer to your board's user manual to learn how to connect the JTAG debugger to your ISDE.

2.3.6.2 Debug steps

To debug the Blinky application, follow these steps:

1. Configure the debugger for your project by clicking **Run > Debugger Configurations ...**

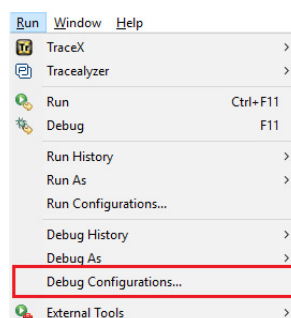


Figure 62: e2 studio ISDE Debug icon

or by selecting the drop-down menu next to the bug icon and selecting **Debugger Configurations ...**

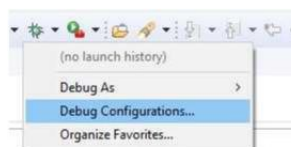


Figure 63: e2 studio ISDE Debugger Configurations selection option

2. Select your debugger configuration in the window. If it is not visible then it must be created

by clicking the **New** icon in the top left corner of the window. Once selected, the **Debug Configuration** window displays the Debug configuration for your Blinky project.

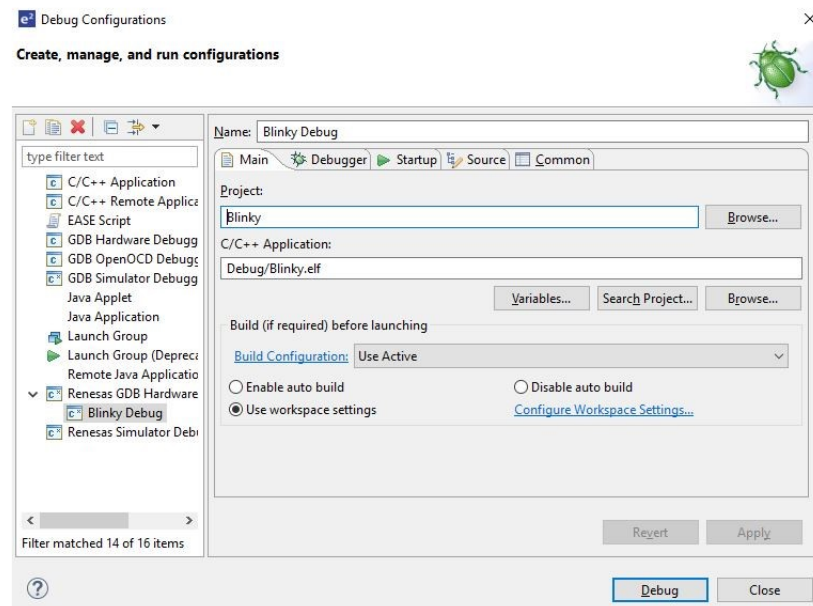
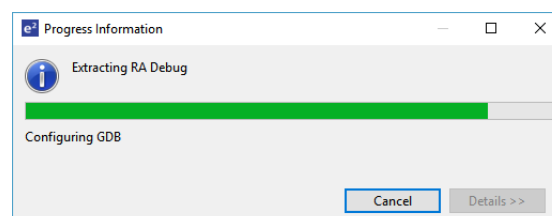


Figure 64: e2 studio ISDE Debugger Configurations window with Blinky project

3. Click **Debug** to begin debugging the application.

4. Extracting RA Debug.



2.3.6.3 Details about the Debug Process

In debug mode, the ISDE executes the following tasks:

1. Downloading the application image to the microcontroller and programming the image to the internal flash memory.
2. Setting a breakpoint at main().
3. Setting the stack pointer register to the stack.
4. Loading the program counter register with the address of the reset vector.
5. Displaying the startup code where the program counter points to.

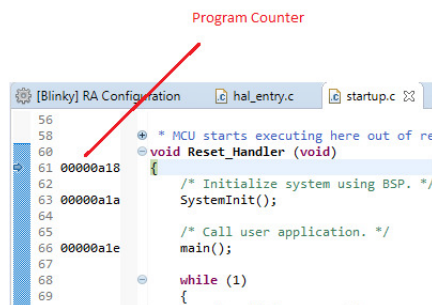


Figure 65: e2 studio ISDE Debugger memory window

2.3.7 Run the Blinky Project

While in Debug mode, click **Run > Resume** or click on the **Play** icon twice.



Figure 66: e2 studio ISDE Debugger Play icon

The LEDs on the board marked LED1, LED2, and LED3 should now be blinking.

2.4 Tutorial: Using HAL Drivers - Programming the WDT

2.4.1 Application WDT

This application uses the WDT Interface implemented by the WDT HAL Driver WDT. This document describes how to use the ISDE and FSP to create an application for the RA MCU Watchdog Timer (WDT) peripheral. This application makes use of the following FSP modules:

- [MCU Board Support Package](#)
- [Watchdog Timer \(r_wdt\)](#)
- [I/O Ports \(r_ioport\)](#)

2.4.2 Creating a WDT Application Using the RA MCU FSP and ISDE

2.4.2.1 Using the FSP and the e2 studio ISDE

The Flexible Software Package (FSP) from Renesas provides a complete driver library for developing RA MCU applications. The FSP provides Hardware Abstraction Layer (HAL) drivers, Board Support Package (BSP) drivers for the developer to use to create applications. The FSP is integrated into the Renesas e² studio Integrated Solution Development Environment (ISDE) based on eclipse providing build (editor, compiler and linker) and debug phases with an extended GNU Debug (GDB) interface.

2.4.2.2 The WDT Application

The flowchart for the WDT application is shown below.

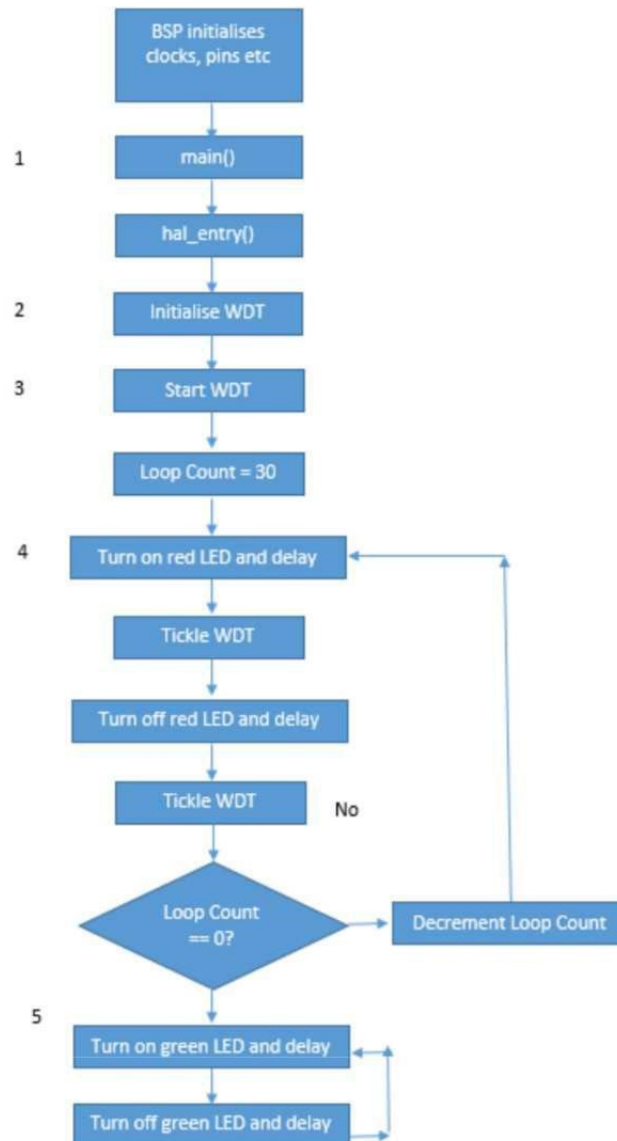


Figure 67: WDT Application flow diagram

2.4.2.3 WDT Application flow

These are the main parts of the WDT application:

1. `main()` calls `hal_entry()`. The function `hal_entry()` is created by the FSP with a placeholder for user code. The code for the WDT will be added to this function.
2. Initialize the WDT, but do not start it.
3. Start the WDT by refreshing it.
4. The red LED is flashed 30 times and refreshes the watchdog each time the LED state is changed.
5. Flash the green LED but DO NOT refresh the watchdog. After the timeout period of the watchdog the device will reset which can be observed by the flashing red LED again as the sequence repeats.

2.4.3 Creating the Project with the ISDE

Start the ISDE and choose a workspace folder in the Workspace Launcher. Configure a new RA MCU project as follows.

1. Select **File > New > RA C/C++ Project**. Then select the template for the project.

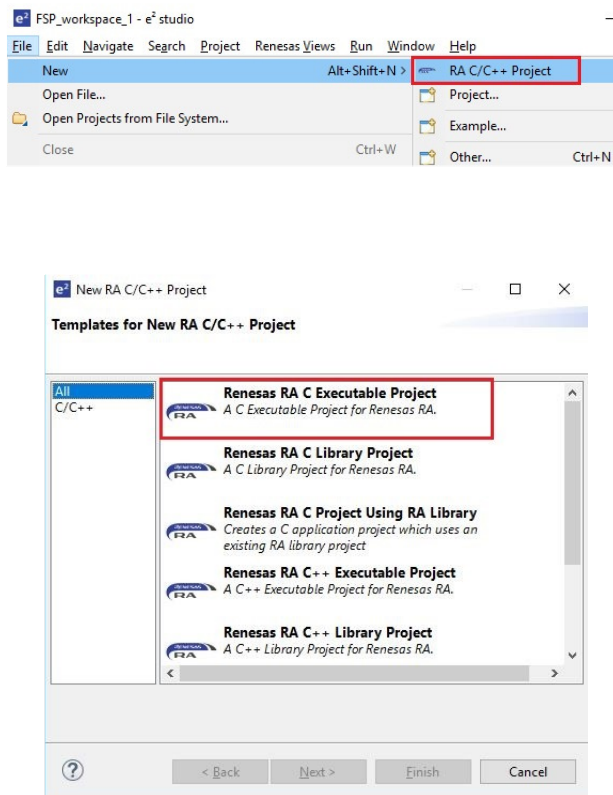


Figure 68: Creating a new project

2. In the ISDE Project **Configuration (RA Project)** window enter a project name, for example, WDT_Application. In addition select the toolchain. If you want to choose new locations for the project unselect **Use default location**. Click **Next**.

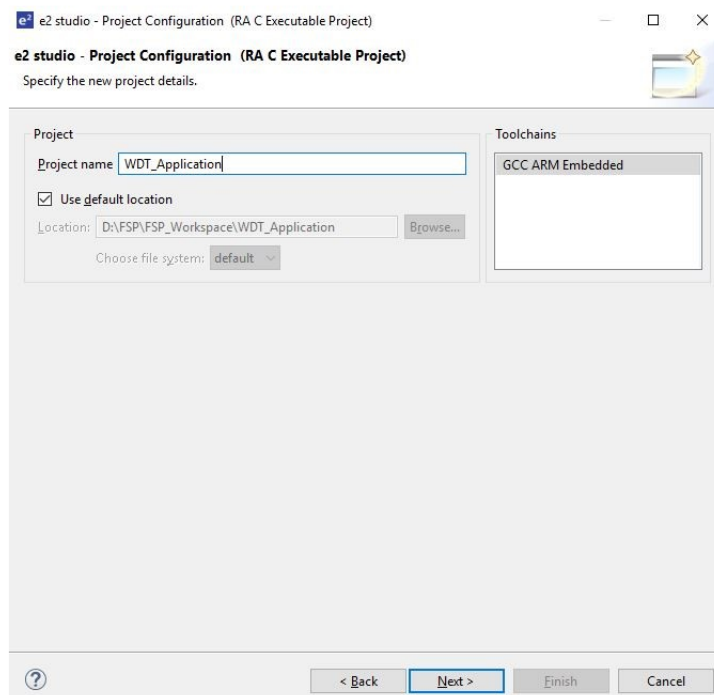


Figure 69: Project configuration (part 1)

3. This application runs on the RA6M3 board. So, for the **Board** select **EK-RA6M3**.

This will automatically populate the **Device** drop-down with the correct device used on this board. Select the **Toolchain** version. Select **J-Link ARM** as the **Debugger**. Click **Next** to configure the project.

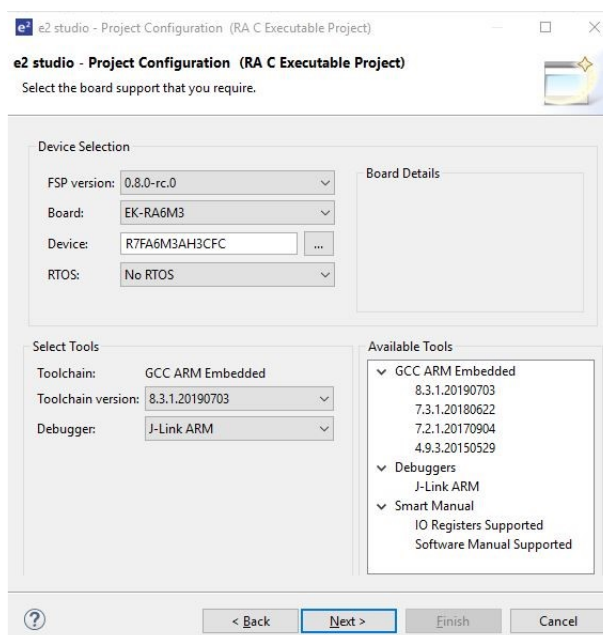


Figure 70: Project configuration (part 2)

The project template is now selected. As no RTOS is required select **Bare Metal - Blinky**.

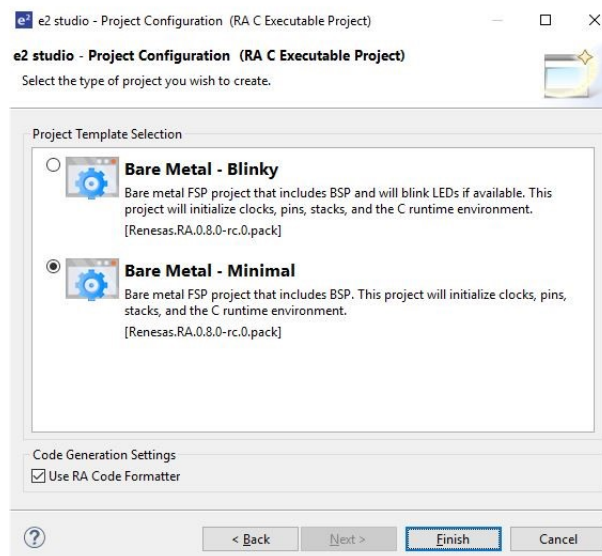


Figure 71: Project configuration (part 3)

4. Click **Finish**.

The ISDE creates the project and opens the **Project Explorer** and **Project Configuration Settings** views with the **Summary** page showing a summary of the project configuration.

2.4.4 Configuring the Project with the ISDE

The e² studio ISDE simplifies and accelerates the project configuration process by providing a GUI interface for selecting the options to configure the project.

The ISDE offers a selection of perspectives presenting different windows to the user depending on the operation in progress. The default perspectives are **C/C++**, **RA Configuration** and **Debug**. The perspective can be changed by selecting a new one from the buttons at the top right of the ISDE.

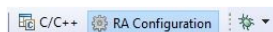


Figure 72: Selecting a perspective

The **C/C++** perspective provides a layout selected for code editing. The **RA Configuration** perspective provides elements for configuring a RA MCU project, and the **Debug** perspective provides a view suited for debugging.

1. In order to configure the project settings ensure the **RA Configuration** perspective is selected.
2. Ensure the **Project Configuration [WDT Application]** is open. It is already open if the Summary information is visible. To open the Project Configuration now or at any time make sure the **RA Configuration** perspective is selected and double-click on the configuration.xml file in the Project Explorer pane on the right side of the ISDE.

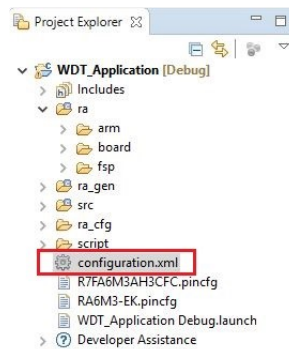


Figure 73: RA MCU Project Configuration Settings

At the base of the Project Configuration view there are several tabs for configuring the project. A project may require changes to some or all of these tabs. The tabs are shown below.

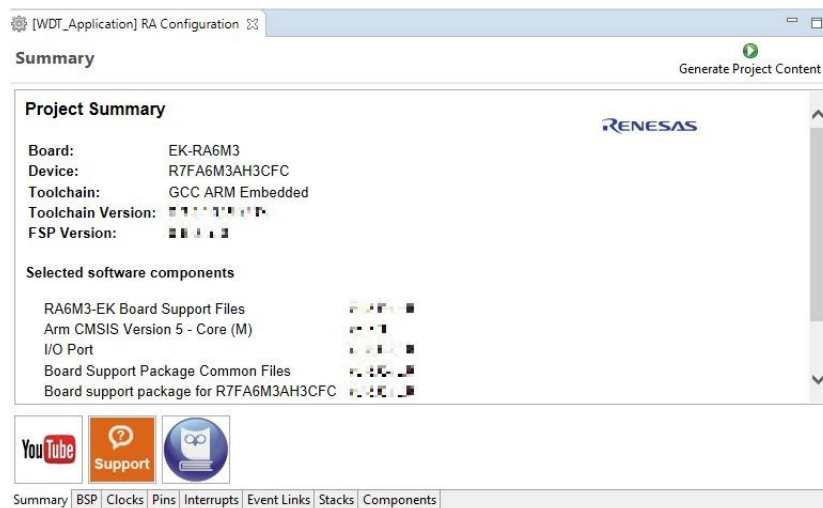


Figure 74: Project Configuration Tabs

2.4.4.1 BSP Tab

The **BSP** tab allows the Board Support Package (BSP) options to be modified from their defaults. For this particular WDT project no changes are required. However, if you want to use the WDT in auto-start mode, you can configure the settings of the OFS0 (Option Function Select Register 0) register in the **BSP** tab. See the RA Hardware User's Manual for details on the WDT autostart mode.

2.4.4.2 Clocks Tab

The **Clocks** tab presents a graphical view of the clock tree of the device. The drop-down boxes in the GUI enables configuration of the various clocks. The WDT uses PCLKB. The default output frequency for this clock is 60 MHz. Ensure this clock is outputting this value.

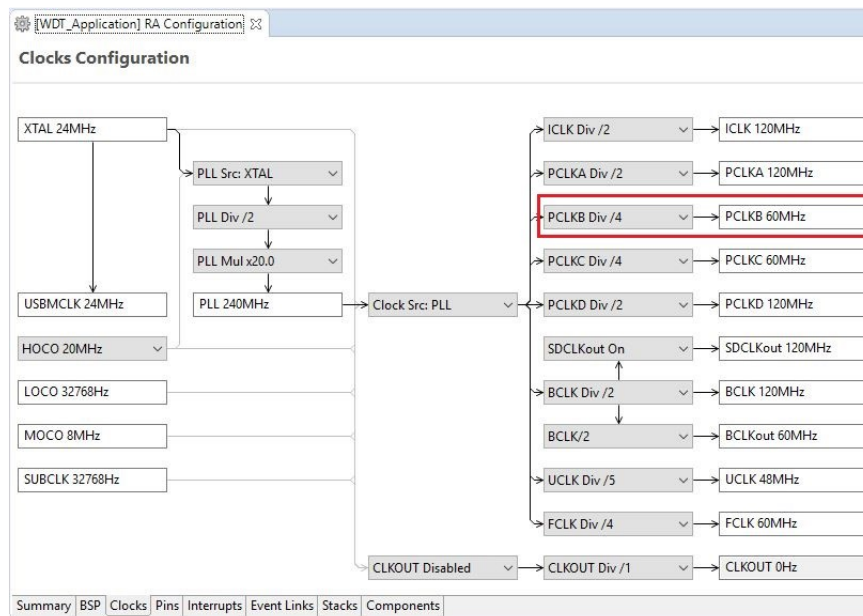


Figure 75: Clock configuration

2.4.4.3 Pins Tab

The **Pins** tab provides a graphical tool for configuring the functionality of the pins of the device. For the WDT project no pin configuration is required. Although the project uses two LEDs connected to pins on the device, these pins are pre-configured as output GPIO pins by the BSP.

2.4.4.4 Stacks Tab

You can add any driver to the project using the **Stacks** tab. The HAL driver IO port pins are added automatically by the ISDE when the project is configured. The WDT application uses no RTOS Resources, so you only need to add the HAL WDT driver.

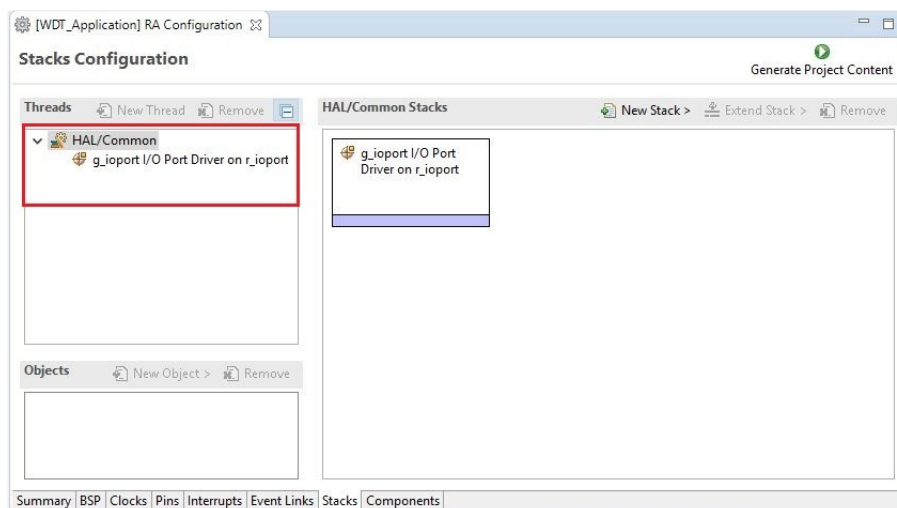


Figure 76: Stacks tab

1. Click on the **HAL/Common Panel** in the Threads Window as indicated in the figure above.

The Stacks Panel becomes a **HAL/Common Stacks** panel and is populated with the modules preselected by the ISDE.

2. Click on **New Stack** to find a pop-up window with the available HAL level drivers.
3. Select **WATCHDOG Driver on r_wdt**.

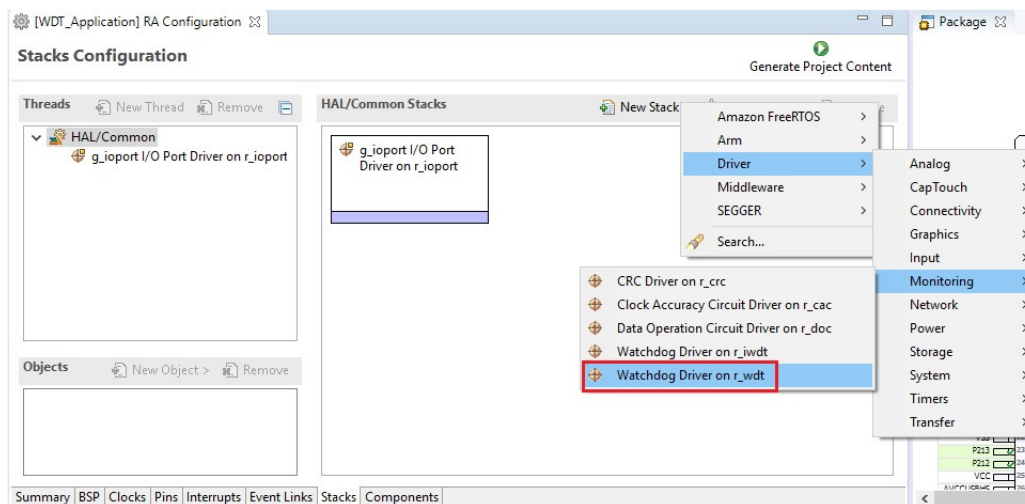


Figure 77: Module Selection

The selected HAL WDT driver is added to the **HAL/Common Stacks** Panel and the **Property** Window shows all configuration options for the selected module. The **Property** tab for the WDT should be visible at the bottom left of the screen. If it is not visible, check that the **RA Configuration** perspective is selected.

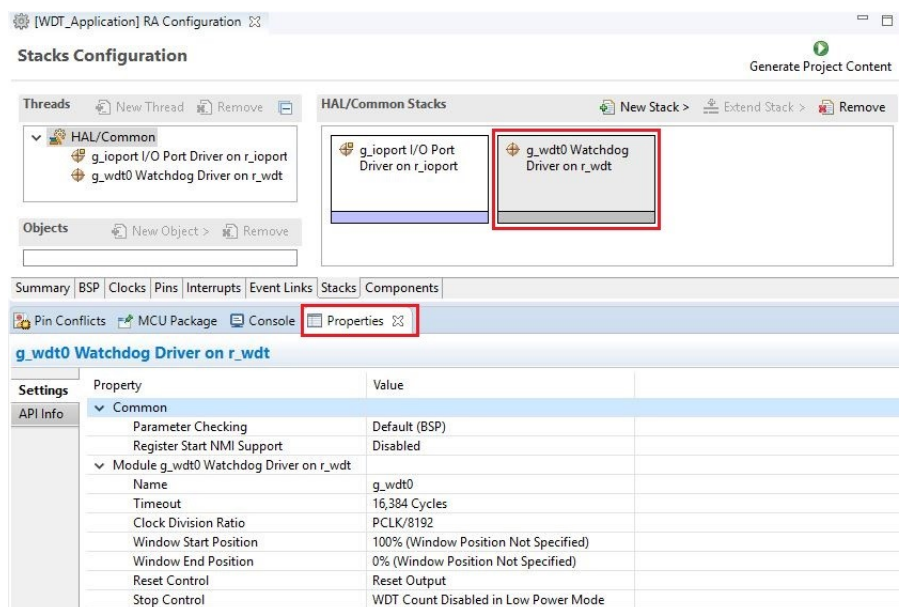


Figure 78: Module Properties

All parameters can be left with their default values.

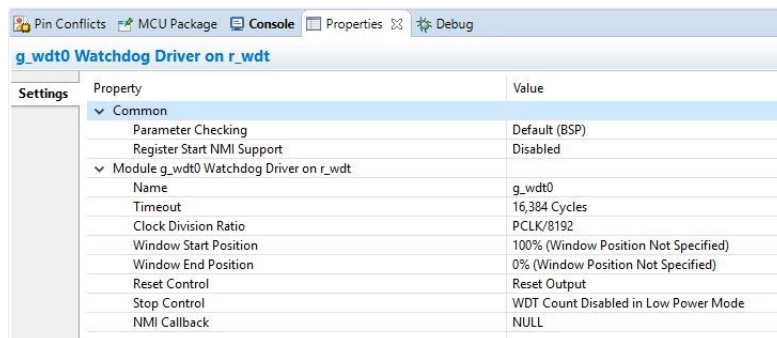


Figure 79: g_wdt WATCHDOG Driver on WDT properties

With PCLKB running at 60 MHz the WDT will reset the device 2.23 seconds after the last refresh.

WDT clock = 60 MHz / 8192 = 7.32 kHz

Cycle time = 1 / 7.324 kHz = 136.53 us

Timeout = 136.53 us x 16384 = 2.23 seconds

Save the **Project Configuration** file and click the **Generate Project Content** button in the top right corner of the **Project Configuration** pane.



Figure 80: Generate Project Content button

The ISDE generates the project files.

2.4.4.5 Components Tab

The components tab is included for reference to see which modules are included in the project. Modules are selected automatically in the Components view after they are added in the Stacks Tab.

For the WDT project ensure that the following modules are selected:

1. HAL_Drivers -> r_ioport
2. HAL_Drivers -> r_wdt

Components Configuration			
Component	Version	Description	Variant
☐ r_iic_master	0.8.0-rc.0	I2C Master Interface	
☐ r_iic_slave	0.8.0-rc.0	I2C Slave Interface	
☒ r_ioport	0.8.0-rc.0	I/O Port	
☐ r_iwdt	0.8.0-rc.0	Independent Watchdog Timer	
☐ r_jpeg	0.8.0-rc.0	JPEG Codec	
☐ r_kint	0.8.0-rc.0	Key Input	
☐ r_lpm	0.8.0-rc.0	Low Power Modes	
☐ r_lvd	0.8.0-rc.0	Low Voltage Detection	
☐ r_rtc	0.8.0-rc.0	Real Time Clock	
☐ r_sce_ra2	0.8.0-rc.0	Secure Cryptography Engine on RA2	
☐ r_sce_ra4	0.8.0-rc.0	Secure Cryptography Engine on RA4	
☐ r_sce_ra6	0.8.0-rc.0	Secure Cryptography Engine on RA6	
☐ r_sci_i2c	0.8.0-rc.0	SCI I2C Master Interface	
☐ r_sci_spi	0.8.0-rc.0	Serial Peripheral Interface on Serial Communic...	
☐ r_sci_uart	0.8.0-rc.0	SCI UART	
☐ r_sdhi	0.8.0-rc.0	SD/MMC Host Interface	
☐ r_spi	0.8.0-rc.0	Serial Peripheral Interface	
☐ r_ssi	0.8.0-rc.0	Serial Sound Interface	
☐ r_usb_basic	0.8.0-rc.0	Universal Serial Bus Basic	
☐ r_usb_pcdc	0.8.0-rc.0	Universal Serial Bus Peripheral Communication...	
☒ r_wdt	0.8.0-rc.0	Watchdog Timer	
☐ rm_freertos_plus_tcp	0.8.0	r_ether to FreeRTOS Plus TCP IP Wrapper	
☐ rm_psa_crypto	0.8.0-rc.0	PSA mbedCrypto	
Summary BSP Clocks Pins Interrupts Event Links Stacks Components			

Figure 81: Component Selection

Note

The list of modules displayed in the Components tab depends on the installed FSP version.

2.4.5 WDT Generated Project Files

Clicking the Generate Project Content button performs the following tasks.

- r_wdt folder and WDT driver contents created at:

ra/fsp/src

- r_wdt_api.h created in:

ra/fsp/inc/api

- r_wdt.h created in:

ra/fsp/inc/instance

The above files are the standard files for the WDT HAL module. They contain no specific project contents. They are the driver files for the WDT. Further information on the contents of these files can be found in the documentation for the WDT HAL module.

Configuration information for the WDT HAL module in the WDT project is found in:

ra_cfg/fsp_cfg/r_wdt_cfg.h

The above file's contents are based upon the **Common** settings in the **g_wdt WATCHDOG Driver on WDT Properties** pane.

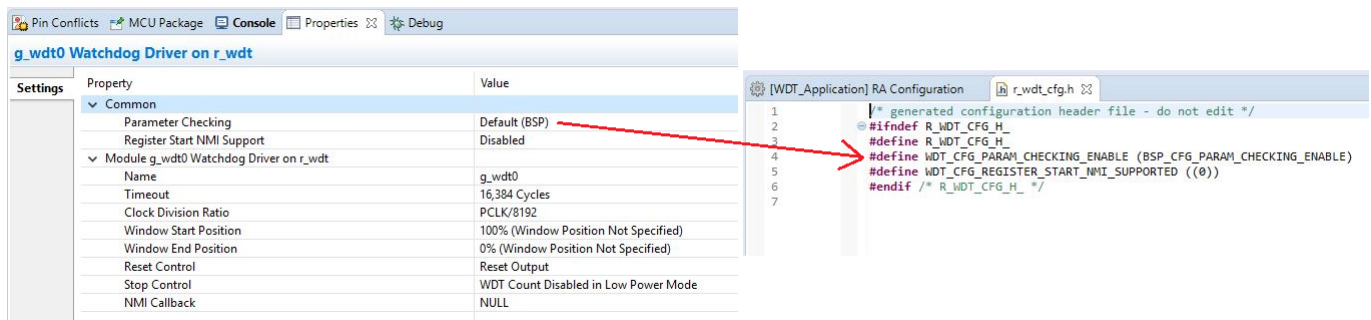


Figure 82: r_wdt_cfg.h contents

Warning

Do not edit any of these files as they are recreated every time the Generate Project Content button is clicked and so any changes will be overwritten.

The r_ioport folder is not created at ra/fsp/src as this module is required by the BSP and so already exists. It is included in the WDT project in order to include the correct header file in ra_gen/hal_data.c-see later in this document for further details. For the same reason the other IOPORT header files- ra/fsp/inc/api/r_ioport_api.h and ra/fsp/inc/instances/r_ioport.h-are not created as they already exist.

In addition to generating the HAL driver files for the WDT and IOPORT files the ISDE also generates files containing configuration data for the WDT and a file where user code can safely be added. These files are shown below.

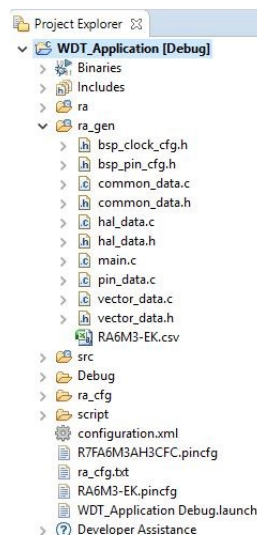


Figure 83: WDT project files

2.4.5.1 WDT hal_data.h

The contents of hal_data.h are shown below.

```
/* generated HAL header file - do not edit */
#ifndef HAL_DATA_H_
```

```
#define HAL_DATA_H_
#include <stdint.h>
#include "bsp_api.h"
#include "common_data.h"
#include "r_wdt.h"
#include "r_wdt_api.h"
#ifdef __cplusplus
extern "C"
{
    #endif
extern const wdt_instance_t g_wdt0;
#ifdef NULL
void NULL(wdt_callback_args_t * p_args);
#endif
extern wdt_instance_ctrl_t g_wdt0_ctrl;
extern const wdt_cfg_t g_wdt0_cfg;
void hal_entry(void);
void g_hal_init(void);
#ifdef __cplusplus
} /* extern "C" */
#endif
#endif /* HAL_DATA_H_ */
```

hal_data.h contains the header files required by the ISDE generated project. In addition this file includes external references to the **g_wdt** instance structure which contains pointers to the configuration, control, api structures used for WDT HAL driver.

Warning

This file is regenerated each time Generate Project Content is clicked and must not be edited.

2.4.5.2 WDT hal_data.c

The contents of hal_data.c are shown below.

```
/* generated HAL source file - do not edit */
#include "hal_data.h"
wdt_instance_ctrl_t g_wdt0_ctrl;
```

```

const wdt_cfg_t g_wdt0_cfg =
{
    .timeout          = WDT_TIMEOUT_16384,
    .clock_division   = WDT_CLOCK_DIVISION_8192,
    .window_start     = WDT_WINDOW_START_100,
    .window_end       = WDT_WINDOW_END_0,
    .reset_control    = WDT_RESET_CONTROL_RESET,
    .stop_control     = WDT_STOP_CONTROL_ENABLE,
    .p_callback       = NULL,
};

/* Instance structure to use this module. */
const wdt_instance_t g_wdt0 =
{.p_ctrl = &g_wdt0_ctrl, .p_cfg = &g_wdt0_cfg, .p_api = &g_wdt_on_wdt};

void g_hal_init (void)
{
    g_common_init();
}

```

hal_data.c contains g_wdt_ctrl which is the control structure for this instance of the WDT HAL driver. This structure should not be initialized as this is done by the driver when it is opened.

The contents of g_wdt_cfg are populated in this file using the **g_wdt WATCHDOG Driver on WDT Properties** pane in the **ISDE Project Configuration HAL** tab. If the contents of this structure do not reflect the settings made in the ISDE, ensure the **Project Configuration** settings are saved in the ISDE before clicking the **Generate Project Content** button.

Warning

This file is regenerated each time Generate Project Content is clicked and so should not be edited.

2.4.5.3 WDT main.c

Contains main() called by the BSP start-up code. main() calls hal_entry() which contains user developed code (see next file). Here are the contents of main.c.

```

/* generated main source file - do not edit*/
#include "hal_data.h"

int main (void)
{
    hal_entry();
}

```

```
return 0;
}
```

Warning

This file is regenerated each time Generate Project Content is clicked and so should not be edited.

2.4.5.4 WDT hal_entry.c

This file contains the function hal_entry() called from main(). User developed code should be placed in this file and function.

For the WDT project edit the contents of this file to contain the code below. This code implements the flowchart in overview section of this document.

```
#include "hal_data.h"
#include "bsp_pin_cfg.h"
#include "r_ioport.h"
#define RED_LED_NO_OF_FLASHES 30
#define RED_LED_PIN BSP_IO_PORT_01_PIN_00
#define GREEN_LED_PIN BSP_IO_PORT_04_PIN_00
#define RED_LED_DELAY_COUNT 1500000
#define GRN_LED_DELAY_COUNT 1200000
volatile uint32_t delay_counter;
volatile uint16_t loop_counter;
void R_BSP_WarmStart(bsp_warm_start_event_t event);
/* global variable to access board LEDs */
extern bsp_leds_t g_bsp_leds;
/*****
*****/
void hal_entry (void) {
    /* Open the WDT */
    R_WDT_Open(&g_wdt0_ctrl, &g_wdt0_cfg);
    /* Start the WDT by refreshing it */
    R_WDT_Refresh(&g_wdt0_ctrl);
    /* Flash the red LED and tickle the WDT for a few seconds */
    for (loop_counter = 0; loop_counter < RED_LED_NO_OF_FLASHES; loop_counter++)
    {
```

```
/* Turn red LED on */
R_IOPORT_PinWrite(&g_ioport_ctrl, RED_LED_PIN, BSP_IO_LEVEL_LOW);
/* Delay */
for (delay_counter = 0; delay_counter < RED_LED_DELAY_COUNT; delay_counter++)
{
/* Do nothing. */
}
/* Refresh WDT */
R_WDT_Refresh(&g_wdt0_ctrl);
R_IOPORT_PinWrite(&g_ioport_ctrl, RED_LED_PIN, BSP_IO_LEVEL_HIGH);
/* Delay */
for (delay_counter = 0; delay_counter < RED_LED_DELAY_COUNT; delay_counter++)
{
/* Do nothing. */
}
/* Refresh WDT */
R_WDT_Refresh(&g_wdt0_ctrl);
}
/* Flash green LED but STOP tickling the WDT. WDT should reset the
 * device */
while (1)
{
/* Turn green LED on */
R_IOPORT_PinWrite(&g_ioport_ctrl, GREEN_LED_PIN, BSP_IO_LEVEL_LOW);
/* Delay */
for (delay_counter = 0; delay_counter < GRN_LED_DELAY_COUNT; delay_counter++)
{
/* Do nothing. */
}
/* Turn green off */
R_IOPORT_PinWrite(&g_ioport_ctrl, GREEN_LED_PIN, BSP_IO_LEVEL_HIGH);
/* Delay */
for (delay_counter = 0; delay_counter < GRN_LED_DELAY_COUNT; delay_counter++)
{
```

```

/* Do nothing. */
    }
}

/*****
*****
void R_BSP_WarmStart (bsp_warm_start_event_t event)
{
    if (BSP_WARM_START_POST_C == event)
    {
        /* C runtime environment and system clocks are setup. */
        /* Configure pins. */
        R_IOPORT_Open(&g_ioport_ctrl, &g_bsp_pin_cfg);
    }
}

```

The WDT HAL driver is called through the interface **g_wdt_on_wdt** defined in **r_wdt.h**. The WDT HAL driver is opened through the open API call using the instance defined in **r_wdt_api.h**:

```

/* Open the WDT */
R_WDT_Open(&g_wdt0_ctrl, &g_wdt0_cfg);

```

The first passed parameter is the pointer to the control structure **g_wdt_ctrl** instantiated in **hal_data.c**. The second parameter is the pointer to the configuration data **g_wdt_cfg** instantiated in the same **hal_data.c** file.

The WDT is started and refreshed through the API call:

```

/* Start the WDT by refreshing it */
R_WDT_Refresh(&g_wdt0_ctrl);

```

Again the first (and only in this case) parameter passed to this API is the pointer to the control structure of this instance of the driver.

2.4.6 Building and Testing the Project

Build the project in the ISDE **Build > Build Project**. The project should build without errors.

To debug the project

1. Connect the JLink debugger between the target board and host PC. Apply power to the board.
2. In the **Project Explorer** pane on the right side of the IDE right-click on the WDT project **WDT_Application** and select **Debug As > Debug Configurations**.
3. Under **Renesas GDB Hardware Debugging** select **WDT_Application Debug** as shown below.

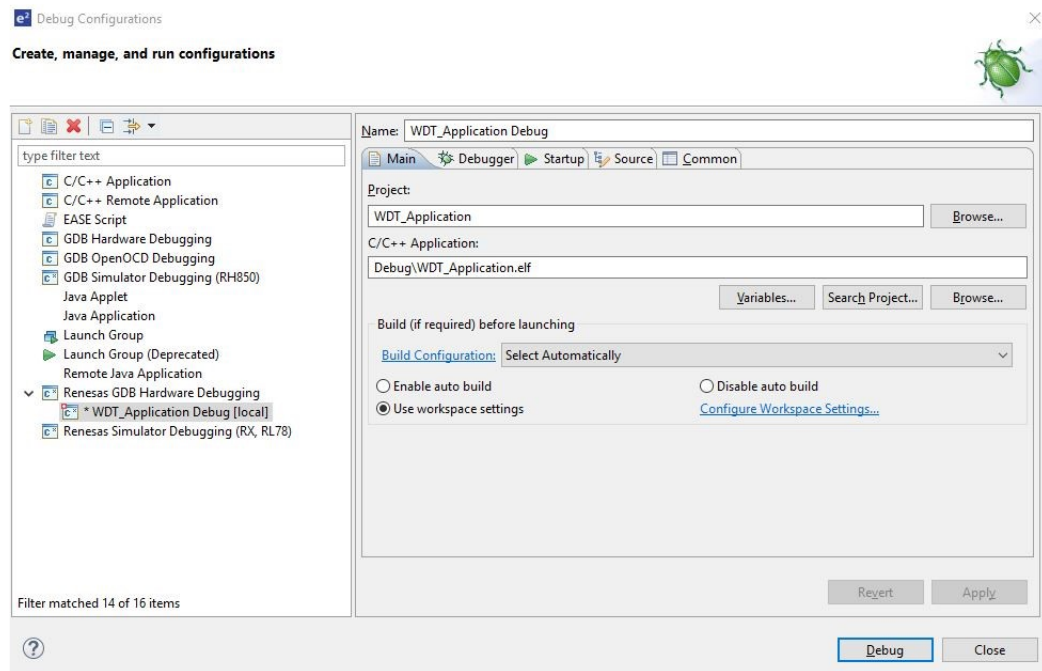
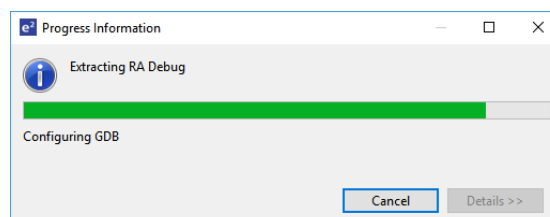


Figure 84: Debug configuration

4. Click the **Debug** button. Click Yes to the debug perspective if asked.



5. The code should run the `Reset_Handler()` function.
6. Resume execution via **Run > Resume**. Execution will stop in `main()` at the call to `hal_entry()`.
7. Resume execution again.

The red LED should start flashing. After 30 flashes the green LED will start flashing and the red LED will stop flashing.

While the green LED is flashing the WDT will underflow and reset the device resulting in the red LED to flash again as the sequence repeats. However, this sequence does not occur when using the debugger because the WDT does not run when connected to the debugger.

1. Stop the debugger in the ISDE via **Run > Terminate**.
2. Click the reset button on the target board. The LEDs begin flashing.

Chapter 3 FSP Architecture

3.1 FSP Architecture Overview

This guide describes the Renesas Flexible Software Package (FSP) architecture and how to use the FSP Application Programming Interface (API).

3.1.1 C99 Use

The FSP uses the ISO/IEC 9899:1999 (C99) C programming language standard. Specific features introduced in C99 that are used include standard integer types (stdint.h), booleans (stdbool.h), designated initializers, and the ability to intermingle declarations and code.

3.1.2 Doxygen

Doxygen is the default documentation tool used by FSP. You can find Doxygen comments throughout the FSP source.

3.1.3 Weak Symbols

Weak symbols are used occasionally in the FSP. They are used to ensure that a project builds even when the user has not defined an optional function.

3.1.4 Memory Allocation

Dynamic memory allocation through use of the malloc() and free() functions are not used in FSP modules; all memory required by FSP modules is allocated in the application and passed to the module in a pointer. Exceptions are considered only for ports of 3rd party code that require dynamic memory.

3.1.5 FSP Terms

Term	Description	Reference
BSP	Short for Board Support Package. In the FSP the BSP provides just enough foundation to allow other FSP modules to work together without issue.	MCU Board Support Package

Module	Modules can be peripheral drivers, purely software, or anything in between. Each module consists of a folder with source code, documentation, and anything else that the customer needs to use the code effectively. Modules are independent units, but they may depend on other modules. Applications can be built by combining multiple modules to provide the user with the features they need.	FSP Modules
Driver	A driver is a specific kind of module that directly modifies registers on the MCU.	-
Interface	An interface contains API definitions that can be shared by modules with similar features. Interfaces are definitions only and do not add to code size.	FSP Interfaces
Stacks	The FSP architecture is designed such that modules work together to form a stack. A stack consists of a top level module and all its dependencies.	FSP Stacks
Module Instance	Single and independent instantiation of a module. An application may require two GPT timers. Each of these timers is a module instance of the r_gpt module.	-
Application	Code that is owned and maintained by the user. Application code may be based on sample application code provided by Renesas, but it is the responsibility of the user to maintain as necessary.	-

Callback Function	This term refers to a function that is called when an event occurs. As an example, suppose the user would like to be notified every second based on the RTC. As part of the RTC configuration, a callback function can be supplied that will be jumped to during each RTC interrupt. When a single callback services multiple events, the arguments contain the triggering event. Callback functions for interrupts should be kept short and handled carefully because when they are called the MCU is still inside of an interrupt, delaying any pending interrupts.	-

3.2 FSP Modules

Modules are the core building block of FSP. Modules can do many different things, but all modules share the basic concept of providing functionality upwards and requiring functionality from below.

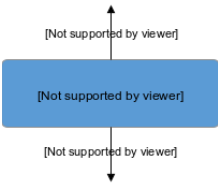


Figure 85: Modules

The amount of functionality provided by a module is determined based on functional use cases. Common functionality required by multiple modules is often placed into a self-contained submodule so it can be reused. Code size, speed and complexity are also considered when defining a module.

The simplest FSP application consists of one module with the Board Support Package (BSP) and the user application on top.

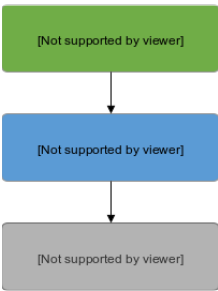


Figure 86: Module with application

The Board Support Package (BSP) is the foundation for FSP modules, providing functionality to determine the MCU used as well as configuring clocks, interrupts and pins. For the sake of clarity, the BSP will be omitted from further diagrams.

3.3 FSP Stacks

When modules are layered atop one another, an FSP stack is formed. The stacking process is performed by matching what one module provides with what another module requires. For example, the SPI module ([Serial Peripheral Interface \(r_spi\)](#)) requires a module that provides the transfer interface ([Transfer Interface](#)) to send or receive data without a CPU interrupt. The transfer interface requirement can be fulfilled by the DTC driver module ([Data Transfer Controller \(r_dtc\)](#)).

Through this methodology the same code can be shared by several modules simultaneously. The example below illustrates how the same DTC module can be used with SPI ([Serial Peripheral Interface \(r_spi\)](#)), UART ([Serial Communications Interface \(SCI\) UART \(r_sci_uart\)](#)) and SDHI ([SD/MMC Host Interface \(r_sdhi\)](#)).

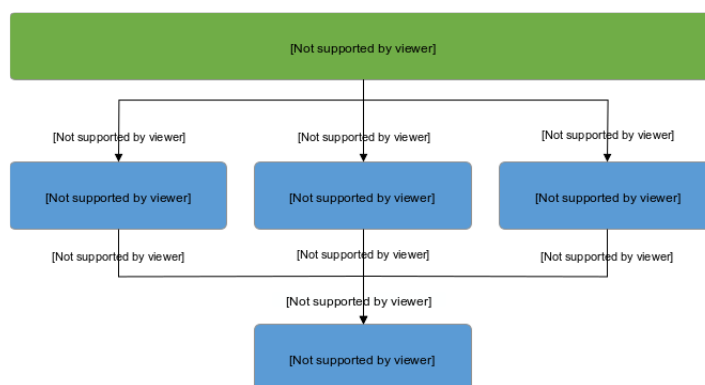


Figure 87: Stacks -- Shared DTC Module

The ability to stack modules ensures the flexibility of the architecture as a whole. If multiple modules include the same functionality issues arise when application features must work across different user designs. To ensure that modules are reusable, any dependent modules must be capable of being swapped out for other modules that provide the same features. The FSP architecture provides this flexibility to swap modules in and out through the use of FSP interfaces.

3.4 FSP Interfaces

At the architecture level, interfaces are the way that modules provide common features. This commonality allows modules that adhere to the same interface to be used interchangeably. Interfaces can be thought of as a contract between two modules - the modules agree to work together using the information that was established in the contract.

On RA hardware there is occasionally an overlap of features between different peripherals. For example, I2C communications can be achieved through use of the IIC peripheral or the SCI peripheral. However, there is a difference in the level of features provided by both peripherals; in I2C mode the SCI peripheral will only support a subset of the capabilities of the fully-featured IIC.

Interfaces aim to provide support for the common features that most users would expect. This means that some of the advanced features of a peripheral (such as IIC) might not be available in the interface. In most cases these features are still available through interface extensions.

In FSP design, interfaces are defined in header files. All interface header files are located in the folder `ra/fsp/inc/api` and end with `*_api.h`. Interface extensions are defined in header files in the folder `ra/fsp/inc/instances`. The following sections detail what makes up an interface.

3.4.1 FSP Interface Enumerations

Whenever possible, interfaces use typed enumerations for function parameters and structure members.

```
typedef enum e_i2c_master_addr_mode
{
    I2C_MASTER_ADDR_MODE_7BIT = 1,    ///< Use 7-bit addressing mode
    I2C_MASTER_ADDR_MODE_10BIT = 2,   ///< Use 10-bit addressing mode
} i2c_master_addr_mode_t;
```

Enumerations remove uncertainty when deciding what values are available for a parameter. FSP enumeration options follow a strict naming convention where the name of the type is prefixed on the available options. Combining the naming convention with the autocomplete feature available in e² studio (Ctrl + Space) provides the benefits of rapid coding while maintaining high readability.

3.4.2 FSP Interface Callback Functions

Callback functions allow modules to asynchronously alert the user application when an event has occurred, such as when a byte has been received over a UART channel or an IRQ pin is toggled. FSP driver modules define and handle the interrupt service routines for RA MCU peripherals to ensure any required hardware procedures are implemented. The interrupt service routines in FSP modules then call the user-defined callbacks to allow the application to respond.

Callback functions must be defined in the user application. They always return void and take a structure for their one parameter. The structure is defined in the interface for the module and is named `<interface>_callback_args_t`. The contents of the structure may vary depending on the interface, but two members are common: `event` and `p_context`.

The `event` member is an enumeration defined in the interface used by the application to determine why the callback was called. Using the UART example, the callback could be triggered for many different reasons, including when a byte is received, all bytes have been transmitted, or a framing error has occurred. The `event` member allows the application to determine which of these three events has occurred and handle it appropriately.

The `p_context` member is used for providing user-specified data to the callback function. In many cases a callback function is shared between multiple channels or module instances; when the callback occurs, the code handling the callback needs context information so that it can determine which module instance the callback is for. For example, if the callback wanted to make a FSP API call in the callback, then at a minimum the callback will need a reference to the relevant control structure. To make this easy, the user can provide a pointer to the control structure as the `p_context`. When the callback occurs, the control structure is passed in the `p_context` element of the callback structure.

Callback functions are called from within an interrupt service routine. For this reason callback functions should be kept as short as possible so they do not affect the real time performance of the

user's system. An example skeleton function for the flash interface callback is shown below.

```
void flash_callback (flash_callback_args_t * p_args)
{
    /* See what event caused this callback. */
    switch (p_args->event)
    {
        case FLASH_EVENT_ERASE_COMPLETE:
            {
                /* Handle event. */
                break;
            }
        case FLASH_EVENT_WRITE_COMPLETE:
            {
                /* Handle event. */
                break;
            }
        case FLASH_EVENT_BLANK:
            {
                /* Handle event. */
                break;
            }
        case FLASH_EVENT_NOT_BLANK:
            {
                /* Handle event. */
                break;
            }
        case FLASH_EVENT_ERR_DF_ACCESS:
            {
                /* Handle error. */
                break;
            }
        case FLASH_EVENT_ERR_CF_ACCESS:
            {
                /* Handle error. */
            }
    }
}
```

```
break;
    }
case FLASH_EVENT_ERR_CMD_LOCKED:
    {
    /* Handle error. */
    break;
    }
case FLASH_EVENT_ERR_FAILURE:
    {
    /* Handle error. */
    break;
    }
case FLASH_EVENT_ERR_ONE_BIT:
    {
    /* Handle error. */
    break;
    }
}
```

When a module is not directly used in the user application (that is, it is not the top layer of the stack), its callback function will be handled by the module above. For example, if a module requires a UART interface module the upper layer module will control and use the UART's callback function. In this case the user would not need to create a callback function for the UART module in their application code.

3.4.3 FSP Interface Data Structures

At a minimum, all FSP interfaces include three data structures: a configuration structure, an API structure, and an instance structure.

3.4.3.1 FSP Interface Configuration Structure

The configuration structure is used for the initial configuration of a module during the <MODULE>_Open() call. The structure consists of members such as channel number, bitrate, and operating mode.

The configuration structure is used purely as an input into the module. It may be stored and referenced by the module, so the configuration structure and anything it references must persist as long as the module is open.

The configuration structure is allocated for each module instance in files generated by the RA configuration tool.

When FSP stacks are used, it is also important to understand that configuration structures only have members that apply to the current interface. If multiple layers in the same stack define the same configuration parameters then it becomes difficult to know where to modify the option. For example, the baud rate for a UART is only defined in the UART module instance. Any modules that use the UART interface rely on the baud rate being provided in the UART module instance and do not offer it in their own configuration structures.

3.4.3.2 FSP Interface API Structure

All interfaces include an API structure which contains function pointers for all the supported interface functions. An example structure for the [Digital to Analog Converter \(r_dac\)](#) is shown below.

```
typedef struct st_dac_api
{
    /** Initial configuration.
     * @par Implemented as
     * - R_DAC_Open()
     * - R_DAC8_Open()
     *
     * @param[in] p_ctrl Pointer to control block. Must be declared by user. Elements
    set here.
     * @param[in] p_cfg Pointer to configuration structure. All elements of this
    structure must be set by user.
     */
    fsp_err_t (* open)(dac_ctrl_t * p_ctrl, dac_cfg_t const * const p_cfg);
    /** Close the D/A Converter.
     * @par Implemented as
     * - R_DAC_Close()
     * - R_DAC8_Close()
     *
     * @param[in] p_ctrl Control block set in dac_api_t::open call for this timer.
     */
    fsp_err_t (* close)(dac_ctrl_t * p_ctrl);
    /** Write sample value to the D/A Converter.
     * @par Implemented as
     * - R_DAC_Write()
     * - R_DAC8_Write()
```



```
*  
  
* @param[in] p_ctrl Control block set in dac_api_t::open call for this timer.  
* @param[in] value Sample value to be written to the D/A Converter.  
*/  
fsp_err_t (* write)(dac_ctrl_t * p_ctrl, uint16_t value);  
/** Start the D/A Converter if it has not been started yet.  
* @par Implemented as  
* - R_DAC_Start()  
* - R_DAC8_Start()  
*  
* @param[in] p_ctrl Control block set in dac_api_t::open call for this timer.  
*/  
fsp_err_t (* start)(dac_ctrl_t * p_ctrl);  
/** Stop the D/A Converter if the converter is running.  
* @par Implemented as  
* - R_DAC_Stop()  
* - R_DAC8_Stop()  
*  
* @param[in] p_ctrl Control block set in dac_api_t::open call for this timer.  
*/  
fsp_err_t (* stop)(dac_ctrl_t * p_ctrl);  
/** Get version and store it in provided pointer p_version.  
* @par Implemented as  
* - R_DAC_VersionGet()  
* - R_DAC8_VersionGet()  
*  
* @param[out] p_version Code and API version used.  
*/  
fsp_err_t (* versionGet)(fsp_version_t * p_version);  
/** Get information about DAC Resolution and store it in provided pointer p_info.  
* @par Implemented as  
* - R_DAC_InfoGet()  
* - R_DAC8_InfoGet()  
*  
*
```

```

* @param[out] p_info Collection of information for this DAC.
*/
fsp_err_t (* infoGet)(dac_info_t * const p_info);
} dac_api_t;

```

The API structure is what allows for modules to easily be swapped in and out for other modules that are instances of the same interface. Let's look at an example application using the DAC interface above.

RA MCUs have an internal DAC peripheral. If the DAC API structure in the DAC interface is not used the application can make calls directly into the module. In the example below the application is making calls to the [R_DAC_Write\(\)](#) function which is provided in the `r_dac` module.

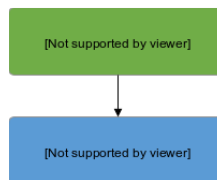


Figure 88: DAC Write example

Now let's assume that the user needs more DAC channels than are available on the MCU and decides to add an external DAC module named `dac_external` using I2C for communications. The application must now distinguish between the two modules, adding complexity and further dependencies to the application.

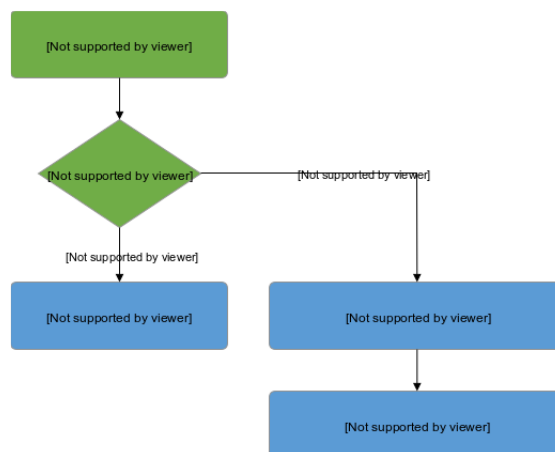


Figure 89: DAC Write with two write modules

The use of interfaces and the API structure allows for the use of an abstracted DAC. This means that no extra logic is needed if the user's `dac_external` module implements the FSP DAC interface, so the application no longer depends upon hard-coded module function names. Instead the application now depends on the DAC interface API which can be implemented by any number of modules.

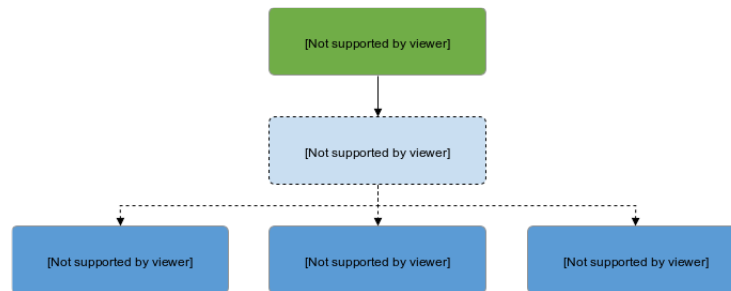


Figure 90: DAC Interface

3.4.3.3 FSP Interface Instance Structure

Every FSP interface also has an instance structure. The instance structure encapsulates everything required to use the module:

- A pointer to the instance API structure ([FSP Instance API](#))
- A pointer to the configuration structure
- A pointer to the control structure

The instance structure is not required at the application layer. It is used to connect modules to their dependencies (other than the BSP).

Instance structures have a standardized name of `<interface>_instance_t`. An example from the [Transfer Interface](#) is shown below.

```
typedef struct st_transfer_instance
{
    transfer_ctrl_t    * p_ctrl; ///< Pointer to the control structure for this
instance
    transfer_cfg_t const * p_cfg;    ///< Pointer to the configuration structure
for this instance
    transfer_api_t const * p_api;    ///< Pointer to the API structure for this
instance
} transfer_instance_t;
```

Note that when an instance structure variable is declared, the API is the only thing that is instance specific, not *module instance* specific. This is because all module instances of the same module share the same underlying module source code. If SPI is being used on SCI channels 0 and 2 then both module instances use the same API while the configuration and control structures are typically different.

3.5 FSP Instances

While interfaces dictate the features that are provided, instances actually implement those features. Each instance is tied to a specific interface. Instances use the enumerations, data structures, and API

prototypes from the interface. This allows an application that uses an interface to swap out the instance when needed.

On RA MCUs some peripherals are used to implement multiple interfaces. In the example below the IIC and SPI peripherals map to only one interface each while the SCI peripheral implements three interfaces.

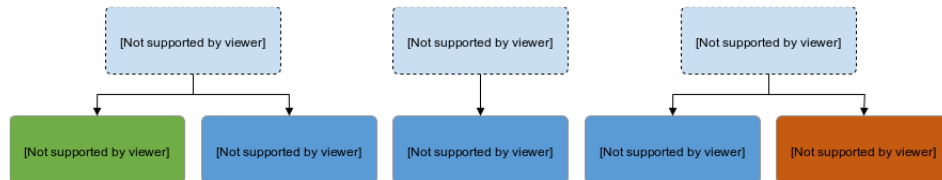


Figure 91: Instances

In FSP design, instances consist of the interface extension and API defined in the instance header file located in the folder `ra/fsp/inc/instances` and the module source `ra/fsp/src/<module>`.

3.5.1 FSP Instance Control Structure

The control structure is used as a unique identifier for the module instance and contains memory required by the module. Elements in the control structure are owned by the module and *must not be modified* by the application. The user allocates storage for a control structure, often as a global variable, then sends a pointer to it into the `<MODULE>_Open()` call for a module. At this point, the module initializes the structure as needed. The user must then send in a pointer to the control structure for all subsequent module calls.

3.5.2 FSP Interface Extensions

In some cases, instances require more information than is provided in the interface. This situation can occur in the following cases:

- An instance offers extra features that are not common to most instances of the interface. An example of this is the start source selection of the GPT ([General PWM Timer \(r_gpt\)](#)). The GPT can be configured to start based on hardware events such as a falling edge on a trigger pin. This feature is not common to all timers, so it is included in the GPT instance.
- An interface must be very generic out of necessity. As an interface becomes more generic, the number of possible instances increases. An example of an interface that must be generic is a block media interface that abstracts functions required by a file system. Possible instances include SD card, SPI Flash, SDRAM, USB, and many more.

The `p_extend` member provides this extension function.

Use of interface extensions is not always necessary. Some instances do not offer an extension since all functionality is provided in the interface. In these cases the `p_extend` member can be set to `NULL`. The documentation for each instance indicates whether an interface extension is available and whether it is mandatory or optional.

3.5.2.1 FSP Extended Configuration Structure

When extended configuration is required it can be supplied through the `p_extend` parameter of the interface configuration structure.

The extended configuration structure is part of the instance, but it is also still considered to be part of the configuration structure. All usage notes about the configuration structure described in [FSP Interface Configuration Structure](#) apply to the extended configuration structure as well.

The extended configuration structure and all typed structures and enumerations required to define it make up the interface extension.

3.5.3 FSP Instance API

Each instance includes a constant global variable tying the interface API functions to the functions provided by the module. The name of this structure is standardized as `g_<interface>_on_<instance>`. Examples include `g_spi_on_spi`, `g_transfer_on_dtc`, and `g_adc_on_adc`. This structure is available to be used through an extern in the instance header file (`r_spi.h`, `r_dtc.h`, and `r_adc.h` respectively).

3.6 FSP API Standards

3.6.1 FSP Function Names

FSP functions start with the uppercase module name (`<MODULE>`). All modules have `<MODULE>_Open()` and `<MODULE>_Close()` functions. The `<MODULE>_Open()` function must be called before any of the other functions. The only exception is the `<MODULE>_VersionGet()` function which is not dependent upon any user provided information.

Other functions that will commonly be found are `<MODULE>_Read()`, `<MODULE>_Write()`, `<MODULE>_InfoGet()`, and `<MODULE>_StatusGet()`. The `<MODULE>_StatusGet()` function provides a status that could change asynchronously, while `<MODULE>_InfoGet()` provides information that cannot change after open or can only be updated by API calls. Example function names include:

- `R_SPI_Read()`, `R_SPI_Write()`, `R_SPI_WriteRead()`
- `R_SDHI_StatusGet()`
- `R_RTC_CalendarAlarmSet()`, `R_RTC_CalendarAlarmGet()`
- `R_FLASH_HP_AccessWindowSet()`, `R_FLASH_HP_AccessWindowClear()`

3.6.2 Use of const in API parameters

The `const` qualifier is used with API parameters whenever possible. An example case is shown below.

```
fsp_err_t R_FLASH_HP_Open(flash_ctrl_t * const p_api_ctrl, flash_cfg_t const * const p_cfg);
```

In this example, `flash_cfg_t` is a structure of configuration parameters for the `r_flash_hp` module. The parameter `p_cfg` is a pointer to this structure. The first `const` qualifier on `p_cfg` ensures the `flash_cfg_t` structure cannot be modified by `R_FLASH_HP_Open()`. This allows the structure to be allocated as a `const` variable and stored in ROM instead of RAM.

The `const` qualifier after the pointer star for both `p_ctrl` and `p_cfg` ensures the FSP function does not modify the input pointer addresses. While not fool-proof by any means this does provide some extra checking inside the FSP code to ensure that arguments that should not be altered are treated as such.

3.6.3 FSP Version Information

All instances supply a `<MODULE>_VersionGet()` function which fills in a structure of type `fsp_version_t`. This structure is made up of two version numbers: one for the interface (the API) and one for the underlying instance that is currently being used.

```
typedef union st_fsp_version
{
    /** Version id */
    uint32_t version_id;
    /** Code version parameters */
    struct
    {
        uint8_t code_version_minor;    ///< Code minor version
        uint8_t code_version_major;    ///< Code major version
        uint8_t api_version_minor;     ///< API minor version
        uint8_t api_version_major;     ///< API major version
    };
} fsp_version_t;
```

The API version ideally never changes, and only rarely if it does. A change to the API may require users to go back and modify their code. The code version (the version of the current instance) may be updated more frequently due to bug fixes, enhancements, and additional features. Changes to the code version typically do not require changes to user code.

3.7 FSP Build Time Configurations

All modules have a build-time configuration header file. Most configuration options are supplied at run time, though options that are rarely used or apply to all instances of a module may be moved to build time. The advantage of using a build-time configuration option is to potentially reduce code size reduction by removing an unused feature.

All modules have a build time option to enable or disable parameter checking for the module. FSP modules check function arguments for validity when possible, though this feature is disabled by default to reduce code size. Enabling it can help catch parameter errors during development and debugging. By default, each module's parameter checking configuration inherits the BSP parameter checking setting (set on the BSP tab of the RA configuration tool). Leaving each module's parameter checking configuration set to Default (BSP) allows parameter checking to be enabled or disabled globally in all FSP code through the parameter checking setting on the BSP tab.

If an error condition can reasonably be avoided it is only checked in a section of code that can be disabled by disabling parameter checking. Most Flex APIs can only return `FSP_SUCCESS` if parameter checking is disabled. An example of an error that cannot be reasonably avoided is the "bus busy" error that occurs when another master is using an I2C bus. This type of error can be returned even if

parameter checking is disabled.

3.8 FSP File Structure

The high-level file structure of an FSP project is shown below.

```
ra_gen
ra
+---fsp
    +---inc
    |   +---api
    |   \---instances
    \---src
        +---bsp
        \---r_module
ra_cfg
+---fsp_cfg
    +---bsp
    +---driver
```

Directly underneath the base ra folder the folders are split into the source and include folders. Include folders are kept separate from the source for easy browsing and easy setup of include paths.

The ra_gen folder contains code generated by the RA configuration tool. This includes global variables for the control structure and configuration structure for each module.

The ra_cfg folder is where configuration header files are stored for each module. See [FSP Build Time Configurations](#) for information on what is provided in these header files.

3.9 FSP Architecture in Practice

3.9.1 FSP Connecting Layers

FSP modules are meant to be both reusable and stackable. It is important to remember that modules are not dependent upon other modules, but upon other interfaces. The user is then free to fulfill the interface using the instance that best fits their needs.

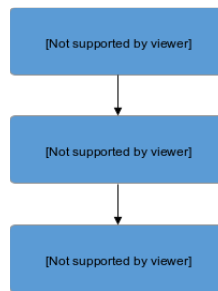


Figure 92: Connecting layers

In the image above interface Y is a dependency of interface X and has its own dependency on interface Z. Interface X only has a dependency on interface Y. Interface X has no knowledge of interface Z. This is a requirement for ensuring that layers can easily be swapped out.

3.9.2 Using FSP Modules in an Application

The typical use of an FSP module involves generating required module data then using the API in the application.

3.9.2.1 Create a Module Instance in the RA Configuration Tool

The RA configuration tool in the Renesas e² studio IDE provides a graphical user interface for setting the parameters of the interface and instance configuration structures. e² studio also automatically includes those structures (once they are configured in the GUI) in application-specific header files that can be included in application code.

The RA configuration tool allocates storage for the control structures, all required configuration structures, and the instance structure in generated files in the ra_gen folder. Use the e² studio **Properties** view to set the values for the members of the configuration structures as needed. Refer to the Configuration section of the module usage notes for documentation about the configuration options.

If the interface has a callback function option then the application must declare and define the function. The return value is always of type void and the parameter to the function is a typed structure of name <interface>_callback_args_t. Once the function has been defined, assign its name to the p_callback member of the configuration structure. Callback function names can be assigned through the e² studio **Properties** window for the selected module.

3.9.2.2 Use the Instance API in the Application

Call the module's <MODULE>_Open() function. Pass pointers to the generated control structure and configuration structure. The names of these structures are based on the 'Name' field provided in the RA configuration tool. The control structure is <Name>_ctrl and the configuration structure is <Name>_cfg. An example <MODULE>_Open() call for an r_rtc module instance named g_clock is:

```
R_RTC_Open(&g_clock_ctrl, &g_clock_cfg);
```

Note

Each layer in the FSP Stack is responsible for calling the API functions of its dependencies. This means that users

are only responsible for calling the API functions at the layer at which they are interfacing. Using the example above of a SPI module with a DTC dependency, the application uses only SPI APIs. The application starts by calling [R_SPI_Open\(\)](#). Internally, the SPI module opens the DTC. It locates [R_DTC_Open\(\)](#) by accessing the dependent transfer interface function pointers from the pointers DTC instances ([spi_cfg_t::p_transfer_tx](#) and [spi_cfg_t::p_transfer_rx](#)) to open the DTC.

Refer to the module usage notes for example code to help get started with any particular module.

Chapter 4 API Reference

This section includes the FSP API Reference for the Module and Interface level functions.

▼BSP

Common code shared by FSP drivers

Common Error Codes

▼MCU Board Support Package

The BSP is responsible for getting the MCU from reset to the user's application. Before reaching the user's application, the BSP sets up the stacks, heap, clocks, interrupts, C runtime environment, and stack monitor

RA2A1

RA4M1

RA6M1

RA6M2

RA6M3

BSP I/O access

This module provides basic read/write access to port pins

▼Modules

Modules are the smallest unit of software available in the FSP. Each module implements one interface

High-Speed Analog Comparator (r_acmphs)

This module implements the [Comparator Interface](#) using the high-speed analog comparator

Low-Power Analog Comparator (r_acmplp)

Driver for the ACMPLP peripheral on RA MCUs. This module implements the [Comparator Interface](#)

Analog to Digital Converter (r_adc)

Driver for the ADC12, ADC14, and ADC16 peripherals on RA MCUs. This module implements the [ADC Interface](#)

Asynchronous General Purpose Timer (r_agt)

Driver for the AGT peripheral on RA MCUs. This module implements the [Timer Interface](#)

Clock Frequency Accuracy Measurement Circuit (r_cac)

Driver for the CAC peripheral on RA MCUs. This module implements the [CAC Interface](#)

Clock Generation Circuit (r_cgc)

Driver for the CGC peripheral on RA MCUs. This module implements the [CGC Interface](#)

Cyclic Redundancy Check (CRC) Calculator (r_crc)

Driver for the CRC peripheral on RA MCUs. This module implements the [CRC Interface](#)

Capacitive Touch Sensing Unit (r_cts)

This HAL driver supports the Capacitive Touch Sensing Unit (CTS). It implements the [CTS](#)

	Interface
Digital to Analog Converter (r_dac)	Driver for the DAC12 peripheral on RA MCUs. This module implements the DAC Interface
Direct Memory Access Controller (r_dmac)	Driver for the DMAC peripheral on RA MCUs. This module implements the Transfer Interface
Data Operation Circuit (r_doc)	Driver for the DOC peripheral on RA MCUs. This module implements the DOC Interface
D/AVE 2D Port Interface (r_drw)	Driver for the DRW peripheral on RA MCUs. This module is a port of D/AVE 2D
Data Transfer Controller (r_dtc)	Driver for the DTC peripheral on RA MCUs. This module implements the Transfer Interface
Event Link Controller (r_elc)	Driver for the ELC peripheral on RA MCUs. This module implements the ELC Interface
Ethernet (r_ether)	Driver for the Ethernet peripheral on RA MCUs. This module implements the Ethernet Interface
Ethernet PHY (r_ether_phy)	The Ethernet PHY module (r_ether_phy) provides an API for standard Ethernet PHY communications applications and uses the ETHERC peripherals. It implements the Ethernet PHY Interface
High-Performance Flash Driver (r_flash_hp)	Driver for the flash memory on RA high-performance MCUs. This module implements the Flash Interface
Low-Power Flash Driver (r_flash_lp)	Driver for the flash memory on RA low-power MCUs. This module implements the Flash Interface
Graphics LCD Controller (r_glcdc)	Driver for the GLCDC peripheral on RA MCUs. This module implements the Display Interface
General PWM Timer (r_gpt)	Driver for the GPT32 and GPT16 peripherals on RA MCUs. This module implements the Timer Interface
Interrupt Controller Unit (r_icu)	Driver for the ICU peripheral on RA MCUs. This module implements the External IRQ Interface
I2C Master on IIC (r_iic_master)	Driver for the IIC peripheral on RA MCUs. This module implements the I2C Master Interface
I2C Slave on IIC (r_iic_slave)	Driver for the IIC peripheral on RA MCUs. This module implements the I2C Slave Interface
I/O Ports (r_ioport)	Driver for the I/O Ports peripheral on RA MCUs. This module implements the I/O Port Interface
Independent Watchdog Timer (r_iwdt)	Driver for the IWDT peripheral on RA MCUs. This module implements the WDT Interface
JPEG Codec (r_jpeg)	Driver for the JPEG peripheral on RA MCUs. This module implements the JPEG Codec Interface

Key Interrupt (r_kint)	Driver for the KINT peripheral on RA MCUs. This module implements the Key Matrix Interface
Low Power Modes (r_lpm)	Driver for the LPM peripheral on RA MCUs. This module implements the Low Power Modes Interface
Low Voltage Detection (r_lvd)	Driver for the LVD peripheral on RA MCUs. This module implements the Low Voltage Detection Interface
Realtime Clock (r_rtc)	Driver for the RTC peripheral on RA MCUs. This module implements the RTC Interface
Serial Communications Interface (SCI) I2C (r_sci_i2c)	Driver for the SCI peripheral on RA MCUs. This module implements the I2C Master Interface
Serial Communications Interface (SCI) SPI (r_sci_spi)	Driver for the SCI peripheral on RA MCUs. This module implements the SPI Interface
Serial Communications Interface (SCI) UART (r_sci_uart)	Driver for the SCI peripheral on RA MCUs. This module implements the UART Interface
SD/MMC Host Interface (r_sdhi)	Driver for the SD/MMC Host Interface (SDHI) peripheral on RA MCUs. This module implements the SD/MMC Interface
Serial Peripheral Interface (r_spi)	Driver for the SPI peripheral on RA MCUs. This module implements the SPI Interface
Serial Sound Interface (r_ssi)	Driver for the SSIE peripheral on RA MCUs. This module implements the I2S Interface
Universal Serial Bus (r_usb_basic)	The USB module (r_usb_basic) provides an API to perform H / W control of USB communication. It implements the USB Interface
Host Mass Storage Class Driver (r_usb_hmsc)	The USB module (r_usb_hmsc) provides an API to perform hardware control of USB communications. It implements the USB Interface
Universal Serial Bus Peripheral Communication Device Class (r_usb_pcdc)	This module is USB Peripheral Communication Device Class Driver (PCDC). This module works in combination with (r_usb_basic module)
Watchdog Timer (r_wdt)	Driver for the WDT peripheral on RA MCUs. This module implements the WDT Interface
SEGGER emWin Port (rm_emwin_port)	SEGGER emWin port for RA MCUs
FreeRTOS Plus FAT (rm_freertos_plus_fat)	Middleware for the Fat File System control on RA MCUs
Amazon FreeRTOS Port (rm_freertos_port)	Amazon FreeRTOS port for RA MCUs
Crypto Middleware (rm_psa_crypto)	Hardware acceleration for the mbedCrypto implementation of the ARM PSA Crypto API
Capacitive Touch Middleware (rm_touch)	This module supports the Capacitive Touch Sensing Unit (CTSU). It implements the Touch

▼Interfaces

ADC Interface

CAC Interface

CGC Interface

Comparator Interface

CRC Interface

CTSU Interface

DAC Interface

Display Interface

DOC Interface

ELC Interface

Ethernet Interface

Ethernet PHY Interface

External IRQ Interface

Flash Interface

I2C Master Interface

I2C Slave Interface

I2S Interface

I/O Port Interface

JPEG Codec Interface

Key Matrix Interface

Low Power Modes Interface

Low Voltage Detection Interface

RTC Interface

SD/MMC Interface

SPI Interface

Timer Interface

Transfer Interface

Middleware Interface

The FSP interfaces provide APIs for common functionality. They can be implemented by one or more modules. Modules can use other modules as dependencies using this interface layer

Interface for A/D Converters

Interface for clock frequency accuracy measurements

Interface for clock generation

Interface for comparators

Interface for cyclic redundancy checking

Interface for Capacitive Touch Sensing Unit (CTSU) functions

Interface for D/A converters

Interface for LCD panel displays

Interface for the Data Operation Circuit

Interface for the Event Link Controller

Interface for Ethernet functions

Interface for Ethernet phy functions

Interface for detecting external interrupts

Interface for the Flash Memory

Interface for I2C master communication

Interface for I2C slave communication

Interface for I2S audio communication

Interface for accessing I/O ports and configuring I/O functionality

Interface for JPEG functions

Interface for key matrix functions

Interface for accessing low power modes

Interface for Low Voltage Detection

Interface for accessing the Realtime Clock

Interface for accessing SD, eMMC, and SDIO devices

Interface for SPI communications

Interface for timer functions

Interface for data transfer functions

UART Interface	Interface for UART communications
USB Interface	Interface for USB functions
USB HMSC Interface	Interface for USB HMSC functions
USB PCDC Interface	Interface for USB PCDC functions
WDT Interface	Interface for watch dog timer functions
Touch Middleware Interface	Interface for Touch Middleware functions

4.1 BSP

Detailed Description

Common code shared by FSP drivers.

Modules

[Common Error Codes](#)

[MCU Board Support Package](#)

The BSP is responsible for getting the MCU from reset to the user's application. Before reaching the user's application, the BSP sets up the stacks, heap, clocks, interrupts, C runtime environment, and stack monitor.

[BSP I/O access](#)

This module provides basic read/write access to port pins.

4.1.1 Common Error Codes

[BSP](#)

Detailed Description

All FSP modules share these common error codes.

Data Structures

union [fsp_version_t](#)

struct [fsp_version_t.__unnamed__](#)

Macros

```
#define FSP_PARAMETER_NOT_USED(p)
```

```
#define FSP_CPP_HEADER
```

```
#define FSP_HEADER
```

Enumerations

```
enum fsp_err_t
```

Data Structure Documentation

◆ fsp_version_t

union fsp_version_t		
Common version structure		
Data Fields		
uint32_t	version_id	Version id
struct fsp_version_t	__unnamed__	Code version parameters

◆ fsp_version_t.__unnamed__

struct fsp_version_t.__unnamed__		
Code version parameters		
Data Fields		
uint8_t	code_version_minor	Code minor version.
uint8_t	code_version_major	Code major version.
uint8_t	api_version_minor	API minor version.
uint8_t	api_version_major	API major version.

Macro Definition Documentation

◆ FSP_PARAMETER_NOT_USED

```
#define FSP_PARAMETER_NOT_USED ( p)
```

This macro is used to suppress compiler messages about a parameter not being used in a function. The nice thing about using this implementation is that it does not take any extra RAM or ROM.

◆ **FSP_CPP_HEADER**

#define FSP_CPP_HEADER

Determine if a C++ compiler is being used. If so, ensure that standard C is used to process the API information.

◆ **FSP_HEADER**

#define FSP_HEADER

FSP Header and Footer definitions

Enumeration Type Documentation◆ **fsp_err_t**

enum fsp_err_t

Common error codes

Enumerator

FSP_ERR_ASSERTION	A critical assertion has failed.
FSP_ERR_INVALID_POINTER	Pointer points to invalid memory location.
FSP_ERR_INVALID_ARGUMENT	Invalid input parameter.
FSP_ERR_INVALID_CHANNEL	Selected channel does not exist.
FSP_ERR_INVALID_MODE	Unsupported or incorrect mode.
FSP_ERR_UNSUPPORTED	Selected mode not supported by this API.
FSP_ERR_NOT_OPEN	Requested channel is not configured or API not open.
FSP_ERR_IN_USE	Channel/peripheral is running/busy.
FSP_ERR_OUT_OF_MEMORY	Allocate more memory in the driver's cfg.h.
FSP_ERR_HW_LOCKED	Hardware is locked.
FSP_ERR_IRQ_BSP_DISABLED	IRQ not enabled in BSP.
FSP_ERR_OVERFLOW	Hardware overflow.
FSP_ERR_UNDERFLOW	

	Hardware underflow.
FSP_ERR_ALREADY_OPEN	Requested channel is already open in a different configuration.
FSP_ERR_APPROXIMATION	Could not set value to exact result.
FSP_ERR_CLAMPED	Value had to be limited for some reason.
FSP_ERR_INVALID_RATE	Selected rate could not be met.
FSP_ERR_ABORTED	An operation was aborted.
FSP_ERR_NOT_ENABLED	Requested operation is not enabled.
FSP_ERR_TIMEOUT	Timeout error.
FSP_ERR_INVALID_BLOCKS	Invalid number of blocks supplied.
FSP_ERR_INVALID_ADDRESS	Invalid address supplied.
FSP_ERR_INVALID_SIZE	Invalid size/length supplied for operation.
FSP_ERR_WRITE_FAILED	Write operation failed.
FSP_ERR_ERASE_FAILED	Erase operation failed.
FSP_ERR_INVALID_CALL	Invalid function call is made.
FSP_ERR_INVALID_HW_CONDITION	Detected hardware is in invalid condition.
FSP_ERR_INVALID_FACTORY_FLASH	Factory flash is not available on this MCU.
FSP_ERR_INVALID_STATE	API or command not valid in the current state.
FSP_ERR_NOT_ERASED	Erase verification failed.
FSP_ERR_SECTOR_RELEASE_FAILED	Sector release failed.
FSP_ERR_INTERNAL	Internal error. Start of RTOS only error codes
FSP_ERR_WAIT_ABORTED	Wait.
FSP_ERR_FRAMING	Framing error occurs. Start of UART specific
FSP_ERR_BREAK_DETECT	Break signal detects.

FSP_ERR_PARITY	Parity error occurs.
FSP_ERR_RXBUF_OVERFLOW	Receive queue overflow.
FSP_ERR_QUEUE_UNAVAILABLE	Can't open s/w queue.
FSP_ERR_INSUFFICIENT_SPACE	Not enough space in transmission circular buffer.
FSP_ERR_INSUFFICIENT_DATA	Not enough data in receive circular buffer.
FSP_ERR_TRANSFER_ABORTED	The data transfer was aborted. Start of SPI specific
FSP_ERR_MODE_FAULT	Mode fault error.
FSP_ERR_READ_OVERFLOW	Read overflow.
FSP_ERR_SPI_PARITY	Parity error.
FSP_ERR_OVERRUN	Overrun error.
FSP_ERR_CLOCK_INACTIVE	Inactive clock specified as system clock. Start of CGC Specific
FSP_ERR_CLOCK_ACTIVE	Active clock source cannot be modified without stopping first.
FSP_ERR_NOT_STABILIZED	Clock has not stabilized after its been turned on/off.
FSP_ERR_PLL_SRC_INACTIVE	PLL initialization attempted when PLL source is turned off.
FSP_ERR_OSC_STOP_DET_ENABLED	Illegal attempt to stop LOCO when Oscillation stop is enabled.
FSP_ERR_OSC_STOP_DETECTED	The Oscillation stop detection status flag is set.
FSP_ERR_OSC_STOP_CLOCK_ACTIVE	Attempt to clear Oscillation Stop Detect Status with PLL/MAIN_OSC active.
FSP_ERR_CLKOUT_EXCEEDED	Output on target output clock pin exceeds maximum supported limit.
FSP_ERR_USB_MODULE_ENABLED	USB clock configure request with USB Module enabled.

FSP_ERR_HARDWARE_TIMEOUT	A register read or write timed out.
FSP_ERR_LOW_VOLTAGE_MODE	Invalid clock setting attempted in low voltage mode.
FSP_ERR_PE_FAILURE	Unable to enter Programming mode. Start of FLASH Specific
FSP_ERR_CMD_LOCKED	Peripheral in command locked state.
FSP_ERR_FCLK	FCLK must be ≥ 4 MHz.
FSP_ERR_INVALID_LINKED_ADDRESS	Function or data are linked at an invalid region of memory.
FSP_ERR_BLANK_CHECK_FAILED	Blank check operation failed.
FSP_ERR_INVALID_CAC_REF_CLOCK	Measured clock rate < reference clock rate. Start of CAC Specific
FSP_ERR_CLOCK_GENERATION	Clock cannot be specified as system clock. Start of GLCD Specific
FSP_ERR_INVALID_TIMING_SETTING	Invalid timing parameter.
FSP_ERR_INVALID_LAYER_SETTING	Invalid layer parameter.
FSP_ERR_INVALID_ALIGNMENT	Invalid memory alignment found.
FSP_ERR_INVALID_GAMMA_SETTING	Invalid gamma correction parameter.
FSP_ERR_INVALID_LAYER_FORMAT	Invalid color format in layer.
FSP_ERR_INVALID_UPDATE_TIMING	Invalid timing for register update.
FSP_ERR_INVALID_CLUT_ACCESS	Invalid access to CLUT entry.
FSP_ERR_INVALID_FADE_SETTING	Invalid fade-in/fade-out setting.
FSP_ERR_INVALID_BRIGHTNESS_SETTING	Invalid gamma correction parameter.
FSP_ERR_JPEG_ERR	JPEG error. Start of JPEG Specific
FSP_ERR_JPEG_SOI_NOT_DETECTED	SOI not detected until EOI detected.
FSP_ERR_JPEG_SOF1_TO_SOFF_DETECTED	

	SOF1 to SOFF detected.
FSP_ERR_JPEG_UNSUPPORTED_PIXEL_FORMAT	Unprovided pixel format detected.
FSP_ERR_JPEG_SOF_ACCURACY_ERROR	SOF accuracy error: other than 8 detected.
FSP_ERR_JPEG_DQT_ACCURACY_ERROR	DQT accuracy error: other than 0 detected.
FSP_ERR_JPEG_COMPONENT_ERROR1	Component error1: the number of SOF0 header components detected is other than 1,3,or 4.
FSP_ERR_JPEG_COMPONENT_ERROR2	Component error2: the number of components differs between SOF0 header and SOS.
FSP_ERR_JPEG_SOF0_DQT_DHT_NOT_DETECTED	SOF0, DQT, and DHT not detected when SOS detected.
FSP_ERR_JPEG_SOS_NOT_DETECTED	SOS not detected: SOS not detected until EOI detected.
FSP_ERR_JPEG_EOI_NOT_DETECTED	EOI not detected (default)
FSP_ERR_JPEG_RESTART_INTERVAL_DATA_NUMBER_ERROR	Restart interval data number error detected.
FSP_ERR_JPEG_IMAGE_SIZE_ERROR	Image size error detected.
FSP_ERR_JPEG_LAST_MCU_DATA_NUMBER_ERROR	Last MCU data number error detected.
FSP_ERR_JPEG_BLOCK_DATA_NUMBER_ERROR	Block data number error detected.
FSP_ERR_JPEG_BUFFERSIZE_NOT_ENOUGH	User provided buffer size not enough.
FSP_ERR_JPEG_UNSUPPORTED_IMAGE_SIZE	JPEG Image size is not aligned with MCU.
FSP_ERR_CALIBRATE_FAILED	Calibration failed. Start of touch panel framework specific
FSP_ERR_IP_HARDWARE_NOT_PRESENT	Requested IP does not exist on this device. Start of IP specific
FSP_ERR_IP_UNIT_NOT_PRESENT	Requested unit does not exist on this device.
FSP_ERR_IP_CHANNEL_NOT_PRESENT	Requested channel does not exist on this device.
FSP_ERR_USB_FAILED	Start of USB specific
FSP_ERR_NO_MORE_BUFFER	

	No more buffer found in the memory block pool. Start of Message framework specific
FSP_ERR_ILLEGAL_BUFFER_ADDRESS	Buffer address is out of block memory pool.
FSP_ERR_INVALID_WORKBUFFER_SIZE	Work buffer size is invalid.
FSP_ERR_INVALID_MSG_BUFFER_SIZE	Message buffer size is invalid.
FSP_ERR_TOO_MANY_BUFFERS	Number of buffer is too many.
FSP_ERR_NO_SUBSCRIBER_FOUND	No message subscriber found.
FSP_ERR_MESSAGE_QUEUE_EMPTY	No message found in the message queue.
FSP_ERR_MESSAGE_QUEUE_FULL	No room for new message in the message queue.
FSP_ERR_ILLEGAL_SUBSCRIBER_LISTS	Message subscriber lists is illegal.
FSP_ERR_BUFFER_RELEASED	Buffer has been released.
FSP_ERR_D2D_ERROR_INIT	Dave/2d has an error in the initialization. Start of 2DG Driver specific
FSP_ERR_D2D_ERROR_DEINIT	Dave/2d has an error in the initialization.
FSP_ERR_D2D_ERROR_RENDERING	Dave/2d has an error in the rendering.
FSP_ERR_D2D_ERROR_SIZE	Dave/2d has an error in the rendering.
FSP_ERR_ETHER_ERROR_NO_DATA	No Data in Receive buffer. Start of ETHER Driver specific
FSP_ERR_ETHER_ERROR_LINK	ETHERC/EDMAC has an error in the Auto-negotiation.
FSP_ERR_ETHER_ERROR_MAGIC_PACKET_MODE	As a Magic Packet is being detected, and transmission/reception is not enabled.
FSP_ERR_ETHER_ERROR_TRANSMIT_BUFFER_FULL	Transmit buffer is not empty.
FSP_ERR_ETHER_ERROR_FILTERING	Detect multicast frame when multicast frame filtering enable.
FSP_ERR_ETHER_ERROR_PHY_COMMUNICATION	ETHERC/EDMAC has an error in the phy

	communication.
FSP_ERR_ETHER_PHY_ERROR_LINK	PHY is not link up. Start of ETHER_PHY Driver specific
FSP_ERR_ETHER_PHY_NOT_READY	PHY has an error in the Auto-negotiation.
FSP_ERR_QUEUE_FULL	Queue is full, cannot queue another data. Start of BYTEQ library specific
FSP_ERR_QUEUE_EMPTY	Queue is empty, no data to dequeue.
FSP_ERR_CTSU_SC_OVERFLOW	Sensor count overflowed when performing CTSU scan. <i>Note</i> User must clear the CTSUSCOVF bit manually.
FSP_ERR_CTSU_RC_OVERFLOW	Reference count overflowed when performing CTSU scan. <i>Note</i> User must clear the CTSURCOVF bit manually.
FSP_ERR_CTSU_ICOMP	Abnormal TSCAP voltage. <i>Note</i> User must clear the CTSUICOMP bit manually.
FSP_ERR_CTSU_OFFSET_ADJUSTMENT_FAILED	Auto tuning algorithm failed.
FSP_ERR_CTSU_SAFETY_CHECK_FAILED	Safety check failed
FSP_ERR_CARD_INIT_FAILED	SD card or eMMC device failed to initialize. Start of SDMMC specific
FSP_ERR_CARD_NOT_INSERTED	SD card not installed.
FSP_ERR_DEVICE_BUSY	Device is holding DAT0 low or another operation is ongoing.
FSP_ERR_CARD_NOT_INITIALIZED	SD card was removed.
FSP_ERR_CARD_WRITE_PROTECTED	Media is write protected.
FSP_ERR_TRANSFER_BUSY	Transfer in progress.
FSP_ERR_RESPONSE	Card did not respond or responded with an error.

FSP_ERR_MEDIA_FORMAT_FAILED	Media format failed. Start of FX_IO specific
FSP_ERR_MEDIA_OPEN_FAILED	Media open failed.
FSP_ERR_CAN_DATA_UNAVAILABLE	No data available. Start of CAN specific
FSP_ERR_CAN_MODE_SWITCH_FAILED	Switching operation modes failed.
FSP_ERR_CAN_INIT_FAILED	Hardware initialization failed.
FSP_ERR_CAN_TRANSMIT_NOT_READY	Transmit in progress.
FSP_ERR_CAN_RECEIVE_MAILBOX	Mailbox is setup as a receive mailbox.
FSP_ERR_CAN_TRANSMIT_MAILBOX	Mailbox is setup as a transmit mailbox.
FSP_ERR_CAN_MESSAGE_LOST	Receive message has been overwritten or overrun.
FSP_ERR_WIFI_CONFIG_FAILED	WiFi module Configuration failed. Start of SF_WIFI Specific
FSP_ERR_WIFI_INIT_FAILED	WiFi module initialization failed.
FSP_ERR_WIFI_TRANSMIT_FAILED	Transmission failed.
FSP_ERR_WIFI_INVALID_MODE	API called when provisioned in client mode.
FSP_ERR_WIFI_FAILED	WiFi Failed.
FSP_ERR_CELLULAR_CONFIG_FAILED	Cellular module Configuration failed. Start of SF_CELLULAR Specific
FSP_ERR_CELLULAR_INIT_FAILED	Cellular module initialization failed.
FSP_ERR_CELLULAR_TRANSMIT_FAILED	Transmission failed.
FSP_ERR_CELLULAR_FW_UPTODATE	Firmware is uptodate.
FSP_ERR_CELLULAR_FW_UPGRADE_FAILED	Firmware upgrade failed.
FSP_ERR_CELLULAR_FAILED	Cellular Failed.
FSP_ERR_CELLULAR_INVALID_STATE	API Called in invalid state.

FSP_ERR_CELLULAR_REGISTRATION_FAILED	Cellular Network registration failed.
FSP_ERR_BLE_FAILED	BLE operation failed. Start of SF_BLE specific
FSP_ERR_BLE_INIT_FAILED	BLE device initialization failed.
FSP_ERR_BLE_CONFIG_FAILED	BLE device configuration failed.
FSP_ERR_BLE_PRF_ALREADY_ENABLED	BLE device Profile already enabled.
FSP_ERR_BLE_PRF_NOT_ENABLED	BLE device not enabled.
FSP_ERR_CRYPTO_CONTINUE	Continue executing function. Start of Crypto specific (0x10000) <i>Note</i> <i>Refer to sf_cryoto_err.h for Crypto error code.</i>
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	Hardware resource busy.
FSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty.
FSP_ERR_CRYPTO_SCE_HRK_INVALID_INDEX	Invalid index.
FSP_ERR_CRYPTO_SCE_RETRY	Retry.
FSP_ERR_CRYPTO_SCE_VERIFY_FAIL	Verify is failed.
FSP_ERR_CRYPTO_SCE_ALREADY_OPEN	HW SCE module is already opened.
FSP_ERR_CRYPTO_NOT_OPEN	Hardware module is not initialized.
FSP_ERR_CRYPTO_UNKNOWN	Some unknown error occurred.
FSP_ERR_CRYPTO_NULL_POINTER	Null pointer input as a parameter.
FSP_ERR_CRYPTO_NOT_IMPLEMENTED	Algorithm/size not implemented.
FSP_ERR_CRYPTO_RNG_INVALID_PARAM	An invalid parameter is specified.
FSP_ERR_CRYPTO_RNG_FATAL_ERROR	A fatal error occurred.
FSP_ERR_CRYPTO_INVALID_SIZE	Size specified is invalid.
FSP_ERR_CRYPTO_INVALID_STATE	Function used in an valid state.
FSP_ERR_CRYPTO_ALREADY_OPEN	

	control block is already opened
FSP_ERR_CRYPT0_INSTALL_KEY_FAILED	Specified input key is invalid.
FSP_ERR_CRYPT0_AUTHENTICATION_FAILED	Authentication failed.
FSP_ERR_CRYPT0_COMMON_NOT_OPENED	Crypto Framework Common is not opened. Start of SF_CRYPT0 specific
FSP_ERR_CRYPT0_HAL_ERROR	Crypto HAL module returned an error.
FSP_ERR_CRYPT0_KEY_BUF_NOT_ENOUGH	Key buffer size is not enough to generate a key.
FSP_ERR_CRYPT0_BUF_OVERFLOW	Attempt to write data larger than what the buffer can hold.
FSP_ERR_CRYPT0_INVALID_OPERATION_MODE	Invalid operation mode.
FSP_ERR_MESSAGE_TOO_LONG	Message for RSA encryption is too long.
FSP_ERR_RSA_DECRYPTION_ERROR	RSA Decryption error.

4.1.2 MCU Board Support Package

BSP

Functions

`fsp_err_t` [R_FSP_VersionGet](#) (`fsp_pack_version_t *const p_version`)

`void` [Reset_Handler](#) (`void`)

`void` [Default_Handler](#) (`void`)

`void` [SystemInit](#) (`void`)

`void` [R_BSP_WarmStart](#) (`bsp_warm_start_event_t event`)

`fsp_err_t` [R_BSP_VersionGet](#) (`fsp_version_t *p_version`)

`void` [R_BSP_SoftwareDelay](#) (`uint32_t delay`, `bsp_delay_units_t units`)

`fsp_err_t` [R_BSP_GroupIrqWrite](#) (`bsp_grp_irq_t irq`,
`void(*p_callback)(bsp_grp_irq_t irq)`)

```
void NMI_Handler (void)
```

```
void R_BSP_RegisterProtectEnable (bsp_reg_protect_t regs_to_protect)
```

```
void R_BSP_RegisterProtectDisable (bsp_reg_protect_t regs_to_unprotect)
```

Detailed Description

The BSP is responsible for getting the MCU from reset to the user's application. Before reaching the user's application, the BSP sets up the stacks, heap, clocks, interrupts, C runtime environment, and stack monitor.

- [BSP Features](#)
- [BSP Clock Configuration](#)
- [System Interrupts](#)
- [Group Interrupts](#)
- [External and Peripheral Interrupts](#)
- [Error Logging](#)
- [BSP Weak Symbols](#)
- [Warm Start Callbacks](#)
- [Register Protection](#)
- [ID Codes](#)
- [Software Delay](#)
- [Board Specific Features](#)
- [Configuration](#)

Overview

BSP Features

BSP Clock Configuration

All system clocks are set up during BSP initialization based on the settings in `bsp_clock_cfg.h`. These settings are derived from clock configuration information provided from the ISDE **Clocks** tab setting.

- Clock configuration is performed prior to initializing the C runtime environment to speed up the startup process, as it is possible to start up on a relatively slow (that is, 32 kHz) clock.
- The BSP implements the required delays to allow the selected clock to stabilize.
- The BSP will configure the CMSIS `SystemCoreClock` variable after clock initialization with the current system clock frequency.

System Interrupts

As RA MCUs are based on the Cortex-M ARM architecture, the NVIC Nested Vectored Interrupt Controller (NVIC) handles exceptions and interrupt configuration, prioritization and interrupt masking. In the ARM architecture, the NVIC handles exceptions. Some exceptions are known as System Exceptions. System exceptions are statically located at the "top" of the vector table and occupy vector numbers 1 to 15. Vector zero is reserved for the MSP Main Stack Pointer (MSP). The remaining 15 system exceptions are shown below:

- Reset
- NMI

- Cortex-M4 Hard Fault Handler
- Cortex-M4 MPU Fault Handler
- Cortex-M4 Bus Fault Handler
- Cortex-M4 Usage Fault Handler
- Reserved
- Reserved
- Reserved
- Reserved
- Cortex-M4 SVCALL Handler
- Cortex-M4 Debug Monitor Handler
- Reserved
- Cortex-M4 PendSV Handler
- Cortex-M4 SysTick Handler

NMI and Hard Fault exceptions are enabled out of reset and have fixed priorities. Other exceptions have configurable priorities and some can be disabled.

Group Interrupts

Group interrupt is the term used to describe the 12 sources that can trigger the Non-Maskable Interrupt (NMI). When an NMI occurs the NMI Handler examines the NMISR (status register) to determine the source of the interrupt. NMI interrupts take precedence over all interrupts, are usable only as CPU interrupts, and cannot activate the RA peripherals Data Transfer Controller (DTC) or Direct Memory Access Controller (DMAC).

Possible group interrupt sources include:

- IWDG Underflow/Refresh Error
- WDT Underflow/Refresh Error
- Voltage-Monitoring 1 Interrupt
- Voltage-Monitoring 2 Interrupt
- VBAT monitor Interrupt
- Oscillation Stop is detected
- NMI pin
- RAM Parity Error
- RAM ECC Error
- MPU Bus Slave Error
- MPU Bus Master Error
- MPU Stack Error

A user may enable notification for one or more group interrupts by registering a callback using the BSP API function [R_BSP_GroupIrqWrite\(\)](#). When an NMI interrupt occurs, the NMI handler checks to see if there is a callback registered for the cause of the interrupt and if so calls the registered callback function.

External and Peripheral Interrupts

User configurable interrupts begin with slot 16. These may be external, or peripheral generated interrupts.

Although the number of available slots for the NVIC interrupt vector table may seem small, the BSP defines up to 512 events that are capable of generating an interrupt. By using Event Mapping, the BSP maps user-enabled events to NVIC interrupts. For an RA6M3 MCU, only 96 of these events may be active at any one time, but the user has flexibility by choosing which events generate the active event.

By allowing the user to select only the events they are interested in as interrupt sources, we are able to provide an interrupt service routine that is fast and event specific.

For example, on other microcontrollers a standard NVIC interrupt vector table might contain a single vector entry for the SCI0 (Serial Communications Interface) peripheral. The interrupt service routine for this would have to check a status register for the 'real' source of the interrupt. In the RA implementation there is a vector entry for each of the SCI0 events that we are interested in.

BSP Weak Symbols

You might wonder how the BSP is able to place ISR addresses in the NVIC table without the user having explicitly defined one. All that is required by the BSP is that the interrupt event be given a priority.

This is accomplished through the use of the 'weak' attribute. The weak attribute causes the declaration to be emitted as a weak symbol rather than a global. A weak symbol is one that can be overridden by an accompanying strong reference with the same name. When the BSP declares a function as weak, user code can define the same function and it will be used in place of the BSP function. By defining all possible interrupt sources as weak, the vector table can be built at compile time and any user declarations (strong references) will be used at runtime.

Weak symbols are supported for ELF targets and also for a.out targets when using the GNU assembler and linker.

Note that in CMSIS system.c, there is also a weak definition (and a function body) for the Warm Start callback function [R_BSP_WarmStart\(\)](#). Because this function is defined in the same file as the weak declaration, it will be called as the 'default' implementation. The function may be overridden by the user by copying the body into their user application and modifying it as necessary. The linker identifies this as the 'strong' reference and uses it.

Warm Start Callbacks

As the BSP is in the process of bringing up the board out of reset, there are three points where the user can request a callback. These are defined as the 'Pre Clock Init', 'Post Clock Init' and 'Post C' warm start callbacks.

As described above, this function is already weakly defined as [R_BSP_WarmStart\(\)](#), so it is a simple matter of redefining the function or copying the existing body from CMSIS system.c into the application code to get a callback. [R_BSP_WarmStart\(\)](#) takes an event parameter of type `bsp_warm_start_event_t` which describes the type of warm start callback being made.

This function is not enabled/disabled and is always called for both events as part of the BSP startup. Therefore it needs a function body, which will not be called if the user is overriding it. The function body is located in system. To use this function just copy this function into your own code and modify it to meet your needs.

Error Logging

When error logging is enabled, the error logging function can be redefined on the command line by defining [FSP_ERROR_LOG\(err\)](#) to the desired function call. The default function implementation is [FSP_ERROR_LOG\(err\)=fsp_error_log\(err, **FILE**, **LINE**\)](#). This implementation uses the predefined macros **FILE** and **LINE** to help identify the location where the error occurred. Removing the line from the function call can reduce code size when error logging is enabled. Some compilers may support other predefined macros like **FUNCTION**, which could be helpful for customizing the error logger.

Register Protection

The BSP register protection functions utilize reference counters to ensure that an application which has specified a certain register and subsequently calls another function doesn't have its register protection settings inadvertently modified.

Each time RegisterProtectDisable() is called, the respective reference counter is incremented.

Each time RegisterProtectEnable() is called, the respective reference counter is decremented.

Both functions will only modify the protection state if their reference counter is zero.

```
/* Enable writing to protected CGC registers */
R_BSP_RegisterProtectDisable(BSP_REG_PROTECT_CGC);
/* Insert code to modify protected CGC registers. */
/* Disable writing to protected CGC registers */
R_BSP_RegisterProtectEnable(BSP_REG_PROTECT_CGC);
```

ID Codes

The ID code is 16 byte value that can be used to protect the MCU from being connected to a debugger or from connecting in Serial Boot Mode. There are different settings that can be set for the ID code; please refer to the hardware manual for your device for available options.

Software Delay

Implements a blocking software delay. A delay can be specified in microseconds, milliseconds or seconds. The delay is implemented based on the system clock rate.

```
/* Delay at least 1 second. Depending on the number of wait states required for the
region of memory
* that the software_delay_loop has been linked in this could take longer. The
default is 4 cycles per loop.
* This can be modified by redefining DELAY_LOOP_CYCLES. BSP_DELAY_UNITS_SECONDS,
BSP_DELAY_UNITS_MILLISECONDS,
* and BSP_DELAY_UNITS_MICROSECONDS can all be used with R_BSP_SoftwareDelay. */
R_BSP_SoftwareDelay(1, BSP_DELAY_UNITS_SECONDS);
```

Critical Section Macros

Implements a critical section. Some MCUs (MCUs with the BASEPRI register) support allowing high priority interrupts to execute during critical sections. On these MCUs, interrupts with priority less than or equal to BSP_CFG_IRQ_MASK_LEVEL_FOR_CRITICAL_SECTION are not serviced in critical sections. Interrupts with higher priority than BSP_CFG_IRQ_MASK_LEVEL_FOR_CRITICAL_SECTION still

execute in critical sections.

```
FSP_CRITICAL_SECTION_DEFINE;

/* Store the current interrupt posture. */

FSP_CRITICAL_SECTION_ENTER;

/* Interrupts cannot run in this section unless their priority is less than
BSP_CFG_IRQ_MASK_LEVEL_FOR_CRITICAL_SECTION. */

/* Restore saved interrupt posture. */

FSP_CRITICAL_SECTION_EXIT;
```

Board Specific Features

The BSP will call the board's initialization function (bsp_init) which can initialize board specific features. Possible board features are listed below.

Board Feature	Description
SDRAM Support	The BSP will initialize SDRAM if the board supports it
QSPI Support	The BSP will initialize QSPI if the board supports it and put it into ROM mode. Use the R_QSPI module to write and erase the QSPI chip.

Configuration

The BSP is heavily data driven with most features and functionality being configured based on the content from configuration files. Configuration files represent the settings specified by the user and are generated by the ISDE when the Generate Project Content button is clicked.

Build Time Configurations for fsp_common

The following build time configurations are defined in fsp_cfg/bsp/bsp_cfg.h:

Configuration	Options	Description
Main stack size (bytes)	Value must be an integer multiple of 8 and between 8 and 0xFFFFFFFF	
Heap size (bytes) - A minimum of 4K (0x1000) is required if standard library functions are to be used.	Value must be 0 or an integer multiple of 8 between 8 and 0xFFFFFFFF. A minimum of 4K (0x1000) is required if standard library functions are to be used.	
MCU Vcc (mV)	Value must between 0 and 4600 (4.6V)	

Parameter checking	• Enabled • Disabled	
Assert Failures	• Return FSP_ERR_ASSERTION • Call fsp_error_log then Return FSP_ERR_ASSERTION • Use assert() to Halt Execution • Disable checks that would return FSP_ERR_ASSERTION	
Error Log	• No Error Log • Errors Logged in fsp_error_log	
ID Code Mode	• Unlocked (Ignore ID) • Locked with All Erase support • Locked	
ID Code (32 Hex Characters)	Value must be a 32 character long hex string	
Soft Reset	• Disabled • Enabled	Support for soft reset. If disabled, registers are assumed to be set to their default value during startup.
PFS Protect	• Disabled • Enabled	Keep the PFS registers locked when they are not being modified. If disabled they will be unlocked during startup.
Main Oscillator Wait Time	• 0.25 us • 128 us • 256 us • 512 us • 1024 us • 2048 us • 4096 us • 8192 us • 16384 us • 32768 us	Number of cycles to wait for the main oscillator clock to stabilize. This setting can be overridden in board_cfg.h
Main Oscillator Clock Source	• External Oscillator • Crystal or Resonator	Select the main oscillator clock source. This setting can be overridden in board_cfg.h
Subclock Populated	• Populated • Not Populated	Select whether or not there is a subclock on the board. This setting can be overridden in board_cfg.h.
Subclock Drive	• Middle (4.4pf) • Standard (12.5pf)	Select the subclock oscillator drive capacitance. This setting can be overridden in

Subclock Stabilization Time (ms)	Value must be between 0 and 10000	board_cfg.h
		Select the subclock oscillator stabilization time. This is only used in the startup code if the subclock is selected as the system clock on the Clocks tab. This setting can be overridden in board_cfg.h

4.1.2.1 RA2A1

[BSP](#) » [MCU Board Support Package](#)

Detailed Description

Build Time Configurations for ra2a1_fsp

The following build time configurations are defined in fsp_cfg/bsp/bsp_mcu_family_cfg.h:

Configuration	Options	Description
OFS0 register settings Independent WDT Start Mode	- IWDT is Disabled - IWDT is automatically activated after a reset (Autostart mode)	
OFS0 register settings Independent WDT Timeout Period	128 cycles 512 cycles 1024 cycles 2048 cycles	
OFS0 register settings Independent WDT Dedicated Clock Frequency Divisor	1 16 32 64 128 256	
OFS0 register settings Independent WDT Window End Position	75% 50% 25% 0% (no window end position)	
OFS0 register settings Independent WDT Window Start Position	25% 50% 75% 100% (no window start position)	
OFS0 register settings Independent	NMI request or interrupt request is enabled	

WDT Reset Interrupt Request Select	Reset is enabled	
OFS0 register settings Independent WDT Stop Control	- Counting continues - Stop counting when in Sleep, Snooze mode, or Software Standby	
OFS0 register settings WDT Start Mode Select	- Automatically activate WDT after a reset (auto-start mode) - Stop WDT after a reset (register-start mode)	
OFS0 register settings WDT Timeout Period	1024 cycles 4096 cycles 8192 cycles 16384 cycles	
OFS0 register settings WDT Clock Frequency Division Ratio	4 64 128 512 2048 8192	
OFS0 register settings WDT Window End Position	75% 50% 25% 0% (no window end position)	
OFS0 register settings WDT Window Start Position	25% 50% 75% 100% (no window start position)	
OFS0 register settings WDT Reset Interrupt Request	- NMI - Reset	
OFS0 register settings WDT Stop Control	- Counting continues - Stop counting when entering Sleep mode	
OFS1 register settings Voltage Detection 0 Circuit Start	- Voltage monitor 0 reset is enabled after reset - Voltage monitor 0 reset is disabled after reset	
OFS1 register settings Voltage Detection 0 Level	3.84 V 2.82 V 2.51 V 1.90 V 1.70 V	
OFS1 register settings HOCO Oscillation Enable	HOCO oscillation is enabled after reset	HOCO must be enabled out of reset because the MCU starts

up in low voltage mode and the HOCO must be operating in low voltage mode.

Use Low Voltage Mode	• Enable • Disable	Use the low voltage mode. This limits the ICLK operating frequency to 4 MHz and requires all clock dividers to be at least 4 when oscillation stop detection is used.
MPU Enable or disable PC Region 0	• Enabled • Disabled	
MPU PC0 Start	Value must be an integer between 0 and 0x000FFFFC (ROM) or between 0x1FF00000 and 0x200FFFFC (RAM)	
MPU PC0 End	Value must be an integer between 0x00000003 and 0x000FFFFF (ROM) or between 0x1FF00003 and 0x200FFFFF (RAM)	
MPU Enable or disable PC Region 1	• Enabled • Disabled	
MPU PC1 Start	Value must be an integer between 0 and 0x000FFFFC (ROM) or between 0x1FF00000 and 0x200FFFFC (RAM)	
MPU PC1 End	Value must be an integer between 0x00000003 and 0x000FFFFF (ROM) or between 0x1FF00003 and 0x200FFFFF (RAM)	
MPU Enable or disable Memory Region 0	• Enabled • Disabled	
MPU Memory Region 0 Start	Value must be an integer between 0 and 0x000FFFFC	
MPU Memory Region 0 End	Value must be an integer between 0x00000003 and 0x000FFFFF	
MPU Enable or disable Memory Region 1	• Enabled • Disabled	
MPU Memory Region 1 Start	Value must be an integer between 0x1FF00000 and 0x200FFFFC	
MPU Memory Region 1 End	Value must be an integer between 0x1FF00003 and 0x200FFFFF	

MPU Enable or disable Memory Region 2	• Enabled • Disabled
MPU Memory Region 2 Start	Value must be an integer between 0x400C0000 and 0x400DFFFC or between 0x40100000 and 0x407FFFFC
MPU Memory Region 2 End	Value must be an integer between 0x400C0003 and 0x400DFFFF or between 0x40100003 and 0x407FFFFF
MPU Enable or disable Memory Region 3	• Enabled • Disabled
MPU Memory Region 3 Start	Value must be an integer between 0x400C0000 and 0x400DFFFC or between 0x40100000 and 0x407FFFFC
MPU Memory Region 3 End	Value must be an integer between 0x400C0003 and 0x400DFFFF or between 0x40100003 and 0x407FFFFF

4.1.2.2 RA4M1

[BSP](#) » [MCU Board Support Package](#)

Detailed Description

Build Time Configurations for ra4m1_fsp

The following build time configurations are defined in fsp_cfg/bsp/bsp_mcu_family_cfg.h:

Configuration	Options	Description
OFS0 register settings Independent WDT Start Mode	• IWDT is Disabled • IWDT is automatically activated after a reset (Autostart mode)	
OFS0 register settings Independent WDT Timeout Period	• 128 cycles • 512 cycles • 1024 cycles • 2048 cycles	
OFS0 register settings Independent WDT Dedicated Clock Frequency Divisor	• 1 • 16 • 32 • 64 • 128 • 256	

OFS0 register settings Independent WDT Window End Position	• 75% • 50% • 25% • 0% (no window end position)
OFS0 register settings Independent WDT Window Start Position	• 25% • 50% • 75% • 100% (no window start position)
OFS0 register settings Independent WDT Reset Interrupt Request Select	• NMI request or interrupt request is enabled • Reset is enabled
OFS0 register settings Independent WDT Stop Control	• Counting continues • Stop counting when in Sleep, Snooze mode, or Software Standby
OFS0 register settings WDT Start Mode Select	• Automatically activate WDT after a reset (auto-start mode) • Stop WDT after a reset (register-start mode)
OFS0 register settings WDT Timeout Period	• 1024 cycles • 4096 cycles • 8192 cycles • 16384 cycles
OFS0 register settings WDT Clock Frequency Division Ratio	• 4 • 64 • 128 • 512 • 2048 • 8192
OFS0 register settings WDT Window End Position	• 75% • 50% • 25% • 0% (no window end position)
OFS0 register settings WDT Window Start Position	• 25% • 50% • 75% • 100% (no window start position)
OFS0 register settings WDT Reset Interrupt Request	• NMI • Reset
OFS0 register settings WDT Stop Control	• Counting continues • Stop counting when

	entering Sleep mode	
OFS1 register settings Voltage Detection 0 Circuit Start	• Voltage monitor 0 reset is enabled after reset • Voltage monitor 0 reset is disabled after reset	
OFS1 register settings Voltage Detection 0 Level	• 3.84 V • 2.82 V • 2.51 V • 1.90 V • 1.70 V	
OFS1 register settings HOCO Oscillation Enable	HOCO oscillation is enabled after reset	HOCO must be enabled out of reset because the MCU starts up in low voltage mode and the HOCO must be operating in low voltage mode.
Use Low Voltage Mode	• Enable • Disable	Use the low voltage mode. This limits the ICLK operating frequency to 4 MHz and requires all clock dividers to be at least 4.
MPU Enable or disable PC Region 0	• Enabled • Disabled	
MPU PC0 Start	Value must be an integer between 0 and 0x00FFFFFC (ROM) or between 0x1FF00000 and 0x200FFFFC (RAM)	
MPU PC0 End	Value must be an integer between 0x00000003 and 0x00FFFFFF (ROM) or between 0x1FF00003 and 0x200FFFFF (RAM)	
MPU Enable or disable PC Region 1	• Enabled • Disabled	
MPU PC1 Start	Value must be an integer between 0 and 0x00FFFFFC (ROM) or between 0x1FF00000 and 0x200FFFFC (RAM)	
MPU PC1 End	Value must be an integer between 0x00000003 and 0x00FFFFFF (ROM) or between 0x1FF00003 and 0x200FFFFF (RAM)	
MPU Enable or disable Memory Region 0	• Enabled • Disabled	
MPU Memory Region 0 Start	Value must be an integer between 0 and 0x00FFFFFC	
MPU Memory Region 0 End	Value must be an integer	

	between 0x00000003 and 0x00FFFFFF
MPU Enable or disable Memory Region 1	• Enabled • Disabled
MPU Memory Region 1 Start	Value must be an integer between 0x1FF00000 and 0x200FFFC
MPU Memory Region 1 End	Value must be an integer between 0x1FF00003 and 0x200FFFFF
MPU Enable or disable Memory Region 2	• Enabled • Disabled
MPU Memory Region 2 Start	Value must be an integer between 0x400C0000 and 0x400DFFFC or between 0x40100000 and 0x407FFFC
MPU Memory Region 2 End	Value must be an integer between 0x400C0003 and 0x400DFFFF or between 0x40100003 and 0x407FFFFF
MPU Enable or disable Memory Region 3	• Enabled • Disabled
MPU Memory Region 3 Start	Value must be an integer between 0x400C0000 and 0x400DFFFC or between 0x40100000 and 0x407FFFC
MPU Memory Region 3 End	Value must be an integer between 0x400C0003 and 0x400DFFFF or between 0x40100003 and 0x407FFFFF

4.1.2.3 RA6M1

[BSP » MCU Board Support Package](#)

Detailed Description

Build Time Configurations for ra6m1_fsp

The following build time configurations are defined in fsp_cfg/bsp/bsp_mcu_family_cfg.h:

Configuration	Options	Description
OFS0 register settings Independent WDT Start Mode	• IWDT is Disabled • IWDT is automatically activated after a reset	

(Autostart mode)

OFS0 register settings Independent WDT Timeout Period	• 128 cycles • 512 cycles • 1024 cycles • 2048 cycles
OFS0 register settings Independent WDT Dedicated Clock Frequency Divisor	• 1 • 16 • 32 • 64 • 128 • 256
OFS0 register settings Independent WDT Window End Position	• 75% • 50% • 25% • 0% (no window end position)
OFS0 register settings Independent WDT Window Start Position	• 25% • 50% • 75% • 100% (no window start position)
OFS0 register settings Independent WDT Reset Interrupt Request Select	• NMI request or interrupt request is enabled • Reset is enabled
OFS0 register settings Independent WDT Stop Control	• Counting continues (Note: Device will not enter Deep Standby Mode when selected. Device will enter Software Standby Mode) • Stop counting when in Sleep, Snooze mode, or Software Standby
OFS0 register settings WDT Start Mode Select	• Automatically activate WDT after a reset (auto-start mode) • Stop WDT after a reset (register-start mode)
OFS0 register settings WDT Timeout Period	• 1024 cycles • 4096 cycles • 8192 cycles • 16384 cycles
OFS0 register settings WDT Clock Frequency Division Ratio	• 4 • 64 • 128 • 512 • 2048 • 8192

OFS0 register settings WDT Window End Position	• 75% • 50% • 25% • 0% (no window end position)
OFS0 register settings WDT Window Start Position	• 25% • 50% • 75% • 100% (no window start position)
OFS0 register settings WDT Reset Interrupt Request	• NMI • Reset
OFS0 register settings WDT Stop Control	• Counting continues • Stop counting when entering Sleep mode
OFS1 register settings Voltage Detection 0 Circuit Start	• Voltage monitor 0 reset is enabled after reset • Voltage monitor 0 reset is disabled after reset
OFS1 register settings Voltage Detection 0 Level	• 2.94 V • 2.87 V • 2.80 V
OFS1 register settings HOCO OScillation Enable	• HOCO oscillation is enabled after reset • HOCO oscillation is disabled after reset
MPU Enable or disable PC Region 0	• Enabled • Disabled
MPU PC0 Start	Value must be an integer between 0 and 0xFFFFFFFFC
MPU PC0 End	Value must be an integer between 0x00000003 and 0xFFFFFFFF
MPU Enable or disable PC Region 1	• Enabled • Disabled
MPU PC1 Start	Value must be an integer between 0 and 0xFFFFFFFFC
MPU PC1 End	Value must be an integer between 0x00000003 and 0xFFFFFFFF
MPU Enable or disable Memory Region 0	• Enabled • Disabled
MPU Memory Region 0 Start	Value must be an integer between 0 and 0x00FFFFFFC

MPU Memory Region 0 End	Value must be an integer between 0x00000003 and 0x00FFFFFF
MPU Enable or disable Memory Region 1	• Enabled • Disabled
MPU Memory Region 1 Start	Value must be an integer between 0x1FF00000 and 0x200FFFC
MPU Memory Region 1 End	Value must be an integer between 0x1FF00003 and 0x200FFFF
MPU Enable or disable Memory Region 2	• Enabled • Disabled
MPU Memory Region 2 Start	Value must be an integer between 0x400C0000 and 0x400DFFFC or between 0x40100000 and 0x407FFFC
MPU Memory Region 2 End	Value must be an integer between 0x400C0003 and 0x400DFFFF or between 0x40100003 and 0x407FFFF
MPU Enable or disable Memory Region 3	• Enabled • Disabled
MPU Memory Region 3 Start	Value must be an integer between 0x400C0000 and 0x400DFFFC or between 0x40100000 and 0x407FFFC
MPU Memory Region 3 End	Value must be an integer between 0x400C0003 and 0x400DFFFF or between 0x40100003 and 0x407FFFF

4.1.2.4 RA6M2

[BSP » MCU Board Support Package](#)

Detailed Description

Build Time Configurations for ra6m2_fsp

The following build time configurations are defined in fsp_cfg/bsp/bsp_mcu_family_cfg.h:

Configuration	Options	Description
OFS0 register settings Independent WDT Start	• IWDT is Disabled • IWDT is automatically	

Mode	activated after a reset (Autostart mode)
OFS0 register settings Independent WDT Timeout Period	• 128 cycles • 512 cycles • 1024 cycles • 2048 cycles
OFS0 register settings Independent WDT Dedicated Clock Frequency Divisor	• 1 • 16 • 32 • 64 • 128 • 256
OFS0 register settings Independent WDT Window End Position	• 75% • 50% • 25% • 0% (no window end position)
OFS0 register settings Independent WDT Window Start Position	• 25% • 50% • 75% • 100% (no window start position)
OFS0 register settings Independent WDT Reset Interrupt Request Select	• NMI request or interrupt request is enabled • Reset is enabled
OFS0 register settings Independent WDT Stop Control	• Counting continues (Note: Device will not enter Deep Standby Mode when selected. Device will enter Software Standby Mode) • Stop counting when in Sleep, Snooze mode, or Software Standby
OFS0 register settings WDT Start Mode Select	• Automatically activate WDT after a reset (auto-start mode) • Stop WDT after a reset (register-start mode)
OFS0 register settings WDT Timeout Period	• 1024 cycles • 4096 cycles • 8192 cycles • 16384 cycles
OFS0 register settings WDT Clock Frequency Division Ratio	• 4 • 64 • 128 • 512 • 2048

	• 8192
OFS0 register settings WDT Window End Position	• 75% • 50% • 25% • 0% (no window end position)
OFS0 register settings WDT Window Start Position	• 25% • 50% • 75% • 100% (no window start position)
OFS0 register settings WDT Reset Interrupt Request	• NMI • Reset
OFS0 register settings WDT Stop Control	• Counting continues • Stop counting when entering Sleep mode
OFS1 register settings Voltage Detection 0 Circuit Start	• Voltage monitor 0 reset is enabled after reset • Voltage monitor 0 reset is disabled after reset
OFS1 register settings Voltage Detection 0 Level	• 2.94 V • 2.87 V • 2.80 V
OFS1 register settings HOCO OScillation Enable	• HOCO oscillation is enabled after reset • HOCO oscillation is disabled after reset
MPU Enable or disable PC Region 0	• Enabled • Disabled
MPU PC0 Start	Value must be an integer between 0 and 0xFFFFFFFFC
MPU PC0 End	Value must be an integer between 0x00000003 and 0xFFFFFFFF
MPU Enable or disable PC Region 1	• Enabled • Disabled
MPU PC1 Start	Value must be an integer between 0 and 0xFFFFFFFFC
MPU PC1 End	Value must be an integer between 0x00000003 and 0xFFFFFFFF
MPU Enable or disable Memory Region 0	• Enabled • Disabled
MPU Memory Region 0 Start	Value must be an integer

	between 0 and 0x00FFFFFFC
MPU Memory Region 0 End	Value must be an integer between 0x00000003 and 0x00FFFFFFF
MPU Enable or disable Memory Region 1	• Enabled • Disabled
MPU Memory Region 1 Start	Value must be an integer between 0x1FF00000 and 0x200FFFFFFC
MPU Memory Region 1 End	Value must be an integer between 0x1FF00003 and 0x200FFFFFFF
MPU Enable or disable Memory Region 2	• Enabled • Disabled
MPU Memory Region 2 Start	Value must be an integer between 0x400C0000 and 0x400DFFFC or between 0x40100000 and 0x407FFFFFFC
MPU Memory Region 2 End	Value must be an integer between 0x400C0003 and 0x400DFFFF or between 0x40100003 and 0x407FFFFFFF
MPU Enable or disable Memory Region 3	• Enabled • Disabled
MPU Memory Region 3 Start	Value must be an integer between 0x400C0000 and 0x400DFFFC or between 0x40100000 and 0x407FFFFFFC
MPU Memory Region 3 End	Value must be an integer between 0x400C0003 and 0x400DFFFF or between 0x40100003 and 0x407FFFFFFF

4.1.2.5 RA6M3

[BSP](#) » [MCU Board Support Package](#)

Detailed Description

Build Time Configurations for ra6m3_fsp

The following build time configurations are defined in fsp_cfg/bsp/bsp_mcu_family_cfg.h:

Configuration	Options	Description

OFS0 register settings Independent WDT Start Mode	• IWDT is Disabled • IWDT is automatically activated after a reset (Autostart mode)
OFS0 register settings Independent WDT Timeout Period	• 128 cycles • 512 cycles • 1024 cycles • 2048 cycles
OFS0 register settings Independent WDT Dedicated Clock Frequency Divisor	• 1 • 16 • 32 • 64 • 128 • 256
OFS0 register settings Independent WDT Window End Position	• 75% • 50% • 25% • 0% (no window end position)
OFS0 register settings Independent WDT Window Start Position	• 25% • 50% • 75% • 100% (no window start position)
OFS0 register settings Independent WDT Reset Interrupt Request Select	• NMI request or interrupt request is enabled • Reset is enabled
OFS0 register settings Independent WDT Stop Control	• Counting continues (Note: Device will not enter Deep Standby Mode when selected. Device will enter Software Standby Mode) • Stop counting when in Sleep, Snooze mode, or Software Standby
OFS0 register settings WDT Start Mode Select	• Automatically activate WDT after a reset (auto-start mode) • Stop WDT after a reset (register-start mode)
OFS0 register settings WDT Timeout Period	• 1024 cycles • 4096 cycles • 8192 cycles • 16384 cycles
OFS0 register settings WDT Clock Frequency Division Ratio	• 4 • 64 • 128

	• 512 • 2048 • 8192
OFS0 register settings WDT Window End Position	• 75% • 50% • 25% • 0% (no window end position)
OFS0 register settings WDT Window Start Position	• 25% • 50% • 75% • 100% (no window start position)
OFS0 register settings WDT Reset Interrupt Request	• NMI • Reset
OFS0 register settings WDT Stop Control	• Counting continues • Stop counting when entering Sleep mode
OFS1 register settings Voltage Detection 0 Circuit Start	• Voltage monitor 0 reset is enabled after reset • Voltage monitor 0 reset is disabled after reset
OFS1 register settings Voltage Detection 0 Level	• 2.94 V • 2.87 V • 2.80 V
OFS1 register settings HOCO OScillation Enable	• HOCO oscillation is enabled after reset • HOCO oscillation is disabled after reset
MPU Enable or disable PC Region 0	• Enabled • Disabled
MPU PC0 Start	Value must be an integer between 0 and 0xFFFFFFF0
MPU PC0 End	Value must be an integer between 0x00000003 and 0xFFFFFFFF
MPU Enable or disable PC Region 1	• Enabled • Disabled
MPU PC1 Start	Value must be an integer between 0 and 0xFFFFFFF0
MPU PC1 End	Value must be an integer between 0x00000003 and 0xFFFFFFFF
MPU Enable or disable Memory Region 0	• Enabled • Disabled

MPU Memory Region 0 Start	Value must be an integer between 0 and 0x00FFFFFFC
MPU Memory Region 0 End	Value must be an integer between 0x00000003 and 0x00FFFFFFF
MPU Enable or disable Memory Region 1	• Enabled • Disabled
MPU Memory Region 1 Start	Value must be an integer between 0x1FF00000 and 0x200FFFFC
MPU Memory Region 1 End	Value must be an integer between 0x1FF00003 and 0x200FFFFF
MPU Enable or disable Memory Region 2	• Enabled • Disabled
MPU Memory Region 2 Start	Value must be an integer between 0x400C0000 and 0x400DFFFC or between 0x40100000 and 0x407FFFFC
MPU Memory Region 2 End	Value must be an integer between 0x400C0003 and 0x400DFFFF or between 0x40100003 and 0x407FFFFF
MPU Enable or disable Memory Region 3	• Enabled • Disabled
MPU Memory Region 3 Start	Value must be an integer between 0x400C0000 and 0x400DFFFC or between 0x40100000 and 0x407FFFFC
MPU Memory Region 3 End	Value must be an integer between 0x400C0003 and 0x400DFFFF or between 0x40100003 and 0x407FFFFF

4.1.3 BSP I/O access

BSP

Functions

```
__STATIC_INLINE uint32_t R_BSP_PinRead (bsp_io_port_pin_t pin)
```

```
__STATIC_INLINE void R_BSP_PinWrite (bsp_io_port_pin_t pin, bsp_io_level_t level)
```

```
__STATIC_INLINE void R_BSP_PinAccessEnable (void)
```

```
__STATIC_INLINE void R_BSP_PinAccessDisable(void)
```

Detailed Description

This module provides basic read/write access to port pins.

Enumerations

```
enum bsp_io_level_t
```

```
enum bsp_io_direction_t
```

```
enum bsp_io_port_t
```

```
enum bsp_io_port_pin_t
```

Enumeration Type Documentation

◆ bsp_io_level_t

enum bsp_io_level_t	
Levels that can be set and read for individual pins	
Enumerator	
BSP_IO_LEVEL_LOW	Low.
BSP_IO_LEVEL_HIGH	High.

◆ bsp_io_direction_t

enum bsp_io_direction_t	
Direction of individual pins	
Enumerator	
BSP_IO_DIRECTION_INPUT	Input.
BSP_IO_DIRECTION_OUTPUT	Output.

◆ **bsp_io_port_t**

enum bsp_io_port_t	
Superset list of all possible IO ports.	
Enumerator	
BSP_IO_PORT_00	IO port 0.
BSP_IO_PORT_01	IO port 1.
BSP_IO_PORT_02	IO port 2.
BSP_IO_PORT_03	IO port 3.
BSP_IO_PORT_04	IO port 4.
BSP_IO_PORT_05	IO port 5.
BSP_IO_PORT_06	IO port 6.
BSP_IO_PORT_07	IO port 7.
BSP_IO_PORT_08	IO port 8.
BSP_IO_PORT_09	IO port 9.
BSP_IO_PORT_10	IO port 10.
BSP_IO_PORT_11	IO port 11.

◆ **bsp_io_port_pin_t**

enum bsp_io_port_pin_t	
Superset list of all possible IO port pins.	
Enumerator	
BSP_IO_PORT_00_PIN_00	IO port 0 pin 0.
BSP_IO_PORT_00_PIN_01	IO port 0 pin 1.
BSP_IO_PORT_00_PIN_02	IO port 0 pin 2.
BSP_IO_PORT_00_PIN_03	IO port 0 pin 3.
BSP_IO_PORT_00_PIN_04	IO port 0 pin 4.

BSP_IO_PORT_00_PIN_05	IO port 0 pin 5.
BSP_IO_PORT_00_PIN_06	IO port 0 pin 6.
BSP_IO_PORT_00_PIN_07	IO port 0 pin 7.
BSP_IO_PORT_00_PIN_08	IO port 0 pin 8.
BSP_IO_PORT_00_PIN_09	IO port 0 pin 9.
BSP_IO_PORT_00_PIN_10	IO port 0 pin 10.
BSP_IO_PORT_00_PIN_11	IO port 0 pin 11.
BSP_IO_PORT_00_PIN_12	IO port 0 pin 12.
BSP_IO_PORT_00_PIN_13	IO port 0 pin 13.
BSP_IO_PORT_00_PIN_14	IO port 0 pin 14.
BSP_IO_PORT_00_PIN_15	IO port 0 pin 15.
BSP_IO_PORT_01_PIN_00	IO port 1 pin 0.
BSP_IO_PORT_01_PIN_01	IO port 1 pin 1.
BSP_IO_PORT_01_PIN_02	IO port 1 pin 2.
BSP_IO_PORT_01_PIN_03	IO port 1 pin 3.
BSP_IO_PORT_01_PIN_04	IO port 1 pin 4.
BSP_IO_PORT_01_PIN_05	IO port 1 pin 5.
BSP_IO_PORT_01_PIN_06	IO port 1 pin 6.
BSP_IO_PORT_01_PIN_07	IO port 1 pin 7.
BSP_IO_PORT_01_PIN_08	IO port 1 pin 8.
BSP_IO_PORT_01_PIN_09	IO port 1 pin 9.
BSP_IO_PORT_01_PIN_10	IO port 1 pin 10.
BSP_IO_PORT_01_PIN_11	IO port 1 pin 11.
BSP_IO_PORT_01_PIN_12	IO port 1 pin 12.

BSP_IO_PORT_01_PIN_13	IO port 1 pin 13.
BSP_IO_PORT_01_PIN_14	IO port 1 pin 14.
BSP_IO_PORT_01_PIN_15	IO port 1 pin 15.
BSP_IO_PORT_02_PIN_00	IO port 2 pin 0.
BSP_IO_PORT_02_PIN_01	IO port 2 pin 1.
BSP_IO_PORT_02_PIN_02	IO port 2 pin 2.
BSP_IO_PORT_02_PIN_03	IO port 2 pin 3.
BSP_IO_PORT_02_PIN_04	IO port 2 pin 4.
BSP_IO_PORT_02_PIN_05	IO port 2 pin 5.
BSP_IO_PORT_02_PIN_06	IO port 2 pin 6.
BSP_IO_PORT_02_PIN_07	IO port 2 pin 7.
BSP_IO_PORT_02_PIN_08	IO port 2 pin 8.
BSP_IO_PORT_02_PIN_09	IO port 2 pin 9.
BSP_IO_PORT_02_PIN_10	IO port 2 pin 10.
BSP_IO_PORT_02_PIN_11	IO port 2 pin 11.
BSP_IO_PORT_02_PIN_12	IO port 2 pin 12.
BSP_IO_PORT_02_PIN_13	IO port 2 pin 13.
BSP_IO_PORT_02_PIN_14	IO port 2 pin 14.
BSP_IO_PORT_02_PIN_15	IO port 2 pin 15.
BSP_IO_PORT_03_PIN_00	IO port 3 pin 0.
BSP_IO_PORT_03_PIN_01	IO port 3 pin 1.
BSP_IO_PORT_03_PIN_02	IO port 3 pin 2.
BSP_IO_PORT_03_PIN_03	IO port 3 pin 3.
BSP_IO_PORT_03_PIN_04	IO port 3 pin 4.

BSP_IO_PORT_03_PIN_05	IO port 3 pin 5.
BSP_IO_PORT_03_PIN_06	IO port 3 pin 6.
BSP_IO_PORT_03_PIN_07	IO port 3 pin 7.
BSP_IO_PORT_03_PIN_08	IO port 3 pin 8.
BSP_IO_PORT_03_PIN_09	IO port 3 pin 9.
BSP_IO_PORT_03_PIN_10	IO port 3 pin 10.
BSP_IO_PORT_03_PIN_11	IO port 3 pin 11.
BSP_IO_PORT_03_PIN_12	IO port 3 pin 12.
BSP_IO_PORT_03_PIN_13	IO port 3 pin 13.
BSP_IO_PORT_03_PIN_14	IO port 3 pin 14.
BSP_IO_PORT_03_PIN_15	IO port 3 pin 15.
BSP_IO_PORT_04_PIN_00	IO port 4 pin 0.
BSP_IO_PORT_04_PIN_01	IO port 4 pin 1.
BSP_IO_PORT_04_PIN_02	IO port 4 pin 2.
BSP_IO_PORT_04_PIN_03	IO port 4 pin 3.
BSP_IO_PORT_04_PIN_04	IO port 4 pin 4.
BSP_IO_PORT_04_PIN_05	IO port 4 pin 5.
BSP_IO_PORT_04_PIN_06	IO port 4 pin 6.
BSP_IO_PORT_04_PIN_07	IO port 4 pin 7.
BSP_IO_PORT_04_PIN_08	IO port 4 pin 8.
BSP_IO_PORT_04_PIN_09	IO port 4 pin 9.
BSP_IO_PORT_04_PIN_10	IO port 4 pin 10.
BSP_IO_PORT_04_PIN_11	IO port 4 pin 11.
BSP_IO_PORT_04_PIN_12	IO port 4 pin 12.

BSP_IO_PORT_04_PIN_13	IO port 4 pin 13.
BSP_IO_PORT_04_PIN_14	IO port 4 pin 14.
BSP_IO_PORT_04_PIN_15	IO port 4 pin 15.
BSP_IO_PORT_05_PIN_00	IO port 5 pin 0.
BSP_IO_PORT_05_PIN_01	IO port 5 pin 1.
BSP_IO_PORT_05_PIN_02	IO port 5 pin 2.
BSP_IO_PORT_05_PIN_03	IO port 5 pin 3.
BSP_IO_PORT_05_PIN_04	IO port 5 pin 4.
BSP_IO_PORT_05_PIN_05	IO port 5 pin 5.
BSP_IO_PORT_05_PIN_06	IO port 5 pin 6.
BSP_IO_PORT_05_PIN_07	IO port 5 pin 7.
BSP_IO_PORT_05_PIN_08	IO port 5 pin 8.
BSP_IO_PORT_05_PIN_09	IO port 5 pin 9.
BSP_IO_PORT_05_PIN_10	IO port 5 pin 10.
BSP_IO_PORT_05_PIN_11	IO port 5 pin 11.
BSP_IO_PORT_05_PIN_12	IO port 5 pin 12.
BSP_IO_PORT_05_PIN_13	IO port 5 pin 13.
BSP_IO_PORT_05_PIN_14	IO port 5 pin 14.
BSP_IO_PORT_05_PIN_15	IO port 5 pin 15.
BSP_IO_PORT_06_PIN_00	IO port 6 pin 0.
BSP_IO_PORT_06_PIN_01	IO port 6 pin 1.
BSP_IO_PORT_06_PIN_02	IO port 6 pin 2.
BSP_IO_PORT_06_PIN_03	IO port 6 pin 3.
BSP_IO_PORT_06_PIN_04	IO port 6 pin 4.

BSP_IO_PORT_06_PIN_05	IO port 6 pin 5.
BSP_IO_PORT_06_PIN_06	IO port 6 pin 6.
BSP_IO_PORT_06_PIN_07	IO port 6 pin 7.
BSP_IO_PORT_06_PIN_08	IO port 6 pin 8.
BSP_IO_PORT_06_PIN_09	IO port 6 pin 9.
BSP_IO_PORT_06_PIN_10	IO port 6 pin 10.
BSP_IO_PORT_06_PIN_11	IO port 6 pin 11.
BSP_IO_PORT_06_PIN_12	IO port 6 pin 12.
BSP_IO_PORT_06_PIN_13	IO port 6 pin 13.
BSP_IO_PORT_06_PIN_14	IO port 6 pin 14.
BSP_IO_PORT_06_PIN_15	IO port 6 pin 15.
BSP_IO_PORT_07_PIN_00	IO port 7 pin 0.
BSP_IO_PORT_07_PIN_01	IO port 7 pin 1.
BSP_IO_PORT_07_PIN_02	IO port 7 pin 2.
BSP_IO_PORT_07_PIN_03	IO port 7 pin 3.
BSP_IO_PORT_07_PIN_04	IO port 7 pin 4.
BSP_IO_PORT_07_PIN_05	IO port 7 pin 5.
BSP_IO_PORT_07_PIN_06	IO port 7 pin 6.
BSP_IO_PORT_07_PIN_07	IO port 7 pin 7.
BSP_IO_PORT_07_PIN_08	IO port 7 pin 8.
BSP_IO_PORT_07_PIN_09	IO port 7 pin 9.
BSP_IO_PORT_07_PIN_10	IO port 7 pin 10.
BSP_IO_PORT_07_PIN_11	IO port 7 pin 11.
BSP_IO_PORT_07_PIN_12	IO port 7 pin 12.

BSP_IO_PORT_07_PIN_13	IO port 7 pin 13.
BSP_IO_PORT_07_PIN_14	IO port 7 pin 14.
BSP_IO_PORT_07_PIN_15	IO port 7 pin 15.
BSP_IO_PORT_08_PIN_00	IO port 8 pin 0.
BSP_IO_PORT_08_PIN_01	IO port 8 pin 1.
BSP_IO_PORT_08_PIN_02	IO port 8 pin 2.
BSP_IO_PORT_08_PIN_03	IO port 8 pin 3.
BSP_IO_PORT_08_PIN_04	IO port 8 pin 4.
BSP_IO_PORT_08_PIN_05	IO port 8 pin 5.
BSP_IO_PORT_08_PIN_06	IO port 8 pin 6.
BSP_IO_PORT_08_PIN_07	IO port 8 pin 7.
BSP_IO_PORT_08_PIN_08	IO port 8 pin 8.
BSP_IO_PORT_08_PIN_09	IO port 8 pin 9.
BSP_IO_PORT_08_PIN_10	IO port 8 pin 10.
BSP_IO_PORT_08_PIN_11	IO port 8 pin 11.
BSP_IO_PORT_08_PIN_12	IO port 8 pin 12.
BSP_IO_PORT_08_PIN_13	IO port 8 pin 13.
BSP_IO_PORT_08_PIN_14	IO port 8 pin 14.
BSP_IO_PORT_08_PIN_15	IO port 8 pin 15.
BSP_IO_PORT_09_PIN_00	IO port 9 pin 0.
BSP_IO_PORT_09_PIN_01	IO port 9 pin 1.
BSP_IO_PORT_09_PIN_02	IO port 9 pin 2.
BSP_IO_PORT_09_PIN_03	IO port 9 pin 3.
BSP_IO_PORT_09_PIN_04	IO port 9 pin 4.

BSP_IO_PORT_09_PIN_05	IO port 9 pin 5.
BSP_IO_PORT_09_PIN_06	IO port 9 pin 6.
BSP_IO_PORT_09_PIN_07	IO port 9 pin 7.
BSP_IO_PORT_09_PIN_08	IO port 9 pin 8.
BSP_IO_PORT_09_PIN_09	IO port 9 pin 9.
BSP_IO_PORT_09_PIN_10	IO port 9 pin 10.
BSP_IO_PORT_09_PIN_11	IO port 9 pin 11.
BSP_IO_PORT_09_PIN_12	IO port 9 pin 12.
BSP_IO_PORT_09_PIN_13	IO port 9 pin 13.
BSP_IO_PORT_09_PIN_14	IO port 9 pin 14.
BSP_IO_PORT_09_PIN_15	IO port 9 pin 15.
BSP_IO_PORT_10_PIN_00	IO port 10 pin 0.
BSP_IO_PORT_10_PIN_01	IO port 10 pin 1.
BSP_IO_PORT_10_PIN_02	IO port 10 pin 2.
BSP_IO_PORT_10_PIN_03	IO port 10 pin 3.
BSP_IO_PORT_10_PIN_04	IO port 10 pin 4.
BSP_IO_PORT_10_PIN_05	IO port 10 pin 5.
BSP_IO_PORT_10_PIN_06	IO port 10 pin 6.
BSP_IO_PORT_10_PIN_07	IO port 10 pin 7.
BSP_IO_PORT_10_PIN_08	IO port 10 pin 8.
BSP_IO_PORT_10_PIN_09	IO port 10 pin 9.
BSP_IO_PORT_10_PIN_10	IO port 10 pin 10.
BSP_IO_PORT_10_PIN_11	IO port 10 pin 11.
BSP_IO_PORT_10_PIN_12	IO port 10 pin 12.

BSP_IO_PORT_10_PIN_13	IO port 10 pin 13.
BSP_IO_PORT_10_PIN_14	IO port 10 pin 14.
BSP_IO_PORT_10_PIN_15	IO port 10 pin 15.
BSP_IO_PORT_11_PIN_00	IO port 11 pin 0.
BSP_IO_PORT_11_PIN_01	IO port 11 pin 1.
BSP_IO_PORT_11_PIN_02	IO port 11 pin 2.
BSP_IO_PORT_11_PIN_03	IO port 11 pin 3.
BSP_IO_PORT_11_PIN_04	IO port 11 pin 4.
BSP_IO_PORT_11_PIN_05	IO port 11 pin 5.
BSP_IO_PORT_11_PIN_06	IO port 11 pin 6.
BSP_IO_PORT_11_PIN_07	IO port 11 pin 7.
BSP_IO_PORT_11_PIN_08	IO port 11 pin 8.
BSP_IO_PORT_11_PIN_09	IO port 11 pin 9.
BSP_IO_PORT_11_PIN_10	IO port 11 pin 10.
BSP_IO_PORT_11_PIN_11	IO port 11 pin 11.
BSP_IO_PORT_11_PIN_12	IO port 11 pin 12.
BSP_IO_PORT_11_PIN_13	IO port 11 pin 13.
BSP_IO_PORT_11_PIN_14	IO port 11 pin 14.
BSP_IO_PORT_11_PIN_15	IO port 11 pin 15.

Function Documentation

◆ **R_BSP_PinRead()**

```
__STATIC_INLINE uint32_t R_BSP_PinRead ( bsp_io_port_pin_t pin)
```

Read the current input level of the pin.

Parameters

[in]	pin	The pin

Return values

Current	input level

◆ **R_BSP_PinWrite()**

```
__STATIC_INLINE void R_BSP_PinWrite ( bsp_io_port_pin_t pin, bsp_io_level_t level )
```

Set a pin to output and set the output level to the level provided

Parameters

[in]	pin	The pin
[in]	level	The level

◆ **R_BSP_PinAccessEnable()**

```
__STATIC_INLINE void R_BSP_PinAccessEnable ( void )
```

Enable access to the PFS registers. Uses a reference counter to protect against interrupts that could occur via multiple threads or an ISR re-entering this code.

◆ **R_BSP_PinAccessDisable()**

```
__STATIC_INLINE void R_BSP_PinAccessDisable ( void )
```

Disable access to the PFS registers. Uses a reference counter to protect against interrupts that could occur via multiple threads or an ISR re-entering this code.

4.2 Modules

Detailed Description

Modules are the smallest unit of software available in the FSP. Each module implements one interface.

Modules

High-Speed Analog Comparator (r_acmphs)

This module implements the [Comparator Interface](#) using the high-speed analog comparator.

Low-Power Analog Comparator (r_acmplp)

Driver for the ACMPLP peripheral on RA MCUs. This module implements the [Comparator Interface](#).

Analog to Digital Converter (r_adc)

Driver for the ADC12, ADC14, and ADC16 peripherals on RA MCUs. This module implements the [ADC Interface](#).

Asynchronous General Purpose Timer (r_agt)

Driver for the AGT peripheral on RA MCUs. This module implements the [Timer Interface](#).

Clock Frequency Accuracy Measurement Circuit (r_cac)

Driver for the CAC peripheral on RA MCUs. This module implements the [CAC Interface](#).

Clock Generation Circuit (r_cgc)

Driver for the CGC peripheral on RA MCUs. This module implements the [CGC Interface](#).

Cyclic Redundancy Check (CRC) Calculator (r_crc)

Driver for the CRC peripheral on RA MCUs. This module implements the [CRC Interface](#).

Capacitive Touch Sensing Unit (r_cts)

This HAL driver supports the Capacitive Touch Sensing Unit (CTS). It implements the [CTS Interface](#).

Digital to Analog Converter (r_dac)

Driver for the DAC12 peripheral on RA MCUs. This module implements the [DAC Interface](#).

Direct Memory Access Controller (r_dmac)

Driver for the DMAC peripheral on RA MCUs. This module implements the [Transfer Interface](#).

Data Operation Circuit (r_doc)

Driver for the DOC peripheral on RA MCUs. This module implements the [DOC Interface](#).

D/AVE 2D Port Interface (r_drw)

Driver for the DRW peripheral on RA MCUs. This module is a port of D/AVE 2D.

Data Transfer Controller (r_dtc)

Driver for the DTC peripheral on RA MCUs. This module implements the [Transfer Interface](#).

Event Link Controller (r_elc)

Driver for the ELC peripheral on RA MCUs. This module implements the [ELC Interface](#).

Ethernet (r_ether)

Driver for the Ethernet peripheral on RA MCUs. This module implements the [Ethernet Interface](#).

Ethernet PHY (r_ether_phy)

The Ethernet PHY module (r_ether_phy) provides an API for standard Ethernet PHY communications applications and uses the ETHERC peripherals. It implements the [Ethernet PHY Interface](#).

High-Performance Flash Driver (r_flash_hp)

Driver for the flash memory on RA high-performance MCUs. This module implements the [Flash Interface](#).

Low-Power Flash Driver (r_flash_lp)

Driver for the flash memory on RA low-power MCUs. This module implements the [Flash Interface](#).

Graphics LCD Controller (r_glcdc)

Driver for the GLCDC peripheral on RA MCUs. This module implements the [Display Interface](#).

General PWM Timer (r_gpt)

Driver for the GPT32 and GPT16 peripherals on RA MCUs. This module implements the [Timer Interface](#).

Interrupt Controller Unit (r_icu)

Driver for the ICU peripheral on RA MCUs. This module implements the [External IRQ Interface](#).

I2C Master on IIC (r_iic_master)

Driver for the IIC peripheral on RA MCUs. This module implements the [I2C Master Interface](#).

I2C Slave on IIC (r_iic_slave)

Driver for the IIC peripheral on RA MCUs. This module implements the [I2C Slave Interface](#).

I/O Ports (r_ioport)

Driver for the I/O Ports peripheral on RA MCUs. This module implements the [I/O Port Interface](#).

Independent Watchdog Timer (r_iwdt)

Driver for the IWDT peripheral on RA MCUs. This module implements the [WDT Interface](#).

JPEG Codec (r_jpeg)

Driver for the JPEG peripheral on RA MCUs. This module implements the [JPEG Codec Interface](#).

Key Interrupt (r_kint)

Driver for the KINT peripheral on RA MCUs. This module implements the [Key Matrix Interface](#).

Low Power Modes (r_lpm)

Driver for the LPM peripheral on RA MCUs. This module implements

the [Low Power Modes Interface](#).

[Low Voltage Detection \(r_lvd\)](#)

Driver for the LVD peripheral on RA MCUs. This module implements the [Low Voltage Detection Interface](#).

[Realtime Clock \(r_rtc\)](#)

Driver for the RTC peripheral on RA MCUs. This module implements the [RTC Interface](#).

[Serial Communications Interface \(SCI\) I2C \(r_sci_i2c\)](#)

Driver for the SCI peripheral on RA MCUs. This module implements the [I2C Master Interface](#).

[Serial Communications Interface \(SCI\) SPI \(r_sci_spi\)](#)

Driver for the SCI peripheral on RA MCUs. This module implements the [SPI Interface](#).

[Serial Communications Interface \(SCI\) UART \(r_sci_uart\)](#)

Driver for the SCI peripheral on RA MCUs. This module implements the [UART Interface](#).

[SD/MMC Host Interface \(r_sdhi\)](#)

Driver for the SD/MMC Host Interface (SDHI) peripheral on RA MCUs. This module implements the [SD/MMC Interface](#).

[Serial Peripheral Interface \(r_spi\)](#)

Driver for the SPI peripheral on RA MCUs. This module implements the [SPI Interface](#).

[Serial Sound Interface \(r_ssi\)](#)

Driver for the SSIE peripheral on RA MCUs. This module implements the [I2S Interface](#).

[Universal Serial Bus \(r_usb_basic\)](#)

The USB module (r_usb_basic) provides an API to perform H / W control of USB communication. It implements the [USB Interface](#).

Host Mass Storage Class Driver (r_usb_hmsc)

The USB module (r_usb_hmsc) provides an API to perform hardware control of USB communications. It implements the [USB Interface](#).

Universal Serial Bus Peripheral Communication Device Class (r_usb_pcdc)

This module is USB Peripheral Communication Device Class Driver (PCDC).
This module works in combination with (r_usb_basic module).

Watchdog Timer (r_wdt)

Driver for the WDT peripheral on RA MCUs. This module implements the [WDT Interface](#).

SEGGER emWin Port (rm_emwin_port)

SEGGER emWin port for RA MCUs.

FreeRTOS Plus FAT (rm_freertos_plus_fat)

Middleware for the Fat File System control on RA MCUs.

Amazon FreeRTOS Port (rm_freertos_port)

Amazon FreeRTOS port for RA MCUs.

Crypto Middleware (rm_psa_crypto)

Hardware acceleration for the mbedCrypto implementation of the ARM PSA Crypto API.

Capacitive Touch Middleware (rm_touch)

This module supports the Capacitive Touch Sensing Unit (CTSU). It implements the [Touch Middleware Interface](#).

4.2.1 High-Speed Analog Comparator (r_acmphs)

Modules

Functions

```
fsp_err_t R_ACMPHS_Open (comparator_ctrl_t *p_ctrl, comparator_cfg_t const
                        *const p_cfg)
```

```
fsp_err_t R_ACMPHS_OutputEnable (comparator_ctrl_t *const p_ctrl)
```

```
fsp_err_t R_ACMPHS_InfoGet (comparator_ctrl_t *const p_ctrl,
                           comparator_info_t *const p_info)
```

```
fsp_err_t R_ACMPHS_StatusGet (comparator_ctrl_t *const p_ctrl,
                              comparator_status_t *const p_status)
```

```
fsp_err_t R_ACMPHS_Close (comparator_ctrl_t *const p_ctrl)
```

```
fsp_err_t R_ACMPHS_VersionGet (fsp_version_t *const p_version)
```

Detailed Description

This module implements the [Comparator Interface](#) using the high-speed analog comparator.

Overview

Features

The ACMPHS HAL module supports the following features:

- Callback on rising edge, falling edge or both
- Configurable debounce filter
- Option to include comparator output on VCOOUT pin or ELC events

Configuration

Build Time Configurations for r_acmphs

The following build time configurations are defined in fsp_cfg/r_acmphs_cfg.h:

Configuration	Options	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	If selected code for parameter checking is included in the build.

Configurations for Comparator Driver on r_acmphs

This module can be added to the Threads tab from New -> Driver -> Analog -> Comparator Driver on r_acmphs:

4.2.2 Low-Power Analog Comparator (r_acmplp)

Modules

Functions

fsp_err_t	R_ACMPLP_Open (comparator_ctrl_t *const p_ctrl, comparator_cfg_t const *const p_cfg)
fsp_err_t	R_ACMPLP_OutputEnable (comparator_ctrl_t *const p_ctrl)
fsp_err_t	R_ACMPLP_InfoGet (comparator_ctrl_t *const p_ctrl, comparator_info_t *const p_info)
fsp_err_t	R_ACMPLP_StatusGet (comparator_ctrl_t *const p_ctrl, comparator_status_t *const p_status)
fsp_err_t	R_ACMPLP_Close (comparator_ctrl_t *const p_ctrl)
fsp_err_t	R_ACMPLP_VersionGet (fsp_version_t *const p_version)

Detailed Description

Driver for the ACMPLP peripheral on RA MCUs. This module implements the [Comparator Interface](#).

Overview

Features

The ACMPLP HAL module supports the following features:

- Normal mode or window mode
- Callback on rising edge, falling edge or both
- Configurable debounce filter
- Option to include comparator output on VCOUT pin or ELC events

Configuration

Build Time Configurations for r_acmplp

The following build time configurations are defined in fsp_cfg/r_acmplp_cfg.h:

Configuration	Options	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	If selected code for parameter checking is included in the build.

Reference Voltage Selection
(ACMPLP1)

- IVREF0
- IVREF1

Reference Voltage Selection for
ACMPLP1.

Configurations for Comparator Driver on r_acmplp

This module can be added to the Threads tab from New -> Driver -> Analog -> Comparator Driver on r_acmplp:

4.2.3 Analog to Digital Converter (r_adc)

Modules

Functions

fsp_err_t R_ADC_Open (adc_ctrl_t *p_ctrl, adc_cfg_t const *const p_cfg)

fsp_err_t R_ADC_ScanCfg (adc_ctrl_t *p_ctrl, adc_channel_cfg_t const *const p_channel_cfg)

fsp_err_t R_ADC_InfoGet (adc_ctrl_t *p_ctrl, adc_info_t *p_adc_info)

fsp_err_t R_ADC_ScanStart (adc_ctrl_t *p_ctrl)

fsp_err_t R_ADC_ScanStop (adc_ctrl_t *p_ctrl)

fsp_err_t R_ADC_StatusGet (adc_ctrl_t *p_ctrl, adc_status_t *p_status)

fsp_err_t R_ADC_Read (adc_ctrl_t *p_ctrl, adc_channel_t const reg_id, uint16_t *const p_data)

fsp_err_t R_ADC_Read32 (adc_ctrl_t *p_ctrl, adc_channel_t const reg_id, uint32_t *const p_data)

fsp_err_t R_ADC_SampleStateCountSet (adc_ctrl_t *p_ctrl, adc_sample_state_t *p_sample)

fsp_err_t R_ADC_Close (adc_ctrl_t *p_ctrl)

fsp_err_t R_ADC_OffsetSet (adc_ctrl_t *const p_ctrl, adc_channel_t const reg_id, int32_t offset)

fsp_err_t R_ADC_Calibrate (adc_ctrl_t *const p_ctrl, void *const p_extend)

fsp_err_t R_ADC_VersionGet (fsp_version_t *const p_version)

Detailed Description

Driver for the ADC12, ADC14, and ADC16 peripherals on RA MCUs. This module implements the [ADC Interface](#).

Overview

Features

The ADC module supports the following features:

- 12, 14, or 16 bit maximum resolution depending on the MCU
- Configure scans to include:
 - Multiple analog channels
 - Temperature sensor channel
 - Voltage sensor channel
- Configurable scan start trigger:
 - Software scan triggers
 - Hardware scan triggers (timer expiration, for example)
 - External scan triggers from the ADTRGn port pins
- Configurable scan mode:
 - Single scan mode, where each trigger starts a single scan
 - Continuous scan mode, where all channels are scanned continuously
 - Group scan mode, where channels are grouped into group A and group B. The groups can be assigned different start triggers, and group A can be given priority over group B. When group A has priority over group B, a group A trigger suspends an ongoing group B scan.
- Supports adding and averaging converted samples
- Optional callback when scan completes
- Supports reading converted data
- Sample and hold support

Configuration

Build Time Configurations for r_adc

The following build time configurations are defined in fsp_cfg/r_adc_cfg.h:

Configuration	Options	Description
Parameter Checking	• BSP • Enabled • Disabled	If selected code for parameter checking is included in the build.

Configurations for ADC Driver on r_adc

This module can be added to the Threads tab from New -> Driver -> Analog -> ADC Driver on r_adc:

4.2.4 Asynchronous General Purpose Timer (r_agt)

Modules

Detailed Description

Driver for the AGT peripheral on RA MCUs. This module implements the [Timer Interface](#).

Overview

Features

The AGT module has the following features:

- Supports periodic mode, one-shot mode, and PWM mode.
- Signal can be output to a pin.
- Configurable period (counts per timer cycle).
- Configurable duty cycle in PWM mode.
- Configurable clock source, including PCLKB, LOCO, SUBCLK, and external sources input to AGTIO.
- Supports runtime reconfiguration of period.
- Supports runtime reconfiguration of duty cycle in PWM mode.
- Supports counting based on an external clock input to AGTIO.
- Supports debounce filter on AGTIO pins.
- Supports measuring pulse width or pulse period.
- APIs are provided to start, stop, and reset the counter.
- APIs are provided to get the current period, source clock frequency, and count direction.
- APIs are provided to get the current timer status and counter value.

Selecting a Timer

RA MCUs have two timer peripherals: the General PWM Timer (GPT) and the Asynchronous General Purpose Timer (AGT). When selecting between them, consider these factors:

	GPT	AGT
Low Power Modes	The GPT can operate in sleep mode.	The AGT can operate in all low power modes (when count source is LOCO or subclock).
Available Channels	The number of GPT channels is device specific. All currently supported MCUs have at least 7 GPT channels.	All MCUs have 2 AGT channels.
Timer Resolution	All MCUs have at least one 32-bit GPT timer.	The AGT timers are 16-bit timers.
Clock Source	The GPT runs off PCLKD with a configurable divider up to 1024. It can also be configured to count ELC events or external pulses.	The AGT runs off PCLKB, LOCO, or subclock with a configurable divider up to 8 for PCLKB or up to 128 for LOCO or subclock.

Configuration

Build Time Configurations for r_agt

The following build time configurations are defined in fsp_cfg/r_agt_cfg.h:

Configuration	Options	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	If selected code for parameter checking is included in the build.
Pin Output Support	• Disabled • Enabled	If selected code for outputting a waveform to a pin is included in the build.
Pin Input Support	• Disabled • Enabled	Enable input support to use pulse width measurement mode, pulse period measurement mode, or input from P402, P402, or AGTIO.

Configurations for Timer Driver on r_agt

This module can be added to the Threads tab from New -> Driver -> Timers -> Timer Driver on r_agt:

4.2.5 Clock Frequency Accuracy Measurement Circuit (r_cac)

Modules

Functions

```
fsp_err_t R_CAC_Open (cac_ctrl_t *const p_ctrl, cac_cfg_t const *const p_cfg)
```

```
fsp_err_t R_CAC_StartMeasurement (cac_ctrl_t *const p_ctrl)
```

```
fsp_err_t R_CAC_StopMeasurement (cac_ctrl_t *const p_ctrl)
```

```
fsp_err_t R_CAC_Read (cac_ctrl_t *const p_ctrl, uint16_t *const p_counter)
```

```
fsp_err_t R_CAC_Close (cac_ctrl_t *const p_ctrl)
```

```
fsp_err_t R_CAC_VersionGet (fsp_version_t *const p_version)
```

Detailed Description

Driver for the CAC peripheral on RA MCUs. This module implements the [CAC Interface](#).

Overview

The interface for the clock frequency accuracy measurement circuit (CAC) peripheral is used to check a system clock frequency with a reference clock signal by counting the number of measurement clock edges that occur between two edges of the reference clock.

Features

- Supports clock frequency-measurement and monitoring based on a reference signal input
- Reference can be either an externally supplied clock source or an internal clock source
- An interrupt request may optionally be generated by a completed measurement, a detected frequency error, or a counter overflow.
- A digital filter is available for an externally supplied reference clock, and dividers are available for both internally supplied measurement and reference clocks.
- Edge-detection options for the reference clock are configurable as rising, falling, or both.

Configuration

Build Time Configurations for r_cac

The following build time configurations are defined in fsp_cfg/r_cac_cfg.h:

Configuration	Options	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	If selected code for parameter checking is included in the build.

Configurations for Clock Accuracy Circuit Driver on r_cac

This module can be added to the Threads tab from New -> Driver -> Monitoring -> Clock Accuracy Circuit Driver on r_cac:

4.2.6 Clock Generation Circuit (r_cgc)

Modules

Functions

fsp_err_t	R_CGC_Open (cgc_ctrl_t *const p_ctrl, cgc_cfg_t const *const p_cfg)
fsp_err_t	R_CGC_ClocksCfg (cgc_ctrl_t *const p_ctrl, cgc_clocks_cfg_t const *const p_clock_cfg)
fsp_err_t	R_CGC_ClockStart (cgc_ctrl_t *const p_ctrl, cgc_clock_t clock_source, cgc_pll_cfg_t const *const p_pll_cfg)
fsp_err_t	R_CGC_ClockStop (cgc_ctrl_t *const p_ctrl, cgc_clock_t clock_source)
fsp_err_t	R_CGC_ClockCheck (cgc_ctrl_t *const p_ctrl, cgc_clock_t clock_source)
fsp_err_t	R_CGC_SystemClockSet (cgc_ctrl_t *const p_ctrl, cgc_clock_t clock_source, cgc_divider_cfg_t const *const p_divider_cfg)

```
fsp_err_t R_CGC_SystemClockGet (cgc_ctrl_t *const p_ctrl, cgc_clock_t *const
p_clock_source, cgc_divider_cfg_t *const p_divider_cfg)
```

```
fsp_err_t R_CGC_OscStopDetectEnable (cgc_ctrl_t *const p_ctrl)
```

```
fsp_err_t R_CGC_OscStopDetectDisable (cgc_ctrl_t *const p_ctrl)
```

```
fsp_err_t R_CGC_OscStopStatusClear (cgc_ctrl_t *const p_ctrl)
```

```
fsp_err_t R_CGC_Close (cgc_ctrl_t *const p_ctrl)
```

```
fsp_err_t R_CGC_VersionGet (fsp_version_t *version)
```

Detailed Description

Driver for the CGC peripheral on RA MCUs. This module implements the [CGC Interface](#).

Overview

Features

The CGC module supports runtime modifications of clock settings. Key features include the following:

- Supports changing the system clock source to any of the following options (provided they are supported on the MCU):
 - High-speed on-chip oscillator (HOCO)
 - Middle-speed on-chip oscillator (MOCO)
 - Low-speed on-chip oscillator (LOCO)
 - Main oscillator (external resonator or external clock input frequency)
 - Sub-clock oscillator (external resonator)
 - PLL (not available on all MCUs)
- When the system core clock frequency changes, the following things are updated:
 - The CMSIS standard global variable SystemCoreClock is updated to reflect the new clock frequency.
 - Wait states for ROM and RAM are adjusted to the minimum supported value for the new clock frequency.
 - The operating power control mode is updated to the minimum supported value for the new clock settings.
- Supports starting or stopping any of the system clock sources
- Supports changing dividers for the internal clocks
- Supports the oscillation stop detection feature

Note

This module is not required for the initial clock configuration. Initial clock settings are configurable on the Clocks tab of the configuration tool. The initial clock settings are applied by the BSP during the startup process before main.

Internal Clocks

The RA microcontrollers have up to seven internal clocks. Not all internal clocks exist on all MCUs. Each clock domain has its own divider that can be updated in [R_CGC_SystemClockSet\(\)](#). The dividers are subject to constraints described in the footnote of the table "Specifications of the Clock Generation Circuit for the internal clocks" in the hardware manual.

The internal clocks include:

- System clock (ICLK): core clock used for CPU, flash, internal SRAM, DTC, and DMAC
- PCLKA/PCLKB/PCLKC/PCLKD: Peripheral clocks, refer to the table "Specifications of the Clock Generation Circuit for the internal clocks" in the hardware manual to see which peripherals are controlled by which clocks.
- FCLK: Clock source for reading data flash and for programming/erasure of both code and data flash.
- BCLK: External bus clock

Configuration

Note

The initial clock settings are configurable on the Clocks tab of the configuration tool.

There is a configuration to enable the HOCO on reset in the OFS1 settings on the BSP tab.

The following clock related settings are configurable in the RA Common section on the BSP tab:

- Main Oscillator Wait Time
- Main Oscillator Clock Source (external oscillator or crystal/resonator)
- Subclock Populated
- Subclock Drive
- Subclock Stabilization Time (ms)

Build Time Configurations for r_cgc

The following build time configurations are defined in fsp_cfg/r_cgc_cfg.h:

Configuration	Options	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	If selected code for parameter checking is included in the build.

Configurations for CGC Driver on r_cgc

This module can be added to the Threads tab from New -> Driver -> System -> CGC Driver on r_cgc:

4.2.7 Cyclic Redundancy Check (CRC) Calculator (r_crc)

Modules

Functions

```
fsp_err_t R_CRC_Open (crc_ctrl_t *const p_ctrl, crc_cfg_t const *const p_cfg)
```

```
fsp_err_t R_CRC_Close (crc_ctrl_t *const p_ctrl)
```



```
fsp_err_t R_CRC_Calculate (crc_ctrl_t *const p_ctrl, crc_input_t *const
                          p_crc_input, uint32_t *calculatedValue)
```

```
fsp_err_t R_CRC_CalculatedValueGet (crc_ctrl_t *const p_ctrl, uint32_t
                                   *calculatedValue)
```

```
fsp_err_t R_CRC_SnoopEnable (crc_ctrl_t *const p_ctrl, uint32_t crc_seed)
```

```
fsp_err_t R_CRC_SnoopDisable (crc_ctrl_t *const p_ctrl)
```

```
fsp_err_t R_CRC_VersionGet (fsp_version_t *const p_version)
```

Detailed Description

Driver for the CRC peripheral on RA MCUs. This module implements the [CRC Interface](#).

Overview

The CRC module provides a API to calculate 8, 16 and 32-bit CRC values on a block of data in memory or a stream of data over a Serial Communication Interface (SCI) channel using industry-standard polynomials.

Features

- CRC module supports the following 8 and 16 bit CRC polynomials which operates on 8-bit data in parallel
 - $X^8 + X^2 + X + 1$ (CRC-8)
 - $X^{16} + X^{15} + X^2 + 1$ (CRC-16)
 - $X^{16} + X^{12} + X^5 + 1$ (CRC-CCITT)
- CRC module supports the following 32 bit CRC polynomials which operates on 32-bit data in parallel
 - $X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + X + 1$ (CRC-32)
 - $X^{32} + X^{28} + X^{27} + X^{26} + X^{25} + X^{23} + X^{22} + X^{20} + X^{19} + X^{18} + X^{14} + X^{13} + X^{11} + X^{10} + X^9 + X^8 + X^6 + 1$ (CRC-32C)
- CRC module can calculate CRC with LSB first or MSB first bit order.

Configuration

Build Time Configurations for r_crc

The following build time configurations are defined in fsp_cfg/r_crc_cfg.h:

Configuration	Options	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	If selected code for parameter checking is included in the build.

Configurations for CRC Driver on r_crc

This module can be added to the Threads tab from New -> Driver -> Monitoring -> CRC Driver on r_crc:

4.2.8 Capacitive Touch Sensing Unit (r_ctsu)

Modules

Functions

`fsp_err_t` `R_CTSU_Open` (`ctsu_ctrl_t *const p_ctrl`, `ctsu_cfg_t const *const p_cfg`)

Opens and configures the CTSU driver module. Implements `ctsu_api_t::open`. [More...](#)

`fsp_err_t` `R_CTSU_ScanStart` (`ctsu_ctrl_t *const p_ctrl`)

This function should be called each time a periodic timer expires. If initial offset tuning is enabled, The first several calls are used to tuning for the sensors. Once that is complete, normal processing of the data from the last scan occurs. If a different control block should be run on the next scan, that is set up as well, then the next scan is started. Implements `ctsu_api_t::scanStart`. [More...](#)

`fsp_err_t` `R_CTSU_DataGet` (`ctsu_ctrl_t *const p_ctrl`, `uint16_t *p_data`)

This function gets the sensor values as scanned by the CTSU. Implements `ctsu_api_t::dataGet`. [More...](#)

`fsp_err_t` `R_CTSU_Close` (`ctsu_ctrl_t *const p_ctrl`)

Disables specified CTSU control block. Implements `transfer_api_t::close`. [More...](#)

`fsp_err_t` `R_CTSU_VersionGet` (`fsp_version_t *const p_version`)

Return CTSU HAL driver version. Implements `ctsu_api_t::versionGet`. [More...](#)

Detailed Description

This HAL driver supports the Capacitive Touch Sensing Unit (CTSUS). It implements the [CTSUS Interface](#).

Overview

The capacitive touch sensing unit HAL driver (r_ctorsu) provides an API to control the CTSU peripheral. Capacitance measurement with various settings is possible by editing the configuration.

Features

- Supports both Self-capacitance multi scan mode and Mutual-capacitance full scan mode.
 - The settings related to scanning can change in detail.
 - Grouping of scans is possible.
- Starts scanning at any time.
 - The scan may be started by a software trigger or an external trigger.
 - The scan completion is signalled by the callback function.
- Gets all results after scans are complete.
- Additional build-time features
 - Optional (build time) DTC support for CTSUWR and CTSURD respectively.
 - Optional (build time) Support for real-time monitoring function by QE. (Not yet available)

Configuration

Build Time Configurations for r_ctorsu

The following build time configurations are defined in fsp_cfg/r_ctorsu_cfg.h:

Configuration	Options	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	If selected code for parameter checking is included in the build.
Enable Support for using DTC	• Enabled • Disabled	If enabled, DTC instances will be included in the build for both transmission and reception.
Interrupt priority level	Interrupt vector number must be an integer greater than 0	Priority level of all CTSU interrupt (CSTU_WR, CTSU_RD, CTSU_FN)
NUM_SELF_ELEMENTS	Interrupt vector number must be an integer greater than 0	Number of self elements
NUM_MUTUAL_ELEMENTS	Interrupt vector number must be an integer greater than 0	Number of mutual elements

Configurations for CTSU Driver on r_ctorsu

This module can be added to the Threads tab from New -> Driver -> CapTouch -> CTSU Driver on r_ctorsu:

4.2.9 Digital to Analog Converter (r_dac)

Modules

Functions

```
fsp_err_t R_DAC_Open (dac_ctrl_t *p_api_ctrl, dac_cfg_t const *const p_cfg)
```

```
fsp_err_t R_DAC_Write (dac_ctrl_t *p_api_ctrl, uint16_t value)
```

```
fsp_err_t R_DAC_Start (dac_ctrl_t *p_api_ctrl)
```

```
fsp_err_t R_DAC_Stop (dac_ctrl_t *p_api_ctrl)
```

```
fsp_err_t R_DAC_Close (dac_ctrl_t *p_api_ctrl)
```

```
fsp_err_t R_DAC_VersionGet (fsp_version_t *p_version)
```

Detailed Description

Driver for the DAC12 peripheral on RA MCUs. This module implements the [DAC Interface](#).

Overview

Features

The DAC module outputs one of 4096 voltage levels between the positive and negative reference voltages.

- Supports setting left-justified or right-justified 12-bit value format for the 16-bit input data registers
- Supports output amplifiers on selected MCUs
- Supports charge pump on selected MCUs
- Operate in synchronous anti-interference mode with the Analog-to-Digital Converter (ADC) module.

Configuration

Build Time Configurations for r_dac

The following build time configurations are defined in fsp_cfg/r_dac_cfg.h:

Configuration	Options	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	If selected code for parameter checking is included in the build.

Configurations for DAC Driver on r_dac

This module can be added to the Threads tab from New -> Driver -> Analog -> DAC Driver on r_dac:

4.2.10 Direct Memory Access Controller (r_dmac)

Modules

Functions

fsp_err_t	R_DMAC_Open (transfer_ctrl_t *const p_api_ctrl, transfer_cfg_t const *const p_cfg)
fsp_err_t	R_DMAC_Reconfigure (transfer_ctrl_t *const p_api_ctrl, transfer_info_t *p_info)
fsp_err_t	R_DMAC_Reset (transfer_ctrl_t *const p_api_ctrl, void const *volatile p_src, void *volatile p_dest, uint16_t const num_transfers)
fsp_err_t	R_DMAC_SoftwareStart (transfer_ctrl_t *const p_api_ctrl, transfer_start_mode_t mode)
fsp_err_t	R_DMAC_SoftwareStop (transfer_ctrl_t *const p_api_ctrl)
fsp_err_t	R_DMAC_Enable (transfer_ctrl_t *const p_api_ctrl)
fsp_err_t	R_DMAC_Disable (transfer_ctrl_t *const p_api_ctrl)
fsp_err_t	R_DMAC_InfoGet (transfer_ctrl_t *const p_api_ctrl, transfer_properties_t *const p_info)
fsp_err_t	R_DMAC_Close (transfer_ctrl_t *const p_api_ctrl)
fsp_err_t	R_DMAC_VersionGet (fsp_version_t *const p_version)

Detailed Description

Driver for the DMAC peripheral on RA MCUs. This module implements the [Transfer Interface](#).

Overview

The Direct Memory Access Controller (DMAC) transfers data from one memory location to another without using the CPU.

Features

- Supports multiple transfer modes
 - Normal transfer
 - Repeat transfer
 - Block transfer
- Address increment, decrement, fixed, or offset modes
- Triggered by ELC events
 - Some exceptions apply, see the Event table in the Event Numbers section of the Interrupt Controller Unit chapter of the hardware manual
- Supports 1, 2, and 4 byte data units

Transfer Modes

The DMAC Module supports three modes of operation.

- **Normal Mode** - In normal mode, a single data unit is transferred every time the configured ELC event is received by the DMAC channel. A data unit can be 1-byte, 2-bytes, or 4-bytes. The source and destination addresses can be fixed, increment, decrement, or add an offset to the next data unit after each transfer. A 16-bit counter decrements after each transfer. When the counter reaches 0, transfers will no longer be triggered by the ELC event and the CPU can be interrupted to signal that all transfers have finished.
- **Repeat Mode** - Repeat mode works the same way as normal mode, however the length is limited to an integer in the range[1,1024]. When the transfer counter reaches 0, the counter is reset to its configured value, the repeat area(source or destination address) resets to its starting address and the block count remaining will decrement by 1. When the block count reaches 0, transfers will no longer be triggered by the ELC event and the CPU may be interrupted to signal that all transfers have finished.
- **Block Mode** - In block mode, the amount of data units transferred by each interrupt can be set to an integer in the range [1,1024]. The number of blocks to transfer can also be configured to a 16-bit number. After each block transfer the repeat area(source or destination address) will reset to the original address and the other address will be incremented or decremented to the next block.

Selecting the DTC or DMAC

The Transfer API is implemented by both DTC and the DMAC so that applications can switch between the DTC and the DMAC. When selecting between them, consider these factors:

	DTC	DMAC
Repeat Mode	• Repeats forever • Max repeat size is 256 x 4 bytes	• Configurable number of repeats • Max repeat size is 1024 x 4 bytes
Block Mode	• Max block size is 256 x 4 bytes	• Max block size is 1024 x 4 bytes
Channels	• One instance per interrupt	• MCU specific (8 channels or less)
Chained Transfers	• Supported	• Not Supported
Software Trigger	• Must use the software ELC event	• Has support for software trigger without using software ELC event • Supports TRANSFER_START_MODE_SINGLE and TRANSFER_START_MODE_REPEAT
Offset Address Mode	• Not supported	• Supported

Interrupts

The DTC and DMAC interrupts behave differently. The DTC uses the configured IELSR event IRQ as the interrupt source whereas each DMAC channel has its own IRQ.

The transfer_info_t::irq setting also behaves a little differently depending on which mode is selected.

Normal Mode

	DTC	DMAC
TRANSFER_IRQ_EACH	Interrupt after each transfer	N/A
TRANSFER_IRQ_END	Interrupt after last transfer	Interrupt after last transfer

Repeat Mode

	DTC	DMAC
TRANSFER_IRQ_EACH	Interrupt after each transfer	Interrupt after each repeat
TRANSFER_IRQ_END	Interrupt after each repeat	Interrupt after last transfer

Block Mode

	DTC	DMAC
TRANSFER_IRQ_EACH	Interrupt after each block	Interrupt after each block
TRANSFER_IRQ_END	Interrupt after last block	Interrupt after last block

Additional Considerations

- The DTC requires a moderate amount of RAM (one [transfer_info_t](#) struct per open instance + DTC_VECTOR_TABLE_SIZE).
- The DTC stores transfer information in RAM and writes back to RAM after each transfer whereas the DMAC stores all transfer information in registers.
- When transfers are configured for more than one activation source, the DTC must fetch the transfer info from RAM on each interrupt. This can cause a higher latency between transfers.

Configuration

Build Time Configurations for r_dmac

The following build time configurations are defined in fsp_cfg/r_dmac_cfg.h:

Configuration	Options	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	If selected code for parameter checking is included in the build.

Configurations for Transfer Driver on r_dmac

This module can be added to the Threads tab from New -> Driver -> Transfer -> Transfer Driver on r_dmac :

4.2.11 Data Operation Circuit (r_doc)

Modules

Functions

`fsp_err_t` `R_DOC_Open` (`doc_ctrl_t` *const p_api_ctrl, `doc_cfg_t` const *const p_cfg)

`fsp_err_t` `R_DOC_Close` (`doc_ctrl_t` *const p_api_ctrl)

`fsp_err_t` `R_DOC_StatusGet` (`doc_ctrl_t` *const p_api_ctrl, `doc_status_t` *const p_status)

`fsp_err_t` `R_DOC_Write` (`doc_ctrl_t` *const p_api_ctrl, `uint16_t` data)

`fsp_err_t` `R_DOC_VersionGet` (`fsp_version_t` *const p_version)

Detailed Description

Driver for the DOC peripheral on RA MCUs. This module implements the [DOC Interface](#).

Overview

Features

The DOC HAL module peripheral is used to compare, add or subtract 16-bit data and can detect the following events:

- A mismatch or match between data values
- Overflow of an addition operation
- Underflow of a subtraction operation

A user-defined callback can be created to inform the CPU when any of above events occur.

Configuration

Build Time Configurations for r_doc

The following build time configurations are defined in `fsp_cfg/r_doc_cfg.h`:

Configuration	Options	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	If selected code for parameter checking is included in the build.

Configurations for Data Operation Circuit Driver on r_doc

This module can be added to the Threads tab from New -> Driver -> Monitoring -> Data Operation Circuit Driver on r_doc:

4.2.12 D/AVE 2D Port Interface (r_drw)

Modules

Driver for the DRW peripheral on RA MCUs. This module is a port of D/AVE 2D.

Overview

Note

The D/AVE 2D Port Interface (D1 layer) does not provide any interfaces to the user. Consult the D/AVE 2D driver documentation for further information.

For cross-platform compatibility purposes the D1 and D2 APIs are not bound by the Flex Software Package coding guidelines for function names and general module functionality.

Configuration

Build Time Configurations for r_drw

The following build time configurations are defined in fsp_cfg/r_drw_cfg.h:

Configuration	Options	Description
Allow Indirect Mode	- Enabled - Disabled	Enable indirect mode to allow no-copy mode for d2_addddlist (see the D/AVE 2D driver documentation for details).
Memory Allocation	- Default - Custom	Set Memory Allocation to Default to use built-in dynamic memory allocation for the D2 heap. This will use an RTOS heap if configured; otherwise, standard C malloc and free will be used. Set to Custom to define your own allocation scheme for the D2 heap. In this case, the developer will need to define the following functions: <pre>void * d1_malloc(size_t size) void d1_free(void * ptr)</pre>

4.2.13 Data Transfer Controller (r_dtc)

Modules

Functions

fsp_err_t	R_DTC_Open (transfer_ctrl_t *const p_api_ctrl, transfer_cfg_t const *const p_cfg)
fsp_err_t	R_DTC_Reconfigure (transfer_ctrl_t *const p_api_ctrl, transfer_info_t *p_info)
fsp_err_t	R_DTC_Reset (transfer_ctrl_t *const p_api_ctrl, void const *volatile p_src, void *volatile p_dest, uint16_t const num_transfers)
fsp_err_t	R_DTC_SoftwareStart (transfer_ctrl_t *const p_api_ctrl, transfer_start_mode_t mode)
fsp_err_t	R_DTC_SoftwareStop (transfer_ctrl_t *const p_api_ctrl)
fsp_err_t	R_DTC_Enable (transfer_ctrl_t *const p_api_ctrl)
fsp_err_t	R_DTC_Disable (transfer_ctrl_t *const p_api_ctrl)
fsp_err_t	R_DTC_InfoGet (transfer_ctrl_t *const p_api_ctrl, transfer_properties_t *const p_properties)
fsp_err_t	R_DTC_Close (transfer_ctrl_t *const p_api_ctrl)
fsp_err_t	R_DTC_VersionGet (fsp_version_t *const p_version)

Detailed Description

Driver for the DTC peripheral on RA MCUs. This module implements the [Transfer Interface](#).

Overview

The Data Transfer Controller (DTC) transfers data from one memory location to another without using the CPU.

The DTC uses a RAM based vector table. Each entry in the vector table corresponds to an entry in the ISR vector table. When the DTC is triggered by an interrupt, it reads the DTC vector table, fetches the transfer information, and then executes the transfer. After the transfer is executed, the DTC writes the updated transfer info back to the location pointed to by the DTC vector table.

Features

- Supports multiple transfer modes
 - Normal transfer
 - Repeat transfer
 - Block transfer
- Chain transfers
- Address increment, decrement or fixed modes
- Can be triggered by any event that has reserved a slot in the interrupt vector table.

- Some exceptions apply, see the Event table in the Event Numbers section of the Interrupt Controller Unit chapter of the hardware manual
- Supports 1, 2, and 4 byte data units

Transfer Modes

The DTC Module supports three modes of operation.

- **Normal Mode** - In normal mode, a single data unit is transferred every time an interrupt is received by the DTC. A data unit can be 1-byte, 2-bytes, or 4-bytes. The source and destination addresses can be fixed, increment or decrement to the next data unit after each transfer. A 16-bit counter(length) decrements after each transfer. When the counter reaches 0, transfers will no longer be triggered by the interrupt source and the CPU can be interrupted to signal that all transfers have finished.
- **Repeat Mode** - Repeat mode works the same way as normal mode, however the length is limited to an integer in the range[1,256]. When the transfer counter reaches 0, the counter is reset to its configured value and the repeat area(source or destination address) resets to its starting address and transfers will still be triggered by the interrupt.
- **Block Mode** - In block mode, the amount of data units transferred by each interrupt can be set to an integer in the range [1,256]. The number of blocks to transfer can also be configured to a 16-bit number. After each block transfer the repeat area(source or destination address) will reset to the original address and the other address will be incremented or decremented to the next block.

Note

1. The source and destination address of the transfer must be aligned to the configured data unit.
2. In normal mode the length can be set to [0,65535]. When the length is set to 0, then the transaction will execute 65536 transfers not 0.
3. In block mode, num_blocks can be set to [0,65535]. When the length is set to 0, then the transaction will execute 65536 transfers not 0.

Chaining Transfers

Multiple transfers can be configured for the same interrupt source by specifying an array of `transfer_info_t` structs instead of just passing a pointer to one. In this configuration, every `transfer_info_t` struct must be configured for a chain mode except for the last one. There are two types of chain mode; CHAIN_MODE_EACH and CHAIN_MODE_END. If a transfer is configured in CHAIN_MODE_EACH then it triggers the next transfer in the chain after it completes each transfer. If a transfer is configured in CHAIN_MODE_END then it triggers the next transfer in the chain after it completes its last transfer.

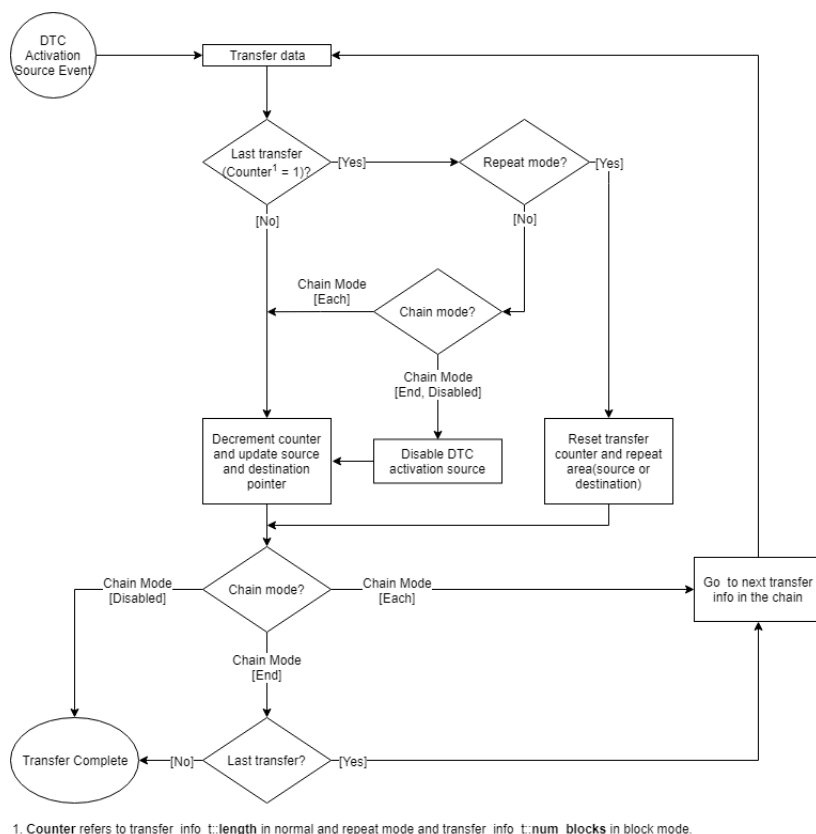


Figure 93: DTC Transfer Flowchart

Selecting the DTC or DMAC

The Transfer API is implemented by both DTC and the DMAC so that applications can switch between the DTC and the DMAC. When selecting between them, consider these factors:

	DTC	DMAC
Repeat Mode	- Repeats forever - Max repeat size is 256 x 4 bytes	- Configurable number of repeats - Max repeat size is 1024 x 4 bytes
Block Mode	Max block size is 256 x 4 bytes	Max block size is 1024 x 4 bytes
Channels	One instance per interrupt	MCU specific (8 channels or less)
Chained Transfers	Supported	Not Supported
Software Trigger	Must use the software ELC event	- Has support for software trigger without using software ELC event - Supports TRANSFER_START_MODE_SINGLE and TRANSFER_START_MODE_REPEAT

Offset Address Mode

• Not supported

• Supported

Additional Considerations

- The DTC requires a moderate amount of RAM (one `transfer_info_t` struct per open instance + `DTC_VECTOR_TABLE_SIZE`).
- The DTC stores transfer information in RAM and writes back to RAM after each transfer whereas the DMAC stores all transfer information in registers.
- When transfers are configured for more than one activation source, the DTC must fetch the transfer info from RAM on each interrupt. This can cause a higher latency between transfers.
- The DTC interrupts the CPU using the activation source's IRQ. Each DMAC channel has its own IRQ.

Interrupts

The DTC and DMAC interrupts behave differently. The DTC uses the configured IELSR event IRQ as the interrupt source whereas each DMAC channel has its own IRQ.

The `transfer_info_t::irq` setting also behaves a little differently depending on which mode is selected.

Normal Mode

	DTC	DMAC
TRANSFER_IRQ_EACH	Interrupt after each transfer	N/A
TRANSFER_IRQ_END	Interrupt after last transfer	Interrupt after last transfer

Repeat Mode

	DTC	DMAC
TRANSFER_IRQ_EACH	Interrupt after each transfer	Interrupt after each repeat
TRANSFER_IRQ_END	Interrupt after each repeat	Interrupt after last transfer

Block Mode

	DTC	DMAC
TRANSFER_IRQ_EACH	Interrupt after each block	Interrupt after each block
TRANSFER_IRQ_END	Interrupt after last block	Interrupt after last block

Note

$DTC\_VECTOR\_TABLE\_SIZE = (ICU\_NVIC\_IRQ\_SOURCES \times 4) \text{ Bytes}$

Configuration

Build Time Configurations for r_dtc

The following build time configurations are defined in `fsp_cfg/r_dtc_cfg.h`:

Configuration	Options	Description
Parameter Checking	- Default (BSP) - Enabled - Disabled	If selected code for parameter checking is included in the build.
Linker section to keep DTC vector table	Configurable String	Section to place the DTC vector table.

Configurations for Transfer Driver on r_dtc

This module can be added to the Threads tab from New -> Driver -> Transfer -> Transfer Driver on r_dtc :

4.2.14 Event Link Controller (r_elc)

Modules

Functions

fsp_err_t R_ELC_Open (elc_ctrl_t *const p_ctrl, elc_cfg_t const *const p_cfg)

fsp_err_t R_ELC_Close (elc_ctrl_t *const p_ctrl)

fsp_err_t R_ELC_SoftwareEventGenerate (elc_ctrl_t *const p_ctrl, elc_software_event_t event_number)

fsp_err_t R_ELC_LinkSet (elc_ctrl_t *const p_ctrl, elc_peripheral_t peripheral, elc_event_t signal)

fsp_err_t R_ELC_LinkBreak (elc_ctrl_t *const p_ctrl, elc_peripheral_t peripheral)

fsp_err_t R_ELC_Enable (elc_ctrl_t *const p_ctrl)

fsp_err_t R_ELC_Disable (elc_ctrl_t *const p_ctrl)

fsp_err_t R_ELC_VersionGet (fsp_version_t *const p_version)

Detailed Description

Driver for the ELC peripheral on RA MCUs. This module implements the [ELC Interface](#).

Overview

The event link controller (ELC) uses the event requests generated by various peripheral modules as source signals to connect (link) them to different modules, allowing direct cooperation between the modules without central processing unit (CPU) intervention. The conceptual diagram below illustrates a potential setup where a pin interrupt triggers a timer which later triggers an ADC conversion and CTSU scan, while at the same time a serial communication interrupt automatically starts a data transfer. These tasks would be automatically handled without the need for polling or interrupt

management.

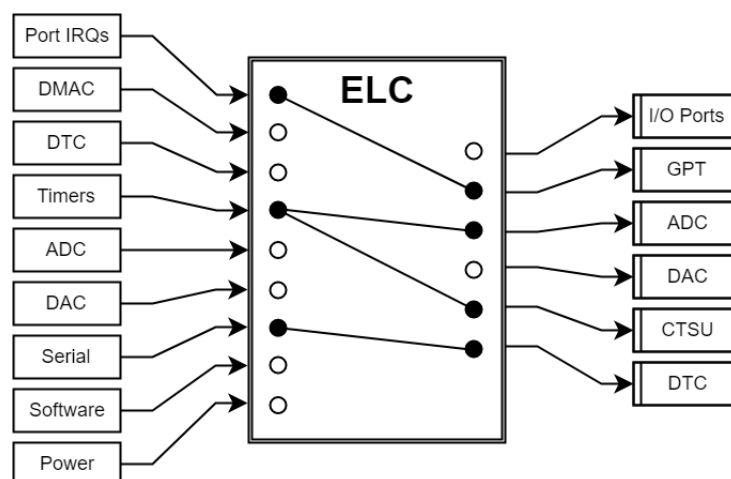


Figure 94: Event Link Controller Conceptual Diagram

In essence, the ELC is an array of multiplexers to route a wide variety of interrupt signals to a subset of peripheral functions. Events are linked by setting the multiplexer for the desired function to the desired signal (through `R_ELC_LinkSet`). The diagram below illustrates one peripheral output of the ELC. In this example, a conversion start is triggered for ADC0 Group A when the GPT0 counter overflows:

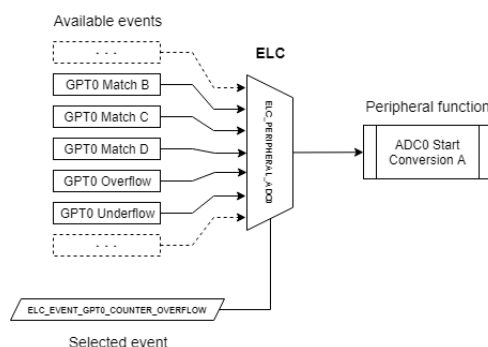


Figure 95: ELC Example

Features

The ELC HAL module can perform the following functions:

- Initialize the ELC to a pre-defined set of links
- Create an event link between two blocks
- Break an event link between two blocks
- Generate one of two software events that interrupt the CPU
- Globally enable or disable event links

A variety of functions can be activated via events, including:

- General-purpose timer (GPT) control
- ADC and DAC conversion start
- Synchronized I/O port output (ports 1-4 only)

- Capacitive touch unit (CTSU) measurement activation

Note

The available sources and peripherals may differ between devices. A full list of selectable peripherals and events is available in the User's Manual for your device.

The source and destination peripherals must be configured to generate and receive events, respectively. Details on how to enable event functionality are located in the User's Manual for your device.

Configuration

To link an event to a peripheral perform the following steps:

1. Configure the operation of the destination peripheral (including any configuration necessary to receive events)
2. Use `R_ELC_LinkSet` to set the desired event link to the peripheral
3. (Optional) If autostart is not enabled, use `R_ELC_Enable` to enable transmission of event signals
4. Configure the signaling module to output the desired event (typically an interrupt)

To disable the event, either use `R_ELC_LinkBreak` to clear the link for a specific event or `R_ELC_Disable` to globally disable event linking.

Note

The ELC module needs no pin, clocking or interrupt configuration; it is merely a mechanism to connect signals between peripherals. However, when linking I/O Ports via the ELC the relevant I/O pins need to be configured as inputs or outputs.

Build Time Configurations for r_elc

The following build time configurations are defined in `fsp_cfg/r_elc_cfg.h`:

Configuration	Options	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	If selected code for parameter checking is included in the build.

Configurations for ELC Driver on r_elc

This module can be added to the Threads tab from New -> Driver -> System -> ELC Driver on r_elc:

4.2.15 Ethernet (r_ether)

Modules

Functions

`fsp_err_t R_ETHER_Open(ether_ctrl_t *const p_ctrl, ether_cfg_t const *const p_cfg)`

After ETHERC, EDMAC and PHY-LSI are reset in software, an auto negotiation of PHY-LSI is begun. Afterwards, the link signal change interrupt is permitted. Implements `ether_api_t::open`. [More...](#)

`fsp_err_t` [R_ETHER_Close](#) (`ether_ctrl_t *const p_ctrl`)

Disables interrupts. Removes power and releases hardware lock. Implements [ether_api_t::close](#). [More...](#)

`fsp_err_t` [R_ETHER_Read](#) (`ether_ctrl_t *const p_ctrl`, `void **const pp_buffer`, `uint32_t *const length_bytes`)

Receive Ethernet frame. Receives data to the location specified by the pointer to the receive buffer, using non-zero-copy communication. Implements [ether_api_t::read](#). [More...](#)

`fsp_err_t` [R_ETHER_BufferRelease](#) (`ether_ctrl_t *const p_ctrl`)

Release the receive buffer. Implements [ether_api_t::BufferRelease](#). [More...](#)

`fsp_err_t` [R_ETHER_Write](#) (`ether_ctrl_t *const p_ctrl`, `void *const p_buffer`, `uint32_t const frame_length`)

Transmit Ethernet frame. Transmits data from the location specified by the pointer to the transmit buffer, with the data size equal to the specified frame length, using non-zero-copy communication. Implements [ether_api_t::write](#). [More...](#)

`fsp_err_t` [R_ETHER_LinkProcess](#) (`ether_ctrl_t *const p_ctrl`)

The Link up processing, the Link down processing, and the magic packet detection processing are executed. Implements [ether_api_t::linkProcess](#). [More...](#)

`fsp_err_t` [R_ETHER_WakeOnLANEnable](#) (`ether_ctrl_t *const p_ctrl`)

The setting of ETHERC is changed from a usual sending and receiving mode to the magic packet detection mode. Implements [ether_api_t::wakeOnLANEnable](#). [More...](#)

`fsp_err_t` [R_ETHER_VersionGet](#) (`fsp_version_t *const p_version`)

Provides API and code version in the user provided pointer. Implements [ether_api_t::versionGet](#). [More...](#)

Detailed Description

Driver for the Ethernet peripheral on RA MCUs. This module implements the [Ethernet Interface](#).

Overview

This module performs Ethernet frame transmission and reception using an Ethernet controller and an Ethernet DMA controller.

Features

The Ethernet module supports the following features:

- Transmit/receive processing(Zero-copy and Non-Zero-copy)
- Callback function with returned event code
- Magic packet detection mode support
- Auto negotiation support
- Flow control support
- Multicast filtering support
- Broadcast filtering support
- Promiscuous mode support

Target Devices

The Ethernet module supports the following devices.

- RA6M3
- RA6M2

Ethernet Frame Format

The Ethernet module supports the Ethernet II/IEEE 802.3 frame format.

Frame Format for Data Transmission and Reception

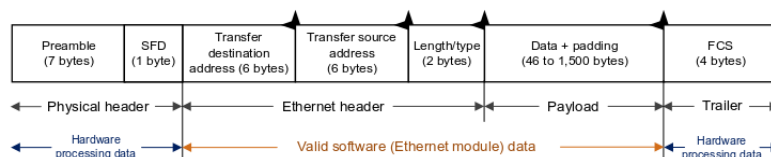


Figure 96: Frame Format Image

The preamble and SFD signal the start of an Ethernet frame. The FCS contains the CRC of the Ethernet frame and is calculated on the transmitting side. When data is received the CRC value of the frame is calculated in hardware, and the Ethernet frame is discarded if the values do not match. When the hardware determines that the data is normal, the valid range of receive data is: (transmission destination address) + (transmission source address) + (length/type) + (data).

PAUSE Frame Format

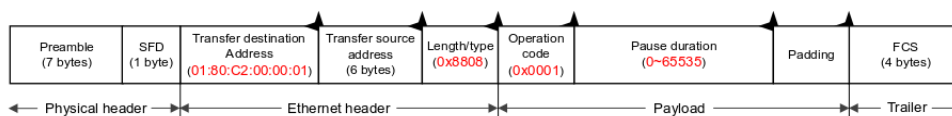


Figure 97: Pause Frame Format Image

The transmission destination address is specified as 01:80:C2:00:00:01 (a multicast address reserved for PAUSE frames). At the start of the payload the length/type is specified as 0x8808 and the operation code as 0x0001. The pause duration in the payload is specified by the value of the automatic PAUSE (AP) bits in the automatic PAUSE frame setting register (APR), or the manual PAUSE time setting (MP) bits in the manual PAUSE frame setting register (MPR).

Magic Packet Frame Format

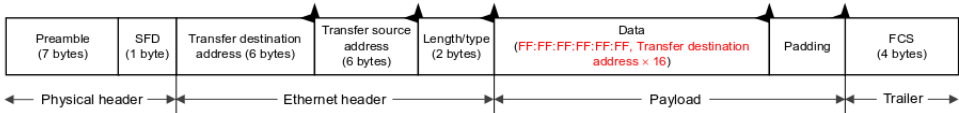


Figure 98: Magic Packet Frame Format Image

In a Magic Packet, the value FF:FF:FF:FF:FF:FF followed by the transmission destination address repeated 16 times is inserted somewhere in the Ethernet frame data.

Configuration

Build Time Configurations for r_ether

The following build time configurations are defined in fsp_cfg/driver/r_ether_cfg.h:

Configuration	Options	Description
Parameter Checking	- Default (BSP) - Enabled - Disabled	If selected code for parameter checking is included in the build.
The polarity of the link signal output by the PHY-LSI	- Fall -> Rise - Rise -> Fall	Specify the polarity of the link signal output by the PHY-LSI. When 0 is specified, link-up and link-down correspond respectively to the fall and rise of the LINKSTA signal. When 1 is specified, link-up and link-down correspond respectively to the rise and fall of the LINKSTA signal.
The link status is detected by LINKSTA signal	- Unused - Used	Use LINKSTA signal for detect link status changes 0 = unused (use PHY-LSI status register) 1 = use (use LINKSTA signal)

Configurations for Ethernet Driver on r_ether

This module can be added to the Threads tab from New -> Driver -> Network -> Ethernet Driver on r_ether:

4.2.16 Ethernet PHY (r_ether_phy)

Modules

Functions

`fsp_err_t` [R_ETHER_PHY_Open](#) ([ether_phy_ctrl_t](#) *const p_ctrl, [ether_phy_cfg_t](#) const *const p_cfg)

Resets Ethernet PHY device. Implements [ether_phy_api_t::open](#). *. [More...](#)

`fsp_err_t` [R_ETHER_PHY_Close](#) ([ether_phy_ctrl_t](#) *const p_ctrl)

Close Ethernet PHY device. Implements [ether_phy_api_t::close](#). [More...](#)

`fsp_err_t` [R_ETHER_PHY_StartAutoNegotiate](#) ([ether_phy_ctrl_t](#) *const p_ctrl)

Starts auto-negotiate. Implements [ether_phy_api_t::startAutoNegotiate](#). [More...](#)

`fsp_err_t` [R_ETHER_PHY_LinkPartnerAbilityGet](#) ([ether_phy_ctrl_t](#) *const p_ctrl, [uint32_t](#) *const p_line_speed_duplex, [uint32_t](#) *const p_local_pause, [uint32_t](#) *const p_partner_pause)

Reports the other side's physical capability. Implements [ether_phy_api_t::linkPartnerAbilityGet](#). [More...](#)

`fsp_err_t` [R_ETHER_PHY_LinkStatusGet](#) ([ether_phy_ctrl_t](#) *const p_ctrl)

Returns the status of the physical link. Implements [ether_phy_api_t::linkStatusGet](#). [More...](#)

`fsp_err_t` [R_ETHER_PHY_VersionGet](#) ([fsp_version_t](#) *const p_version)

Provides API and code version in the user provided pointer. Implements [ether_phy_api_t::versionGet](#). [More...](#)

Detailed Description

The Ethernet PHY module (r_ether_phy) provides an API for standard Ethernet PHY communications applications and uses the ETHERC peripherals. It implements the [Ethernet PHY Interface](#).

Overview

The Ethernet PHY module provides Ethernet phy functionality.

Features

The Ethernet PHY module supports the following features:

- Auto negotiation support
- Flow control support
- Link status check support

Target Devices

The Ethernet module supports the following devices.

- RA6M3
- RA6M2

Accessing the MII and RMII Registers

Use the PIR register to access the MII and RMII registers in the PHY-LSI. Serial data in the MII and RMII management frame format is transmitted and received through the ET0_MDC and ET0_MDIO pins controlled by software.

MI and RMII management frame format

Table lists the MII and RMII management frame formats.

Access type		MII and RMII management frame							
	Item	PRE	ST	OP	PHYAD	REGAD	TA	DATA	IDLE
	Number of bits	32	2	2	5	5	2	16	1
Read		1...1	01	10	00001	RRRRR	Z0	DDDDD DDDDD DDDDD D	Z
Write		1...1	01	01	00001	RRRRR	10	DDDDD DDDDD DDDDD D	Z

Note

- *PRE (preamble):* Send 32 consecutive 1s.
- *ST (start of frame):* Send 01b.
- *OP (operation code):* Send 10b for read or 01b for write.
- *PHYAD (PHY address):* Up to 32 PHY-LSIs can be connected to one MAC. PHY-LSIs are selected with these 5 bits. When the
- *PHY-LSI address is 1, send 00001b.*
- *REGAD (register address):* One register is selected from up to 32 registers in the PHY-LSI. When the register address is 1, send 00001b.
- *TA (turnaround):* Use 2-bit turnaround time to avoid contention between the register address and data during a read operation.
Send 10b during a write operation. Release the bus for 1 bit during a read operation (Z is output).
(This is indicated as Z0 because 0 is output from the PHY-LSI on the next clock cycle.)
- *DATA (data):* 16-bit data. Sequentially send or receive starting from the MSB.
- *IDLE (IDLE condition):* Wait time before inputting the next MII or RMII management format. Release the bus during a write

operation (Z is output). No control is required, because a bus was already released during a read operation.

Configuration

Build Time Configurations for r_ether_phy

The following build time configurations are defined in fsp_cfg/driver/r_ether_phy_cfg.h:

Configuration	Options	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	If selected code for parameter checking is included in the build.
Select PHY	• Default • KSZ8091RNB • KSZ8041 • DP83620	Select PHY chip to use.

Configurations for Ethernet Driver on r_ether_phy

This module can be added to the Threads tab from New -> Driver -> Network -> Ethernet Driver on r_ether_phy:

4.2.17 High-Performance Flash Driver (r_flash_hp)

Modules

Functions

fsp_err_t	R_FLASH_HP_Open (flash_ctrl_t *const p_api_ctrl, flash_cfg_t const *const p_cfg)
fsp_err_t	R_FLASH_HP_Write (flash_ctrl_t *const p_api_ctrl, uint32_t const src_address, uint32_t flash_address, uint32_t const num_bytes)
fsp_err_t	R_FLASH_HP_Erase (flash_ctrl_t *const p_api_ctrl, uint32_t const address, uint32_t const num_blocks)
fsp_err_t	R_FLASH_HP_BlankCheck (flash_ctrl_t *const p_api_ctrl, uint32_t const address, uint32_t num_bytes, flash_result_t *blank_check_result)
fsp_err_t	R_FLASH_HP_Close (flash_ctrl_t *const p_api_ctrl)
fsp_err_t	R_FLASH_HP_StatusGet (flash_ctrl_t *const p_api_ctrl, flash_status_t *const p_status)
fsp_err_t	R_FLASH_HP_AccessWindowSet (flash_ctrl_t *const p_api_ctrl, uint32_t const start_addr, uint32_t const end_addr)

```
fsp_err_t R_FLASH_HP_AccessWindowClear (flash_ctrl_t *const p_api_ctrl)
```

```
fsp_err_t R_FLASH_HP_IdCodeSet (flash_ctrl_t *const p_api_ctrl, uint8_t const
*const p_id_code, flash_id_code_mode_t mode)
```

```
fsp_err_t R_FLASH_HP_Reset (flash_ctrl_t *const p_api_ctrl)
```

```
fsp_err_t R_FLASH_HP_UpdateFlashClockFreq (flash_ctrl_t *const p_api_ctrl)
```

```
fsp_err_t R_FLASH_HP_StartUpAreaSelect (flash_ctrl_t *const p_api_ctrl,
flash_startup_area_swap_t swap_type, bool is_temporary)
```

```
fsp_err_t R_FLASH_HP_VersionGet (fsp_version_t *const p_version)
```

```
fsp_err_t R_FLASH_HP_InfoGet (flash_ctrl_t *const p_api_ctrl, flash_info_t
*const p_info)
```

Detailed Description

Driver for the flash memory on RA high-performance MCUs. This module implements the [Flash Interface](#).

Overview

The Flash HAL module APIs allow an application to write, erase and blank check both the data and ROM flash areas that reside within the MCU. The amount of flash memory available varies across MCU parts.

Features

The R_FLASH_HP module has the following key features:

- Blocking and non-blocking erasing, writing and blank-checking of data flash.
- Blocking erasing, writing and blank-checking of code flash.
- Callback functions for completion of non-blocking data-flash operations.
- Access window (write protection) for ROM Flash, allowing only specified areas of code flash to be erased or written.
- Boot block-swapping.
- ID code programming support.

Configuration

Build Time Configurations for r_flash_hp

The following build time configurations are defined in fsp_cfg/r_flash_hp_cfg.h:

Configuration	Options	Description
Parameter Checking	• Default (BSP) • Enabled	If selected code for parameter checking is included in the

	• Disabled	build.
Code Flash Programming Enable	• Enabled • Disabled	Controls whether or not code-flash programming is enabled. Disabling reduces the amount of ROM and RAM used by the API.
Data Flash Programming Enable	• Enabled • Disabled	Controls whether or not data-flash programming is enabled. Disabling reduces the amount of ROM used by the API.

Configurations for Flash Driver on r_flash_hp

This module can be added to the Threads tab from New -> Driver -> Storage -> Flash Driver on r_flash_hp:

4.2.18 Low-Power Flash Driver (r_flash_lp)

Modules

Functions

fsp_err_t	R_FLASH_LP_Open (flash_ctrl_t *const p_api_ctrl, flash_cfg_t const *const p_cfg)
fsp_err_t	R_FLASH_LP_Write (flash_ctrl_t *const p_api_ctrl, uint32_t const src_address, uint32_t flash_address, uint32_t const num_bytes)
fsp_err_t	R_FLASH_LP_Erase (flash_ctrl_t *const p_api_ctrl, uint32_t const address, uint32_t const num_blocks)
fsp_err_t	R_FLASH_LP_BlankCheck (flash_ctrl_t *const p_api_ctrl, uint32_t const address, uint32_t num_bytes, flash_result_t *blank_check_result)
fsp_err_t	R_FLASH_LP_Close (flash_ctrl_t *const p_api_ctrl)
fsp_err_t	R_FLASH_LP_StatusGet (flash_ctrl_t *const p_api_ctrl, flash_status_t *const p_status)
fsp_err_t	R_FLASH_LP_AccessWindowSet (flash_ctrl_t *const p_api_ctrl, uint32_t const start_addr, uint32_t const end_addr)
fsp_err_t	R_FLASH_LP_AccessWindowClear (flash_ctrl_t *const p_api_ctrl)
fsp_err_t	R_FLASH_LP_IdCodeSet (flash_ctrl_t *const p_api_ctrl, uint8_t const *const p_id_code, flash_id_code_mode_t mode)
fsp_err_t	R_FLASH_LP_Reset (flash_ctrl_t *const p_api_ctrl)


```
fsp_err_t R_FLASH_LP_StartUpAreaSelect (flash_ctrl_t *const p_api_ctrl,
                                         flash_startup_area_swap_t swap_type, bool is_temporary)
```

```
fsp_err_t R_FLASH_LP_UpdateFlashClockFreq (flash_ctrl_t *const p_api_ctrl)
```

```
fsp_err_t R_FLASH_LP_VersionGet (fsp_version_t *const p_version)
```

```
fsp_err_t R_FLASH_LP_InfoGet (flash_ctrl_t *const p_api_ctrl, flash_info_t *const
                               p_info)
```

Detailed Description

Driver for the flash memory on RA low-power MCUs. This module implements the [Flash Interface](#).

Overview

The Flash HAL module APIs allow an application to write, erase and blank check both the data and code flash areas that reside within the MCU. The amount of flash memory available varies across MCU parts.

Features

The Low-Power Flash HAL module has the following key features:

- Blocking and non-blocking erasing, writing and blank-checking of data flash.
- Blocking erasing, writing and blank checking of code flash.
- Callback functions for completion of non-blocking data flash operations.
- Access window (write protection) for code flash, allowing only specified areas of code flash to be erased or written.
- Boot block-swapping.
- ID code programming support.

Configuration

Build Time Configurations for r_flash_lp

The following build time configurations are defined in fsp_cfg/r_flash_lp_cfg.h:

Configuration	Options	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	If selected code for parameter checking is included in the build.
Code Flash Programming Enable	• Enabled • Disabled	Controls whether or not code-flash programming is enabled. Disabling reduces the amount of ROM and RAM used by the API.
Data Flash Programming Enable	• Enabled	Controls whether or not data-

- Disabled

flash programming is enabled. Disabling reduces the amount of ROM used by the API.

Configurations for Flash Driver on r_flash_lp

This module can be added to the Threads tab from New -> Driver -> Storage -> Flash Driver on r_flash_lp:

4.2.19 Graphics LCD Controller (r_glcdc)

Modules

Functions

`fsp_err_t R_GLCDC_Open (display_ctrl_t *const p_api_ctrl, display_cfg_t const *const p_cfg)`

Open GLCDC module. [More...](#)

`fsp_err_t R_GLCDC_Close (display_ctrl_t *const p_api_ctrl)`

Close GLCDC module. [More...](#)

`fsp_err_t R_GLCDC_Start (display_ctrl_t *const p_api_ctrl)`

Start GLCDC module. [More...](#)

`fsp_err_t R_GLCDC_Stop (display_ctrl_t *const p_api_ctrl)`

Stop GLCDC module. [More...](#)

`fsp_err_t R_GLCDC_LayerChange (display_ctrl_t const *const p_api_ctrl, display_runtime_cfg_t const *const p_cfg, display_frame_layer_t layer)`

Change layer parameters of GLCDC module at runtime. [More...](#)

`fsp_err_t R_GLCDC_BufferChange (display_ctrl_t const *const p_api_ctrl, uint8_t *const framebuffer, display_frame_layer_t layer)`

Change the framebuffer pointer for a layer. [More...](#)

`fsp_err_t R_GLCDC_ColorCorrection (display_ctrl_t const *const p_api_ctrl, display_correction_t const *const p_correction)`

Perform color correction through the GLCDC module. [More...](#)

```
fsp_err_t R_GLCDC_ClutUpdate (display_ctrl_t const *const p_api_ctrl,
                             display_clut_cfg_t const *const p_clut_cfg, display_frame_layer_t
                             layer)
```

Update a color look-up table (CLUT) in the GLCDC module. [More...](#)

```
fsp_err_t R_GLCDC_StatusGet (display_ctrl_t const *const p_api_ctrl,
                             display_status_t *const p_status)
```

Get status of GLCDC module. [More...](#)

```
fsp_err_t R_GLCDC_VersionGet (fsp_version_t *p_version)
```

Get version of R_GLCDC module. [More...](#)

Detailed Description

Driver for the GLCDC peripheral on RA MCUs. This module implements the [Display Interface](#).

Overview

The GLCDC is a multi-stage graphics output peripheral designed to automatically generate timing and data signals for LCD panels. As part of its internal pipeline the two internal graphics layers can be repositioned, alpha blended, color corrected, dithered and converted to and from a wide variety of pixel formats.

Features

The following features are available:

Feature	Options
Input color formats	ARGB8888, ARGB4444, ARGB1555, RGB888 (32-bit), RGB565, CLUT 8bpp, CLUT 4bpp, CLUT 1bpp
Output color formats	RGB888, RGB666, RGB565, Serial RGB888 (8-bit parallel)
Correction processes	Alpha blending, positioning, brightness and contrast, gamma correction, dithering
Timing signals	Dot clock, Vsync, Hsync, Vertical and horizontal data enable (DE)
Maximum resolution	Up to 1020 x 1008 pixels (dependent on sync signal width)
Maximum dot clock	60MHz for serial RGB mode, 54MHz otherwise

Internal clock divisors	1-9, 12, 16, 24, 32
Interrupts	Vsync (line detect), Underflow
Other functions	Byte-order and endianness control, line repeat function

Configuration

Build Time Configurations for r_glcdc

The following build time configurations are defined in fsp_cfg/r_glcdc_cfg.h:

Configuration	Options	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	If selected code for parameter checking is included in the build.
Color Correction	• On • Off	If selected code to adjust brightness, contrast and gamma settings is included in the build. When disabled all color correction configuration options are ignored.

4.2.20 General PWM Timer (r_gpt)

Modules

Functions

fsp_err_t	R_GPT_Stop (timer_ctrl_t *const p_ctrl)
fsp_err_t	R_GPT_Start (timer_ctrl_t *const p_ctrl)
fsp_err_t	R_GPT_Reset (timer_ctrl_t *const p_ctrl)
fsp_err_t	R_GPT_Enable (timer_ctrl_t *const p_ctrl)
fsp_err_t	R_GPT_Disable (timer_ctrl_t *const p_ctrl)
fsp_err_t	R_GPT_PeriodSet (timer_ctrl_t *const p_ctrl, uint32_t const period_counts)
fsp_err_t	R_GPT_DutyCycleSet (timer_ctrl_t *const p_ctrl, uint32_t const duty_cycle_counts, uint32_t const pin)
fsp_err_t	R_GPT_InfoGet (timer_ctrl_t *const p_ctrl, timer_info_t *const p_info)

fsp_err_t R_GPT_StatusGet (timer_ctrl_t *const p_ctrl, timer_status_t *const p_status)

fsp_err_t R_GPT_Close (timer_ctrl_t *const p_ctrl)

fsp_err_t R_GPT_VersionGet (fsp_version_t *const p_version)

Detailed Description

Driver for the GPT32 and GPT16 peripherals on RA MCUs. This module implements the [Timer Interface](#).

Overview

The GPT module can be used to count events, measure external input signals, generate a periodic interrupt, or output a periodic or PWM signal to a GTIOC pin.

This module supports the GPT peripherals GPT32EH, GPT32E, GPT32, and GPT16. GPT16 is a 16-bit timer. The other peripherals (GPT32EH, GPT32E, and GPT32) are 32-bit timers. The 32-bit timers are all treated the same in this module from the API perspective.

Features

The GPT module has the following features:

- Supports periodic mode, one-shot mode, and PWM mode.
- Supports count source of PCLK, GTETRG pins, GTIOC pins, or ELC events.
- Supports debounce filter on GTIOC pins.
- Signal can be output to a pin.
- Configurable period (counts per timer cycle).
- Configurable duty cycle in PWM mode.
- Supports runtime reconfiguration of period.
- Supports runtime reconfiguration of duty cycle in PWM mode.
- APIs are provided to start, stop, and reset the counter.
- APIs are provided to get the current period, source clock frequency, and count direction.
- APIs are provided to get the current timer status and counter value.
- Supports start, stop, clear, count up, count down, and capture by external sources from GTETRG pins, GTIOC pins, or ELC events.

Selecting a Timer

RA MCUs have two timer peripherals: the General PWM Timer (GPT) and the Asynchronous General Purpose Timer (AGT). When selecting between them, consider these factors:

	GPT	AGT
Low Power Modes	The GPT can operate in sleep mode.	The AGT can operate in all low power modes.
Available Channels	The number of GPT channels is device specific. All currently supported MCUs have at least 7 GPT channels.	All MCUs have 2 AGT channels.

Timer Resolution	All MCUs have at least one 32-bit GPT timer.	The AGT timers are 16-bit timers.
Clock Source	The GPT runs off PCLKD with a configurable divider up to 1024. It can also be configured to count ELC events or external pulses.	The AGT runs off PCLKB, LOCO, or subclock.

Configuration

Build Time Configurations for r_gpt

The following build time configurations are defined in fsp_cfg/r_gpt_cfg.h:

Configuration	Options	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	If selected code for parameter checking is included in the build.
Pin Output Support	• Disabled • Enabled	If selected code for outputting a waveform to a pin is included in the build.

Configurations for Timer Driver on r_gpt

This module can be added to the Threads tab from New -> Driver -> Timers -> Timer Driver on r_gpt:

4.2.21 Interrupt Controller Unit (r_icu)

Modules

Functions

```
fsp_err_t R_ICU_ExtIrqOpen (external_irq_ctrl_t *const p_api_ctrl,
                           external_irq_cfg_t const *const p_cfg)
```

```
fsp_err_t R_ICU_ExtIrqEnable (external_irq_ctrl_t *const p_api_ctrl)
```

```
fsp_err_t R_ICU_ExtIrqDisable (external_irq_ctrl_t *const p_api_ctrl)
```

```
fsp_err_t R_ICU_ExtIrqVersionGet (fsp_version_t *const p_version)
```

```
fsp_err_t R_ICU_ExtIrqClose (external_irq_ctrl_t *const p_api_ctrl)
```

Detailed Description

Driver for the ICU peripheral on RA MCUs. This module implements the [External IRQ Interface](#).

Overview

The Interrupt Controller Unit (ICU) controls which event signals are linked to the NVIC, DTC, and DMAC modules. R_ICU software module only implements the [External IRQ Interface](#). The external_irq interface is for configuring interrupts to fire when a trigger condition is detected on an external IRQ pin.

Features

- Supports configuring interrupts for IRQ pins on the target MCUs
 - Enabling and disabling interrupt generation.
 - Configuring interrupt trigger on rising edge, falling edge, both edges, or low level signal.
 - Enabling and disabling the IRQ noise filter.
- Supports configuring a user callback function, which will be invoked by the HAL module when an external pin interrupt is generated.

Configuration

Build Time Configurations for r_icu

The following build time configurations are defined in fsp_cfg/r_icu_cfg.h:

Configuration	Options	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	If selected code for parameter checking is included in the build.

Configurations for External IRQ Driver on r_icu

This module can be added to the Threads tab from New -> Driver -> Input -> External IRQ Driver on r_icu:

4.2.22 I2C Master on IIC (r_iic_master)

Modules

Functions

`fsp_err_t` [R_IIC_MASTER_Open](#) (`i2c_master_ctrl_t *const p_api_ctrl`, `i2c_master_cfg_t const *const p_cfg`)

`fsp_err_t` [R_IIC_MASTER_Read](#) (`i2c_master_ctrl_t *const p_api_ctrl`, `uint8_t *const p_dest`, `uint32_t const bytes`, `bool const restart`)

```
fsp_err_t R_IIC_MASTER_Write (i2c_master_ctrl_t *const p_api_ctrl, uint8_t
*const p_src, uint32_t const bytes, bool const restart)
```

```
fsp_err_t R_IIC_MASTER_Abort (i2c_master_ctrl_t *const p_api_ctrl)
```

```
fsp_err_t R_IIC_MASTER_SlaveAddressSet (i2c_master_ctrl_t *const p_api_ctrl,
uint32_t const slave, i2c_master_addr_mode_t const addr_mode)
```

```
fsp_err_t R_IIC_MASTER_Close (i2c_master_ctrl_t *const p_api_ctrl)
```

```
fsp_err_t R_IIC_MASTER_VersionGet (fsp_version_t *const p_version)
```

Detailed Description

Driver for the IIC peripheral on RA MCUs. This module implements the [I2C Master Interface](#).

Overview

The I2C master on IIC HAL module supports transactions with an I2C Slave device. Callbacks must be provided which would be invoked when a transmission or receive has been completed. The callback arguments will contain information about the transaction status, bytes transferred and a pointer to the user defined context.

Features

- Supports multiple transmission rates
 - Standard Mode Support with up to 100-kHz transaction rate.
 - Fast Mode Support with up to 400-kHz transaction rate.
 - Fast Mode Plus Support with up to 1-MHz transaction rate.
- I2C Master Read from a slave device.
- I2C Master Write to a slave device.
- Abort any in-progress transactions.
- Set the address of the slave device.
- Non-blocking behavior is achieved by the use of callbacks.
- Additional build-time features
 - Optional (build time) DTC support for read and write respectively.
 - Optional (build time) support for 10-bit slave addressing.

Configuration

Build Time Configurations for r_iic_master

The following build time configurations are defined in fsp_cfg/r_iic_master_cfg.h:

Configuration	Options	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	If selected code for parameter checking is included in the build.
DTC on Transmission and	• Enabled	If enabled, DTC instances will

Reception	• Disabled	be included in the build for both transmission and reception.
10-bit slave addressing	• Enabled • Disabled	If enabled, the driver will support 10-bit slave addressing mode along with the default 7-bit slave addressing mode.

Configurations for I2C Master Driver on r_iic_master

This module can be added to the Threads tab from New -> Driver -> Connectivity -> I2C Master Driver on r_iic_master:

4.2.23 I2C Slave on IIC (r_iic_slave)

Modules

Functions

`fsp_err_t` `R_IIC_SLAVE_Open` (`i2c_slave_ctrl_t` *const `p_api_ctrl`, `i2c_slave_cfg_t` const *const `p_cfg`)

`fsp_err_t` `R_IIC_SLAVE_Read` (`i2c_slave_ctrl_t` *const `p_api_ctrl`, `uint8_t` *const `p_dest`, `uint32_t` const `bytes`)

`fsp_err_t` `R_IIC_SLAVE_Write` (`i2c_slave_ctrl_t` *const `p_api_ctrl`, `uint8_t` *const `p_src`, `uint32_t` const `bytes`)

`fsp_err_t` `R_IIC_SLAVE_Close` (`i2c_slave_ctrl_t` *const `p_api_ctrl`)

`fsp_err_t` `R_IIC_SLAVE_VersionGet` (`fsp_version_t` *const `p_version`)

Detailed Description

Driver for the IIC peripheral on RA MCUs. This module implements the [I2C Slave Interface](#).

Overview

Features

- Supports multiple transmission rates
 - Standard Mode Support with up to 100-kHz transaction rate.
 - Fast Mode Support with up to 400-kHz transaction rate.
 - Fast Mode Plus Support with up to 1-MHz transaction rate.
- Reads data written by master device.
- Write data which is read by master device.
- Can be assigned a 10-bit address.
- Clock stretching is supported and can be implemented via callbacks.
- Provides Transmission/Reception transaction size in the callback.
- I2C Slave can notify the following events via callbacks: Transmission/Reception Request, Transmission/Reception Request for more data, Transmission/Reception Completion, Error

Condition.

Configuration

Build Time Configurations for r_iic_slave

The following build time configurations are defined in fsp_cfg/r_iic_slave_cfg.h:

Configuration	Options	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	If selected code for parameter checking is included in the build.

Configurations for I2C Slave Driver on r_iic_slave

This module can be added to the Threads tab from New -> Driver -> Connectivity -> I2C Slave Driver on r_iic_slave:

4.2.24 I/O Ports (r_ioport)

Modules

Functions

fsp_err_t	R_IOPORT_Open (ioport_ctrl_t *const p_ctrl, const ioport_cfg_t *p_cfg)
fsp_err_t	R_IOPORT_Close (ioport_ctrl_t *const p_ctrl)
fsp_err_t	R_IOPORT_PinsCfg (ioport_ctrl_t *const p_ctrl, const ioport_cfg_t *p_cfg)
fsp_err_t	R_IOPORT_PinCfg (ioport_ctrl_t *const p_ctrl, bsp_io_port_pin_t pin, uint32_t cfg)
fsp_err_t	R_IOPORT_PinEventInputRead (ioport_ctrl_t *const p_ctrl, bsp_io_port_pin_t pin, bsp_io_level_t *p_pin_event)
fsp_err_t	R_IOPORT_PinEventOutputWrite (ioport_ctrl_t *const p_ctrl, bsp_io_port_pin_t pin, bsp_io_level_t pin_value)
fsp_err_t	R_IOPORT_PinRead (ioport_ctrl_t *const p_ctrl, bsp_io_port_pin_t pin, bsp_io_level_t *p_pin_value)
fsp_err_t	R_IOPORT_PinWrite (ioport_ctrl_t *const p_ctrl, bsp_io_port_pin_t pin, bsp_io_level_t level)
fsp_err_t	R_IOPORT_PortDirectionSet (ioport_ctrl_t *const p_ctrl, bsp_io_port_t port, ioport_size_t direction_values, ioport_size_t mask)

```
fsp_err_t R_IOPORT_PortEventInputRead (ioport_ctrl_t *const p_ctrl,
                                         bsp_io_port_t port, ioport_size_t *event_data)
```

```
fsp_err_t R_IOPORT_PortEventOutputWrite (ioport_ctrl_t *const p_ctrl,
                                           bsp_io_port_t port, ioport_size_t event_data, ioport_size_t
                                           mask_value)
```

```
fsp_err_t R_IOPORT_PortRead (ioport_ctrl_t *const p_ctrl, bsp_io_port_t port,
                              ioport_size_t *p_port_value)
```

```
fsp_err_t R_IOPORT_PortWrite (ioport_ctrl_t *const p_ctrl, bsp_io_port_t port,
                               ioport_size_t value, ioport_size_t mask)
```

```
fsp_err_t R_IOPORT_EthernetModeCfg (ioport_ctrl_t *const p_ctrl,
                                      ioport_ethernet_channel_t channel, ioport_ethernet_mode_t mode)
```

```
fsp_err_t R_IOPORT_VersionGet (fsp_version_t *p_data)
```

Detailed Description

Driver for the I/O Ports peripheral on RA MCUs. This module implements the [I/O Port Interface](#).

Overview

The I/O port pins operate as general I/O port pins, I/O pins for peripheral modules, interrupt input pins, analog I/O, port group function for the ELC, or bus control pins.

Features

The I/O PORT HAL module can not only configure the direction of the pin/pins but also other options provided as follows:

- Pull-up
- NMOS/PMOS
- Drive strength
- Event edge trigger (falling, rising or both)
- Whether the pin is to be used as an IRQ pin
- Whether the pin is to be used as an analog pin
- Whether the pin is to be used as a peripheral pin and which peripheral

The module also provides the following functionality:

- Sets event output data
- Reads event input data

Configuration

The I/O PORT HAL module must be configured by the user for the desired operation. The operating state of an I/O pin can be set via the RA configurator. When the RA project is built, a pin configuration file is created. When the application runs, the BSP will configure the MCU IO port accordingly, using the same API functions mentioned in this document.

Build Time Configurations for r_ioport

The following build time configurations are defined in fsp_cfg/r_ioport_cfg.h:

Configuration	Options	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	If selected code for parameter checking is included in the build.

Configurations for I/O Port Driver on r_ioport

This module can be added to the Threads tab from New -> Driver -> System -> I/O Port Driver on r_ioport:

4.2.25 Independent Watchdog Timer (r_iwdt)

Modules

Functions

fsp_err_t	R_IWDT_Refresh (wdt_ctrl_t *const p_api_ctrl)
fsp_err_t	R_IWDT_Open (wdt_ctrl_t *const p_api_ctrl, wdt_cfg_t const *const p_cfg)
fsp_err_t	R_IWDT_StatusClear (wdt_ctrl_t *const p_api_ctrl, const wdt_status_t status)
fsp_err_t	R_IWDT_StatusGet (wdt_ctrl_t *const p_api_ctrl, wdt_status_t *const p_status)
fsp_err_t	R_IWDT_CounterGet (wdt_ctrl_t *const p_api_ctrl, uint32_t *const p_count)
fsp_err_t	R_IWDT_TimeoutGet (wdt_ctrl_t *const p_api_ctrl, wdt_timeout_values_t *const p_timeout)
fsp_err_t	R_IWDT_VersionGet (fsp_version_t *const p_data)

Detailed Description

Driver for the IWDT peripheral on RA MCUs. This module implements the [WDT Interface](#).

Overview

The independent watchdog timer is used to recover from unexpected errors in an application. The timer must be refreshed periodically in the permitted count window by the application. If the count is allowed to underflow or refresh occurs outside of the valid refresh period, the IWDT resets the device

or generates an NMI.

Features

The IWDT HAL module has the following key features:

- When the IWDT underflows or is refreshed outside of the permitted refresh window, one of the following events can occur:
 - Resetting of the device
 - Generation of an NMI
- The IWDT begins counting at reset.

Selecting a Watchdog

RA MCUs have two watchdog peripherals: the watchdog timer (WDT) and the independent watchdog timer (IWDT). When selecting between them, consider these factors:

	WDT	IWDT
Start Mode	The WDT can be started from the application (register start mode) or configured by hardware to start automatically (auto start mode).	The IWDT can only be configured by hardware to start automatically.
Clock Source	The WDT runs off a peripheral clock.	The IWDT has its own clock source which improves safety.

Configuration

The IWDT can be configured using the OFS0 register settings on the BSP tab.

Build Time Configurations for r_iwdt

The following build time configurations are defined in fsp_cfg/r_iwdt_cfg.h:

Configuration	Options	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	If selected code for parameter checking is included in the build.

Configurations for Watchdog Driver on r_iwdt

This module can be added to the Threads tab from New -> Driver -> Monitoring -> Watchdog Driver on r_iwdt:

4.2.26 JPEG Codec (r_jpeg)

Modules

Functions

`fsp_err_t R_JPEG_Decode_Open (jpeg_decode_ctrl_t *const p_api_ctrl, jpeg_decode_cfg_t const *const p_cfg)`

Initialize the JPEG Codec module. [More...](#)

`fsp_err_t R_JPEG_Decode_OutputBufferSet (jpeg_decode_ctrl_t *p_api_ctrl, void *p_output_buffer, uint32_t output_buffer_size)`

Assign output buffer to the JPEG Codec for storing output data. [More...](#)

`fsp_err_t R_JPEG_Decode_LinesDecodedGet (jpeg_decode_ctrl_t *p_api_ctrl, uint32_t *p_lines)`

Returns the number of lines decoded into the output buffer. [More...](#)

`fsp_err_t R_JPEG_Decode_HorizontalStrideSet (jpeg_decode_ctrl_t *p_api_ctrl, uint32_t horizontal_stride)`

Configure horizontal stride setting. [More...](#)

`fsp_err_t R_JPEG_Decode_InputBufferSet (jpeg_decode_ctrl_t *const p_api_ctrl, void *p_data_buffer, uint32_t data_buffer_size)`

Assign input data buffer to JPEG codec for processing. [More...](#)

`fsp_err_t R_JPEG_Decode_Close (jpeg_decode_ctrl_t *p_api_ctrl)`

Cancel an outstanding JPEG codec operation and close the device. [More...](#)

`fsp_err_t R_JPEG_Decode_ImageSizeGet (jpeg_decode_ctrl_t *p_api_ctrl, uint16_t *p_horizontal_size, uint16_t *p_vertical_size)`

Obtain the size of the image. This operation is valid during JPEG decoding operation. [More...](#)

`fsp_err_t R_JPEG_Decode_StatusGet (jpeg_decode_ctrl_t *p_api_ctrl, jpeg_decode_status_t *p_status)`

Get the status of the JPEG codec. This function can also be used to poll the device. [More...](#)

`fsp_err_t R_JPEG_Decode_ImageSubsampleSet (jpeg_decode_ctrl_t *const p_api_ctrl, jpeg_decode_subsample_t horizontal_subsample, jpeg_decode_subsample_t vertical_subsample)`

Configure horizontal and vertical subsample. [More...](#)

`fsp_err_t` `R_JPEG_Declare_PixelFormatGet (jpeg_decode_ctrl_t *p_api_ctrl, jpeg_decode_color_space_t *p_color_space)`

Get the input pixel format. [More...](#)

`fsp_err_t` `R_JPEG_Declare_VersionGet (fsp_version_t *p_version)`

Get the version of the JPEG Codec driver. [More...](#)

Detailed Description

Driver for the JPEG peripheral on RA MCUs. This module implements the [JPEG Codec Interface](#).

Overview

The JPEG Codec is a hardware block providing JPEG image encode and decode functionality in parallel with other functions. Images can optionally be partially processed facilitating streaming applications.

Features

The JPEG Codec provides a number of options useful in a variety of applications:

- Basic encoding and decoding
- Streaming input and/or output
- Decoding JPEGs of unknown size
- Shrink (sub-sample) an image during the decoding process
- Rearrange input and output byte order (byte, word and/or longword swap)
- JPEG error detection

The specifications for the codec are as follows:

Feature	Options
Decompression input formats	Baseline YCbCr 4:4:4, 4:2:2, 4:2:0 and 4:1:1
Decompression output formats	ARGB8888, RGB565
Byte reordering	Byte, halfword and/or word swapping on input and output
Interrupt sources	Image size acquired, input/output data pause, decode complete, error

Configuration

Build Time Configurations for r_jpeg

The following build time configurations are defined in fsp_cfg/r_jpeg_cfg.h:

Configuration	Options	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	If selected code for parameter checking is included in the build.
Decode Support	• Enabled • Disabled	If selected code for decoding JPEG images is included in the build.
Encode Support	Disabled	If selected code for encoding JPEG images is included in the build.

4.2.27 Key Interrupt (r_kint)

Modules

Functions

`fsp_err_t R_KINT_Open (keymatrix_ctrl_t *const p_api_ctrl, keymatrix_cfg_t const *const p_cfg)`

`fsp_err_t R_KINT_Enable (keymatrix_ctrl_t *const p_api_ctrl)`

`fsp_err_t R_KINT_Disable (keymatrix_ctrl_t *const p_api_ctrl)`

`fsp_err_t R_KINT_Close (keymatrix_ctrl_t *const p_api_ctrl)`

`fsp_err_t R_KINT_VersionGet (fsp_version_t *const p_version)`

Detailed Description

Driver for the KINT peripheral on RA MCUs. This module implements the [Key Matrix Interface](#).

Overview

The KINT module configures the Key Interrupt (KINT) peripheral to detect rising or falling edges on any of the KINT channels. When such an event is detected on any of the configured pins, the module generates an interrupt.

Features

- Rising and falling edges on KINT channels
- A callback for notifying the application when edges are detected on the configured channels
- Supports a matrix keypad with edges on any two channels

Configuration

Build Time Configurations for r_kint

The following build time configurations are defined in fsp_cfg/r_kint_cfg.h:

Configuration	Options	Description
Parameter Checking Enable	• Default (BSP) • Enabled • Disabled	If selected code for parameter checking is included in the build.

Configurations for Key Matrix Driver on r_kint

This module can be added to the Threads tab from New -> Driver -> Input -> Key Matrix Driver on r_kint:

4.2.28 Low Power Modes (r_lpm)

Modules

Functions

`fsp_err_t R_LPM_Open (lpm_ctrl_t *const p_api_ctrl, lpm_cfg_t const *const p_cfg)`

`fsp_err_t R_LPM_Close (lpm_ctrl_t *const p_api_ctrl)`

`fsp_err_t R_LPM_LowPowerReconfigure (lpm_ctrl_t *const p_api_ctrl, lpm_cfg_t const *const p_cfg)`

`fsp_err_t R_LPM_LowPowerModeEnter (lpm_ctrl_t *const p_api_ctrl)`

`fsp_err_t R_LPM_VersionGet (fsp_version_t *const p_version)`

`fsp_err_t R_LPM_IoKeepClear (lpm_ctrl_t *const p_api_ctrl)`

Detailed Description

Driver for the LPM peripheral on RA MCUs. This module implements the [Low Power Modes Interface](#).

Overview

The low power modes driver is used to configure and place the device into the desired low power mode. Various sources can be configured to wake from standby, request snooze mode, end snooze mode or end deep standby mode.

Features

The LPM HAL module has the following key features:

- Supports the following low power modes:
 - Deep Software Standby mode (On supported MCUs)
 - Software Standby mode
 - Sleep mode
 - Snooze mode
- Supports reducing power consumption when in deep software standby mode through internal power supply control and by resetting the states of I/O ports.
- Supports disabling and enabling the MCU's other hardware peripherals

Configuration

Build Time Configurations for r_lpm

The following build time configurations are defined in fsp_cfg/r_lpm_cfg.h:

Configuration	Options	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	If selected code for parameter checking is included in the build.

Configurations for Low Power Modes Driver on r_lpm

This module can be added to the Threads tab from New -> Driver -> Power -> Low Power Modes Driver on r_lpm:

4.2.29 Low Voltage Detection (r_lvd)

Modules

Functions

```
fsp_err_t R_LVD_Open (lvd_ctrl_t *const p_api_ctrl, lvd_cfg_t const *const p_cfg)
```

```
fsp_err_t R_LVD_Close (lvd_ctrl_t *const p_api_ctrl)
```

```
fsp_err_t R_LVD_StatusGet (lvd_ctrl_t *const p_api_ctrl, lvd_status_t *p_lvd_status)
```

```
fsp_err_t R_LVD_StatusClear (lvd_ctrl_t *const p_api_ctrl)
```

```
fsp_err_t R_LVD_VersionGet (fsp_version_t *const p_version)
```

Detailed Description

Driver for the LVD peripheral on RA MCUs. This module implements the [Low Voltage Detection Interface](#).

Overview

The Low Voltage Detection module configures the voltage monitors to detect when V_{CC} crosses a specified threshold.

Features

The LVD HAL module supports the following functions:

- Two run-time configurable voltage monitors (Voltage Monitor 1, Voltage Monitor 2)
 - Configurable voltage threshold
 - Digital filter (Available on specific MCUs)
 - Support for both interrupt or polling
 - NMI or maskable interrupt can be configured
 - Rising, falling, or both edge event detection
 - Support for resetting the MCU when V_{CC} falls below configured threshold.

Configuration

Build Time Configurations for r_lvd

The following build time configurations are defined in fsp_cfg/r_lvd_cfg.h:

Configuration	Options	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	If selected code for parameter checking is included in the build.

Configurations for Low Voltage Detection Driver on r_lvd

This module can be added to the Threads tab from New -> Driver -> Power -> Low Voltage Detection Driver on r_lvd:

4.2.30 Realtime Clock (r_rtc)

Modules

Functions

`fsp_err_t R_RTC_Open (rtc_ctrl_t *const p_ctrl, rtc_cfg_t const *const p_cfg)`

`fsp_err_t R_RTC_Close (rtc_ctrl_t *const p_ctrl)`

`fsp_err_t R_RTC_CalendarTimeSet (rtc_ctrl_t *const p_ctrl, rtc_time_t *const p_time)`

`fsp_err_t R_RTC_CalendarTimeGet (rtc_ctrl_t *const p_ctrl, rtc_time_t *const p_time)`

```
fsp_err_t R_RTC_CalendarAlarmSet (rtc_ctrl_t *const p_ctrl, rtc_alarm_time_t
                                     *const p_alarm)
```

```
fsp_err_t R_RTC_CalendarAlarmGet (rtc_ctrl_t *const p_ctrl, rtc_alarm_time_t
                                     *const p_alarm)
```

```
fsp_err_t R_RTC_PeriodicIrqRateSet (rtc_ctrl_t *const p_ctrl,
                                     rtc_periodic_irq_select_t const rate)
```

```
fsp_err_t R_RTC_ErrorAdjustmentSet (rtc_ctrl_t *const p_ctrl,
                                     rtc_error_adjustment_cfg_t const *const err_adj_cfg)
```

```
fsp_err_t R_RTC_InfoGet (rtc_ctrl_t *const p_ctrl, rtc_info_t *const p_rtc_info)
```

```
fsp_err_t R_RTC_VersionGet (fsp_version_t *version)
```

Detailed Description

Driver for the RTC peripheral on RA MCUs. This module implements the [RTC Interface](#).

Overview

The RTC HAL module configures the RTC module and controls clock, calendar and alarm functions. A callback can be used to respond to the alarm and periodic interrupt.

Features

- RTC time and date get and set.
- RTC time and date alarm get and set.
- RTC time counter start and stop.
- RTC alarm and periodic event notification.

The RTC HAL module supports three different interrupt types:

- An alarm interrupt generated on a match of any combination of year, month, day, day of the week, hour, minute or second
- A periodic interrupt generated every 2, 1, 1/2, 1/4, 1/8, 1/16, 1/32, 1/64, 1/128, or 1/256 second(s)
- A carry interrupt is used internally when reading time from the RTC calendar to get accurate time readings.

Note

See section "23.3.5 Reading 64-Hz Counter and Time" of the RA6M3 manual R01UH0886EJ0100 for more details.

A user-defined callback function can be registered (in the [rtc_api_t::open](#) API call) and will be called from the interrupt service routine (ISR) for alarm and periodic interrupt. When called, it is passed a pointer to a structure ([rtc_callback_args_t](#)) that holds a user-defined context pointer and an indication of which type of interrupt was fired.

Date and Time validation

"Parameter Checking" needs to be enabled if date and time validation is required for `calendarTimeSet` and `calendarAlarmSet` APIs. If "Parameter Checking" is enabled, the 'day of the week' field is automatically calculated and updated by the driver for the provided date. When using the `calendarAlarmSet` API, only the fields which have their corresponding match flag set are written to the registers. Other register fields are reset to default value.

Sub-Clock error adjustment (Time Error Adjustment Function)

The time error adjustment function is used to correct errors, running fast or slow, in the time caused by variation in the precision of oscillation by the sub-clock oscillator. Because 32,768 cycles of the sub-clock oscillator constitute 1 second of operation when the sub-clock oscillator is selected, the clock runs fast if the sub-clock frequency is high and slow if the sub-clock frequency is low. The time error adjustment functions include:

- Automatic adjustment
- Adjustment by software

The error adjustment is reset every time RTC is reconfigured or time is set.

Note

RTC driver configurations do not do error adjustment internally while initializing the driver. Application must make calls to the error adjustment api's for desired adjustment. See section 26.3.8 "Time Error Adjustment Function" of the RA6M3 manual R01UH0886EJ0100) for more details on this feature

Configuration

Build Time Configurations for r_rtc

The following build time configurations are defined in `fsp_cfg/r_rtc_cfg.h`:

Configuration	Options	Description
Parameter Checking Enable	• Default (BSP) • Enabled • Disabled	If selected code for parameter checking is included in the build.

Configurations for RTC Driver on r_rtc

This module can be added to the Threads tab from New -> Driver -> Timers -> RTC Driver on r_rtc:

4.2.31 Serial Communications Interface (SCI) I2C (r_sci_i2c)

Modules

Functions

`fsp_err_t` `R_SCI_I2C_VersionGet` (`fsp_version_t` *const p_version)

`fsp_err_t` `R_SCI_I2C_Open` (`i2c_master_ctrl_t` *const p_api_ctrl, `i2c_master_cfg_t` const *const p_cfg)

`fsp_err_t` `R_SCI_I2C_Close` (`i2c_master_ctrl_t` *const p_api_ctrl)

```
fsp_err_t R_SCI_I2C_Read (i2c_master_ctrl_t *const p_api_ctrl, uint8_t *const
p_dest, uint32_t const bytes, bool const restart)
```

```
fsp_err_t R_SCI_I2C_Write (i2c_master_ctrl_t *const p_api_ctrl, uint8_t *const
p_src, uint32_t const bytes, bool const restart)
```

```
fsp_err_t R_SCI_I2C_Abort (i2c_master_ctrl_t *const p_api_ctrl)
```

```
fsp_err_t R_SCI_I2C_SlaveAddressSet (i2c_master_ctrl_t *const p_api_ctrl,
uint32_t const slave, i2c_master_addr_mode_t const addr_mode)
```

Detailed Description

Driver for the SCI peripheral on RA MCUs. This module implements the [I2C Master Interface](#).

Overview

The Simple I2C master on SCI HAL module supports transactions with an I2C Slave device. Callbacks must be provided which would be invoked when a transmission or receive has been completed. The callback arguments will contain information about the transaction status, bytes transferred and a pointer to the user defined context.

Features

- Supports multiple transmission rates
 - Standard Mode Support with up to 100 kHz transaction rate.
 - Fast Mode Support with up to 400 kHz transaction rate.
- SDA Delay in nanoseconds can be specified as a part of the configuration.
- I2C Master Read from a slave device.
- I2C Master Write to a slave device.
- Abort any in-progress transactions.
- Set the address of the slave device.
- Non-blocking behavior is achieved by the use of callbacks.
- Additional build-time features
 - Optional (build time) DTC support for read and write respectively.
 - Optional (build time) support for 10-bit slave addressing.

Configuration

Build Time Configurations for r_sci_i2c

The following build time configurations are defined in fsp_cfg/r_sci_i2c_cfg.h:

Configuration	Options	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	If selected code for parameter checking is included in the build.
DTC on Transmission and	• Enabled	If enabled, DTC instances will

Reception	• Disabled	be included in the build for both transmission and reception.
10-bit slave addressing	• Enabled • Disabled	If enabled, the driver will support 10-bit slave addressing mode along with the default 7-bit slave addressing mode.

Configurations for I2C Master Driver on r_sci_i2c

This module can be added to the Threads tab from New -> Driver -> Connectivity -> I2C Master Driver on r_sci_i2c:

4.2.32 Serial Communications Interface (SCI) SPI (r_sci_spi)

Modules

Functions

fsp_err_t	R_SCI_SPI_Open (spi_ctrl_t *p_api_ctrl, spi_cfg_t const *const p_cfg)
fsp_err_t	R_SCI_SPI_Read (spi_ctrl_t *const p_api_ctrl, void *p_dest, uint32_t const length, spi_bit_width_t const bit_width)
fsp_err_t	R_SCI_SPI_Write (spi_ctrl_t *const p_api_ctrl, void const *p_src, uint32_t const length, spi_bit_width_t const bit_width)
fsp_err_t	R_SCI_SPI_WriteRead (spi_ctrl_t *const p_api_ctrl, void const *p_src, void *p_dest, uint32_t const length, spi_bit_width_t const bit_width)
fsp_err_t	R_SCI_SPI_Close (spi_ctrl_t *const p_api_ctrl)
fsp_err_t	R_SCI_SPI_VersionGet (fsp_version_t *p_version)
fsp_err_t	R_SCI_SPI_CalculateBtrate (uint32_t bitrate, sci_spi_div_setting_t *sclk_div, bool use_mddr)

Detailed Description

Driver for the SCI peripheral on RA MCUs. This module implements the [SPI Interface](#).

Overview

Features

- Standard SPI Modes
 - Master or Slave Mode
 - Clock Polarity (CPOL)
 - CPOL=0 SCLK is low when idle
 - CPOL=1 SCLK is high when idle

- Clock Phase (CPHA)
 - CPHA=0 Data Sampled on the even edge of SCLK
 - CPHA=1 Data Sampled on the odd edge of SCLK
- MSB/LSB first
- Configurable bit rate
- DTC Support
- Callback Events
 - Transfer Complete
 - RX Overflow Error (The SCI shift register is copied to the data register before previous data was read)

Configuration

Build Time Configurations for r_sci_spi

The following build time configurations are defined in fsp_cfg/r_sci_spi_cfg.h:

Configuration	Options	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	If selected code for parameter checking is included in the build.
DTC Support	• Enabled • Disabled	If support for transferring data using the DTC will be compiled in.

Configurations for SPI Driver on r_sci_spi

This module can be added to the Threads tab from New -> Driver -> Connectivity -> SPI Driver on r_sci_spi:

4.2.33 Serial Communications Interface (SCI) UART (r_sci_uart)

Modules

Functions

`fsp_err_t` `R_SCI_UART_Open` (`uart_ctrl_t` *const p_api_ctrl, `uart_cfg_t` const *const p_cfg)

`fsp_err_t` `R_SCI_UART_Read` (`uart_ctrl_t` *const p_api_ctrl, `uint8_t` *const p_dest, `uint32_t` const bytes)

`fsp_err_t` `R_SCI_UART_Write` (`uart_ctrl_t` *const p_api_ctrl, `uint8_t` const *const p_src, `uint32_t` const bytes)

`fsp_err_t` `R_SCI_UART_BaudSet` (`uart_ctrl_t` *const p_api_ctrl, void const *const p_baud_setting)


```
fsp_err_t R_SCI_UART_InfoGet (uart_ctrl_t *const p_api_ctrl, uart_info_t *const p_info)
```

```
fsp_err_t R_SCI_UART_Close (uart_ctrl_t *const p_api_ctrl)
```

```
fsp_err_t R_SCI_UART_VersionGet (fsp_version_t *p_version)
```

```
fsp_err_t R_SCI_UART_Abort (uart_ctrl_t *const p_api_ctrl, uart_dir_t communication_to_abort)
```

```
fsp_err_t R_SCI_UART_BaudCalculate (uint32_t baudrate, bool bitrate_modulation, uint32_t baud_rate_error_x_1000, baud_setting_t *const p_baud_setting)
```

Detailed Description

Driver for the SCI peripheral on RA MCUs. This module implements the [UART Interface](#).

Overview

Features

The SCI UART module supports the following features:

- Full-duplex UART communication
- Interrupt-driven data transmission and reception
- Invoking the user-callback function with an event code (RX/TX complete, TX data empty, RX char, error, etc)
- Baud-rate change at run-time
- Bit rate modulation and noise cancellation
- RS232 CTS/RTS hardware flow control (with an associated pin)
- RS485 Half/Full Duplex flow control
- Integration with the DTC transfer module
- Abort in-progress read/write operations
- FIFO support on supported channels

Configuration

Build Time Configurations for r_sci_uart

The following build time configurations are defined in fsp_cfg/r_sci_uart_cfg.h:

Configuration	Options	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	If selected code for parameter checking is included in the build.
FIFO Support	• Enable • Disable	Enable FIFO support for the SCI_UART module.

DTC Support	• Enable • Disable	Enable DTC support for the SCI_UART module.
RS232/RS485 Flow Control Support	• Enable • Disable	Enable RS232 and RS485 flow control support using a user provided pin.

Configurations for UART Driver on r_sci_uart

This module can be added to the Threads tab from New -> Driver -> Connectivity -> UART Driver on r_sci_uart:

4.2.34 SD/MMC Host Interface (r_sdhi)

Modules

Functions

fsp_err_t	R_SDHI_Open (sdmmc_ctrl_t *const p_api_ctrl, sdmmc_cfg_t const *const p_cfg)
fsp_err_t	R_SDHI_Medialnit (sdmmc_ctrl_t *const p_api_ctrl, sdmmc_device_t *const p_device)
fsp_err_t	R_SDHI_Read (sdmmc_ctrl_t *const p_api_ctrl, uint8_t *const p_dest, uint32_t const start_sector, uint32_t const sector_count)
fsp_err_t	R_SDHI_Write (sdmmc_ctrl_t *const p_api_ctrl, uint8_t const *const p_source, uint32_t const start_sector, uint32_t const sector_count)
fsp_err_t	R_SDHI_Readlo (sdmmc_ctrl_t *const p_api_ctrl, uint8_t *const p_data, uint32_t const function, uint32_t const address)
fsp_err_t	R_SDHI_Writel0 (sdmmc_ctrl_t *const p_api_ctrl, uint8_t *const p_data, uint32_t const function, uint32_t const address, sdmmc_io_write_mode_t const read_after_write)
fsp_err_t	R_SDHI_ReadloExt (sdmmc_ctrl_t *const p_api_ctrl, uint8_t *const p_dest, uint32_t const function, uint32_t const address, uint32_t *const count, sdmmc_io_transfer_mode_t transfer_mode, sdmmc_io_address_mode_t address_mode)
fsp_err_t	R_SDHI_Writel0Ext (sdmmc_ctrl_t *const p_api_ctrl, uint8_t const *const p_source, uint32_t const function, uint32_t const address, uint32_t const count, sdmmc_io_transfer_mode_t transfer_mode, sdmmc_io_address_mode_t address_mode)
fsp_err_t	R_SDHI_IoIntEnable (sdmmc_ctrl_t *const p_api_ctrl, bool enable)
fsp_err_t	R_SDHI_StatusGet (sdmmc_ctrl_t *const p_api_ctrl, sdmmc_status_t *const p_status)

```
fsp_err_t R_SDHI_Erase (sdmmc_ctrl_t *const p_api_ctrl, uint32_t const
start_sector, uint32_t const sector_count)
```

```
fsp_err_t R_SDHI_Close (sdmmc_ctrl_t *const p_api_ctrl)
```

```
fsp_err_t R_SDHI_VersionGet (fsp_version_t *const p_version)
```

Detailed Description

Driver for the SD/MMC Host Interface (SDHI) peripheral on RA MCUs. This module implements the [SD/MMC Interface](#).

Overview

Features

- Supports the following memory devices: SDSC (SD Standard Capacity), SDHC (SD High Capacity), and SDXC (SD Extended Capacity)
 - Supports reading, writing and erasing SD memory devices
 - Supports 1-bit or 4-bit bus
 - Supports detection of device write protection (SD cards only)
- Automatically configures the clock to the maximum clock rate supported by both host (MCU) and device
- Supports hardware acceleration using DMAC or DTC
- Supports callback notification when an operation completes or an error occurs

Configuration

Build Time Configurations for r_sdhi

The following build time configurations are defined in fsp_cfg/r_sdhi_cfg.h:

Configuration	Options	Description
Parameter Checking Enable	• Default (BSP) • Enabled • Disabled	If selected code for parameter checking is included in the build.
Unaligned Access Support	• Disabled • Enabled	If enabled, code for supporting buffers that are not aligned on a 4-byte boundary is included in the build. Only disable this if all buffers passed to the driver are 4-byte aligned.

Configurations for SD/MMC Driver on r_sdhi

This module can be added to the Threads tab from New -> Driver -> Storage -> SD/MMC Driver on r_sdhi:

4.2.35 Serial Peripheral Interface (r_spi)

Modules

Functions

`fsp_err_t` `R_SPI_Open` (`spi_ctrl_t` *p_api_ctrl, `spi_cfg_t` const *const p_cfg)

`fsp_err_t` `R_SPI_Read` (`spi_ctrl_t` *const p_api_ctrl, void *p_dest, `uint32_t` const length, `spi_bit_width_t` const bit_width)

`fsp_err_t` `R_SPI_Write` (`spi_ctrl_t` *const p_api_ctrl, void const *p_src, `uint32_t` const length, `spi_bit_width_t` const bit_width)

`fsp_err_t` `R_SPI_WriteRead` (`spi_ctrl_t` *const p_api_ctrl, void const *p_src, void *p_dest, `uint32_t` const length, `spi_bit_width_t` const bit_width)

`fsp_err_t` `R_SPI_Close` (`spi_ctrl_t` *const p_api_ctrl)

`fsp_err_t` `R_SPI_VersionGet` (`fsp_version_t` *p_version)

`fsp_err_t` `R_SPI_CalculateBtrate` (`uint32_t` bitrate, `rspck_div_setting_t` *spck_div)

Detailed Description

Driver for the SPI peripheral on RA MCUs. This module implements the [SPI Interface](#).

Overview

Features

- Standard SPI Modes
 - Master or Slave Mode
 - Clock Polarity (CPOL)
 - CPOL=0 SCLK is low when idle
 - CPOL=1 SCLK is high when idle
 - Clock Phase (CPHA)
 - CPHA=0 Data Sampled on the even edge of SCLK (Master Mode Only)
 - CPHA=1 Data Sampled on the odd edge of SCLK
 - MSB/LSB first
 - 8-Bit, 16-Bit, 32-Bit data frames
 - Hardware endian swap in 16-Bit and 32-Bit mode
 - 3-Wire or 4-Wire Mode
- Configurable bitrate
- Supports Full Duplex or Transmit Only Mode
- DTC Support
- Callback Events
 - Transfer Complete
 - RX Overflow Error (The SPI shift register is copied to the data register before previous data was read)

- TX Underrun Error (No data to load into shift register for transmitting)
- Parity Error (When parity is enabled and a parity error is detected)

Configuration

Build Time Configurations for r_spi

The following build time configurations are defined in fsp_cfg/r_spi_cfg.h:

Configuration	Options	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	If selected code for parameter checking is included in the build.
Enable Support for using DTC	• Enabled • Disabled	If enabled, DTC instances will be included in the build for both transmission and reception.
Enable Transmitting from RXI Interrupt	• Enabled • Disabled	If enabled, DTC instances will be included in the build for both transmission and reception.

Configurations for SPI Driver on r_spi

This module can be added to the Threads tab from New -> Driver -> Connectivity -> SPI Driver on r_spi:

4.2.36 Serial Sound Interface (r_ssi)

Modules

Functions

`fsp_err_t` `R_SSI_Open` (`i2s_ctrl_t *const p_ctrl`, `i2s_cfg_t const *const p_cfg`)

`fsp_err_t` `R_SSI_Stop` (`i2s_ctrl_t *const p_ctrl`)

`fsp_err_t` `R_SSI_StatusGet` (`i2s_ctrl_t *const p_ctrl`, `i2s_status_t *const p_status`)

`fsp_err_t` `R_SSI_Write` (`i2s_ctrl_t *const p_ctrl`, `void const *const p_src`, `uint32_t const bytes`)

`fsp_err_t` `R_SSI_Read` (`i2s_ctrl_t *const p_ctrl`, `void *const p_dest`, `uint32_t const bytes`)

`fsp_err_t` `R_SSI_WriteRead` (`i2s_ctrl_t *const p_ctrl`, `void const *const p_src`, `void *const p_dest`, `uint32_t const bytes`)

```
fsp_err_t R_SSI_Mute (i2s_ctrl_t *const p_ctrl, i2s_mute_t const mute_enable)
```

```
fsp_err_t R_SSI_Close (i2s_ctrl_t *const p_ctrl)
```

```
fsp_err_t R_SSI_VersionGet (fsp_version_t *const p_version)
```

Detailed Description

Driver for the SSIE peripheral on RA MCUs. This module implements the [I2S Interface](#).

Overview

Features

The SSI module supports the following features:

- Transmission and reception of uncompressed audio data using the standard I2S protocol
- Full-duplex I2S communication (channel 0 only)
- Integration with the DTC transfer module
- Internal connection to GPT GTIOC1A timer output to generate the audio clock
- Callback function notification when all data is loaded into the SSI FIFO

Configuration

Build Time Configurations for r_ssi

The following build time configurations are defined in fsp_cfg/r_ssi_cfg.h:

Configuration	Options	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	If selected code for parameter checking is included in the build.
DTC Support	• Enabled • Disabled	If code for DTC transfer support is included in the build.

Configurations for I2S Driver on r_ssi

This module can be added to the Threads tab from New -> Driver -> Connectivity -> I2S Driver on r_ssi:

4.2.37 Universal Serial Bus (r_usb_basic)

Modules

Functions

```
fsp_err_t R_USB_Open (usb_ctrl_t *const p_api_ctrl, usb_cfg_t const *const
```

p_cfg, usb_instance_transfer_t *p_api_trans)

Applies power to the USB module specified in the argument (p_ctrl).
[More...](#)

[fsp_err_t](#) [R_USB_Close](#) (usb_ctrl_t *const p_api_ctrl, usb_instance_transfer_t *p_api_trans)

Terminates power to the USB module specified in argument (p_ctrl). USB0 module stops when USB_IP0 is specified to the member (module), USB1 module stops when USB_IP1 is specified to the member (module). [More...](#)

[fsp_err_t](#) [R_USB_Read](#) (usb_ctrl_t *const p_api_ctrl, uint8_t *p_buf, uint32_t size, usb_instance_transfer_t *p_api_trans)

Bulk/interrupt data transfer and control data transfer. [More...](#)

[fsp_err_t](#) [R_USB_Write](#) (usb_ctrl_t *const p_api_ctrl, uint8_t *p_buf, uint32_t size, usb_instance_transfer_t *p_api_trans)

Bulk/Interrupt data transfer and control data transfer. [More...](#)

[fsp_err_t](#) [R_USB_Stop](#) (usb_ctrl_t *const p_api_ctrl, [usb_transfer_t](#) type, usb_instance_transfer_t *p_api_trans)

Requests a data read/write transfer be terminated when a data read/write transfer is being performed. [More...](#)

[fsp_err_t](#) [R_USB_Suspend](#) (usb_ctrl_t *const p_api_ctrl, usb_instance_transfer_t *p_api_trans)

Sends a SUSPEND signal from the USB module assigned to the member (module) of the usb_ctrl_t structure. [More...](#)

[fsp_err_t](#) [R_USB_Resume](#) (usb_ctrl_t *const p_api_ctrl, usb_instance_transfer_t *p_api_trans)

Sends a RESUME signal from the USB module assigned to the member (module) of the usb_ctrl_t structure. [More...](#)

[fsp_err_t](#) [R_USB_VbusSet](#) (usb_ctrl_t *const p_api_ctrl, uint16_t state, usb_instance_transfer_t *p_api_trans)

Specifies starting or stopping the VBUS supply. [More...](#)

[fsp_err_t](#) [R_USB_InfoGet](#) (usb_ctrl_t *const p_api_ctrl, usb_info_t *p_info)

Obtains completed USB-related events. [More...](#)

fsp_err_t [R_USB_PipeRead](#) (usb_ctrl_t *const p_api_ctrl, uint8_t *p_buf, uint32_t size, usb_instance_transfer_t *p_api_trans)

Requests a data read (bulk/interrupt transfer) via the pipe specified in the argument. [More...](#)

fsp_err_t [R_USB_PipeWrite](#) (usb_ctrl_t *const p_api_ctrl, uint8_t *p_buf, uint32_t size, usb_instance_transfer_t *p_api_trans)

Requests a data write (bulk/interrupt transfer). [More...](#)

fsp_err_t [R_USB_PipeStop](#) (usb_ctrl_t *const p_api_ctrl, usb_instance_transfer_t *p_api_trans)

Terminates a data read/write operation. [More...](#)

fsp_err_t [R_USB_UsedPipesGet](#) (usb_ctrl_t *const p_api_ctrl, uint16_t *p_pipe)

Gets the selected pipe number (number of the pipe that has completed initialization) via bit map information. [More...](#)

fsp_err_t [R_USB_PipeInfoGet](#) (usb_ctrl_t *const p_api_ctrl, usb_pipe_t *p_info)

Gets the following pipe information regarding the pipe specified in the argument (p_ctrl) member (pipe): endpoint number, transfer type, transfer direction and maximum packet size. [More...](#)

fsp_err_t [R_USB_PullUp](#) (uint8_t state)

This API enables or disables pull-up of D+/D- line. [More...](#)

fsp_err_t [R_USB_EventGet](#) (usb_ctrl_t *const p_api_ctrl, usb_status_t *event)

Obtains completed USB related events. [More...](#)

fsp_err_t [R_USB_VersionGet](#) (fsp_version_t *const p_version)

Returns the version of this module. [More...](#)

fsp_err_t [R_USB_ModuleNumberGet](#) (usb_ctrl_t *const p_api_ctrl, uint8_t *module_number)

This API gets the module number. [More...](#)

`fsp_err_t` [R_USB_ClassTypeGet](#) (`usb_ctrl_t *const p_api_ctrl, usb_class_t *class_type`)

This API gets the class type. [More...](#)

`fsp_err_t` [R_USB_DeviceAddressGet](#) (`usb_ctrl_t *const p_api_ctrl, uint8_t *device_address`)

This API gets the device address. [More...](#)

`fsp_err_t` [R_USB_PipeNumberGet](#) (`usb_ctrl_t *const p_api_ctrl, uint8_t *pipe_number`)

This API gets the pipe number. [More...](#)

`fsp_err_t` [R_USB_DeviceStateGet](#) (`usb_ctrl_t *const p_api_ctrl, uint16_t *state`)

This API gets the state of the device. [More...](#)

`fsp_err_t` [R_USB_DataSizeGet](#) (`usb_ctrl_t *const p_api_ctrl, uint32_t *data_size`)

This API gets the data size. [More...](#)

`fsp_err_t` [R_USB_SetupGet](#) (`usb_ctrl_t *const p_api_ctrl, usb_setup_t *setup`)

This API gets the setup type. [More...](#)

Detailed Description

The USB module (r_usb_basic) provides an API to perform H / W control of USB communication. It implements the [USB Interface](#).

Overview

The USB module performs USB hardware control. The USB module operates in combination with one type of sample device class drivers provided by Renesas.

Features

The USB module has the following key features:

- Overall
 - Supporting USB Host or USB Peripheral.
 - Device connect/disconnect, suspend/resume, and USB bus reset processing.
 - Control transfer on pipe 0.
 - Data transfer on pipes 1 to 9. (Bulk or Interrupt transfer)
 - This driver supports RTOS version (hereinafter called "RTOS") and Non-OS version

(hereinafter called "Non-OS"). RTOS uses the realtime OS (FreeRTOS). Non-OS does not use the real time OS.

- Host mode
 - In host mode, enumeration as Low-speed/Full-speed/Hi-speed device (However, operating speed is different by devices ability.)
 - Transfer error determination and transfer retry.
- Peripheral mode
 - In peripheral mode, enumeration as USB Host of USB1.1/2.0/3.0.

Configuration

Build Time Configurations for r_usb_basic

The following build time configurations are defined in fsp_cfg/r_usb_basic_cfg.h:

Configuration	Options	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	If selected code for parameter checking is included in the build.
USB Operating mode Setting	• Host mode • Peri mode	If Peri mode is selected, USB operates as Peripheral.
Device Class Setting	• Host Communication Device Class • Host Human Interface Device Class • Host Mass Storage Class • Host Vendor Class • Peripheral Communication Device Class • Peripheral Human Interface Device Class • Peripheral Mass Storage Class • Peripheral Vendor Class	Set USB to work in the selected class.
DTC use setting	• Uses DTC • Does not use DTC	When it is enabled, it will operate using DTC.
DMA use setting	• Uses DMA • Does not use DMA	When it is enabled, it will operate using DMA.
DMA channel setting for transmission using USB0 module	• Uses DMAC0 • Uses DMAC1 • Uses DMAC2 • Uses DMAC3 • Uses DMAC4 • Uses DMAC5 • Uses DMAC6 • Uses DMAC7	Use the set channel for transmission.
DMA channel setting for reception using USB0 module	• Uses DMAC0 • Uses DMAC1	Use the set channel for reception.

	• Uses DMAC2 • Uses DMAC3 • Uses DMAC4 • Uses DMAC5 • Uses DMAC6 • Uses DMAC7	
DMA channel setting for transmission using USB1 module	• Uses DMAC0 • Uses DMAC1 • Uses DMAC2 • Uses DMAC3 • Uses DMAC4 • Uses DMAC5 • Uses DMAC6 • Uses DMAC7	Use the set channel for transmission.
DMA channel setting for reception using USB1 module	• Uses DMAC0 • Uses DMAC1 • Uses DMAC2 • Uses DMAC3 • Uses DMAC4 • Uses DMAC5 • Uses DMAC6 • Uses DMAC7	Use the set channel for reception.
PLL clock frequency setting	• 24MHz • 20MHz • Other than 24/20MHz	In the case of a USB module other than USB1 module, this definition is ignored.
CPU bus access wait setting	See e2 studio for available options.	CPU Bus Access Wait Select(CPU Bus Wait Register (BUSWAIT)BWAIT[3:0]) 2-17 access cycle wait
Setting the battery charging function	• Using the battery charging function • Not using the battery charging function	Not using the battery charging function Using the battery charging function
Setting the power source IC	• High assert • Low assert	Select High assert or Low assert.
Setting USB port operation when using the battery charging function	• DCP enabled • DCP disabled	Please select whether to deactivate or activate the DCP.
Setting USB module to be used	• Using USB0 module • Using USB1 module	During peripheral operation, select whether to use USB 0 or 1.
Setting whether to notify the application when receiving the request(SET_INTERFACE/SET_FEATURE/CLEAR_FEATURE)	• Not notifying. • Notifying	Please choose whether it corresponds to the class request.
Select whether to use the double buffer function.	• Not Using double buffer • Using double buffer	Please choose whether it corresponds to the double buffer.

Select whether to use the continuous transfer mode.

- Not Using continuous transfer mode
- Using continuous transfer mode

Please choose whether it corresponds to the continuous transfer mode.

FreeRTOS Integration

- Do not use FreeRTOS.
- Use FreeRTOS.

Select whether to use FreeRTOS with USB.

Configurations for USB Driver on r_usb_basic

This module can be added to the Threads tab from New -> Middleware -> USB -> USB Driver on r_usb_basic:

4.2.38 Host Mass Storage Class Driver (r_usb_hmsc)

Modules

The USB module (r_usb_hmsc) provides an API to perform hardware control of USB communications. It implements the [USB Interface](#).

This module is USB Basic Host and Peripheral. It works in combination with Driver (r_usb_basic module).

Overview

The r_usb_hmsc module, when used in combination with the r_usb_basic module, operates as a USB host mass storage class driver (HMSC). HMSC is built on the USB mass storage class Bulk-Only Transport (BOT) protocol. It is possible to communicate with BOT-compatible USB storage devices by combining it with the file system and storage device driver. This module should be used in combination with the FreeRTOS+FAT File System.

Features

The r_usb_hmsc module has the following key features:

- Checking of connected USB storage devices (to determine whether or not operation is supported)
- Storage command communication using the BOT protocol
- Support for SFF-8070i (ATAPI) USB mass storage subclass
- Sharing of a single pipe for IN/OUT directions or multiple devices
- Maximum 4 USB storage devices can be connected

Class Driver Overview

1. Class Requests

The class requests supported by this driver are shown below.

Request	Description
GetMaxLun	Gets the maximum number of units that are supported.

MassStrageReset

Cancels a protocol error.

2. Storage Commands

This driver supports the following storage command.

- TEST_UNIT_READY
- REQUEST_SENSE
- MODE_SELECT10
- MODE_SENSE10
- PREVENT_ALLOW
- READ_FORMAT_CAPACITY
- READ10
- WRITE10

Configuration

Clock Configuration

Refer to [Universal Serial Bus \(r_usb_basic\)](#) basic module.

Pin Configuration

Refer to [Universal Serial Bus \(r_usb_basic\)](#) basic module.

Usage Notes

- This driver is not guaranteed to provide USB communication operation. The customer should verify operation when utilizing it in a system and confirm the ability to connect to a variety of different types of devices.
- This module must be incorporated into a project using r_usb_basic. Once incorporated into a project, use the API to perform USB hardware control.
- This driver is confirmed for operation in combination with the FreeRTOS+FAT File System.

Limitations

1. Some MSC devices may be unable to connect (because they are not recognized as storage devices).
2. MSC devices that return values of 1 or higher in response to the GetMaxLun command (mass storage class command) are not supported.
3. Maximum 4 USB storage devices can be connected.
4. USB storage devices with a sector size of 512 bytes can be connected.
5. A device that does not respond to the READ_CAPACITY command operates as a device with a sector size of 512 bytes.

Examples

USB HMSC Example

Example Operating Environment

The following shows an example operating environment for the HMSC.

Refer to the associated instruction manuals for details on setting up the evaluation board and using the emulator, etc.

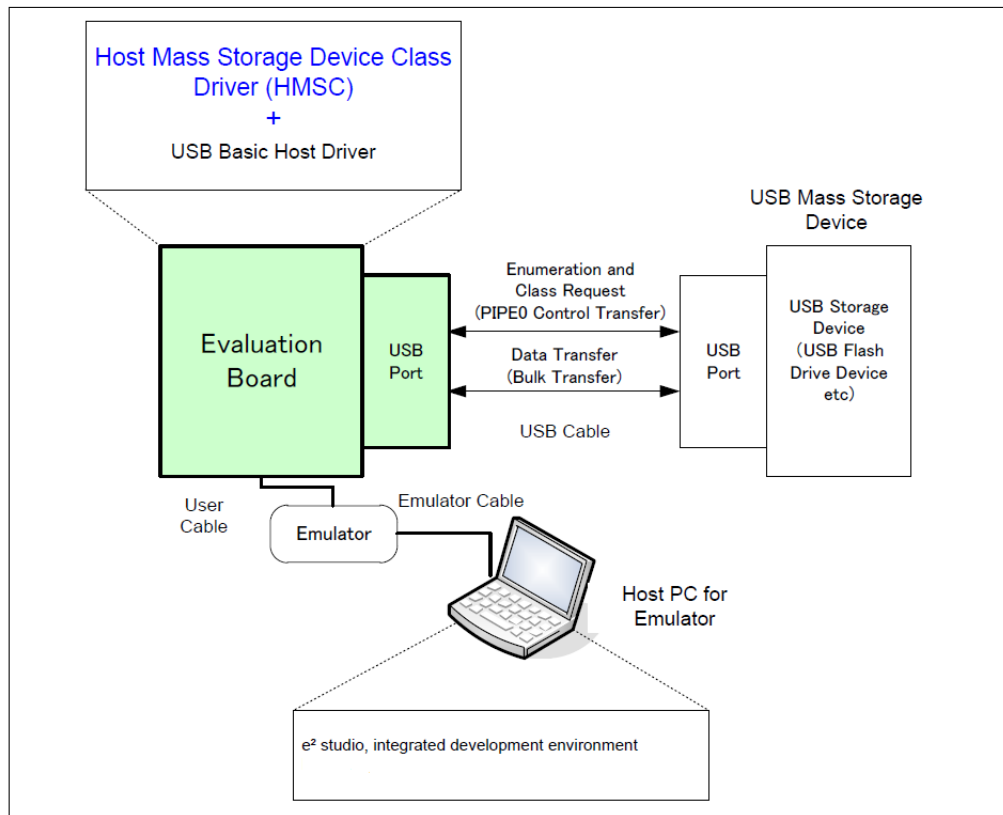


Figure 99: Example Operating Environment

Application Specifications

The main functions of the application are as follows:

1. Performs enumeration and drive recognition processing on MSC devices.
2. After the above processing finishes, the application writes the file hmscdemo.txt to the MSC device once.
3. After writing the above file, the APL repeatedly reads the file hmscdemo.txt. It continues to read the file repeatedly until the switch is pressed again.

Application Processing (for RTOS)

This application has two tasks. An overview of the processing in these two tasks is provided below.

usb_apl_task

1. After start up, MCU pin setting, USB controller initialization, and application program initialization are performed.
2. The MSC device is attached to the kit. When enumeration and drive recognition processing have completed, the USB driver calls the callback function (usb_apl_callback). In the callback function (usb_apl_callback), the application task is notified of the USB completion event using the FreeRTOS functionality.
3. In the application task, information regarding the USB completion event about which

notification was received from the callback function is retrieved using the real-time OS functionality.

4. If the USB completion event (the event member of the `usb_ctrl_t` structure) retrieved in step 2 above is `USB_STS_CONFIGURED` then, based on the USB completion event, the MSC device is mounted and the file is written to the MSC device.
5. If the USB completion event (the event member of the `usb_ctrl_t` structure) retrieved in step 2 above is `USB_STS_DETACH`, the application initializes the variables for state management.

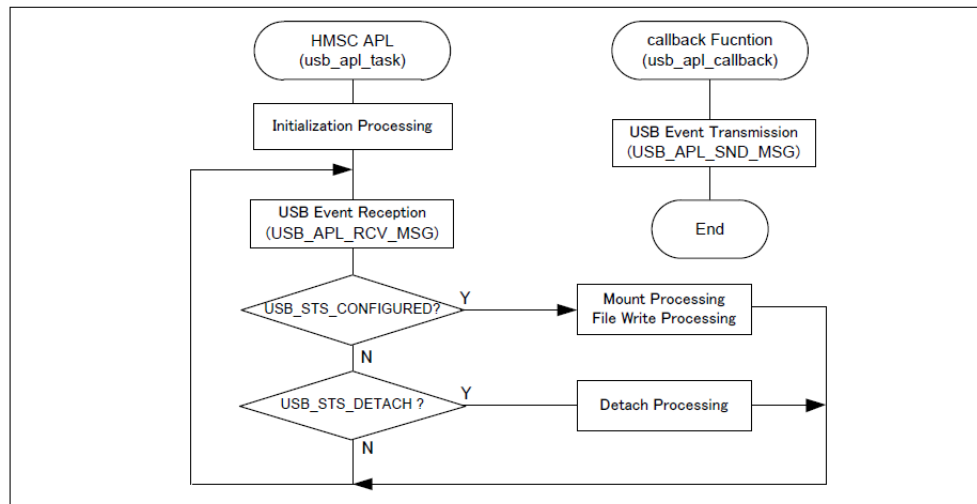


Figure 100: `usb_apl_task`

file_read_task

Of the application tasks `usb_apl_task` and `file_read_task`, `file_read_task` is processed while `usb_apl_task` is in the wait state. This task performs file read processing on the file that was written to the MSC device (`hmscdemo.txt`).

This is an hmsc example of minimal use of the USB in an application.

```

void usb_hmsc_example (void)
{
    usb_instance_ctrl_t ctrl;
    usb_instance_transfer_t trans;
    usb_instance_transfer_t *p_mess;
    uint8_t g_buf[USB_VALUE_64];
    capacity_list_t *pcl;
    FF_Disk_t *USB_ret = NULL;
    size_t size_return;
    int close_err;
    apl_init();
    usb_pin_setting(); /* USB pin function and port mode setting. */

```

```
trans.module_number = USB_IP0;
trans.type = USB_CLASS_HMSC;
g_usb_on_usb.open(&ctrl, &g_usb0_cfg, &trans);
g_usb0_cfg.p_usb_apl_callback = &usb_apl_callback;
R_USB_Callback(g_usb0_cfg.p_usb_apl_callback);
usb_configured = 0;
while (1)
{
    USB_APL_RCV_MSG(USB_APL_MBX, (usb_msg_t **)&p_mess);
    trans = *p_mess;
    switch (trans.event)
    {
    case USB_STATUS_CONFIGURED :
        g_buf[0] = USB_VALUE_3FH; /* Page Code */
        g_hmsc_on_usb.strgcmd(&ctrl, g_buf, USB_ATAPI_READ_FORMAT_CAPACITY,
&trans);

        pcl = (capacity_list_t *)g_buf;
        if(pcl->current_capacity_header.descriptor_code == 0x02)
        {
            usb_device_capacity_blocks = \
                ((uint32_t)pcl->current_capacity_header.number_of_blocks[0]
<< 24) | \
                ((uint32_t)pcl->current_capacity_header.number_of_blocks[1]
<< 16) | \
                ((uint32_t)pcl->current_capacity_header.number_of_blocks[2]
<< 8) | \
                ((uint32_t)pcl->current_capacity_header.number_of_blocks[3]);
            usb_configured = 1;
        }
        USB_ret = FF_DiskInit( (char*)main_USB_DISK_NAME, &disk_info);
        if(NULL == USB_ret)
        {
            printf("File Init Fail");
        }
    }
```



```
        R_USB_HmscSemGet();
/* Open the source file in read only mode. */
        pxSourceFile = ff_fopen((const char *)"TEST_USB.txt", (const char *)"w");
if(0 != pxSourceFile)
    {
/* Write however many bytes were read from the source file into the destination
file. */
        size_return = ff_fwrite( g_file_data, sizeof(g_file_data), 1,
pxSourceFile );
        if(1 == size_return)
            {
                g_isFileWrite = USB_APL_YES;
            }
        else
            {
                printf("File Write Fail");
            }
        close_err = ff_fclose( pxSourceFile );
        if(0 != close_err)
            {
                printf("File Close Fail");
            }
            }
        R_USB_HmscSemRel();
break;
case USB_STATUS_DETACH :
        g_isFileWrite = USB_APL_NO;
break;
        default :
break;
    } /* switch( event ) */
} /* while(1) */
} /* End of function usb_hmsc_example() */
```

4.2.39 Universal Serial Bus Peripheral Communication Device Class (r_usb_pcdc)

Modules

This module is USB Peripheral Communication Device Class Driver (PCDC).
This module works in combination with (r_usb_basic module).

Overview

The r_usb_pcdc module combines with the r_usb_basic module to provide USB Peripheral It operates as a communication device class driver (hereinafter referred to as PCDC).
PCDC conforms to Abstract Control Model of USB communication device class specification (hereinafter referred to as CDC) and can communicate with USB host.

Features

The r_usb_pcdc module has the following key features:

- Data transfer to and from a USB host.
- Response to CDC class requests.
- Provision of communication device class notification transmit service.

Basic Functions

CDC conforms to the communication device class specification Abstract Control Model subclass.

Abstract Control Model Overview

The Abstract Control Model subclass of CDC is a technology that bridges the gap between USB devices and earlier modems (employing RS-232C connections), enabling use of application programs designed for older modems. The class requests and class notifications supported are listed below.

Class Requests (Host to Peripheral)

This driver notifies to the application program when receiving the following class request.

Request	Code	Description
SetLineCoding	0x20	Makes communication line settings (communication speed, data length, parity bit, and stop bit length).
GetLineCoding	0x21	Acquires the communication line setting state.
SetControlLineState	0x22	Makes communication line control signal (RTS, DTR) settings.

For details concerning the Abstract Control Model requests, refer to Table 11, [Requests - Abstract Control Model] in [USB Communications Class Subclass Specification for PSTN Devices], Revision 1.2.

Data Format of Class Requests

The data format of the class requests supported by the class driver software is described below.

1.SetLineCoding

This is the class request the host transmits to perform the UART line setting.
The SetLineCoding data format is shown below.

SetLineCoding Format

bmRequestType	bRequest	wValue	wIndex	wLength	Data
0x21	SET_LINE_CODING(0x20)	0x00	0x0	0x07	Line Coding Structure

Line Coding Structure

Offset	Field	Size	Value	Description
0	DwDTERate	4	Number	Data terminal speed (bps)
4	BcharFormat	1	Number	Stop bits: 0 - 1 stop bit 1 - 1.5 stop bits 1 - 1.5 stop bits 2 - 2 stop bits
5	BparityType	1	Number	Parity: 0 - None 1 - Odd 2 - Even
6	BdataBits	1		Data bits (5, 6, 7, 8)

2.GetLineCoding

This is the class request the host transmits to request the UART line state.
The GetLineCoding data format is shown below.

GetLineCoding Format

bmRequestType	bRequest	wValue	wIndex	wLength	Data
0xA1	GET_LINE_CODING(0x21)	0x00	0x0	0x07	Line Coding Structure

3.SetControlLineState

This is a class request that the host sends to set up the signal for flow controls of UART.
 This software does not support RTS/DTR control.
 The SET_CONTROL_LINE_STATE data format is shown below.

SET_CONTROL_LINE_STATE Format

bmRequestType	bRequest	wValue	wIndex	wLength	Data
0x21	SET_CONTROL_LINE_STATE(0x22)	Control Signal Bitmap	0x0	0x00	None

Control Signal Bitmap

Bit Position	Description
D15 to D2	Reserved (reset to 0)
D1	DCE transmit function control: 0 - RTS Off 1 - RTS On
D0	Notification of DTE ready state: 0 - DTR Off 1 - DTR On

Class Notifications (Peripheral to Host)

The table below shows the class notification support / non-support of this S / W.

Notification	Code	Description	Supported
NETWORK_CONNECTION	0x00	Notification of network connection state	No
RESPONSE_AVAILABLE	0x01	Response to GET_ENCAPSULATED_RESPONSE	No
SERIAL_STATE	0x20	Notification of serial line state	Yes

1. Serial State

The host is notified of the serial state when a change in the UART port state is detected.
 This software supports the detection of overrun, parity and framing errors. A state notification is performed when a change from normal state to error is detected. However, notification is not continually transmitted when an error is continually detected.

SerialState Format

bmRequestType	bRequest	wValue	wIndex	wLength	Data
0xA1	SERIAL_STATE(0x20)	0x00	0x0	0x02	UART State bitmap

UART state bitmap format

Bits	Field	Description	Supported
D15 to D7		Reserved	-
D6	b_over_run	Overrun error detected	Yes
D5	b_parity	Parity error detected	Yes
D4	b_framing	Framing error detected	Yes
D3	b_ring_signal	INCOMING signal (ring signal) detected	No
D2	b_break	Break signal detected	No
D1	btx_arrier	Data Set Ready: Line connected and ready for communication	No
D0	brx_carrier	Data Carrier Detect: Carrier detected on line	No

PC Virtual COM-port Usage

The CDC device can be used as a virtual COM port when operating in Windows OS.

Use a PC running Windows OS, and connect an board. After USB enumeration, the CDC class requests GetLineCoding and SetControlLineState are executed by the target, and the CDC device is registered in Windows Device Manager as a virtual COM device.

Registering the CDC device as a virtual COM-port in Windows Device Manager enables data communication with the CDC device via a terminal app such as [HyperTerminal] which comes standard with Windows OS. When changing settings of the serial port in the Windows terminal application, the UART setting is propagated to the firmware via the class request SetLineCoding. Data input (or file transmission) from the terminal app window is transmitted to the board using endpoint 2 (EP2); data from the board side is transmitted to the PC using EP1.

When the last packet of data received is the maximum packet size, and the terminal determines that there is continuous data, the received data may not be displayed in the terminal. If the received data is smaller than the maximum packet size, the data received up to that point is displayed in the terminal.

The received data is outputted on the terminal when the data less than Maximum packet size is received.

Configuration

Build Time Configurations for r_usb_pcdc

The following build time configurations are defined in fsp_cfg/r_usb_pcdc_cfg.h:

Configuration	Options	Description
Select which pipe to use for bulk IN transfer during PCDC operation.	• Using USB PIPE1 • Using USB PIPE2 • Using USB PIPE3 • Using USB PIPE4 • Using USB PIPE5	Please choose between 1 and 5.

Select which pipe to use for bulk OUT transfer during PCDC operation.

- Using USB PIPE1
- Using USB PIPE2
- Using USB PIPE3
- Using USB PIPE4
- Using USB PIPE5

Please choose between 1 and 5.

Select which pipe to use for Interrupt IN transfer during PCDC operation.

- Using USB PIPE6
- Using USB PIPE7
- Using USB PIPE8
- Using USB PIPE9

Please choose between 6 and 9.

Configurations for USB PCDC driver on r_usb_pcdc

This module can be added to the Threads tab from New -> Middleware -> USB -> USB PCDC driver on r_usb_pcdc:

4.2.40 Watchdog Timer (r_wdt)

Modules

Functions

`fsp_err_t R_WDT_Refresh (wdt_ctrl_t *const p_ctrl)`

`fsp_err_t R_WDT_Open (wdt_ctrl_t *const p_ctrl, wdt_cfg_t const *const p_cfg)`

`fsp_err_t R_WDT_StatusClear (wdt_ctrl_t *const p_ctrl, const wdt_status_t status)`

`fsp_err_t R_WDT_StatusGet (wdt_ctrl_t *const p_ctrl, wdt_status_t *const p_status)`

`fsp_err_t R_WDT_CounterGet (wdt_ctrl_t *const p_ctrl, uint32_t *const p_count)`

`fsp_err_t R_WDT_TimeoutGet (wdt_ctrl_t *const p_ctrl, wdt_timeout_values_t *const p_timeout)`

`fsp_err_t R_WDT_VersionGet (fsp_version_t *const p_version)`

Detailed Description

Driver for the WDT peripheral on RA MCUs. This module implements the [WDT Interface](#).

Overview

The watchdog timer is used to recover from unexpected errors in an application. The watchdog timer must be refreshed periodically in the permitted count window by the application. If the count is allowed to underflow or refresh occurs outside of the valid refresh period, the WDT resets the device or generates an NMI.

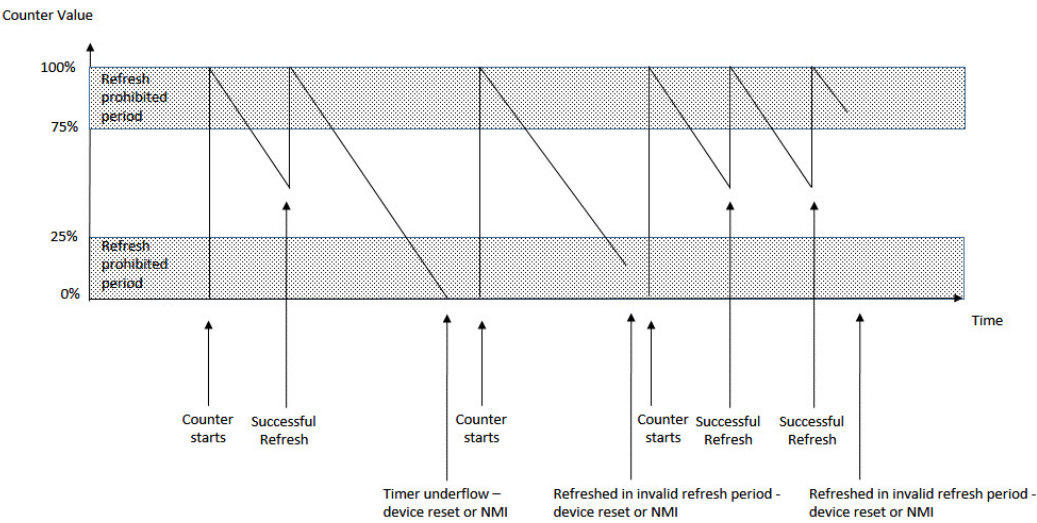


Figure 101: Watchdog Timer Operation Example

Features

The WDT HAL module has the following key features:

- When the WDT underflows or is refreshed outside of the permitted refresh window, one of the following events can occur:
 - Resetting of the device
 - Generation of an NMI
- The WDT has two supported modes:
 - In auto start mode, the WDT begins counting at reset.
 - In register start mode, the WDT can be started from the application.

Selecting a Watchdog

RA MCUs have two watchdog peripherals: the watchdog timer (WDT) and the independent watchdog timer (IWDT). When selecting between them, consider these factors:

	WDT	IWDT
Start Mode	The WDT can be started from the application (register start mode) or configured by hardware to start automatically (auto start mode).	The IWDT can only be configured by hardware to start automatically.
Clock Source	The WDT runs off a peripheral clock.	The IWDT has its own clock source which improves safety.

Configuration

When using register start mode, configure the watchdog timer on the Threads tab.

Note

When using auto start mode, configurations on the Threads tab are ignored. Configure the watchdog using the OFS settings on the BSP tab.

Build Time Configurations for r_wdt

The following build time configurations are defined in fsp_cfg/r_wdt_cfg.h:

Configuration	Options	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	If selected code for parameter checking is included in the build.
Register Start NMI Support	• Enabled • Disabled	If enabled, code for NMI support in register start mode is included in the build.

Configurations for Watchdog Driver on r_wdt

This module can be added to the Threads tab from New -> Driver -> Monitoring -> Watchdog Driver on r_wdt:

4.2.41 SEGGER emWin Port (rm_emwin_port)

Modules

SEGGER emWin port for RA MCUs.

Overview

The SEGGER emWin RA Port module provides the configuration and hardware acceleration support necessary for use of emWin on RA products. The port provides full integration with the graphics peripherals (GLCDC, DRW and JPEG) as well as Amazon FreeRTOS.

Note

This port layer primarily enables hardware acceleration and background handling of many display operations and does not contain code intended to be directly called by the user. Please consult the SEGGER emWin User Guide (UM03001) for details on how to use emWin in your project.

Hardware Acceleration

The following functions are currently performed with hardware acceleration:

- Drawing bitmaps (ARGB8888 and RGB565)

- Rectangle fill
- Line and shape drawing
- Anti-aliased operations
 - Circle stroke and fill
 - Polygon stroke and fill
 - Lines and arcs
- JPEG decoding
- LCD panel data conversion and output

Configuration

Build Time Configurations for rm_emwin_port

The following build time configurations are defined in fsp_cfg/rm_emwin_port_cfg.h:

Configuration	Options	Description
Memory Allocation GUI Heap Size	Value must be a non-negative integer	Set the size of the heap to be allocated for use exclusively by emWin.
Memory Allocation Section for GUI Heap	Configurable String	Specify the section in which to allocate the GUI heap.
Memory Allocation Maximum Layers	Value must be a non-negative integer	Set the maximum number of available display layers.
Configuration RTOS Support	• Enabled • Disabled	Enable or disable RTOS awareness (multithreading support).
Configuration Touch Panel Support	• Enabled • Disabled	Enable or disable touch panel support.
Configuration Mouse Support	• Enabled • Disabled	Enable or disable support for mouse input.
Configuration Memory Devices	• Enabled • Disabled	Enable or disable support for memory devices, which allow the user to allocate their own memory in the GUI heap.
Configuration Text Rotation	• Enabled • Disabled	Enable or disable support for displaying rotated text.
Configuration Window Manager	• Enabled • Disabled	Enable or disable the emWin Window Manager (WM).
Configuration Bidirectional Text	• Enabled • Disabled	Enable or disable support for bidirectional text (such as Arabic or Hebrew).
Configuration Debug Logging Level	• None (0) • Parameter checking only (1) • All checks enabled (2) • Log errors (3)	Set the debug logging level.

- Log warnings (4)
- Log all messages (5)

JPEG Decoding Error Timeout	Value must be a non-negative integer	Set the timeout for JPEG decoding operations (in RTOS ticks) in the event of a decode error.
JPEG Decoding Input Buffer Size	Value must be a non-negative integer	Set the size of the JPEG decode input buffer (in bytes). This buffer is used to ensure 8-byte alignment of input data. Specifying a size smaller than the size of the JPEG to decode will use additional interrupts to stream data in during the decoding process.
JPEG Decoding Output Buffer Size	Value must be a non-negative integer	Set the size of the JPEG decode output buffer (in bytes). An output buffer smaller than the size of a decoded image will use additional interrupts to stream the data into a framebuffer.
JPEG Decoding Section for Buffers	Configurable String	Specify the section in which to allocate the JPEG work buffers.

4.2.42 FreeRTOS Plus FAT (rm_freertos_plus_fat)

Modules

Functions

fsp_err_t	RM_FREERTOS_PLUS_FAT_Open (freertos_plus_fat_ctrl_t *p_ctrl, freertos_plus_fat_cfg_t *p_cfg) Returns the version of this module. The version number is encoded such that the top two bytes are the major version number and the bottom two bytes are the minor version number. More...
fsp_err_t	RM_FREERTOS_PLUS_FAT_Close (freertos_plus_fat_ctrl_t *p_ctrl) Returns the version of this module. The version number is encoded such that the top two bytes are the major version number and the bottom two bytes are the minor version number. More...
fsp_err_t	RM_FREERTOS_PLUS_FAT_VersionGet (fsp_version_t *const p_version) Returns the version of this module. The version number is encoded such that the top two bytes are the major version number and the

bottom two bytes are the minor version number. [More...](#)

Detailed Description

Middleware for the Fat File System control on RA MCUs.

Overview

The FreeRTOS Plus FAT performs Fat File System control. This middleware is based on open source. Please refer to the following URL for details. https://www.freertos.org/FreeRTOS-Plus/FreeRTOS_Plus_FAT/index.html

Features

The FreeRTOS Plus FAT module supports the following features:

- File read support
- File write support

Configuration

Configurations for FreeRTOS+FAT

This module can be added to the Threads tab from New -> FreeRTOS+ -> FreeRTOS+FAT:

Configuration	Options	Description
pcDeviceName	Name must be a valid C symbol	Name must be a valid C symbol
Partition	Must be a valid number	Select the partition.
ulNumberOfSectors	Name must be a valid number	Select the ulNumberOfSectors.
NumberOfMemory	Must be a valid number	Select the NumberOfMemory.
ulSectorSize	Must be a valid number	Select the ulSectorSize.
xBlockDeviceIsReentrant	• Disable Reentrant • Enable Reentrant	Select the Reentrant.
ulSignature	Must be a valid number	Select the ulSignature.
bPartitionNumber	Must be a valid number	Select the bPartitionNumber.
device_type	• FREERTOS_PLUS_FAT_D EVICE_TYPE_USB • FREERTOS_PLUS_FAT_D EVICE_TYPE_END	Select the device_type.
status	• FREERTOS_PLUS_FAT_D EVICE_STATUS_UNINITUI ALIZED • FREERTOS_PLUS_FAT_D EVICE_STATUS_INITUIALI	Select the status.

ZED

Build Time Configurations for rm_freertos_plus_fat

The following build time configurations are defined in freertos_plus/FreeRTOSFATConfig.h:

4.2.43 Amazon FreeRTOS Port (rm_freertos_port)**Modules**

Amazon FreeRTOS port for RA MCUs.

Overview*Note*

The FreeRTOS Port does not provide any interfaces to the user. Consult the AWS FreeRTOS documentation at <https://www.freertos.org/> for further information.

Features

The RA FreeRTOS port supports the following features:

- Standard AWS FreeRTOS configurations
- Hardware stack monitor

Configuration**Build Time Configurations for all**

The following build time configurations are defined in aws/FreeRTOSConfig.h:

Configuration	Options	Description
General Custom FreeRTOSConfig.h	Configurable String	Add a path to your custom FreeRTOSConfig.h file. It can be used to override some or all of the configurations defined here, and to define additional configurations.
General Use Preemption	• Enabled • Disabled	Set to Enabled to use the preemptive RTOS scheduler, or Disabled to use the cooperative RTOS scheduler.
General Use Port Optimised Task Selection	• Enabled • Disabled	Some FreeRTOS ports have two methods of selecting the next task to execute - a generic method, and a method that is specific to that port. The Generic method:

Is used when Use Port Optimized Task Selection is set to 0, or when a port specific method is not implemented. Can be used with all FreeRTOS ports.

Is completely written in C, making it less efficient than a port specific method.

Does not impose a limit on the maximum number of available priorities.

A port specific method:

Is not available for all ports.

Is used when Use Port Optimized Task Selection is Enabled.

Relies on one or more architecture specific assembly instructions (typically a Count Leading Zeros [CLZ] or equivalent instruction) so can only be used with the architecture for which it was specifically written.

Is more efficient than the generic method.

Typically imposes a limit of 32 on the maximum number of available priorities.

General|Use Tickless Idle

- Enabled
- Disabled

Set Use Tickless Idle to Enabled to use the low power tickless mode, or Disabled to keep the tick interrupt running at all times. Low power tickless implementations are not provided for all FreeRTOS ports.

Hooks|Use Idle Hook

- Enabled
- Disabled

Set to Enabled if you wish to use an idle hook, or Disabled to omit an idle hook.

Hooks|Use Malloc Failed Hook

- Enabled
- Disabled

The kernel uses a call to `pvPortMalloc()` to allocate memory from the heap each time a task, queue or semaphore is created. The official FreeRTOS download includes four sample memory allocation schemes for this purpose. The schemes are implemented in the `heap_1.c`, `heap_2.c`, `heap_3.c`, `heap_4.c`

			and heap_5.c source files respectively. Use Malloc Failed Hook is only relevant when one of these three sample schemes is being used. The malloc() failed hook function is a hook (or callback) function that, if defined and configured, will be called if pvPortMalloc() ever returns NULL. NULL will be returned only if there is insufficient FreeRTOS heap memory remaining for the requested allocation to succeed. If Use Malloc Failed Hook is Enabled then the application must define a malloc() failed hook function. If Use Malloc Failed Hook is set to Disabled then the malloc() failed hook function will not be called, even if one is defined. Malloc() failed hook functions must have the name and prototype shown below. <pre>void vApplicationMallocFailedHook(void);</pre> If Use Timers and Use Daemon Task Startup Hook are both Enabled then the application must define a hook function that has the exact name and prototype as shown below. The hook function will be called exactly once when the RTOS daemon task (also known as the timer service task) executes for the first time. Any application initialisation code that needs the RTOS to be running can be placed in the hook function. <pre>void void vApplicationDaemonTaskStartupHook(void);</pre> Set to Enabled if you wish to use an tick hook, or Disabled to omit an tick hook. Enter the frequency in Hz at which the internal clock that
Hooks Use Daemon Task Startup Hook	• Enabled • Disabled		
Hooks Use Tick Hook	• Enabled • Disabled		
General Cpu Clock Hz		Configurable String	

General Tick Rate Hz	Must be an integer and greater than 0	drives the peripheral used to generate the tick interrupt will be executing - this is normally the same clock that drives the internal CPU clock. This value is required in order to correctly configure timer peripherals. The frequency of the RTOS tick interrupt. The tick interrupt is used to measure time. Therefore a higher tick frequency means time can be measured to a higher resolution. However, a high tick frequency also means that the RTOS kernel will use more CPU time so be less efficient. The RTOS demo applications all use a tick rate of 1000Hz. This is used to test the RTOS kernel and is higher than would normally be required. More than one task can share the same priority. The RTOS scheduler will share processor time between tasks of the same priority by switching between the tasks during each RTOS tick. A high tick rate frequency will therefore also have the effect of reducing the 'time slice' given to each task.
General Max Priorities	Must be an integer and greater than 0	The number of priorities available to the application tasks. Any number of tasks can share the same priority. Each available priority consumes RAM within the RTOS kernel so this value should not be set any higher than actually required by your application.
General Minimal Stack Size	Must be an integer and greater than 0	The size of the stack used by the idle task. Generally this should not be reduced from the value set in the FreeRTOSConfig.h file provided with the demo application for the port you are using. Like the stack size parameter to the xTaskCreate() and xTaskCreateStatic() functions,

			the stack size is specified in words, not bytes. If each item placed on the stack is 32-bits, then a stack size of 100 means 400 bytes (each 32-bit stack item consuming 4 bytes).
General	Max Task Name Len	Must be an integer and greater than 0	The maximum permissible length of the descriptive name given to a task when the task is created. The length is specified in the number of characters including the NULL termination byte.
Stats	Use Trace Facility	• Enabled • Disabled	Set to Enabled if you wish to include additional structure members and functions to assist with execution visualisation and tracing.
Stats	Use Stats Formatting Functions	• Enabled • Disabled	Set Use Trace Facility and Use Stats Formatting Functions to Enabled to include the vTaskList() and vTaskGetRunTimeStats() functions in the build. Setting either to Disabled will omit vTaskList() and vTaskGetRunTimeStats() from the build.
General	Use 16 Bit Ticks	Disabled	Time is measured in 'ticks' - which is the number of times the tick interrupt has executed since the RTOS kernel was started. The tick count is held in a variable of type TickType_t. Defining configUSE_16_BIT_TICKS as 1 causes TickType_t to be defined (typedef'ed) as an unsigned 16bit type. Defining configUSE_16_BIT_TICKS as 0 causes TickType_t to be defined (typedef'ed) as an unsigned 32bit type. Using a 16 bit type will greatly improve performance on 8 and 16 bit architectures, but limits the maximum specifiable time period to 65535 'ticks'. Therefore, assuming a tick frequency of 250Hz, the maximum time a task can delay

General|Idle Should Yield

- Enabled
- Disabled

or block when a 16bit counter is used is 262 seconds, compared to 17179869 seconds when using a 32bit counter.

This parameter controls the behaviour of tasks at the idle priority. It only has an effect if: The preemptive scheduler is being used.

The application creates tasks that run at the idle priority. If Use Time Slicing is Enabled then tasks that share the same priority will time slice. If none of the tasks get preempted then it might be assumed that each task at a given priority will be allocated an equal amount of processing time - and if the priority is above the idle priority then this is indeed the case. When tasks share the idle priority the behaviour can be slightly different. If Idle Should Yield is Enabled then the idle task will yield immediately if any other task at the idle priority is ready to run. This ensures the minimum amount of time is spent in the idle task when application tasks are available for scheduling. This behaviour can however have undesirable effects (depending on the needs of your application) as depicted below:

The diagram above shows the execution pattern of four tasks that are all running at the idle priority. Tasks A, B and C are application tasks. Task I is the idle task. A context switch occurs with regular period at times T0, T1, ..., T6. When the idle task yields task A starts to execute - but the idle task has already consumed some of the current time slice. This results in task I and task A effectively sharing the same time slice. The application tasks B and C therefore get more processing

time than the application task A.

This situation can be avoided by:

If appropriate, using an idle hook in place of separate tasks at the idle priority.

Creating all application tasks at a priority greater than the idle priority.

Setting Idle Should Yield to Disabled.

Setting Idle Should Yield to Disabled prevents the idle task from yielding processing time until the end of its time slice.

This ensure all tasks at the idle priority are allocated an equal amount of processing time (if none of the tasks get pre-empted) - but at the cost of a greater proportion of the total processing time being allocated to the idle task.

General|Use Task Notifications

- Enabled
- Disabled

Setting Use Task Notifications to Enabled will include direct to task notification functionality and its associated API in the build.

Setting Use Task Notifications to Disabled will exclude direct to task notification functionality and its associated API from the build.

Each task consumes 8 additional bytes of RAM when direct to task notifications are included in the build.

General|Use Mutexes

- Enabled
- Disabled

Set to Enabled to include mutex functionality in the build, or Disabled to omit mutex functionality from the build.

Readers should familiarise themselves with the differences between mutexes and binary semaphores in relation to the FreeRTOS functionality.

General|Use Recursive Mutexes

- Enabled
- Disabled

Set to Enabled to include recursive mutex functionality in the build, or Disabled to omit

General Use Counting Semaphores	• Enabled • Disabled	recursive mutex functionality from the build. Set to Enabled to include counting semaphore functionality in the build, or Disabled to omit counting semaphore functionality from the build.
Hooks Check For Stack Overflow	• Enabled • Disabled	The stack overflow detection page describes the use of this parameter. This is not recommended for RA MCUs with hardware stack monitor support. RA MCU designs should enable the RA hardware stack monitor instead.
General Queue Registry Size	Must be an integer and greater than 0	The queue registry has two purposes, both of which are associated with RTOS kernel aware debugging: It allows a textual name to be associated with a queue for easy queue identification within a debugging GUI. It contains the information required by a debugger to locate each registered queue and semaphore. The queue registry has no purpose unless you are using a RTOS kernel aware debugger. Registry Size defines the maximum number of queues and semaphores that can be registered. Only those queues and semaphores that you want to view using a RTOS kernel aware debugger need be registered. See the API reference documentation for <code>vQueueAddToRegistry()</code> and <code>vQueueUnregisterQueue()</code> for more information.
General Use Queue Sets	• Enabled • Disabled	Set to Enabled to include queue set functionality (the ability to block, or pend, on multiple queues and semaphores), or Disabled to omit queue set functionality.
General Use Time Slicing	• Enabled • Disabled	If Use Time Slicing is Enabled, FreeRTOS uses prioritised preemptive scheduling with

General Use Newlib Reentrant	• Enabled • Disabled	time slicing. That means the RTOS scheduler will always run the highest priority task that is in the Ready state, and will switch between tasks of equal priority on every RTOS tick interrupt. If Use Time Slicing is Disabled then the RTOS scheduler will still run the highest priority task that is in the Ready state, but will not switch between tasks of equal priority just because a tick interrupt has occurred. If Use Newlib Reentrant is Enabled then a newlib reent structure will be allocated for each created task. Note Newlib support has been included by popular demand, but is not used by the FreeRTOS maintainers themselves. FreeRTOS is not responsible for resulting newlib operation. User must be familiar with newlib and must provide system-wide implementations of the necessary stubs. Be warned that (at the time of writing) the current newlib design implements a system-wide malloc() that must be provided with locks.
General Enable Backward Compatibility	• Enabled • Disabled	The FreeRTOS.h header file includes a set of #define macros that map the names of data types used in versions of FreeRTOS prior to version 8.0.0 to the names used in FreeRTOS version 8.0.0. The macros allow application code to update the version of FreeRTOS they are built against from a pre 8.0.0 version to a post 8.0.0 version without modification. Setting Enable Backward Compatibility to Disabled in FreeRTOSConfig.h excludes the macros from the build, and in so doing allowing validation that no pre version 8.0.0 names are being used.

General Num Thread Local Storage Pointers	Must be an integer and greater than 0	Sets the number of indexes in each task's thread local storage array.
General Stack Depth Type	Configurable String	Sets the type used to specify the stack depth in calls to <code>xTaskCreate()</code>, and various other places stack sizes are used (for example, when returning the stack high water mark). Older versions of FreeRTOS specified stack sizes using variables of type <code>UBaseType_t</code>, but that was found to be too restrictive on 8-bit microcontrollers. Stack Depth Type removes that restriction by enabling application developers to specify the type to use.
General Message Buffer Length Type	Configurable String	FreeRTOS Message buffers use variables of type <code>Message Buffer Length Type</code> to store the length of each message. If <code>Message Buffer Length Type</code> is not defined then it will default to <code>size_t</code>. If the messages stored in a message buffer will never be larger than 255 bytes then defining <code>Message Buffer Length Type</code> to <code>uint8_t</code> will save 3 bytes per message on a 32-bit microcontroller. Likewise if the messages stored in a message buffer will never be larger than 65535 bytes then defining <code>Message Buffer Length Type</code> to <code>uint16_t</code> will save 2 bytes per message on a 32-bit microcontroller.
Memory Allocation Support Static Allocation	• Enabled • Disabled	If Support Static Allocation is Enabled then RTOS objects can be created using RAM provided by the application writer. If Support Static Allocation is Disabled then RTOS objects can only be created using RAM allocated from the FreeRTOS heap. If Support Static Allocation is left undefined it will default to 0.

If Support Static Allocation is Enabled then the application writer must also provide two callback functions: `vApplicationGetIdleTaskMemory()` to provide the memory for use by the RTOS Idle task, and (if Use Timers is Enabled) `vApplicationGetTimerTaskMemory()` to provide memory for use by the RTOS Daemon/Timer Service task. Examples are provided below.

```

/* Support Static Allocation is
Enabled, so the application
must provide an
implementation of vApplication
GetIdleTaskMemory() to provide
the memory that is
used by the Idle task. */
void
vApplicationGetIdleTaskMemory
( StaticTask_t
**ppxIdleTaskTCBBuffer,<br>
StackType_t
**ppxIdleTaskStackBuffer,<br>
uint32_t *pulIdleTaskStackSize )
{
/* If the buffers to be provided
to the Idle task are declared
inside this
function then they must be
declared static - otherwise they
will be allocated on
the stack and so not exists after
this function exits. */
static StaticTask_t
xIdleTaskTCB;
static StackType_t
uxIdleTaskStack[
configMINIMAL_STACK_SIZE ];

/* Pass out a pointer to the
StaticTask_t structure in which
the Idle task's
state will be stored. */
*ppxIdleTaskTCBBuffer =

/* Pass out the array that will be
used as the Idle task's stack. */
*ppxIdleTaskStackBuffer =
uxIdleTaskStack;

```

```

/* Pass out the size of the array
pointed to by
*ppxIdleTaskStackBuffer.
Note that, as the array is
necessarily of type
StackType_t,
configMINIMAL_STACK_SIZE is
specified in words, not bytes. */
*pullIdleTaskStackSize =
configMINIMAL_STACK_SIZE;
}
/*-----*/
-----*/

```

```

/* Support Static Allocation and
Use Timers are both Enabled,
so the
application must provide an
implementation of vApplication
GetTimerTaskMemory()
to provide the memory that is
used by the Timer service task.
*/
void vApplicationGetTimerTask
Memory( StaticTask_t
**ppxTimerTaskTCBBuffer,<br>
StackType_t **ppxTimerTaskSta
ckBuffer,<br> uint32_t
*pullTimerTaskStackSize )
{
/* If the buffers to be provided
to the Timer task are declared
inside this
function then they must be
declared static - otherwise they
will be allocated on
the stack and so not exists after
this function exits. */
static StaticTask_t
xTimerTaskTCB;
static StackType_t
uxTimerTaskStack[ configTIMER
_TASK_STACK_DEPTH ];

```

```

/* Pass out a pointer to the
StaticTask_t structure in which
the Timer
task's state will be stored. */
*ppxTimerTaskTCBBuffer =

```

```

/* Pass out the array that will be
used as the Timer task's stack.
*/
*ppxTimerTaskStackBuffer =

```

```
uxTimerTaskStack;
```

```
/* Pass out the size of the array
pointed to by
*ppxTimerTaskStackBuffer.
Note that, as the array is
necessarily of type
StackType_t,
configTIMER_TASK_STACK_DEPTH
is specified in words, not
bytes. */
*puTimerTaskStackSize = configTIMER_TASK_STACK_DEPTH;
}
```

Examples of the callback functions that must be provided by the application to supply the RAM used by the Idle and Timer Service tasks if Support Static Allocation is Enabled.

See the Static Vs Dynamic Memory Allocation page for more information.

If Support Dynamic Allocation is Enabled then RTOS objects can be created using RAM that is automatically allocated from the FreeRTOS heap. If Support Dynamic Allocation is set to 0 then RTOS objects can only be created using RAM provided by the application writer.

See the Static Vs Dynamic Memory Allocation page for more information.

The total amount of RAM available in the FreeRTOS heap. This value will only be used if Support Dynamic Allocation is Enabled and the application makes use of one of the sample memory allocation schemes provided in the FreeRTOS source code download. See the memory configuration section for further details.

By default the FreeRTOS heap is declared by FreeRTOS and

Memory Allocation|Support
Dynamic Allocation

- Enabled
- Disabled

Memory Allocation|Total Heap
Size

Must be an integer and greater
than 0

Memory Allocation|Application
Allocated Heap

- Enabled
- Disabled

placed in memory by the linker. Setting Application Allocated Heap to Enabled allows the heap to instead be declared by the application writer, which allows the application writer to place the heap wherever they like in memory.

If heap_1.c, heap_2.c or heap_4.c is used, and Application Allocated Heap is Enabled, then the application writer must provide a uint8_t array with the exact name and dimension as shown below. The array will be used as the FreeRTOS heap. How the array is placed at a specific memory location is dependent on the compiler being used - refer to your compiler's documentation.

```
uint8_t ucHeap[
configTOTAL_HEAP_SIZE];
```

Stats Generate Run Time Stats	• Enabled • Disabled	The Run Time Stats page describes the use of this parameter.
Timers Use Timers	• Enabled • Disabled	Set to Enabled to include software timer functionality, or Disabled to omit software timer functionality. See the FreeRTOS software timers page for a full description.
Timers Timer Task Priority	Must be an integer and greater than 0	Sets the priority of the software timer service/daemon task. See the FreeRTOS software timers page for a full description.
Timers Timer Queue Length	Must be an integer and greater than 0	Sets the length of the software timer command queue. See the FreeRTOS software timers page for a full description.
Timers Timer Task Stack Depth	Must be an integer and greater than 0	Sets the stack depth allocated to the software timer service/daemon task. See the FreeRTOS software timers page for a full description.
General Library Max Syscall Interrupt Priority	MCU Specific Options	The highest interrupt priority that can be used by any interrupt service routine that makes calls to interrupt safe FreeRTOS API functions. DO

NOT CALL INTERRUPT SAFE
FREERTOS API FUNCTIONS
FROM ANY INTERRUPT THAT
HAS A HIGHER PRIORITY THAN
THIS! (higher priorities are
lower numeric values)

Below is explanation for macros
that are set based on this value
from FreeRTOS website.

In the RA port, configKERNEL_IN
TERRUPT_PRIORITY is not used
and the kernel runs at the
lowest priority.

Note in the following discussion
that only API functions that end
in "FromISR" can be called from
within an interrupt service
routine.

configMAX_SYSCALL_INTERRUPT_
T_PRIORITY sets the highest
interrupt priority from which
interrupt safe FreeRTOS API
functions can be called.

A full interrupt nesting model is
achieved by setting configMAX_
SYSCALL_INTERRUPT_PRIORITY
above (that is, at a higher
priority level) than configKERN
EL_INTERRUPT_PRIORITY. This
means the FreeRTOS kernel
does not completely disable
interrupts, even inside critical
sections. Further, this is
achieved without the
disadvantages of a segmented
kernel architecture.

Interrupts that do not call API
functions can execute at
priorities above configMAX_SY
SCALL_INTERRUPT_PRIORITY and
therefore never be delayed by
the RTOS kernel execution.

A special note for ARM Cortex-M
users: Please read the page
dedicated to interrupt priority
settings on ARM Cortex-M
devices. As a minimum,
remember that ARM Cortex-M

cores use numerically low priority numbers to represent HIGH priority interrupts, which can seem counter-intuitive and is easy to forget! If you wish to assign an interrupt a low priority do NOT assign it a priority of 0 (or other low numeric value) as this can result in the interrupt actually having the highest priority in the system - and therefore potentially make your system crash if this priority is above `configMAX_SYSCALL_INTERRUPT_PRIORITY`.

The lowest priority on a ARM Cortex-M core is in fact 255 - however different ARM Cortex-M vendors implement a different number of priority bits and supply library functions that expect priorities to be specified in different ways. For example, on the RA6M3 the lowest priority you can specify is 15 - and the highest priority you can specify is 0.

General|Assert

Configurable String

The semantics of the `configASSERT()` macro are the same as the standard C `assert()` macro. An assertion is triggered if the parameter passed into `configASSERT()` is zero. `configASSERT()` is called throughout the FreeRTOS source files to check how the application is using FreeRTOS. It is highly recommended to develop FreeRTOS applications with `configASSERT()` defined.

The example definition (shown at the top of the file and replicated below) calls `vAssertCalled()`, passing in the file name and line number of the triggering `configASSERT()` call (`__FILE__` and `__LINE__` are standard macros provided by most compilers). This is just for demonstration as `vAssertCalled()` is not a

FreeRTOS function, configASSERT() can be defined to take whatever action the application writer deems appropriate.

It is normal to define configASSERT() in such a way that it will prevent the application from executing any further. This is for two reasons; stopping the application at the point of the assertion allows the cause of the assertion to be debugged, and executing past a triggered assertion will probably result in a crash anyway.

Note defining configASSERT() will increase both the application code size and execution time. When the application is stable the additional overhead can be removed by simply commenting out the configASSERT() definition in FreeRTOSConfig.h.

```
/* Define configASSERT() to call vAssertCalled() if the assertion fails. The assertion has failed if the value of the parameter passed into configASSERT() equals zero. */  
#define configASSERT( x ) if( ( x ) == 0 ) vAssertCalled( __FILE__, __LINE__ )
```

If running FreeRTOS under the control of a debugger, then configASSERT() can be defined to just disable interrupts and sit in a loop, as demonstrated below. That will have the effect of stopping the code on the line that failed the assert test - pausing the debugger will then immediately take you to the offending line so you can see why it failed.

```
/* Define configASSERT() to disable interrupts and sit in a loop. */
```

General|Include Application
Defined Privileged Functions

- Enabled
- Disabled

```
#define configASSERT( ( x ) ) if(
( x ) == 0 ) {
taskDISABLE_INTERRUPTS();
for( ;; ); }
```

Include Application Defined Privileged Functions is only used by FreeRTOS MPU. If Include Application Defined Privileged Functions is Enabled then the application writer must provide a header file called "application_defined_privileged_functions.h", in which functions the application writer needs to execute in privileged mode can be implemented. Note that, despite having a .h extension, the header file should contain the implementation of the C functions, not just the functions' prototypes.

Functions implemented in "application_defined_privileged_functions.h" must save and restore the processor's privilege state using the prvRaisePrivilege() function and portRESET_PRIVILEGE() macro respectively. For example, if a library provided print function accesses RAM that is outside of the control of the application writer, and therefore cannot be allocated to a memory protected user mode task, then the print function can be encapsulated in a privileged function using the following code:

```
void MPU_debug_printf( const
char *pcMessage )
{
/* State the privilege level of
the processor when the
function was called. */
 BaseType_t xRunningPrivileged
= prvRaisePrivilege();

/* Call the library function,
which now has access to all
RAM. */
debug_printf( pcMessage );
```

```

/* Reset the processor privilege
level to its original value. */
portRESET_PRIVILEGE(
xRunningPrivileged );
}

```

This technique should only be use during development, and not deployment, as it circumvents the memory protection.

Optional Functions vTaskPrioritySet() Function	• Enabled • Disabled	Include vTaskPrioritySet() function in build
Optional Functions uxTaskPriorityGet() Function	• Enabled • Disabled	Include uxTaskPriorityGet() function in build
Optional Functions vTaskDelete() Function	• Enabled • Disabled	Include vTaskDelete() function in build
Optional Functions vTaskSuspend() Function	• Enabled • Disabled	Include vTaskSuspend() function in build
Optional Functions xResumeFromISR() Function	• Enabled • Disabled	Include xResumeFromISR() function in build
Optional Functions vTaskDelayUntil() Function	• Enabled • Disabled	Include vTaskDelayUntil() function in build
Optional Functions vTaskDelay() Function	• Enabled • Disabled	Include vTaskDelay() function in build
Optional Functions xTaskGetSchedulerState() Function	• Enabled • Disabled	Include xTaskGetSchedulerState() function in build
Optional Functions xTaskGetCurrentTaskHandle() Function	• Enabled • Disabled	Include xTaskGetCurrentTaskHandle() function in build
Optional Functions uxTaskGetStackHighWaterMark() Function	• Enabled • Disabled	Include uxTaskGetStackHighWaterMark() function in build
Optional Functions xTaskGetIdleTaskHandle() Function	• Enabled • Disabled	Include xTaskGetIdleTaskHandle() function in build
Optional Functions eTaskGetState() Function	• Enabled • Disabled	Include eTaskGetState() function in build
Optional Functions xEventGroup	• Enabled	Include

pSetBitFromISR() Function	• Disabled	xEventGroupSetBitFromISR() function in build
Optional Functions xTimerPendFunctionCall() Function	• Enabled • Disabled	Include xTimerPendFunctionCall() function in build
Optional Functions xTaskAbortDelay() Function	• Enabled • Disabled	Include xTaskAbortDelay() function in build
Optional Functions xTaskGetHandle() Function	• Enabled • Disabled	Include xTaskGetHandle() function in build
Optional Functions xTaskResumeFromISR() Function	• Enabled • Disabled	Include xTaskResumeFromISR() function in build
RA Hardware Stack Monitor	• Enabled • Disabled	Include RA stack monitor

4.2.44 Crypto Middleware (rm_psa_crypto)

Modules

Functions

`fsp_err_t` [RM_PSA_CRYPTO_TRNG_Read](#) (uint8_t *const p_rngbuf, uint32_t num_req_bytes, uint32_t *p_num_gen_bytes)

Reads requested length of random data from the TRNG. Generate nbytes of random bytes and store them in p_rngbuf buffer. [More...](#)

`int` [mbedtls_platform_setup](#) (mbedtls_platform_context *ctx)

`void` [mbedtls_platform_teardown](#) (mbedtls_platform_context *ctx)

Detailed Description

Hardware acceleration for the mbedCrypto implementation of the ARM PSA Crypto API.

Overview

Note

The PSA Crypto module does not provide any interfaces to the user. This release uses the mbed-Crypto version 1.1.0 which conforms to the PSA Crypto API 1.0 beta2 specification. Consult the ARM mbedCrypto documentation at https://github.com/ARMmbed/mbed-crypto/blob/mbedcrypto-1.1.0/docs/getting_started.md for further information.

Features

The PSA_Crypto module provides hardware support for the following PSA Crypto operations

- SHA256 calculation
- SHA224 calculation
- AES128/256.
 - Plain-Text Key generation
 - Encryption with no padding and with PKCS7 padding.
 - Decryption
 - CBC and CTR modes
- RSA2048
 - Plain-Text Key Generation
 - Signing
 - Verification
- Random number generation

Configuration

Build Time Configurations for mbedCrypto

The following build time configurations are defined in fsp_cfg/mbedtls/config.h:

Configuration	Options	Description
Hardware Acceleration TRNG	Enabled	Defines MBEDTLS_ENTROPY_HARDWARE_ALT.
Hardware Acceleration Hash SHA256/224	MCU Specific Options	Defines MBEDTLS_SHA256_ALT and MBEDTLS_SHA256_PROCESS_ALT.
Hardware Acceleration Cipher AES	MCU Specific Options	Defines MBEDTLS_AES_SETKEY_ENC_ALT, MBEDTLS_AES_SETKEY_DEC_ALT, MBEDTLS_AES_ENCRYPT_ALT and MBEDTLS_AES_DECRYPT_ALT
Hardware Acceleration Public Key Cryptography (PKC) RSA	MCU Specific Options	Defines MBEDTLS_RSA_ALT.
Hardware Acceleration Secure Crypto Engine Initialization	Enabled	MBEDTLS_PLATFORM_SETUP_TEAR_DOWN_ALT
Platform MBEDTLS_HAVE_ASM	• Define • Undefine	MBEDTLS_HAVE_ASM
Platform MBEDTLS_NO_UDBL_DIVISION	• Define • Undefine	MBEDTLS_NO_UDBL_DIVISION
Platform MBEDTLS_NO_64BIT_MULTIPLICATION	• Define • Undefine	MBEDTLS_NO_64BIT_MULTIPLICATION
Platform MBEDTLS_HAVE_SSE2	• Define • Undefine	MBEDTLS_HAVE_SSE2
Platform MBEDTLS_HAVE_TIME	• Define • Undefine	MBEDTLS_HAVE_TIME
Platform MBEDTLS_HAVE_TIME_DATE	• Define	MBEDTLS_HAVE_TIME_DATE

DATE	• Undefine	
Platform MBEDTLS_PLATFORM_MEMORY	• Define • Undefine	MBEDTLS_PLATFORM_MEMORY
Platform MBEDTLS_PLATFORM_NO_STD_FUNCTIONS	• Define • Undefine	MBEDTLS_PLATFORM_NO_STD_FUNCTIONS
Platform Alternate MBEDTLS_PLATFORM_EXIT_ALT	• Define • Undefine	MBEDTLS_PLATFORM_EXIT_ALT
Platform Alternate MBEDTLS_PLATFORM_TIME_ALT	• Define • Undefine	MBEDTLS_PLATFORM_TIME_ALT
Platform Alternate MBEDTLS_PLATFORM_FPRINTF_ALT	• Define • Undefine	MBEDTLS_PLATFORM_FPRINTF_ALT
Platform Alternate MBEDTLS_PLATFORM_PRINTF_ALT	• Define • Undefine	MBEDTLS_PLATFORM_PRINTF_ALT
Platform Alternate MBEDTLS_PLATFORM_SNPRINTF_ALT	• Define • Undefine	MBEDTLS_PLATFORM_SNPRINTF_ALT
Platform Alternate MBEDTLS_PLATFORM_VSNPRINTF_ALT	• Define • Undefine	MBEDTLS_PLATFORM_VSNPRINTF_ALT
Platform Alternate MBEDTLS_PLATFORM_NV_SEED_ALT	• Define • Undefine	MBEDTLS_PLATFORM_NV_SEED_ALT
General MBEDTLS_DEPRECATED_WARNING	• Define • Undefine	MBEDTLS_DEPRECATED_WARNING
General MBEDTLS_DEPRECATED_REMOVED	• Define • Undefine	MBEDTLS_DEPRECATED_REMOVED
General MBEDTLS_CHECK_PARAMS	• Define • Undefine	MBEDTLS_CHECK_PARAMS
Platform MBEDTLS_TIMING_ALT	• Define • Undefine	MBEDTLS_TIMING_ALT
Cipher Alternate MBEDTLS_AES_ALT	• Define • Undefine	MBEDTLS_AES_ALT
Cipher Alternate MBEDTLS_ARC4_ALT	• Define • Undefine	MBEDTLS_ARC4_ALT
Cipher Alternate MBEDTLS_ARIA_ALT	• Define • Undefine	MBEDTLS_ARIA_ALT
Cipher Alternate MBEDTLS_BLOWFISH_ALT	• Define • Undefine	MBEDTLS_BLOWFISH_ALT
Cipher Alternate MBEDTLS_CAMELLIA_ALT	• Define • Undefine	MBEDTLS_CAMELLIA_ALT
Cipher Alternate MBEDTLS_CCM_ALT	• Define • Undefine	MBEDTLS_CCM_ALT
Cipher Alternate MBEDTLS_CHACHA20_ALT	• Define • Undefine	MBEDTLS_CHACHA20_ALT
Cipher Alternate MBEDTLS_CHACHA20POLY_ALT	• Define	MBEDTLS_CHACHA20POLY_ALT

CHAPOLY_ALT	• Undefine	
Cipher Alternate MBEDTLS_CMAC_ALT	• Define • Undefine	MBEDTLS_CMAC_ALT
Cipher Alternate MBEDTLS_DES_ALT	• Define • Undefine	MBEDTLS_DES_ALT
Public Key Cryptography (PKC) DHM Alternate MBEDTLS_DHM_ALT	• Define • Undefine	MBEDTLS_DHM_ALT
Public Key Cryptography (PKC) ECC Alternate MBEDTLS_ECJPAKE_ALT	• Define • Undefine	MBEDTLS_ECJPAKE_ALT
Cipher Alternate MBEDTLS_GCM_ALT	• Define • Undefine	MBEDTLS_GCM_ALT
Cipher Alternate MBEDTLS_NIST_KW_ALT	• Define • Undefine	MBEDTLS_NIST_KW_ALT
Hash Alternate MBEDTLS_MD2_ALT	• Define • Undefine	MBEDTLS_MD2_ALT
Hash Alternate MBEDTLS_MD4_ALT	• Define • Undefine	MBEDTLS_MD4_ALT
Hash Alternate MBEDTLS_MD5_ALT	• Define • Undefine	MBEDTLS_MD5_ALT
Message Authentication Code (MAC) Alternate MBEDTLS_POLY1305_ALT	• Define • Undefine	MBEDTLS_POLY1305_ALT
Hash Alternate MBEDTLS_RIPEMD160_ALT	• Define • Undefine	MBEDTLS_RIPEMD160_ALT
Hash Alternate MBEDTLS_SHA1_ALT	• Define • Undefine	MBEDTLS_SHA1_ALT
Hash Alternate MBEDTLS_SHA512_ALT	• Define • Undefine	MBEDTLS_SHA512_ALT
Cipher Alternate MBEDTLS_XTEA_ALT	• Define • Undefine	MBEDTLS_XTEA_ALT
Public Key Cryptography (PKC) ECC Alternate MBEDTLS_ECP_ALT	• Define • Undefine	MBEDTLS_ECP_ALT
Hash Alternate MBEDTLS_MD2_PROCESS_ALT	• Define • Undefine	MBEDTLS_MD2_PROCESS_ALT
Hash Alternate MBEDTLS_MD4_PROCESS_ALT	• Define • Undefine	MBEDTLS_MD4_PROCESS_ALT
Hash Alternate MBEDTLS_MD5_PROCESS_ALT	• Define • Undefine	MBEDTLS_MD5_PROCESS_ALT
Hash Alternate MBEDTLS_RIPEMD160_PROCESS_ALT	• Define • Undefine	MBEDTLS_RIPEMD160_PROCESS_ALT

Hash Alternate MBEDTLS_SHA1_PROCESS_ALT	• Define • Undefine	MBEDTLS_SHA1_PROCESS_ALT
Hash Alternate MBEDTLS_SHA512_PROCESS_ALT	• Define • Undefine	MBEDTLS_SHA512_PROCESS_ALT
Cipher Alternate MBEDTLS_DES_SETKEY_ALT	• Define • Undefine	MBEDTLS_DES_SETKEY_ALT
Cipher Alternate MBEDTLS_DES_CRYPT_ECB_ALT	• Define • Undefine	MBEDTLS_DES_CRYPT_ECB_ALT
Cipher Alternate MBEDTLS_DES3_CRYPT_ECB_ALT	• Define • Undefine	MBEDTLS_DES3_CRYPT_ECB_ALT
Public Key Cryptography (PKC) ECC MBEDTLS_ECDH_GEN_PUBLIC_ALT	• Define • Undefine	MBEDTLS_ECDH_GEN_PUBLIC_ALT
Public Key Cryptography (PKC) ECC MBEDTLS_ECDH_COMPUTE_SHARED_ALT	• Define • Undefine	MBEDTLS_ECDH_COMPUTE_SHARED_ALT
Public Key Cryptography (PKC) ECC Alternate MBEDTLS_ECDSA_VERIFY_ALT	• Define • Undefine	MBEDTLS_ECDSA_VERIFY_ALT
Public Key Cryptography (PKC) ECC Alternate MBEDTLS_ECDSA_SIGN_ALT	• Define • Undefine	MBEDTLS_ECDSA_SIGN_ALT
Public Key Cryptography (PKC) ECC Alternate MBEDTLS_ECDSA_GENKEY_ALT	• Define • Undefine	MBEDTLS_ECDSA_GENKEY_ALT
Public Key Cryptography (PKC) ECC Alternate MBEDTLS_ECDSA_GENKEY_ALT	• Define • Undefine	MBEDTLS_ECDSA_GENKEY_ALT
Public Key Cryptography (PKC) ECC Alternate MBEDTLS_ECP_INTERNAL_ALT	• Define • Undefine	MBEDTLS_ECP_INTERNAL_ALT
Public Key Cryptography (PKC) ECC Alternate MBEDTLS_ECP_RANDOMIZE_JAC_ALT	• Define • Undefine	MBEDTLS_ECP_RANDOMIZE_JAC_ALT
Public Key Cryptography (PKC) ECC Alternate MBEDTLS_ECP_ADD_MIXED_ALT	• Define • Undefine	MBEDTLS_ECP_ADD_MIXED_ALT
Public Key Cryptography (PKC) ECC Alternate MBEDTLS_ECP_DOUBLE_JAC_ALT	• Define • Undefine	MBEDTLS_ECP_DOUBLE_JAC_ALT
Public Key Cryptography (PKC) ECC Alternate MBEDTLS_ECP_NORMALIZE_JAC_MANY_ALT	• Define • Undefine	MBEDTLS_ECP_NORMALIZE_JAC_MANY_ALT
Public Key Cryptography (PKC) ECC Alternate MBEDTLS_ECP_NORMALIZE_JAC_ALT	• Define • Undefine	MBEDTLS_ECP_NORMALIZE_JAC_ALT

ORMALIZE_JAC_ALT

Public Key Cryptography (PKC)|
ECC|Alternate|MBEDTLS_ECP_D
OUBLE_ADD_MXZ_ALT

- Define
- Undefine

MBEDTLS_ECP_DOUBLE_ADD_M
XZ_ALT

Public Key Cryptography (PKC)|
ECC|Alternate|MBEDTLS_ECP_R
ANDOMIZE_MXZ_ALT

- Define
- Undefine

MBEDTLS_ECP_RANDOMIZE_MX
Z_ALT

Public Key Cryptography (PKC)|
ECC|Alternate|MBEDTLS_ECP_N
ORMALIZE_MXZ_ALT

- Define
- Undefine

MBEDTLS_ECP_NORMALIZE_MX
Z_ALT

RNG|MBEDTLS_TEST_NULL_ENT
ROPY

- Define
- Undefine

MBEDTLS_TEST_NULL_ENTROPY

Cipher|AES|MBEDTLS_AES_ROM
_TABLES

- Define
- Undefine

MBEDTLS_AES_ROM_TABLES

Cipher|AES|MBEDTLS_AES_FEW
ER_TABLES

- Define
- Undefine

MBEDTLS_AES_FEWER_TABLES

Cipher|MBEDTLS_CAMELLIA_SM
ALL_MEMORY

- Define
- Undefine

MBEDTLS_CAMELLIA_SMALL_ME
MORY

Cipher|MBEDTLS_CIPHER_MODE
_CBC

- Define
- Undefine

MBEDTLS_CIPHER_MODE_CBC

Cipher|MBEDTLS_CIPHER_MODE
_CFB

- Define
- Undefine

MBEDTLS_CIPHER_MODE_CFB

Cipher|MBEDTLS_CIPHER_MODE
_CTR

- Define
- Undefine

MBEDTLS_CIPHER_MODE_CTR

Cipher|MBEDTLS_CIPHER_MODE
_OFB

- Define
- Undefine

MBEDTLS_CIPHER_MODE_OFB

Cipher|MBEDTLS_CIPHER_MODE
_XTS

- Define
- Undefine

MBEDTLS_CIPHER_MODE_XTS

Cipher|MBEDTLS_CIPHER_NULL_
CIPHER

- Define
- Undefine

MBEDTLS_CIPHER_NULL_CIPHER

Cipher|MBEDTLS_CIPHER_PADDI
NG_PKCS7

- Define
- Undefine

MBEDTLS_CIPHER_PADDING_PK
CS7

Cipher|MBEDTLS_CIPHER_PADDI
NG_ONE_AND_ZEROS

- Define
- Undefine

MBEDTLS_CIPHER_PADDING_ON
E_AND_ZEROS

Cipher|MBEDTLS_CIPHER_PADDI
NG_ZEROS_AND_LEN

- Define
- Undefine

MBEDTLS_CIPHER_PADDING_ZE
ROS_AND_LEN

Cipher|MBEDTLS_CIPHER_PADDI
NG_ZEROS

- Define
- Undefine

MBEDTLS_CIPHER_PADDING_ZE
ROS

Public Key Cryptography (PKC)|
ECC|Curves|MBEDTLS_ECP_DP_
SECP192R1_ENABLED

- Define
- Undefine

MBEDTLS_ECP_DP_SECP192R1_
ENABLED

Public Key Cryptography (PKC)|
ECC|Curves|MBEDTLS_ECP_DP_
SECP224R1_ENABLED

- Define
- Undefine

MBEDTLS_ECP_DP_SECP224R1_
ENABLED

SECP224R1_ENABLED

Public Key Cryptography (PKC)|
ECC|Curves|MBEDTLS_ECP_DP_
SECP256R1_ENABLED

- Define
- Undefine

MBEDTLS_ECP_DP_SECP256R1_
ENABLED

Public Key Cryptography (PKC)|
ECC|Curves|MBEDTLS_ECP_DP_
SECP384R1_ENABLED

- Define
- Undefine

MBEDTLS_ECP_DP_SECP384R1_
ENABLED

Public Key Cryptography (PKC)|
ECC|Curves|MBEDTLS_ECP_DP_
SECP521R1_ENABLED

- Define
- Undefine

MBEDTLS_ECP_DP_SECP521R1_
ENABLED

Public Key Cryptography (PKC)|
ECC|Curves|MBEDTLS_ECP_DP_
SECP192K1_ENABLED

- Define
- Undefine

MBEDTLS_ECP_DP_SECP192K1_
ENABLED

Public Key Cryptography (PKC)|
ECC|Curves|MBEDTLS_ECP_DP_
SECP224K1_ENABLED

- Define
- Undefine

MBEDTLS_ECP_DP_SECP224K1_
ENABLED

Public Key Cryptography (PKC)|
ECC|Curves|MBEDTLS_ECP_DP_
SECP256K1_ENABLED

- Define
- Undefine

MBEDTLS_ECP_DP_SECP256K1_
ENABLED

Public Key Cryptography (PKC)|
ECC|Curves|MBEDTLS_ECP_DP_
BP256R1_ENABLED

- Define
- Undefine

MBEDTLS_ECP_DP_BP256R1_EN
ABLED

Public Key Cryptography (PKC)|
ECC|Curves|MBEDTLS_ECP_DP_
BP384R1_ENABLED

- Define
- Undefine

MBEDTLS_ECP_DP_BP384R1_EN
ABLED

Public Key Cryptography (PKC)|
ECC|Curves|MBEDTLS_ECP_DP_
BP512R1_ENABLED

- Define
- Undefine

MBEDTLS_ECP_DP_BP512R1_EN
ABLED

Public Key Cryptography (PKC)|
ECC|Curves|MBEDTLS_ECP_DP_
CURVE25519_ENABLED

- Define
- Undefine

MBEDTLS_ECP_DP_CURVE25519_
ENABLED

Public Key Cryptography (PKC)|
ECC|Curves|MBEDTLS_ECP_DP_
CURVE448_ENABLED

- Define
- Undefine

MBEDTLS_ECP_DP_CURVE448_E
NABLED

Public Key Cryptography (PKC)|
ECC|MBEDTLS_ECP_NIST_OPTIM

- Define
- Undefine

MBEDTLS_ECP_NIST_OPTIM

Public Key Cryptography (PKC)|
ECC|MBEDTLS_ECP_RESTARTAB
LE

- Define
- Undefine

MBEDTLS_ECP_RESTARTABLE

Public Key Cryptography (PKC)|
ECC|MBEDTLS_ECDH_LEGACY_C
ONTEXT

- Define
- Undefine

MBEDTLS_ECDH_LEGACY_CONT
EXT

Public Key Cryptography (PKC)|
ECC|MBEDTLS_ECDSA_DETERMI
NISTIC

- Define
- Undefine

MBEDTLS_ECDSA_DETERMINIST
IC

Public Key Cryptography (PKC) ECC MBEDTLS_PK_PARSE_EC_EXTENDED	• Define • Undefine	MBEDTLS_PK_PARSE_EC_EXTENDED
General MBEDTLS_ERROR_STRERROR_DUMMY	• Define • Undefine	MBEDTLS_ERROR_STRERROR_DUMMY
Public Key Cryptography (PKC) MBEDTLS_GENPRIME	• Define • Undefine	MBEDTLS_GENPRIME
Storage MBEDTLS_FS_IO	• Define • Undefine	MBEDTLS_FS_IO
RNG MBEDTLS_NO_DEFAULT_ENTROPY_SOURCES	• Define • Undefine	MBEDTLS_NO_DEFAULT_ENTROPY_SOURCES
Platform MBEDTLS_NO_PLATFORM_ENTROPY	• Define • Undefine	MBEDTLS_NO_PLATFORM_ENTROPY
RNG MBEDTLS_ENTROPY_FORCE_SHA256	• Define • Undefine	MBEDTLS_ENTROPY_FORCE_SHA256
RNG MBEDTLS_ENTROPY_NV_SEED	• Define • Undefine	MBEDTLS_ENTROPY_NV_SEED
Storage MBEDTLS_PSA_CRYPTOKEY_FILE_ID_ENCODES_OWNER	• Define • Undefine	MBEDTLS_PSA_CRYPTOKEY_FILE_ID_ENCODES_OWNER
General MBEDTLS_MEMORY_DEBUG	• Define • Undefine	MBEDTLS_MEMORY_DEBUG
General MBEDTLS_MEMORY_BACKTRACE	• Define • Undefine	MBEDTLS_MEMORY_BACKTRACE
Public Key Cryptography (PKC) RSA MBEDTLS_PK_RSA_ALT_SUPPORT	• Define • Undefine	MBEDTLS_PK_RSA_ALT_SUPPORT
Public Key Cryptography (PKC) MBEDTLS_PKCS1_V15	• Define • Undefine	MBEDTLS_PKCS1_V15
Public Key Cryptography (PKC) MBEDTLS_PKCS1_V21	• Define • Undefine	MBEDTLS_PKCS1_V21
General MBEDTLS_PSA_CRYPTOSPMP	• Define • Undefine	MBEDTLS_PSA_CRYPTOSPMP
RNG MBEDTLS_PSA_INJECT_ENTROPY	• Define • Undefine	MBEDTLS_PSA_INJECT_ENTROPY
Public Key Cryptography (PKC) RSA MBEDTLS_RSA_NO_CRT	• Define • Undefine	MBEDTLS_RSA_NO_CRT
General MBEDTLS_SELF_TEST	• Define • Undefine	MBEDTLS_SELF_TEST
Hash MBEDTLS_SHA256_SMALLER	• Define • Undefine	MBEDTLS_SHA256_SMALLER
General MBEDTLS_THREADING_ALT	• Define • Undefine	MBEDTLS_THREADING_ALT

General MBEDTLS_THREADING_PTHREAD	• Define • Undefine	MBEDTLS_THREADING_PTHREAD
General MBEDTLS_USE_PSA_CRYPTO	• Define • Undefine	MBEDTLS_USE_PSA_CRYPTO
General MBEDTLS_VERSION_FEATURES	• Define • Undefine	MBEDTLS_VERSION_FEATURES
Platform MBEDTLS_AESNI_C	• Define • Undefine	MBEDTLS_AESNI_C
Cipher MBEDTLS_AES_C	Define	MBEDTLS_AES_C
Cipher MBEDTLS_ARC4_C	• Define • Undefine	MBEDTLS_ARC4_C
Public Key Cryptography (PKC) MBEDTLS_ASN1_PARSE_C	• Define • Undefine	MBEDTLS_ASN1_PARSE_C
Public Key Cryptography (PKC) MBEDTLS_ASN1_WRITE_C	• Define • Undefine	MBEDTLS_ASN1_WRITE_C
Public Key Cryptography (PKC) MBEDTLS_BASE64_C	• Define • Undefine	MBEDTLS_BASE64_C
Public Key Cryptography (PKC) MBEDTLS_BIGNUM_C	• Define • Undefine	MBEDTLS_BIGNUM_C
Cipher MBEDTLS_BLOWFISH_C	• Define • Undefine	MBEDTLS_BLOWFISH_C
Cipher MBEDTLS_CAMELLIA_C	• Define • Undefine	MBEDTLS_CAMELLIA_C
Cipher MBEDTLS_ARIA_C	• Define • Undefine	MBEDTLS_ARIA_C
Cipher MBEDTLS_CCM_C	• Define • Undefine	MBEDTLS_CCM_C
Cipher MBEDTLS_CHACHA20_C	• Define • Undefine	MBEDTLS_CHACHA20_C
Cipher MBEDTLS_CHACHAPOLY_C	• Define • Undefine	MBEDTLS_CHACHAPOLY_C
Cipher MBEDTLS_CIPHER_C	• Define • Undefine	MBEDTLS_CIPHER_C
Message Authentication Code (MAC) MBEDTLS_CMAC_C	• Define • Undefine	MBEDTLS_CMAC_C
RNG MBEDTLS_CTR_DRBG_C	• Define • Undefine	MBEDTLS_CTR_DRBG_C
Cipher MBEDTLS_DES_C	• Define • Undefine	MBEDTLS_DES_C
Public Key Cryptography (PKC) DHM MBEDTLS_DHM_C	• Define • Undefine	MBEDTLS_DHM_C
Public Key Cryptography	• Define	MBEDTLS_ECDH_C

(PKC) ECC MBEDTLS_ECDH_C	• Undefine	
Public Key Cryptography (PKC) ECC MBEDTLS_ECDSA_C	• Define • Undefine	MBEDTLS_ECDSA_C
Public Key Cryptography (PKC) ECC MBEDTLS_ECJPAKE_C	• Define • Undefine	MBEDTLS_ECJPAKE_C
Public Key Cryptography (PKC) ECC MBEDTLS_ECP_C	• Define • Undefine	MBEDTLS_ECP_C
Platform MBEDTLS_ENTROPY_C	• Define • Undefine	MBEDTLS_ENTROPY_C
General MBEDTLS_ERROR_C	• Define • Undefine	MBEDTLS_ERROR_C
Cipher MBEDTLS_GCM_C	• Define • Undefine	MBEDTLS_GCM_C
RNG MBEDTLS_HAVEGE_C	• Define • Undefine	MBEDTLS_HAVEGE_C
Message Authentication Code (MAC) MBEDTLS_HKDF_C	• Define • Undefine	MBEDTLS_HKDF_C
Message Authentication Code (MAC) MBEDTLS_HMAC_DRBG_C	• Define • Undefine	MBEDTLS_HMAC_DRBG_C
Cipher MBEDTLS_NIST_KW_C	• Define • Undefine	MBEDTLS_NIST_KW_C
Hash MBEDTLS_MD_C	• Define • Undefine	MBEDTLS_MD_C
Hash MBEDTLS_MD2_C	• Define • Undefine	MBEDTLS_MD2_C
Hash MBEDTLS_MD4_C	• Define • Undefine	MBEDTLS_MD4_C
Hash MBEDTLS_MD5_C	• Define • Undefine	MBEDTLS_MD5_C
General MBEDTLS_MEMORY_BUFFER_ALLOC_C	• Define • Undefine	MBEDTLS_MEMORY_BUFFER_ALLOC_C
Public Key Cryptography (PKC) MBEDTLS_OID_C	• Define • Undefine	MBEDTLS_OID_C
Cipher MBEDTLS_PADLOCK_C	• Define • Undefine	MBEDTLS_PADLOCK_C
Public Key Cryptography (PKC) MBEDTLS_PEM_PARSE_C	• Define • Undefine	MBEDTLS_PEM_PARSE_C
Public Key Cryptography (PKC) MBEDTLS_PEM_WRITE_C	• Define • Undefine	MBEDTLS_PEM_WRITE_C
Public Key Cryptography (PKC) MBEDTLS_PK_C	• Define • Undefine	MBEDTLS_PK_C

Public Key Cryptography (PKC) MBEDTLS_PK_PARSE_C	• Define • Undefine	MBEDTLS_PK_PARSE_C
Public Key Cryptography (PKC) MBEDTLS_PK_WRITE_C	• Define • Undefine	MBEDTLS_PK_WRITE_C
Public Key Cryptography (PKC) MBEDTLS_PKCS5_C	• Define • Undefine	MBEDTLS_PKCS5_C
Public Key Cryptography (PKC) MBEDTLS_PKCS12_C	• Define • Undefine	MBEDTLS_PKCS12_C
Platform MBEDTLS_PLATFORM_C	• Define • Undefine	MBEDTLS_PLATFORM_C
Message Authentication Code (MAC) MBEDTLS_POLY1305_C	• Define • Undefine	MBEDTLS_POLY1305_C
General MBEDTLS_PSA_CRYPTOC	• Define • Undefine	MBEDTLS_PSA_CRYPTOC
Storage MBEDTLS_PSA_CRYPTOSTORAGE_C	• Define • Undefine	MBEDTLS_PSA_CRYPTOSTORAGE_C
Storage MBEDTLS_PSA_ITS_FILE_C	• Define • Undefine	MBEDTLS_PSA_ITS_FILE_C
Hash MBEDTLS_RIPEMD160_C	• Define • Undefine	MBEDTLS_RIPEMD160_C
Public Key Cryptography (PKC) RSA MBEDTLS_RSA_C	• Define • Undefine	MBEDTLS_RSA_C
Hash MBEDTLS_SHA1_C	• Define • Undefine	MBEDTLS_SHA1_C
Hash MBEDTLS_SHA256_C	• Define • Undefine	MBEDTLS_SHA256_C
Hash MBEDTLS_SHA512_C	• Define • Undefine	MBEDTLS_SHA512_C
General MBEDTLS_THREADING_C	• Define • Undefine	MBEDTLS_THREADING_C
General MBEDTLS_TIMING_C	• Define • Undefine	MBEDTLS_TIMING_C
General MBEDTLS_VERSION_C	• Define • Undefine	MBEDTLS_VERSION_C
Cipher MBEDTLS_XTEA_C	• Define • Undefine	MBEDTLS_XTEA_C
Public Key Cryptography (PKC) MBEDTLS_MPI_WINDOW_SIZE	• Define • Undefine	MBEDTLS_MPI_WINDOW_SIZE
Public Key Cryptography (PKC) MBEDTLS_MPI_WINDOW_SIZE value	Configurable String	MBEDTLS_MPI_WINDOW_SIZE value
Public Key Cryptography	• Define	MBEDTLS_MPI_MAX_SIZE

(PKC) MBEDTLS_MPI_MAX_SIZE	• Undefine	
Public Key Cryptography (PKC) MBEDTLS_MPI_MAX_SIZE value	Configurable String	MBEDTLS_MPI_MAX_SIZE value
RNG MBEDTLS_CTR_DRBG_ENTROPY_LEN	• Define • Undefine	RNG MBEDTLS_CTR_DRBG_ENTROPY_LEN
RNG MBEDTLS_CTR_DRBG_ENTROPY_LEN value	Configurable String	RNG value MBEDTLS_CTR_DRBG_ENTROPY_LEN
RNG MBEDTLS_CTR_DRBG_RESEED_INTERVAL	• Define • Undefine	RNG MBEDTLS_CTR_DRBG_RESEED_INTERVAL
RNG MBEDTLS_CTR_DRBG_RESEED_INTERVAL value	Configurable String	RNG value MBEDTLS_CTR_DRBG_RESEED_INTERVAL
RNG MBEDTLS_CTR_DRBG_MAX_INPUT	• Define • Undefine	MBEDTLS_CTR_DRBG_MAX_INPUT
RNG MBEDTLS_CTR_DRBG_MAX_INPUT value	Configurable String	MBEDTLS_CTR_DRBG_MAX_INPUT value
RNG MBEDTLS_CTR_DRBG_MAX_REQUEST	• Define • Undefine	MBEDTLS_CTR_DRBG_MAX_REQUEST
RNG MBEDTLS_CTR_DRBG_MAX_REQUEST value	Configurable String	MBEDTLS_CTR_DRBG_MAX_REQUEST value
RNG MBEDTLS_CTR_DRBG_MAX_SEED_INPUT	• Define • Undefine	MBEDTLS_CTR_DRBG_MAX_SEED_INPUT
RNG MBEDTLS_CTR_DRBG_MAX_SEED_INPUT value	Configurable String	MBEDTLS_CTR_DRBG_MAX_SEED_INPUT value
RNG MBEDTLS_CTR_DRBG_USE_128_BIT_KEY	• Define • Undefine	MBEDTLS_CTR_DRBG_USE_128_BIT_KEY
RNG MBEDTLS_HMAC_DRBG_RESEED_INTERVAL	• Define • Undefine	MBEDTLS_HMAC_DRBG_RESEED_INTERVAL
RNG MBEDTLS_HMAC_DRBG_RESEED_INTERVAL value	Configurable String	MBEDTLS_HMAC_DRBG_RESEED_INTERVAL value
RNG MBEDTLS_HMAC_DRBG_MAX_INPUT	• Define • Undefine	MBEDTLS_HMAC_DRBG_MAX_INPUT
RNG MBEDTLS_HMAC_DRBG_MAX_INPUT value	Configurable String	MBEDTLS_HMAC_DRBG_MAX_INPUT value
RNG MBEDTLS_HMAC_DRBG_MAX_REQUEST	• Define • Undefine	MBEDTLS_HMAC_DRBG_MAX_REQUEST
RNG MBEDTLS_HMAC_DRBG_MAX_REQUEST value	Configurable String	MBEDTLS_HMAC_DRBG_MAX_REQUEST value
RNG MBEDTLS_HMAC_DRBG_MAX_SEED_INPUT	• Define • Undefine	MBEDTLS_HMAC_DRBG_MAX_SEED_INPUT
RNG MBEDTLS_HMAC_DRBG_MAX_SEED_INPUT value	Configurable String	MBEDTLS_HMAC_DRBG_MAX_SEED_INPUT value

Public Key Cryptography (PKC) ECC MBEDTLS_ECP_MAX_BITS	• Define • Undefine	MBEDTLS_ECP_MAX_BITS
Public Key Cryptography (PKC) ECC MBEDTLS_ECP_MAX_BITS value	Configurable String	MBEDTLS_ECP_MAX_BITS value
Public Key Cryptography (PKC) ECC MBEDTLS_ECP_WINDOW_SIZE	• Define • Undefine	MBEDTLS_ECP_WINDOW_SIZE
Public Key Cryptography (PKC) ECC MBEDTLS_ECP_WINDOW_SIZE value	Configurable String	MBEDTLS_ECP_WINDOW_SIZE value
Public Key Cryptography (PKC) ECC MBEDTLS_ECP_FIXED_POINT_OPTIM	• Define • Undefine	MBEDTLS_ECP_FIXED_POINT_OPTIM
Public Key Cryptography (PKC) ECC MBEDTLS_ECP_FIXED_POINT_OPTIM value	Configurable String	MBEDTLS_ECP_FIXED_POINT_OPTIM value
RNG MBEDTLS_ENTROPY_MAX_SOURCES	• Define • Undefine	MBEDTLS_ENTROPY_MAX_SOURCES
RNG MBEDTLS_ENTROPY_MAX_SOURCES value	Configurable String	MBEDTLS_ENTROPY_MAX_SOURCES value
RNG MBEDTLS_ENTROPY_MAX_GATHER	• Define • Undefine	MBEDTLS_ENTROPY_MAX_GATHER
RNG MBEDTLS_ENTROPY_MAX_GATHER value	Configurable String	MBEDTLS_ENTROPY_MAX_GATHER value
RNG MBEDTLS_ENTROPY_MIN_HARDWARE	• Define • Undefine	MBEDTLS_ENTROPY_MIN_HARDWARE
RNG MBEDTLS_ENTROPY_MIN_HARDWARE value	Configurable String	MBEDTLS_ENTROPY_MIN_HARDWARE value
General MBEDTLS_MEMORY_ALIGN_MULTIPLE	• Define • Undefine	MBEDTLS_MEMORY_ALIGN_MULTIPLE
General MBEDTLS_MEMORY_ALIGN_MULTIPLE value	Configurable String	MBEDTLS_MEMORY_ALIGN_MULTIPLE value
Platform MBEDTLS_PLATFORM_STD_CALLOC	• Define • Undefine	MBEDTLS_PLATFORM_STD_CALLOC
Platform MBEDTLS_PLATFORM_STD_CALLOC value	Configurable String	MBEDTLS_PLATFORM_STD_CALLOC value
Platform MBEDTLS_PLATFORM_STD_FREE	• Define • Undefine	MBEDTLS_PLATFORM_STD_FREE
Platform MBEDTLS_PLATFORM_STD_FREE value	Configurable String	MBEDTLS_PLATFORM_STD_FREE value
Platform MBEDTLS_PLATFORM_STD_EXIT	• Define • Undefine	MBEDTLS_PLATFORM_STD_EXIT

Platform MBEDTLS_PLATFORM_STD_EXIT value	Configurable String	MBEDTLS_PLATFORM_STD_EXIT value
Platform MBEDTLS_PLATFORM_STD_TIME	• Define • Undefine	MBEDTLS_PLATFORM_STD_TIME
Platform MBEDTLS_PLATFORM_STD_TIME value	Configurable String	MBEDTLS_PLATFORM_STD_TIME value
Platform MBEDTLS_PLATFORM_STD_FPRINTF	• Define • Undefine	MBEDTLS_PLATFORM_STD_FPRINTF
Platform MBEDTLS_PLATFORM_STD_FPRINTF value	Configurable String	MBEDTLS_PLATFORM_STD_FPRINTF value
Platform MBEDTLS_PLATFORM_STD_PRINTF	• Define • Undefine	MBEDTLS_PLATFORM_STD_PRINTF
Platform MBEDTLS_PLATFORM_STD_PRINTF value	Configurable String	MBEDTLS_PLATFORM_STD_PRINTF value
Platform MBEDTLS_PLATFORM_STD_SNPRINTF	• Define • Undefine	MBEDTLS_PLATFORM_STD_SNPRINTF
Platform MBEDTLS_PLATFORM_STD_SNPRINTF value	Configurable String	MBEDTLS_PLATFORM_STD_SNPRINTF value
Platform MBEDTLS_PLATFORM_STD_EXIT_SUCCESS	• Define • Undefine	MBEDTLS_PLATFORM_STD_EXIT_SUCCESS
Platform MBEDTLS_PLATFORM_STD_EXIT_SUCCESS value	Configurable String	MBEDTLS_PLATFORM_STD_EXIT_SUCCESS value
Platform MBEDTLS_PLATFORM_STD_EXIT_FAILURE	• Define • Undefine	MBEDTLS_PLATFORM_STD_EXIT_FAILURE
Platform MBEDTLS_PLATFORM_STD_EXIT_FAILURE value	Configurable String	MBEDTLS_PLATFORM_STD_EXIT_FAILURE value
Platform MBEDTLS_PLATFORM_STD_NV_SEED_READ	• Define • Undefine	MBEDTLS_PLATFORM_STD_NV_SEED_READ
Platform MBEDTLS_PLATFORM_STD_NV_SEED_READ value	Configurable String	MBEDTLS_PLATFORM_STD_NV_SEED_READ value
Platform MBEDTLS_PLATFORM_STD_NV_SEED_WRITE	• Define • Undefine	MBEDTLS_PLATFORM_STD_NV_SEED_WRITE
Platform MBEDTLS_PLATFORM_STD_NV_SEED_WRITE value	Configurable String	MBEDTLS_PLATFORM_STD_NV_SEED_WRITE value
Platform MBEDTLS_PLATFORM_STD_NV_SEED_FILE	• Define • Undefine	MBEDTLS_PLATFORM_STD_NV_SEED_FILE
Platform MBEDTLS_PLATFORM_STD_NV_SEED_FILE value	Configurable String	MBEDTLS_PLATFORM_STD_NV_SEED_FILE value
Platform MBEDTLS_PLATFORM_CALLOC_MACRO	• Define • Undefine	MBEDTLS_PLATFORM_CALLOC_MACRO
Platform MBEDTLS_PLATFORM_CALLOC_MACRO value	Configurable String	MBEDTLS_PLATFORM_CALLOC_MACRO value

Platform MBEDTLS_PLATFORM_FREE_MACRO	• Define • Undefine	MBEDTLS_PLATFORM_FREE_MACRO
Platform MBEDTLS_PLATFORM_FREE_MACRO value	Configurable String	MBEDTLS_PLATFORM_FREE_MACRO value
Platform MBEDTLS_PLATFORM_EXIT_MACRO	• Define • Undefine	MBEDTLS_PLATFORM_EXIT_MACRO
Platform MBEDTLS_PLATFORM_EXIT_MACRO value	Configurable String	MBEDTLS_PLATFORM_EXIT_MACRO value
Platform MBEDTLS_PLATFORM_TIME_MACRO	• Define • Undefine	MBEDTLS_PLATFORM_TIME_MACRO
Platform MBEDTLS_PLATFORM_TIME_MACRO value	Configurable String	MBEDTLS_PLATFORM_TIME_MACRO value
Platform MBEDTLS_PLATFORM_TIME_TYPE_MACRO	• Define • Undefine	MBEDTLS_PLATFORM_TIME_TYPE_MACRO
Platform MBEDTLS_PLATFORM_TIME_TYPE_MACRO value	Configurable String	MBEDTLS_PLATFORM_TIME_TYPE_MACRO value
Platform MBEDTLS_PLATFORM_FPRINTF_MACRO	• Define • Undefine	MBEDTLS_PLATFORM_FPRINTF_MACRO
Platform MBEDTLS_PLATFORM_FPRINTF_MACRO value	Configurable String	MBEDTLS_PLATFORM_FPRINTF_MACRO value
Platform MBEDTLS_PLATFORM_PRINTF_MACRO	• Define • Undefine	MBEDTLS_PLATFORM_PRINTF_MACRO
Platform MBEDTLS_PLATFORM_PRINTF_MACRO value	Configurable String	MBEDTLS_PLATFORM_PRINTF_MACRO value
Platform MBEDTLS_PLATFORM_SNPRINTF_MACRO	• Define • Undefine	MBEDTLS_PLATFORM_SNPRINTF_MACRO
Platform MBEDTLS_PLATFORM_SNPRINTF_MACRO value	Configurable String	MBEDTLS_PLATFORM_SNPRINTF_MACRO value
Platform MBEDTLS_PLATFORM_VSNPRINTF_MACRO	• Define • Undefine	MBEDTLS_PLATFORM_VSNPRINTF_MACRO
Platform MBEDTLS_PLATFORM_VSNPRINTF_MACRO value	Configurable String	MBEDTLS_PLATFORM_VSNPRINTF_MACRO value
Platform MBEDTLS_PLATFORM_NV_SEED_READ_MACRO	• Define • Undefine	MBEDTLS_PLATFORM_NV_SEED_READ_MACRO
Platform MBEDTLS_PLATFORM_NV_SEED_READ_MACRO value	Configurable String	MBEDTLS_PLATFORM_NV_SEED_READ_MACRO value
Platform MBEDTLS_PLATFORM_NV_SEED_WRITE_MACRO	• Define • Undefine	MBEDTLS_PLATFORM_NV_SEED_WRITE_MACRO
Platform MBEDTLS_PLATFORM_NV_SEED_WRITE_MACRO value	Configurable String	MBEDTLS_PLATFORM_NV_SEED_WRITE_MACRO value
Platform Alternate MBEDTLS_PLATFORM_ZEROIZE_ALT	• Define • Undefine	MBEDTLS_PLATFORM_ZEROIZE_ALT

Platform|Alternate|MBEDTLS_PL
ATFORM_GMTIME_R_ALT

- Define
- Undefine

MBEDTLS_PLATFORM_GMTIME_
R_ALT

4.2.45 Capacitive Touch Middleware (rm_touch)

Modules

Functions

`fsp_err_t` `RM_TOUCH_Open` (`touch_ctrl_t *const p_ctrl`, `touch_cfg_t const *const p_cfg`)

Opens and configures the TOUCH Middle module. Implements `touch_api_t::open`. [More...](#)

`fsp_err_t` `RM_TOUCH_ScanStart` (`touch_ctrl_t *const p_ctrl`)

This function should be called each time a periodic timer expires. [More...](#)

`fsp_err_t` `RM_TOUCH_DataGet` (`touch_ctrl_t *const p_ctrl`, `uint64_t *p_button_status`, `uint16_t *p_slider_position`, `uint16_t *p_wheel_position`)

Gets the 64-bit mask indicating which buttons are pressed. [More...](#)

`fsp_err_t` `RM_TOUCH_Close` (`touch_ctrl_t *const p_ctrl`)

Disables specified TOUCH control block. Implements `transfer_api_t::close`. [More...](#)

`fsp_err_t` `RM_TOUCH_VersionGet` (`fsp_version_t *const p_version`)

Detailed Description

This module supports the Capacitive Touch Sensing Unit (CTSUS). It implements the [Touch Middleware Interface](#).

Overview

This module controls the CTSU API and provides touch buttons, sliders, and wheels. By editing the settings, the user can make various settings for these. The CTSU HAL driver is always required.

Features

`um_touch_slider_5position` `um_touch_button_on_off`

- Supports touch buttons(Self and Mutual), sliders and wheels
- Supports touch buttons(Self and Mutual), sliders, and wheels.
 - The button status shows the status of up to 64 buttons in 64 bitmap.
 - The slider position is in the range of 0 to 100.
 - The Wheel position is in the range of 0 to 360.
- Starts scanning at any time.
 - The scan may be started by a software trigger or an external trigger.
 - The scan completion is signalled by the callback function.
- Gets all results after scans are complete.
- Additional build-time features
 - Optional (build time) support for real-time monitoring function by QE. (Not yet available)

Configuration

Build Time Configurations for rm_touch

The following build time configurations are defined in fsp_cfg/rm_touch_cfg.h:

Configuration	Options	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	If selected code for parameter checking is included in the build.
QE_UPDATE_MONITOR	• Enabled • Disabled	If enabled,
Number of buttons	Name must be a valid C symbol	Number of buttons
Number of sliders	Name must be a valid C symbol	Number of sliders
Number of wheels	Name must be a valid C symbol	Number of wheels

Configurations for TOUCH Driver on rm_touch

This module can be added to the Threads tab from New -> Middleware -> CapTouch -> TOUCH Driver on rm_touch:

4.3 Interfaces

Detailed Description

The FSP interfaces provide APIs for common functionality. They can be implemented by one or more modules. Modules can use other modules as dependencies using this interface layer.

Modules

ADC Interface

Interface for A/D Converters.

CAC Interface

Interface for clock frequency accuracy measurements.

CGC Interface

Interface for clock generation.

Comparator Interface

Interface for comparators.

CRC Interface

Interface for cyclic redundancy checking.

CTSU Interface

Interface for Capacitive Touch Sensing Unit (CTSU) functions.

DAC Interface

Interface for D/A converters.

Display Interface

Interface for LCD panel displays.

DOC Interface

Interface for the Data Operation Circuit.

ELC Interface

Interface for the Event Link Controller.

Ethernet Interface

Interface for Ethernet functions.

Ethernet PHY Interface

Interface for Ethernet phy functions.

External IRQ Interface

Interface for detecting external interrupts.

Flash Interface

Interface for the Flash Memory.

I2C Master Interface

Interface for I2C master communication.

I2C Slave Interface

Interface for I2C slave communication.

I2S Interface

Interface for I2S audio communication.

I/O Port Interface

Interface for accessing I/O ports and configuring I/O functionality.

JPEG Codec Interface

Interface for JPEG functions.

Key Matrix Interface

Interface for key matrix functions.

Low Power Modes Interface

Interface for accessing low power modes.

Low Voltage Detection Interface

Interface for Low Voltage Detection.

RTC Interface

Interface for accessing the Realtime Clock.

SD/MMC Interface

Interface for accessing SD, eMMC, and SDIO devices.

SPI Interface

Interface for SPI communications.

Timer Interface

Interface for timer functions.

Transfer Interface

Interface for data transfer functions.

UART Interface

Interface for UART communications.

USB Interface

Interface for USB functions.

USB HMSC Interface

Interface for USB HMSC functions.

USB PCDC Interface

Interface for USB PCDC functions.

WDT Interface

Interface for watch dog timer functions.

Touch Middleware Interface

Interface for Touch Middleware functions.

4.3.1 ADC Interface

Interfaces

Detailed Description

Interface for A/D Converters.

Summary

The ADC interface provides standard ADC functionality including one-shot mode (single scan), continuous scan and group scan. It also allows configuration of hardware and software triggers for starting scans. After each conversion an interrupt can be triggered, and if a callback function is provided, the call back is invoked with the appropriate event information.

Implemented by: [Analog to Digital Converter \(r_adc\)](#)

Data Structures

struct	adc_sample_state_t
struct	adc_status_t
struct	adc_callback_args_t
struct	adc_info_t
struct	adc_channel_cfg_t
struct	adc_cfg_t
struct	adc_api_t
struct	adc_instance_t

Typedefs

typedef void	adc_ctrl_t

Enumerations

enum	adc_mode_t
enum	adc_resolution_t
enum	adc_alignment_t
enum	adc_add_t
enum	adc_clear_t
enum	adc_trigger_t
enum	adc_sample_state_reg_t
enum	adc_event_t

enum [adc_group_a_t](#)enum [adc_channel_t](#)enum [adc_state_t](#)

Data Structure Documentation

◆ [adc_sample_state_t](#)

struct adc_sample_state_t		
ADC sample state configuration		
Data Fields		
adc_sample_state_reg_t	reg_id	Sample state register ID.
uint8_t	num_states	Number of sampling states for conversion. Ch16-20/21 use the same value.

◆ [adc_status_t](#)

struct adc_status_t		
ADC status.		
Data Fields		
adc_state_t	state	Current state.

◆ [adc_callback_args_t](#)

struct adc_callback_args_t		
ADC callback arguments definitions		
Data Fields		
uint16_t	unit	ADC device in use.
adc_event_t	event	ADC callback event.
void const *	p_context	Placeholder for user data.
adc_channel_t	channel	Channel of conversion result. Only valid for ADC_EVENT_CONVERSION_COMPLETE.

◆ [adc_info_t](#)

struct adc_info_t		
ADC Information Structure for Transfer Interface		
Data Fields		
__I uint16_t *	p_address	The address to start reading the data from.

uint32_t	length	The total number of transfers to read.
transfer_size_t	transfer_size	The size of each transfer.
elc_peripheral_t	elc_peripheral	Name of the peripheral in the ELC list.
elc_event_t	elc_event	Name of the ELC event for the peripheral.
uint32_t	calibration_data	Temperature sensor calibration data (0xFFFFFFFF if unsupported) for reference voltage.
int16_t	slope_microvolts	Temperature sensor slope in microvolts/degrees C.
bool	calibration_ongoing	Calibration is in progress.

◆ **adc_channel_cfg_t**

struct adc_channel_cfg_t		
ADC channel(s) configuration		
Data Fields		
uint32_t	scan_mask	Channels/bits: bit 0 is ch0; bit 15 is ch15. Use ADC_MASK_CHANNEL_x.
uint32_t	scan_mask_group_b	Valid for group modes. Use ADC_MASK_CHANNEL_x.
uint32_t	add_mask	Valid if add enabled in Open(). Use ADC_MASK_CHANNEL_x.
adc_group_a_t	priority_group_a	Valid for group modes.
uint8_t	sample_hold_mask	Channels/bits 0-2. Use ADC_MASK_CHANNEL_x.
uint8_t	sample_hold_states	Number of states to be used for sample and hold. Affects channels 0-2.

◆ **adc_cfg_t**

struct adc_cfg_t	
ADC general configuration	
Data Fields	
uint16_t	unit
	ADC Unit to be used.

adc_mode_t	mode
	ADC operation mode.
adc_resolution_t	resolution
	ADC resolution 8, 10, or 12-bit.
adc_alignment_t	alignment
	Specify left or right alignment; ignored if addition used.
adc_add_t	add_average_count
	Add or average samples.
adc_clear_t	clearing
	Clear after read.
adc_trigger_t	trigger
	Default and Group A trigger source.
adc_trigger_t	trigger_group_b
	Group B trigger source; valid only for group mode.
IRQn_Type	scan_end_irq
	Scan end IRQ number.
IRQn_Type	scan_end_b_irq
	Scan end group B IRQ number.
uint8_t	scan_end_ipl

	Scan end interrupt priority.
uint8_t	scan_end_b_ipl
	Scan end group B interrupt priority.
void(*	p_callback)(adc_callback_args_t *p_args)
	Callback function; set to NULL for none.
void const *	p_context
	Placeholder for user data. Passed to the user callback in <code>adc_api_t::adc_callback_args_t</code> .
void const *	p_extend
	Extension parameter for hardware specific settings.

◆ **adc_api_t**

struct adc_api_t	
ADC functions implemented at the HAL layer will follow this API.	
Data Fields	
fsp_err_t (*	open)(adc_ctrl_t *const p_ctrl, adc_cfg_t const *const p_cfg)
fsp_err_t (*	scanCfg)(adc_ctrl_t *const p_ctrl, adc_channel_cfg_t const *const p_channel_cfg)
fsp_err_t (*	scanStart)(adc_ctrl_t *const p_ctrl)
fsp_err_t (*	scanStop)(adc_ctrl_t *const p_ctrl)
fsp_err_t (*	scanStatusGet)(adc_ctrl_t *const p_ctrl, adc_status_t *p_status)
fsp_err_t (*	read)(adc_ctrl_t *const p_ctrl, adc_channel_t const reg_id, uint16_t

	*const p_data)
fsp_err_t(*	read32)(adc_ctrl_t *const p_ctrl, adc_channel_t const reg_id, uint32_t *const p_data)
fsp_err_t(*	sampleStateCountSet)(adc_ctrl_t *const p_ctrl, adc_sample_state_t *p_sample)
fsp_err_t(*	calibrate)(adc_ctrl_t *const p_ctrl, void *const p_extend)
fsp_err_t(*	offsetSet)(adc_ctrl_t *const p_ctrl, adc_channel_t const reg_id, int32_t const offset)
fsp_err_t(*	close)(adc_ctrl_t *const p_ctrl)
fsp_err_t(*	infoGet)(adc_ctrl_t *const p_ctrl, adc_info_t *const p_adc_info)
fsp_err_t(*	versionGet)(fsp_version_t *const p_version)

Field Documentation

◆ open

fsp_err_t(* adc_api_t::open) (adc_ctrl_t *const p_ctrl, adc_cfg_t const *const p_cfg)

Initialize ADC Unit; apply power, set the operational mode, trigger sources, interrupt priority, and configurations common to all channels and sensors.

Implemented as

- R_ADC_Open()
- R_SDADC_Open()

Precondition

Configure peripheral clocks, ADC pins and IRQs prior to calling this function.

Parameters

[in]	p_ctrl	Pointer to control handle structure
[in]	p_cfg	Pointer to configuration structure

◆ **scanCfg**

```
fsp_err_t(* adc_api_t::scanCfg) (adc_ctrl_t *const p_ctrl, adc_channel_cfg_t const *const p_channel_cfg)
```

Configure the scan including the channels, groups, and scan triggers to be used for the unit that was initialized in the open call. Some configurations are not supported for all implementations. See implementation for details.

Implemented as

- [R_ADC_ScanCfg\(\)](#)
- [R_SDADC_ScanConfigure\(\)](#)

Parameters

[in]	p_ctrl	Pointer to control handle structure
[in]	p_channel_cfg	Pointer to scan configuration structure

◆ **scanStart**

```
fsp_err_t(* adc_api_t::scanStart) (adc_ctrl_t *const p_ctrl)
```

Start the scan (in case of a software trigger), or enable the hardware trigger.

Implemented as

- [R_ADC_ScanStart\(\)](#)
- [R_SDADC_ScanStart\(\)](#)

Parameters

[in]	p_ctrl	Pointer to control handle structure

◆ **scanStop**

```
fsp_err_t(* adc_api_t::scanStop) (adc_ctrl_t *const p_ctrl)
```

Stop the ADC scan (in case of a software trigger), or disable the hardware trigger.

Implemented as

- [R_ADC_ScanStop\(\)](#)
- [R_SDADC_ScanStop\(\)](#)

Parameters

[in]	p_ctrl	Pointer to control handle structure

◆ **scanStatusGet**

```
fsp_err_t(* adc_api_t::scanStatusGet) (adc_ctrl_t *const p_ctrl, adc_status_t *p_status)
```

Check scan status.

Implemented as

- [R_ADC_StatusGet\(\)](#)
- [R_SDADC_StatusGet\(\)](#)

Parameters

[in]	p_ctrl	Pointer to control handle structure
[out]	p_status	Pointer to store current status in

◆ **read**

```
fsp_err_t(* adc_api_t::read) (adc_ctrl_t *const p_ctrl, adc_channel_t const reg_id, uint16_t *const p_data)
```

Read ADC conversion result.

Implemented as

- [R_ADC_Read\(\)](#)
- [R_SDADC_Read\(\)](#)

Parameters

[in]	p_ctrl	Pointer to control handle structure
[in]	reg_id	ADC channel to read (see enumeration <code>adc_channel_t</code>)
[in]	p_data	Pointer to variable to load value into.

◆ read32

```
fsp_err_t(* adc_api_t::read32) (adc_ctrl_t *const p_ctrl, adc_channel_t const reg_id, uint32_t *const p_data)
```

Read ADC conversion result into a 32-bit word.

Implemented as

- R_SDADC_Read32()

Parameters

[in]	p_ctrl	Pointer to control handle structure
[in]	reg_id	ADC channel to read (see enumeration adc_channel_t)
[in]	p_data	Pointer to variable to load value into.

◆ sampleStateCountSet

```
fsp_err_t(* adc_api_t::sampleStateCountSet) (adc_ctrl_t *const p_ctrl, adc_sample_state_t *p_sample)
```

Set the sample state count for the specified channel. Not supported for all implementations. See implementation for details.

Implemented as

- R_ADC_SetSampleStateCount()

Parameters

[in]	p_ctrl	Pointer to control handle structure
[in]	p_sample	Pointer to the ADC channels and corresponding sample states to be set

◆ **calibrate**

```
fsp_err_t(* adc_api_t::calibrate) (adc_ctrl_t *const p_ctrl, void *const p_extend)
```

Calibrate ADC or associated PGA (programmable gain amplifier). The driver may require implementation specific arguments to the p_extend input. Not supported for all implementations. See implementation for details.

Implemented as

- R_SDADC_Calibrate()

Parameters

[in]	p_ctrl	Pointer to control handle structure
[in]	p_extend	Pointer to implementation specific arguments

◆ **offsetSet**

```
fsp_err_t(* adc_api_t::offsetSet) (adc_ctrl_t *const p_ctrl, adc_channel_t const reg_id, int32_t const offset)
```

Set offset for input PGA configured for differential input. Not supported for all implementations. See implementation for details.

Implemented as

- R_SDADC_OffsetSet()

Parameters

[in]	p_ctrl	Pointer to control handle structure
[in]	reg_id	ADC channel to read (see enumeration adc_channel_t)
[in]	offset	See implementation for details.

◆ **close**

```
fsp_err_t(* adc_api_t::close) (adc_ctrl_t *const p_ctrl)
```

Close the specified ADC unit by ending any scan in progress, disabling interrupts, and removing power to the specified A/D unit.

Implemented as

- [R_ADC_Close\(\)](#)
- [R_SDADC_Close\(\)](#)

Parameters

[in]	p_ctrl	Pointer to control handle structure

◆ **infoGet**

```
fsp_err_t(* adc_api_t::infoGet) (adc_ctrl_t *const p_ctrl, adc_info_t *const p_adc_info)
```

Return the ADC data register address of the first (lowest number) channel and the total number of bytes to be read in order for the DTC/DMAC to read the conversion results of all configured channels. Return the temperature sensor calibration and slope data.

Implemented as

- [R_ADC_InfoGet\(\)](#)
- [R_SDADC_InfoGet\(\)](#)

Parameters

[in]	p_ctrl	Pointer to control handle structure
[out]	p_adc_info	Pointer to ADC information structure

◆ **versionGet**

```
fsp_err_t(* adc_api_t::versionGet) (fsp_version_t *const p_version)
```

Retrieve the API version.

Implemented as

- [R_ADC_VersionGet\(\)](#)
- [R_SDADC_VersionGet\(\)](#)

Precondition

This function retrieves the API version.

Parameters

[in]	p_version	Pointer to version structure

◆ **adc_instance_t**

```
struct adc_instance_t
```

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields

<code>adc_ctrl_t *</code>	<code>p_ctrl</code>	Pointer to the control structure for this instance.
<code>adc_cfg_t const *</code>	<code>p_cfg</code>	Pointer to the configuration structure for this instance.
<code>adc_channel_cfg_t const *</code>	<code>p_channel_cfg</code>	Pointer to the channel configuration structure for this instance.
<code>adc_api_t const *</code>	<code>p_api</code>	Pointer to the API structure for this instance.

Typedef Documentation

◆ `adc_ctrl_t`

```
typedef void adc_ctrl_t
```

ADC control block. Allocate using driver instance control structure from driver instance header file.

Enumeration Type Documentation

◆ `adc_mode_t`

```
enum adc_mode_t
```

ADC operation mode definitions

Enumerator

<code>ADC_MODE_SINGLE_SCAN</code>	Single scan - one or more channels.
<code>ADC_MODE_GROUP_SCAN</code>	Two trigger sources to trigger scan for two groups which contain one or more channels.
<code>ADC_MODE_CONTINUOUS_SCAN</code>	Continuous scan - one or more channels.

◆ **adc_resolution_t**

enum adc_resolution_t	
ADC data resolution definitions	
Enumerator	
ADC_RESOLUTION_12_BIT	12 bit resolution
ADC_RESOLUTION_10_BIT	10 bit resolution
ADC_RESOLUTION_8_BIT	8 bit resolution
ADC_RESOLUTION_14_BIT	14 bit resolution
ADC_RESOLUTION_16_BIT	16 bit resolution
ADC_RESOLUTION_24_BIT	24 bit resolution

◆ **adc_alignment_t**

enum adc_alignment_t	
ADC data alignment definitions	
Enumerator	
ADC_ALIGNMENT_RIGHT	Data alignment right.
ADC_ALIGNMENT_LEFT	Data alignment left.

◆ **adc_add_t**

enum adc_add_t	
ADC data sample addition and averaging options	
Enumerator	
ADC_ADD_OFF	Addition turned off for channels/sensors.
ADC_ADD_TWO	Add two samples.
ADC_ADD_THREE	Add three samples.
ADC_ADD_FOUR	Add four samples.
ADC_ADD_SIXTEEN	Add sixteen samples.
ADC_ADD_AVERAGE_TWO	Average two samples.
ADC_ADD_AVERAGE_FOUR	Average four samples.
ADC_ADD_AVERAGE_EIGHT	Average eight samples.
ADC_ADD_AVERAGE_SIXTEEN	Add sixteen samples.

◆ **adc_clear_t**

enum adc_clear_t	
ADC clear after read definitions	
Enumerator	
ADC_CLEAR_AFTER_READ_OFF	Clear after read off.
ADC_CLEAR_AFTER_READ_ON	Clear after read on.

◆ **adc_trigger_t**

enum adc_trigger_t	
ADC trigger mode definitions	
Enumerator	
ADC_TRIGGER_SOFTWARE	Software trigger; not for group modes.
ADC_TRIGGER_SYNC_ELC	Synchronous trigger via ELC.
ADC_TRIGGER_ASYNC_EXTERNAL	External asynchronous trigger; not for group modes.

◆ **adc_sample_state_reg_t**

enum <code>adc_sample_state_reg_t</code>	
ADC sample state registers	
Enumerator	
<code>ADC_SAMPLE_STATE_CHANNEL_0</code>	Sample state register channel 0.
<code>ADC_SAMPLE_STATE_CHANNEL_1</code>	Sample state register channel 1.
<code>ADC_SAMPLE_STATE_CHANNEL_2</code>	Sample state register channel 2.
<code>ADC_SAMPLE_STATE_CHANNEL_3</code>	Sample state register channel 3.
<code>ADC_SAMPLE_STATE_CHANNEL_4</code>	Sample state register channel 4.
<code>ADC_SAMPLE_STATE_CHANNEL_5</code>	Sample state register channel 5.
<code>ADC_SAMPLE_STATE_CHANNEL_6</code>	Sample state register channel 6.
<code>ADC_SAMPLE_STATE_CHANNEL_7</code>	Sample state register channel 7.
<code>ADC_SAMPLE_STATE_CHANNEL_8</code>	Sample state register channel 8.
<code>ADC_SAMPLE_STATE_CHANNEL_9</code>	Sample state register channel 9.
<code>ADC_SAMPLE_STATE_CHANNEL_10</code>	Sample state register channel 10.
<code>ADC_SAMPLE_STATE_CHANNEL_11</code>	Sample state register channel 11.
<code>ADC_SAMPLE_STATE_CHANNEL_12</code>	Sample state register channel 12.
<code>ADC_SAMPLE_STATE_CHANNEL_13</code>	Sample state register channel 13.
<code>ADC_SAMPLE_STATE_CHANNEL_14</code>	Sample state register channel 14.
<code>ADC_SAMPLE_STATE_CHANNEL_15</code>	Sample state register channel 15.
<code>ADC_SAMPLE_STATE_CHANNEL_16_TO_31</code>	Sample state register channel 16 to 31.

◆ **adc_event_t**

enum <code>adc_event_t</code>	
ADC callback event definitions	
Enumerator	
<code>ADC_EVENT_SCAN_COMPLETE</code>	Normal/Group A scan complete.
<code>ADC_EVENT_SCAN_COMPLETE_GROUP_B</code>	Group B scan complete.
<code>ADC_EVENT_CALIBRATION_COMPLETE</code>	Calibration complete.
<code>ADC_EVENT_CONVERSION_COMPLETE</code>	Conversion complete.

◆ **adc_group_a_t**

enum <code>adc_group_a_t</code>	
ADC action for group A interrupts group B scan. This enumeration is used to specify the priority between Group A and B in group mode.	
Enumerator	
<code>ADC_GROUP_A_PRIORITY_OFF</code>	Group A ignored and does not interrupt ongoing group B scan.
<code>ADC_GROUP_A_GROUP_B_WAIT_FOR_TRIGGER</code>	Group A interrupts Group B(single scan) which restarts at next Group B trigger.
<code>ADC_GROUP_A_GROUP_B_RESTART_SCAN</code>	Group A interrupts Group B(single scan) which restarts immediately after Group A scan is complete.
<code>ADC_GROUP_A_GROUP_B_CONTINUOUS_SCAN</code>	Group A interrupts Group B(continuous scan) which continues scanning without a new Group B trigger.

◆ **adc_channel_t**

enum <code>adc_channel_t</code>	
ADC channels	
Enumerator	
<code>ADC_CHANNEL_0</code>	ADC channel 0.
<code>ADC_CHANNEL_1</code>	ADC channel 1.
<code>ADC_CHANNEL_2</code>	ADC channel 2.
<code>ADC_CHANNEL_3</code>	ADC channel 3.
<code>ADC_CHANNEL_4</code>	ADC channel 4.
<code>ADC_CHANNEL_5</code>	ADC channel 5.
<code>ADC_CHANNEL_6</code>	ADC channel 6.
<code>ADC_CHANNEL_7</code>	ADC channel 7.
<code>ADC_CHANNEL_8</code>	ADC channel 8.
<code>ADC_CHANNEL_9</code>	ADC channel 9.
<code>ADC_CHANNEL_10</code>	ADC channel 10.
<code>ADC_CHANNEL_11</code>	ADC channel 11.
<code>ADC_CHANNEL_12</code>	ADC channel 12.
<code>ADC_CHANNEL_13</code>	ADC channel 13.
<code>ADC_CHANNEL_14</code>	ADC channel 14.
<code>ADC_CHANNEL_15</code>	ADC channel 15.
<code>ADC_CHANNEL_16</code>	ADC channel 16.
<code>ADC_CHANNEL_17</code>	ADC channel 17.
<code>ADC_CHANNEL_18</code>	ADC channel 18.
<code>ADC_CHANNEL_19</code>	ADC channel 19.
<code>ADC_CHANNEL_20</code>	ADC channel 20.

ADC_CHANNEL_21	ADC channel 21.
ADC_CHANNEL_22	ADC channel 22.
ADC_CHANNEL_23	ADC channel 23.
ADC_CHANNEL_24	ADC channel 24.
ADC_CHANNEL_25	ADC channel 25.
ADC_CHANNEL_26	ADC channel 26.
ADC_CHANNEL_27	ADC channel 27.
ADC_CHANNEL_TEMPERATURE	Temperature sensor output.
ADC_CHANNEL_VOLT	Internal reference voltage.

◆ adc_state_t

enum <code>adc_state_t</code>	
ADC states.	
Enumerator	
ADC_STATE_IDLE	ADC is idle.
ADC_STATE_SCAN_IN_PROGRESS	ADC scan in progress.

4.3.2 CAC Interface

Interfaces

Detailed Description

Interface for clock frequency accuracy measurements.

Summary

The interface for the clock frequency accuracy measurement circuit (CAC) peripheral is used to check a system clock frequency with a reference clock signal by counting the number of pulses of the clock to be measured.

Implemented by: [Clock Frequency Accuracy Measurement Circuit \(r_cac\)](#)

Data Structures

struct [cac_ref_clock_config_t](#)

struct [cac_meas_clock_config_t](#)

struct [cac_callback_args_t](#)

struct [cac_cfg_t](#)

struct [cac_api_t](#)

struct [cac_instance_t](#)

Typedefs

typedef void [cac_ctrl_t](#)

Enumerations

enum [cac_event_t](#)

enum [cac_clock_type_t](#)

enum [cac_clock_source_t](#)

enum [cac_ref_divider_t](#)

enum [cac_ref_digfilter_t](#)

enum [cac_ref_edge_t](#)

enum [cac_meas_divider_t](#)

Data Structure Documentation

◆ [cac_ref_clock_config_t](#)

struct [cac_ref_clock_config_t](#)

Structure defining the settings that apply to reference clock configuration.

Data Fields

cac_ref_divider_t	divider	Divider specification for the Reference clock.
cac_clock_source_t	clock	Clock source for the Reference clock.
cac_ref_digfilter_t	digfilter	Digital filter selection for the CACREF ext clock.
cac_ref_edge_t	edge	Edge detection for the

Reference clock.

◆ **cac_meas_clock_config_t**

struct cac_meas_clock_config_t

Structure defining the settings that apply to measurement clock configuration.

Data Fields

cac_meas_divider_t

divider

Divider specification for the Measurement clock.

cac_clock_source_t

clock

Clock source for the Measurement clock.

◆ **cac_callback_args_t**

struct cac_callback_args_t

Callback function parameter data

Data Fields

cac_event_t

event

The event can be used to identify what caused the callback.

void const *

p_context

Value provided in configuration structure.

◆ **cac_cfg_t**

struct cac_cfg_t

CAC Configuration

Data Fields

cac_ref_clock_config_t

cac_ref_clock

reference clock specific settings

cac_meas_clock_config_t

cac_meas_clock

measurement clock specific settings

uint16_t

cac_upper_limit

the upper limit counter threshold

uint16_t

cac_lower_limit

the lower limit counter threshold

IRQn_Type	mendi_irq
	Measurement End IRQ number.
IRQn_Type	ovfi_irq
	Measurement Overflow IRQ number.
IRQn_Type	ferri_irq
	Frequency Error IRQ number.
uint8_t	mendi_ipl
	Measurement end interrupt priority.
uint8_t	ovfi_ipl
	Overflow interrupt priority.
uint8_t	ferri_ipl
	Frequency error interrupt priority.
void(*)	p_callback)(cac_callback_args_t *p_args)
	Callback provided when a CAC interrupt ISR occurs.
void const *	p_context
	Passed to user callback in cac_callback_args_t .
void const *	p_extend
	CAC hardware dependent configuration */.

◆ **cac_api_t**

struct cac_api_t

CAC functions implemented at the HAL layer API

Data Fieldsfsp_err_t(*) **open**)(cac_ctrl_t *const p_ctrl, cac_cfg_t const *const p_cfg)fsp_err_t(*) **startMeasurement**)(cac_ctrl_t *const p_ctrl)fsp_err_t(*) **stopMeasurement**)(cac_ctrl_t *const p_ctrl)fsp_err_t(*) **read**)(cac_ctrl_t *const p_ctrl, uint16_t *const p_counter)fsp_err_t(*) **close**)(cac_ctrl_t *const p_ctrl)fsp_err_t(*) **versionGet**)(fsp_version_t *p_version)**Field Documentation**◆ **open**

fsp_err_t(*) cac_api_t::open) (cac_ctrl_t *const p_ctrl, cac_cfg_t const *const p_cfg)

Open function for CAC device.

Parameters

[out]	p_ctrl	Pointer to CAC device control. Must be declared by user.
[in]	cac_cfg_t	Pointer to CAC configuration structure.

◆ **startMeasurement**

fsp_err_t(*) cac_api_t::startMeasurement) (cac_ctrl_t *const p_ctrl)

Begin a measurement for the CAC peripheral.

Parameters

[in]	p_ctrl	Pointer to CAC device control.

◆ **stopMeasurement**

```
fsp_err_t(* cac_api_t::stopMeasurement) (cac_ctrl_t *const p_ctrl)
```

End a measurement for the CAC peripheral.

Parameters

[in]	p_ctrl	Pointer to CAC device control.

◆ **read**

```
fsp_err_t(* cac_api_t::read) (cac_ctrl_t *const p_ctrl, uint16_t *const p_counter)
```

Read function for CAC peripheral.

Parameters

[in]	p_ctrl	Control for the CAC device context.
[in]	p_counter	Pointer to variable in which to store the current CACNTBR register contents.

◆ **close**

```
fsp_err_t(* cac_api_t::close) (cac_ctrl_t *const p_ctrl)
```

Close function for CAC device.

Parameters

[in]	p_ctrl	Pointer to CAC device control.

◆ **versionGet**

```
fsp_err_t(* cac_api_t::versionGet) (fsp_version_t *p_version)
```

Get the CAC API and code version information.

Parameters

[out]	p_version	is value returned.

◆ **cac_instance_t**

```
struct cac_instance_t
```

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields

<code>cac_ctrl_t *</code>	<code>p_ctrl</code>	Pointer to the control structure for this instance.
<code>cac_cfg_t const *</code>	<code>p_cfg</code>	Pointer to the configuration structure for this instance.
<code>cac_api_t const *</code>	<code>p_api</code>	Pointer to the API structure for this instance.

Typedef Documentation

◆ `cac_ctrl_t`

typedef void `cac_ctrl_t`

CAC control block. Allocate an instance specific control block to pass into the CAC API calls.

Implemented as

- `cac_instance_ctrl_t`

Enumeration Type Documentation

◆ `cac_event_t`

enum `cac_event_t`

Event types returned by the ISR callback when used in CAC interrupt mode

Enumerator

<code>CAC_EVENT_FREQUENCY_ERROR</code>	Frequency error.
<code>CAC_EVENT_MEASUREMENT_COMPLETE</code>	Measurement complete.
<code>CAC_EVENT_COUNTER_OVERFLOW</code>	Counter overflow.

◆ `cac_clock_type_t`

enum `cac_clock_type_t`

Enumeration of the two possible clocks.

Enumerator

<code>CAC_CLOCK_MEASURED</code>	Measurement clock.
<code>CAC_CLOCK_REFERENCE</code>	Reference clock.

◆ **cac_clock_source_t**

enum <code>cac_clock_source_t</code>	
Enumeration of the possible clock sources for both the reference and measurement clocks.	
Enumerator	
<code>CAC_CLOCK_SOURCE_MAIN_OSC</code>	Main clock oscillator.
<code>CAC_CLOCK_SOURCE_SUBCLOCK</code>	Sub-clock.
<code>CAC_CLOCK_SOURCE_HOCO</code>	HOCO (High speed on chip oscillator)
<code>CAC_CLOCK_SOURCE_MOCO</code>	MOCO (Middle speed on chip oscillator)
<code>CAC_CLOCK_SOURCE_LOCO</code>	LOCO (Middle speed on chip oscillator)
<code>CAC_CLOCK_SOURCE_PCLKB</code>	PCLKB (Peripheral Clock B)
<code>CAC_CLOCK_SOURCE_IWDT</code>	IWDT- Dedicated on-chip oscillator.
<code>CAC_CLOCK_SOURCE_EXTERNAL</code>	Externally supplied measurement clock on CACREF pin.

◆ **cac_ref_divider_t**

enum <code>cac_ref_divider_t</code>	
Enumeration of available dividers for the reference clock.	
Enumerator	
<code>CAC_REF_DIV_32</code>	Reference clock divided by 32.
<code>CAC_REF_DIV_128</code>	Reference clock divided by 128.
<code>CAC_REF_DIV_1024</code>	Reference clock divided by 1024.
<code>CAC_REF_DIV_8192</code>	Reference clock divided by 8192.

◆ **cac_ref_digfilter_t**

enum cac_ref_digfilter_t	
Enumeration of available digital filter settings for an external reference clock.	
Enumerator	
CAC_REF_DIGITAL_FILTER_OFF	No digital filter on the CACREF pin for reference clock.
CAC_REF_DIGITAL_FILTER_1	Sampling clock for digital filter = measuring frequency.
CAC_REF_DIGITAL_FILTER_4	Sampling clock for digital filter = measuring frequency/4.
CAC_REF_DIGITAL_FILTER_16	Sampling clock for digital filter = measuring frequency/16.

◆ **cac_ref_edge_t**

enum cac_ref_edge_t	
Enumeration of available edge detect settings for the reference clock.	
Enumerator	
CAC_REF_EDGE_RISE	Rising edge detect for the Reference clock.
CAC_REF_EDGE_FALL	Falling edge detect for the Reference clock.
CAC_REF_EDGE_BOTH	Both Rising and Falling edges detect for the Reference clock.

◆ **cac_meas_divider_t**

enum <code>cac_meas_divider_t</code>	
Enumeration of available dividers for the measurement clock	
Enumerator	
<code>CAC_MEAS_DIV_1</code>	Measurement clock divided by 1.
<code>CAC_MEAS_DIV_4</code>	Measurement clock divided by 4.
<code>CAC_MEAS_DIV_8</code>	Measurement clock divided by 8.
<code>CAC_MEAS_DIV_32</code>	Measurement clock divided by 32.

4.3.3 CGC Interface[Interfaces](#)**Detailed Description**

Interface for clock generation.

Summary

The CGC interface provides the ability to configure and use all of the CGC module's capabilities. Among the capabilities is the selection of several clock sources to use as the system clock source. Additionally, the system clocks can be divided down to provide a wide range of frequencies for various system and peripheral needs.

Clock stability can be checked and clocks may also be stopped to save power when not needed. The API has a function to return the frequency of the system and system peripheral clocks at run time. There is also a feature to detect when the main oscillator has stopped, with the option of calling a user provided callback function.

The CGC interface is implemented by:

- [Clock Generation Circuit \(`r\_cgc`\)](#)

Data Structures

struct	<code>cgc_callback_args_t</code>
struct	<code>cgc_pll_cfg_t</code>
union	<code>cgc_divider_cfg_t</code>

struct [cgc_cfg_t](#)

struct [cgc_clocks_cfg_t](#)

struct [cgc_api_t](#)

struct [cgc_instance_t](#)

Typedefs

typedef void [cgc_ctrl_t](#)

Enumerations

enum [cgc_event_t](#)

enum [cgc_clock_t](#)

enum [cgc_pll_div_t](#)

enum [cgc_pll_mul_t](#)

enum [cgc_sys_clock_div_t](#)

enum [cgc_usb_clock_div_t](#)

enum [cgc_clock_change_t](#)

Data Structure Documentation

◆ **cgc_callback_args_t**

struct cgc_callback_args_t		
Callback function parameter data		
Data Fields		
cgc_event_t	event	The event can be used to identify what caused the callback.
void const *	p_context	Placeholder for user data.

◆ **cgc_pll_cfg_t**

struct cgc_pll_cfg_t		
Clock configuration structure - Used as an input parameter to the cgc_api_t::clockStart function for the PLL clock.		
Data Fields		
cgc_clock_t	source_clock	PLL source clock (main oscillator or HOCO)

cgc_pll_div_t	divider	PLL divider.
cgc_pll_mul_t	multiplier	PLL multiplier.

◆ **cgc_divider_cfg_t**

union cgc_divider_cfg_t		
Clock configuration structure - Used as an input parameter to the cgc_api_t::systemClockSet and cgc_api_t::systemClockGet functions.		
Data Fields		
uint32_t	sckdivcr_w	(@ 0x4001E020) System clock Division control register
struct cgc_divider_cfg_t	__unnamed__	

◆ **cgc_cfg_t**

struct cgc_cfg_t
Configuration options.

◆ **cgc_clocks_cfg_t**

struct cgc_clocks_cfg_t		
Clock configuration		
Data Fields		
cgc_clock_t	system_clock	System clock source enumeration.
cgc_pll_cfg_t	pll_cfg	PLL configuration structure.
cgc_divider_cfg_t	divider_cfg	Clock dividers structure.
cgc_clock_change_t	loco_state	State of LOCO.
cgc_clock_change_t	moco_state	State of MOCO.
cgc_clock_change_t	hoco_state	State of HOCO.
cgc_clock_change_t	mainosc_state	State of Main oscillator.
cgc_clock_change_t	pll_state	State of PLL.

◆ **cgc_api_t**

struct cgc_api_t		
CGC functions implemented at the HAL layer follow this API.		
Data Fields		
fsp_err_t (*	open)(cgc_ctrl_t *const p_ctrl, cgc_cfg_t const *const p_cfg)	
fsp_err_t (*	clocksCfg)(cgc_ctrl_t *const p_ctrl, cgc_clocks_cfg_t const *const p_clock_cfg)	

<code>fsp_err_t(*</code>	<code>clockStart)(cgc_ctrl_t *const p_ctrl, cgc_clock_t clock_source, cgc_pll_cfg_t const *const p_pll_cfg)</code>
<code>fsp_err_t(*</code>	<code>clockStop)(cgc_ctrl_t *const p_ctrl, cgc_clock_t clock_source)</code>
<code>fsp_err_t(*</code>	<code>clockCheck)(cgc_ctrl_t *const p_ctrl, cgc_clock_t clock_source)</code>
<code>fsp_err_t(*</code>	<code>systemClockSet)(cgc_ctrl_t *const p_ctrl, cgc_clock_t clock_source, cgc_divider_cfg_t const *const p_divider_cfg)</code>
<code>fsp_err_t(*</code>	<code>systemClockGet)(cgc_ctrl_t *const p_ctrl, cgc_clock_t *const p_clock_source, cgc_divider_cfg_t *const p_divider_cfg)</code>
<code>fsp_err_t(*</code>	<code>oscStopDetectEnable)(cgc_ctrl_t *const p_ctrl)</code>
<code>fsp_err_t(*</code>	<code>oscStopDetectDisable)(cgc_ctrl_t *const p_ctrl)</code>
<code>fsp_err_t(*</code>	<code>oscStopStatusClear)(cgc_ctrl_t *const p_ctrl)</code>
<code>fsp_err_t(*</code>	<code>close)(cgc_ctrl_t *const p_ctrl)</code>
<code>fsp_err_t(*</code>	<code>versionGet)(fsp_version_t *p_version)</code>

Field Documentation

◆ open

`fsp_err_t(* cgc_api_t::open) (cgc_ctrl_t *const p_ctrl, cgc_cfg_t const *const p_cfg)`

Initial configuration

Implemented as

- `R_CGC_Open()`

Parameters

[in]	<code>p_ctrl</code>	Pointer to instance control block
[in]	<code>p_cfg</code>	Pointer to configuration

◆ **clocksCfg**

```
fsp_err_t(* cgc_api_t::clocksCfg) (cgc_ctrl_t *const p_ctrl, cgc_clocks_cfg_t const *const p_clock_cfg)
```

Configure all system clocks.

Implemented as

◦ [R_CGC_ClocksCfg\(\)](#)

Parameters

[in]	p_ctrl	Pointer to instance control block
[in]	p_clock_cfg	Pointer to desired configuration of system clocks

◆ **clockStart**

```
fsp_err_t(* cgc_api_t::clockStart) (cgc_ctrl_t *const p_ctrl, cgc_clock_t clock_source, cgc_pll_cfg_t const *const p_pll_cfg)
```

Start a clock.

Implemented as

◦ [R_CGC_ClockStart\(\)](#)

Parameters

[in]	p_ctrl	Pointer to instance control block
[in]	clock_source	Clock source to start
[in]	p_pll_cfg	Pointer to PLL configuration, can be NULL if clock_source is not CGC_CLOCK_PLL

◆ **clockStop**

```
fsp_err_t(* cgc_api_t::clockStop) (cgc_ctrl_t *const p_ctrl, cgc_clock_t clock_source)
```

Stop a clock.

Implemented as

◦ [R_CGC_ClockStop\(\)](#)

Parameters

[in]	p_ctrl	Pointer to instance control block
[in]	clock_source	The clock source to stop

◆ **clockCheck**

```
fsp_err_t(* cgc_api_t::clockCheck) (cgc_ctrl_t *const p_ctrl, cgc_clock_t clock_source)
```

Check the stability of the selected clock.

Implemented as

◦ [R_CGC_ClockCheck\(\)](#)

Parameters

[in]	p_ctrl	Pointer to instance control block
[in]	clock_source	Which clock source to check for stability

◆ **systemClockSet**

```
fsp_err_t(* cgc_api_t::systemClockSet) (cgc_ctrl_t *const p_ctrl, cgc_clock_t clock_source, cgc_divider_cfg_t const *const p_divider_cfg)
```

Set the system clock.

Implemented as

◦ [R_CGC_SystemClockSet\(\)](#)

Parameters

[in]	p_ctrl	Pointer to instance control block
[in]	clock_source	Clock source to set as system clock
[in]	p_divider_cfg	Pointer to the clock divider configuration

◆ **systemClockGet**

```
fsp_err_t(* cgc_api_t::systemClockGet) (cgc_ctrl_t *const p_ctrl, cgc_clock_t *const p_clock_source,
cgc_divider_cfg_t *const p_divider_cfg)
```

Get the system clock information.

Implemented as

◦ [R_CGC_SystemClockGet\(\)](#)

Parameters

[in]	p_ctrl	Pointer to instance control block
[out]	p_clock_source	Returns the current system clock
[out]	p_divider_cfg	Returns the current system clock dividers

◆ **oscStopDetectEnable**

```
fsp_err_t(* cgc_api_t::oscStopDetectEnable) (cgc_ctrl_t *const p_ctrl)
```

Enable and optionally register a callback for Main Oscillator stop detection.

Implemented as

◦ [R_CGC_OscStopDetectEnable\(\)](#)

Parameters

[in]	p_ctrl	Pointer to instance control block
[in]	p_callback	Callback function that will be called by the NMI interrupt when an oscillation stop is detected. If the second argument is "false", then this argument can be NULL.
[in]	enable	Enable/disable Oscillation Stop Detection

◆ **oscStopDetectDisable**

```
fsp_err_t(* cgc_api_t::oscStopDetectDisable) (cgc_ctrl_t *const p_ctrl)
```

Disable Main Oscillator stop detection.

Implemented as

- [R_CGC_OscStopDetectDisable\(\)](#)

Parameters

[in]	p_ctrl	Pointer to instance control block

◆ **oscStopStatusClear**

```
fsp_err_t(* cgc_api_t::oscStopStatusClear) (cgc_ctrl_t *const p_ctrl)
```

Clear the oscillator stop detection flag.

Implemented as

- [R_CGC_OscStopStatusClear\(\)](#)

Parameters

[in]	p_ctrl	Pointer to instance control block

◆ **close**

```
fsp_err_t(* cgc_api_t::close) (cgc_ctrl_t *const p_ctrl)
```

Close the CGC driver.

Implemented as

- [R_CGC_Close\(\)](#)

Parameters

[in]	p_ctrl	Pointer to instance control block

◆ **versionGet**

fsp_err_t(* cgc_api_t::versionGet) (fsp_version_t *p_version)

Gets the CGC driver version.

Implemented as

◦ R_CGC_VersionGet()

Parameters

[out]	p_version	Code and API version used

◆ **cgc_instance_t**

struct cgc_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields

cgc_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
cgc_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
cgc_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation◆ **cgc_ctrl_t**

typedef void cgc_ctrl_t

CGC control block. Allocate an instance specific control block to pass into the CGC API calls.

Implemented as

◦ cgc_instance_ctrl_t

Enumeration Type Documentation◆ **cgc_event_t**

enum cgc_event_t

Events that can trigger a callback function

Enumerator

CGC_EVENT_OSC_STOP_DETECT	Oscillator stop detection has caused the event.

◆ **cgc_clock_t**

enum cgc_clock_t	
System clock source identifiers - The source of ICLK, BCLK, FCLK, PCLKS A-D and UCLK prior to the system clock divider	
Enumerator	
CGC_CLOCK_HOCO	The high speed on chip oscillator.
CGC_CLOCK_MOCO	The middle speed on chip oscillator.
CGC_CLOCK_LOCO	The low speed on chip oscillator.
CGC_CLOCK_MAIN_OSC	The main oscillator.
CGC_CLOCK_SUBCLOCK	The subclock oscillator.
CGC_CLOCK_PLL	The PLL oscillator.

◆ **cgc_pll_div_t**

enum cgc_pll_div_t	
PLL divider values	
Enumerator	
CGC_PLL_DIV_1	PLL divider of 1.
CGC_PLL_DIV_2	PLL divider of 2.
CGC_PLL_DIV_3	PLL divider of 3 (S7, S5 only)
CGC_PLL_DIV_4	PLL divider of 4 (S3 only)

◆ **cgc_pll_mul_t**

enum cgc_pll_mul_t	
PLL multiplier values	
Enumerator	
CGC_PLL_MUL_8_0	PLL multiplier of 8.0.
CGC_PLL_MUL_9_0	PLL multiplier of 9.0.
CGC_PLL_MUL_10_0	PLL multiplier of 10.0.
CGC_PLL_MUL_10_5	PLL multiplier of 10.5.
CGC_PLL_MUL_11_0	PLL multiplier of 11.0.
CGC_PLL_MUL_11_5	PLL multiplier of 11.5.
CGC_PLL_MUL_12_0	PLL multiplier of 12.0.
CGC_PLL_MUL_12_5	PLL multiplier of 12.5.
CGC_PLL_MUL_13_0	PLL multiplier of 13.0.
CGC_PLL_MUL_13_5	PLL multiplier of 13.5.
CGC_PLL_MUL_14_0	PLL multiplier of 14.0.
CGC_PLL_MUL_14_5	PLL multiplier of 14.5.
CGC_PLL_MUL_15_0	PLL multiplier of 15.0.
CGC_PLL_MUL_15_5	PLL multiplier of 15.5.
CGC_PLL_MUL_16_0	PLL multiplier of 16.0.
CGC_PLL_MUL_16_5	PLL multiplier of 16.5.
CGC_PLL_MUL_17_0	PLL multiplier of 17.0.
CGC_PLL_MUL_17_5	PLL multiplier of 17.5.
CGC_PLL_MUL_18_0	PLL multiplier of 18.0.
CGC_PLL_MUL_18_5	PLL multiplier of 18.5.
CGC_PLL_MUL_19_0	PLL multiplier of 19.0.

CGC_PLL_MUL_19_5	PLL multiplier of 19.5.
CGC_PLL_MUL_20_0	PLL multiplier of 20.0.
CGC_PLL_MUL_20_5	PLL multiplier of 20.5.
CGC_PLL_MUL_21_0	PLL multiplier of 21.0.
CGC_PLL_MUL_21_5	PLL multiplier of 21.5.
CGC_PLL_MUL_22_0	PLL multiplier of 22.0.
CGC_PLL_MUL_22_5	PLL multiplier of 22.5.
CGC_PLL_MUL_23_0	PLL multiplier of 23.0.
CGC_PLL_MUL_23_5	PLL multiplier of 23.5.
CGC_PLL_MUL_24_0	PLL multiplier of 24.0.
CGC_PLL_MUL_24_5	PLL multiplier of 24.5.
CGC_PLL_MUL_25_0	PLL multiplier of 25.0.
CGC_PLL_MUL_25_5	PLL multiplier of 25.5.
CGC_PLL_MUL_26_0	PLL multiplier of 26.0.
CGC_PLL_MUL_26_5	PLL multiplier of 26.5.
CGC_PLL_MUL_27_0	PLL multiplier of 27.0.
CGC_PLL_MUL_27_5	PLL multiplier of 27.5.
CGC_PLL_MUL_28_0	PLL multiplier of 28.0.
CGC_PLL_MUL_28_5	PLL multiplier of 28.5.
CGC_PLL_MUL_29_0	PLL multiplier of 29.0.
CGC_PLL_MUL_29_5	PLL multiplier of 29.5.
CGC_PLL_MUL_30_0	PLL multiplier of 30.0.
CGC_PLL_MUL_31_0	PLL multiplier of 31.0.

◆ **cgc_sys_clock_div_t**

enum cgc_sys_clock_div_t	
System clock divider vlues - The individually selectable divider of each of the system clocks, ICLK, BCLK, FCLK, PCLKS A-D.	
Enumerator	
CGC_SYS_CLOCK_DIV_1	System clock divided by 1.
CGC_SYS_CLOCK_DIV_2	System clock divided by 2.
CGC_SYS_CLOCK_DIV_4	System clock divided by 4.
CGC_SYS_CLOCK_DIV_8	System clock divided by 8.
CGC_SYS_CLOCK_DIV_16	System clock divided by 16.
CGC_SYS_CLOCK_DIV_32	System clock divided by 32.
CGC_SYS_CLOCK_DIV_64	System clock divided by 64.

◆ **cgc_usb_clock_div_t**

enum cgc_usb_clock_div_t	
USB clock divider values	
Enumerator	
CGC_USB_CLOCK_DIV_3	Divide USB source clock by 3.
CGC_USB_CLOCK_DIV_4	Divide USB source clock by 4.
CGC_USB_CLOCK_DIV_5	Divide USB source clock by 5.

◆ **cgc_clock_change_t**

enum <code>cgc_clock_change_t</code>	
Clock options	
Enumerator	
<code>CGC_CLOCK_CHANGE_START</code>	Start the clock.
<code>CGC_CLOCK_CHANGE_STOP</code>	Stop the clock.
<code>CGC_CLOCK_CHANGE_NONE</code>	No change to the clock.

4.3.4 Comparator Interface

Interfaces

Detailed Description

Interface for comparators.

Summary

The comparator interface provides standard comparator functionality, including generating an event when the comparator result changes.

Implemented by: [High-Speed Analog Comparator \(r_acmphs\)](#) [Low-Power Analog Comparator \(r_acmplp\)](#)

Data Structures

struct	<code>comparator_info_t</code>
struct	<code>comparator_status_t</code>
struct	<code>comparator_callback_args_t</code>
struct	<code>comparator_cfg_t</code>
struct	<code>comparator_api_t</code>
struct	<code>comparator_instance_t</code>

Macros

#define	<code>COMPARATOR_API_VERSION_MAJOR</code>

Typedefs

```
typedef void comparator\_ctrl\_t
```

Enumerations

```
enum comparator\_mode\_t
```

```
enum comparator\_trigger\_t
```

```
enum comparator\_polarity\_invert\_t
```

```
enum comparator\_pin\_output\_t
```

```
enum comparator\_filter\_t
```

```
enum comparator\_state\_t
```

Data Structure Documentation

◆ [comparator_info_t](#)

struct comparator_info_t		
Comparator information.		
Data Fields		
uint32_t	min_stabilization_wait_us	Minimum stabilization wait time in microseconds.

◆ [comparator_status_t](#)

struct comparator_status_t		
Comparator status.		
Data Fields		
comparator_state_t	state	Current comparator state.

◆ [comparator_callback_args_t](#)

struct comparator_callback_args_t		
Callback function parameter data		
Data Fields		
void const *	p_context	Placeholder for user data. Set in comparator_api_t::open function in comparator_cfg_t .
uint32_t	channel	The physical hardware channel that caused the interrupt.

◆ [comparator_cfg_t](#)

struct comparator_cfg_t	
User configuration structure, used in open function	
Data Fields	
uint8_t	channel
	Hardware channel used.
comparator_mode_t	mode
	Normal or window mode.
comparator_trigger_t	trigger
	Trigger setting.
comparator_filter_t	filter
	Digital filter clock divisor setting.
comparator_polarity_invert_t	invert
	Whether to invert output.
comparator_pin_output_t	pin_output
	Whether to include output on output pin.
uint8_t	vref_select
	Internal Vref Select.
uint8_t	ipl
	Interrupt priority.
IRQn_Type	irq

	NVIC interrupt number.
void(*	p_callback)(comparator_callback_args_t *p_args)
void const *	p_context
void const *	p_extend
	Comparator hardware dependent configuration.

Field Documentation

◆ [p_callback](#)

void(* comparator_cfg_t::p_callback) ([comparator_callback_args_t](#) *p_args)

Callback called when comparator event occurs.

◆ [p_context](#)

void const* comparator_cfg_t::p_context

Placeholder for user data. Passed to the user callback in [comparator_callback_args_t](#).

◆ [comparator_api_t](#)

struct comparator_api_t

Comparator functions implemented at the HAL layer will follow this API.

Data Fields

fsp_err_t (*	open)(comparator_ctrl_t *const p_ctrl, comparator_cfg_t const *const p_cfg)
fsp_err_t (*	outputEnable)(comparator_ctrl_t *const p_ctrl)
fsp_err_t (*	infoGet)(comparator_ctrl_t *const p_ctrl, comparator_info_t *const p_info)
fsp_err_t (*	statusGet)(comparator_ctrl_t *const p_ctrl, comparator_status_t *const p_status)
fsp_err_t (*	close)(comparator_ctrl_t *const p_ctrl)

`fsp_err_t(* versionGet )(fsp_version_t *const p_version)`

Field Documentation

◆ open

`fsp_err_t(* comparator_api_t::open) (comparator_ctrl_t *const p_ctrl, comparator_cfg_t const *const p_cfg)`

Initialize the comparator.

Implemented as

- `R_ACMPHS_Open()`
- `R_ACMPLP_Open()`

Parameters

[in]	p_ctrl	Pointer to instance control block
[in]	p_cfg	Pointer to configuration

◆ outputEnable

`fsp_err_t(* comparator_api_t::outputEnable) (comparator_ctrl_t *const p_ctrl)`

Start the comparator.

Implemented as

- `R_ACMPHS_OutputEnable()`
- `R_ACMPLP_OutputEnable()`

Parameters

[in]	p_ctrl	Pointer to instance control block

◆ **infoGet**

```
fsp_err_t(* comparator_api_t::infoGet) (comparator_ctrl_t *const p_ctrl, comparator_info_t *const p_info)
```

Provide information such as the recommended minimum stabilization wait time.

Implemented as

- R_ACMPHS_InfoGet()
- R_ACMPPLP_InfoGet()

Parameters

[in]	p_ctrl	Pointer to instance control block
[out]	p_info	Comparator information stored here

◆ **statusGet**

```
fsp_err_t(* comparator_api_t::statusGet) (comparator_ctrl_t *const p_ctrl, comparator_status_t *const p_status)
```

Provide current comparator status.

Implemented as

- R_ACMPHS_StatusGet()
- R_ACMPPLP_StatusGet()

Parameters

[in]	p_ctrl	Pointer to instance control block
[out]	p_status	Status stored here

◆ **close**

```
fsp_err_t(* comparator_api_t::close) (comparator_ctrl_t *const p_ctrl)
```

Stop the comparator.

Implemented as

- R_ACMPHS_Close()
- R_ACMPPLP_Close()

Parameters

[in]	p_ctrl	Pointer to instance control block

◆ **versionGet**

fsp_err_t(* comparator_api_t::versionGet) (fsp_version_t *const p_version)

Retrieve the API version.

Implemented as

- R_ACMPHS_VersionGet()
- R_ACMPLP_VersionGet()

Precondition

This function retrieves the API version.

Parameters

[in]	p_version	Pointer to version structure

◆ **comparator_instance_t**

struct comparator_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields

comparator_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
comparator_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
comparator_api_t const *	p_api	Pointer to the API structure for this instance.

Macro Definition Documentation◆ **COMPARATOR_API_VERSION_MAJOR**

#define COMPARATOR_API_VERSION_MAJOR

Includes board and MCU related header files. Version Number of API.

Typedef Documentation◆ **comparator_ctrl_t**

typedef void comparator_ctrl_t

Comparator control block. Allocate an instance specific control block to pass into the comparator API calls.

Implemented as

- acmphs_instance_ctrl_t
- acmplp_instance_ctrl_t

Enumeration Type Documentation

◆ comparator_mode_t

enum comparator_mode_t	
Select whether to invert the polarity of the comparator output.	
Enumerator	
COMPARATOR_MODE_NORMAL	Normal mode.
COMPARATOR_MODE_WINDOW	Window mode, not supported by all implementations.

◆ comparator_trigger_t

enum comparator_trigger_t	
Trigger type: rising edge, falling edge, both edges, low level.	
Enumerator	
COMPARATOR_TRIGGER_RISING	Rising edge trigger.
COMPARATOR_TRIGGER_FALLING	Falling edge trigger.
COMPARATOR_TRIGGER_BOTH_EDGE	Both edges trigger.

◆ comparator_polarity_invert_t

enum comparator_polarity_invert_t	
Select whether to invert the polarity of the comparator output.	
Enumerator	
COMPARATOR_POLARITY_INVERT_OFF	Do not invert polarity.
COMPARATOR_POLARITY_INVERT_ON	Invert polarity.

◆ **comparator_pin_output_t**

enum comparator_pin_output_t	
Select whether to include the comparator output on the output pin.	
Enumerator	
COMPARATOR_PIN_OUTPUT_OFF	Do not include comparator output on output pin.
COMPARATOR_PIN_OUTPUT_ON	Include comparator output on output pin.

◆ **comparator_filter_t**

enum comparator_filter_t	
Comparator digital filtering sample clock divisor settings.	
Enumerator	
COMPARATOR_FILTER_OFF	Disable debounce filter.
COMPARATOR_FILTER_1	Filter using PCLK divided by 1, not supported by all implementations.
COMPARATOR_FILTER_8	Filter using PCLK divided by 8.
COMPARATOR_FILTER_16	Filter using PCLK divided by 16, not supported by all implementations.
COMPARATOR_FILTER_32	Filter using PCLK divided by 32.

◆ **comparator_state_t**

enum comparator_state_t	
Current comparator state.	
Enumerator	
COMPARATOR_STATE_OUTPUT_LOW	$VCMP < VREF$ if polarity is not inverted, $VCMP > VREF$ if inverted.
COMPARATOR_STATE_OUTPUT_HIGH	$VCMP > VREF$ if polarity is not inverted, $VCMP < VREF$ if inverted.
COMPARATOR_STATE_OUTPUT_DISABLED	comparator_api_t::outputEnable() has not been called

4.3.5 CRC Interface

Interfaces

Detailed Description

Interface for cyclic redundancy checking.

Summary

The CRC (Cyclic Redundancy Check) calculator generates CRC codes using five different polynomials including 8 bit, 16 bit, and 32 bit variations. Calculation can be performed by sending data to the block using the CPU or by snooping on read or write activity on one of 10 SCI channels.

Implemented by:

- Cyclic Redundancy Check (CRC) Calculator ([r_crc](#))

Data Structures

struct	crc_cfg_t
struct	crc_api_t
struct	crc_instance_t

Typedefs

typedef void	crc_ctrl_t

Enumerations

enum	crc_polynomial_t
enum	crc_bit_order_t
enum	crc_snoop_direction_t

Data Structure Documentation

◆ [crc_cfg_t](#)

struct crc_cfg_t		
User configuration structure, used in open function		
Data Fields		
crc_polynomial_t	polynomial	CRC Generating Polynomial Switching (GPS)

crc_bit_order_t	bit_order	CRC Calculation Switching (LMS)
crc_snoop_address_t	snoop_address	Register Snoop Address (CRCSA)
void const *	p_extend	CRC Hardware Dependent Configuration.

◆ [crc_api_t](#)

struct [crc_api_t](#)

CRC driver structure. General CRC functions implemented at the HAL layer will follow this API.

Data Fields

fsp_err_t (*	open)(crc_ctrl_t *const p_ctrl, crc_cfg_t const *const p_cfg)
fsp_err_t (*	close)(crc_ctrl_t *const p_ctrl)
fsp_err_t (*	crcResultGet)(crc_ctrl_t *const p_ctrl, uint32_t *crc_result)
fsp_err_t (*	snoopEnable)(crc_ctrl_t *const p_ctrl, uint32_t crc_seed)
fsp_err_t (*	snoopDisable)(crc_ctrl_t *const p_ctrl)
fsp_err_t (*	calculate)(crc_ctrl_t *const p_ctrl, crc_input_t *const p_crc_input, uint32_t *p_crc_result)
fsp_err_t (*	versionGet)(fsp_version_t *version)

Field Documentation

◆ open

```
fsp_err_t(* crc_api_t::open) (crc_ctrl_t *const p_ctrl, crc_cfg_t const *const p_cfg)
```

Open the CRC driver module.

Implemented as

◦ R_CRC_Open()

Parameters

[in]	p_ctrl	Pointer to CRC device handle.
[in]	p_cfg	Pointer to a configuration structure.

◆ close

```
fsp_err_t(* crc_api_t::close) (crc_ctrl_t *const p_ctrl)
```

Close the CRC module driver

Implemented as

◦ R_CRC_Close()

Parameters

[in]	p_ctrl	Pointer to crc device handle

Return values

FSP_SUCCESS	Configuration was successful.

◆ crcResultGet

```
fsp_err_t(* crc_api_t::crcResultGet) (crc_ctrl_t *const p_ctrl, uint32_t *crc_result)
```

Return the current calculated value.

Implemented as

◦ R_CRC_CalculatedValueGet()

Parameters

[in]	p_ctrl	Pointer to CRC device handle.
[out]	crc_result	The calculated value from the last CRC calculation.

◆ **snoopEnable**

```
fsp_err_t(* crc_api_t::snoopEnable) (crc_ctrl_t *const p_ctrl, uint32_t crc_seed)
```

Configure and Enable snooping.

Implemented as

◦ R_CRC_SnoopEnable()

Parameters

[in]	p_ctrl	Pointer to CRC device handle.
[in]	crc_seed	CRC seed.

◆ **snoopDisable**

```
fsp_err_t(* crc_api_t::snoopDisable) (crc_ctrl_t *const p_ctrl)
```

Disable snooping.

Implemented as

◦ R_CRC_SnoopDisable()

Parameters

[in]	p_ctrl	Pointer to CRC device handle.

◆ **calculate**

```
fsp_err_t(* crc_api_t::calculate) (crc_ctrl_t *const p_ctrl, crc_input_t *const p_crc_input, uint32_t *p_crc_result)
```

Perform a CRC calculation on a block of data.

Implemented as

◦ R_CRC_Calculate()

Parameters

[in]	p_ctrl	Pointer to crc device handle.
[in]	p_crc_input	A pointer to structure for CRC inputs
[out]	crc_result	The calculated value of the CRC calculation.

◆ **versionGet**

```
fsp_err_t(* crc_api_t::versionGet) (fsp_version_t *version)
```

Get the driver version based on compile time macros.

Implemented as

- [R_CRC_VersionGet\(\)](#)

◆ **crc_instance_t**

```
struct crc_instance_t
```

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields

crc_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
crc_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
crc_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation◆ **crc_ctrl_t**

```
typedef void crc\_ctrl\_t
```

CRC control block. Allocate an instance specific control block to pass into the CRC API calls.

Implemented as

- [crc_instance_ctrl_t](#)

Enumeration Type Documentation

◆ **crc_polynomial_t**

enum crc_polynomial_t	
CRC Generating Polynomial Switching (GPS).	
Enumerator	
CRC_POLYNOMIAL_CRC_8	8-bit CRC-8 ($X^8 + X^2 + X + 1$)
CRC_POLYNOMIAL_CRC_16	16-bit CRC-16 ($X^{16} + X^{15} + X^2 + 1$)
CRC_POLYNOMIAL_CRC_CCITT	16-bit CRC-CCITT ($X^{16} + X^{12} + X^5 + 1$)
CRC_POLYNOMIAL_CRC_32	32-bit CRC-32 ($X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + X + 1$)
CRC_POLYNOMIAL_CRC_32C	32-bit CRC-32C ($X^{32} + X^{28} + X^{27} + X^{26} + X^{25} + X^{23} + X^{22} + X^{20} + X^{19} + X^{18} + X^{14} + X^{13} + X^{11} + X^{10} + X^9 + X^8 + X^6 + 1$)

◆ **crc_bit_order_t**

enum crc_bit_order_t	
CRC Calculation Switching (LMS)	
Enumerator	
CRC_BIT_ORDER_LMS_LSB	Generates CRC for LSB first communication.
CRC_BIT_ORDER_LMS_MSB	Generates CRC for MSB first communication.

◆ **crc_snoop_direction_t**

enum crc_snoop_direction_t	
Snoop-On-Write/Read Switch (CRCSWR)	
Enumerator	
CRC_SNOOP_DIRECTION_RECEIVE	Snoop-on-read.
CRC_SNOOP_DIRECTION_TRANSMIT	Snoop-on-write.

4.3.6 CTSU Interface

Interfaces

Detailed Description

Interface for Capacitive Touch Sensing Unit (CTSU) functions.

Summary

The CTSU interface provides CTSU functionality.

The CTSU interface can be implemented by:

- [Capacitive Touch Sensing Unit \(r_ctsu\)](#)

Data Structures

struct	ctsu_callback_args_t

struct	ctsu_element_cfg_t

struct	ctsu_cfg_t

struct	ctsu_api_t

struct	ctsu_instance_t

Typedefs

typedef void	ctsu_ctrl_t

Enumerations

enum	ctsu_event_t

enum	ctsu_cap_t

enum	ctsu_txvsel_t

enum	ctsu_txvsel2_t

enum	ctsu_atune0_t

enum	ctsu_atune1_t

enum	ctsu_atune12_t

enum	ctsu_clk_t

enum	ctsu_md_t

enum [ctsu_posel_t](#)enum [ctsu_ssdiv_t](#)

Data Structure Documentation

◆ [ctsu_callback_args_t](#)

struct ctsu_callback_args_t		
Callback function parameter data		
Data Fields		
ctsu_event_t	event	The event can be used to identify what caused the callback.
void const *	p_context	Placeholder for user data. Set in CTSU_api_t::open function in ctsu_cfg_t .

◆ [ctsu_element_cfg_t](#)

struct ctsu_element_cfg_t		
CTSU Configuration parameters. Element Configuration		
Data Fields		
ctsu_ssdiv_t	ssdiv	CTSU Spectrum Diffusion Frequency Division Setting (CTSU Only)
uint16_t	so	CTSU Sensor Offset Adjustment.
uint8_t	snum	CTSU Measurement Count Setting.
uint8_t	sdpa	CTSU Base Clock Setting.

◆ [ctsu_cfg_t](#)

struct ctsu_cfg_t		
User configuration structure, used in open function		
Data Fields		
ctsu_cap_t	cap	
	CTSU Measurement Operation Start Trigger Select.	
ctsu_txvsel_t	txvsel	
	CTSU Transmission Power Supply Select.	

ctsu_txvsel2_t	txvsel2
	CTSU Transmission Power Supply Select 2 (CTS2 Only)
ctsu_atune0_t	atune0
	CTSU Power Supply Operating Mode Setting.
ctsu_atune1_t	atune1
	CTSU Power Supply Capacity Adjustment (CTS2 Only)
ctsu_atune12_t	atune12
	CTSU Power Supply Capacity Adjustment (CTS2 Only)
ctsu_clk_t	clk
	CTSU Operating Clock Select.
ctsu_md_t	md
	CTSU Measurement Mode Select.
ctsu_posel_t	posel
	CTSU Non-Measured Channel Output Select (CTS2 Only)
uint8_t	ctsuchac0
	TS00-TS07 enable mask.
uint8_t	ctsuchac1
	TS08-TS15 enable mask.

uint8_t	ctsuhac2
	TS16-TS23 enable mask.
uint8_t	ctsuhac3
	TS24-TS31 enable mask.
uint8_t	ctsuhac4
	TS32-TS39 enable mask.
uint8_t	ctsuhtrc0
	TS00-TS07 mutual-tx mask.
uint8_t	ctsuhtrc1
	TS08-TS15 mutual-tx mask.
uint8_t	ctsuhtrc2
	TS16-TS23 mutual-tx mask.
uint8_t	ctsuhtrc3
	TS24-TS31 mutual-tx mask.
uint8_t	ctsuhtrc4
	TS32-TS39 mutual-tx mask.
ctsuh_element_cfg_t const *	p_elements
	Pointer to elements configuration array.
uint8_t	num_rx

	Number of receive terminals.
uint8_t	num_tx
	Number of transmit terminals.
uint16_t	threshold_3freq
	CTSU majority threshold at three frequency (CTSU2 Only)
uint16_t	num_moving_average
	Number of moving average for measurement data.
bool	tunning_enable
	Initial offset tuning flag.
uint8_t	number
	Configuration number for QE monitor.
void(*	p_callback)(ctsu_callback_args_t *p_args)
	Callback provided when CTSUFN ISR occurs.
transfer_instance_t const *	p_transfer_tx
	DTC instance for transmit at CTSUWR. Set to NULL if unused.
transfer_instance_t const *	p_transfer_rx
	DTC instance for receive at CTSURD. Set to NULL if unused.
IRQn_Type	write_irq
	CTSU_CTSUWR interrupt vector.

IRQn_Type	read_irq
	CTSU_CTSURD interrupt vector.
IRQn_Type	end_irq
	CTSU_CTSUFN interrupt vector.
void const *	p_context
	User defined context passed into callback function.
void const *	p_extend
	Pointer to extended configuration by instance of interface.

◆ ctsu_api_t

struct ctsu_api_t

Functions implemented at the HAL layer will follow this API.

Data Fields

[fsp_err_t](#)(* [open](#))(ctsu_ctrl_t *const p_ctrl, [ctsu_cfg_t](#) const *const p_cfg)

[fsp_err_t](#)(* [scanStart](#))(ctsu_ctrl_t *const p_ctrl)

[fsp_err_t](#)(* [dataGet](#))(ctsu_ctrl_t *const p_ctrl, uint16_t *p_data)

[fsp_err_t](#)(* [close](#))(ctsu_ctrl_t *const p_ctrl)

[fsp_err_t](#)(* [versionGet](#))(fsp_version_t *const p_data)

Field Documentation

◆ **open**

```
fsp_err_t(* ctsu_api_t::open) (ctsu_ctrl_t *const p_ctrl, ctsu_cfg_t const *const p_cfg)
```

Open driver.

Implemented as

◦ R_CTSU_Open()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_cfg	Pointer to pin configuration structure.

◆ **scanStart**

```
fsp_err_t(* ctsu_api_t::scanStart) (ctsu_ctrl_t *const p_ctrl)
```

Scan start.

Implemented as

◦ R_CTSU_ScanStart()

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ **dataGet**

```
fsp_err_t(* ctsu_api_t::dataGet) (ctsu_ctrl_t *const p_ctrl, uint16_t *p_data)
```

Data get.

Implemented as

◦ R_CTSU_DataGet()

Parameters

[in]	p_ctrl	Pointer to control structure.
[out]	p_data	Pointer to get data array.

◆ **close**

```
fsp_err_t(* ctsu_api_t::close) (ctsu_ctrl_t *const p_ctrl)
```

Close driver.

Implemented as

◦ [R_CTSU_Close\(\)](#)

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ **versionGet**

```
fsp_err_t(* ctsu_api_t::versionGet) (fsp_version_t *const p_data)
```

Return the version of the driver.

Implemented as

◦ [R_CTSU_VersionGet\(\)](#)

Parameters

[in]	p_ctrl	Pointer to control structure.
[out]	p_data	Memory address to return version information to.

◆ **ctsu_instance_t**

```
struct ctsu_instance_t
```

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields

ctsu_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
ctsu_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
ctsu_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ **ctsu_ctrl_t**typedef void [ctsu_ctrl_t](#)

CTSU Control block. Allocate an instance specific control block to pass into the API calls.

Implemented as

- [ctsu_instance_ctrl_t](#)

Enumeration Type Documentation◆ **ctsu_event_t**enum [ctsu_event_t](#)

CTSU Events for callback function

Enumerator

CTSU_EVENT_SCAN_COMPLETE	Normal end.
CTSU_EVENT_OVERFLOW	Sensor counter overflow (CTSUST.CTSUSOVF set)
CTSU_EVENT_ICOMP	Abnormal TSCAP voltage (CTSUERRS.CTSUICOMP set)
CTSU_EVENT_ICOMP1	Abnormal sensor current (CTSUSR.ICOMP1 set)

◆ **ctsu_cap_t**enum [ctsu_cap_t](#)

CTSU Measurement Operation Start Trigger Select

Enumerator

CTSU_CAP_SOFTWARE	Scan started by software trigger.
CTSU_CAP_EXTERNAL	Scan started by external trigger, DTC use only.

◆ **ctsu_txvsel_t**

enum ctsu_txvsel_t	
CTSU Transmission Power Supply Select	
Enumerator	
CTSU_TXVSEL_VCC	VCC selected.
CTSU_TXVSEL_INTERNAL_POWER	Internal logic power supply selected.

◆ **ctsu_txvsel2_t**

enum ctsu_txvsel2_t	
CTSU Transmission Power Supply Select 2 (CTSU2 Only)	
Enumerator	
CTSU_TXVSEL_MODE	Follow TXVSEL setting.
CTSU_TXVSEL_VCC_PRIVATE	VCC private selected.

◆ **ctsu_atune0_t**

enum ctsu_atune0_t	
CTSU Power Supply Operating Mode Setting	
Enumerator	
CTSU_ATUNE0_NORMAL	Normal operating mode.
CTSU_ATUNE0_LOW	Low-voltage operating mode.

◆ **ctsu_atune1_t**

enum ctsu_atune1_t	
CTSU Power Supply Capacity Adjustment (CTSU Only)	
Enumerator	
CTSU_ATUNE1_NORMAL	Normal output(40uA)
CTSU_ATUNE1_HIGH	High-current output(80uA)

◆ **ctsu_atune12_t**

enum ctsu_atune12_t	
CTSU Power Supply Capacity Adjustment (CTSU2 Only)	
Enumerator	
CTSU_ATUNE12_80UA	High-current output(80uA)
CTSU_ATUNE12_40UA	Normal output(40uA)
CTSU_ATUNE12_20UA	Low-current output(20uA)
CTSU_ATUNE12_160UA	Very high-current output(160uA)

◆ **ctsu_clk_t**

enum ctsu_clk_t	
CTSU Operating Clock Select	
Enumerator	
CTSU_CLK_DIV_1	PCLKB.
CTSU_CLK_DIV_2	PCLKB divided by 2.
CTSU_CLK_DIV_4	PCLKB divided by 4.
CTSU_CLK_DIV_8	PCLKB divided by 8 (CTSU2 Only)

◆ **ctsu_md_t**

enum ctsu_md_t	
CTSU Measurement Mode Select	
Enumerator	
CTSU_MODE_SELF_MULTI_SCAN	Self-capacitance multi scan mode.
CTSU_MODE_MUTUAL_FULL_SCAN	Mutual capacitance full scan mode.
CTSU_MODE_MUTUAL_CFC_SCAN	Mutual capacitance cfc scan mode (CTSU2 Only)
CTSU_MODE_CURRENT_SCAN	Current scan mode (CTSU2 Only)

◆ **ctsu_posel_t**

enum ctsu_posel_t	
CTSU Non-Measured Channel Output Select (CTSU2 Only)	
Enumerator	
CTSU_POSEL_LOW_GPIO	Output low through GPIO.
CTSU_POSEL_HI_Z	Hi-Z.
CTSU_POSEL_LOW	Output low through the power setting by the TXVSEL[1:0] bits.
CTSU_POSEL_SAME_PULSE	Same phase pulse output as transmission channel through the power setting by the TXVSEL[1:0] bits.

◆ **ctsu_ssdiv_t**

enum ctsu_ssdiv_t	
CTSU Spectrum Diffusion Frequency Division Setting (CTSU Only)	
Enumerator	
CTSU_SSDIV_4000	4.00 <= Base clock frequency(MHz)
CTSU_SSDIV_2000	2.00 <= Base clock frequency(MHz) < 4.00
CTSU_SSDIV_1330	1.33 <= Base clock frequency(MHz) < 2.00
CTSU_SSDIV_1000	1.00 <= Base clock frequency(MHz) < 1.33
CTSU_SSDIV_0800	0.80 <= Base clock frequency(MHz) < 1.00
CTSU_SSDIV_0670	0.67 <= Base clock frequency(MHz) < 0.80
CTSU_SSDIV_0570	0.57 <= Base clock frequency(MHz) < 0.67
CTSU_SSDIV_0500	0.50 <= Base clock frequency(MHz) < 0.57
CTSU_SSDIV_0440	0.44 <= Base clock frequency(MHz) < 0.50
CTSU_SSDIV_0400	0.40 <= Base clock frequency(MHz) < 0.44
CTSU_SSDIV_0360	0.36 <= Base clock frequency(MHz) < 0.40
CTSU_SSDIV_0330	0.33 <= Base clock frequency(MHz) < 0.36
CTSU_SSDIV_0310	0.31 <= Base clock frequency(MHz) < 0.33
CTSU_SSDIV_0290	0.29 <= Base clock frequency(MHz) < 0.31
CTSU_SSDIV_0270	0.27 <= Base clock frequency(MHz) < 0.29
CTSU_SSDIV_0000	0.00 <= Base clock frequency(MHz) < 0.27

4.3.7 DAC Interface

Interfaces

Detailed Description

Interface for D/A converters.

Summary

The DAC interface provides standard Digital/Analog Converter functionality. A DAC application writes digital sample data to the device and generates analog output on the DAC output pin.

Implemented by: [Digital to Analog Converter \(r_dac\)](#)

Data Structures

struct [dac_info_t](#)

struct [dac_cfg_t](#)

struct [dac_api_t](#)

struct [dac_instance_t](#)

Typedefs

typedef void [dac_ctrl_t](#)

Enumerations

enum [dac_data_format_t](#)

Data Structure Documentation

◆ [dac_info_t](#)

struct dac_info_t		
DAC information structure to store various information for a DAC		
Data Fields		
uint8_t	bit_width	Resolution of the DAC.

◆ [dac_cfg_t](#)

struct dac_cfg_t		
DAC Open API configuration parameter		
Data Fields		
uint8_t	channel	ID associated with this DAC channel.
bool	ad_da_synchronized	AD/DA synchronization.
dac_data_format_t	data_format	Data format.
bool	output_amplifier_enabled	Output amplifier enable.
void const *	p_extend	

◆ dac_api_t

struct dac_api_t		
DAC driver structure. General DAC functions implemented at the HAL layer follow this API.		
Data Fields		
fsp_err_t(*)	open	((dac_ctrl_t *p_ctrl, dac_cfg_t const *const p_cfg)
fsp_err_t(*)	close	((dac_ctrl_t *p_ctrl)
fsp_err_t(*)	write	((dac_ctrl_t *p_ctrl, uint16_t value)
fsp_err_t(*)	start	((dac_ctrl_t *p_ctrl)
fsp_err_t(*)	stop	((dac_ctrl_t *p_ctrl)
fsp_err_t(*)	versionGet	((fsp_version_t *p_version)
fsp_err_t(*)	infoGet	((dac_info_t *const p_info)
Field Documentation		
◆ open		
fsp_err_t(*) dac_api_t::open) (dac_ctrl_t *p_ctrl, dac_cfg_t const *const p_cfg)		
Initial configuration.		
Implemented as		
◦ R_DAC_Open() ◦ R_DAC8_Open()		
Parameters		
[in]	p_ctrl	Pointer to control block. Must be declared by user. Elements set here.
[in]	p_cfg	Pointer to configuration structure. All elements of this structure must be set by user.

◆ close

```
fsp_err_t(* dac_api_t::close) (dac_ctrl_t *p_ctrl)
```

Close the D/A Converter.

Implemented as

- [R_DAC_Close\(\)](#)
- [R_DAC8_Close\(\)](#)

Parameters

[in]	p_ctrl	Control block set in dac_api_t::open call for this timer.

◆ write

```
fsp_err_t(* dac_api_t::write) (dac_ctrl_t *p_ctrl, uint16_t value)
```

Write sample value to the D/A Converter.

Implemented as

- [R_DAC_Write\(\)](#)
- [R_DAC8_Write\(\)](#)

Parameters

[in]	p_ctrl	Control block set in dac_api_t::open call for this timer.
[in]	value	Sample value to be written to the D/A Converter.

◆ start

```
fsp_err_t(* dac_api_t::start) (dac_ctrl_t *p_ctrl)
```

Start the D/A Converter if it has not been started yet.

Implemented as

- [R_DAC_Start\(\)](#)
- [R_DAC8_Start\(\)](#)

Parameters

[in]	p_ctrl	Control block set in dac_api_t::open call for this timer.

◆ stop

```
fsp_err_t(* dac_api_t::stop) (dac_ctrl_t *p_ctrl)
```

Stop the D/A Converter if the converter is running.

Implemented as

- [R_DAC_Stop\(\)](#)
- [R_DAC8_Stop\(\)](#)

Parameters

[in]	p_ctrl	Control block set in dac_api_t::open call for this timer.

◆ versionGet

```
fsp_err_t(* dac_api_t::versionGet) (fsp_version_t *p_version)
```

Get version and store it in provided pointer p_version.

Implemented as

- [R_DAC_VersionGet\(\)](#)
- [R_DAC8_VersionGet\(\)](#)

Parameters

[out]	p_version	Code and API version used.

◆ infoGet

```
fsp_err_t(* dac_api_t::infoGet) (dac_info_t *const p_info)
```

Get information about DAC Resolution and store it in provided pointer p_info.

Implemented as

- [R_DAC_InfoGet\(\)](#)
- [R_DAC8_InfoGet\(\)](#)

Parameters

[out]	p_info	Collection of information for this DAC.

◆ dac_instance_t

```
struct dac_instance_t
```

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields

dac_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.

<code>dac_cfg_t</code> const *	<code>p_cfg</code>	Pointer to the configuration structure for this instance.
<code>dac_api_t</code> const *	<code>p_api</code>	Pointer to the API structure for this instance.

Typedef Documentation

◆ `dac_ctrl_t`

typedef void <code>dac_ctrl_t</code>
DAC control block. Allocate an instance specific control block to pass into the DAC API calls.
Implemented as ◦ <code>dac_instance_ctrl_t</code>

Enumeration Type Documentation

◆ `dac_data_format_t`

enum <code>dac_data_format_t</code>	
DAC Open API data format settings.	
Enumerator	
<code>DAC_DATA_FORMAT_FLUSH_RIGHT</code>	LSB of data is flush to the right leaving the top 4 bits unused.
<code>DAC_DATA_FORMAT_FLUSH_LEFT</code>	MSB of data is flush to the left leaving the bottom 4 bits unused.

4.3.8 Display Interface

Interfaces

Detailed Description

Interface for LCD panel displays.

Summary

The display interface provides standard display functionality:

- Signal timing configuration for LCD panels with RGB interface.
- Dot clock source selection (internal or external) and frequency divider.

- Blending of multiple graphics layers on the background screen.
- Color correction (brightness/configuration/gamma correction).
- Interrupts and callback function.

Implemented by: [Graphics LCD Controller \(r_glcdc\)](#)

Data Structures

struct	display_timing_t
struct	display_color_t
struct	display_coordinate_t
struct	display_brightness_t
struct	display_contrast_t
struct	display_correction_t
struct	gamma_correction_t
struct	display_gamma_correction_t
struct	display_clut_t
struct	display_input_cfg_t
struct	display_output_cfg_t
struct	display_layer_t
struct	display_callback_args_t
struct	display_cfg_t
struct	display_runtime_cfg_t
struct	display_clut_cfg_t
struct	display_status_t
struct	display_api_t
struct	display_instance_t

Typedefs

typedef void	display_ctrl_t

Enumerations

enum [display_frame_layer_t](#)

enum [display_state_t](#)

enum [display_event_t](#)

enum [display_in_format_t](#)

enum [display_out_format_t](#)

enum [display_endian_t](#)

enum [display_color_order_t](#)

enum [display_signal_polarity_t](#)

enum [display_sync_edge_t](#)

enum [display_fade_control_t](#)

enum [display_fade_status_t](#)

Data Structure Documentation

◆ display_timing_t

struct display_timing_t		
Display signal timing setting		
Data Fields		
uint16_t	total_cyc	Total cycles in one line or total lines in one frame.
uint16_t	display_cyc	Active video cycles or lines.
uint16_t	back_porch	Back porch cycles or lines.
uint16_t	sync_width	Sync signal asserting width.
display_signal_polarity_t	sync_polarity	Sync signal polarity.

◆ display_color_t

struct display_color_t
RGB Color setting

◆ display_coordinate_t

struct display_coordinate_t

Contrast (gain) correction setting		
Data Fields		
int16_t	x	Coordinate X, this allows to set signed value.
int16_t	y	Coordinate Y, this allows to set signed value.

◆ **display_brightness_t**

struct display_brightness_t		
Brightness (DC) correction setting		
Data Fields		
bool	enable	Brightness Correction On/Off.
uint16_t	r	Brightness (DC) adjustment for R channel.
uint16_t	g	Brightness (DC) adjustment for G channel.
uint16_t	b	Brightness (DC) adjustment for B channel.

◆ **display_contrast_t**

struct display_contrast_t		
Contrast (gain) correction setting		
Data Fields		
bool	enable	Contrast Correction On/Off.
uint8_t	r	Contrast (gain) adjustment for R channel.
uint8_t	g	Contrast (gain) adjustment for G channel.
uint8_t	b	Contrast (gain) adjustment for B channel.

◆ **display_correction_t**

struct display_correction_t		
Color correction setting		
Data Fields		
display_brightness_t	brightness	Brightness.
display_contrast_t	contrast	Contrast.

◆ **gamma_correction_t**

struct gamma_correction_t		

Gamma correction setting for each color		
Data Fields		
bool	enable	Gamma Correction On/Off.
uint16_t *	gain	Gain adjustment.
uint16_t *	threshold	Start threshold.

◆ display_gamma_correction_t

struct display_gamma_correction_t		
Gamma correction setting		
Data Fields		
gamma_correction_t	r	Gamma correction for R channel.
gamma_correction_t	g	Gamma correction for G channel.
gamma_correction_t	b	Gamma correction for B channel.

◆ display_clut_t

struct display_clut_t		
CLUT setting		
Data Fields		
uint32_t	color_num	The number of colors in CLUT.
const uint32_t *	p_clut	Address of the area storing the CLUT data (in ARGB8888 format)

◆ display_input_cfg_t

struct display_input_cfg_t		
Graphics plane input configuration structure		
Data Fields		
uint32_t *	p_base	Base address to the frame buffer.
uint16_t	hsize	Horizontal pixel size in a line.
uint16_t	vsize	Vertical pixel size in a frame.
uint32_t	hstride	Memory stride (bytes) in a line.
display_in_format_t	format	Input format setting.
bool	line_descending_enable	Line descending enable.
bool	lines_repeat_enable	Line repeat enable.

uint16_t	lines_repeat_times	Expected number of line repeating.

◆ **display_output_cfg_t**

struct display_output_cfg_t		
Display output configuration structure		
Data Fields		
display_timing_t	htiming	Horizontal display cycle setting.
display_timing_t	vtiming	Vertical display cycle setting.
display_out_format_t	format	Output format setting.
display_endian_t	endian	Bit order of output data.
display_color_order_t	color_order	Color order in pixel.
display_signal_polarity_t	data_enable_polarity	Data Enable signal polarity.
display_sync_edge_t	sync_edge	Signal sync edge selection.
display_color_t	bg_color	Background color.
display_brightness_t	brightness	Brightness setting.
display_contrast_t	contrast	Contrast setting.
display_gamma_correction_t *	p_gamma_correction	Pointer to gamma correction setting.
bool	dithering_on	Dithering on/off.

◆ **display_layer_t**

struct display_layer_t		
Graphics layer blend setup parameter structure		
Data Fields		
display_coordinate_t	coordinate	Blending location (starting point of image)
display_color_t	bg_color	Color outside region.
display_fade_control_t	fade_control	Layer fade-in/out control on/off.
uint8_t	fade_speed	Layer fade-in/out frame rate.

◆ **display_callback_args_t**

struct display_callback_args_t		
Display callback parameter definition		
Data Fields		
display_event_t	event	Event code.
void const *	p_context	Context provided to user during

callback.

◆ **display_cfg_t**

struct display_cfg_t

Display main configuration structure

Data Fields

display_input_cfg_t	input [2]
	Graphics input frame setting. More...
display_output_cfg_t	output
	Graphics output frame setting.
display_layer_t	layer [2]
	Graphics layer blend setting.
uint8_t	line_detect_ipl
	Line detect interrupt priority.
uint8_t	underflow_1_ipl
	Underflow 1 interrupt priority.
uint8_t	underflow_2_ipl
	Underflow 2 interrupt priority.
IRQn_Type	line_detect_irq
	Line detect interrupt vector.
IRQn_Type	underflow_1_irq
	Underflow 1 interrupt vector.

IRQn_Type	underflow_2_irq
	Underflow 2 interrupt vector.
void(*	p_callback)(display_callback_args_t *p_args)
	Pointer to callback function. More...
void const *	p_context
	User defined context passed into callback function.
void const *	p_extend
	Display hardware dependent configuration. More...

Field Documentation

◆ input

[display_input_cfg_t](#) display_cfg_t::input[2]

Graphics input frame setting.

Generic configuration for display devices

◆ p_callback

void(* display_cfg_t::p_callback) ([display_callback_args_t](#) *p_args)

Pointer to callback function.

Configuration for display event processing

◆ p_extend

void const* display_cfg_t::p_extend

Display hardware dependent configuration.

Pointer to display peripheral specific configuration

◆ display_runtime_cfg_t

struct display_runtime_cfg_t

Display main configuration structure

Data Fields

display_input_cfg_t	input	Graphics input frame setting. Generic configuration for display devices
display_layer_t	layer	Graphics layer alpha blending setting.

◆ **display_clut_cfg_t**

struct display_clut_cfg_t		
Display CLUT configuration structure		
Data Fields		
uint32_t *	p_base	Pointer to CLUT source data.
uint16_t	start	Beginning of CLUT entry to be updated.
uint16_t	size	Size of CLUT entry to be updated.

◆ **display_status_t**

struct display_status_t		
Display Status		
Data Fields		
display_state_t	state	Status of GLCDC module.
display_fade_status_t	fade_status[DISPLAY_FRAME_LAYER_2 +1]	Status of fade-in/fade-out status.

◆ **display_api_t**

struct display_api_t		
Shared Interface definition for display peripheral		
Data Fields		
fsp_err_t (*	open)(display_ctrl_t *const p_ctrl, display_cfg_t const *const p_cfg)	
fsp_err_t (*	close)(display_ctrl_t *const p_ctrl)	
fsp_err_t (*	start)(display_ctrl_t *const p_ctrl)	
fsp_err_t (*	stop)(display_ctrl_t *const p_ctrl)	
fsp_err_t (*	layerChange)(display_ctrl_t const *const p_ctrl, display_runtime_cfg_t const *const p_cfg, display_frame_layer_t frame)	

<code>fsp_err_t(*</code>	<code>bufferChange)(display_ctrl_t const *const p_ctrl, uint8_t *const framebuffer, display_frame_layer_t frame)</code>
<code>fsp_err_t(*</code>	<code>correction)(display_ctrl_t const *const p_ctrl, display_correction_t const *const p_param)</code>
<code>fsp_err_t(*</code>	<code>clut)(display_ctrl_t const *const p_ctrl, display_clut_cfg_t const *const p_clut_cfg, display_frame_layer_t frame)</code>
<code>fsp_err_t(*</code>	<code>statusGet)(display_ctrl_t const *const p_ctrl, display_status_t *const p_status)</code>
<code>fsp_err_t(*</code>	<code>versionGet)(fsp_version_t *p_version)</code>

Field Documentation

◆ open

`fsp_err_t(* display_api_t::open) (display_ctrl_t *const p_ctrl, display_cfg_t const *const p_cfg)`

Open display device.

Implemented as

- `R_GLCDC_Open()`

Parameters

[in,out]	<code>p_ctrl</code>	Pointer to display interface control block. Must be declared by user. Value set here.
[in]	<code>p_cfg</code>	Pointer to display configuration structure. All elements of this structure must be set by user.

◆ close

```
fsp_err_t(* display_api_t::close) (display_ctrl_t *const p_ctrl)
```

Close display device.

Implemented as

- R_GLCDC_Close()

Parameters

[in]	p_ctrl	Pointer to display interface control block.

◆ start

```
fsp_err_t(* display_api_t::start) (display_ctrl_t *const p_ctrl)
```

Display start.

Implemented as

- R_GLCDC_Start()

Parameters

[in]	p_ctrl	Pointer to display interface control block.

◆ stop

```
fsp_err_t(* display_api_t::stop) (display_ctrl_t *const p_ctrl)
```

Display stop.

Implemented as

- R_GLCDC_Stop()

Parameters

[in]	p_ctrl	Pointer to display interface control block.

◆ **layerChange**

```
fsp_err_t(* display_api_t::layerChange) (display_ctrl_t const *const p_ctrl, display_runtime_cfg_t const *const p_cfg, display_frame_layer_t frame)
```

Change layer parameters at runtime.

Implemented as

◦ [R_GLCDC_LayerChange\(\)](#)

Parameters

[in]	p_ctrl	Pointer to display interface control block.
[in]	p_cfg	Pointer to run-time layer configuration structure.
[in]	frame	Number of graphic frames.

◆ **bufferChange**

```
fsp_err_t(* display_api_t::bufferChange) (display_ctrl_t const *const p_ctrl, uint8_t *const framebuffer, display_frame_layer_t frame)
```

Change layer framebuffer pointer.

Implemented as

◦ [R_GLCDC_BufferChange\(\)](#)

Parameters

[in]	p_ctrl	Pointer to display interface control block.
[in]	framebuffer	Pointer to desired framebuffer.
[in]	frame	Number of graphic frames.

◆ **correction**

```
fsp_err_t(* display_api_t::correction) (display_ctrl_t const *const p_ctrl, display_correction_t const *const p_param)
```

Color correction.

Implemented as

◦ [R_GLCDC_ColorCorrection\(\)](#)

Parameters

[in]	p_ctrl	Pointer to display interface control block.
[in]	param	Pointer to color correction configuration structure.

◆ **clut**

```
fsp_err_t(* display_api_t::clut) (display_ctrl_t const *const p_ctrl, display_clut_cfg_t const *const p_clut_cfg, display_frame_layer_t frame)
```

Set CLUT for display device.

Implemented as

◦ [R_GLCDC_ClutUpdate\(\)](#)

Parameters

[in]	p_ctrl	Pointer to display interface control block.
[in]	p_clut_cfg	Pointer to CLUT configuration structure.
[in]	frame	Number of frame buffer corresponding to the CLUT.

◆ **statusGet**

```
fsp_err_t(* display_api_t::statusGet) (display_ctrl_t const *const p_ctrl, display_status_t *const p_status)
```

Get status for display device.

Implemented as

◦ [R_GLCDC_StatusGet\(\)](#)

Parameters

[in]	p_ctrl	Pointer to display interface control block.
[in]	status	Pointer to display interface status structure.

◆ **versionGet**

```
fsp_err_t(* display_api_t::versionGet) (fsp_version_t *p_version)
```

Get version.

Implemented as

◦ [R_GLCDC_VersionGet\(\)](#)

Parameters

[in]	p_version	Pointer to the memory to store the version information.

◆ **display_instance_t**

```
struct display_instance_t
```

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields

display_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
display_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
display_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ **display_ctrl_t**typedef void [display_ctrl_t](#)

Display control block. Allocate an instance specific control block to pass into the display API calls.

Implemented as

- [glcdc_instance_ctrl_t](#) Display control block

Enumeration Type Documentation◆ **display_frame_layer_t**enum [display_frame_layer_t](#)

Display frame number

Enumerator

DISPLAY_FRAME_LAYER_1

Frame layer 1.

DISPLAY_FRAME_LAYER_2

Frame layer 2.

◆ **display_state_t**enum [display_state_t](#)

Display interface operation state

Enumerator

DISPLAY_STATE_CLOSED

Display closed.

DISPLAY_STATE_OPENED

Display opened.

DISPLAY_STATE_DISPLAYING

Displaying.

◆ **display_event_t**

enum <code>display_event_t</code>	
Display event codes	
Enumerator	
DISPLAY_EVENT_GR1_UNDERFLOW	Graphics frame1 underflow occurs.
DISPLAY_EVENT_GR2_UNDERFLOW	Graphics frame2 underflow occurs.
DISPLAY_EVENT_LINE_DETECTION	Designated line is processed.

◆ **display_in_format_t**

enum <code>display_in_format_t</code>	
Input format setting	
Enumerator	
DISPLAY_IN_FORMAT_32BITS_ARGB8888	ARGB8888, 32 bits.
DISPLAY_IN_FORMAT_32BITS_RGB888	RGB888, 32 bits.
DISPLAY_IN_FORMAT_16BITS_RGB565	RGB565, 16 bits.
DISPLAY_IN_FORMAT_16BITS_ARGB1555	ARGB1555, 16 bits.
DISPLAY_IN_FORMAT_16BITS_ARGB4444	ARGB4444, 16 bits.
DISPLAY_IN_FORMAT_CLUT8	CLUT8.
DISPLAY_IN_FORMAT_CLUT4	CLUT4.
DISPLAY_IN_FORMAT_CLUT1	CLUT1.

◆ **display_out_format_t**

enum <code>display_out_format_t</code>	
Output format setting	
Enumerator	
DISPLAY_OUT_FORMAT_24BITS_RGB888	RGB888, 24 bits.
DISPLAY_OUT_FORMAT_18BITS_RGB666	RGB666, 18 bits.
DISPLAY_OUT_FORMAT_16BITS_RGB565	RGB565, 16 bits.
DISPLAY_OUT_FORMAT_8BITS_SERIAL	SERIAL, 8 bits.

◆ **display_endian_t**

enum <code>display_endian_t</code>	
Data endian select	
Enumerator	
DISPLAY_ENDIAN_LITTLE	Little-endian.
DISPLAY_ENDIAN_BIG	Big-endian.

◆ **display_color_order_t**

enum <code>display_color_order_t</code>	
RGB color order select	
Enumerator	
DISPLAY_COLOR_ORDER_RGB	Color order RGB.
DISPLAY_COLOR_ORDER_BGR	Color order BGR.

◆ **display_signal_polarity_t**

enum display_signal_polarity_t	
Polarity of a signal select	
Enumerator	
DISPLAY_SIGNAL_POLARITY_LOACTIVE	Low active signal.
DISPLAY_SIGNAL_POLARITY_HIACTIVE	High active signal.

◆ **display_sync_edge_t**

enum display_sync_edge_t	
Signal synchronization edge select	
Enumerator	
DISPLAY_SIGNAL_SYNC_EDGE_RISING	Signal is synchronized to rising edge.
DISPLAY_SIGNAL_SYNC_EDGE_FALLING	Signal is synchronized to falling edge.

◆ **display_fade_control_t**

enum display_fade_control_t	
Fading control	
Enumerator	
DISPLAY_FADE_CONTROL_NONE	Applying no fading control.
DISPLAY_FADE_CONTROL_FADEIN	Applying fade-in control.
DISPLAY_FADE_CONTROL_FADEOUT	Applying fade-out control.

◆ display_fade_status_t

enum display_fade_status_t	
Fading status	
Enumerator	
DISPLAY_FADE_STATUS_NOT_UNDERWAY	Fade-in/fade-out is not in progress.
DISPLAY_FADE_STATUS_FADING_UNDERWAY	Fade-in or fade-out is in progress.
DISPLAY_FADE_STATUS_PENDING	Fade-in/fade-out is configured but not yet started.

4.3.9 DOC Interface

Interfaces

Detailed Description

Interface for the Data Operation Circuit.

Defines the API and data structures for the DOC implementation of the Data Operation Circuit (DOC) interface.

Summary

This module implements the DOC_API using the Data Operation Circuit (DOC).

Implemented by: [Data Operation Circuit \(r_doc\)](#)

Data Structures

struct	doc_status_t
struct	doc_callback_args_t
struct	doc_cfg_t
struct	doc_api_t
struct	doc_instance_t

Macros

#define	DOC_API_VERSION_MAJOR

Typedefs

```
typedef void doc\_ctrl\_t
```

Enumerations

```
enum doc\_event\_t
```

Data Structure Documentation

◆ [doc_status_t](#)

```
struct doc\_status\_t
```

DOC status

◆ [doc_callback_args_t](#)

```
struct doc\_callback\_args\_t
```

Callback function parameter data.

Data Fields

void const *	p_context	Set in doc_api_t::open function in doc_cfg_t . Placeholder for user data.

◆ [doc_cfg_t](#)

```
struct doc\_cfg\_t
```

User configuration structure, used in the open function.

Data Fields

doc_event_t	event
	Select enumerated value from doc_event_t .
uint16_t	doc_data
	initial/reference value for DODSR register.
uint8_t	ipl
	DOC interrupt priority.
IRQn_Type	irq

	NVIC interrupt number assigned to this instance.
void(*	p_callback)(doc_callback_args_t *p_args)
void const *	p_context

Field Documentation

◆ p_callback

void(* doc_cfg_t::p_callback) (doc_callback_args_t *p_args)

Callback provided when a DOC ISR occurs.

◆ p_context

void const* doc_cfg_t::p_context

Placeholder for user data. Passed to the user callback in [doc_callback_args_t](#).

◆ doc_api_t

struct doc_api_t

Data Operation Circuit (DOC) API structure. DOC functions implemented at the HAL layer will follow this API.

Data Fields

fsp_err_t(*	open)(doc_ctrl_t *const p_ctrl, doc_cfg_t const *const p_cfg)
fsp_err_t(*	close)(doc_ctrl_t *const p_ctrl)
fsp_err_t(*	statusGet)(doc_ctrl_t *const p_ctrl, doc_status_t *p_status)
fsp_err_t(*	write)(doc_ctrl_t *const p_ctrl, uint16_t data)
fsp_err_t(*	versionGet)(fsp_version_t *const p_version)

Field Documentation

◆ open

```
fsp_err_t(* doc_api_t::open) (doc_ctrl_t *const p_ctrl, doc_cfg_t const *const p_cfg)
```

Initial configuration.

Implemented as

◦ R_DOC_Open()

Parameters

[in]	p_ctrl	Pointer to control block. Must be declared by user. Elements set here.
[in]	p_cfg	Pointer to configuration structure. All elements of this structure must be set by user.

◆ close

```
fsp_err_t(* doc_api_t::close) (doc_ctrl_t *const p_ctrl)
```

Allow the driver to be reconfigured. Will reduce power consumption.

Implemented as

◦ R_DOC_Close()

Parameters

[in]	p_ctrl	Control block set in doc_api_t::open call.

◆ statusGet

```
fsp_err_t(* doc_api_t::statusGet) (doc_ctrl_t *const p_ctrl, doc_status_t *p_status)
```

Gets the result of addition/subtraction and stores it in the provided pointer p_data.

Implemented as

◦ R_DOC_StatusGet()

Parameters

[in]	p_ctrl	Control block set in doc_api_t::open call.
[out]	p_data	Provides the 16 bit result of the addition/subtraction operation at the user defined location.

◆ **write**

```
fsp_err_t(* doc_api_t::write) (doc_ctrl_t *const p_ctrl, uint16_t data)
```

Write to the DODIR register.

Implemented as

◦ [R_DOC_Write\(\)](#)

Parameters

[in]	p_ctrl	Control block set in doc_api_t::open call.
[in]	data	data to be written to DOC DODIR register.

◆ **versionGet**

```
fsp_err_t(* doc_api_t::versionGet) (fsp_version_t *const p_version)
```

Get version and stores it in provided pointer p_version.

Implemented as

◦ [R_DOC_VersionGet\(\)](#)

Parameters

[out]	p_version	Code and API version used.

◆ **doc_instance_t**

```
struct doc_instance_t
```

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields

doc_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
doc_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
doc_api_t const *	p_api	Pointer to the API structure for this instance.

Macro Definition Documentation◆ **DOC_API_VERSION_MAJOR**

```
#define DOC_API_VERSION_MAJOR
```

Register definitions, common services and error codes.

Typedef Documentation

◆ doc_ctrl_t

```
typedef void doc_ctrl_t
```

DOC control block. Allocate an instance specific control block to pass into the DOC API calls.

Implemented as

- [doc_instance_ctrl_t](#)

Enumeration Type Documentation

◆ doc_event_t

```
enum doc_event_t
```

Event that can trigger a callback function.

Enumerator

DOC_EVENT_COMPARISON_MISMATCH	Comparison of data has resulted in a mismatch.
DOC_EVENT_ADDITION	Addition of data has resulted in a value greater than H'FFFF.
DOC_EVENT_SUBTRACTION	Subtraction of data has resulted in a value less than H'0000.
DOC_EVENT_COMPARISON_MATCH	Comparison of data has resulted in a match.

4.3.10 ELC Interface

Interfaces

Detailed Description

Interface for the Event Link Controller.

Data Structures

```
struct elc_cfg_t
```

```
struct elc_api_t
```

```
struct elc_instance_t
```

Typedefs

```
typedef void elc\_ctrl\_t
```

Enumerations

```
enum elc\_peripheral\_t
```

```
enum elc\_software\_event\_t
```

Data Structure Documentation

◆ [elc_cfg_t](#)

```
struct elc\_cfg\_t
```

Main configuration structure for the Event Link Controller

Data Fields

elc_event_t const	link[ELC_PERIPHERAL_NUM]	Event link register (ELSR) settings.

◆ [elc_api_t](#)

```
struct elc\_api\_t
```

ELC driver structure. General ELC functions implemented at the HAL layer follow this API.

Data Fields

fsp_err_t (*	open)(elc_ctrl_t *const p_ctrl, elc_cfg_t const *const p_cfg)

fsp_err_t (*	close)(elc_ctrl_t *const p_ctrl)

fsp_err_t (*	softwareEventGenerate)(elc_ctrl_t *const p_ctrl, elc_software_event_t event_num)

fsp_err_t (*	linkSet)(elc_ctrl_t *const p_ctrl, elc_peripheral_t peripheral, elc_event_t signal)

fsp_err_t (*	linkBreak)(elc_ctrl_t *const p_ctrl, elc_peripheral_t peripheral)

fsp_err_t (*	enable)(elc_ctrl_t *const p_ctrl)

fsp_err_t (*	disable)(elc_ctrl_t *const p_ctrl)

`fsp_err_t(* versionGet )(fsp_version_t *const p_version)`

Field Documentation

◆ open

`fsp_err_t(* elc_api_t::open) (elc_ctrl_t *const p_ctrl, elc_cfg_t const *const p_cfg)`

Initialize all links in the Event Link Controller.

Implemented as

◦ `R_ELC_Open()`

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_cfg	Pointer to configuration structure.

◆ close

`fsp_err_t(* elc_api_t::close) (elc_ctrl_t *const p_ctrl)`

Disable all links in the Event Link Controller and close the API.

Implemented as

◦ `R_ELC_Close()`

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ softwareEventGenerate

`fsp_err_t(* elc_api_t::softwareEventGenerate) (elc_ctrl_t *const p_ctrl, elc_software_event_t event_num)`

Generate a software event in the Event Link Controller.

Implemented as

◦ `R_ELC_SoftwareEventGenerate()`

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	eventNum	Software event number to be generated.

◆ **linkSet**

```
fsp_err_t(* elc_api_t::linkSet) (elc_ctrl_t *const p_ctrl, elc_peripheral_t peripheral, elc_event_t signal)
```

Create a single event link.

Implemented as

◦ [R_ELC_LinkSet\(\)](#)

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	peripheral	The peripheral block that will receive the event signal.
[in]	signal	The event signal.

◆ **linkBreak**

```
fsp_err_t(* elc_api_t::linkBreak) (elc_ctrl_t *const p_ctrl, elc_peripheral_t peripheral)
```

Break an event link.

Implemented as

◦ [R_ELC_LinkBreak\(\)](#)

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	peripheral	The peripheral that should no longer be linked.

◆ **enable**

```
fsp_err_t(* elc_api_t::enable) (elc_ctrl_t *const p_ctrl)
```

Enable the operation of the Event Link Controller.

Implemented as

◦ [R_ELC_Enable\(\)](#)

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ **disable**

```
fsp_err_t(* elc_api_t::disable) (elc_ctrl_t *const p_ctrl)
```

Disable the operation of the Event Link Controller.

Implemented as

◦ [R_ELC_Disable\(\)](#)

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ **versionGet**

```
fsp_err_t(* elc_api_t::versionGet) (fsp_version_t *const p_version)
```

Get the driver version based on compile time macros.

Implemented as

◦ [R_ELC_VersionGet\(\)](#)

Parameters

[in]	p_ctrl	Pointer to control structure.
[out]	p_version	is value returned.

◆ **elc_instance_t**

```
struct elc_instance_t
```

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields

elc_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
elc_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
elc_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation◆ **elc_ctrl_t**

```
typedef void elc_ctrl_t
```

ELC control block. Allocate an instance specific control block to pass into the ELC API calls.

Implemented as

◦ [elc_instance_ctrl_t](#)

Enumeration Type Documentation

◆ `elc_peripheral_t`

enum `elc_peripheral_t`

Possible peripherals to be linked to event signals (not all available on all MCUs)

◆ `elc_software_event_t`

enum `elc_software_event_t`

Software event number

Enumerator

`ELC_SOFTWARE_EVENT_0`

Software event 0.

`ELC_SOFTWARE_EVENT_1`

Software event 1.

4.3.11 Ethernet Interface

Interfaces

Detailed Description

Interface for Ethernet functions.

Summary

The Ethernet interface provides Ethernet functionality. The Ethernet interface supports the following features:

- Transmit/receive processing(Blocking and Non-Blocking)
- Callback function with returned event code
- Magic packet detection mode support
- Auto negotiation support
- Flow control support
- Multicast filtering support

Implemented by:

- [Ethernet \(`r\_ether`\)](#)

Data Structures

struct `ether_instance_descriptor_t`

struct [ether_callback_args_t](#)

struct [ether_cfg_t](#)

struct [ether_api_t](#)

struct [ether_instance_t](#)

Typedefs

typedef void [ether_ctrl_t](#)

Enumerations

enum [ether_wake_on_lan_t](#)

enum [ether_flow_control_t](#)

enum [ether_multicast_t](#)

enum [ether_promiscuous_t](#)

enum [ether_zerocopy_t](#)

enum [ether_event_t](#)

Data Structure Documentation

◆ ether_instance_descriptor_t

struct ether_instance_descriptor_t
EDMAC descriptor as defined in the hardware manual. Structure must be packed at 1 byte.

◆ ether_callback_args_t

struct ether_callback_args_t		
Callback function parameter data		
Data Fields		
uint32_t	channel	Device channel number.
ether_event_t	event	Event code.
uint32_t	status_ecsr	ETHERC status register for interrupt handler.
uint32_t	status_eesr	ETHERC/EDMAC status register for interrupt handler.
void const *	p_context	Placeholder for user data. Set in ether_api_t::open function in

ether_cfg_t.

◆ ether_cfg_t

struct ether_cfg_t

Configuration parameters.

Data Fields

uint8_t	channel
	Channel.
ether_zerocopy_t	zerocopy
	Zero copy enable or disable in Read/Write function.
ether_multicast_t	multicast
	Multicast enable or disable.
ether_promiscuous_t	promiscuous
	Promiscuous mode enable or disable.
ether_flow_control_t	flow_control
	Flow control functionally enable or disable.
uint32_t	broadcast_filter
	Limit of the number of broadcast frames received continuously.
uint8_t *	p_mac_address
	Pointer of MAC address.
ether_instance_descriptor_t *	p_rx_descriptors
	Transmission descriptor.

<code>ether_instance_descriptor_t *</code>	<code>p_tx_descriptors</code>
	Receive descriptor.
<code>uint8_t</code>	<code>num_tx_descriptors</code>
	Number of transmission descriptor.
<code>uint8_t</code>	<code>num_rx_descriptors</code>
	Number of receive descriptor.
<code>uint8_t **</code>	<code>pp_ether_buffers</code>
	Transmit and receive buffer.
<code>uint32_t</code>	<code>ether_buffer_size</code>
	Size of transmit and receive buffer.
<code>IRQn_Type</code>	<code>irq</code>
	NVIC interrupt number.
<code>uint32_t</code>	<code>interrupt_priority</code>
	NVIC interrupt priority.
<code>void(*</code>	<code>p_callback)(ether_callback_args_t *p_args)</code>
	Callback provided when an ISR occurs.
<code>ether_phy_instance_t const *</code>	<code>p_ether_phy_instance</code>
	Pointer to ETHER_PHY instance.

void const *	p_context
void const *	p_extend
	Placeholder for user extension.

Field Documentation

◆ [p_context](#)

void const* ether_cfg_t::p_context

Placeholder for user data. Passed to the user callback in [ether_callback_args_t](#).

◆ [ether_api_t](#)

struct ether_api_t

Functions implemented at the HAL layer will follow this API.

Data Fields

fsp_err_t (*	open)(ether_ctrl_t *const p_api_ctrl, ether_cfg_t const *const p_cfg)
fsp_err_t (*	close)(ether_ctrl_t *const p_api_ctrl)
fsp_err_t (*	read)(ether_ctrl_t *const p_api_ctrl, void **const pp_buffer, uint32_t *const length_bytes)
fsp_err_t (*	bufferRelease)(ether_ctrl_t *const p_api_ctrl)
fsp_err_t (*	write)(ether_ctrl_t *const p_api_ctrl, void *const p_buffer, uint32_t const frame_length)
fsp_err_t (*	linkProcess)(ether_ctrl_t *const p_api_ctrl)
fsp_err_t (*	wakeOnLANEnable)(ether_ctrl_t *const p_api_ctrl)
fsp_err_t (*	versionGet)(fsp_version_t *const p_data)

Field Documentation

◆ open

```
fsp_err_t(* ether_api_t::open) (ether_ctrl_t *const p_api_ctrl, ether_cfg_t const *const p_cfg)
```

Open driver.

Implemented as

◦ R_ETHER_Open()

Parameters

[in]	p_api_ctrl	Pointer to control structure.
[in]	p_cfg	Pointer to pin configuration structure.

◆ close

```
fsp_err_t(* ether_api_t::close) (ether_ctrl_t *const p_api_ctrl)
```

Close driver.

Implemented as

◦ R_ETHER_Close()

Parameters

[in]	p_api_ctrl	Pointer to control structure.

◆ read

```
fsp_err_t(* ether_api_t::read) (ether_ctrl_t *const p_api_ctrl, void **const pp_buffer, uint32_t *const length_bytes)
```

Read packet.

Implemented as

◦ R_ETHER_Read()

Parameters

[in]	p_api_ctrl	Pointer to control structure.
[in]	p_buffer	Pointer to where to store read data.
[in]	length_bytes	Number of bytes in buffer

◆ **bufferRelease**

```
fsp_err_t(* ether_api_t::bufferRelease) (ether_ctrl_t *const p_api_ctrl)
```

Release rx buffer from buffer pool process in zero copy read operation.

Implemented as

◦ [R_ETHER_BufferRelease\(\)](#)

Parameters

[in]	p_api_ctrl	Pointer to control structure.

◆ **write**

```
fsp_err_t(* ether_api_t::write) (ether_ctrl_t *const p_api_ctrl, void *const p_buffer, uint32_t const frame_length)
```

Write packet.

Implemented as

◦ [R_ETHER_Write\(\)](#)

Parameters

[in]	p_api_ctrl	Pointer to control structure.
[in]	p_buffer	Pointer to where to load write data.
[in]	frame_length	Send ethernet frame size(without 4 bytes of CRC data size).

◆ **linkProcess**

```
fsp_err_t(* ether_api_t::linkProcess) (ether_ctrl_t *const p_api_ctrl)
```

Process link.

Implemented as

◦ [R_ETHER_LinkProcess\(\)](#)

Parameters

[in]	p_api_ctrl	Pointer to control structure.

◆ **wakeOnLANEnable**

```
fsp_err_t(* ether_api_t::wakeOnLANEnable) (ether_ctrl_t *const p_api_ctrl)
```

Enable magic packet detection.

Implemented as

◦ [R_ETHER_WakeOnLANEnable\(\)](#)

Parameters

[in]	p_api_ctrl	Pointer to control structure.

◆ **versionGet**

```
fsp_err_t(* ether_api_t::versionGet) (fsp_version_t *const p_data)
```

Return the version of the driver.

Implemented as

◦ [R_ETHER_VersionGet\(\)](#)

Parameters

[in]	p_api_ctrl	Pointer to control structure.
[out]	p_data	Memory address to return version information to.

◆ **ether_instance_t**

```
struct ether_instance_t
```

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields

ether_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
ether_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
ether_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ **ether_ctrl_t**typedef void [ether_ctrl_t](#)

Control block. Allocate an instance specific control block to pass into the API calls.

Implemented as

- [ether_instance_ctrl_t](#)

Enumeration Type Documentation◆ **ether_wake_on_lan_t**enum [ether_wake_on_lan_t](#)

Wake on Lan

Enumerator

ETHER_WAKE_ON_LAN_DISABLE

Disable Wake on Lan.

ETHER_WAKE_ON_LAN_ENABLE

Enable Wake on Lan.

◆ **ether_flow_control_t**enum [ether_flow_control_t](#)

Flow control functionality

Enumerator

ETHER_FLOW_CONTROL_DISABLE

Disable flow control functionality.

ETHER_FLOW_CONTROL_ENABLE

Enable flow control functionality with pause frames.

◆ **ether_multicast_t**enum [ether_multicast_t](#)

Multicast Filter

Enumerator

ETHER_MULTICAST_DISABLE

Disable reception of multicast frames.

ETHER_MULTICAST_ENABLE

Enable reception of multicast frames.

◆ **ether_promiscuous_t**

enum ether_promiscuous_t	
Promiscuous Mode	
Enumerator	
ETHER_PROMISCUOUS_DISABLE	Only receive packets with current MAC address, multicast, and broadcast.
ETHER_PROMISCUOUS_ENABLE	Receive all packets.

◆ **ether_zerocopy_t**

enum ether_zerocopy_t	
Zero copy	
Enumerator	
ETHER_ZEROCOPY_DISABLE	Disable zero copy in Read/Write function.
ETHER_ZEROCOPY_ENABLE	Enable zero copy in Read/Write function.

◆ **ether_event_t**

enum ether_event_t	
Event code of callback function	
Enumerator	
ETHER_EVENT_WAKEON_LAN	Magic packet detection event.
ETHER_EVENT_LINK_ON	Link up detection event.
ETHER_EVENT_LINK_OFF	Link down detection event.
ETHER_EVENT_INTERRUPT	Interrupt event.

4.3.12 Ethernet PHY Interface[Interfaces](#)**Detailed Description**

Interface for Ethernet phy functions.

Summary

The Ethernet PHY interface provides Ethernet phy functionality. The Ethernet PHY interface supports the following features:

- Auto negotiation support
- Flow control support
- Link status check support

Implemented by:

- [Ethernet PHY \(r_ether_phy\)](#)

Data Structures

struct [ether_phy_cfg_t](#)

struct [ether_phy_api_t](#)

struct [ether_phy_instance_t](#)

Typedefs

typedef void [ether_phy_ctrl_t](#)

Enumerations

enum [ether_phy_flow_control_t](#)

enum [ether_phy_link_speed_t](#)

enum [ether_phy_mii_type_t](#)

Data Structure Documentation

◆ ether_phy_cfg_t

struct ether_phy_cfg_t		
Configuration parameters.		
Data Fields		
uint8_t	channel	Channel.
uint8_t	phy_lsi_address	Address of Phy-LSI.
uint32_t	phy_reset_wait_time	Wait time for Phy-LSI reboot.
int32_t	mii_bit_access_wait_time	Wait time for MII/RMII access.
ether_phy_flow_control_t	flow_control	Flow control functionally enable or disable.

ether_phy_mii_type_t	mii_type	Interface type is MII or RMII.
void const *	p_context	Placeholder for user data. Passed to the user callback in ether_phy_callback_args_t.
void const *	p_extend	Placeholder for user extension.

◆ ether_phy_api_t

struct ether_phy_api_t		
Functions implemented at the HAL layer will follow this API.		
Data Fields		
fsp_err_t (*	open)(ether_phy_ctrl_t *const p_api_ctrl, ether_phy_cfg_t const *const p_cfg)	
fsp_err_t (*	close)(ether_phy_ctrl_t *const p_api_ctrl)	
fsp_err_t (*	startAutoNegotiate)(ether_phy_ctrl_t *const p_api_ctrl)	
fsp_err_t (*	linkPartnerAbilityGet)(ether_phy_ctrl_t *const p_api_ctrl, uint32_t *const p_line_speed_duplex, uint32_t *const p_local_pause, uint32_t *const p_partner_pause)	
fsp_err_t (*	linkStatusGet)(ether_phy_ctrl_t *const p_api_ctrl)	
fsp_err_t (*	versionGet)(fsp_version_t *const p_data)	
Field Documentation		

◆ open

`fsp_err_t(* ether_phy_api_t::open) (ether_phy_ctrl_t *const p_api_ctrl, ether_phy_cfg_t const *const p_cfg)`

Open driver.

Implemented as

◦ `R_ETHER_PHY_Open()`

Parameters

[in]	p_api_ctrl	Pointer to control structure.
[in]	p_cfg	Pointer to pin configuration structure.

◆ close

`fsp_err_t(* ether_phy_api_t::close) (ether_phy_ctrl_t *const p_api_ctrl)`

Close driver.

Implemented as

◦ `R_ETHER_PHY_Close()`

Parameters

[in]	p_api_ctrl	Pointer to control structure.

◆ startAutoNegotiate

`fsp_err_t(* ether_phy_api_t::startAutoNegotiate) (ether_phy_ctrl_t *const p_api_ctrl)`

Start auto negotiation.

Implemented as

◦ `R_ETHER_PHY_StartAutoNegotiate()`

Parameters

[in]	p_api_ctrl	Pointer to control structure.

◆ **linkPartnerAbilityGet**

```
fsp_err_t(* ether_phy_api_t::linkPartnerAbilityGet) (ether_phy_ctrl_t *const p_api_ctrl, uint32_t *const p_line_speed_duplex, uint32_t *const p_local_pause, uint32_t *const p_partner_pause)
```

Get the partner ability.

Implemented as

◦ [R_ETHER_PHY_LinkPartnerAbilityGet\(\)](#)

Parameters

[in]	p_api_ctrl	Pointer to control structure.
[out]	p_line_speed_duplex	Pointer to the location of both the line speed and the duplex.
[out]	p_local_pause	Pointer to the location to store the local pause bits..
[out]	p_partner_pause	Pointer to the location to store the partner pause bits.

◆ **linkStatusGet**

```
fsp_err_t(* ether_phy_api_t::linkStatusGet) (ether_phy_ctrl_t *const p_api_ctrl)
```

Get Link status from phy-LSI interface.

Implemented as

◦ [R_ETHER_PHY_LinkStatusGet\(\)](#)

Parameters

[in]	p_api_ctrl	Pointer to control structure.

◆ **versionGet**

```
fsp_err_t(* ether_phy_api_t::versionGet) (fsp_version_t *const p_data)
```

Return the version of the driver.

Implemented as

◦ [R_ETHER_PHY_VersionGet\(\)](#)

Parameters

[out]	p_data	Memory address to return version information to.

◆ **ether_phy_instance_t**

```
struct ether_phy_instance_t
```

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields		
<code>ether_phy_ctrl_t *</code>	<code>p_ctrl</code>	Pointer to the control structure for this instance.
<code>ether_phy_cfg_t const *</code>	<code>p_cfg</code>	Pointer to the configuration structure for this instance.
<code>ether_phy_api_t const *</code>	<code>p_api</code>	Pointer to the API structure for this instance.

Typedef Documentation

◆ `ether_phy_ctrl_t`

typedef void `ether_phy_ctrl_t`

Control block. Allocate an instance specific control block to pass into the API calls.

Implemented as

- `ether_phy_instance_ctrl_t`

Enumeration Type Documentation

◆ `ether_phy_flow_control_t`

enum `ether_phy_flow_control_t`

Flow control functionality

Enumerator

<code>ETHER_PHY_FLOW_CONTROL_DISABLE</code>	Disable flow control functionality.
<code>ETHER_PHY_FLOW_CONTROL_ENABLE</code>	Enable flow control functionality with pause frames.

◆ ether_phy_link_speed_t

enum ether_phy_link_speed_t	
Link speed	
Enumerator	
ETHER_PHY_LINK_SPEED_NO_LINK	Link is not established.
ETHER_PHY_LINK_SPEED_10H	Link status is 10Mbit/s and half duplex.
ETHER_PHY_LINK_SPEED_10F	Link status is 10Mbit/s and full duplex.
ETHER_PHY_LINK_SPEED_100H	Link status is 100Mbit/s and half duplex.
ETHER_PHY_LINK_SPEED_100F	Link status is 100Mbit/s and full duplex.

◆ ether_phy_mii_type_t

enum ether_phy_mii_type_t	
Media-independent interface	
Enumerator	
ETHER_PHY_MII_TYPE_MII	MII.
ETHER_PHY_MII_TYPE_RMII	RMII.

4.3.13 External IRQ Interface

Interfaces

Detailed Description

Interface for detecting external interrupts.

Summary

The External IRQ Interface is for configuring interrupts to fire when a trigger condition is detected on an external IRQ pin.

The External IRQ Interface can be implemented by:

- [Interrupt Controller Unit \(r_icu\)](#)

Data Structures

struct [external_irq_callback_args_t](#)

struct [external_irq_cfg_t](#)

struct [external_irq_api_t](#)

struct [external_irq_instance_t](#)

Macros

#define [EXTERNAL_IRQ_API_VERSION_MAJOR](#)
EXTERNAL IRQ API version number (Major)

#define [EXTERNAL_IRQ_API_VERSION_MINOR](#)
EXTERNAL IRQ API version number (Minor)

Typedefs

typedef void [external_irq_ctrl_t](#)

Enumerations

enum [external_irq_trigger_t](#)

enum [external_irq_pclk_div_t](#)

Data Structure Documentation

◆ external_irq_callback_args_t

struct external_irq_callback_args_t		
Callback function parameter data		
Data Fields		
void const *	p_context	Placeholder for user data. Set in external_irq_api_t::open function in external_irq_cfg_t .
uint32_t	channel	The physical hardware channel that caused the interrupt.

◆ external_irq_cfg_t

struct external_irq_cfg_t
User configuration structure, used in open function
Data Fields

uint8_t	channel
	Hardware channel used.
uint8_t	ipl
	Interrupt priority.
IRQn_Type	irq
	NVIC interrupt number assigned to this instance.
external_irq_trigger_t	trigger
	Trigger setting.
external_irq_pclk_div_t	pclk_div
	Digital filter clock divisor setting.
bool	filter_enable
	Digital filter enable/disable setting.
void(*	p_callback)(external_irq_callback_args_t *p_args)
void const *	p_context
void const *	p_extend
	External IRQ hardware dependent configuration.
Field Documentation	


```
void(* external_irq_cfg_t::p_callback) (external_irq_callback_args_t *p_args)
```

◆ **p_context**

```
void const* external_irq_cfg_t::p_context
```

◆ external_irq_api_t

```
struct external_irq_api_t
```

Data Fields	

```
fsp_err_t( *open )(external_irq_ctrl_t *const p_ctrl, external_irq_cfg_t const
*const p_cfg)
```


[illegible]

◆ open

```
fsp_err_t(* external_irq_api_t::open) (external_irq_ctrl_t *const p_ctrl, external_irq_cfg_t const *const p_cfg)
```

Initial configuration.

Implemented as

◦ R_ICU_ExternallrqOpen()

Parameters

[out]	p_ctrl	Pointer to control block. Must be declared by user. Value set here.
[in]	p_cfg	Pointer to configuration structure. All elements of the structure must be set by user.

◆ enable

```
fsp_err_t(* external_irq_api_t::enable) (external_irq_ctrl_t *const p_ctrl)
```

Enable callback when an external trigger condition occurs.

Implemented as

◦ R_ICU_ExternallrqEnable()

Parameters

[in]	p_ctrl	Control block set in Open call for this external interrupt.

◆ disable

```
fsp_err_t(* external_irq_api_t::disable) (external_irq_ctrl_t *const p_ctrl)
```

Disable callback when external trigger condition occurs.

Implemented as

◦ R_ICU_ExternallrqDisable()

Parameters

[in]	p_ctrl	Control block set in Open call for this external interrupt.

◆ **close**

```
fsp_err_t(* external_irq_api_t::close) (external_irq_ctrl_t *const p_ctrl)
```

Allow driver to be reconfigured. May reduce power consumption.

Implemented as

◦ [R_ICU_ExtIrqClose\(\)](#)

Parameters

[in]	p_ctrl	Control block set in Open call for this external interrupt.

◆ **versionGet**

```
fsp_err_t(* external_irq_api_t::versionGet) (fsp_version_t *const p_version)
```

Get version and store it in provided pointer p_version.

Implemented as

◦ [R_ICU_ExtIrqVersionGet\(\)](#)

Parameters

[out]	p_version	Code and API version used.

◆ **external_irq_instance_t**

```
struct external_irq_instance_t
```

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields

external_irq_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
external_irq_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
external_irq_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ **external_irq_ctrl_t**typedef void [external_irq_ctrl_t](#)

External IRQ control block. Allocate an instance specific control block to pass into the external IRQ API calls.

Implemented as

- [icu_instance_ctrl_t](#)

Enumeration Type Documentation◆ **external_irq_trigger_t**enum [external_irq_trigger_t](#)

Condition that will trigger an interrupt when detected.

Enumerator

EXTERNAL_IRQ_TRIG_FALLING	Falling edge trigger.
EXTERNAL_IRQ_TRIG_RISING	Rising edge trigger.
EXTERNAL_IRQ_TRIG_BOTH_EDGE	Both edges trigger.
EXTERNAL_IRQ_TRIG_LEVEL_LOW	Low level trigger.

◆ **external_irq_pclk_div_t**enum [external_irq_pclk_div_t](#)

External IRQ input pin digital filtering sample clock divisor settings. The digital filter rejects trigger conditions that are shorter than 3 periods of the filter clock.

Enumerator

EXTERNAL_IRQ_PCLK_DIV_BY_1	Filter using PCLK divided by 1.
EXTERNAL_IRQ_PCLK_DIV_BY_8	Filter using PCLK divided by 8.
EXTERNAL_IRQ_PCLK_DIV_BY_32	Filter using PCLK divided by 32.
EXTERNAL_IRQ_PCLK_DIV_BY_64	Filter using PCLK divided by 64.

4.3.14 Flash Interface

Interfaces

Detailed Description

Interface for the Flash Memory.

Summary

The Flash interface provides the ability to read, write, erase, and blank check the code flash and data flash regions.

The Flash interface is implemented by:

- [Low-Power Flash Driver \(r_flash_lp\)](#)

Data Structures

struct [flash_block_info_t](#)

struct [flash_regions_t](#)

struct [flash_info_t](#)

struct [flash_callback_args_t](#)

struct [flash_cfg_t](#)

struct [flash_api_t](#)

struct [flash_instance_t](#)

Typedefs

typedef void [flash_ctrl_t](#)

Enumerations

enum [flash_result_t](#)

enum [flash_startup_area_swap_t](#)

enum [flash_event_t](#)

enum [flash_id_code_mode_t](#)

enum [flash_status_t](#)

Data Structure Documentation

◆ [flash_block_info_t](#)

struct flash_block_info_t		
Flash block details stored in factory flash.		
Data Fields		
uint32_t	block_section_st_addr	Starting address for this block section (blocks of this size)
uint32_t	block_section_end_addr	Ending address for this block section (blocks of this size)
uint32_t	block_size	Flash erase block size.
uint32_t	block_size_write	Flash write block size.

◆ **flash_regions_t**

struct flash_regions_t		
Flash block details		
Data Fields		
uint32_t	num_regions	Length of block info array.
flash_block_info_t const *	p_block_array	Block info array base address.

◆ **flash_info_t**

struct flash_info_t		
Information about the flash blocks		
Data Fields		
flash_regions_t	code_flash	Information about the code flash regions.
flash_regions_t	data_flash	Information about the code flash regions.

◆ **flash_callback_args_t**

struct flash_callback_args_t		
Callback function parameter data		
Data Fields		
flash_event_t	event	Event can be used to identify what caused the callback (flash ready or error).
void const *	p_context	Placeholder for user data. Set in flash_api_t::open function in::flash_cfg_t.

◆ **flash_cfg_t**

struct flash_cfg_t		
FLASH Configuration		

Data Fields	
bool	data_flash_bgo
	True if BGO (Background Operation) is enabled for Data Flash.
void(*	p_callback)(flash_callback_args_t *p_args)
	Callback provided when a Flash interrupt ISR occurs.
void const *	p_extend
	FLASH hardware dependent configuration.
void const *	p_context
	Placeholder for user data. Passed to user callback in flash_callback_args_t .
uint8_t	ipl
	Flash ready interrupt priority.
IRQn_Type	irq
	Flash ready interrupt number.
uint8_t	err_ipl
	Flash error interrupt priority (unused in r_flash_lp)
IRQn_Type	err_irq
	Flash error interrupt number (unused in r_flash_lp)

◆ flash_api_t

struct flash_api_t
Shared Interface definition for FLASH

Data Fields	
<code>fsp_err_t(*</code>	<code>open)(flash_ctrl_t *const p_ctrl, flash_cfg_t const *const p_cfg)</code>
<code>fsp_err_t(*</code>	<code>write)(flash_ctrl_t *const p_ctrl, uint32_t const src_address, uint32_t const flash_address, uint32_t const num_bytes)</code>
<code>fsp_err_t(*</code>	<code>erase)(flash_ctrl_t *const p_ctrl, uint32_t const address, uint32_t const num_blocks)</code>
<code>fsp_err_t(*</code>	<code>blankCheck)(flash_ctrl_t *const p_ctrl, uint32_t const address, uint32_t const num_bytes, flash_result_t *const p_blank_check_result)</code>
<code>fsp_err_t(*</code>	<code>infoGet)(flash_ctrl_t *const p_ctrl, flash_info_t *const p_info)</code>
<code>fsp_err_t(*</code>	<code>close)(flash_ctrl_t *const p_ctrl)</code>
<code>fsp_err_t(*</code>	<code>statusGet)(flash_ctrl_t *const p_ctrl, flash_status_t *const p_status)</code>
<code>fsp_err_t(*</code>	<code>accessWindowSet)(flash_ctrl_t *const p_ctrl, uint32_t const start_addr, uint32_t const end_addr)</code>
<code>fsp_err_t(*</code>	<code>accessWindowClear)(flash_ctrl_t *const p_ctrl)</code>
<code>fsp_err_t(*</code>	<code>idCodeSet)(flash_ctrl_t *const p_ctrl, uint8_t const *const p_id_bytes, flash_id_code_mode_t mode)</code>
<code>fsp_err_t(*</code>	<code>reset)(flash_ctrl_t *const p_ctrl)</code>
<code>fsp_err_t(*</code>	<code>updateFlashClockFreq)(flash_ctrl_t *const p_ctrl)</code>
<code>fsp_err_t(*</code>	<code>startupAreaSelect)(flash_ctrl_t *const p_ctrl, flash_startup_area_swap_t swap_type, bool is_temporary)</code>
<code>fsp_err_t(*</code>	<code>versionGet)(fsp_version_t *p_version)</code>

Field Documentation

◆ open

```
fsp_err_t(* flash_api_t::open) (flash_ctrl_t *const p_ctrl, flash_cfg_t const *const p_cfg)
```

Open FLASH device.

Implemented as

- R_FLASH_LP_Open()
- R_FLASH_HP_Open()

Parameters

[out]	p_ctrl	Pointer to FLASH device control. Must be declared by user. Value set here.
[in]	flash_cfg_t	Pointer to FLASH configuration structure. All elements of this structure must be set by the user.

◆ **write**

```
fsp_err_t(* flash_api_t::write) (flash_ctrl_t *const p_ctrl, uint32_t const src_address, uint32_t const flash_address, uint32_t const num_bytes)
```

Write FLASH device.

Implemented as

- R_FLASH_LP_Write()
- R_FLASH_HP_Write()

Parameters

[in]	p_ctrl	Control for the FLASH device context.
[in]	src_address	Address of the buffer containing the data to write to Flash.
[in]	flash_address	Code Flash or Data Flash address to write. The address must be on a programming line boundary.
[in]	num_bytes	The number of bytes to write. This number must be a multiple of the programming size. For Code Flash this is FLASH_MIN_PGM_SIZE_CF. For Data Flash this is FLASH_MIN_PGM_SIZE_DF.

Warning

Specifying a number that is not a multiple of the programming size will result in SF_FLASH_ERR_BYTES being returned and no data written.

◆ erase

`fsp_err_t(* flash_api_t::erase) (flash_ctrl_t *const p_ctrl, uint32_t const address, uint32_t const num_blocks)`

Erase FLASH device.

Implemented as

`R_FLASH_LP_Erase()` `R_FLASH_HP_Erase()`

Parameters

[in]	p_ctrl	Control for the FLASH device.
[in]	address	The block containing this address is the first block erased.
[in]	num_blocks	Specifies the number of blocks to be erased, the starting block determined by the block_erase_address.

◆ **blankCheck**

```
fsp_err_t(* flash_api_t::blankCheck) (flash_ctrl_t *const p_ctrl, uint32_t const address, uint32_t const num_bytes, flash_result_t *const p_blank_check_result)
```

Blank check FLASH device.

Implemented as

- R_FLASH_LP_BlankCheck()
- R_FLASH_HP_BlankCheck()

Parameters

[in]	p_ctrl	Control for the FLASH device context.
[in]	address	The starting address of the Flash area to blank check.
[in]	num_bytes	Specifies the number of bytes that need to be checked. See the specific handler for details.
[out]	p_blank_check_result	Pointer that will be populated by the API with the results of the blank check operation in non-BGO (blocking) mode. In this case the blank check operation completes here and the result is returned. In Data Flash BGO mode the blank check operation is only started here and the result obtained later when the supplied callback routine is called. In this case FLASH_RESULT_BGO_ACTIVE will be returned in p_blank_check_result.

◆ infoGet

```
fsp_err_t(* flash_api_t::infoGet) (flash_ctrl_t *const p_ctrl, flash_info_t *const p_info)
```

Close FLASH device.

Implemented as

- R_FLASH_LP_InfoGet()
- R_FLASH_HP_InfoGet()

Parameters

[in]	p_ctrl	Pointer to FLASH device control.
[out]	p_info	Pointer to FLASH info structure.

◆ close

```
fsp_err_t(* flash_api_t::close) (flash_ctrl_t *const p_ctrl)
```

Close FLASH device.

Implemented as

- R_FLASH_LP_Close()
- R_FLASH_HP_Close()

Parameters

[in]	p_ctrl	Pointer to FLASH device control.

◆ statusGet

```
fsp_err_t(* flash_api_t::statusGet) (flash_ctrl_t *const p_ctrl, flash_status_t *const p_status)
```

Get Status for FLASH device.

Implemented as

- R_FLASH_LP_StatusGet()
- R_FLASH_HP_StatusGet()

Parameters

[in]	p_ctrl	Pointer to FLASH device control.
[out]	p_status	Pointer to the current flash status.

◆ **accessWindowSet**

```
fsp_err_t(* flash_api_t::accessWindowSet) (flash_ctrl_t *const p_ctrl, uint32_t const start_addr,
uint32_t const end_addr)
```

Set Access Window for FLASH device.

Implemented as

- R_FLASH_LP_AccessWindowSet()
- R_FLASH_HP_AccessWindowSet()

Parameters

[in]	p_ctrl	Pointer to FLASH device control.
[in]	start_addr	Determines the Starting block for the Code Flash access window.
[in]	end_addr	Determines the Ending block for the Code Flash access window. This address will not be within the access window.

◆ **accessWindowClear**

```
fsp_err_t(* flash_api_t::accessWindowClear) (flash_ctrl_t *const p_ctrl)
```

Clear any existing Code Flash access window for FLASH device.

Implemented as

- R_FLASH_LP_AccessWindowClear()
- R_FLASH_HP_AccessWindowClear()

Parameters

[in]	p_ctrl	Pointer to FLASH device control.
[in]	start_addr	Determines the Starting block for the Code Flash access window.
[in]	end_addr	Determines the Ending block for the Code Flash access window.

◆ **idCodeSet**

```
fsp_err_t(* flash_api_t::idCodeSet) (flash_ctrl_t *const p_ctrl, uint8_t const *const p_id_bytes,
flash_id_code_mode_t mode)
```

Set ID Code for FLASH device. Setting the ID code can restrict access to the device. The ID code will be required to connect to the device. Bits 126 and 127 are set based on the mode.

For example, uint8_t id_bytes[] = {0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xdd, 0xee, 0x00}; with mode FLASH_ID_CODE_MODE_LOCKED_WITH_ALL_ERASE_SUPPORT will result in an ID code of 00112233445566778899aabbccddeec0

With mode FLASH_ID_CODE_MODE_LOCKED, it will result in an ID code of 00112233445566778899aabbccddeec80

Implemented as

- R_FLASH_LP_IdCodeSet()
- R_FLASH_HP_IdCodeSet()

Parameters

[in]	p_ctrl	Pointer to FLASH device control.
[in]	p_id_bytes	Ponter to the ID Code to be written.
[in]	mode	Mode used for checking the ID code.

◆ **reset**

```
fsp_err_t(* flash_api_t::reset) (flash_ctrl_t *const p_ctrl)
```

Reset function for FLASH device.

Implemented as

- R_FLASH_LP_Reset()
- R_FLASH_HP_Reset()

Parameters

[in]	p_ctrl	Pointer to FLASH device control.

◆ **updateFlashClockFreq**

```
fsp_err_t(* flash_api_t::updateFlashClockFreq) (flash_ctrl_t *const p_ctrl)
```

Update Flash clock frequency (FCLK) and recalculate timeout values

Implemented as

- R_FLASH_LP_UpdateFlashClockFreq()
- R_FLASH_HP_UpdateFlashClockFreq()

Parameters

[in]	p_ctrl	Pointer to FLASH device control.

◆ **startupAreaSelect**

```
fsp_err_t(* flash_api_t::startupAreaSelect) (flash_ctrl_t *const p_ctrl, flash_startup_area_swap_t swap_type, bool is_temporary)
```

Select which block - Default (Block 0) or Alternate (Block 1) is used as the start-up area block.

Implemented as

- R_FLASH_LP_StartUpAreaSelect()
- R_FLASH_HP_StartUpAreaSelect()

Parameters

[in]	p_ctrl	Pointer to FLASH device control.
[in]	swap_type	FLASH_STARTUP_AREA_BLOCK0, FLASH_STARTUP_AREA_BLOCK1 or FLASH_STARTUP_AREA_BTFLG.
[in]	is_temporary	True or false. See table below.

swap_type	is_temporary	Operation
FLASH_STARTUP_AREA_BLOCK0	false	On next reset Startup area will be Block 0.
FLASH_STARTUP_AREA_BLOCK1	true	Startup area is immediately, but temporarily switched to Block 1.
FLASH_STARTUP_AREA_BTFLG	true	Startup area is immediately, but temporarily switched to the Block determined by the Configuration BTFLG.

◆ **versionGet**

```
fsp_err_t(* flash_api_t::versionGet) (fsp_version_t *p_version)
```

Get Flash driver version.

Implemented as

- R_FLASH_LP_VersionGet()
- R_FLASH_HP_VersionGet()

Parameters

[out]	p_version	Returns version.

◆ **flash_instance_t**

```
struct flash_instance_t
```

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields

flash_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
flash_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
flash_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation◆ **flash_ctrl_t**

```
typedef void flash_ctrl_t
```

Flash control block. Allocate an instance specific control block to pass into the flash API calls.

Implemented as

- flash_lp_instance_ctrl_t
- flash_hp_instance_ctrl_t

Enumeration Type Documentation

◆ **flash_result_t**

enum flash_result_t	
Result type for certain operations	
Enumerator	
FLASH_RESULT_BLANK	Return status for Blank Check Function.
FLASH_RESULT_NOT_BLANK	Return status for Blank Check Function.
FLASH_RESULT_BGO_ACTIVE	Flash is configured for BGO mode. Result is returned in callback.

◆ **flash_startup_area_swap_t**

enum flash_startup_area_swap_t	
Parameter for specifying the startup area swap being requested by startupAreaSelect()	
Enumerator	
FLASH_STARTUP_AREA_BTFLG	Startup area will be set based on the value of the BTFLG.
FLASH_STARTUP_AREA_BLOCK0	Startup area will be set to Block 0.
FLASH_STARTUP_AREA_BLOCK1	Startup area will be set to Block 1.

◆ **flash_event_t**

enum <code>flash_event_t</code>	
Event types returned by the ISR callback when used in Data Flash BGO mode	
Enumerator	
<code>FLASH_EVENT_ERASE_COMPLETE</code>	Erase operation successfully completed.
<code>FLASH_EVENT_WRITE_COMPLETE</code>	Write operation successfully completed.
<code>FLASH_EVENT_BLANK</code>	Blank check operation successfully completed. Specified area is blank.
<code>FLASH_EVENT_NOT_BLANK</code>	Blank check operation successfully completed. Specified area is NOT blank.
<code>FLASH_EVENT_ERR_DF_ACCESS</code>	Data Flash operation failed. Can occur when writing an unerased section.
<code>FLASH_EVENT_ERR_CF_ACCESS</code>	Code Flash operation failed. Can occur when writing an unerased section.
<code>FLASH_EVENT_ERR_CMD_LOCKED</code>	Operation failed, FCU is in Locked state (often result of an illegal command)
<code>FLASH_EVENT_ERR_FAILURE</code>	Erase or Program Operation failed.
<code>FLASH_EVENT_ERR_ONE_BIT</code>	A 1-bit error has been corrected when reading the flash memory area by the sequencer.

◆ **flash_id_code_mode_t**

enum <code>flash_id_code_mode_t</code>	
ID Code Modes for writing to ID code registers	
Enumerator	
<code>FLASH_ID_CODE_MODE_UNLOCKED</code>	ID code is ignored.
<code>FLASH_ID_CODE_MODE_LOCKED_WITH_ALL_ERASE_SUPPORT</code>	ID code is checked. All erase is available.
<code>FLASH_ID_CODE_MODE_LOCKED</code>	ID code is checked.

◆ **flash_status_t**

enum <code>flash_status_t</code>	
Flash status	
Enumerator	
<code>FLASH_STATUS_IDLE</code>	The flash is idle.
<code>FLASH_STATUS_BUSY</code>	The flash is currently processing a command.

4.3.15 I2C Master Interface

Interfaces

Detailed Description

Interface for I2C master communication.

Summary

The I2C master interface provides a common API for I2C HAL drivers. The I2C master interface supports:

- Interrupt driven transmit/receive processing
- Callback function support which can return an event code

Implemented by:

- [I2C Master on IIC \(r_iic_master\)](#)

Data Structures

struct [i2c_master_callback_args_t](#)

struct [i2c_master_cfg_t](#)

struct [i2c_master_api_t](#)

struct [i2c_master_instance_t](#)

Typedefs

typedef void [i2c_master_ctrl_t](#)

Enumerations

enum [i2c_master_rate_t](#)enum [i2c_master_addr_mode_t](#)enum [i2c_master_event_t](#)

Data Structure Documentation

◆ i2c_master_callback_args_t

struct i2c_master_callback_args_t		
I2C callback parameter definition		
Data Fields		
void const *const	p_context	Pointer to user-provided context.
i2c_master_event_t const	event	Event code.

◆ i2c_master_cfg_t

struct i2c_master_cfg_t	
I2C configuration block	
Data Fields	
uint8_t	channel
	Identifier recognizable by implementation. More...
i2c_master_rate_t	rate
	Device's maximum clock rate from enum i2c_rate_t .
uint32_t	slave
	The address of the slave device.
i2c_master_addr_mode_t	addr_mode
	Indicates how slave fields should be interpreted.
uint8_t	ipl
	Interrupt priority level. Same for RXI, TXI, TEI and ERI.

IRQn_Type	rx_irq
	Receive IRQ number.
IRQn_Type	tx_irq
	Transmit IRQ number.
IRQn_Type	tei_irq
	Transmit end IRQ number.
IRQn_Type	eri_irq
	Error IRQ number.
transfer_instance_t const *	p_transfer_tx
	DTC instance for I2C transmit. Set to NULL if unused. More...
transfer_instance_t const *	p_transfer_rx
	DTC instance for I2C receive. Set to NULL if unused.
void(*)	p_callback)(i2c_master_callback_args_t *p_args)
	Pointer to callback function. More...
void const *	p_context
	Pointer to the user-provided context.
void const *	p_extend
	Any configuration data needed by the hardware. More...

Field Documentation

◆ channel

uint8_t i2c_master_cfg_t::channel

Identifier recognizable by implementation.

Generic configuration

◆ p_transfer_tx

transfer_instance_t const* i2c_master_cfg_t::p_transfer_tx

DTC instance for I2C transmit. Set to NULL if unused.

DTC support

◆ p_callback

void(* i2c_master_cfg_t::p_callback) (i2c_master_callback_args_t *p_args)

Pointer to callback function.

Parameters to control software behavior

◆ p_extend

void const* i2c_master_cfg_t::p_extend

Any configuration data needed by the hardware.

Implementation-specific configuration

◆ i2c_master_api_t

struct i2c_master_api_t

Interface definition for I2C access as master

Data Fields

fsp_err_t(*	open)(i2c_master_ctrl_t *const p_ctrl, i2c_master_cfg_t const *const p_cfg)

fsp_err_t(*	read)(i2c_master_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const bytes, bool const restart)

fsp_err_t(*	write)(i2c_master_ctrl_t *const p_ctrl, uint8_t *const p_src, uint32_t const bytes, bool const restart)

fsp_err_t(*	abort)(i2c_master_ctrl_t *const p_ctrl)

fsp_err_t(*	slaveAddressSet)(i2c_master_ctrl_t *const p_ctrl, uint32_t const

	slave, i2c_master_addr_mode_t const addr_mode)

fsp_err_t(*	close)(i2c_master_ctrl_t *const p_ctrl)

fsp_err_t(*	versionGet)(fsp_version_t *const p_version)

Field Documentation

◆ open

fsp_err_t(* i2c_master_api_t::open) (i2c_master_ctrl_t *const p_ctrl, i2c_master_cfg_t const *const p_cfg)

Opens the I2C Master driver and initializes the hardware.

Implemented as

◦ R_IIC_MASTER_Open()

Parameters

[in]	p_ctrl	Pointer to control block. Must be declared by user. Elements are set here.
[in]	p_cfg	Pointer to configuration structure.

◆ read

fsp_err_t(* i2c_master_api_t::read) (i2c_master_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const bytes, bool const restart)

Performs a read operation on an I2C Master device.

Implemented as

◦ R_IIC_MASTER_Read()

Parameters

[in]	p_ctrl	Pointer to control block set in i2c_api_master_t::open call.
[in]	p_dest	Pointer to the location to store read data.
[in]	bytes	Number of bytes to read.
[in]	restart	Specify if the restart condition should be issued after reading.

◆ **write**

```
fsp_err_t(* i2c_master_api_t::write) (i2c_master_ctrl_t *const p_ctrl, uint8_t *const p_src, uint32_t const bytes, bool const restart)
```

Performs a write operation on an I2C Master device.

Implemented as

◦ [R_IIC_MASTER_Write\(\)](#)

Parameters

[in]	p_ctrl	Pointer to control block set in i2c_api_master_t::open call.
[in]	p_src	Pointer to the location to get write data from.
[in]	bytes	Number of bytes to write.
[in]	restart	Specify if the restart condition should be issued after writing.

◆ **abort**

```
fsp_err_t(* i2c_master_api_t::abort) (i2c_master_ctrl_t *const p_ctrl)
```

Performs a reset of the peripheral.

Implemented as

◦ [R_IIC_MASTER_Abort\(\)](#)

Parameters

[in]	p_ctrl	Pointer to control block set in i2c_api_master_t::open call.

◆ **slaveAddressSet**

```
fsp_err_t(* i2c_master_api_t::slaveAddressSet) (i2c_master_ctrl_t *const p_ctrl, uint32_t const slave,
i2c_master_addr_mode_t const addr_mode)
```

Sets address of the slave device without reconfiguring the bus.

Implemented as

◦ [R_IIC_MASTER_SlaveAddressSet\(\)](#)

Parameters

[in]	p_ctrl	Pointer to control block set in i2c_api_master_t::open call.
[in]	slave_address	Address of the slave device.
[in]	address_mode	Addressing mode.

◆ **close**

```
fsp_err_t(* i2c_master_api_t::close) (i2c_master_ctrl_t *const p_ctrl)
```

Closes the driver and releases the I2C Master device.

Implemented as

◦ [R_IIC_MASTER_Close\(\)](#)

Parameters

[in]	p_ctrl	Pointer to control block set in i2c_api_master_t::open call.

◆ **versionGet**

```
fsp_err_t(* i2c_master_api_t::versionGet) (fsp_version_t *const p_version)
```

Gets version information and stores it in the provided version struct.

Implemented as

◦ [R_IIC_MASTER_VersionGet\(\)](#)

Parameters

[out]	p_version	Code and API version used.

◆ **i2c_master_instance_t**

```
struct i2c_master_instance_t
```

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields

<code>i2c_master_ctrl_t *</code>	<code>p_ctrl</code>	Pointer to the control structure for this instance.
<code>i2c_master_cfg_t const *</code>	<code>p_cfg</code>	Pointer to the configuration structure for this instance.
<code>i2c_master_api_t const *</code>	<code>p_api</code>	Pointer to the API structure for this instance.

Typedef Documentation

◆ `i2c_master_ctrl_t`

```
typedef void i2c_master_ctrl_t
```

I2C control block. Allocate an instance specific control block to pass into the I2C API calls.

Implemented as

- `iic_master_instance_ctrl_t`

Enumeration Type Documentation

◆ `i2c_master_rate_t`

```
enum i2c_master_rate_t
```

Communication speed options

Enumerator

<code>I2C_MASTER_RATE_STANDARD</code>	100 kHz
<code>I2C_MASTER_RATE_FAST</code>	400 kHz
<code>I2C_MASTER_RATE_FASTPLUS</code>	1 MHz

◆ `i2c_master_addr_mode_t`

```
enum i2c_master_addr_mode_t
```

Addressing mode options

Enumerator

<code>I2C_MASTER_ADDR_MODE_7BIT</code>	Use 7-bit addressing mode.
<code>I2C_MASTER_ADDR_MODE_10BIT</code>	Use 10-bit addressing mode.

◆ **i2c_master_event_t**

enum i2c_master_event_t	
Callback events	
Enumerator	
I2C_MASTER_EVENT_ABORTED	A transfer was aborted.
I2C_MASTER_EVENT_RX_COMPLETE	A receive operation was completed successfully.
I2C_MASTER_EVENT_TX_COMPLETE	A transmit operation was completed successfully.

4.3.16 I2C Slave Interface[Interfaces](#)**Detailed Description**

Interface for I2C slave communication.

Summary

The I2C slave interface provides a common API for I2C HAL drivers. The I2C slave interface supports:

- Interrupt driven transmit/receive processing
- Callback function support which returns a event codes

Implemented by:

- [I2C Slave on IIC \(r_iic_slave\)](#)

Data Structures

struct [i2c_slave_callback_args_t](#)

struct [i2c_slave_cfg_t](#)

struct [i2c_slave_api_t](#)

struct [i2c_slave_instance_t](#)

Typedefs

typedef void [i2c_slave_ctrl_t](#)

Enumerations

enum [i2c_slave_rate_t](#)

enum [i2c_slave_addr_mode_t](#)

enum [i2c_slave_event_t](#)

Data Structure Documentation

◆ [i2c_slave_callback_args_t](#)

struct i2c_slave_callback_args_t		
I2C callback parameter definition		
Data Fields		
void const *const	p_context	Pointer to user-provided context.
uint32_t const	bytes	Number of received/transmitted bytes in buffer.
i2c_slave_event_t const	event	Event code.

◆ [i2c_slave_cfg_t](#)

struct i2c_slave_cfg_t	
I2C configuration block	
Data Fields	
uint8_t	channel
	Identifier recognizable by implementation. More...
i2c_slave_rate_t	rate
	Device's maximum clock rate from enum i2c_rate_t .
uint16_t	slave
	The address of the slave device.
i2c_slave_addr_mode_t	addr_mode
	Indicates how slave fields should be interpreted.

IRQn_Type	rx_irq
	Receive IRQ number.
IRQn_Type	tx_irq
	Transmit IRQ number.
IRQn_Type	tei_irq
	Transmit end IRQ number.
IRQn_Type	eri_irq
	Error IRQ number.
uint8_t	ipl
	Interrupt priority level.
void(*)	p_callback)(i2c_slave_callback_args_t *p_args)
	Pointer to callback function. More...
void const *	p_context
	Pointer to the user-provided context.
void const *	p_extend
	Any configuration data needed by the hardware. More...

Field Documentation

◆ **channel**

```
uint8_t i2c_slave_cfg_t::channel
```

Identifier recognizable by implementation.

Generic configuration

◆ **p_callback**

```
void(* i2c_slave_cfg_t::p_callback) (i2c_slave_callback_args_t *p_args)
```

Pointer to callback function.

Parameters to control software behavior

◆ **p_extend**

```
void const* i2c_slave_cfg_t::p_extend
```

Any configuration data needed by the hardware.

Implementation-specific configuration

◆ **i2c_slave_api_t**

```
struct i2c_slave_api_t
```

Interface definition for I2C access as slave

Data Fields

<code>fsp_err_t(*</code>	<code>open)(i2c_slave_ctrl_t *const p_ctrl, i2c_slave_cfg_t const *const p_cfg)</code>

<code>fsp_err_t(*</code>	<code>read)(i2c_slave_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const bytes)</code>

<code>fsp_err_t(*</code>	<code>write)(i2c_slave_ctrl_t *const p_ctrl, uint8_t *const p_src, uint32_t const bytes)</code>

<code>fsp_err_t(*</code>	<code>close)(i2c_slave_ctrl_t *const p_ctrl)</code>

<code>fsp_err_t(*</code>	<code>versionGet)(fsp_version_t *const p_version)</code>

Field Documentation

◆ open

```
fsp_err_t(* i2c_slave_api_t::open) (i2c_slave_ctrl_t *const p_ctrl, i2c_slave_cfg_t const *const p_cfg)
```

Opens the I2C Slave driver and initializes the hardware.

Implemented as

- R_IIC_SLAVE_Open()

Parameters

[in]	p_ctrl	Pointer to control block. Must be declared by user. Elements are set here.
[in]	p_cfg	Pointer to configuration structure.

◆ read

```
fsp_err_t(* i2c_slave_api_t::read) (i2c_slave_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const bytes)
```

Performs a read operation on an I2C Slave device.

Implemented as

- R_IIC_SLAVE_Read()

Parameters

[in]	p_ctrl	Pointer to control block set in <code>i2c_slave_api_t::open</code> call.
[in]	p_dest	Pointer to the location to store read data.
[in]	bytes	Number of bytes to read.

◆ **write**

```
fsp_err_t(* i2c_slave_api_t::write) (i2c_slave_ctrl_t *const p_ctrl, uint8_t *const p_src, uint32_t const bytes)
```

Performs a write operation on an I2C Slave device.

Implemented as

◦ [R_IIC_SLAVE_Write\(\)](#)

Parameters

[in]	p_ctrl	Pointer to control block set in i2c_slave_api_t::open call.
[in]	p_src	Pointer to the location to get write data from.
[in]	bytes	Number of bytes to write.

◆ **close**

```
fsp_err_t(* i2c_slave_api_t::close) (i2c_slave_ctrl_t *const p_ctrl)
```

Closes the driver and releases the I2C Slave device.

Implemented as

◦ [R_IIC_SLAVE_Close\(\)](#)

Parameters

[in]	p_ctrl	Pointer to control block set in i2c_slave_api_t::open call.

◆ **versionGet**

```
fsp_err_t(* i2c_slave_api_t::versionGet) (fsp_version_t *const p_version)
```

Gets version information and stores it in the provided version struct.

Implemented as

◦ [R_IIC_SLAVE_VersionGet\(\)](#)

Parameters

[out]	p_version	Code and API version used.

◆ **i2c_slave_instance_t**

```
struct i2c_slave_instance_t
```

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields

i2c_slave_ctrl_t *	p_ctrl	Pointer to the control structure

		for this instance.
<code>i2c_slave_cfg_t</code> const *	<code>p_cfg</code>	Pointer to the configuration structure for this instance.
<code>i2c_slave_api_t</code> const *	<code>p_api</code>	Pointer to the API structure for this instance.

Typedef Documentation

◆ `i2c_slave_ctrl_t`

<code>typedef void i2c_slave_ctrl_t</code>
I2C control block. Allocate an instance specific control block to pass into the I2C API calls.
Implemented as
◦ <code>iic_slave_instance_ctrl_t</code>

Enumeration Type Documentation

◆ `i2c_slave_rate_t`

enum i2c_slave_rate_t	
Communication speed options	
Enumerator	
I2C_SLAVE_RATE_STANDARD	100 kHz
I2C_SLAVE_RATE_FAST	400 kHz
I2C_SLAVE_RATE_FASTPLUS	1 MHz

◆ `i2c_slave_addr_mode_t`

enum i2c_slave_addr_mode_t	
Addressing mode options	
Enumerator	
I2C_SLAVE_ADDR_MODE_7BIT	Use 7-bit addressing mode.
I2C_SLAVE_ADDR_MODE_10BIT	Use 10-bit addressing mode.

◆ **i2c_slave_event_t**

enum i2c_slave_event_t	
Callback events	
Enumerator	
I2C_SLAVE_EVENT_ABORTED	A transfer was aborted.
I2C_SLAVE_EVENT_RX_COMPLETE	A receive operation was completed successfully.
I2C_SLAVE_EVENT_TX_COMPLETE	A transmit operation was completed successfully.
I2C_SLAVE_EVENT_RX_REQUEST	A read operation expected from slave. Detected a write from master.
I2C_SLAVE_EVENT_TX_REQUEST	A write operation expected from slave. Detected a read from master.
I2C_SLAVE_EVENT_RX_MORE_REQUEST	configured to be read in slave. A read operation expected from slave. Master sends out more data than
I2C_SLAVE_EVENT_TX_MORE_REQUEST	configured to be written by slave. A write operation expected from slave. Master requests more data than

4.3.17 I2S Interface[Interfaces](#)**Detailed Description**

Interface for I2S audio communication.

Summary

The I2S (Inter-IC Sound) interface provides APIs and definitions for I2S audio communication.

Known Implementations[Serial Sound Interface \(r_ssi\)](#)

Data Structures

struct	i2s_callback_args_t
struct	i2s_status_t
struct	i2s_cfg_t
struct	i2s_api_t
struct	i2s_instance_t

Macros

#define	I2S_API_VERSION_MAJOR

Typedefs

typedef void	i2s_ctrl_t

Enumerations

enum	i2s_pcm_width_t
enum	i2s_word_length_t
enum	i2s_event_t
enum	i2s_mode_t
enum	i2s_mute_t
enum	i2s_ws_continue_t
enum	i2s_state_t

Data Structure Documentation

◆ [i2s_callback_args_t](#)

struct i2s_callback_args_t		
Callback function parameter data		
Data Fields		
void const *	p_context	Placeholder for user data. Set in i2s_api_t::open function in i2s_cfg_t .
i2s_event_t	event	The event can be used to identify what caused the callback (overflow or error).

◆ **i2s_status_t**

struct i2s_status_t
I2S status.

◆ **i2s_cfg_t**

struct i2s_cfg_t	
User configuration structure, used in open function	
Data Fields	
uint32_t	channel
i2s_pcm_width_t	pcm_width
	Audio PCM data width.
i2s_word_length_t	word_length
	Audio word length, bits must be $\geq$ i2s_cfg_t::pcm_width bits.
i2s_ws_continue_t	ws_continue
	Whether to continue WS transmission during idle state.
i2s_mode_t	operating_mode
	Master or slave mode.
transfer_instance_t const *	p_transfer_tx
transfer_instance_t const *	p_transfer_rx
void(*	p_callback)(i2s_callback_args_t *p_args)
void const *	p_context
void const *	p_extend
	Extension parameter for hardware specific settings.

uint8_t	rx_ipl
	Receive interrupt priority.
uint8_t	tx_ipl
	Transmit interrupt priority.
uint8_t	idle_err_ipl
	Idle/Error interrupt priority.
IRQn_Type	tx_irq
	Transmit IRQ number.
IRQn_Type	rx_irq
	Receive IRQ number.
IRQn_Type	int_irq
	Idle/Error IRQ number.

Field Documentation

◆ channel

uint32_t i2s_cfg_t::channel

Select a channel corresponding to the channel number of the hardware.

◆ p_transfer_tx

[transfer_instance_t](#) const* i2s_cfg_t::p_transfer_tx

To use DTC during write, link a DTC instance here. Set to NULL if unused.

◆ **p_transfer_rx**

<code>transfer_instance_t</code> const* <code>i2s_cfg_t::p_transfer_rx</code>

To use DTC during read, link a DTC instance here. Set to NULL if unused.

◆ **p_callback**

<code>void(* i2s_cfg_t::p_callback) (i2s_callback_args_t *p_args)</code>

Callback provided when an I2S ISR occurs. Set to NULL for no CPU interrupt.

◆ **p_context**

<code>void</code> const* <code>i2s_cfg_t::p_context</code>

Placeholder for user data. Passed to the user callback in `i2s_callback_args_t`.

◆ **i2s_api_t**

<code>struct i2s_api_t</code>

I2S functions implemented at the HAL layer will follow this API.

Data Fields

<code>fsp_err_t(*</code>	<code>open) (i2s_ctrl_t *const p_ctrl, i2s_cfg_t const *const p_cfg)</code>

<code>fsp_err_t(*</code>	<code>stop) (i2s_ctrl_t *const p_ctrl)</code>

<code>fsp_err_t(*</code>	<code>mute) (i2s_ctrl_t *const p_ctrl, i2s_mute_t const mute_enable)</code>

<code>fsp_err_t(*</code>	<code>write) (i2s_ctrl_t *const p_ctrl, void const *const p_src, uint32_t const bytes)</code>

<code>fsp_err_t(*</code>	<code>read) (i2s_ctrl_t *const p_ctrl, void *const p_dest, uint32_t const bytes)</code>

<code>fsp_err_t(*</code>	<code>writeRead) (i2s_ctrl_t *const p_ctrl, void const *const p_src, void *const p_dest, uint32_t const bytes)</code>

<code>fsp_err_t(*</code>	<code>statusGet) (i2s_ctrl_t *const p_ctrl, i2s_status_t *const p_status)</code>

<code>fsp_err_t(*</code>	<code>close) (i2s_ctrl_t *const p_ctrl)</code>

<code>fsp_err_t(*</code>	<code>versionGet) (fsp_version_t *const p_version)</code>

Field Documentation

◆ open

```
fsp_err_t(* i2s_api_t::open) (i2s_ctrl_t *const p_ctrl, i2s_cfg_t const *const p_cfg)
```

Initial configuration.

Implemented as

- [R_SSI_Open\(\)](#)

Precondition

Peripheral clocks and any required output pins should be configured prior to calling this function.

Note

To reconfigure after calling this function, call [i2s_api_t::close](#) first.

Parameters

[in]	p_ctrl	Pointer to control block. Must be declared by user. Elements set here.
[in]	p_cfg	Pointer to configuration structure. All elements of this structure must be set by user.

◆ stop

```
fsp_err_t(* i2s_api_t::stop) (i2s_ctrl_t *const p_ctrl)
```

Stop communication. Communication is stopped when callback is called with I2S_EVENT_IDLE.

Implemented as

- [R_SSI_Stop\(\)](#)

Parameters

[in]	p_ctrl	Control block set in i2s_api_t::open call for this instance.

◆ **mute**

```
fsp_err_t(* i2s_api_t::mute) (i2s_ctrl_t *const p_ctrl, i2s_mute_t const mute_enable)
```

Enable or disable mute.

Implemented as

◦ [R_SSI_Mute\(\)](#)

Parameters

[in]	p_ctrl	Control block set in i2s_api_t::open call for this instance.
[in]	mute_enable	Whether to enable or disable mute.

◆ **write**

```
fsp_err_t(* i2s_api_t::write) (i2s_ctrl_t *const p_ctrl, void const *const p_src, uint32_t const bytes)
```

Write I2S data. All transmit data is queued when callback is called with I2S_EVENT_TX_EMPTY. Transmission is complete when callback is called with I2S_EVENT_IDLE.

Implemented as

◦ [R_SSI_Write\(\)](#)

Parameters

[in]	p_ctrl	Control block set in i2s_api_t::open call for this instance.
[in]	p_src	Buffer of PCM samples. Must be 4 byte aligned.
[in]	bytes	Number of bytes in the buffer. Recommended requesting a multiple of 8 bytes. If not a multiple of 8, padding 0s will be added to transmission to make it a multiple of 8.

◆ read

`fsp_err_t(* i2s_api_t::read) (i2s_ctrl_t *const p_ctrl, void *const p_dest, uint32_t const bytes)`

Read I2S data. Reception is complete when callback is called with I2S_EVENT_RX_EMPTY.

Implemented as

- `R_SSI_Read()`

Parameters

[in]	p_ctrl	Control block set in i2s_api_t::open call for this instance.
[in]	p_dest	Buffer to store PCM samples. Must be 4 byte aligned.
[in]	bytes	Number of bytes in the buffer. Recommended requesting a multiple of 8 bytes. If not a multiple of 8, receive will stop at the multiple of 8 below requested bytes.

◆ **writeRead**

```
fsp_err_t(* i2s_api_t::writeRead) (i2s_ctrl_t *const p_ctrl, void const *const p_src, void *const p_dest, uint32_t const bytes)
```

Simultaneously write and read I2S data. Transmission and reception are complete when callback is called with I2S_EVENT_IDLE.

Implemented as

◦ [R_SSI_WriteRead\(\)](#)

Parameters

[in]	p_ctrl	Control block set in i2s_api_t::open call for this instance.
[in]	p_src	Buffer of PCM samples. Must be 4 byte aligned.
[in]	p_dest	Buffer to store PCM samples. Must be 4 byte aligned.
[in]	bytes	Number of bytes in the buffers. Recommended requesting a multiple of 8 bytes. If not a multiple of 8, padding 0s will be added to transmission to make it a multiple of 8, and receive will stop at the multiple of 8 below requested bytes.

◆ **statusGet**

```
fsp_err_t(* i2s_api_t::statusGet) (i2s_ctrl_t *const p_ctrl, i2s_status_t *const p_status)
```

Get current status and store it in provided pointer p_status.

Implemented as

◦ [R_SSI_StatusGet\(\)](#)

Parameters

[in]	p_ctrl	Control block set in i2s_api_t::open call for this instance.
[out]	p_status	Current status of the driver.

◆ **close**

```
fsp_err_t(* i2s_api_t::close) (i2s_ctrl_t *const p_ctrl)
```

Allows driver to be reconfigured and may reduce power consumption.

Implemented as

- [R_SSI_Close\(\)](#)

Parameters

[in]	p_ctrl	Control block set in i2s_api_t::open call for this instance.

◆ **versionGet**

```
fsp_err_t(* i2s_api_t::versionGet) (fsp_version_t *const p_version)
```

Get version and store it in provided pointer p_version.

Implemented as

- [R_SSI_VersionGet\(\)](#)

Parameters

[out]	p_version	Code and API version used.

◆ **i2s_instance_t**

```
struct i2s_instance_t
```

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields

i2s_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
i2s_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
i2s_api_t const *	p_api	Pointer to the API structure for this instance.

Macro Definition Documentation◆ **I2S_API_VERSION_MAJOR**

```
#define I2S_API_VERSION_MAJOR
```

Register definitions, common services and error codes.

Typedef Documentation

◆ **i2s_ctrl_t**typedef void **i2s_ctrl_t**

I2S control block. Allocate an instance specific control block to pass into the I2S API calls.

Implemented as

- [ssi_instance_ctrl_t](#)

Enumeration Type Documentation◆ **i2s_pcm_width_t**enum **i2s_pcm_width_t**

Audio PCM width

Enumerator

I2S_PCM_WIDTH_8_BITS	Using 8-bit PCM.
I2S_PCM_WIDTH_16_BITS	Using 16-bit PCM.
I2S_PCM_WIDTH_18_BITS	Using 18-bit PCM.
I2S_PCM_WIDTH_20_BITS	Using 20-bit PCM.
I2S_PCM_WIDTH_22_BITS	Using 22-bit PCM.
I2S_PCM_WIDTH_24_BITS	Using 24-bit PCM.
I2S_PCM_WIDTH_32_BITS	Using 24-bit PCM.

◆ **i2s_word_length_t**

enum i2s_word_length_t	
Audio system word length.	
Enumerator	
I2S_WORD_LENGTH_8_BITS	Using 8-bit system word length.
I2S_WORD_LENGTH_16_BITS	Using 16-bit system word length.
I2S_WORD_LENGTH_24_BITS	Using 24-bit system word length.
I2S_WORD_LENGTH_32_BITS	Using 32-bit system word length.
I2S_WORD_LENGTH_48_BITS	Using 48-bit system word length.
I2S_WORD_LENGTH_64_BITS	Using 64-bit system word length.
I2S_WORD_LENGTH_128_BITS	Using 128-bit system word length.
I2S_WORD_LENGTH_256_BITS	Using 256-bit system word length.

◆ **i2s_event_t**

enum i2s_event_t	
Events that can trigger a callback function	
Enumerator	
I2S_EVENT_IDLE	Communication is idle.
I2S_EVENT_TX_EMPTY	Transmit buffer is below FIFO trigger level.
I2S_EVENT_RX_FULL	Receive buffer is above FIFO trigger level.

◆ **i2s_mode_t**

enum i2s_mode_t	
I2S communication mode	
Enumerator	
I2S_MODE_SLAVE	Slave mode.
I2S_MODE_MASTER	Master mode.

◆ **i2s_mute_t**

enum i2s_mute_t	
Mute audio samples.	
Enumerator	
I2S_MUTE_OFF	Disable mute.
I2S_MUTE_ON	Enable mute.

◆ **i2s_ws_continue_t**

enum i2s_ws_continue_t	
Whether to continue WS (word select line) transmission during idle state.	
Enumerator	
I2S_WS_CONTINUE_ON	Enable WS continue mode.
I2S_WS_CONTINUE_OFF	Disable WS continue mode.

◆ **i2s_state_t**

enum i2s_state_t	
Possible status values returned by i2s_api_t::statusGet .	
Enumerator	
I2S_STATE_IN_USE	I2S is in use.
I2S_STATE_STOPPED	I2S is stopped.

4.3.18 I/O Port Interface

Interfaces

Detailed Description

Interface for accessing I/O ports and configuring I/O functionality.

Summary

The IOPort shared interface provides the ability to access the IOPorts of a device at both bit and port level. Port and pin direction can be changed.

IOPORT Interface description: [I/O Ports \(r_ioport\)](#)

Data Structures

struct [ioport_pin_cfg_t](#)

struct [ioport_cfg_t](#)

struct [ioport_api_t](#)

struct [ioport_instance_t](#)

Typedefs

typedef uint16_t [ioport_size_t](#)
IO port size on this device. [More...](#)

typedef void [ioport_ctrl_t](#)

Enumerations

enum [ioport_peripheral_t](#)

enum [ioport_ethernet_channel_t](#)

enum [ioport_ethernet_mode_t](#)

enum [ioport_cfg_options_t](#)

enum [ioport_pwpr_t](#)

Data Structure Documentation

◆ [ioport_pin_cfg_t](#)

struct ioport_pin_cfg_t		
Pin identifier and pin PFS pin configuration value		
Data Fields		
uint32_t	pin_cfg	Pin PFS configuration - Use ioport_cfg_options_t parameters to configure.
bsp_io_port_pin_t	pin	Pin identifier.

◆ ioport_cfg_t

struct ioport_cfg_t		
Multiple pin configuration data for loading into PFS registers by R_IOPORT_Init()		
Data Fields		
uint16_t	number_of_pins	Number of pins for which there is configuration data.
ioport_pin_cfg_t const *	p_pin_cfg_data	Pin configuration data.

◆ ioport_api_t

struct ioport_api_t	
IOPort driver structure. IOPort functions implemented at the HAL layer will follow this API.	
Data Fields	
fsp_err_t(*)	open)(ioport_ctrl_t *const p_ctrl, const ioport_cfg_t *p_cfg)
fsp_err_t(*)	close)(ioport_ctrl_t *const p_ctrl)
fsp_err_t(*)	pinsCfg)(ioport_ctrl_t *const p_ctrl, const ioport_cfg_t *p_cfg)
fsp_err_t(*)	pinCfg)(ioport_ctrl_t *const p_ctrl, bsp_io_port_pin_t pin, uint32_t cfg)
fsp_err_t(*)	pinEventInputRead)(ioport_ctrl_t *const p_ctrl, bsp_io_port_pin_t pin, bsp_io_level_t *p_pin_event)
fsp_err_t(*)	pinEventOutputWrite)(ioport_ctrl_t *const p_ctrl, bsp_io_port_pin_t pin, bsp_io_level_t pin_value)
fsp_err_t(*)	pinEthernetModeCfg)(ioport_ctrl_t *const p_ctrl, ioport_ethernet_channel_t channel, ioport_ethernet_mode_t mode)

<code>fsp_err_t(*</code>	<code>pinRead)(ioport_ctrl_t *const p_ctrl, bsp_io_port_pin_t pin, bsp_io_level_t *p_pin_value)</code>
<code>fsp_err_t(*</code>	<code>pinWrite)(ioport_ctrl_t *const p_ctrl, bsp_io_port_pin_t pin, bsp_io_level_t level)</code>
<code>fsp_err_t(*</code>	<code>portDirectionSet)(ioport_ctrl_t *const p_ctrl, bsp_io_port_t port, ioport_size_t direction_values, ioport_size_t mask)</code>
<code>fsp_err_t(*</code>	<code>portEventInputRead)(ioport_ctrl_t *const p_ctrl, bsp_io_port_t port, ioport_size_t *p_event_data)</code>
<code>fsp_err_t(*</code>	<code>portEventOutputWrite)(ioport_ctrl_t *const p_ctrl, bsp_io_port_t port, ioport_size_t event_data, ioport_size_t mask_value)</code>
<code>fsp_err_t(*</code>	<code>portRead)(ioport_ctrl_t *const p_ctrl, bsp_io_port_t port, ioport_size_t *p_port_value)</code>
<code>fsp_err_t(*</code>	<code>portWrite)(ioport_ctrl_t *const p_ctrl, bsp_io_port_t port, ioport_size_t value, ioport_size_t mask)</code>
<code>fsp_err_t(*</code>	<code>versionGet)(fsp_version_t *p_data)</code>

Field Documentation

◆ open

`fsp_err_t(* ioport_api_t::open) (ioport_ctrl_t *const p_ctrl, const ioport_cfg_t *p_cfg)`

Initialize internal driver data and initial pin configurations. Called during startup. Do not call this API during runtime. Use `ioport_api_t::pinsCfg` for runtime reconfiguration of multiple pins.

Implemented as

- `R_IOPORT_Open()`

Parameters

[in]	p_cfg	Pointer to pin configuration data array.

◆ **close**

```
fsp_err_t(* ioport_api_t::close) (ioport_ctrl_t *const p_ctrl)
```

Close the API.

Implemented as

◦ [R_IOPORT_Close\(\)](#)

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ **pinsCfg**

```
fsp_err_t(* ioport_api_t::pinsCfg) (ioport_ctrl_t *const p_ctrl, const ioport_cfg_t *p_cfg)
```

Configure multiple pins.

Implemented as

◦ [R_IOPORT_PinsCfg\(\)](#)

Parameters

[in]	p_cfg	Pointer to pin configuration data array.

◆ **pinCfg**

```
fsp_err_t(* ioport_api_t::pinCfg) (ioport_ctrl_t *const p_ctrl, bsp_io_port_pin_t pin, uint32_t cfg)
```

Configure settings for an individual pin.

Implemented as

◦ [R_IOPORT_PinCfg\(\)](#)

Parameters

[in]	pin	Pin to be read.
[in]	cfg	Configuration options for the pin.

◆ **pinEventInputRead**

```
fsp_err_t(* ioport_api_t::pinEventInputRead) (ioport_ctrl_t *const p_ctrl, bsp_io_port_pin_t pin,
bsp_io_level_t *p_pin_event)
```

Read the event input data of the specified pin and return the level.

Implemented as

◦ [R_IOPORT_PinEventInputRead\(\)](#)

Parameters

[in]	pin	Pin to be read.
[in]	p_pin_event	Pointer to return the event data.

◆ **pinEventOutputWrite**

```
fsp_err_t(* ioport_api_t::pinEventOutputWrite) (ioport_ctrl_t *const p_ctrl, bsp_io_port_pin_t pin,
bsp_io_level_t pin_value)
```

Write pin event data.

Implemented as

◦ [R_IOPORT_PinEventOutputWrite\(\)](#)

Parameters

[in]	pin	Pin event data is to be written to.
[in]	pin_value	Level to be written to pin output event.

◆ **pinEthernetModeCfg**

```
fsp_err_t(* ioport_api_t::pinEthernetModeCfg) (ioport_ctrl_t *const p_ctrl, ioport_ethernet_channel_t
channel, ioport_ethernet_mode_t mode)
```

Configure the PHY mode of the Ethernet channels.

Implemented as

◦ [R_IOPORT_EthernetModeCfg\(\)](#)

Parameters

[in]	channel	Channel configuration will be set for.
[in]	mode	PHY mode to set the channel to.

◆ **pinRead**

```
fsp_err_t(* ioport_api_t::pinRead) (ioport_ctrl_t *const p_ctrl, bsp_io_port_pin_t pin, bsp_io_level_t *p_pin_value)
```

Read level of a pin.

Implemented as

◦ [R_IOPORT_PinRead\(\)](#)

Parameters

[in]	pin	Pin to be read.
[in]	p_pin_value	Pointer to return the pin level.

◆ **pinWrite**

```
fsp_err_t(* ioport_api_t::pinWrite) (ioport_ctrl_t *const p_ctrl, bsp_io_port_pin_t pin, bsp_io_level_t level)
```

Write specified level to a pin.

Implemented as

◦ [R_IOPORT_PinWrite\(\)](#)

Parameters

[in]	pin	Pin to be written to.
[in]	level	State to be written to the pin.

◆ **portDirectionSet**

```
fsp_err_t(* ioport_api_t::portDirectionSet) (ioport_ctrl_t *const p_ctrl, bsp_io_port_t port, ioport_size_t direction_values, ioport_size_t mask)
```

Set the direction of one or more pins on a port.

Implemented as

◦ [R_IOPORT_PortDirectionSet\(\)](#)

Parameters

[in]	port	Port being configured.
[in]	direction_values	Value controlling direction of pins on port (1 - output, 0 - input).
[in]	mask	Mask controlling which pins on the port are to be configured.

◆ **portEventInputRead**

```
fsp_err_t(* ioport_api_t::portEventInputRead) (ioport_ctrl_t *const p_ctrl, bsp_io_port_t port,
ioport_size_t *p_event_data)
```

Read captured event data for a port.

Implemented as

- [R_IOPORT_PortEventInputRead\(\)](#)

Parameters

[in]	port	Port to be read.
[in]	p_event_data	Pointer to return the event data.

◆ **portEventOutputWrite**

```
fsp_err_t(* ioport_api_t::portEventOutputWrite) (ioport_ctrl_t *const p_ctrl, bsp_io_port_t port,
ioport_size_t event_data, ioport_size_t mask_value)
```

Write event output data for a port.

Implemented as

- [R_IOPORT_PortEventOutputWrite\(\)](#)

Parameters

[in]	port	Port event data will be written to.
[in]	event_data	Data to be written as event data to specified port.
[in]	mask_value	Each bit set to 1 in the mask corresponds to that bit's value in event data. being written to port.

◆ **portRead**

```
fsp_err_t(* ioport_api_t::portRead) (ioport_ctrl_t *const p_ctrl, bsp_io_port_t port, ioport_size_t *p_port_value)
```

Read states of pins on the specified port.

Implemented as

◦ [R_IOPORT_PortRead\(\)](#)

Parameters

[in]	port	Port to be read.
[in]	p_port_value	Pointer to return the port value.

◆ **portWrite**

```
fsp_err_t(* ioport_api_t::portWrite) (ioport_ctrl_t *const p_ctrl, bsp_io_port_t port, ioport_size_t value, ioport_size_t mask)
```

Write to multiple pins on a port.

Implemented as

◦ [R_IOPORT_PortWrite\(\)](#)

Parameters

[in]	port	Port to be written to.
[in]	value	Value to be written to the port.
[in]	mask	Mask controlling which pins on the port are written to.

◆ **versionGet**

```
fsp_err_t(* ioport_api_t::versionGet) (fsp_version_t *p_data)
```

Return the version of the IOPort driver.

Implemented as

◦ [R_IOPORT_VersionGet\(\)](#)

Parameters

[out]	p_data	Memory address to return version information to.

◆ **ioport_instance_t**

```
struct ioport_instance_t
```

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields		
ioport_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
ioport_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
ioport_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ [ioport_size_t](#)

typedef uint16_t ioport_size_t
IO port size on this device.
IO port type used with ports

◆ [ioport_ctrl_t](#)

typedef void ioport_ctrl_t
IOPORT control block. Allocate an instance specific control block to pass into the IOPORT API calls.
Implemented as ◦ ioport_instance_ctrl_t

Enumeration Type Documentation

◆ **ioport_peripheral_t**

enum ioport_peripheral_t	
Superset of all peripheral functions.	
Enumerator	
IOPORT_PERIPHERAL_IO	Pin will functions as an IO pin
IOPORT_PERIPHERAL_DEBUG	Pin will function as a DEBUG pin
IOPORT_PERIPHERAL_AGT	Pin will function as an AGT peripheral pin
IOPORT_PERIPHERAL_GPT0	Pin will function as a GPT peripheral pin
IOPORT_PERIPHERAL_GPT1	Pin will function as a GPT peripheral pin
IOPORT_PERIPHERAL_SCI0_2_4_6_8	Pin will function as an SCI peripheral pin
IOPORT_PERIPHERAL_SCI1_3_5_7_9	Pin will function as an SCI peripheral pin
IOPORT_PERIPHERAL_SPI	Pin will function as a SPI peripheral pin
IOPORT_PERIPHERAL_IIC	Pin will function as a IIC peripheral pin
IOPORT_PERIPHERAL_KEY	Pin will function as a KEY peripheral pin
IOPORT_PERIPHERAL_CLKOUT_COMP_RTC	Pin will function as a clock/comparator/RTC peripheral pin
IOPORT_PERIPHERAL_CAC_AD	Pin will function as a CAC/ADC peripheral pin
IOPORT_PERIPHERAL_BUS	Pin will function as a BUS peripheral pin
IOPORT_PERIPHERAL_CTSU	Pin will function as a CTSU peripheral pin
IOPORT_PERIPHERAL_LCDC	Pin will function as a segment LCD peripheral pin
IOPORT_PERIPHERAL_DALI	Pin will function as a DALI peripheral pin
IOPORT_PERIPHERAL_CAN	Pin will function as a CAN peripheral pin
IOPORT_PERIPHERAL_QSPI	Pin will function as a QSPI peripheral pin
IOPORT_PERIPHERAL_SSI	Pin will function as an SSI peripheral pin
IOPORT_PERIPHERAL_USB_FS	Pin will function as a USB full speed peripheral pin

IOPORT_PERIPHERAL_USB_HS	Pin will function as a USB high speed peripheral pin
IOPORT_PERIPHERAL_SDHI_MMC	Pin will function as an SD/MMC peripheral pin
IOPORT_PERIPHERAL_ETHER_MII	Pin will function as an Ethernet MII peripheral pin
IOPORT_PERIPHERAL_ETHER_RMII	Pin will function as an Ethernet RMMI peripheral pin
IOPORT_PERIPHERAL_PDC	Pin will function as a PDC peripheral pin
IOPORT_PERIPHERAL_LCD_GRAPHICS	Pin will function as a graphics LCD peripheral pin
IOPORT_PERIPHERAL_TRACE	Pin will function as a debug trace peripheral pin
IOPORT_PERIPHERAL_END	Marks end of enum - used by parameter checking

◆ ioport_ethernet_channel_t

enum ioport_ethernet_channel_t	
Superset of Ethernet channels.	
Enumerator	
IOPORT_ETHERNET_CHANNEL_0	Used to select Ethernet channel 0.
IOPORT_ETHERNET_CHANNEL_1	Used to select Ethernet channel 1.
IOPORT_ETHERNET_CHANNEL_END	Marks end of enum - used by parameter checking.

◆ **ioport_ethernet_mode_t**

enum ioport_ethernet_mode_t	
Superset of Ethernet PHY modes.	
Enumerator	
IOPORT_ETHERNET_MODE_RMII	Ethernet PHY mode set to MII.
IOPORT_ETHERNET_MODE_MII	Ethernet PHY mode set to RMII.
IOPORT_ETHERNET_MODE_END	Marks end of enum - used by parameter checking.

◆ **ioport_cfg_options_t**

enum ioport_cfg_options_t	
Options to configure pin functions	
Enumerator	
IOPORT_CFG_PORT_DIRECTION_INPUT	Sets the pin direction to input (default)
IOPORT_CFG_PORT_DIRECTION_OUTPUT	Sets the pin direction to output.
IOPORT_CFG_PORT_OUTPUT_LOW	Sets the pin level to low.
IOPORT_CFG_PORT_OUTPUT_HIGH	Sets the pin level to high.
IOPORT_CFG_PULLUP_ENABLE	Enables the pin's internal pull-up.
IOPORT_CFG_PIM_TTL	Enables the pin's input mode.
IOPORT_CFG_NMOS_ENABLE	Enables the pin's NMOS open-drain output.
IOPORT_CFG_PMO5_ENABLE	Enables the pin's PMOS open-drain output.
IOPORT_CFG_DRIVE_MID	Sets pin drive output to medium.
IOPORT_CFG_DRIVE_MID_IIC	Sets pin to drive output needed for IIC on a 20mA port.
IOPORT_CFG_DRIVE_HIGH	Sets pin drive output to high.
IOPORT_CFG_EVENT_RISING_EDGE	Sets pin event trigger to rising edge.
IOPORT_CFG_EVENT_FALLING_EDGE	Sets pin event trigger to falling edge.
IOPORT_CFG_EVENT_BOTH_EDGES	Sets pin event trigger to both edges.
IOPORT_CFG_IRQ_ENABLE	Sets pin as an IRQ pin.
IOPORT_CFG_ANALOG_ENABLE	Enables pin to operate as an analog pin.
IOPORT_CFG_PERIPHERAL_PIN	Enables pin to operate as a peripheral pin.

◆ **ioport_pwpr_t**

enum <code>ioport_pwpr_t</code>	
Enumerator	
<code>IOPORT_PFS_WRITE_DISABLE</code>	Disable PFS write access.
<code>IOPORT_PFS_WRITE_ENABLE</code>	Enable PFS write access.

4.3.19 JPEG Codec Interface[Interfaces](#)**Detailed Description**

Interface for JPEG functions.

Data Structures

struct `jpeg_decode_callback_args_t`

struct `jpeg_decode_cfg_t`

struct `jpeg_decode_api_t`

struct `jpeg_decode_instance_t`

Macros

#define `JPEG_DECODE_API_VERSION_MAJOR`

Typedefs

typedef void `jpeg_decode_ctrl_t`

Enumerations

enum `jpeg_decode_color_space_t`

enum `jpeg_data_order_t`

enum `jpeg_decode_pixel_format_t`

enum `jpeg_decode_status_t`

enum `jpeg_decode_subsample_t`

enum [jpeg_decode_count_enable_t](#)

enum [jpeg_decode_resume_mode_t](#)

Data Structure Documentation

◆ jpeg_decode_callback_args_t

struct jpeg_decode_callback_args_t		
Callback status structure		
Data Fields		
jpeg_decode_status_t	status	JPEG status.
void const *	p_context	Pointer to user-provided context.

◆ jpeg_decode_cfg_t

struct jpeg_decode_cfg_t	
User configuration structure, used in open function.	
Data Fields	
jpeg_decode_color_space_t	color_space
	Color space.
jpeg_data_order_t	input_data_order
	Input data stream byte order.
jpeg_data_order_t	output_data_order
	Output data stream byte order.
jpeg_decode_pixel_format_t	pixel_format
	Pixel format.
uint8_t	alpha_value
	Alpha value to be applied to decoded pixel data. Only valid for ARGB8888 format.

IRQn_Type	jedi_irq
	Data transfer interrupt IRQ number.
IRQn_Type	jdti_irq
	Decompression interrupt IRQ number.
uint8_t	jdti_ipl
	Data transfer interrupt priority.
uint8_t	jedi_ipl
	Decompression interrupt priority.
void(*	p_callback)(jpeg_decode_callback_args_t *p_args)
	User-supplied callback functions.
void const *	p_context
	Placeholder for user data. Passed to user callback in jpeg_decode_callback_args_t .

◆ jpeg_decode_api_t

struct jpeg_decode_api_t	
JPEG functions implemented at the HAL layer will follow this API.	
Data Fields	
fsp_err_t (*	open)(jpeg_decode_ctrl_t *const p_ctrl, jpeg_decode_cfg_t const *const p_cfg)
fsp_err_t (*	outputBufferSet)(jpeg_decode_ctrl_t *const p_ctrl, void *p_buffer, uint32_t buffer_size)
fsp_err_t (*	horizontalStrideSet)(jpeg_decode_ctrl_t *const p_ctrl, uint32_t

	horizontal_stride)
fsp_err_t(*	imageSubsampleSet)(jpeg_decode_ctrl_t *const p_ctrl, jpeg_decode_subsample_t horizontal_subsample, jpeg_decode_subsample_t vertical_subsample)
fsp_err_t(*	inputBufferSet)(jpeg_decode_ctrl_t *const p_ctrl, void *p_buffer, uint32_t buffer_size)
fsp_err_t(*	linesDecodedGet)(jpeg_decode_ctrl_t *const p_ctrl, uint32_t *const p_lines)
fsp_err_t(*	imageSizeGet)(jpeg_decode_ctrl_t *const p_ctrl, uint16_t *p_horizontal_size, uint16_t *p_vertical_size)
fsp_err_t(*	statusGet)(jpeg_decode_ctrl_t *const p_ctrl, jpeg_decode_status_t *const p_status)
fsp_err_t(*	close)(jpeg_decode_ctrl_t *const p_ctrl)
fsp_err_t(*	versionGet)(fsp_version_t *p_version)
fsp_err_t(*	pixelFormatGet)(jpeg_decode_ctrl_t *const p_ctrl, jpeg_decode_color_space_t *const p_color_space)

Field Documentation

◆ **open**

```
fsp_err_t(* jpeg_decode_api_t::open) (jpeg_decode_ctrl_t *const p_ctrl, jpeg_decode_cfg_t const *const p_cfg)
```

Initial configuration

Implemented as

◦ [R_JPEG_Decode_Open\(\)](#)

Precondition

none

Parameters

[in,out]	p_ctrl	Pointer to control block. Must be declared by user. Elements set here.
[in]	p_cfg	Pointer to configuration structure. All elements of this structure must be set by user.

◆ **outputBufferSet**

```
fsp_err_t(* jpeg_decode_api_t::outputBufferSet) (jpeg_decode_ctrl_t *const p_ctrl, void *p_buffer, uint32_t buffer_size)
```

Assign output buffer to JPEG codec for storing output data.

Implemented as

◦ [R_JPEG_Decode_OutputBufferSet\(\)](#)

Precondition

The JPEG codec module must have been opened properly.

Note

The buffer starting address must be 8-byte aligned. For the decoding process, the HLD driver automatically computes the number of lines of the image to decoded so the output data fits into the given space. If the supplied output buffer is not able to hold the entire frame, the application should call the Output Full Callback function so it can be notified when additional buffer space is needed.

Parameters

[in]	p_ctrl	Control block set in jpeg_decode_api_t::open call.
[in]	p_buffer	Pointer to the output buffer space
[in]	buffer_size	Size of the output buffer

◆ **horizontalStrideSet**

```
fsp_err_t(* jpeg_decode_api_t::horizontalStrideSet) (jpeg_decode_ctrl_t *const p_ctrl, uint32_t horizontal_stride)
```

Configure the horizontal stride value.

Implemented as

◦ [R_JPEG_Decode_HorizontalStrideSet\(\)](#)

Precondition

The JPEG codec module must have been opened properly.

Parameters

[in]	p_ctrl	Control block set in jpeg_decode_api_t::open call.
[in]	horizontal_stride	Horizontal stride value to be used for the decoded image data.
[in]	buffer_size	Size of the output buffer

◆ **imageSubsampleSet**

```
fsp_err_t(* jpeg_decode_api_t::imageSubsampleSet) (jpeg_decode_ctrl_t *const p_ctrl, jpeg_decode_subsample_t horizontal_subsample, jpeg_decode_subsample_t vertical_subsample)
```

Configure the horizontal and vertical subsample settings.

Implemented as

◦ [R_JPEG_Decode_ImageSubsampleSet\(\)](#)

Precondition

The JPEG codec module must have been opened properly.

Parameters

[in]	p_ctrl	Control block set in jpeg_decode_api_t::open call.
[in]	horizontal_subsample	Horizontal subsample value
[in]	vertical_subsample	Vertical subsample value

◆ **inputBufferSet**

```
fsp_err_t(* jpeg_decode_api_t::inputBufferSet) (jpeg_decode_ctrl_t *const p_ctrl, void *p_buffer,
uint32_t buffer_size)
```

Assign input data buffer to JPEG codec.

Implemented as

- [R_JPEG_Decode_InputBufferSet\(\)](#)

Precondition

the JPEG codec module must have been opened properly.

Note

The buffer starting address must be 8-byte aligned.

Parameters

[in]	p_ctrl	Control block set in jpeg_decode_api_t::open call.
[in]	p_buffer	Pointer to the input buffer space
[in]	buffer_size	Size of the input buffer

◆ **linesDecodedGet**

```
fsp_err_t(* jpeg_decode_api_t::linesDecodedGet) (jpeg_decode_ctrl_t *const p_ctrl, uint32_t *const
p_lines)
```

Return the number of lines decoded into the output buffer.

Implemented as

- [R_JPEG_Decode_LinesDecodedGet\(\)](#)

Precondition

the JPEG codec module must have been opened properly.

Parameters

[in]	p_ctrl	Control block set in jpeg_decode_api_t::open call.
[out]	p_lines	Number of lines decoded

◆ **imageSizeGet**

```
fsp_err_t(* jpeg_decode_api_t::imageSizeGet) (jpeg_decode_ctrl_t *const p_ctrl, uint16_t
*p_horizontal_size, uint16_t *p_vertical_size)
```

Retrieve image size during decoding operation.

Implemented as

- [R_JPEG_Decode_ImageSizeGet\(\)](#)

Precondition

the JPEG codec module must have been opened properly.

Note

If the encoding or the decoding operation is finished without errors, the HLD driver automatically closes the device. In this case, application does not need to explicitly close the JPEG device.

Parameters

[in]	p_ctrl	Control block set in jpeg_decode_api_t::open call.
[out]	p_horizontal_size	Image horizontal size, in number of pixels.
[out]	p_vertical_size	Image vertical size, in number of pixels.

◆ **statusGet**

```
fsp_err_t(* jpeg_decode_api_t::statusGet) (jpeg_decode_ctrl_t *const p_ctrl, jpeg_decode_status_t
*const p_status)
```

Retrieve current status of the JPEG codec module.

Implemented as

- [R_JPEG_Decode_StatusGet\(\)](#)

Precondition

the JPEG codec module must have been opened properly.

Parameters

[in]	p_ctrl	Control block set in jpeg_decode_api_t::open call.
[out]	p_status	JPEG module status

◆ **close**

```
fsp_err_t(* jpeg_decode_api_t::close) (jpeg_decode_ctrl_t *const p_ctrl)
```

Cancel an outstanding operation.

Implemented as

- [R_JPEG_Decode_Close\(\)](#)

Precondition

the JPEG codec module must have been opened properly.

Note

If the encoding or the decoding operation is finished without errors, the HLD driver automatically closes the device. In this case, application does not need to explicitly close the JPEG device.

Parameters

[in]	p_ctrl	Control block set in jpeg_decode_api_t::open call.

◆ **versionGet**

```
fsp_err_t(* jpeg_decode_api_t::versionGet) (fsp_version_t *p_version)
```

Get version and store it in provided pointer p_version.

Implemented as

- [R_JPEG_Decode_VersionGet\(\)](#)

Parameters

[out]	p_version	Code and API version used.

◆ **pixelFormatGet**

```
fsp_err_t(* jpeg_decode_api_t::pixelFormatGet) (jpeg_decode_ctrl_t *const p_ctrl,  
jpeg_decode_color_space_t *const p_color_space)
```

Get the input pixel format.

Implemented as

- [R_JPEG_Decode_PixelFormatGet\(\)](#)

Precondition

the JPEG codec module must have been opened properly.

Parameters

[in]	p_ctrl	Control block set in jpeg_decode_api_t::open call.
[out]	p_color_space	JPEG input format.

◆ **jpeg_decode_instance_t**

struct jpeg_decode_instance_t		
This structure encompasses everything that is needed to use an instance of this interface.		
Data Fields		
jpeg_decode_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
jpeg_decode_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
jpeg_decode_api_t const *	p_api	Pointer to the API structure for this instance.

Macro Definition Documentation

◆ JPEG_DECODE_API_VERSION_MAJOR

#define JPEG_DECODE_API_VERSION_MAJOR
Register definitions, common services and error codes. Configuration for this module

Typedef Documentation

◆ jpeg_decode_ctrl_t

typedef void jpeg_decode_ctrl_t
JPEG decode control block. Allocate an instance specific control block to pass into the JPEG decode API calls.
Implemented as
◦ jpeg_decode_instance_ctrl_t

Enumeration Type Documentation

◆ jpeg_decode_color_space_t

enum jpeg_decode_color_space_t	
Image color space definitions	
Enumerator	
JPEG_DECODE_COLOR_SPACE_YCBCR444	Color Space YCbCr 444.
JPEG_DECODE_COLOR_SPACE_YCBCR422	Color Space YCbCr 422.
JPEG_DECODE_COLOR_SPACE_YCBCR420	Color Space YCbCr 420.
JPEG_DECODE_COLOR_SPACE_YCBCR411	Color Space YCbCr 411.

◆ jpeg_data_order_t

enum jpeg_data_order_t	
Multi-byte Data Format	
Enumerator	
JPEG_DATA_ORDER_NORMAL	(1)(2)(3)(4)(5)(6)(7)(8) Normal byte order
JPEG_DATA_ORDER_BYTE_SWAP	(2)(1)(4)(3)(6)(5)(8)(7) Byte Swap
JPEG_DATA_ORDER_WORD_SWAP	(3)(4)(1)(2)(7)(8)(5)(6) Word Swap
JPEG_DATA_ORDER_WORD_BYTE_SWAP	(4)(3)(2)(1)(8)(7)(6)(5) Word-Byte Swap
JPEG_DATA_ORDER_LONGWORD_SWAP	(5)(6)(7)(8)(1)(2)(3)(4) Longword Swap
JPEG_DATA_ORDER_LONGWORD_BYTE_SWAP	(6)(5)(8)(7)(2)(1)(4)(3) Longword Byte Swap
JPEG_DATA_ORDER_LONGWORD_WORD_SWAP	(7)(8)(5)(6)(3)(4)(1)(2) Longword Word Swap
JPEG_DATA_ORDER_LONGWORD_WORD_BYTE_SWAP	(8)(7)(6)(5)(4)(3)(2)(1) Longword Word Byte Swap

◆ jpeg_decode_pixel_format_t

enum jpeg_decode_pixel_format_t	
Pixel Data Format	
Enumerator	
JPEG_DECODE_PIXEL_FORMAT_ARGB8888	Pixel Data ARGB8888 format.
JPEG_DECODE_PIXEL_FORMAT_RGB565	Pixel Data RGB565 format.

◆ jpeg_decode_status_t

enum jpeg_decode_status_t	
JPEG HLD driver internal status information. The driver can simultaneously be in more than any one status at the same time. Parse the status bit-fields using the definitions in this enum to determine driver status	
Enumerator	
JPEG_DECODE_STATUS_NOT_OPEN	JPEG codec module is not yet open.
JPEG_DECODE_STATUS_OPEN	JPEG Codec module is open, and is not operational.
JPEG_DECODE_STATUS_RUNNING	JPEG Codec is running.
JPEG_DECODE_STATUS_DONE	JPEG Codec has successfully finished the operation.
JPEG_DECODE_STATUS_INPUT_PAUSE	JPEG Codec paused waiting for more input data.
JPEG_DECODE_STATUS_OUTPUT_PAUSE	JPEG Codec paused after decoded the number of lines specified by user.
JPEG_DECODE_STATUS_IMAGE_SIZE_READY	JPEG decoding operation obtained image size, and paused.
JPEG_DECODE_STATUS_ERROR	JPEG Codec module encountered an error.
JPEG_DECODE_STATUS_HEADER_PROCESSING	JPEG Codec module is reading the JPEG header information.

◆ jpeg_decode_subsample_t

enum jpeg_decode_subsample_t	
Data type for horizontal and vertical subsample settings. This setting applies only to the decoding operation.	
Enumerator	
JPEG_DECODE_OUTPUT_NO_SUBSAMPLE	No subsample. The image is decoded with no reduction in size.
JPEG_DECODE_OUTPUT_SUBSAMPLE_HALF	The output image size is reduced by half.
JPEG_DECODE_OUTPUT_SUBSAMPLE_ONE_QUARTER	The output image size is reduced to one-quarter.
JPEG_DECODE_OUTPUT_SUBSAMPLE_ONE_EIGHTH	The output image size is reduced to one-eighth.

◆ jpeg_decode_count_enable_t

enum jpeg_decode_count_enable_t	
Data type for decoding count mode enable.	
Enumerator	
JPEG_DECODE_COUNT_DISABLE	Count mode disable.
JPEG_DECODE_COUNT_ENABLE	Count mode enable.

◆ jpeg_decode_resume_mode_t

enum jpeg_decode_resume_mode_t	
Data type for decoding count mode enable.	
Enumerator	
JPEG_DECODE_COUNT_MODE_ADDRESS_CONTINUE	The data buffer address will not be initialized when resuming image data lines.
JPEG_DECODE_COUNT_MODE_ADDRESS_REINITIALIZE	The data buffer address will be initialized when resuming image data lines.

4.3.20 Key Matrix Interface

Interfaces

Detailed Description

Interface for key matrix functions.

Summary

The KEYMATRIX interface provides standard KeyMatrix functionality including event generation on a rising or falling edge for one or more channels at the same time. The generated event indicates all channels that are active in that instant via a bit mask. This allows the interface to be used with a matrix configuration or a one-to-one hardware implementation that is triggered on either a rising or a falling edge.

Implemented by:

- [Key Interrupt \(r_kint\)](#)

Data Structures

struct [keymatrix_callback_args_t](#)

struct [keymatrix_cfg_t](#)

struct [keymatrix_api_t](#)

struct [keymatrix_instance_t](#)

Macros

```
#define KEYMATRIX\_API\_VERSION\_MAJOR
KEY MATRIX API version number (Major)
```

```
#define KEYMATRIX\_API\_VERSION\_MINOR
KEY MATRIX API version number (Minor)
```

Typedefs

typedef void [keymatrix_ctrl_t](#)

Enumerations

enum [keymatrix_trigger_t](#)

Data Structure Documentation

◆ [keymatrix_callback_args_t](#)

struct keymatrix_callback_args_t		
Callback function parameter data		
Data Fields		
void const *	p_context	Holder for user data. Set in keymatrix_api_t::open function in keymatrix_cfg_t .
uint32_t	channel_mask	Bit vector representing the physical hardware channel(s) that caused the interrupt.

◆ **keymatrix_cfg_t**

struct keymatrix_cfg_t	
User configuration structure, used in open function	
Data Fields	
uint32_t	channel_mask
	Key Input channel(s). Bit mask of channels to open.
keymatrix_trigger_t	trigger
	Key Input trigger setting.
uint8_t	ipl
	Interrupt priority level.
IRQn_Type	irq
	NVIC IRQ number.
void(*)	p_callback)(keymatrix_callback_args_t *p_args)
	Callback for key interrupt ISR.
void const *	p_context
	Holder for user data. Passed to callback in keymatrix_user_cb_data_t .

void const *	p_extend
	Extension parameter for hardware specific settings.

◆ keymatrix_api_t

struct keymatrix_api_t		
Key Matrix driver structure. Key Matrix functions implemented at the HAL layer will use this API.		
Data Fields		
fsp_err_t(*	open	)(keymatrix_ctrl_t *const p_ctrl, keymatrix_cfg_t const *const p_cfg)
fsp_err_t(*	enable	)(keymatrix_ctrl_t *const p_ctrl)
fsp_err_t(*	disable	)(keymatrix_ctrl_t *const p_ctrl)
fsp_err_t(*	close	)(keymatrix_ctrl_t *const p_ctrl)
fsp_err_t(*	versionGet	)(fsp_version_t *const p_version)
Field Documentation		
◆ open		
fsp_err_t(* keymatrix_api_t::open) (keymatrix_ctrl_t *const p_ctrl, keymatrix_cfg_t const *const p_cfg)		
Initial configuration.		
Implemented as		
◦ R_KINT_KEYMATRIX_Open()		
Parameters		
[out]	p_ctrl	Pointer to control block. Must be declared by user. Value set in this function.
[in]	p_cfg	Pointer to configuration structure. All elements of the structure must be set by user.

◆ enable

`fsp_err_t(* keymatrix_api_t::enable) (keymatrix_ctrl_t *const p_ctrl)`

Enable Key interrupt

Implemented as

- R_KINT_KEYMATRIX_Enable()

Parameters

[in]	p_ctrl	Control block pointer set in Open call for this Key interrupt.

◆ disable

`fsp_err_t(* keymatrix_api_t::disable) (keymatrix_ctrl_t *const p_ctrl)`

Disable Key interrupt.

Implemented as

- R_KINT_KEYMATRIX_Disable()

Parameters

[in]	p_ctrl	Control block pointer set in Open call for this Key interrupt.

◆ close

`fsp_err_t(* keymatrix_api_t::close) (keymatrix_ctrl_t *const p_ctrl)`

Allow driver to be reconfigured. May reduce power consumption.

Implemented as

- R_KINT_KEYMATRIX_Close()

Parameters

[in]	p_ctrl	Control block pointer set in Open call for this Key interrupt.

◆ **versionGet**

```
fsp_err_t(* keymatrix_api_t::versionGet) (fsp_version_t *const p_version)
```

Get version and store it in provided pointer p_version.

Implemented as

- [R_KINT_VersionGet\(\)](#)

Parameters

[out]	p_version	Code and API version used.

◆ **keymatrix_instance_t**

```
struct keymatrix_instance_t
```

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields

keymatrix_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
keymatrix_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
keymatrix_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation◆ **keymatrix_ctrl_t**

```
typedef void keymatrix\_ctrl\_t
```

Key matrix control block. Allocate an instance specific control block to pass into the key matrix API calls.

Implemented as

- [kint_instance_ctrl_t](#)

Enumeration Type Documentation

◆ **keymatrix_trigger_t**

enum <code>keymatrix_trigger_t</code>	
Trigger type: rising edge, falling edge	
Enumerator	
<code>KEYMATRIX_TRIG_FALLING</code>	Falling edge trigger.
<code>KEYMATRIX_TRIG_RISING</code>	Rising edge trigger.

4.3.21 Low Power Modes Interface**Interfaces****Detailed Description**

Interface for accessing low power modes.

Summary

This section defines the API for the LPM (Low Power Mode) Driver. The LPM Driver provides functions for controlling power consumption by configuring and transitioning to a low power mode. The LPM driver supports configuration of MCU low power modes using the LPM hardware peripheral. The LPM driver supports low power modes deep standby, standby, sleep, and snooze.

Note

Not all low power modes are available on all MCUs.

The LPM interface is implemented by:

- [Low Power Modes \(r_lpm\)](#)

Data Structures

struct [lpm_cfg_t](#)

struct [lpm_api_t](#)

struct [lpm_instance_t](#)

Macros

#define [LPM_API_VERSION_MAJOR](#)

Typedefs

typedef void [lpm_ctrl_t](#)

Enumerations

enum [lpm_mode_t](#)

enum [lpm_snooze_request_t](#)

enum [lpm_snooze_end_t](#)

enum [lpm_snooze_cancel_t](#)

enum [lpm_snooze_dtc_t](#)

enum [lpm_standby_wake_source_t](#)

enum [lpm_io_port_t](#)

enum [lpm_power_supply_t](#)

enum [lpm_deep_standby_cancel_edge_t](#)

enum [lpm_deep_standby_cancel_source_t](#)

enum [lpm_output_port_enable_t](#)

Data Structure Documentation

◆ lpm_cfg_t

struct lpm_cfg_t		
User configuration structure, used in open function		
Data Fields		
lpm_mode_t	low_power_mode	Low Power Mode
lpm_standby_wake_source_bits_t	standby_wake_sources	Bitwise list of sources to wake from standby
lpm_snooze_request_t	snooze_request_source	Snooze request source
lpm_snooze_end_bits_t	snooze_end_sources	Bitwise list of snooze end sources
lpm_snooze_cancel_t	snooze_cancel_sources	list of snooze cancel sources
lpm_snooze_dtc_t	dtc_state_in_snooze	State of DTC in snooze mode, enabled or disabled
void const *	p_extend	Placeholder for extension.

◆ lpm_api_t

struct lpm_api_t

lpm driver structure. General lpm functions implemented at the HAL layer will follow this API.

Data Fields

<code>fsp_err_t(*</code>	<code>open</code> <code>)(lpm_ctrl_t *const p_api_ctrl, lpm_cfg_t const *const p_cfg)</code>

<code>fsp_err_t(*</code>	<code>close</code> <code>)(lpm_ctrl_t *const p_api_ctrl)</code>

<code>fsp_err_t(*</code>	<code>lowPowerReconfigure</code> <code>)(lpm_ctrl_t *const p_api_ctrl, lpm_cfg_t const *const p_cfg)</code>

<code>fsp_err_t(*</code>	<code>lowPowerModeEnter</code> <code>)(lpm_ctrl_t *const p_api_ctrl)</code>

<code>fsp_err_t(*</code>	<code>ioKeepClear</code> <code>)(lpm_ctrl_t *const p_api_ctrl)</code>

<code>fsp_err_t(*</code>	<code>versionGet</code> <code>)(fsp_version_t *const p_version)</code>

Field Documentation

◆ open

<code>fsp_err_t(* lpm_api_t::open)</code> <code>(lpm_ctrl_t *const p_api_ctrl, lpm_cfg_t const *const p_cfg)</code>

Initialization function

Implemented as

- `R_LPM_Init()`

◆ close

<code>fsp_err_t(* lpm_api_t::close)</code> <code>(lpm_ctrl_t *const p_api_ctrl)</code>

Initialization function

Implemented as

- `R_LPM_Close()`

◆ **lowPowerReconfigure**

```
fsp_err_t(* lpm_api_t::lowPowerReconfigure) (lpm_ctrl_t *const p_api_ctrl, lpm_cfg_t const *const p_cfg)
```

Configure a low power mode.

Implemented as

- R_LPM_LowPowerConfigure()

Parameters

[in]	p_cfg	Pointer to configuration structure. All elements of this structure must be set by user.

◆ **lowPowerModeEnter**

```
fsp_err_t(* lpm_api_t::lowPowerModeEnter) (lpm_ctrl_t *const p_api_ctrl)
```

Enter low power mode (sleep/standby/deep standby) using WFI macro. Function will return after waking from low power mode.

Implemented as

- R_LPM_LowPowerModeEnter()

◆ **ioKeepClear**

```
fsp_err_t(* lpm_api_t::ioKeepClear) (lpm_ctrl_t *const p_api_ctrl)
```

Clear the IOKEEP bit after deep software standby.

- **Implemented as**
- R_LPM_IoKeepClear()

◆ **versionGet**

```
fsp_err_t(* lpm_api_t::versionGet) (fsp_version_t *const p_version)
```

Get the driver version based on compile time macros.

Implemented as

- R_LPM_VersionGet()

Parameters

[out]	p_version	Code and API version used.

◆ **lpm_instance_t**

```
struct lpm_instance_t
```

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields		
lpm_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
lpm_cfg_t const *const	p_cfg	Pointer to the configuration structure for this instance.
lpm_api_t const *const	p_api	Pointer to the API structure for this instance.

Macro Definition Documentation

◆ LPM_API_VERSION_MAJOR

#define LPM_API_VERSION_MAJOR
Register definitions, common services and error codes.

Typedef Documentation

◆ lpm_ctrl_t

typedef void lpm_ctrl_t
LPM control block. Allocate an instance specific control block to pass into the LPM API calls.
Implemented as ◦ lpm_instance_ctrl_t

Enumeration Type Documentation

◆ lpm_mode_t

enum lpm_mode_t	
Low power modes	
Enumerator	
LPM_MODE_SLEEP	Sleep mode.
LPM_MODE_STANDBY	Software Standby mode.
LPM_MODE_STANDBY_SNOOZE	Software Standby mode with Snooze mode enabled.
LPM_MODE_DEEP	Deep Software Standby mode.

◆ **lpm_snooze_request_t**

enum lpm_snooze_request_t	
Snooze request sources	
Enumerator	
LPM_SNOOZE_REQUEST_RXD0_FALLING	Enable RXD0 falling edge snooze request.
LPM_SNOOZE_REQUEST_IRQ0	Enable IRQ0 pin snooze request.
LPM_SNOOZE_REQUEST_IRQ1	Enable IRQ1 pin snooze request.
LPM_SNOOZE_REQUEST_IRQ2	Enable IRQ2 pin snooze request.
LPM_SNOOZE_REQUEST_IRQ3	Enable IRQ3 pin snooze request.
LPM_SNOOZE_REQUEST_IRQ4	Enable IRQ4 pin snooze request.
LPM_SNOOZE_REQUEST_IRQ5	Enable IRQ5 pin snooze request.
LPM_SNOOZE_REQUEST_IRQ6	Enable IRQ6 pin snooze request.
LPM_SNOOZE_REQUEST_IRQ7	Enable IRQ7 pin snooze request.
LPM_SNOOZE_REQUEST_IRQ8	Enable IRQ8 pin snooze request.
LPM_SNOOZE_REQUEST_IRQ9	Enable IRQ9 pin snooze request.
LPM_SNOOZE_REQUEST_IRQ10	Enable IRQ10 pin snooze request.
LPM_SNOOZE_REQUEST_IRQ11	Enable IRQ11 pin snooze request.
LPM_SNOOZE_REQUEST_IRQ12	Enable IRQ12 pin snooze request.
LPM_SNOOZE_REQUEST_IRQ13	Enable IRQ13 pin snooze request.
LPM_SNOOZE_REQUEST_IRQ14	Enable IRQ14 pin snooze request.
LPM_SNOOZE_REQUEST_IRQ15	Enable IRQ15 pin snooze request.
LPM_SNOOZE_REQUEST_KEY	Enable KR snooze request.
LPM_SNOOZE_REQUEST_ACMPS0	Enable High-speed analog comparator 0 snooze request.
LPM_SNOOZE_REQUEST_RTC_ALARM	Enable RTC alarm snooze request.

LPM_SNOOZE_REQUEST_RTC_PERIOD	Enable RTC period snooze request.
LPM_SNOOZE_REQUEST_AGT1_UNDERFLOW	Enable AGT1 underflow snooze request.
LPM_SNOOZE_REQUEST_AGT1_COMPARE_A	Enable AGT1 compare match A snooze request.
LPM_SNOOZE_REQUEST_AGT1_COMPARE_B	Enable AGT1 compare match B snooze request.

◆ lpm_snooze_end_t

enum lpm_snooze_end_t	
Snooze end control	
Enumerator	
LPM_SNOOZE_END_STANDBY_WAKE_SOURCES	Transition from Snooze to Normal mode directly.
LPM_SNOOZE_END_AGT1_UNDERFLOW	AGT1 underflow.
LPM_SNOOZE_END_DTC_TRANS_COMPLETE	Last DTC transmission completion.
LPM_SNOOZE_END_DTC_TRANS_COMPLETE_NEGATED	Not Last DTC transmission completion.
LPM_SNOOZE_END_ADC0_COMPARE_MATCH	ADC Channel 0 compare match.
LPM_SNOOZE_END_ADC0_COMPARE_MISMATCH	ADC Channel 0 compare mismatch.
LPM_SNOOZE_END_ADC1_COMPARE_MATCH	ADC 1 compare match.
LPM_SNOOZE_END_ADC1_COMPARE_MISMATCH	ADC 1 compare mismatch.
LPM_SNOOZE_END_SCI0_ADDRESS_MATCH	SCI0 address mismatch.

◆ **lpm_snooze_cancel_t**

enum <code>lpm_snooze_cancel_t</code>	
Snooze cancel control	
Enumerator	
LPM_SNOOZE_CANCEL_SOURCE_ADC0_WCMPPM	ADC Channel 0 window compare match.
LPM_SNOOZE_CANCEL_SOURCE_ADC0_WCMPUM	ADC Channel 0 window compare mismatch.
LPM_SNOOZE_CANCEL_SOURCE_ADC1_WCMPPM	ADC Channel 1 window compare match.
LPM_SNOOZE_CANCEL_SOURCE_ADC1_WCMPUM	ADC Channel 1 window compare mismatch.
LPM_SNOOZE_CANCEL_SOURCE_SCI0_AM	SCI0 address match event.
LPM_SNOOZE_CANCEL_SOURCE_SCI0_RXI_OR_ERI	SCI0 receive error.
LPM_SNOOZE_CANCEL_SOURCE_DTC_COMPLETE	DTC transfer completion.
LPM_SNOOZE_CANCEL_SOURCE_DOC_DOPCI	Data operation circuit interrupt.
LPM_SNOOZE_CANCEL_SOURCE_CTSU_CTSUFN	CTSU measurement end interrupt.

◆ **lpm_snooze_dtc_t**

enum <code>lpm_snooze_dtc_t</code>	
DTC Enable in Snooze Mode	
Enumerator	
LPM_SNOOZE_DTC_DISABLE	Disable DTC operation.
LPM_SNOOZE_DTC_ENABLE	Enable DTC operation.

◆ **lpm_standby_wake_source_t**

enum lpm_standby_wake_source_t	
Wake from standby mode sources, does not apply to sleep or deep standby modes	
Enumerator	
LPM_STANDBY_WAKE_SOURCE_IRQ0	IRQ0.
LPM_STANDBY_WAKE_SOURCE_IRQ1	IRQ1.
LPM_STANDBY_WAKE_SOURCE_IRQ2	IRQ2.
LPM_STANDBY_WAKE_SOURCE_IRQ3	IRQ3.
LPM_STANDBY_WAKE_SOURCE_IRQ4	IRQ4.
LPM_STANDBY_WAKE_SOURCE_IRQ5	IRQ5.
LPM_STANDBY_WAKE_SOURCE_IRQ6	IRQ6.
LPM_STANDBY_WAKE_SOURCE_IRQ7	IRQ7.
LPM_STANDBY_WAKE_SOURCE_IRQ8	IRQ8.
LPM_STANDBY_WAKE_SOURCE_IRQ9	IRQ9.
LPM_STANDBY_WAKE_SOURCE_IRQ10	IRQ10.
LPM_STANDBY_WAKE_SOURCE_IRQ11	IRQ11.
LPM_STANDBY_WAKE_SOURCE_IRQ12	IRQ12.
LPM_STANDBY_WAKE_SOURCE_IRQ13	IRQ13.
LPM_STANDBY_WAKE_SOURCE_IRQ14	IRQ14.
LPM_STANDBY_WAKE_SOURCE_IRQ15	IRQ15.
LPM_STANDBY_WAKE_SOURCE_IWDT	Independent watchdog interrupt.
LPM_STANDBY_WAKE_SOURCE_KEY	Key interrupt.
LPM_STANDBY_WAKE_SOURCE_LVD1	Low Voltage Detection 1 interrupt.
LPM_STANDBY_WAKE_SOURCE_LVD2	Low Voltage Detection 2 interrupt.
LPM_STANDBY_WAKE_SOURCE_VBATT	VBATT Monitor interrupt.

LPM_STANDBY_WAKE_SOURCE_ACMPS0	Analog Comparator High-speed 0 interrupt.
LPM_STANDBY_WAKE_SOURCE_ACMPLP0	Analog Comparator Low-speed 0 interrupt.
LPM_STANDBY_WAKE_SOURCE_RTCALM	RTC Alarm interrupt.
LPM_STANDBY_WAKE_SOURCE_RTCPRD	RTC Period interrupt.
LPM_STANDBY_WAKE_SOURCE_USBHS	USB High-speed interrupt.
LPM_STANDBY_WAKE_SOURCE_USBFS	USB Full-speed interrupt.
LPM_STANDBY_WAKE_SOURCE_AGT1UD	AGT1 underflow interrupt.
LPM_STANDBY_WAKE_SOURCE_AGT1CA	AGT1 compare match A interrupt.
LPM_STANDBY_WAKE_SOURCE_AGT1CB	AGT1 compare match B interrupt.
LPM_STANDBY_WAKE_SOURCE_IIC0	I2C 0 interrupt.

◆ **lpm_io_port_t**

enum <code>lpm_io_port_t</code>	
I/O port state after Deep Software Standby mode	
Enumerator	
LPM_IO_PORT_RESET	When the Deep Software Standby mode is canceled, the I/O ports are in the reset state
LPM_IO_PORT_NO_CHANGE	When the Deep Software Standby mode is canceled, the I/O ports are in the same state as in the Deep Software Standby mode

◆ **lpm_power_supply_t**

enum lpm_power_supply_t	
Power supply control	
Enumerator	
LPM_POWER_SUPPLY_DEEPCUT0	Power to the standby RAM, Low-speed on-chip oscillator, AGTn, and USBFS/HS resume detecting unit is supplied in deep software standby mode
LPM_POWER_SUPPLY_DEEPCUT1	Power to the standby RAM, Low-speed on-chip oscillator, AGTn, and USBFS/HS resume detecting unit is not supplied in deep software standby mode
LPM_POWER_SUPPLY_DEEPCUT3	Power to the standby RAM, Low-speed on-chip oscillator, AGTn, and USBFS/HS resume detecting unit is not supplied in deep software standby mode. In addition, LVD is disabled and the low power function in a poweron reset circuit is enabled

◆ **lpm_deep_standby_cancel_edge_t**

enum lpm_deep_standby_cancel_edge_t	
Deep Standby Interrupt Edge	
Enumerator	
LPM_DEEP_STANDBY_CANCEL_SOURCE_EDGE_NONE	No options for a deep standby cancel source.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ0_RISING	IRQ0-DS Pin Rising Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ0_FALLING	IRQ0-DS Pin Falling Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ1_RISING	IRQ1-DS Pin Rising Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ1_FALLING	IRQ1-DS Pin Falling Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ2_RISING	IRQ2-DS Pin Rising Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ2_FALLING	IRQ2-DS Pin Falling Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ3_RISING	IRQ3-DS Pin Rising Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ3_FALLING	IRQ3-DS Pin Falling Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ4_RISING	IRQ4-DS Pin Rising Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ4_FALLING	IRQ4-DS Pin Falling Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ5_RISING	IRQ5-DS Pin Rising Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ5_FALLING	IRQ5-DS Pin Falling Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ6_RISING	IRQ6-DS Pin Rising Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ6_FALLING	IRQ6-DS Pin Falling Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ7_RISING	IRQ7-DS Pin Rising Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ7_FALLING	IRQ7-DS Pin Falling Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ8_RISING	IRQ8-DS Pin Rising Edge.

LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ8_FALLING	IRQ8-DS Pin Falling Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ9_RISING	IRQ9-DS Pin Rising Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ9_FALLING	IRQ9-DS Pin Falling Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ10_RISING	IRQ10-DS Pin Rising Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ10_FALLING	IRQ10-DS Pin Falling Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ11_RISING	IRQ11-DS Pin Rising Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ11_FALLING	IRQ11-DS Pin Falling Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ12_RISING	IRQ12-DS Pin Rising Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ12_FALLING	IRQ12-DS Pin Falling Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ13_RISING	IRQ13-DS Pin Rising Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ13_FALLING	IRQ13-DS Pin Falling Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ14_RISING	IRQ14-DS Pin Rising Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ14_FALLING	IRQ14-DS Pin Falling Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_LVD1_RISING	LVD1 Rising Slope.
LPM_DEEP_STANDBY_CANCEL_SOURCE_LVD1_FALLING	LVD1 Falling Slope.
LPM_DEEP_STANDBY_CANCEL_SOURCE_LVD2_RISING	LVD2 Rising Slope.
LPM_DEEP_STANDBY_CANCEL_SOURCE_LVD2_FALLING	LVD2 Falling Slope.
LPM_DEEP_STANDBY_CANCEL_SOURCE_NMI_RISING	NMI Pin Rising Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_NMI_FALLING	NMI Pin Falling Edge.

◆ **lpm_deep_standby_cancel_source_t**

enum <code>lpm_deep_standby_cancel_source_t</code>	
Deep Standby cancel sources	
Enumerator	
<code>LPM_DEEP_STANDBY_CANCEL_SOURCE_RESET_ONLY</code>	Cancel deep standby only by reset.
<code>LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ0</code>	IRQ0.
<code>LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ1</code>	IRQ1.
<code>LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ2</code>	IRQ2.
<code>LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ3</code>	IRQ3.
<code>LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ4</code>	IRQ4.
<code>LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ5</code>	IRQ5.
<code>LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ6</code>	IRQ6.
<code>LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ7</code>	IRQ7.
<code>LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ8</code>	IRQ8.
<code>LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ9</code>	IRQ9.
<code>LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ10</code>	IRQ10.
<code>LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ11</code>	IRQ11.
<code>LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ12</code>	IRQ12.
<code>LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ13</code>	IRQ13.
<code>LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ14</code>	IRQ14.
<code>LPM_DEEP_STANDBY_CANCEL_SOURCE_LVD1</code>	LVD1.
<code>LPM_DEEP_STANDBY_CANCEL_SOURCE_LVD2</code>	LVD2.
<code>LPM_DEEP_STANDBY_CANCEL_SOURCE_RTC_INTERVAL</code>	RTC Interval Interrupt.
<code>LPM_DEEP_STANDBY_CANCEL_SOURCE_RTC_ALARM</code>	RTC Alarm Interrupt.

LPM_DEEP_STANDBY_CANCEL_SOURCE_NMI	NMI.
LPM_DEEP_STANDBY_CANCEL_SOURCE_USBFS	USBFS Suspend/Resume.
LPM_DEEP_STANDBY_CANCEL_SOURCE_USBHS	USBHS Suspend/Resume.
LPM_DEEP_STANDBY_CANCEL_SOURCE_AGT1	AGT1 Underflow.

◆ lpm_output_port_enable_t

enum lpm_output_port_enable_t	
Output port enable	
Enumerator	
LPM_OUTPUT_PORT_ENABLE_HIGH_IMPEDANCE	0: In Software Standby Mode or Deep Software Standby Mode, the address output pins, data output pins, and other bus control signal output pins are set to the high-impedance state. In Snooze, the status of the address bus and bus control signals are same as before entering Software Standby Mode.
LPM_OUTPUT_PORT_ENABLE_RETAIN	1: In Software Standby Mode, the address output pins, data output pins, and other bus control signal output pins retain the output state.

4.3.22 Low Voltage Detection Interface

Interfaces

Detailed Description

Interface for Low Voltage Detection.

Summary

The LVD driver provides functions for configuring the LVD voltage monitors and detectors.

Implemented by:

- [Low Voltage Detection \(r_lvd\)](#)

Data Structures

struct	lvd_status_t
struct	lvd_callback_args_t
struct	lvd_cfg_t
struct	lvd_api_t
struct	lvd_instance_t

Macros

#define	LVD_API_VERSION_MAJOR

Typedefs

typedef void	lvd_ctrl_t

Enumerations

enum	lvd_threshold_t
enum	lvd_response_t
enum	lvd_voltage_slope_t
enum	lvd_sample_clock_t
enum	lvd_negation_delay_t
enum	lvd_threshold_crossing_t
enum	lvd_current_state_t

Data Structure Documentation

◆ [lvd_status_t](#)

struct lvd_status_t		
Current state of a voltage monitor.		
Data Fields		
lvd_threshold_crossing_t	crossing_detected	Threshold crossing detection (latched)
lvd_current_state_t	current_state	Instantaneous status of monitored voltage (above or below threshold)

◆ [lvd_callback_args_t](#)

struct lvd_callback_args_t

LVD callback parameter definition

Data Fields		
uint32_t	monitor_number	Monitor number.
lvd_current_state_t	current_state	Current state of the voltage monitor.
void const *	p_context	Placeholder for user data.

◆ [lvd_cfg_t](#)struct [lvd_cfg_t](#)

LVD configuration structure

Data Fields

uint32_t	monitor_number
lvd_threshold_t	voltage_threshold
lvd_response_t	detection_response
lvd_voltage_slope_t	voltage_slope
lvd_negation_delay_t	negation_delay
lvd_sample_clock_t	sample_clock_divisor
IRQn_Type	irq
uint8_t	monitor_ipl
void(*	p_callback)(lvd_callback_args_t *p_args)
void const *	p_context
void const *	p_extend

Field Documentation

◆ **monitor_number**

uint32_t lvd_cfg_t::monitor_number

Monitor number, 1, 2, ...

◆ **voltage_threshold**

lvd_threshold_t lvd_cfg_t::voltage_threshold

Threshold for out of range voltage detection

◆ **detection_response**

lvd_response_t lvd_cfg_t::detection_response

Response on detecting a threshold crossing

◆ **voltage_slope**

lvd_voltage_slope_t lvd_cfg_t::voltage_slope

Direction of voltage crossing that will trigger a detection (Rising Edge, Falling Edge, Both).

◆ **negation_delay**

lvd_negation_delay_t lvd_cfg_t::negation_delay

Negation of LVD signal follows reset or voltage in range

◆ **sample_clock_divisor**

lvd_sample_clock_t lvd_cfg_t::sample_clock_divisor

Sample clock divider, use LVD_SAMPLE_CLOCK_DISABLED to disable digital filtering

◆ **irq**

IRQn_Type lvd_cfg_t::irq

Interrupt number.

◆ **monitor_ipl**

uint8_t lvd_cfg_t::monitor_ipl

Interrupt priority level.

◆ **p_callback**

void(* lvd_cfg_t::p_callback) (lvd_callback_args_t *p_args)

User function to be called from interrupt

◆ **p_context**

void const* lvd_cfg_t::p_context

Placeholder for user data. Passed to the user callback in

◆ **p_extend**

void const* lvd_cfg_t::p_extend

Extension parameter for hardware specific settings

◆ **lvd_api_t**

struct lvd_api_t

LVD driver API structure. LVD driver functions implemented at the HAL layer will adhere to this API.

Data Fieldsfsp_err_t(* [open](#))(lvd_ctrl_t *const p_ctrl, lvd_cfg_t const *const p_cfg)fsp_err_t(* [statusGet](#))(lvd_ctrl_t *const p_ctrl, lvd_status_t *p_lvd_status)fsp_err_t(* [statusClear](#))(lvd_ctrl_t *const p_ctrl)fsp_err_t(* [close](#))(lvd_ctrl_t *const p_ctrl)fsp_err_t(* [versionGet](#))(fsp_version_t *const p_version)**Field Documentation**◆ **open**

fsp_err_t(* lvd_api_t::open) (lvd_ctrl_t *const p_ctrl, lvd_cfg_t const *const p_cfg)

Initializes a low voltage detection driver according to the passed-in configuration structure.

Implemented as◦ [R_LVD_Open\(\)](#)**Parameters**

[in]	p_ctrl	Pointer to control structure for the driver instance
[in]	p_cfg	Pointer to the configuration structure for the driver instance

◆ **statusGet**

```
fsp_err_t(* lvd_api_t::statusGet) (lvd_ctrl_t *const p_ctrl, lvd_status_t *p_lvd_status)
```

Get the current state of the monitor, (threshold crossing detected, voltage currently above or below threshold). Must be used if the peripheral was initialized with `lvd_response_t` set to `LVD_RESPONSE_NONE`.

Implemented as

◦ [R_LVD_StatusGet\(\)](#)

Parameters

[in]	p_ctrl	Pointer to the control structure for the driver instance
[in,out]	p_lvd_status	Pointer to a lvd_status_t structure

◆ **statusClear**

```
fsp_err_t(* lvd_api_t::statusClear) (lvd_ctrl_t *const p_ctrl)
```

Clears the latched status of the monitor. Must be used if the peripheral was initialized with `lvd_response_t` set to `LVD_RESPONSE_NONE`.

Implemented as

◦ [R_LVD_StatusClear\(\)](#)

Parameters

[in]	p_ctrl	Pointer to the control structure for the driver instance

◆ **close**

```
fsp_err_t(* lvd_api_t::close) (lvd_ctrl_t *const p_ctrl)
```

Disables the LVD peripheral. Closes the driver instance.

Implemented as

◦ [R_LVD_Close\(\)](#)

Parameters

[in]	p_ctrl	Pointer to the control structure for the driver instance

◆ **versionGet**

```
fsp_err_t(* lvd_api_t::versionGet) (fsp_version_t *const p_version)
```

Returns the LVD driver version based on compile time macros.

Implemented as

◦ [R_LVD_VersionGet\(\)](#)

Parameters

[in,out]	p_version	Pointer to version structure

◆ **lvd_instance_t**

```
struct lvd_instance_t
```

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields

lvd_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
lvd_cfg_t const *	p_cfg	Pointer to the configuration structure for this interface instance.
lvd_api_t const *	p_api	Pointer to the API structure for this interface instance.

Macro Definition Documentation◆ **LVD_API_VERSION_MAJOR**

```
#define LVD_API_VERSION_MAJOR
```

Register definitions, common services, and error codes.

Typedef Documentation◆ **lvd_ctrl_t**

```
typedef void lvd_ctrl_t
```

LVD control block. Allocate an instance specific control block to pass into the LVD API calls.

Implemented as

◦ [lvd_instance_ctrl_t](#)

Enumeration Type Documentation

◆ **lvd_threshold_t**enum **lvd_threshold_t**

Voltage detection level The thresholds supported by each MCU are in the MCU User's Manual as well as in the r_lvd module description on the stack tab of the RA project.

Enumerator

LVD_THRESHOLD_MONITOR_1_LEVEL_4_29V	4.29V
LVD_THRESHOLD_MONITOR_1_LEVEL_4_14V	4.14V
LVD_THRESHOLD_MONITOR_1_LEVEL_4_02V	4.02V
LVD_THRESHOLD_MONITOR_1_LEVEL_3_84V	3.84V
LVD_THRESHOLD_MONITOR_1_LEVEL_3_10V	3.10V
LVD_THRESHOLD_MONITOR_1_LEVEL_3_00V	3.00V
LVD_THRESHOLD_MONITOR_1_LEVEL_2_90V	2.90V
LVD_THRESHOLD_MONITOR_1_LEVEL_2_79V	2.79V
LVD_THRESHOLD_MONITOR_1_LEVEL_2_68V	2.68V
LVD_THRESHOLD_MONITOR_1_LEVEL_2_58V	2.58V
LVD_THRESHOLD_MONITOR_1_LEVEL_2_48V	2.48V
LVD_THRESHOLD_MONITOR_1_LEVEL_2_20V	2.20V
LVD_THRESHOLD_MONITOR_1_LEVEL_1_96V	1.96V
LVD_THRESHOLD_MONITOR_1_LEVEL_1_86V	1.86V
LVD_THRESHOLD_MONITOR_1_LEVEL_1_75V	1.75V
LVD_THRESHOLD_MONITOR_1_LEVEL_1_65V	1.65V
LVD_THRESHOLD_MONITOR_1_LEVEL_2_99V	2.99V
LVD_THRESHOLD_MONITOR_1_LEVEL_2_92V	2.92V
LVD_THRESHOLD_MONITOR_1_LEVEL_2_85V	2.85V
LVD_THRESHOLD_MONITOR_2_LEVEL_4_29V	4.29V
LVD_THRESHOLD_MONITOR_2_LEVEL_4_14V	4.14V

LVD_THRESHOLD_MONITOR_2_LEVEL_4_02V	4.02V
LVD_THRESHOLD_MONITOR_2_LEVEL_3_84V	3.84V
LVD_THRESHOLD_MONITOR_2_LEVEL_2_99V	2.99V
LVD_THRESHOLD_MONITOR_2_LEVEL_2_92V	2.92V
LVD_THRESHOLD_MONITOR_2_LEVEL_2_85V	2.85V

◆ lvd_response_t

enum lvd_response_t	
Response types for handling threshold crossing event.	
Enumerator	
LVD_RESPONSE_NMI	Non-maskable interrupt.
LVD_RESPONSE_INTERRUPT	Maskable interrupt.
LVD_RESPONSE_RESET	Reset.
LVD_RESPONSE_NONE	No response, status must be requested via statusGet function.

◆ lvd_voltage_slope_t

enum lvd_voltage_slope_t	
The direction from which Vcc must cross the threshold to trigger a detection (rising, falling, or both).	
Enumerator	
LVD_VOLTAGE_SLOPE_RISING	When VCC $\geq$ Vdet2 (rise) is detected.
LVD_VOLTAGE_SLOPE_FALLING	When VCC $<$ Vdet2 (drop) is detected.
LVD_VOLTAGE_SLOPE_BOTH	When drop and rise are detected.

◆ **lvd_sample_clock_t**

enum <code>lvd_sample_clock_t</code>	
Sample clock divider, use LVD_SAMPLE_CLOCK_DISABLED to disable digital filtering	
Enumerator	
LVD_SAMPLE_CLOCK_LOCO_DIV_2	Digital filter sample clock is LOCO divided by 2.
LVD_SAMPLE_CLOCK_LOCO_DIV_4	Digital filter sample clock is LOCO divided by 4.
LVD_SAMPLE_CLOCK_LOCO_DIV_8	Digital filter sample clock is LOCO divided by 8.
LVD_SAMPLE_CLOCK_LOCO_DIV_16	Digital filter sample clock is LOCO divided by 16.
LVD_SAMPLE_CLOCK_DISABLED	Digital filter is disabled.

◆ **lvd_negation_delay_t**

enum <code>lvd_negation_delay_t</code>	
Negation delay of LVD reset signal follows reset or voltage in range	
Enumerator	
LVD_NEGATION_DELAY_FROM_VOLTAGE	Negation follows a stabilization time (tLVDn) after VCC > Vdet1 is detected. If a transition to software standby or deep software standby is to be made, the only possible value for the RN bit is LVD_NEGATION_DELAY_FROM_VOLTAGE
LVD_NEGATION_DELAY_FROM_RESET	Negation follows a stabilization time (tLVDn) after assertion of the LVDn reset. If a transition to software standby or deep software standby is to be made, the only possible value for the RN bit is LVD_NEGATION_DELAY_FROM_VOLTAGE

◆ **lvd_threshold_crossing_t**

enum <code>lvd_threshold_crossing_t</code>	
Threshold crossing detection (latched)	
Enumerator	
LVD_THRESHOLD_CROSSING_NOT_DETECTED	Threshold crossing has not been detected.
LVD_THRESHOLD_CROSSING_DETECTED	Threshold crossing has been detected.

◆ **lvd_current_state_t**

enum <code>lvd_current_state_t</code>	
Instantaneous status of VCC (above or below threshold)	
Enumerator	
<code>LVD_CURRENT_STATE_BELOW_THRESHOLD</code>	VCC < threshold.
<code>LVD_CURRENT_STATE_ABOVE_THRESHOLD</code>	VCC >= threshold or monitor is disabled.

4.3.23 RTC Interface[Interfaces](#)**Detailed Description**

Interface for accessing the Realtime Clock.

Summary

The RTC Interface is for configuring Real Time Clock (RTC) functionality including alarm, periodic notification and error adjustment.

The Real Time Clock Interface can be implemented by:

- [Realtime Clock \(`r\_rtc`\)](#)

Data Structures

struct [rtc_callback_args_t](#)

struct [rtc_error_adjustment_cfg_t](#)

struct [rtc_alarm_time_t](#)

struct [rtc_info_t](#)

struct [rtc_cfg_t](#)

struct [rtc_api_t](#)

struct [rtc_instance_t](#)

Macros

```
#define RTC_API_VERSION_MAJOR
```

Typedefs

```
typedef struct tm rtc_time_t
```

```
typedef void rtc_ctrl_t
```

Enumerations

```
enum rtc_event_t
```

```
enum rtc_clock_source_t
```

```
enum rtc_status_t
```

```
enum rtc_error_adjustment_t
```

```
enum rtc_error_adjustment_mode_t
```

```
enum rtc_error_adjustment_period_t
```

```
enum rtc_periodic_irq_select_t
```

Data Structure Documentation

◆ rtc_callback_args_t

struct rtc_callback_args_t		
Callback function parameter data		
Data Fields		
rtc_event_t	event	The event can be used to identify what caused the callback (compare match or error).
void const *	p_context	Placeholder for user data.

◆ rtc_error_adjustment_cfg_t

struct rtc_error_adjustment_cfg_t		
Time error adjustment value configuration		
Data Fields		
rtc_error_adjustment_mode_t	adjustment_mode	Automatic Adjustment Enable/Disable.
rtc_error_adjustment_period_t	adjustment_period	Error Adjustment period.
rtc_error_adjustment_t	adjustment_type	Time error adjustment setting.
uint32_t	adjustment_value	Value of the prescaler for error

adjustment.

◆ **rtc_alarm_time_t**

struct rtc_alarm_time_t

Alarm time setting structure

Data Fields

rtc_time_t	time	Time structure.
bool	sec_match	Enable the alarm based on a match of the seconds field.
bool	min_match	Enable the alarm based on a match of the minutes field.
bool	hour_match	Enable the alarm based on a match of the hours field.
bool	mday_match	Enable the alarm based on a match of the days field.
bool	mon_match	Enable the alarm based on a match of the months field.
bool	year_match	Enable the alarm based on a match of the years field.
bool	dayofweek_match	Enable the alarm based on a match of the dayofweek field.

◆ **rtc_info_t**

struct rtc_info_t

RTC Information Structure for information returned by infoGet()

Data Fields

rtc_clock_source_t	clock_source	Clock source for the RTC block.
rtc_status_t	status	RTC run status.

◆ **rtc_cfg_t**

struct rtc_cfg_t

User configuration structure, used in open function

Data Fields

rtc_clock_source_t	clock_source
	Clock source for the RTC block.
uint32_t	freq_compare_value_loco
	The frequency comparison value for LOCO.

<code>rtc_error_adjustment_cfg_t</code> const *const	<code>p_err_cfg</code>
	Pointer to Error Adjustment configuration.
<code>uint8_t</code>	<code>alarm_ipl</code>
	Alarm interrupt priority.
<code>IRQn_Type</code>	<code>alarm_irq</code>
	Alarm interrupt vector.
<code>uint8_t</code>	<code>periodic_ipl</code>
	Periodic interrupt priority.
<code>IRQn_Type</code>	<code>periodic_irq</code>
	Periodic interrupt vector.
<code>uint8_t</code>	<code>carry_ipl</code>
	Carry interrupt priority.
<code>IRQn_Type</code>	<code>carry_irq</code>
	Carry interrupt vector.
<code>void(*</code>	<code>p_callback</code>)(rtc_callback_args_t *p_args)
	Called from the ISR.
<code>void const *</code>	<code>p_context</code>
	User defined context passed into callback function.

void const *	p_extend
	RTC hardware dependant configuration.

◆ **rtc_api_t**

struct rtc_api_t

RTC driver structure. General RTC functions implemented at the HAL layer follow this API.

Data Fields

fsp_err_t (*	open)(rtc_ctrl_t *const p_ctrl, rtc_cfg_t const *const p_cfg)
fsp_err_t (*	close)(rtc_ctrl_t *const p_ctrl)
fsp_err_t (*	calendarTimeSet)(rtc_ctrl_t *const p_ctrl, rtc_time_t *const p_time)
fsp_err_t (*	calendarTimeGet)(rtc_ctrl_t *const p_ctrl, rtc_time_t *const p_time)
fsp_err_t (*	calendarAlarmSet)(rtc_ctrl_t *const p_ctrl, rtc_alarm_time_t *const p_alarm)
fsp_err_t (*	calendarAlarmGet)(rtc_ctrl_t *const p_ctrl, rtc_alarm_time_t *const p_alarm)
fsp_err_t (*	periodicIrqRateSet)(rtc_ctrl_t *const p_ctrl, rtc_periodic_irq_select_t const rate)
fsp_err_t (*	errorAdjustmentSet)(rtc_ctrl_t *const p_ctrl, rtc_error_adjustment_cfg_t const *const err_adj_cfg)
fsp_err_t (*	infoGet)(rtc_ctrl_t *const p_ctrl, rtc_info_t *const p_rtc_info)
fsp_err_t (*	versionGet)(fsp_version_t *const version)

Field Documentation

◆ **open**

```
fsp_err_t(* rtc_api_t::open) (rtc_ctrl_t *const p_ctrl, rtc_cfg_t const *const p_cfg)
```

Open the RTC driver.

Implemented as

◦ [R_RTC_Open\(\)](#)

Parameters

[in]	p_ctrl	Pointer to RTC device handle
[in]	p_cfg	Pointer to the configuration structure

◆ **close**

```
fsp_err_t(* rtc_api_t::close) (rtc_ctrl_t *const p_ctrl)
```

Close the RTC driver.

Implemented as

◦ [R_RTC_Close\(\)](#)

Parameters

[in]	p_ctrl	Pointer to RTC device handle.

◆ **calendarTimeSet**

```
fsp_err_t(* rtc_api_t::calendarTimeSet) (rtc_ctrl_t *const p_ctrl, rtc_time_t *const p_time)
```

Set the calendar time and start the calendar counter

Implemented as

◦ [R_RTC_CalendarTimeSet\(\)](#)

Parameters

[in]	p_ctrl	Pointer to RTC device handle
[in]	p_time	Pointer to a time structure that contains the time to set
[in]	clock_start	Flag that starts the clock right after it is set

◆ **calendarTimeGet**

```
fsp_err_t(* rtc_api_t::calendarTimeGet) (rtc_ctrl_t *const p_ctrl, rtc_time_t *const p_time)
```

Get the calendar time.

Implemented as

◦ [R_RTC_CalendarTimeGet\(\)](#)

Parameters

[in]	p_ctrl	Pointer to RTC device handle
[out]	p_time	Pointer to a time structure that contains the time to get

◆ **calendarAlarmSet**

```
fsp_err_t(* rtc_api_t::calendarAlarmSet) (rtc_ctrl_t *const p_ctrl, rtc_alarm_time_t *const p_alarm)
```

Set the calendar alarm time and enable the alarm interrupt.

Implemented as

◦ [R_RTC_CalendarAlarmSet\(\)](#)

Parameters

[in]	p_ctrl	Pointer to RTC device handle
[in]	p_alarm	Pointer to an alarm structure that contains the alarm time to set
[in]	irq_enable_flag	Enable the ALARM irq if set

◆ **calendarAlarmGet**

```
fsp_err_t(* rtc_api_t::calendarAlarmGet) (rtc_ctrl_t *const p_ctrl, rtc_alarm_time_t *const p_alarm)
```

Get the calendar alarm time.

Implemented as

◦ [R_RTC_CalendarAlarmGet\(\)](#)

Parameters

[in]	p_ctrl	Pointer to RTC device handle
[out]	p_alarm	Pointer to an alarm structure to fill up with the alarm time

◆ **periodicIrqRateSet**

```
fsp_err_t(* rtc_api_t::periodicIrqRateSet) (rtc_ctrl_t *const p_ctrl, rtc_periodic_irq_select_t const rate)
```

Set the periodic irq rate

Implemented as

◦ [R_RTC_PeriodicIrqRateSet\(\)](#)

Parameters

[in]	p_ctrl	Pointer to RTC device handle
[in]	rate	Rate of periodic interrupts

◆ **errorAdjustmentSet**

```
fsp_err_t(* rtc_api_t::errorAdjustmentSet) (rtc_ctrl_t *const p_ctrl, rtc_error_adjustment_cfg_t const *const err_adj_cfg)
```

Set time error adjustment.

Implemented as

[R_RTC_ErrorAdjustmentSet\(\)](#)

Parameters

[in]	p_ctrl	Pointer to control handle structure
[in]	err_adj_cfg	Pointer to the Error Adjustment Config

◆ **infoGet**

```
fsp_err_t(* rtc_api_t::infoGet) (rtc_ctrl_t *const p_ctrl, rtc_info_t *const p_rtc_info)
```

Return the currently configure clock source for the RTC

Implemented as

◦ [R_RTC_InfoGet\(\)](#)

Parameters

[in]	p_ctrl	Pointer to control handle structure
[out]	p_rtc_info	Pointer to RTC information structure

◆ **versionGet**

```
fsp_err_t(* rtc_api_t::versionGet) (fsp_version_t *const version)
```

Gets version and stores it in provided pointer p_version.

Implemented as

- `R_RTC_VersionGet()`

Parameters

[out]	p_version	Code and API version used

◆ **rtc_instance_t**

```
struct rtc_instance_t
```

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields

<code>rtc_ctrl_t *</code>	p_ctrl	Pointer to the control structure for this instance.
<code>rtc_cfg_t const *</code>	p_cfg	Pointer to the configuration structure for this instance.
<code>rtc_api_t const *</code>	p_api	Pointer to the API structure for this instance.

Macro Definition Documentation◆ **RTC_API_VERSION_MAJOR**

```
#define RTC_API_VERSION_MAJOR
```

Use of time structure, tm

Typedef Documentation◆ **rtc_time_t**

```
typedef struct tm rtc_time_t
```

Date and time structure defined in C standard library <time.h>

◆ **rtc_ctrl_t**typedef void [rtc_ctrl_t](#)

RTC control block. Allocate an instance specific control block to pass into the RTC API calls.

Implemented as

- [rtc_instance_ctrl_t](#)

Enumeration Type Documentation◆ **rtc_event_t**enum [rtc_event_t](#)

Events that can trigger a callback function

Enumerator

RTC_EVENT_ALARM_IRQ

Real Time Clock ALARM IRQ.

RTC_EVENT_PERIODIC_IRQ

Real Time Clock PERIODIC IRQ.

◆ **rtc_clock_source_t**enum [rtc_clock_source_t](#)

Clock source for the RTC block

Enumerator

RTC_CLOCK_SOURCE_SUBCLK

Sub-clock oscillator.

RTC_CLOCK_SOURCE_LOCO

Low power On Chip Oscillator.

◆ **rtc_status_t**enum [rtc_status_t](#)

RTC run state

Enumerator

RTC_STATUS_STOPPED

RTC counter is stopped.

RTC_STATUS_RUNNING

RTC counter is running.

◆ **rtc_error_adjustment_t**

enum <code>rtc_error_adjustment_t</code>	
Time error adjustment settings	
Enumerator	
<code>RTC_ERROR_ADJUSTMENT_NONE</code>	Adjustment is not performed.
<code>RTC_ERROR_ADJUSTMENT_ADD_PRESCALER</code>	Adjustment is performed by the addition to the prescaler.
<code>RTC_ERROR_ADJUSTMENT_SUBTRACT_PRESCALER</code>	Adjustment is performed by the subtraction from the prescaler.

◆ **rtc_error_adjustment_mode_t**

enum <code>rtc_error_adjustment_mode_t</code>	
Time error adjustment mode settings	
Enumerator	
<code>RTC_ERROR_ADJUSTMENT_MODE_MANUAL</code>	Adjustment mode is set to manual.
<code>RTC_ERROR_ADJUSTMENT_MODE_AUTOMATIC</code>	Adjustment mode is set to automatic.

◆ **rtc_error_adjustment_period_t**

enum <code>rtc_error_adjustment_period_t</code>	
Time error adjustment period settings	
Enumerator	
<code>RTC_ERROR_ADJUSTMENT_PERIOD_1_MINUTE</code>	Adjustment period is set to every one minute.
<code>RTC_ERROR_ADJUSTMENT_PERIOD_10_SECOND</code>	Adjustment period is set to every ten second.
<code>RTC_ERROR_ADJUSTMENT_PERIOD_NONE</code>	Adjustment period not supported in manual mode.

◆ **rtc_periodic_irq_select_t**

enum <code>rtc_periodic_irq_select_t</code>	
Periodic Interrupt select	
Enumerator	
<code>RTC_PERIODIC_IRQ_SELECT_1_DIV_BY_256_SECOND</code>	A periodic irq is generated every 1/256 second.
<code>RTC_PERIODIC_IRQ_SELECT_1_DIV_BY_128_SECOND</code>	A periodic irq is generated every 1/128 second.
<code>RTC_PERIODIC_IRQ_SELECT_1_DIV_BY_64_SECOND</code>	A periodic irq is generated every 1/64 second.
<code>RTC_PERIODIC_IRQ_SELECT_1_DIV_BY_32_SECOND</code>	A periodic irq is generated every 1/32 second.
<code>RTC_PERIODIC_IRQ_SELECT_1_DIV_BY_16_SECOND</code>	A periodic irq is generated every 1/16 second.
<code>RTC_PERIODIC_IRQ_SELECT_1_DIV_BY_8_SECOND</code>	A periodic irq is generated every 1/8 second.
<code>RTC_PERIODIC_IRQ_SELECT_1_DIV_BY_4_SECOND</code>	A periodic irq is generated every 1/4 second.
<code>RTC_PERIODIC_IRQ_SELECT_1_DIV_BY_2_SECOND</code>	A periodic irq is generated every 1/2 second.
<code>RTC_PERIODIC_IRQ_SELECT_1_SECOND</code>	A periodic irq is generated every 1 second.
<code>RTC_PERIODIC_IRQ_SELECT_2_SECOND</code>	A periodic irq is generated every 2 seconds.

4.3.24 SD/MMC Interface[Interfaces](#)**Detailed Description**

Interface for accessing SD, eMMC, and SDIO devices.

Summary

The `r_sdhi` interface provides standard SD and eMMC media functionality. This interface also supports SDIO.

The SD/MMC interface is implemented by:

• SD/MMC Host Interface (r_sdhi)

Data Structures

	struct	sdmmc_status_t
	struct	sdmmc_device_t
	struct	sdmmc_callback_args_t
	struct	sdmmc_cfg_t
	struct	sdmmc_api_t
	struct	sdmmc_instance_t

Typedefs

	typedef void	sdmmc_ctrl_t

Enumerations

	enum	sdmmc_card_type_t
	enum	sdmmc_bus_width_t
	enum	sdmmc_io_transfer_mode_t
	enum	sdmmc_io_address_mode_t
	enum	sdmmc_io_write_mode_t
	enum	sdmmc_event_t
	enum	sdmmc_card_detect_t
	enum	sdmmc_write_protect_t
	enum	sdmmc_r1_state_t

Data Structure Documentation

◆ [sdmmc_status_t](#)

struct sdmmc_status_t		
Current status.		
Data Fields		
bool	initialized	False if card was removed (only applies if MCU supports card)

		detection and SDnCD pin is connected), true otherwise. If ready is false, call sdmmc_api_t::medialnit to reinitialize it
bool	transfer_in_progress	true = Card is busy
bool	card_inserted	Card detect status, true if card detect is not used.

◆ **sdmmc_device_t**

struct sdmmc_device_t		
Information obtained from the media device.		
Data Fields		
sdmmc_card_type_t	card_type	SD, eMMC, or SDIO.
bool	write_protected	true = Card is write protected
uint32_t	clock_rate	Current clock rate.
uint32_t	sector_count	Sector count.
uint32_t	erase_sector_count	Minimum erasable unit (in 512 byte sectors)

◆ **sdmmc_callback_args_t**

struct sdmmc_callback_args_t		
Callback function parameter data		
Data Fields		
sdmmc_event_t	event	The event can be used to identify what caused the callback.
sdmmc_response_t	response	Response from card, only valid if SDMMC_EVENT_RESPONSE is set in event.
void const *	p_context	Placeholder for user data.

◆ **sdmmc_cfg_t**

struct sdmmc_cfg_t	
SD/MMC Configuration	
Data Fields	
uint8_t	channel
	Channel of SD/MMC host interface.

<code>sdmmc_bus_width_t</code>	<code>bus_width</code>
	Device bus width is 1, 4 or 8 bits wide.
<code>transfer_instance_t const *</code>	<code>p_lower_lvl_transfer</code>
	Transfer instance used to move data with DMA or DTC.
<code>void(*</code>	<code>p_callback)(sdmmc_callback_args_t *p_args)</code>
	Pointer to callback function.
<code>void const *</code>	<code>p_context</code>
	User defined context passed into callback function.
<code>void const *</code>	<code>p_extend</code>
	SD/MMC hardware dependent configuration.
<code>uint32_t</code>	<code>block_size</code>
<code>sdmmc_card_detect_t</code>	<code>card_detect</code>
<code>sdmmc_write_protect_t</code>	<code>write_protect</code>
<code>IRQn_Type</code>	<code>access_irq</code>
	Access IRQ number.
<code>IRQn_Type</code>	<code>sdio_irq</code>
	SDIO IRQ number.
<code>IRQn_Type</code>	<code>card_irq</code>
	Card IRQ number.

IRQn_Type	dma_req_irq
	DMA request IRQ number.
uint8_t	access_ipl
	Access interrupt priority.
uint8_t	sdio_ipl
	SDIO interrupt priority.
uint8_t	card_ipl
	Card interrupt priority.
uint8_t	dma_req_ipl
	DMA request interrupt priority.

Field Documentation

◆ block_size

[uint32_t sdmmc_cfg_t::block_size](#)

Block size in bytes. Block size must be 512 bytes for SD cards and eMMC devices. Block size can be 1-512 bytes for SDIO.

◆ card_detect

[sdmmc_card_detect_t sdmmc_cfg_t::card_detect](#)

Whether or not card detection is used.

◆ write_protect

[sdmmc_write_protect_t sdmmc_cfg_t::write_protect](#)

Select whether or not to use the write protect pin. Select Not Used if the MCU or device does not have a write protect pin.

◆ sdmmc_api_t

[struct sdmmc_api_t](#)

SD/MMC functions implemented at the HAL layer API.

Data Fields

<code>fsp_err_t(*</code>	<code>open)(sdmmc_ctrl_t *const p_ctrl, sdmmc_cfg_t const *const p_cfg)</code>
<code>fsp_err_t(*</code>	<code>mediaInit)(sdmmc_ctrl_t *const p_ctrl, sdmmc_device_t *const p_device)</code>
<code>fsp_err_t(*</code>	<code>read)(sdmmc_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const start_sector, uint32_t const sector_count)</code>
<code>fsp_err_t(*</code>	<code>write)(sdmmc_ctrl_t *const p_ctrl, uint8_t const *const p_source, uint32_t const start_sector, uint32_t const sector_count)</code>
<code>fsp_err_t(*</code>	<code>readlo)(sdmmc_ctrl_t *const p_ctrl, uint8_t *const p_data, uint32_t const function, uint32_t const address)</code>
<code>fsp_err_t(*</code>	<code>writelow)(sdmmc_ctrl_t *const p_ctrl, uint8_t *const p_data, uint32_t const function, uint32_t const address, sdmmc_io_write_mode_t const read_after_write)</code>
<code>fsp_err_t(*</code>	<code>readloExt)(sdmmc_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const function, uint32_t const address, uint32_t *const count, sdmmc_io_transfer_mode_t transfer_mode, sdmmc_io_address_mode_t address_mode)</code>
<code>fsp_err_t(*</code>	<code>writelowExt)(sdmmc_ctrl_t *const p_ctrl, uint8_t const *const p_source, uint32_t const function, uint32_t const address, uint32_t const count, sdmmc_io_transfer_mode_t transfer_mode, sdmmc_io_address_mode_t address_mode)</code>
<code>fsp_err_t(*</code>	<code>ioIntEnable)(sdmmc_ctrl_t *const p_ctrl, bool enable)</code>
<code>fsp_err_t(*</code>	<code>statusGet)(sdmmc_ctrl_t *const p_ctrl, sdmmc_status_t *const p_status)</code>
<code>fsp_err_t(*</code>	<code>erase)(sdmmc_ctrl_t *const p_ctrl, uint32_t const start_sector, uint32_t const sector_count)</code>
<code>fsp_err_t(*</code>	<code>close)(sdmmc_ctrl_t *const p_ctrl)</code>

`fsp_err_t(* versionGet )(fsp_version_t *const p_version)`

Field Documentation

◆ open

`fsp_err_t(* sdmmc_api_t::open) (sdmmc_ctrl_t *const p_ctrl, sdmmc_cfg_t const *const p_cfg)`

Open the SD/MMC driver.

Implemented as
`R_SDHI_Open()`

Parameters

[in]	p_ctrl	Pointer to SD/MMC instance control block.
[in]	p_cfg	Pointer to SD/MMC instance configuration structure.

◆ mediaInit

`fsp_err_t(* sdmmc_api_t::mediaInit) (sdmmc_ctrl_t *const p_ctrl, sdmmc_device_t *const p_device)`

Initializes an SD/MMC device. If the device is a card, the card must be plugged in prior to calling this API. This API blocks until the device initialization procedure is complete.

Implemented as
`R_SDHI_MediaInit()`

Parameters

[in]	p_ctrl	Pointer to SD/MMC instance control block.
[out]	p_device	Pointer to store device information.

◆ read

```
fsp_err_t(* sdmmc_api_t::read) (sdmmc_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const
start_sector, uint32_t const sector_count)
```

Read data from an SD/MMC channel. This API is not supported for SDIO devices.

Implemented as

[R_SDHI_Read\(\)](#)

Parameters

[in]	p_ctrl	Pointer to an open SD/MMC instance control block.
[out]	p_dest	Pointer to data buffer to read data to.
[in]	start_sector	First sector address to read.
[in]	sector_count	Number of sectors to read. All sectors must be in the range of sdmmc_device_t::sector_count .

◆ write

```
fsp_err_t(* sdmmc_api_t::write) (sdmmc_ctrl_t *const p_ctrl, uint8_t const *const p_source, uint32_t
const start_sector, uint32_t const sector_count)
```

Write data to SD/MMC channel. This API is not supported for SDIO devices.

Implemented as

[R_SDHI_Write\(\)](#)

Parameters

[in]	p_ctrl	Pointer to an open SD/MMC instance control block.
[in]	p_source	Pointer to data buffer to write data from.
[in]	start_sector	First sector address to write to.
[in]	sector_count	Number of sectors to write. All sectors must be in the range of sdmmc_device_t::sector_count .

◆ readlo

```
fsp_err_t(* sdmmc_api_t::readlo) (sdmmc_ctrl_t *const p_ctrl, uint8_t *const p_data, uint32_t const
function, uint32_t const address)
```

Read one byte of I/O data from an SDIO device. This API is not supported for SD or eMMC memory devices.

Implemented as

[R_SDHI_Readlo\(\)](#)

Parameters

[in]	p_ctrl	Pointer to an open SD/MMC instance control block.
[out]	p_data	Pointer to location to store data byte.
[in]	function	SDIO Function Number.
[in]	address	SDIO register address.

◆ writelo

```
fsp_err_t(* sdmmc_api_t::writelo) (sdmmc_ctrl_t *const p_ctrl, uint8_t *const p_data, uint32_t const
function, uint32_t const address, sdmmc_io_write_mode_t const read_after_write)
```

Write one byte of I/O data to an SDIO device. This API is not supported for SD or eMMC memory devices.

Implemented as

[R_SDHI_Writelo\(\)](#)

Parameters

[in]	p_ctrl	Pointer to an open SD/MMC instance control block.
[in,out]	p_data	Pointer to data byte to write. Read data is also provided here if read_after_write is true.
[in]	function	SDIO Function Number.
[in]	address	SDIO register address.
[in]	read_after_write	Whether or not to read back the same register after writing

◆ readIoExt

```
fsp_err_t(* sdmmc_api_t::readIoExt) (sdmmc_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t
const function, uint32_t const address, uint32_t *const count, sdmmc_io_transfer_mode_t
transfer_mode, sdmmc_io_address_mode_t address_mode)
```

Read multiple bytes or blocks of I/O data from an SDIO device. This API is not supported for SD or eMMC memory devices.

Implemented as
R_SDHI_ReadIoExt()

Parameters

[in]	p_ctrl	Pointer to an open SD/MMC instance control block.
[out]	p_dest	Pointer to data buffer to read data to.
[in]	function	SDIO Function Number.
[in]	address	SDIO register address.
[in]	count	Number of bytes or blocks to read, maximum 512 bytes or 511 blocks.
[in]	transfer_mode	Byte or block mode
[in]	address_mode	Fixed or incrementing address mode

◆ **writeloExt**

```
fsp_err_t(* sdmmc_api_t::writeloExt) (sdmmc_ctrl_t *const p_ctrl, uint8_t const *const p_source,
uint32_t const function, uint32_t const address, uint32_t const count, sdmmc_io_transfer_mode_t
transfer_mode, sdmmc_io_address_mode_t address_mode)
```

Write multiple bytes or blocks of I/O data to an SDIO device. This API is not supported for SD or eMMC memory devices.

Implemented as

R_SDHI_WriteloExt()

Parameters

[in]	p_ctrl	Pointer to an open SD/MMC instance control block.
[in]	p_source	Pointer to data buffer to write data from.
[in]	function_number	SDIO Function Number.
[in]	address	SDIO register address.
[in]	count	Number of bytes or blocks to write, maximum 512 bytes or 511 blocks.
[in]	transfer_mode	Byte or block mode
[in]	address_mode	Fixed or incrementing address mode

◆ **ioIntEnable**

```
fsp_err_t(* sdmmc_api_t::ioIntEnable) (sdmmc_ctrl_t *const p_ctrl, bool enable)
```

Enables SDIO interrupt for SD/MMC instance. This API is not supported for SD or eMMC memory devices.

Implemented as

R_SDHI_IoIntEnable

Parameters

[in]	p_ctrl	Pointer to an open SD/MMC instance control block.
[in]	enable	Interrupt enable = true, interrupt disable = false.

◆ **statusGet**

```
fsp_err_t(* sdmmc_api_t::statusGet) (sdmmc_ctrl_t *const p_ctrl, sdmmc_status_t *const p_status)
```

Get SD/MMC device status.

Implemented as

[R_SDHI_StatusGet\(\)](#)

Parameters

[in]	p_ctrl	Pointer to an open SD/MMC instance control block.
[out]	p_status	Pointer to current driver status.

◆ **erase**

```
fsp_err_t(* sdmmc_api_t::erase) (sdmmc_ctrl_t *const p_ctrl, uint32_t const start_sector, uint32_t const sector_count)
```

Erase SD/MMC sectors. The sector size for erase is fixed at 512 bytes. This API is not supported for SDIO devices.

Implemented as

[R_SDHI_Erase](#)

Parameters

[in]	p_ctrl	Pointer to an open SD/MMC instance control block.
[in]	start_sector	First sector to erase. Must be a multiple of sdmmc_device_t::erase_sector_count .
[in]	sector_count	Number of sectors to erase. Must be a multiple of sdmmc_device_t::erase_sector_count . All sectors must be in the range of sdmmc_device_t::sector_count .

◆ **close**

```
fsp_err_t(* sdmmc_api_t::close) (sdmmc_ctrl_t *const p_ctrl)
```

Close open SD/MMC device.

Implemented as

[R_SDHI_Close\(\)](#)

Parameters

[in]	p_ctrl	Pointer to an open SD/MMC instance control block.

◆ **versionGet**

```
fsp_err_t(* sdmmc_api_t::versionGet) (fsp_version_t *const p_version)
```

Returns the version of the SD/MMC driver.

Implemented as

[R_SDHI_VersionGet\(\)](#)

Parameters

[out]	p_version	Pointer to return version information to.

◆ **sdmmc_instance_t**

```
struct sdmmc_instance_t
```

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields

sdmmc_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
sdmmc_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
sdmmc_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation◆ **sdmmc_ctrl_t**

```
typedef void sdmmc_ctrl_t
```

SD/MMC control block. Allocate an instance specific control block to pass into the SD/MMC API calls.

Implemented as

◦ [sdmmc_instance_ctrl_t](#)

Enumeration Type Documentation

◆ sdmmc_card_type_t

enum <code>sdmmc_card_type_t</code>	
SD/MMC media uses SD protocol or MMC protocol.	
Enumerator	
SDMMC_CARD_TYPE_MMC	The media is an eMMC device.
SDMMC_CARD_TYPE_SD	The media is an SD card.
SDMMC_CARD_TYPE_SDIO	The media is an SDIO card.

◆ sdmmc_bus_width_t

enum <code>sdmmc_bus_width_t</code>	
SD/MMC data bus is 1, 4 or 8 bits wide.	
Enumerator	
SDMMC_BUS_WIDTH_1_BIT	Data bus is 1 bit wide.
SDMMC_BUS_WIDTH_4_BITS	Data bus is 4 bits wide.
SDMMC_BUS_WIDTH_8_BITS	Data bus is 8 bits wide.

◆ sdmmc_io_transfer_mode_t

enum <code>sdmmc_io_transfer_mode_t</code>	
SDIO transfer mode, configurable in SDIO read/write extended commands.	
Enumerator	
SDMMC_IO_MODE_TRANSFER_BYTE	SDIO byte transfer mode.
SDMMC_IO_MODE_TRANSFER_BLOCK	SDIO block transfer mode.

◆ **sdmmc_io_address_mode_t**

enum sdmmc_io_address_mode_t	
SDIO address mode, configurable in SDIO read/write extended commands.	
Enumerator	
SDMMC_IO_ADDRESS_MODE_FIXED	Write all data to the same address.
SDMMC_IO_ADDRESS_MODE_INCREMENT	Increment destination address after each write.

◆ **sdmmc_io_write_mode_t**

enum sdmmc_io_write_mode_t	
Controls the RAW (read after write) flag of CMD52. Used to read back the status after writing a control register.	
Enumerator	
SDMMC_IO_WRITE_MODE_NO_READ	Write only (do not read back)
SDMMC_IO_WRITE_READ_AFTER_WRITE	Read back the register after write.

◆ **sdmmc_event_t**

enum sdmmc_event_t	
Events that can trigger a callback function	
Enumerator	
SDMMC_EVENT_CARD_REMOVED	Card removed event.
SDMMC_EVENT_CARD_INSERTED	Card inserted event.
SDMMC_EVENT_RESPONSE	Response event.
SDMMC_EVENT_SDIO	IO event.
SDMMC_EVENT_TRANSFER_COMPLETE	Read or write complete.
SDMMC_EVENT_TRANSFER_ERROR	Read or write failed.

◆ **sdmmc_card_detect_t**

enum sdmmc_card_detect_t	
Card detection configuration options.	
Enumerator	
SDMMC_CARD_DETECT_NONE	Card detection unused.
SDMMC_CARD_DETECT_CD	Card detection using the CD pin.

◆ **sdmmc_write_protect_t**

enum sdmmc_write_protect_t	
Write protection configuration options.	
Enumerator	
SDMMC_WRITE_PROTECT_NONE	Write protection unused.
SDMMC_WRITE_PROTECT_WP	Write protection using WP pin.

◆ **sdmmc_r1_state_t**

enum sdmmc_r1_state_t	
Card state when receiving the prior command.	
Enumerator	
SDMMC_R1_STATE_IDLE	Idle State.
SDMMC_R1_STATE_READY	Ready State.
SDMMC_R1_STATE_IDENT	Identification State.
SDMMC_R1_STATE_STBY	Stand-by State.
SDMMC_R1_STATE_TRAN	Transfer State.
SDMMC_R1_STATE_DATA	Sending-data State.
SDMMC_R1_STATE_RCV	Receive-data State.
SDMMC_R1_STATE_PRG	Programming State.
SDMMC_R1_STATE_DIS	Disconnect State (between programming and stand-by)
SDMMC_R1_STATE_IO	This is an I/O card and memory states do not apply.

4.3.25 SPI Interface[Interfaces](#)**Detailed Description**

Interface for SPI communications.

Summary

Provides a common interface for communication using the SPI Protocol.

Implemented by:

- [Serial Peripheral Interface \(r_spi\)](#)
- [Serial Communications Interface \(SCI\) SPI \(r_sci_spi\)](#)

Data Structures

struct [spi_callback_args_t](#)

struct [spi_cfg_t](#)

struct [spi_api_t](#)

struct [spi_instance_t](#)

Typedefs

typedef void [spi_ctrl_t](#)

Enumerations

enum [spi_bit_width_t](#)

enum [spi_mode_t](#)

enum [spi_clk_phase_t](#)

enum [spi_clk_polarity_t](#)

enum [spi_mode_fault_t](#)

enum [spi_bit_order_t](#)

enum [spi_event_t](#)

Data Structure Documentation

◆ [spi_callback_args_t](#)

struct spi_callback_args_t		
Common callback parameter definition		
Data Fields		
uint32_t	channel	Device channel number.
spi_event_t	event	Event code.
void const *	p_context	Context provided to user during callback.

◆ [spi_cfg_t](#)

struct spi_cfg_t	
SPI interface configuration	
Data Fields	
uint8_t	channel

	Channel number to be used.
IRQn_Type	rxi_irq
	Receive Buffer Full IRQ number.
IRQn_Type	txi_irq
	Transmit Buffer Empty IRQ number.
IRQn_Type	tei_irq
	Transfer Complete IRQ number.
IRQn_Type	eri_irq
	Error IRQ number.
uint8_t	rxi_ipl
	Receive Interrupt priority.
uint8_t	txi_ipl
	Transmit Interrupt priority.
uint8_t	tei_ipl
	Transfer Complete Interrupt priority.
uint8_t	eri_ipl
	Error Interrupt priority.
spi_mode_t	operating_mode
	Select master or slave operating mode.

spi_clk_phase_t	clk_phase
	Data sampling on odd or even clock edge.
spi_clk_polarity_t	clk_polarity
	Clock level when idle.
spi_mode_fault_t	mode_fault
	Mode fault error (master/slave conflict) flag.
spi_bit_order_t	bit_order
	Select to transmit MSB/LSB first.
transfer_instance_t const *	p_transfer_tx
	To use SPI DTC/DMA write transfer, link a DTC/DMA instance here. Set to NULL if unused.
transfer_instance_t const *	p_transfer_rx
	To use SPI DTC/DMA read transfer, link a DTC/DMA instance here. Set to NULL if unused.
void(*	p_callback)(spi_callback_args_t *p_args)
	Pointer to user callback function.
void const *	p_context
	User defined context passed to callback function.
void const *	p_extend
	Extended SPI hardware dependent configuration.

◆ **spi_api_t**

struct spi_api_t

Shared Interface definition for SPI

Data Fieldsfsp_err_t(*) **open**)(spi_ctrl_t *p_ctrl, spi_cfg_t const *const p_cfg)fsp_err_t(*) **read**)(spi_ctrl_t *const p_ctrl, void *p_dest, uint32_t const length, spi_bit_width_t const bit_width)fsp_err_t(*) **write**)(spi_ctrl_t *const p_ctrl, void const *p_src, uint32_t const length, spi_bit_width_t const bit_width)fsp_err_t(*) **writeRead**)(spi_ctrl_t *const p_ctrl, void const *p_src, void *p_dest, uint32_t const length, spi_bit_width_t const bit_width)fsp_err_t(*) **close**)(spi_ctrl_t *const p_ctrl)fsp_err_t(*) **versionGet**)(fsp_version_t *p_version)**Field Documentation**◆ **open**

fsp_err_t(*) spi_api_t::open) (spi_ctrl_t *p_ctrl, spi_cfg_t const *const p_cfg)

Initialize a channel for SPI communication mode.

Implemented as

- R_SPI_Open()
- R_SCI_SPI_Open()

Parameters

[in,out]	p_ctrl	Pointer to user-provided storage for the control block.
[in]	p_cfg	Pointer to SPI configuration structure.

◆ read

```
fsp_err_t(* spi_api_t::read) (spi_ctrl_t *const p_ctrl, void *p_dest, uint32_t const length,
spi_bit_width_t const bit_width)
```

Receive data from a SPI device.

Implemented as

- R_SPI_Read()
- R_SCI_SPI_Read()

Parameters

[in]	p_ctrl	Pointer to the control block for the channel.
[in]	length	Number of units of data to be transferred (unit size specified by the bit_width).
[in]	bit_width	Data bit width to be transferred.
[out]	p_dest	Pointer to destination buffer into which data will be copied that is received from a SPI device. It is the responsibility of the caller to ensure that adequate space is available to hold the requested data count.

◆ write

`fsp_err_t(* spi_api_t::write) (spi_ctrl_t *const p_ctrl, void const *p_src, uint32_t const length, spi_bit_width_t const bit_width)`

Transmit data to a SPI device.

Implemented as

- `R_SPI_Write()`
- `R_SCI_SPI_Write()`

Parameters

[in]	p_ctrl	Pointer to the control block for the channel.
[in]	p_src	Pointer to a source data buffer from which data will be transmitted to a SPI device. The argument must not be NULL.
[in]	length	Number of units of data to be transferred (unit size specified by the bit_width).
[in]	bit_width	Data bit width to be transferred.

◆ **writeRead**

```
fsp_err_t(* spi_api_t::writeRead) (spi_ctrl_t *const p_ctrl, void const *p_src, void *p_dest, uint32_t
const length, spi_bit_width_t const bit_width)
```

Simultaneously transmit data to a SPI device while receiving data from a SPI device (full duplex).

Implemented as

- R_SPI_WriteRead()
- R_SCI_SPI_WriteRead()

Parameters

[in]	p_ctrl	Pointer to the control block for the channel.
[in]	p_src	Pointer to a source data buffer from which data will be transmitted to a SPI device. The argument must not be NULL.
[out]	p_dest	Pointer to destination buffer into which data will be copied that is received from a SPI device. It is the responsibility of the caller to ensure that adequate space is available to hold the requested data count. The argument must not be NULL.
[in]	length	Number of units of data to be transferred (unit size specified by the bit_width).
[in]	bit_width	Data bit width to be transferred.

◆ **close**

```
fsp_err_t(* spi_api_t::close) (spi_ctrl_t *const p_ctrl)
```

Remove power to the SPI channel designated by the handle and disable the associated interrupts.

Implemented as

- R_SPI_Close()
- R_SCI_SPI_Close()

Parameters

[in]	p_ctrl	Pointer to the control block for the channel.

◆ **versionGet**

```
fsp_err_t(* spi_api_t::versionGet) (fsp_version_t *p_version)
```

Get the version information of the underlying driver.

Implemented as

- R_SPI_VersionGet()
- R_SCI_SPI_VersionGet()

Parameters

[out]	p_version	pointer to memory location to return version number

◆ **spi_instance_t**

```
struct spi_instance_t
```

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields

spi_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
spi_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
spi_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation◆ **spi_ctrl_t**

```
typedef void spi_ctrl_t
```

SPI control block. Allocate an instance specific control block to pass into the SPI API calls.

Implemented as

- sci_spi_instance_ctrl_t
- spi_instance_ctrl_t

Enumeration Type Documentation

◆ **spi_bit_width_t**

enum <code>spi_bit_width_t</code>	
Data bit width	
Enumerator	
<code>SPI_BIT_WIDTH_8_BITS</code>	Data bit width is 8 bits byte.
<code>SPI_BIT_WIDTH_16_BITS</code>	Data bit width is 16 bits word.
<code>SPI_BIT_WIDTH_32_BITS</code>	Data bit width is 32 bits long word.

◆ **spi_mode_t**

enum <code>spi_mode_t</code>	
Master or slave operating mode	
Enumerator	
<code>SPI_MODE_MASTER</code>	Channel operates as SPI master.
<code>SPI_MODE_SLAVE</code>	Channel operates as SPI slave.

◆ **spi_clk_phase_t**

enum <code>spi_clk_phase_t</code>	
Clock phase	
Enumerator	
<code>SPI_CLK_PHASE_EDGE_ODD</code>	0: Data sampling on odd edge, data variation on even edge
<code>SPI_CLK_PHASE_EDGE_EVEN</code>	1: Data variation on odd edge, data sampling on even edge

◆ **spi_clk_polarity_t**

enum <code>spi_clk_polarity_t</code>	
Clock polarity	
Enumerator	
<code>SPI_CLK_POLARITY_LOW</code>	0: Clock polarity is low when idle
<code>SPI_CLK_POLARITY_HIGH</code>	1: Clock polarity is high when idle

◆ **spi_mode_fault_t**

enum <code>spi_mode_fault_t</code>	
Mode fault error flag. This error occurs when the device is setup as a master, but the SS/LA line does not seem to be controlled by the master. This usually happens when the connecting device is also acting as master. A similar situation can also happen when configured as a slave.	
Enumerator	
<code>SPI_MODE_FAULT_ERROR_ENABLE</code>	Mode fault error flag on.
<code>SPI_MODE_FAULT_ERROR_DISABLE</code>	Mode fault error flag off.

◆ **spi_bit_order_t**

enum <code>spi_bit_order_t</code>	
Bit order	
Enumerator	
<code>SPI_BIT_ORDER_MSB_FIRST</code>	Send MSB first in transmission.
<code>SPI_BIT_ORDER_LSB_FIRST</code>	Send LSB first in transmission.

◆ **spi_event_t**

enum <code>spi_event_t</code>	
SPI events	
Enumerator	
<code>SPI_EVENT_TRANSFER_COMPLETE</code>	The data transfer was completed.
<code>SPI_EVENT_TRANSFER_ABORTED</code>	The data transfer was aborted.
<code>SPI_EVENT_ERR_MODE_FAULT</code>	Mode fault error.
<code>SPI_EVENT_ERR_READ_OVERFLOW</code>	Read overflow error.
<code>SPI_EVENT_ERR_PARITY</code>	Parity error.
<code>SPI_EVENT_ERR_OVERRUN</code>	Overrun error.
<code>SPI_EVENT_ERR_FRAMING</code>	Framing error.
<code>SPI_EVENT_ERR_MODE_UNDERRUN</code>	Underrun error.

4.3.26 Timer Interface

Interfaces

Detailed Description

Interface for timer functions.

Summary

The general timer interface provides standard timer functionality including periodic mode, one-shot mode, PWM output, and free-running timer mode. After each timer cycle (overflow or underflow), an interrupt can be triggered.

If an instance supports output compare mode, it is provided in the extension configuration `timer_on_<instance>_cfg_t` defined in `r_<instance>.h`.

Implemented by:

- General PWM Timer (`r_gpt`)
- Asynchronous General Purpose Timer (`r_agt`)

Data Structures

struct [timer_callback_args_t](#)

struct [timer_info_t](#)

struct [timer_status_t](#)

struct [timer_cfg_t](#)

struct [timer_api_t](#)

struct [timer_instance_t](#)

Typedefs

typedef void [timer_ctrl_t](#)

Enumerations

enum [timer_event_t](#)

enum [timer_variant_t](#)

enum [timer_state_t](#)

enum [timer_mode_t](#)

enum [timer_direction_t](#)

enum [timer_source_div_t](#)

Data Structure Documentation

◆ timer_callback_args_t

struct timer_callback_args_t		
Callback function parameter data		
Data Fields		
void const *	p_context	Placeholder for user data. Set in timer_api_t::open function in timer_cfg_t .
timer_event_t	event	The event can be used to identify what caused the callback.
uint32_t	capture	

◆ timer_info_t

struct timer_info_t

Timer information structure to store various information for a timer resource

Data Fields		
timer_direction_t	count_direction	Clock counting direction of the timer resource.
uint32_t	clock_frequency	Clock frequency of the timer resource.
uint32_t	period_counts	Time in clock counts until timer will expire.

◆ timer_status_t

struct timer_status_t		
Current timer status.		
Data Fields		
uint32_t	counter	Current counter value.
timer_state_t	state	Current timer state (running or stopped)

◆ timer_cfg_t

struct timer_cfg_t	
User configuration structure, used in open function	
Data Fields	
timer_mode_t	mode
	Select enumerated value from timer_mode_t.
uint32_t	period_counts
	Period in raw timer counts.
timer_source_div_t	source_div
	Source clock divider.
uint32_t	duty_cycle_counts
	Duty cycle in counts.
uint8_t	channel

uint8_t	cycle_end_ipl
	Cycle end interrupt priority.
IRQn_Type	cycle_end_irq
	Cycle end interrupt.
void(*	p_callback)(timer_callback_args_t *p_args)
void const *	p_context
void const *	p_extend
	Extension parameter for hardware specific settings.

Field Documentation

◆ channel

uint8_t timer_cfg_t::channel

Select a channel corresponding to the channel number of the hardware.

◆ p_callback

void(* timer_cfg_t::p_callback) (timer_callback_args_t *p_args)

Callback provided when a timer ISR occurs. Set to NULL for no CPU interrupt.

◆ p_context

void const* timer_cfg_t::p_context

Placeholder for user data. Passed to the user callback in [timer_callback_args_t](#).

◆ timer_api_t

struct timer_api_t

Timer API structure. General timer functions implemented at the HAL layer follow this API.

Data Fields

fsp_err_t(* [open](#))(timer_ctrl_t *const p_ctrl, timer_cfg_t const *const p_cfg)

<code>fsp_err_t(*</code>	<code>start)(timer_ctrl_t *const p_ctrl)</code>
<code>fsp_err_t(*</code>	<code>stop)(timer_ctrl_t *const p_ctrl)</code>
<code>fsp_err_t(*</code>	<code>reset)(timer_ctrl_t *const p_ctrl)</code>
<code>fsp_err_t(*</code>	<code>enable)(timer_ctrl_t *const p_ctrl)</code>
<code>fsp_err_t(*</code>	<code>disable)(timer_ctrl_t *const p_ctrl)</code>
<code>fsp_err_t(*</code>	<code>periodSet)(timer_ctrl_t *const p_ctrl, uint32_t const period)</code>
<code>fsp_err_t(*</code>	<code>dutyCycleSet)(timer_ctrl_t *const p_ctrl, uint32_t const duty_cycle_counts, uint32_t const pin)</code>
<code>fsp_err_t(*</code>	<code>infoGet)(timer_ctrl_t *const p_ctrl, timer_info_t *const p_info)</code>
<code>fsp_err_t(*</code>	<code>statusGet)(timer_ctrl_t *const p_ctrl, timer_status_t *const p_status)</code>
<code>fsp_err_t(*</code>	<code>close)(timer_ctrl_t *const p_ctrl)</code>
<code>fsp_err_t(*</code>	<code>versionGet)(fsp_version_t *const p_version)</code>

Field Documentation

◆ open

```
fsp_err_t(* timer_api_t::open) (timer_ctrl_t *const p_ctrl, timer_cfg_t const *const p_cfg)
```

Initial configuration.

Implemented as

- R_GPT_Open()
- R_AGT_Open()

Parameters

[in]	p_ctrl	Pointer to control block. Must be declared by user. Elements set here.
[in]	p_cfg	Pointer to configuration structure. All elements of this structure must be set by user.

◆ start

```
fsp_err_t(* timer_api_t::start) (timer_ctrl_t *const p_ctrl)
```

Start the counter.

Implemented as

- R_GPT_Start()
- R_AGT_Start()

Parameters

[in]	p_ctrl	Control block set in timer_api_t::open call for this timer.

◆ stop

```
fsp_err_t(* timer_api_t::stop) (timer_ctrl_t *const p_ctrl)
```

Stop the counter.

Implemented as

- R_GPT_Stop()
- R_AGT_Stop()

Parameters

[in]	p_ctrl	Control block set in timer_api_t::open call for this timer.

◆ reset

```
fsp_err_t(* timer_api_t::reset) (timer_ctrl_t *const p_ctrl)
```

Reset the counter to the initial value.

Implemented as

- R_GPT_Reset()
- R_AGT_Reset()

Parameters

[in]	p_ctrl	Control block set in timer_api_t::open call for this timer.

◆ enable

```
fsp_err_t(* timer_api_t::enable) (timer_ctrl_t *const p_ctrl)
```

Enables input capture.

Implemented as

- R_GPT_Enable()
- R_AGT_Enable()

Parameters

[in]	p_ctrl	Control block set in timer_api_t::open call for this timer.

◆ disable

```
fsp_err_t(* timer_api_t::disable) (timer_ctrl_t *const p_ctrl)
```

Disables input capture.

Implemented as

- R_GPT_Disable()
- R_AGT_Disable()

Parameters

[in]	p_ctrl	Control block set in timer_api_t::open call for this timer.

◆ **periodSet**

```
fsp_err_t(* timer_api_t::periodSet) (timer_ctrl_t *const p_ctrl, uint32_t const period)
```

Set the time until the timer expires. See implementation for details of period update timing.

Implemented as

- [R_GPT_PeriodSet\(\)](#)
- [R_AGT_PeriodSet\(\)](#)

Note

Timer expiration may or may not generate a CPU interrupt based on how the timer is configured in [timer_api_t::open](#).

Parameters

[in]	p_ctrl	Control block set in timer_api_t::open call for this timer.
[in]	p_period	Time until timer should expire.

◆ **dutyCycleSet**

```
fsp_err_t(* timer_api_t::dutyCycleSet) (timer_ctrl_t *const p_ctrl, uint32_t const duty_cycle_counts, uint32_t const pin)
```

Sets the number of counts for the pin level to be high. If the timer is counting, the updated duty cycle is reflected after the next timer expiration.

Implemented as

- [R_GPT_DutyCycleSet\(\)](#)
- [R_AGT_DutyCycleSet\(\)](#)

Parameters

[in]	p_ctrl	Control block set in timer_api_t::open call for this timer.
[in]	duty_cycle_counts	Time until duty cycle should expire.
[in]	pin	Which output pin to update. See implementation for details.

◆ **infoGet**

```
fsp_err_t(* timer_api_t::infoGet) (timer_ctrl_t *const p_ctrl, timer_info_t *const p_info)
```

Stores timer information in p_info.

Implemented as

- R_GPT_InfoGet()
- R_AGT_InfoGet()

Parameters

[in]	p_ctrl	Control block set in timer_api_t::open call for this timer.
[out]	p_info	Collection of information for this timer.

◆ **statusGet**

```
fsp_err_t(* timer_api_t::statusGet) (timer_ctrl_t *const p_ctrl, timer_status_t *const p_status)
```

Get the current counter value and timer state and store it in p_status.

Implemented as

- R_GPT_StatusGet()
- R_AGT_StatusGet()

Parameters

[in]	p_ctrl	Control block set in timer_api_t::open call for this timer.
[out]	p_status	Current status of this timer.

◆ **close**

```
fsp_err_t(* timer_api_t::close) (timer_ctrl_t *const p_ctrl)
```

Allows driver to be reconfigured and may reduce power consumption.

Implemented as

- R_GPT_Close()
- R_AGT_Close()

Parameters

[in]	p_ctrl	Control block set in timer_api_t::open call for this timer.

◆ **versionGet**

```
fsp_err_t(* timer_api_t::versionGet) (fsp_version_t *const p_version)
```

Get version and store it in provided pointer p_version.

Implemented as

- [R_GPT_VersionGet\(\)](#)
- [R_AGT_VersionGet\(\)](#)

Parameters

[out]	p_version	Code and API version used.

◆ **timer_instance_t**

```
struct timer_instance_t
```

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields

timer_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
timer_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
timer_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation◆ **timer_ctrl_t**

```
typedef void timer_ctrl_t
```

Timer control block. Allocate an instance specific control block to pass into the timer API calls.

Implemented as

- [gpt_instance_ctrl_t](#)
- [agt_instance_ctrl_t](#)

Enumeration Type Documentation

◆ **timer_event_t**

enum timer_event_t	
Events that can trigger a callback function	
Enumerator	
TIMER_EVENT_CYCLE_END	Requested timer delay has expired or timer has wrapped around.
TIMER_EVENT_CAPTURE_A	A capture has occurred on signal A.
TIMER_EVENT_CAPTURE_B	A capture has occurred on signal B.

◆ **timer_variant_t**

enum timer_variant_t	
Timer variant types.	
Enumerator	
TIMER_VARIANT_32_BIT	32-bit timer
TIMER_VARIANT_16_BIT	16-bit timer

◆ **timer_state_t**

enum timer_state_t	
Possible status values returned by timer_api_t::statusGet .	
Enumerator	
TIMER_STATE_STOPPED	Timer is stopped.
TIMER_STATE_COUNTING	Timer is running.

◆ **timer_mode_t**

enum timer_mode_t	
Timer operational modes	
Enumerator	
TIMER_MODE_PERIODIC	Timer will restart after delay periods.
TIMER_MODE_ONE_SHOT	Timer will stop after delay periods.
TIMER_MODE_PWM	Timer generate PWM output.

◆ **timer_direction_t**

enum timer_direction_t	
Direction of timer count	
Enumerator	
TIMER_DIRECTION_DOWN	Timer count goes up.
TIMER_DIRECTION_UP	Timer count goes down.

◆ **timer_source_div_t**

enum <code>timer_source_div_t</code>	
PCLK divisors	
Enumerator	
<code>TIMER_SOURCE_DIV_1</code>	Timer clock source divided by 1.
<code>TIMER_SOURCE_DIV_2</code>	Timer clock source divided by 2.
<code>TIMER_SOURCE_DIV_4</code>	Timer clock source divided by 4.
<code>TIMER_SOURCE_DIV_8</code>	Timer clock source divided by 8.
<code>TIMER_SOURCE_DIV_16</code>	Timer clock source divided by 16.
<code>TIMER_SOURCE_DIV_32</code>	Timer clock source divided by 32.
<code>TIMER_SOURCE_DIV_64</code>	Timer clock source divided by 64.
<code>TIMER_SOURCE_DIV_128</code>	Timer clock source divided by 128.
<code>TIMER_SOURCE_DIV_256</code>	Timer clock source divided by 256.
<code>TIMER_SOURCE_DIV_1024</code>	Timer clock source divided by 1024.

4.3.27 Transfer Interface

Interfaces

Detailed Description

Interface for data transfer functions.

Summary

The transfer interface supports background data transfer (no CPU intervention).

Implemented by:

- [Data Transfer Controller \(r_dtc\)](#)
- [Direct Memory Access Controller \(r_dmac\)](#)

Data Structures

struct [transfer_properties_t](#)

struct [transfer_info_t](#)

struct [transfer_cfg_t](#)

struct [transfer_api_t](#)

struct [transfer_instance_t](#)

Typedefs

typedef void [transfer_ctrl_t](#)

Enumerations

enum [transfer_mode_t](#)

enum [transfer_size_t](#)

enum [transfer_addr_mode_t](#)

enum [transfer_repeat_area_t](#)

enum [transfer_chain_mode_t](#)

enum [transfer_irq_t](#)

enum [transfer_start_mode_t](#)

Data Structure Documentation

◆ transfer_properties_t

struct transfer_properties_t		
Driver specific information.		
Data Fields		
uint32_t	block_count_max	Maximum number of blocks.
uint32_t	block_count_remaining	Number of blocks remaining.
uint32_t	transfer_length_max	Maximum number of transfers.
uint32_t	transfer_length_remaining	Number of transfers remaining.

◆ transfer_info_t

struct transfer_info_t
This structure specifies the properties of the transfer.

Warning

When using DTC, this structure corresponds to the descriptor block registers required by the DTC. The following components may be modified by the driver: `p_src`, `p_dest`, `num_blocks`, and `length`.
 When using DTC, do NOT reuse this structure to configure multiple transfers. Each transfer must have a unique [transfer_info_t](#).
 When using DTC, this structure must not be allocated in a temporary location. Any instance of this structure must remain in scope until the transfer it is used for is closed.

Note

When using DTC, consider placing instances of this structure in a protected section of memory.

Data Fields		
union transfer_info_t	__unnamed__	
void const *volatile	<code>p_src</code>	Source pointer.
void *volatile	<code>p_dest</code>	Destination pointer.
volatile uint16_t	<code>num_blocks</code>	Number of blocks to transfer when using TRANSFER_MODE_BLOCK (both DTC and DMAC) and TRANSFER_MODE_REPEAT (DMAC only), unused in other modes.
volatile uint16_t	<code>length</code>	Length of each transfer. Range limited for TRANSFER_MODE_BLOCK and TRANSFER_MODE_REPEAT , see HAL driver for details.

◆ transfer_cfg_t

struct transfer_cfg_t		
Driver configuration set in transfer_api_t::open . All elements except <code>p_extend</code> are required and must be initialized.		
Data Fields		
transfer_info_t *	<code>p_info</code>	Pointer to transfer configuration options. If using chain transfer (DTC only), this can be a pointer to an array of chained transfers that will be completed in order.
void const *	<code>p_extend</code>	Extension parameter for hardware specific settings.

◆ transfer_api_t

struct transfer_api_t		
Transfer functions implemented at the HAL layer will follow this API.		
Data Fields		
fsp_err_t (*	<code>open</code>)	(transfer_ctrl_t *const <code>p_ctrl</code> , transfer_cfg_t const *const <code>p_cfg</code>)

<code>fsp_err_t(*</code>	<code>reconfigure)(transfer_ctrl_t *const p_ctrl, transfer_info_t *p_info)</code>
<code>fsp_err_t(*</code>	<code>reset)(transfer_ctrl_t *const p_ctrl, void const *p_src, void *p_dest, uint16_t const num_transfers)</code>
<code>fsp_err_t(*</code>	<code>enable)(transfer_ctrl_t *const p_ctrl)</code>
<code>fsp_err_t(*</code>	<code>disable)(transfer_ctrl_t *const p_ctrl)</code>
<code>fsp_err_t(*</code>	<code>softwareStart)(transfer_ctrl_t *const p_ctrl, transfer_start_mode_t mode)</code>
<code>fsp_err_t(*</code>	<code>softwareStop)(transfer_ctrl_t *const p_ctrl)</code>
<code>fsp_err_t(*</code>	<code>infoGet)(transfer_ctrl_t *const p_ctrl, transfer_properties_t *const p_properties)</code>
<code>fsp_err_t(*</code>	<code>close)(transfer_ctrl_t *const p_ctrl)</code>
<code>fsp_err_t(*</code>	<code>versionGet)(fsp_version_t *const p_version)</code>

Field Documentation

◆ **open**

```
fsp_err_t(* transfer_api_t::open) (transfer_ctrl_t *const p_ctrl, transfer_cfg_t const *const p_cfg)
```

Initial configuration.

Implemented as

- R_DTC_Open()
- R_DMAC_Open()

Parameters

[in,out]	p_ctrl	Pointer to control block. Must be declared by user. Elements set here.
[in]	p_cfg	Pointer to configuration structure. All elements of this structure must be set by user.

◆ **reconfigure**

```
fsp_err_t(* transfer_api_t::reconfigure) (transfer_ctrl_t *const p_ctrl, transfer_info_t *p_info)
```

Reconfigure the transfer. Enable the transfer if p_info is valid.

Implemented as

- R_DTC_Reconfigure()
- R_DMAC_Reconfigure()

Parameters

[in,out]	p_ctrl	Pointer to control block. Must be declared by user. Elements set here.
[in]	p_info	Pointer to a new transfer info structure.

◆ **reset**

```
fsp_err_t(* transfer_api_t::reset) (transfer_ctrl_t *const p_ctrl, void const *p_src, void *p_dest,
uint16_t const num_transfers)
```

Reset source address pointer, destination address pointer, and/or length, keeping all other settings the same. Enable the transfer if p_src, p_dest, and length are valid.

Implemented as

- R_DTC_Reset()
- R_DMAMC_Reset()

Parameters

[in]	p_ctrl	Control block set in transfer_api_t::open call for this transfer.
[in]	p_src	Pointer to source. Set to NULL if source pointer should not change.
[in]	p_dest	Pointer to destination. Set to NULL if destination pointer should not change.
[in]	num_transfers	Transfer length in normal mode or number of blocks in block mode. In DMAMC only, resets number of repeats (initially stored in transfer_info_t::num_blocks) in repeat mode. Not used in repeat mode for DTC.

◆ **enable**

```
fsp_err_t(* transfer_api_t::enable) (transfer_ctrl_t *const p_ctrl)
```

Enable transfer. Transfers occur after the activation source event (or when transfer_api_t::start is called if ELC_EVENT_ELC_NONE is chosen as activation source).

Implemented as

- R_DTC_Enable()
- R_DMAMC_Enable()

Parameters

[in]	p_ctrl	Control block set in transfer_api_t::open call for this transfer.

◆ disable

`fsp_err_t(* transfer_api_t::disable) (transfer_ctrl_t *const p_ctrl)`

Disable transfer. Transfers do not occur after the activation source event (or when `transfer_api_t::start` is called if `ELC_EVENT_ELC_NONE` is chosen as the DMAC activation source).

Note
If a transfer is in progress, it will be completed. Subsequent transfer requests do not cause a transfer.

Implemented as

- `R_DTC_Disable()`
- `R_DMAC_Disable()`

Parameters

[in]	p_ctrl	Control block set in transfer_api_t::open call for this transfer.

◆ softwareStart

`fsp_err_t(* transfer_api_t::softwareStart) (transfer_ctrl_t *const p_ctrl, transfer_start_mode_t mode)`

Start transfer in software.

Warning
Only works if `ELC_EVENT_ELC_NONE` is chosen as the DMAC activation source.

Note
Not supported for DTC.

Implemented as

- `R_DMAC_SoftwareStart()`

Parameters

[in]	p_ctrl	Control block set in transfer_api_t::open call for this transfer.
[in]	mode	Select mode from transfer_start_mode_t .

◆ **softwareStop**

```
fsp_err_t(* transfer_api_t::softwareStop) (transfer_ctrl_t *const p_ctrl)
```

Stop transfer in software. The transfer will stop after completion of the current transfer.

Note

Not supported for DTC.

Only applies for transfers started with TRANSFER_START_MODE_REPEAT.

Warning

Only works if ELC_EVENT_ELC_NONE is chosen as the DMAC activation source.

Implemented as

- R_DMAMAC_SoftwareStop()

Parameters

[in]	p_ctrl	Control block set in transfer_api_t::open call for this transfer.

◆ **infoGet**

```
fsp_err_t(* transfer_api_t::infoGet) (transfer_ctrl_t *const p_ctrl, transfer_properties_t *const p_properties)
```

Provides information about this transfer.

Implemented as

- R_DTC_InfoGet()
- R_DMAMAC_InfoGet()

Parameters

[in]	p_ctrl	Control block set in transfer_api_t::open call for this transfer.
[out]	p_properties	Driver specific information.

◆ **close**

```
fsp_err_t(* transfer_api_t::close) (transfer_ctrl_t *const p_ctrl)
```

Releases hardware lock. This allows a transfer to be reconfigured using [transfer_api_t::open](#).

Implemented as

- R_DTC_Close()
- R_DMAMAC_Close()

Parameters

[in]	p_ctrl	Control block set in transfer_api_t::open call for this transfer.

◆ **versionGet**

```
fsp_err_t(* transfer_api_t::versionGet) (fsp_version_t *const p_version)
```

Gets version and stores it in provided pointer p_version.

Implemented as

- R_DTC_VersionGet()
- R_DMACE_VersionGet()

Parameters

[out]	p_version	Code and API version used.

◆ **transfer_instance_t**

```
struct transfer_instance_t
```

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields

transfer_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
transfer_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
transfer_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation◆ **transfer_ctrl_t**

```
typedef void transfer_ctrl_t
```

Transfer control block. Allocate an instance specific control block to pass into the transfer API calls.

Implemented as

- dtc_instance_ctrl_t
- dmac_instance_ctrl_t

Enumeration Type Documentation

◆ **transfer_mode_t**enum **transfer_mode_t**

Transfer mode describes what will happen when a transfer request occurs.

Enumerator

TRANSFER_MODE_NORMAL	In normal mode, each transfer request causes a transfer of transfer_size_t from the source pointer to the destination pointer. The transfer length is decremented and the source and address pointers are updated according to transfer_addr_mode_t . After the transfer length reaches 0, transfer requests will not cause any further transfers.
TRANSFER_MODE_REPEAT	Repeat mode is like normal mode, except that when the transfer length reaches 0, the pointer to the repeat area and the transfer length will be reset to their initial values. If DMAC is used, the transfer repeats only transfer_info_t::num_blocks times. After the transfer repeats transfer_info_t::num_blocks times, transfer requests will not cause any further transfers. If DTC is used, the transfer repeats continuously (no limit to the number of repeat transfers).
TRANSFER_MODE_BLOCK	In block mode, each transfer request causes transfer_info_t::length transfers of transfer_size_t . After each individual transfer, the source and destination pointers are updated according to transfer_addr_mode_t . After the block transfer is complete, transfer_info_t::num_blocks is decremented. After the transfer_info_t::num_blocks reaches 0, transfer requests will not cause any further transfers.

◆ **transfer_size_t**

enum transfer_size_t	
Transfer size specifies the size of each individual transfer. Total transfer length = transfer_size_t * transfer_length_t	
Enumerator	
TRANSFER_SIZE_1_BYTE	Each transfer transfers a 8-bit value.
TRANSFER_SIZE_2_BYTE	Each transfer transfers a 16-bit value.
TRANSFER_SIZE_4_BYTE	Each transfer transfers a 32-bit value.

◆ **transfer_addr_mode_t**

enum transfer_addr_mode_t	
Address mode specifies whether to modify (increment or decrement) pointer after each transfer.	
Enumerator	
TRANSFER_ADDR_MODE_FIXED	Address pointer remains fixed after each transfer.
TRANSFER_ADDR_MODE_OFFSET	Offset is added to the address pointer after each transfer.
TRANSFER_ADDR_MODE_INCREMENTED	Address pointer is incremented by associated transfer_size_t after each transfer.
TRANSFER_ADDR_MODE_DECREMENTED	Address pointer is decremented by associated transfer_size_t after each transfer.

◆ **transfer_repeat_area_t**

enum transfer_repeat_area_t	
Repeat area options (source or destination). In TRANSFER_MODE_REPEAT , the selected pointer returns to its original value after transfer_info_t::length transfers. In TRANSFER_MODE_BLOCK , the selected pointer returns to its original value after each transfer.	
Enumerator	
TRANSFER_REPEAT_AREA_DESTINATION	Destination area repeated in TRANSFER_MODE_REPEAT or TRANSFER_MODE_BLOCK .
TRANSFER_REPEAT_AREA_SOURCE	Source area repeated in TRANSFER_MODE_REPEAT or TRANSFER_MODE_BLOCK .

◆ **transfer_chain_mode_t**

enum transfer_chain_mode_t	
Chain transfer mode options.	
Note <i>Only applies for DTC.</i>	
Enumerator	
TRANSFER_CHAIN_MODE_DISABLED	Chain mode not used.
TRANSFER_CHAIN_MODE_EACH	Switch to next transfer after a single transfer from this transfer_info_t .
TRANSFER_CHAIN_MODE_END	Complete the entire transfer defined in this transfer_info_t before chaining to next transfer.

◆ **transfer_irq_t**

enum transfer_irq_t	
Interrupt options.	
Enumerator	
TRANSFER_IRQ_END	Interrupt occurs only after last transfer. If this transfer is chained to a subsequent transfer, the interrupt will occur only after subsequent chained transfer(s) are complete. Warning DTC triggers the interrupt of the activation source. Choosing TRANSFER_IRQ_END with DTC will prevent activation source interrupts until the transfer is complete.
TRANSFER_IRQ_EACH	Interrupt occurs after each transfer. Note <i>Not available in all HAL drivers. See HAL driver for details.</i>

◆ **transfer_start_mode_t**

enum transfer_start_mode_t	
Select whether to start single or repeated transfer with software start.	
Enumerator	
TRANSFER_START_MODE_SINGLE	Software start triggers single transfer.
TRANSFER_START_MODE_REPEAT	Software start transfer continues until transfer is complete.

4.3.28 UART Interface

Interfaces

Detailed Description

Interface for UART communications.

Summary

The UART interface provides common APIs for UART HAL drivers. The UART interface supports the following features:

- Full-duplex UART communication
- Interrupt driven transmit/receive processing
- Callback function with returned event code
- Runtime baud-rate change
- Hardware resource locking during a transaction
- CTS/RTS hardware flow control support (with an associated IOPORT pin)

Implemented by:

- [Serial Communications Interface \(SCI\) UART \(r_sci_uart\)](#)

Data Structures

	struct	uart_info_t
	struct	uart_callback_args_t
	struct	uart_cfg_t
	struct	uart_api_t
	struct	uart_instance_t

Typedefs

	typedef void	uart_ctrl_t

Enumerations

	enum	uart_event_t
	enum	uart_data_bits_t
	enum	uart_parity_t
	enum	uart_stop_bits_t
	enum	uart_dir_t

Data Structure Documentation

◆ [uart_info_t](#)

struct uart_info_t
UART driver specific information
Data Fields

uint32_t	write_bytes_max	Maximum bytes that can be written at this time. Only applies if uart_cfg_t::p_transfer_tx is not NULL.
uint32_t	read_bytes_max	Maximum bytes that are available to read at one time. Only applies if uart_cfg_t::p_transfer_rx is not NULL.

◆ **uart_callback_args_t**

struct uart_callback_args_t		
UART Callback parameter definition		
Data Fields		
uint32_t	channel	Device channel number.
uart_event_t	event	Event code.
uint32_t	data	Contains the next character received for the events UART_EVENT_RX_CHAR, UART_EVENT_ERR_PARITY, UART_EVENT_ERR_FRAMING, or UART_EVENT_ERR_OVERFLOW. Otherwise unused.
void const *	p_context	Context provided to user during callback.

◆ **uart_cfg_t**

struct uart_cfg_t	
UART Configuration	
Data Fields	
uint8_t	channel
	Select a channel corresponding to the channel number of the hardware.
uart_data_bits_t	data_bits
	Data bit length (8 or 7 or 9)
uart_parity_t	parity
	Parity type (none or odd or even)

<code>uart_stop_bits_t</code>	<code>stop_bits</code>
	Stop bit length (1 or 2)
<code>uint8_t</code>	<code>rx_ipl</code>
	Receive interrupt priority.
<code>IRQn_Type</code>	<code>rx_irq</code>
	Receive interrupt IRQ number.
<code>uint8_t</code>	<code>tx_ipl</code>
	Transmit interrupt priority.
<code>IRQn_Type</code>	<code>tx_irq</code>
	Transmit interrupt IRQ number.
<code>uint8_t</code>	<code>tei_ipl</code>
	Transmit end interrupt priority.
<code>IRQn_Type</code>	<code>tei_irq</code>
	Transmit end interrupt IRQ number.
<code>uint8_t</code>	<code>eri_ipl</code>
	Error interrupt priority.
<code>IRQn_Type</code>	<code>eri_irq</code>
	Error interrupt IRQ number.

<code>transfer_instance_t</code> const *	<code>p_transfer_rx</code>
<code>transfer_instance_t</code> const *	<code>p_transfer_tx</code>
<code>void</code> (*	<code>p_callback</code>)(<code>uart_callback_args_t</code> * <code>p_args</code>)
	Pointer to callback function.
<code>void</code> const *	<code>p_context</code>
	User defined context passed into callback function.
<code>void</code> const *	<code>p_extend</code>
	UART hardware dependent configuration.

Field Documentation

◆ `p_transfer_rx`

`transfer_instance_t` const* `uart_cfg_t::p_transfer_rx`

Optional transfer instance used to receive multiple bytes without interrupts. Set to NULL if unused. If NULL, the number of bytes allowed in the read API is limited to one byte at a time.

◆ `p_transfer_tx`

`transfer_instance_t` const* `uart_cfg_t::p_transfer_tx`

Optional transfer instance used to send multiple bytes without interrupts. Set to NULL if unused. If NULL, the number of bytes allowed in the write APIs is limited to one byte at a time.

◆ `uart_api_t`

`struct uart_api_t`

Shared Interface definition for UART

Data Fields

<code>fsp_err_t</code> (*	<code>open</code>)(<code>uart_ctrl_t</code> *const <code>p_ctrl</code> , <code>uart_cfg_t</code> const *const <code>p_cfg</code>)
<code>fsp_err_t</code> (*	<code>read</code>)(<code>uart_ctrl_t</code> *const <code>p_ctrl</code> , <code>uint8_t</code> *const <code>p_dest</code> , <code>uint32_t</code> const <code>bytes</code>)
<code>fsp_err_t</code> (*	<code>write</code>)(<code>uart_ctrl_t</code> *const <code>p_ctrl</code> , <code>uint8_t</code> const *const <code>p_src</code> , <code>uint32_t</code>

	const bytes)
<code>fsp_err_t(*</code>	<code>baudSet)(uart_ctrl_t *const p_ctrl, void const *const p_baudrate_info)</code>
<code>fsp_err_t(*</code>	<code>infoGet)(uart_ctrl_t *const p_ctrl, uart_info_t *const p_info)</code>
<code>fsp_err_t(*</code>	<code>communicationAbort)(uart_ctrl_t *const p_ctrl, uart_dir_t communication_to_abort)</code>
<code>fsp_err_t(*</code>	<code>close)(uart_ctrl_t *const p_ctrl)</code>
<code>fsp_err_t(*</code>	<code>versionGet)(fsp_version_t *p_version)</code>

Field Documentation

◆ open

`fsp_err_t(* uart_api_t::open) (uart_ctrl_t *const p_ctrl, uart_cfg_t const *const p_cfg)`

Open UART device.

Implemented as

- R_SCI_UartOpen()

Parameters

[in,out]	<code>p_ctrl</code>	Pointer to the UART control block Must be declared by user. Value set here.
[in]	<code>uart_cfg_t</code>	Pointer to UART configuration structure. All elements of this structure must be set by user.

◆ read

```
fsp_err_t(* uart_api_t::read) (uart_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const bytes)
```

Read from UART device. The read buffer is used until the read is complete. When a transfer is complete, the callback is called with event `UART_EVENT_RX_COMPLETE`. Bytes received outside an active transfer are received in the callback function with event `UART_EVENT_RX_CHAR`. The maximum transfer size is reported by [infoGet\(\)](#).

Implemented as

- `R_SCI_UartRead()`

Parameters

[in]	p_ctrl	Pointer to the UART control block for the channel.
[in]	p_dest	Destination address to read data from.
[in]	bytes	Read data length.

◆ write

```
fsp_err_t(* uart_api_t::write) (uart_ctrl_t *const p_ctrl, uint8_t const *const p_src, uint32_t const bytes)
```

Write to UART device. The write buffer is used until write is complete. Do not overwrite write buffer contents until the write is finished. When the write is complete (all bytes are fully transmitted on the wire), the callback called with event `UART_EVENT_TX_COMPLETE`. The maximum transfer size is reported by [infoGet\(\)](#).

Implemented as

- `R_SCI_UartWrite()`

Parameters

[in]	p_ctrl	Pointer to the UART control block.
[in]	p_src	Source address to write data to.
[in]	bytes	Write data length.

◆ **baudSet**

```
fsp_err_t(* uart_api_t::baudSet) (uart_ctrl_t *const p_ctrl, void const *const p_baudrate_info)
```

Change baud rate.

Warning

Calling this API aborts any in-progress transmission and disables reception until the new baud settings have been applied.

Implemented as

- R_SCI_UartBaudSet()

Parameters

[in]	p_ctrl	Pointer to the UART control block.
[in]	p_baudrate_info	Pointer to module specific information for configuring baud rate.

◆ **infoGet**

```
fsp_err_t(* uart_api_t::infoGet) (uart_ctrl_t *const p_ctrl, uart_info_t *const p_info)
```

Get the driver specific information.

Implemented as

- R_SCI_UartInfoGet()

Parameters

[in]	p_ctrl	Pointer to the UART control block.
[in]	baudrate	Baud rate in bps.

◆ **communicationAbort**

```
fsp_err_t(* uart_api_t::communicationAbort) (uart_ctrl_t *const p_ctrl, uart_dir_t communication_to_abort)
```

Abort ongoing transfer.

Implemented as

- R_SCI_UartAbort()

Parameters

[in]	p_ctrl	Pointer to the UART control block.
[in]	communication_to_abort	Type of abort request.

◆ **close**

```
fsp_err_t(* uart_api_t::close) (uart_ctrl_t *const p_ctrl)
```

Close UART device.

Implemented as

- R_SCI_UartClose()

Parameters

[in]	p_ctrl	Pointer to the UART control block.

◆ **versionGet**

```
fsp_err_t(* uart_api_t::versionGet) (fsp_version_t *p_version)
```

Get version.

Implemented as

- R_SCI_UartVersionGet()

Parameters

[in]	p_version	Pointer to the memory to store the version information.

◆ **uart_instance_t**

```
struct uart_instance_t
```

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields

uart_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
uart_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
uart_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ **uart_ctrl_t**typedef void [uart_ctrl_t](#)

UART control block. Allocate an instance specific control block to pass into the UART API calls.

Implemented as

- [sci_uart_instance_ctrl_t](#)

Enumeration Type Documentation◆ **uart_event_t**enum [uart_event_t](#)

UART Event codes

Enumerator

UART_EVENT_RX_COMPLETE	Receive complete event.
UART_EVENT_TX_COMPLETE	Transmit complete event.
UART_EVENT_RX_CHAR	Character received.
UART_EVENT_ERR_PARITY	Parity error event.
UART_EVENT_ERR_FRAMING	Mode fault error event.
UART_EVENT_ERR_OVERFLOW	FIFO Overflow error event.
UART_EVENT_BREAK_DETECT	Break detect error event.
UART_EVENT_TX_DATA_EMPTY	Last byte is transmitting, ready for more data.

◆ **uart_data_bits_t**enum [uart_data_bits_t](#)

UART Data bit length definition

Enumerator

UART_DATA_BITS_8	Data bits 8-bit.
UART_DATA_BITS_7	Data bits 7-bit.
UART_DATA_BITS_9	Data bits 9-bit.

◆ **uart_parity_t**

enum uart_parity_t	
UART Parity definition	
Enumerator	
UART_PARITY_OFF	No parity.
UART_PARITY_EVEN	Even parity.
UART_PARITY_ODD	Odd parity.

◆ **uart_stop_bits_t**

enum uart_stop_bits_t	
UART Stop bits definition	
Enumerator	
UART_STOP_BITS_1	Stop bit 1-bit.
UART_STOP_BITS_2	Stop bits 2-bit.

◆ **uart_dir_t**

enum uart_dir_t	
UART transaction definition	
Enumerator	
UART_DIR_RX_TX	Both RX and TX.
UART_DIR_RX	Only RX.
UART_DIR_TX	Only TX.

4.3.29 USB Interface[Interfaces](#)**Detailed Description**

Interface for USB functions.

Summary

The USB interface provides USB functionality.

The USB interface can be implemented by:

- [Universal Serial Bus \(r_usb_basic\)](#)

Data Structures

struct [usb_api_t](#)

struct [usb_instance_t](#)

Macros

#define [USB_API_VERSION_MINOR](#)

Minor version of the API.

#define [USB_API_VERSION_MAJOR](#)

Major version of the API.

#define [USB_BREQUEST](#)

b15-8

#define [USB_GET_STATUS](#)

USB Standard request Get Status.

#define [USB_CLEAR_FEATURE](#)

USB Standard request Clear Feature.

#define [USB_REQRESERVED](#)

USB Standard request Reqreserved.

#define [USB_SET_FEATURE](#)

USB Standard request Set Feature.

#define [USB_REQRESERVED1](#)

USB Standard request Reqreserved1.

#define [USB_SET_ADDRESS](#)

USB Standard request Set Address.

#define [USB_GET_DESCRIPTOR](#)

USB Standard request Get Descriptor.

#define [USB_SET_DESCRIPTOR](#)

USB Standard request Set Descriptor.

#define [USB_GET_CONFIGURATION](#)

USB Standard request Get Configuration.

#define [USB_SET_CONFIGURATION](#)

USB Standard request Set Configuration.

#define [USB_GET_INTERFACE](#)

USB Standard request Get Interface.

#define [USB_SET_INTERFACE](#)

USB Standard request Set Interface.

#define [USB_SYNCH_FRAME](#)

USB Standard request Synch Frame.

#define [USB_HOST_TO_DEV](#)

From host to device.

#define [USB_DEV_TO_HOST](#)

From device to host.

#define [USB_STANDARD](#)

Standard Request.

```
#define USB_CLASS
Class Request.
```

```
#define USB_VENDOR
Vendor Request.
```

```
#define USB_DEVICE
Device.
```

```
#define USB_INTERFACE
Interface.
```

```
#define USB_ENDPOINT
End Point.
```

```
#define USB_OTHER
Other.
```

```
#define USB_NULL
NULL pointer.
```

```
#define USB_IP0
USB0 module.
```

```
#define USB_IP1
USB1 module.
```

```
#define USB_PIPE0
Pipe Number0.
```

```
#define USB_PIPE1
Pipe Number1.
```

```
#define USB_PIPE2  
Pipe Number2.
```

```
#define USB_PIPE3  
Pipe Number3.
```

```
#define USB_PIPE4  
Pipe Number4.
```

```
#define USB_PIPE5  
Pipe Number5.
```

```
#define USB_PIPE6  
Pipe Number6.
```

```
#define USB_PIPE7  
Pipe Number7.
```

```
#define USB_PIPE8  
Pipe Number8.
```

```
#define USB_PIPE9  
Pipe Number9.
```

```
#define USB_EP0  
End Point Number0.
```

```
#define USB_EP1  
End Point Number1.
```

```
#define USB_EP2  
End Point Number2.
```

```
#define USB_EP3
```

End Point Number3.

#define [USB_EP4](#)

End Point Number4.

#define [USB_EP5](#)

End Point Number5.

#define [USB_EP6](#)

End Point Number6.

#define [USB_EP7](#)

End Point Number7.

#define [USB_EP8](#)

End Point Number8.

#define [USB_EP9](#)

End Point Number9.

#define [USB_EP10](#)

End Point Number10.

#define [USB_EP11](#)

End Point Number11.

#define [USB_EP12](#)

End Point Number12.

#define [USB_EP13](#)

End Point Number13.

#define [USB_EP14](#)

End Point Number14.

```
#define USB_EP15
        End Point Number15.
```

```
#define USB_DT_DEVICE
        Device Descriptor.
```

```
#define USB_DT_CONFIGURATION
        Configuration Descriptor.
```

```
#define USB_DT_STRING
        String Descriptor.
```

```
#define USB_DT_INTERFACE
        Interface Descriptor.
```

```
#define USB_DT_ENDPOINT
        Endpoint Descriptor.
```

```
#define USB_DT_DEVICE_QUALIFIER
        Device Qualifier Descriptor.
```

```
#define USB_DT_OTHER_SPEED_CONF
        Other Speed Configuration Descriptor.
```

```
#define USB_DT_INTERFACE_POWER
        Interface Power Descriptor.
```

```
#define USB_DT_OTGDESCRIPTOR
        OTG Descriptor.
```

```
#define USB_DT_HUBDESCRIPTOR
        HUB descriptor.
```

```
#define USB_IFCLS_NOT
Un corresponding Class.
```

```
#define USB_IFCLS_AUD
Audio Class.
```

```
#define USB_IFCLS_CDC
CDC Class.
```

```
#define USB_IFCLS_CDCC
CDC-Control Class.
```

```
#define USB_IFCLS_HID
HID Class.
```

```
#define USB_IFCLS_PHY
Physical Class.
```

```
#define USB_IFCLS_IMG
Image Class.
```

```
#define USB_IFCLS_PRN
Printer Class.
```

```
#define USB_IFCLS_MAS
Mass Storage Class.
```

```
#define USB_IFCLS_HUB
HUB Class.
```

```
#define USB_IFCLS_CDCD
CDC-Data Class.
```

```
#define USB_IFCLS_CHIP
```

Chip/Smart Card Class.

#define [USB_IFCLS_CNT](#)
Content-Security Class.

#define [USB_IFCLS_VID](#)
Video Class.

#define [USB_IFCLS_DIAG](#)
Diagnostic Device.

#define [USB_IFCLS_WIRE](#)
Wireless Controller.

#define [USB_IFCLS_APL](#)
Application-Specific.

#define [USB_IFCLS_VEN](#)
Vendor-Specific Class.

#define [USB_EP_IN](#)
In Endpoint.

#define [USB_EP_OUT](#)
Out Endpoint.

#define [USB_EP_ISO](#)
Isochronous Transfer.

#define [USB_EP_BULK](#)
Bulk Transfer.

#define [USB_EP_INT](#)
Interrupt Transfer.

```
#define USB_CF_RESERVED
Reserved(set to 1)
```

```
#define USB_CF_SELFP
Self Powered.
```

```
#define USB_CF_BUSP
Bus Powered.
```

```
#define USB_CF_RWUPON
Remote Wake up ON.
```

```
#define USB_CF_RWUPOFF
Remote Wake up OFF.
```

```
#define USB_DD_BLENGTH
Device Descriptor Length.
```

```
#define USB_CD_BLENGTH
Configuration Descriptor Length.
```

```
#define USB_ID_BLENGTH
Interface Descriptor Length.
```

```
#define USB_ED_BLENGTH
Endpoint Descriptor Length.
```

Enumerations

```
enum usb_speed_t
```

```
enum usb_setup_status_t
```

```
enum usb_status_t
```

```
enum usb_class_t
```


enum [usb_bcport_t](#)enum [usb_onoff_t](#)enum [usb_transfer_t](#)enum [usb_transfer_type_t](#)enum [usb_mode_t](#)enum [usb_compliancetest_status_t](#)

Data Structure Documentation

◆ usb_api_t

struct usb_api_t

WDT functions implemented at the HAL layer will follow this API.

Data Fields

fsp_err_t (*	open)(usb_ctrl_t *const p_api_ctrl, usb_cfg_t const *const p_cfg, usb_instance_transfer_t *p_api_trans)

fsp_err_t (*	close)(usb_ctrl_t *const p_api_ctrl, usb_instance_transfer_t *p_api_trans)

fsp_err_t (*	read)(usb_ctrl_t *const p_api_ctrl, uint8_t *p_buf, uint32_t size, usb_instance_transfer_t *p_api_trans)

fsp_err_t (*	write)(usb_ctrl_t *const p_api_ctrl, uint8_t *p_buf, uint32_t size, usb_instance_transfer_t *p_api_trans)

fsp_err_t (*	stop)(usb_ctrl_t *const p_api_ctrl, usb_transfer_t type, usb_instance_transfer_t *p_api_trans)

fsp_err_t (*	suspend)(usb_ctrl_t *const p_api_ctrl, usb_instance_transfer_t *p_api_trans)

fsp_err_t (*	resume)(usb_ctrl_t *const p_api_ctrl, usb_instance_transfer_t *p_api_trans)

fsp_err_t (*	vbusSet)(usb_ctrl_t *const p_api_ctrl, uint16_t state,

	usb_instance_transfer_t *p_api_trans)
fsp_err_t(*)	infoGet)(usb_ctrl_t *const p_api_ctrl, usb_info_t *p_info)
fsp_err_t(*)	pipeRead)(usb_ctrl_t *const p_api_ctrl, uint8_t *p_buf, uint32_t size, usb_instance_transfer_t *p_api_trans)
fsp_err_t(*)	pipeWrite)(usb_ctrl_t *const p_api_ctrl, uint8_t *p_buf, uint32_t size, usb_instance_transfer_t *p_api_trans)
fsp_err_t(*)	pipeStop)(usb_ctrl_t *const p_api_ctrl, usb_instance_transfer_t *p_api_trans)
fsp_err_t(*)	usedPipesGet)(usb_ctrl_t *const p_api_ctrl, uint16_t *p_pipe)
fsp_err_t(*)	pipeInfoGet)(usb_ctrl_t *const p_api_ctrl, usb_pipe_t *p_info)
fsp_err_t(*)	versionGet)(fsp_version_t *const p_version)
fsp_err_t(*)	eventGet)(usb_ctrl_t *const p_api_ctrl, usb_status_t *event)
fsp_err_t(*)	pullup)(uint8_t state)
fsp_err_t(*)	modulenumberget)(usb_ctrl_t *const p_api_ctrl, uint8_t *module_number)
fsp_err_t(*)	classtypeget)(usb_ctrl_t *const p_api_ctrl, usb_class_t *class_type)
fsp_err_t(*)	deviceaddressget)(usb_ctrl_t *const p_api_ctrl, uint8_t *device_address)
fsp_err_t(*)	pipenumberget)(usb_ctrl_t *const p_api_ctrl, uint8_t *pipe_number)
fsp_err_t(*)	devicestateget)(usb_ctrl_t *const p_api_ctrl, uint16_t *state)
fsp_err_t(*)	datasizeget)(usb_ctrl_t *const p_api_ctrl, uint32_t *data_size)

```
fsp_err_t(* setupget )(usb_ctrl_t *const p_api_ctrl, usb_setup_t *setup)
```

Field Documentation

◆ open

```
fsp_err_t(* usb_api_t::open) (usb_ctrl_t *const p_api_ctrl, usb_cfg_t const *const p_cfg,  
usb_instance_transfer_t *p_api_trans)
```

Start the USB module

Implemented as

- R_USB_Open()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_cfg	Pointer to configuration structure.
[in]	p_api_trans	pointer to transfer structure.

◆ close

```
fsp_err_t(* usb_api_t::close) (usb_ctrl_t *const p_api_ctrl, usb_instance_transfer_t *p_api_trans)
```

Stop the USB module

Implemented as

- R_USB_Close()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_api_trans	pointer to transfer structure.

◆ read

```
fsp_err_t(*usb_api_t::read)(usb_ctrl_t *const p_api_ctrl, uint8_t *p_buf, uint32_t size,
usb_instance_transfer_t *p_api_trans)
```

Request USB data read

Implemented as

◦ [R_USB_Read\(\)](#)

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_buf	Pointer to area that stores read data.
[in]	size	Read request size.
[in]	p_api_trans	pointer to transfer structure.

◆ write

```
fsp_err_t(*usb_api_t::write)(usb_ctrl_t *const p_api_ctrl, uint8_t *p_buf, uint32_t size,
usb_instance_transfer_t *p_api_trans)
```

Request USB data write

Implemented as

◦ [R_USB_Write\(\)](#)

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_buf	Pointer to area that stores write data.
[in]	size	Read request size.
[in]	p_api_trans	pointer to transfer structure.

◆ stop

```
fsp_err_t(* usb_api_t::stop) (usb_ctrl_t *const p_api_ctrl, usb_transfer_t type,
usb_instance_transfer_t *p_api_trans)
```

Stop USB data read/write processing

Implemented as

◦ R_USB_Stop()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	type	Receive (USB_READ) or send (USB_WRITE).
[in]	p_api_trans	pointer to transfer structure.

◆ suspend

```
fsp_err_t(* usb_api_t::suspend) (usb_ctrl_t *const p_api_ctrl, usb_instance_transfer_t *p_api_trans)
```

Request suspend

Implemented as

◦ R_USB_Suspend()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_api_trans	pointer to transfer structure.

◆ resume

```
fsp_err_t(* usb_api_t::resume) (usb_ctrl_t *const p_api_ctrl, usb_instance_transfer_t *p_api_trans)
```

Request resume

Implemented as

◦ R_USB_Resume()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_api_trans	pointer to transfer structure.

◆ **vbusSet**

```
fsp_err_t(*usb_api_t::vbusSet)(usb_ctrl_t *const p_api_ctrl, uint16_t state, usb_instance_transfer_t *p_api_trans)
```

Sets VBUS supply start/stop.

Implemented as

◦ R_USB_VbusSet()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	state	VBUS supply start/stop specification
[in]	p_api_trans	pointer to transfer structure.

◆ **infoGet**

```
fsp_err_t(*usb_api_t::infoGet)(usb_ctrl_t *const p_api_ctrl, usb_info_t *p_info)
```

Get information on USB device.

Implemented as

◦ R_USB_InfomationGet()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_info	Pointer to usb_info_t structure area.

◆ **pipeRead**

```
fsp_err_t(*usb_api_t::pipeRead)(usb_ctrl_t *const p_api_ctrl, uint8_t *p_buf, uint32_t size, usb_instance_transfer_t *p_api_trans)
```

Request data read from specified pipe

Implemented as

◦ R_USB_PipeRead()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_buf	Pointer to area that stores read data.
[in]	size	Read request size.
[in]	p_api_trans	pointer to transfer structure.

◆ **pipeWrite**

```
fsp_err_t(*usb_api_t::pipeWrite)(usb_ctrl_t *const p_api_ctrl, uint8_t *p_buf, uint32_t size,
usb_instance_transfer_t *p_api_trans)
```

Request data write to specified pipe

Implemented as

◦ [R_USB_PipeWrite\(\)](#)

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_buf	Pointer to area that stores write data.
[in]	size	Read request size.
[in]	p_api_trans	pointer to transfer structure.

◆ **pipeStop**

```
fsp_err_t(*usb_api_t::pipeStop)(usb_ctrl_t *const p_api_ctrl, usb_instance_transfer_t *p_api_trans)
```

Stop USB data read/write processing to specified pipe

Implemented as

◦ [R_USB_PipeStop\(\)](#)

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_api_trans	pointer to transfer structure.

◆ **usedPipesGet**

```
fsp_err_t(*usb_api_t::usedPipesGet)(usb_ctrl_t *const p_api_ctrl, uint16_t *p_pipe)
```

Get pipe number

Implemented as

◦ [R_USB_UsedPipesGet\(\)](#)

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_pipe	Pointer to area that stores the selected pipe number (bit map information).

◆ **pipeInfoGet**

```
fsp_err_t(* usb_api_t::pipeInfoGet) (usb_ctrl_t *const p_api_ctrl, usb_pipe_t *p_info)
```

Get pipe information

Implemented as

◦ [R_USB_PipeInfoGet\(\)](#)

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_info	Pointer to usb_pipe_t structure area.

◆ **versionGet**

```
fsp_err_t(* usb_api_t::versionGet) (fsp_version_t *const p_version)
```

Get the driver version

Implemented as

◦ [R_USB_VersionGet\(\)](#)

Parameters

[out]	version	Version number.

◆ **eventGet**

```
fsp_err_t(* usb_api_t::eventGet) (usb_ctrl_t *const p_api_ctrl, usb_status_t *event)
```

Return USB-related completed events (Non-OS only)

Implemented as

◦ [R_USB_EventGet\(\)](#)

Parameters

[in]	p_ctrl	Pointer to control structure.
[out]	event	Pointer to event.

◆ **pullup**

`fsp_err_t(* usb_api_t::pullup) (uint8_t state)`

Pull-up enable/disable setting of D+/D- line.

Implemented as

◦ `R_USB_PullUp()`

Parameters

[in]	state	Pull-up enable/disable setting.

◆ **modulenumberget**

`fsp_err_t(* usb_api_t::modulenumberget) (usb_ctrl_t *const p_api_ctrl, uint8_t *module_number)`

This API gets the module number.

Implemented as

◦ `R_USB_ModuleNumberGet()`

Parameters

[in]	p_api_ctrl	USB control structure.
[out]	module_number	Module number to get.

◆ **classtypeget**

`fsp_err_t(* usb_api_t::classtypeget) (usb_ctrl_t *const p_api_ctrl, usb_class_t *class_type)`

This API gets the module number.

Implemented as

◦ `R_USB_ClassTypeGet()`

Parameters

[in]	p_api_ctrl	USB control structure.
[out]	class_type	Class type to get.

◆ **deviceaddressget**

```
fsp_err_t(* usb_api_t::deviceaddressget) (usb_ctrl_t *const p_api_ctrl, uint8_t *device_address)
```

This API gets the device address.

Implemented as

- [R_USB_DeviceAddressGet\(\)](#)

Parameters

[in]	p_api_ctrl	USB control structure.
[out]	device_address	device address to get.

◆ **pipenumberget**

```
fsp_err_t(* usb_api_t::pipenumberget) (usb_ctrl_t *const p_api_ctrl, uint8_t *pipe_number)
```

This API gets the pipe number.

Implemented as

- [R_USB_PipeNumberGet\(\)](#)

Parameters

[in]	p_api_ctrl	USB control structure.
[out]	pipe_number	Pipe number to get.

◆ **devicestateget**

```
fsp_err_t(* usb_api_t::devicestateget) (usb_ctrl_t *const p_api_ctrl, uint16_t *state)
```

This API gets the state of the device.

Implemented as

- [R_USB_DeviceStateGet\(\)](#)

Parameters

[in]	p_api_ctrl	USB control structure.
[out]	state	device state to get.

◆ **datasizeget**

```
fsp_err_t(* usb_api_t::datasizeget) (usb_ctrl_t *const p_api_ctrl, uint32_t *data_size)
```

This API gets the data size.

Implemented as

- [R_USB_DataSizeGet\(\)](#)

Parameters

[in]	p_api_ctrl	USB control structure.
[out]	data_size	Data size to get.

◆ **setupget**

```
fsp_err_t(* usb_api_t::setupget) (usb_ctrl_t *const p_api_ctrl, usb_setup_t *setup)
```

This API gets the setup type.

Implemented as

- [R_USB_SetupGet\(\)](#)

Parameters

[in]	p_api_ctrl	USB control structure.
[out]	setup	Setup type to get.

◆ **usb_instance_t**

```
struct usb_instance_t
```

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields

usb_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
usb_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
usb_api_t const *	p_api	Pointer to the API structure for this instance.

Enumeration Type Documentation

◆ **usb_speed_t**

enum usb_speed_t	
USB speed type	
Enumerator	
USB_SPEED_LS	Low speed operation.
USB_SPEED_FS	Full speed operation.
USB_SPEED_HS	Hi speed operation.

◆ **usb_setup_status_t**

enum usb_setup_status_t	
USB request result	
Enumerator	
USB_SETUP_STATUS_ACK	ACK response.
USB_SETUP_STATUS_STALL	STALL response.

◆ **usb_status_t**

enum usb_status_t	
USB driver status	
Enumerator	
USB_STATUS_POWERED	Powered State.
USB_STATUS_DEFAULT	Default State.
USB_STATUS_ADDRESS	Address State.
USB_STATUS_CONFIGURED	Configured State.
USB_STATUS_SUSPEND	Suspend State.
USB_STATUS_RESUME	Resume State.
USB_STATUS_DETACH	Detach State.
USB_STATUS_REQUEST	Request State.
USB_STATUS_REQUEST_COMPLETE	Request Complete State.
USB_STATUS_READ_COMPLETE	Read Complete State.
USB_STATUS_WRITE_COMPLETE	Write Complete State.
USB_STATUS_BC	battery Charge State
USB_STATUS_OVERCURRENT	Over Current state.
USB_STATUS_NOT_SUPPORT	Device Not Support.
USB_STATUS_NONE	None Status.
USB_STATUS_MSC_CMD_COMPLETE	MSC_CMD Complete.

◆ **usb_class_t**

enum usb_class_t	
USB class type	
Enumerator	
USB_CLASS_PCDC	PCDC Class.
USB_CLASS_PCDCC	PCDCC Class.
USB_CLASS_PHID	PHID Class.
USB_CLASS_PVND	PVND Class.
USB_CLASS_HCDC	HCDC Class.
USB_CLASS_HCDCC	HCDCC Class.
USB_CLASS_HHID	HHID Class.
USB_CLASS_HVND	HVND Class.
USB_CLASS_HMSC	HMSC Class.
USB_CLASS_PMSC	PMSC Class.
USB_CLASS_REQUEST	USB Class Request.

◆ **usb_bcport_t**

enum usb_bcport_t	
USB battery charging type	
Enumerator	
USB_BCPORT_SDP	SDP port settings.
USB_BCPORT_CDP	CDP port settings.
USB_BCPORT_DCP	DCP port settings.

◆ **usb_onoff_t**

enum <code>usb_onoff_t</code>	
USB status	
Enumerator	
USB_OFF	USB Off State.
USB_ON	USB On State.

◆ **usb_transfer_t**

enum <code>usb_transfer_t</code>	
USB read / write type	
Enumerator	
USB_TRANSFER_READ	Data Receive communication.
USB_TRANSFER_WRITE	Data transmission communication.

◆ **usb_transfer_type_t**

enum <code>usb_transfer_type_t</code>	
USB transfer type	
Enumerator	
USB_TRANSFER_TYPE_BULK	Bulk communication.
USB_TRANSFER_TYPE_INT	Interrupt communication.
USB_TRANSFER_TYPE_ISO	Isochronous communication.

◆ **usb_mode_t**

enum <code>usb_mode_t</code>	
Enumerator	
USB_MODE_HOST	Host mode.
USB_MODE_PERI	Peripheral mode.

◆ **usb_compliancetest_status_t**

enum <code>usb_compliancetest_status_t</code>	
Enumerator	
<code>USB_COMPLIANCETEST_ATTACH</code>	Device Attach Detection.
<code>USB_COMPLIANCETEST_DETACH</code>	Device Detach Detection.
<code>USB_COMPLIANCETEST_TPL</code>	TPL device connect.
<code>USB_COMPLIANCETEST_NOTTPL</code>	Not TPL device connect.
<code>USB_COMPLIANCETEST_HUB</code>	USB Hub connect.
<code>USB_COMPLIANCETEST_OVRC</code>	Over current.
<code>USB_COMPLIANCETEST_NORES</code>	Response Time out for Control Read Transfer.
<code>USB_COMPLIANCETEST_SETUP_ERR</code>	Setup Transaction Error.

4.3.30 USB HMSC Interface[Interfaces](#)**Detailed Description**

Interface for USB HMSC functions.

Summary

The USB HMSC interface provides USB HMSC functionality.

The USB HMSC interface can be implemented by:

- [Host Mass Storage Class Driver \(r_usb_hmsc\)](#)

Data Structures

```
struct usb\_hmsc\_api\_t
```

Enumerations

```
enum usb\_atapi\_t
```

```
enum usb\_csw\_result\_t
```


Data Structure Documentation

usb_hmsc_api_t

struct usb_hmsc_api_t

WDT functions implemented at the HAL layer will follow this API.

Data Fields

fsp_err_t(*	strgcmd	)(usb_ctrl_t *const p_api_ctrl, uint8_t *buf, uint16_t command, usb_instance_transfer_t *p_api_trans)
fsp_err_t(*	drivenoget	)(usb_ctrl_t *const p_api_ctrl, uint8_t *p_drive, usb_instance_transfer_t *p_api_trans)
fsp_err_t(*	drive2addr	)(uint16_t side, usb_utr_t *devadr)

Field Documentation

strgcmd

fsp_err_t(* usb_hmsc_api_t::strgcmd) (usb_ctrl_t *const p_api_ctrl, uint8_t *buf, uint16_t command, usb_instance_transfer_t *p_api_trans)

Start the USB_HMSC module

Implemented as

◦ R_USB_HmscStrgCmd()

Parameters

[in]	p_api_ctrl	Pointer to control structure.
[in]	buf	Pointer to the buffer area to store the transfer data.
[in]	command	ATAPI command.
[in]	p_api_trans	pointer to transfer structure.

◆ **drivenoget**

```
fsp_err_t(*usb_hmsc_api_t::drivenoget)(usb_ctrl_t *const p_api_ctrl, uint8_t *p_drive,
usb_instance_transfer_t *p_api_trans)
```

Stop the USB_HMSC module

Implemented as

- R_USB_HmscDriveNoGet()

Parameters

[in]	p_api_ctrl	Pointer to control structure.
[out]	p_drive	Store address for Drive No.
[in]	p_api_trans	pointer to transfer structure.

◆ **drive2addr**

```
fsp_err_t(*usb_hmsc_api_t::drive2addr)(uint16_t side, usb_utr_t *devadr)
```

Retrieves device address

Implemented as

- R_USB_HmscSmpDrive2Addr()

Parameters

[in]	side	Drive number.
[out]	devadr	Pointer to usb_utr_t structure.

Enumeration Type Documentation

◆ **usb_atapi_t**

enum usb_atapi_t	
Enumerator	
USB_ATAPI_TEST_UNIT_READY	ATAPI command Test Unit Ready.
USB_ATAPI_REQUEST_SENSE	ATAPI command Request Sense.
USB_ATAPI_FORMAT_UNIT	ATAPI command Format Unit.
USB_ATAPI_INQUIRY	ATAPI command Inquiry.
USB_ATAPI_MODE_SELECT6	ATAPI command Mode Select6.
USB_ATAPI_MODE_SENSE6	ATAPI command Mode Sense6.
USB_ATAPI_START_STOP_UNIT	ATAPI command Start Stop Unit.
USB_ATAPI_PREVENT_ALLOW	ATAPI command Prevent Allow.
USB_ATAPI_READ_FORMAT_CAPACITY	ATAPI command Read Format Capacity.
USB_ATAPI_READ_CAPACITY	ATAPI command Read Capacity.
USB_ATAPI_READ10	ATAPI command Read10.
USB_ATAPI_WRITE10	ATAPI command Write10.
USB_ATAPI_SEEK	ATAPI command Seek.
USB_ATAPI_WRITE_AND_VERIFY	ATAPI command Write and Verify.
USB_ATAPI_VERIFY10	ATAPI command Verify10.
USB_ATAPI_MODE_SELECT10	ATAPI command Mode Select10.
USB_ATAPI_MODE_SENSE10	ATAPI command Mode Sense10.

◆ **usb_csw_result_t**

enum usb_csw_result_t	
Enumerator	
USB_CSW_RESULT_SUCCESS	CSW was successful.
USB_CSW_RESULT_FAIL	CSW failed.
USB_CSW_RESULT_PHASE	CSW has phase error.

4.3.31 USB PCDC Interface[Interfaces](#)**Detailed Description**

Interface for USB PCDC functions.

Summary

The USB interface provides USB functionality.

The USB PCDC interface can be implemented by:

- [Universal Serial Bus Peripheral Communication Device Class \(r_usb_pcdc\)](#)

Macros

```
#define USB_PCDC_SET_LINE_CODING
Command code for Set Line Codeing.
```

```
#define USB_PCDC_GET_LINE_CODING
Command code for Get Line Codeing.
```

```
#define USB_PCDC_SET_CONTROL_LINE_STATE
Command code for Control Line State.
```

```
#define USB_PCDC_SERIAL_STATE
Serial State Code.
```

```
#define USB_PCDC_SETUP_TBL_BSIZE  
    setup packet table size (uint16_t * 5)
```

4.3.32 WDT Interface

Interfaces

Detailed Description

Interface for watch dog timer functions.

Summary

The WDT interface for the Watchdog Timer (WDT) peripheral provides watchdog functionality including resetting the device or generating an interrupt.

The watchdog timer interface can be implemented by:

- [Watchdog Timer \(r_wdt\)](#)
- [Independent Watchdog Timer \(r_iwdt\)](#)

Data Structures

struct	wdt_callback_args_t

struct	wdt_timeout_values_t

struct	wdt_cfg_t

struct	wdt_api_t

struct	wdt_instance_t

Typedefs

typedef void	wdt_ctrl_t

Enumerations

enum	wdt_timeout_t

enum	wdt_clock_division_t

enum	wdt_window_start_t

enum	wdt_window_end_t

enum [wdt_reset_control_t](#)enum [wdt_stop_control_t](#)enum [wdt_status_t](#)

Data Structure Documentation

◆ [wdt_callback_args_t](#)

struct wdt_callback_args_t		
Callback function parameter data		
Data Fields		
void const *	p_context	Placeholder for user data. Set in wdt_api_t::open function in wdt_cfg_t .

◆ [wdt_timeout_values_t](#)

struct wdt_timeout_values_t		
WDT timeout data. Used to return frequency of WDT clock and timeout period		
Data Fields		
uint32_t	clock_frequency_hz	Frequency of watchdog clock after divider.
uint32_t	timeout_clocks	Timeout period in units of watchdog clock ticks.

◆ [wdt_cfg_t](#)

struct wdt_cfg_t	
WDT configuration parameters.	
Data Fields	
wdt_timeout_t	timeout
	Timeout period.
wdt_clock_division_t	clock_division
	Clock divider.
wdt_window_start_t	window_start
	Refresh permitted window start position.

wdt_window_end_t	window_end
	Refresh permitted window end position.
wdt_reset_control_t	reset_control
	Select NMI or reset generated on underflow.
wdt_stop_control_t	stop_control
	Select whether counter operates in sleep mode.
<code>void(*</code>	p_callback <code>)(wdt_callback_args_t *p_args)</code>
	Callback provided when a WDT NMI ISR occurs.
<code>void const *</code>	p_context
<code>void const *</code>	p_extend
	Placeholder for user extension.

Field Documentation

◆ [p_context](#)

`void const* wdt_cfg_t::p_context`

Placeholder for user data. Passed to the user callback in [wdt_callback_args_t](#).

◆ [wdt_api_t](#)

`struct wdt_api_t`

WDT functions implemented at the HAL layer will follow this API.

Data Fields

[fsp_err_t](#)(* [open](#))([wdt_ctrl_t](#) *const p_ctrl, [wdt_cfg_t](#) const *const p_cfg)

[fsp_err_t](#)(* [refresh](#))([wdt_ctrl_t](#) *const p_ctrl)

<code>fsp_err_t(*</code>	<code>statusGet)(wdt_ctrl_t *const p_ctrl, wdt_status_t *const p_status)</code>
<code>fsp_err_t(*</code>	<code>statusClear)(wdt_ctrl_t *const p_ctrl, const wdt_status_t status)</code>
<code>fsp_err_t(*</code>	<code>counterGet)(wdt_ctrl_t *const p_ctrl, uint32_t *const p_count)</code>
<code>fsp_err_t(*</code>	<code>timeoutGet)(wdt_ctrl_t *const p_ctrl, wdt_timeout_values_t *const p_timeout)</code>
<code>fsp_err_t(*</code>	<code>versionGet)(fsp_version_t *const p_data)</code>

Field Documentation

◆ open

`fsp_err_t(* wdt_api_t::open) (wdt_ctrl_t *const p_ctrl, wdt_cfg_t const *const p_cfg)`

Initialize the WDT in register start mode. In auto-start mode with NMI output it registers the NMI callback.

Implemented as

- `R_WDT_Open()`
- `R_IWDT_Open()`

Parameters

[in]	<code>p_ctrl</code>	Pointer to control structure.
[in]	<code>p_cfg</code>	Pointer to pin configuration structure.

◆ refresh

`fsp_err_t(* wdt_api_t::refresh) (wdt_ctrl_t *const p_ctrl)`

Refresh the watchdog timer.

Implemented as

- `R_WDT_Refresh()`
- `R_IWDT_Refresh()`

Parameters

[in]	<code>p_ctrl</code>	Pointer to control structure.

◆ **statusGet**

```
fsp_err_t(* wdt_api_t::statusGet) (wdt_ctrl_t *const p_ctrl, wdt_status_t *const p_status)
```

Read the status of the WDT.

Implemented as

- R_WDT_StatusGet()
- R_IWDT_StatusGet()

Parameters

[in]	p_ctrl	Pointer to control structure.
[out]	p_status	Pointer to variable to return status information through.

◆ **statusClear**

```
fsp_err_t(* wdt_api_t::statusClear) (wdt_ctrl_t *const p_ctrl, const wdt_status_t status)
```

Clear the status flags of the WDT.

Implemented as

- R_WDT_StatusClear()
- R_IWDT_StatusClear()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	status	Status condition(s) to clear.

◆ **counterGet**

```
fsp_err_t(* wdt_api_t::counterGet) (wdt_ctrl_t *const p_ctrl, uint32_t *const p_count)
```

Read the current WDT counter value.

Implemented as

- R_WDT_CounterGet()
- R_IWDT_CounterGet()

Parameters

[in]	p_ctrl	Pointer to control structure.
[out]	p_count	Pointer to variable to return current WDT counter value.

◆ **timeoutGet**

```
fsp_err_t(* wdt_api_t::timeoutGet) (wdt_ctrl_t *const p_ctrl, wdt_timeout_values_t *const p_timeout)
```

Read the watchdog timeout values.

Implemented as

- R_WDT_TimeoutGet()
- R_IWDT_TimeoutGet()

Parameters

[in]	p_ctrl	Pointer to control structure.
[out]	p_timeout	Pointer to structure to return timeout values.

◆ **versionGet**

```
fsp_err_t(* wdt_api_t::versionGet) (fsp_version_t *const p_data)
```

Return the version of the driver.

Implemented as

- R_WDT_VersionGet()
- R_IWDT_VersionGet()

Parameters

[in]	p_ctrl	Pointer to control structure.
[out]	p_data	Memory address to return version information to.

◆ **wdt_instance_t**

```
struct wdt_instance_t
```

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields

wdt_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
wdt_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
wdt_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ **wdt_ctrl_t**typedef void [wdt_ctrl_t](#)

WDT control block. Allocate an instance specific control block to pass into the WDT API calls.

Implemented as

- [wdt_instance_ctrl_t](#)
- [iwdt_instance_ctrl_t](#)

Enumeration Type Documentation◆ **wdt_timeout_t**enum [wdt_timeout_t](#)

WDT time-out periods.

Enumerator

WDT_TIMEOUT_128	128 clock cycles
WDT_TIMEOUT_512	512 clock cycles
WDT_TIMEOUT_1024	1024 clock cycles
WDT_TIMEOUT_2048	2048 clock cycles
WDT_TIMEOUT_4096	4096 clock cycles
WDT_TIMEOUT_8192	8192 clock cycles
WDT_TIMEOUT_16384	16384 clock cycles

◆ **wdt_clock_division_t**

enum wdt_clock_division_t	
WDT clock division ratio.	
Enumerator	
WDT_CLOCK_DIVISION_1	CLK/1.
WDT_CLOCK_DIVISION_4	CLK/4.
WDT_CLOCK_DIVISION_16	CLK/16.
WDT_CLOCK_DIVISION_32	CLK/32.
WDT_CLOCK_DIVISION_64	CLK/64.
WDT_CLOCK_DIVISION_128	CLK/128.
WDT_CLOCK_DIVISION_256	CLK/256.
WDT_CLOCK_DIVISION_512	CLK/512.
WDT_CLOCK_DIVISION_2048	CLK/2048.
WDT_CLOCK_DIVISION_8192	CLK/8192.

◆ **wdt_window_start_t**

enum wdt_window_start_t	
WDT refresh permitted period window start position.	
Enumerator	
WDT_WINDOW_START_25	Start position = 25%.
WDT_WINDOW_START_50	Start position = 50%.
WDT_WINDOW_START_75	Start position = 75%.
WDT_WINDOW_START_100	Start position = 100%.

◆ **wdt_window_end_t**

enum wdt_window_end_t	
WDT refresh permitted period window end position.	
Enumerator	
WDT_WINDOW_END_75	End position = 75%.
WDT_WINDOW_END_50	End position = 50%.
WDT_WINDOW_END_25	End position = 25%.
WDT_WINDOW_END_0	End position = 0%.

◆ **wdt_reset_control_t**

enum wdt_reset_control_t	
WDT Counter underflow and refresh error control.	
Enumerator	
WDT_RESET_CONTROL_NMI	NMI request when counter underflows.
WDT_RESET_CONTROL_RESET	Reset request when counter underflows.

◆ **wdt_stop_control_t**

enum wdt_stop_control_t	
WDT Counter operation in sleep mode.	
Enumerator	
WDT_STOP_CONTROL_DISABLE	Count will not stop when device enters sleep mode.
WDT_STOP_CONTROL_ENABLE	Count will automatically stop when device enters sleep mode.

◆ **wdt_status_t**

enum <code>wdt_status_t</code>	
WDT status	
Enumerator	
<code>WDT_STATUS_NO_ERROR</code>	No status flags set.
<code>WDT_STATUS_UNDERFLOW_ERROR</code>	Underflow flag set.
<code>WDT_STATUS_REFRESH_ERROR</code>	Refresh error flag set. Refresh outside of permitted window.
<code>WDT_STATUS_UNDERFLOW_AND_REFRESH_ERROR</code>	Underflow and refresh error flags set.

4.3.33 Touch Middleware Interface[Interfaces](#)**Detailed Description**

Interface for Touch Middleware functions.

Summary

The TOUCH interface provides TOUCH functionality.

The TOUCH interface can be implemented by:

- [Capacitive Touch Middleware \(rm_touch\)](#)

Data Structures

struct	touch_button_cfg_t
struct	touch_slider_cfg_t
struct	touch_wheel_cfg_t
struct	touch_cfg_t
struct	touch_api_t
struct	touch_instance_t

Typedefs

```
typedef void touch_ctrl_t
```

```
typedef struct touch_callback_args_t
st_ctsu_callback_args
```

Data Structure Documentation

◆ touch_button_cfg_t

struct touch_button_cfg_t		
Configuration of each button		
Data Fields		
uint8_t	elem_index	Element number used by this button.
uint16_t	threshold	Touch/non-touch judgment threshold.
uint16_t	hysteresis	Threshold hysteresis for chattering prevention.

◆ touch_slider_cfg_t

struct touch_slider_cfg_t		
Configuration of each slider		
Data Fields		
uint8_t const *	p_elem_index	Element number array used by this slider.
uint8_t	num_elements	Number of elements used by this slider.
uint16_t	threshold	Position calculation start threshold value.

◆ touch_wheel_cfg_t

struct touch_wheel_cfg_t		
Configuration of each wheel		
Data Fields		
uint8_t const *	p_elem_index	Element number array used by this wheel.
uint8_t	num_elements	Number of elements used by this wheel.
uint16_t	threshold	Position calculation start threshold value.

◆ touch_cfg_t

struct touch_cfg_t		
User configuration structure, used in open function		
Data Fields		
touch_button_cfg_t const *	p_buttons	Pointer to array of button configuration.
touch_slider_cfg_t const *	p_sliders	Pointer to array of slider configuration.
touch_wheel_cfg_t const *	p_wheels	Pointer to array of wheel configuration.
uint8_t	num_buttons	Number of buttons.
uint8_t	num_sliders	Number of sliders.
uint8_t	num_wheels	Number of wheels.
uint8_t	on_freq	The cumulative number of determinations of ON.
uint8_t	off_freq	The cumulative number of determinations of OFF.
uint16_t	drift_freq	Base value drift frequency. [0 : no use].
uint16_t	cancel_freq	Maximum continuous ON. [0 : no use].
ctsu_instance_t const *	p_ctsu_instance	Pointer to CTSU instance.
void const *	p_context	User defined context passed into callback function.
void const *	p_extend	Pointer to extended configuration by instance of interface.

◆ touch_api_t

struct touch_api_t	
Functions implemented at the HAL layer will follow this API.	
Data Fields	
fsp_err_t(*	open)(touch_ctrl_t *const p_ctrl, touch_cfg_t const *const p_cfg)
fsp_err_t(*	scanStart)(touch_ctrl_t *const p_ctrl)
fsp_err_t(*	dataGet)(touch_ctrl_t *const p_ctrl, uint64_t *p_button_status, uint16_t *p_slider_position, uint16_t *p_wheel_position)

<code>fsp_err_t(*</code>	<code>close)(touch_ctrl_t *const p_ctrl)</code>
<code>fsp_err_t(*</code>	<code>versionGet)(fsp_version_t *const p_data)</code>

Field Documentation

◆ open

<code>fsp_err_t(* touch_api_t::open) (touch_ctrl_t *const p_ctrl, touch_cfg_t const *const p_cfg)</code>

Open driver.

Implemented as

- RM_TOUCH_Open()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_cfg	Pointer to pin configuration structure.

◆ scanStart

<code>fsp_err_t(* touch_api_t::scanStart) (touch_ctrl_t *const p_ctrl)</code>

Scan start.

Implemented as

- RM_TOUCH_ScanStart()

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ **dataGet**

```
fsp_err_t(* touch_api_t::dataGet) (touch_ctrl_t *const p_ctrl, uint64_t *p_button_status, uint16_t *p_slider_position, uint16_t *p_wheel_position)
```

Data get.

Implemented as

◦ [RM_TOUCH_DataGet\(\)](#)

Parameters

[in]	p_ctrl	Pointer to control structure.
[out]	p_button_status	Pointer to get data bitmap.
[out]	p_slider_position	Pointer to get data array.
[out]	p_wheel_position	Pointer to get data array.

◆ **close**

```
fsp_err_t(* touch_api_t::close) (touch_ctrl_t *const p_ctrl)
```

Close driver.

Implemented as

◦ [RM_TOUCH_Close\(\)](#)

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ **versionGet**

```
fsp_err_t(* touch_api_t::versionGet) (fsp_version_t *const p_data)
```

Return the version of the driver.

Implemented as

◦ [RM_TOUCH_VersionGet\(\)](#)

Parameters

[in]	p_ctrl	Pointer to control structure.
[out]	p_data	Memory address to return version information to.

◆ **touch_instance_t**

```
struct touch_instance_t
```

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields

touch_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
touch_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
touch_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ touch_ctrl_t

```
typedef void touch\_ctrl\_t
```

Control block. Allocate an instance specific control block to pass into the API calls.

Implemented as

- [touch_instance_ctrl_t](#)

◆ touch_callback_args_t

```
typedef struct st_ctsu_callback_args touch\_callback\_args\_t
```

Callback function parameter data

Renesas FSP
Copyright © (2019) Renesas Electronics Corporation. All Rights Reserved.

User's Manual

Publication Date: Revision 0.81 Nov.08.19

Renesas FSP v0.80
User's Manual