RSL15 Developer's Guide

M-20887-003
September 2023

Table of Contents

CHAPTER 1

Introduction

1.1 SUMMARY

IMPORTANT: onsemi plans to lead in replacing the terms “white list", "master" and "slave” as noted in this product release. We have a plan to work with other companies to identify an industry wide solution that can eradicate non-inclusive terminology but maintains the technical relationship of the original wording. Once new terminologies are agreed upon, we will update all documentation live on the website and in all future released documents.

The *RSL15 Developer's Guide* is for those who are designing software and firmware to run on RSL15. This group of topics serves as a guide for creating your own applications, based on samples or from scratch, and offers tips, tricks and best practices for developing with RSL15.

This group of topics is intended to be used in conjunction with the *RSL15 Firmware Reference*, *RSL15 Hardware Reference*, and the provided third-party documentation, after a user has completed the startup steps outlined in the *RSL15 Getting Started Guide*.

1.2 DOCUMENT CONVENTIONS

The following typographical conventions are used in this documentation:

`monospace font`	Assembly code, macros, functions, registers, defines and addresses.
italics	File and path names, or any portion of them.
`<angle brackets and bold>`	Optional parameters and placeholders for specific information. To use an optional parameter or replace a placeholder, specify the information within the brackets; do not include the brackets themselves.
Bold	GUI items (text that can be seen on a screen).

Note, Important, Caution, Warning

Information requiring special notice is presented in several attention-grabbing formats depending on the consequences of ignoring the information:

NOTE: Significant supplemental information, hints, or tips.

IMPORTANT: Information that is more significant than a Note; intended to help you avoid frustration.

CAUTION: Information that can prevent you from damaging equipment or software.

WARNING: Information that can prevent harm to humans.

Registers:

Registers are shown in `monospace font` using their full descriptors, depending on which core the register is accessing. The full description takes the form **`<PREFIX><GROUP>_<REGISTER>`**.

All registers are accessible from the Arm Cortex-M33 processor.

A register prefix of `D_` is used in the following circumstances:

- In cases where there are multiple instances of a block of registers, the summary of the registers at the beginning of the Register section have slightly different names from the detailed register sections below that table. For example, the `DMA*_CFG0` registers are referred to as `DMA_CFG0` when we are defining bit-fields and settings.

The firmware provides access to these registers in two ways:

- In the flat header files (e.g.: *sk5_hw_flat_cid*.h*), each register is individually accessible by directly using the naming provided in this manual. This is helpful for assembly and low-level C programming.
- In the normal header files (e.g.: *sk5_hw_cid*.h*), each register group forms a structure, with the registers being defined as members within that structure. The structures defined by these header files provide access to registers under the naming conventions `PREFIX_GROUP->REGISTER` (for the structure) and `GROUP->REGISTER` (for the register).
- For more information, see the Hardware Definitions chapter of the *RSL15 Firmware Reference*.

Default settings for registers and bit fields are marked with an asterisk (*).

Any undefined bits must be written to 0, if they are written at all.

Numbers

In general, numbers are presented in decimal notation. In cases where hexadecimal or binary notation is more convenient, these numbers are identified by the prefixes "0x" and "0b" respectively. For example, the decimal number 123456 can also be represented as 0x1E240 or 0b11110001001000000.

Sample Rates

All sample rates specified are the final decimated sample rates, unless stated otherwise.

1.3 FURTHER READING

The following documents are installed with the RSL15 system, in the default location *C:/Users/*`**<your_user_name>**`*/AppData/Local/Arm/Packs/ONSemiconductor/RSL15/*`**<version_number>**`*/documentation*. These manuals are available only in PDF format:

- *Arm TrustZone CryptoCell-312 Software Developers Manual*
- multiple CEVA manuals in the */ceva* folder

For even more information, consult these publicly-available documents:

- *Armv8M Architecture Reference Manual* (PDF download available from https://developer.arm.com/documentation/ddi0553/latest).
- *Arm Cortex-M33 Processor Technical Reference Manual*, revision r1p0, from https://developer.arm.com/documentation/100230/0100
- *Bluetooth Core Specification version 5.2*, available from https://www.bluetooth.com/specifications/adopted-specifications

- TrustZone documentation available from the Arm website at https://developer.arm.com/ip-products/security-ip/trustzone/trustzone-for-cortex-m
- Other ArmCortex-M33 publications, available from the Arm website at https://developer.arm.com/ip-products/processors/cortex-m/cortex-m33

For information about the Evaluation and Development Board Manual and its schematics, go to the RSL15 web page and navigate to the EVB page.

CHAPTER 2

From Blinky to Building Your Own Application

Following this recommended progression with the SDK allows you to quickly become familiar with the software environment and learn to navigate, execute and modify the sample code with the ultimate goal of using the tools to build your own application.

2.1 PROGRESSION

The sample code offers many single-topic samples to enable you to learn about and develop one function or feature at a time in isolation. Once you are comfortable, you can work your way up to full-featured integrated samples, applying techniques you have learned from using the single function samples.

The progression from the simplest sample application to the most integrated application is:

1. Blinky: following the *RSL15 Getting Started Guide* to install the tools and execute blinky.
2. Single function samples: build and execute samples such as sleep, uart or SPI and experiment with configurations for your end application.
3. Wireless samples: build and execute samples such as *ble_peripheral_server* to experiment with a wireless Bluetooth Low Energy connection and develop configurations for your application.
4. Integrated samples: build and execute samples such as *ble_peripheral_server_sleep* and *ble_peripheral_server_sleep_fota* to learn interactions of a complete system.
5. See Section 4 “How to Build Your Own RSL15 Application” on page 13 to add additional software components such as UART to the integrated sample to form the basis of your own application.

The illustrates the progression in getting comfortable with building your own applications for RSL15.

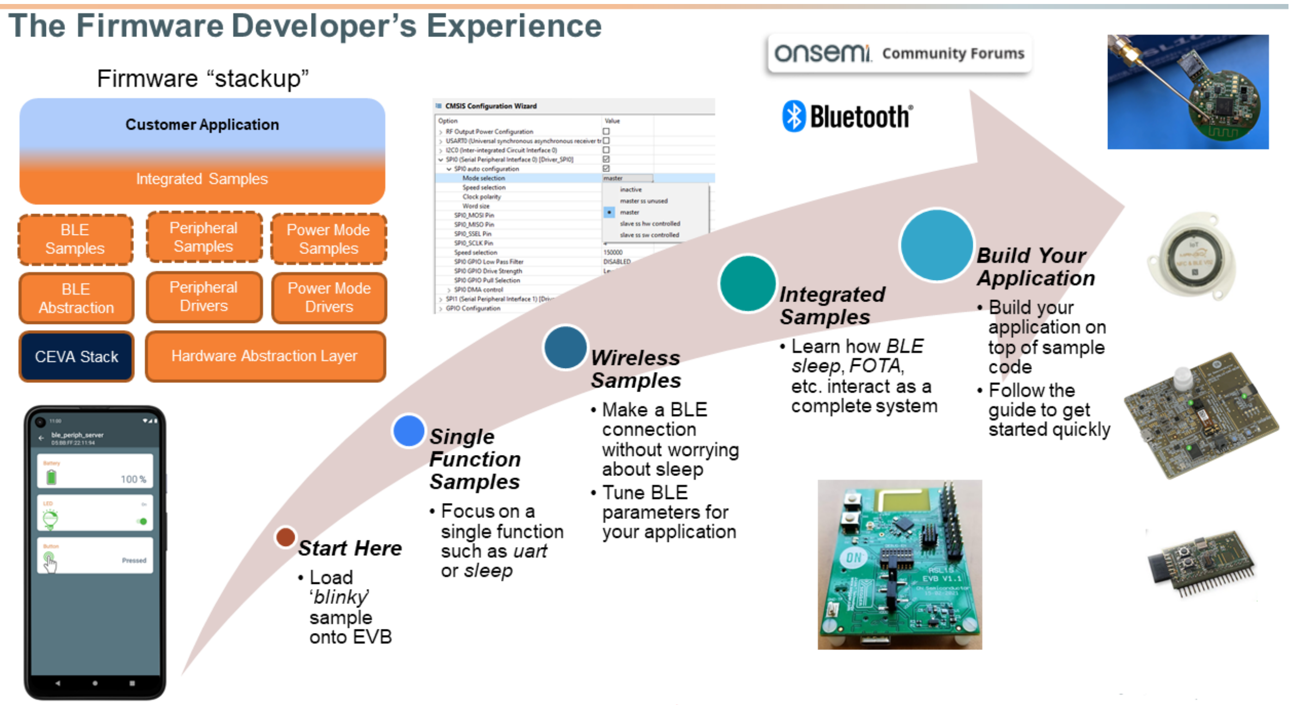

The Firmware Developer’s Experience
Firmware “stackup”
Customer Application
Integrated Samples
BLE Samples
Peripheral Samples
Power Mode Samples
BLE Abstraction
Peripheral Drivers
Power Mode Drivers
CEVA Stack
Hardware Abstraction Layer
onsemi. Community Forums
Bluetooth
Start Here
• Load ‘blinky’ sample onto EVB
Single Function Samples
• Focus on a single function such as uart or sleep
Wireless Samples
• Make a BLE connection without worrying about sleep
• Tune BLE parameters for your application
Integrated Samples
• Learn how BLE sleep, FOTA, etc. interact as a complete system
Build Your Application
• Build your application on top of sample code
• Follow the guide to get started quickly

Figure 1. The Firmware Developer’s Experience

CHAPTER 3

RSL15 CMSIS-Pack Component Description

This topic explains key concepts about the structure of the RSL15 CMSIS-Pack, its relationship to the project workspace, and methods for modifying linked files when necessary.

3.1 THE RSL15 CMSIS-PACK COMPONENT STRUCTURE

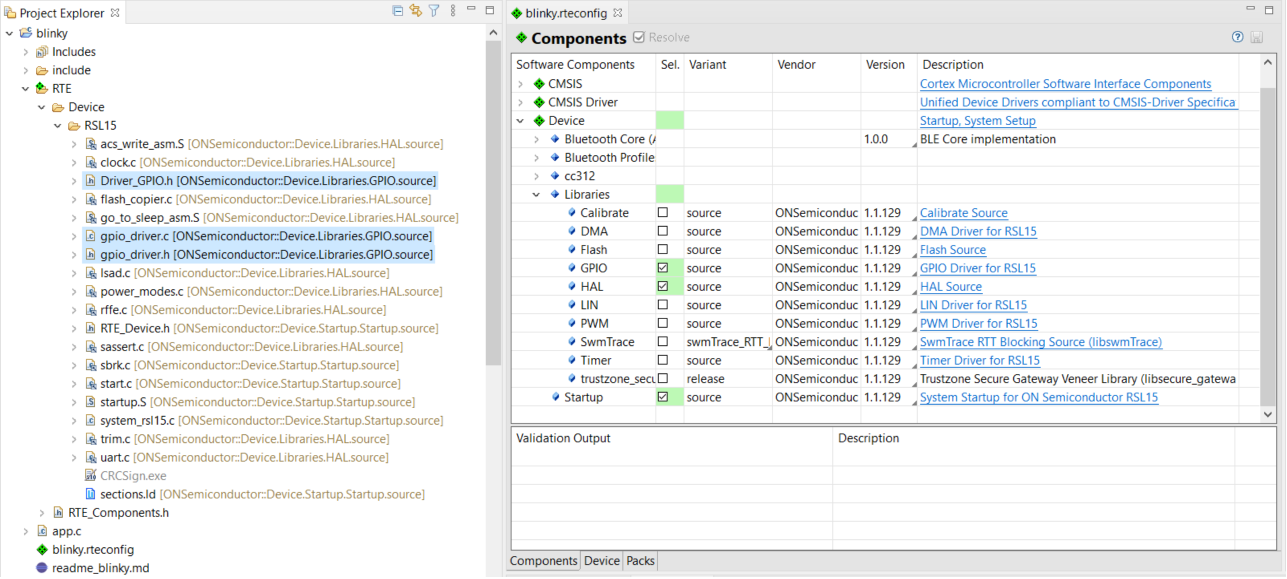
3.1.1 Introduction

The CMSIS-Pack is a convenient method to package and deliver sample code. The structure of the pack, projects and workspaces allow for easy update when new packs are available.

The sample code project workspaces contain the application files, but not the system-level files. The system-level files remain in the CMSIS-Pack, and are linked to the project workspaces. This allows for easy updating of these linked system-level files when new packs are installed, without changing the user application code to which these files are linked. For most use cases, customers do not need to modify these linked system level files.

3.1.2 What is a Component?

A component is a group of files in the CMSIS-Pack that are relevant to a particular project. For example, the GPIO component in the *blinky* sample project consists of *gpio_driver.c*, *gpio_driver.h* and *Driver_GPIO.h*. This is shown in the "GPIO Component" figure (Figure 2), which shows a list of the files in the GPIO component.



```
blinky
  Includes
  include
  RTE
    Device
      RSL15
        acs_write_asm.S [ONSemiconductor::Device.Libraries.HAL.source]
        clock.c [ONSemiconductor::Device.Libraries.HAL.source]
        Driver_GPIO.h [ONSemiconductor::Device.Libraries.GPIO.source]
        flash_copier.c [ONSemiconductor::Device.Libraries.HAL.source]
        go_to_sleep_asm.S [ONSemiconductor::Device.Libraries.HAL.source]
        gpio_driver.c [ONSemiconductor::Device.Libraries.GPIO.source]
        gpio_driver.h [ONSemiconductor::Device.Libraries.GPIO.source]
        lsad.c [ONSemiconductor::Device.Libraries.HAL.source]
        power_modes.c [ONSemiconductor::Device.Libraries.HAL.source]
        rffe.c [ONSemiconductor::Device.Libraries.HAL.source]
        RTE_Device.h [ONSemiconductor::Device.Startup.Startup.source]
        sassert.c [ONSemiconductor::Device.Libraries.HAL.source]
        sbrk.c [ONSemiconductor::Device.Startup.Startup.source]
        start.c [ONSemiconductor::Device.Startup.Startup.source]
        startup.S [ONSemiconductor::Device.Startup.Startup.source]
        system_rsl15.c [ONSemiconductor::Device.Startup.Startup.source]
        trim.c [ONSemiconductor::Device.Libraries.HAL.source]
        uart.c [ONSemiconductor::Device.Libraries.HAL.source]
        CRCSign.exe
        sections.ld [ONSemiconductor::Device.Startup.Startup.source]
    RTE_Components.h
  app.c
  blinky.rteconfig
  readme_blinky.md
```

Software Components	Sel.	Variant	Vendor	Version	Description
CMSIS					Cortex Microcontroller Software Interface Components
CMSIS Driver					Unified Device Drivers compliant to CMSIS-Driver Specifica
Device					Startup, System Setup
Bluetooth Core (/				1.0.0	BLE Core implementation
Bluetooth Profile					
cc312					
Libraries					
Calibrate	☐	source	ONSemiconduc	1.1.129	Calibrate Source
DMA	☐	source	ONSemiconduc	1.1.129	DMA Driver for RSL15
Flash	☐	source	ONSemiconduc	1.1.129	Flash Source
GPIO	☑	source	ONSemiconduc	1.1.129	GPIO Driver for RSL15
HAL	☑	source	ONSemiconduc	1.1.129	HAL Source
LIN	☐	source	ONSemiconduc	1.1.129	LIN Driver for RSL15
PWM	☐	source	ONSemiconduc	1.1.129	PWM Driver for RSL15
SwmTrace	☐	swmTrace_RTT_	ONSemiconduc	1.1.129	SwmTrace RTT Blocking Source (libswmTrace)
Timer	☐	source	ONSemiconduc	1.1.129	Timer Driver for RSL15
trustzone_secu	☐	release	ONSemiconduc	1.1.129	Trustzone Secure Gateway Veneer Library (libsecure_gatewa
Startup	☑	source	ONSemiconduc	1.1.129	System Startup for ON Semiconductor RSL15

Validation Output	Description

Figure 2. GPIO Component

Component Files

The files of a software component are used in development toolchains when an application is being built. The way the files are handled depends on the attributes of the files, as follows:

- Libraries, source files, and header files without an attribute cannot be modified in the Keil µVision IDE, but can be modified in the onsemi and IAR Embedded Workbench IDEs. These files without attributes are stored in the folders of the software component and are directly included from this location into the project. This means you need to be careful in modifying these files, because your changes apply to all the other applications that use these files.

- Header files of compiled libraries are linked/read-only, because modifying the header without the library source has no effect on the library.
- Source files and header files that have the `config` attribute are copied to the project, so that the user can edit them according to what is needed in the application.

The "Display of a Software Component’s Files in Different Development Tools" figure (Figure 3) shows how a software component’s files are displayed in the three different development tools. In the figure, files that do not have an attribute are highlighted for the onsemi IDE and the IAR IDE; for the Keil IDE, they have a lock icon. Files with `attr=“config”` are not highlighted.

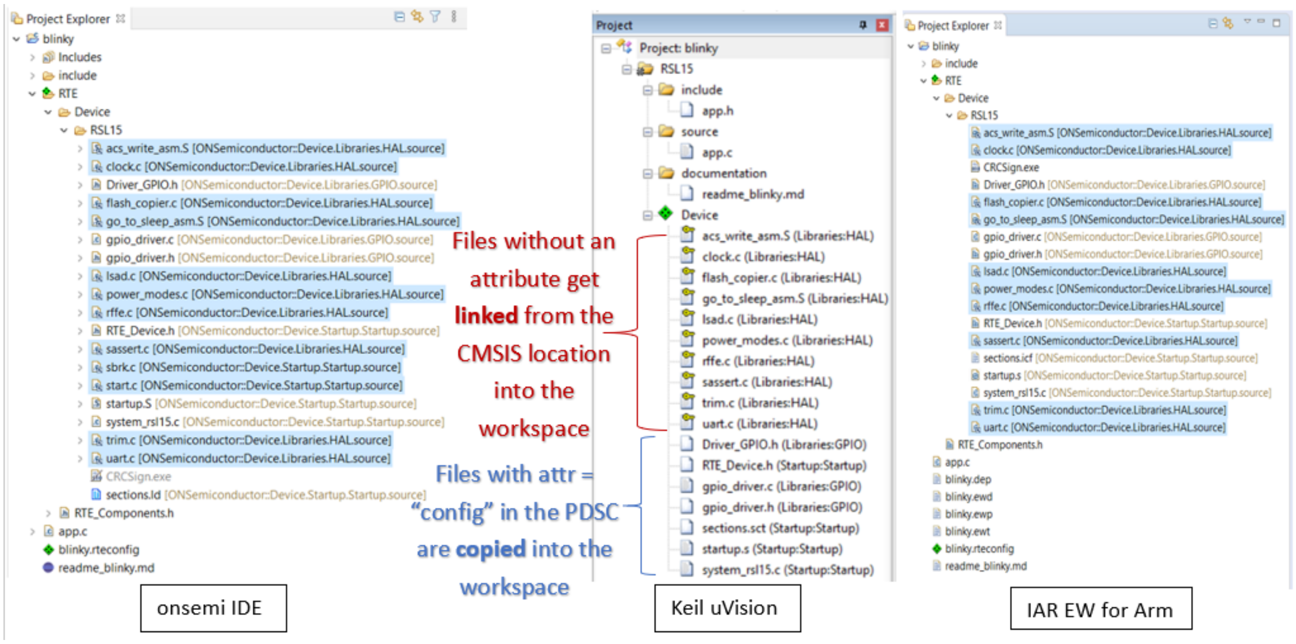
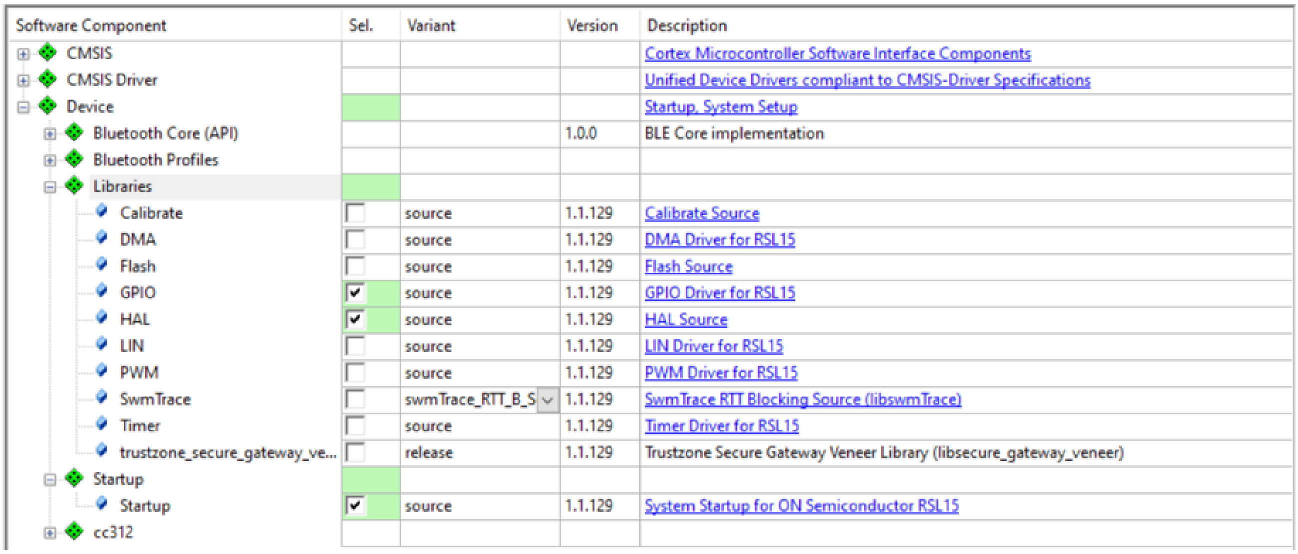

Project Explorer
blinky
Includes
include
RTE
Device
RSL15
acs_write_asm.S [ONSemiconductor::Device.Libraries.HAL.source]
clock.c [ONSemiconductor::Device.Libraries.HAL.source]
Driver_GPIO.h [ONSemiconductor::Device.Libraries.GPIO.source]
flash_copier.c [ONSemiconductor::Device.Libraries.HAL.source]
go_to_sleep_asm.S [ONSemiconductor::Device.Libraries.HAL.source]
gpio_driver.c [ONSemiconductor::Device.Libraries.GPIO.source]
gpio_driver.h [ONSemiconductor::Device.Libraries.GPIO.source]
lsad.c [ONSemiconductor::Device.Libraries.HAL.source]
power_modes.c [ONSemiconductor::Device.Libraries.HAL.source]
rffe.c [ONSemiconductor::Device.Libraries.HAL.source]
RTE_Device.h [ONSemiconductor::Device.Startup.Startup.source]
sassert.c [ONSemiconductor::Device.Libraries.HAL.source]
sbrk.c [ONSemiconductor::Device.Startup.Startup.source]
start.c [ONSemiconductor::Device.Startup.Startup.source]
startup.S [ONSemiconductor::Device.Startup.Startup.source]
system_rsl15.c [ONSemiconductor::Device.Startup.Startup.source]
trim.c [ONSemiconductor::Device.Libraries.HAL.source]
uart.c [ONSemiconductor::Device.Libraries.HAL.source]
CRCSign.exe
sections.ld [ONSemiconductor::Device.Startup.Startup.source]
RTE_Components.h
app.c
blinky.rteconfig
readme_blinky.md
onsemi IDE
Files without an attribute get linked from the CMSIS location into the workspace
Files with attr = “config” in the PDSC are copied into the workspace
Project
Project: blinky
RSL15
include
app.h
source
app.c
documentation
readme_blinky.md
Device
acs_write_asm.S (Libraries:HAL)
clock.c (Libraries:HAL)
flash_copier.c (Libraries:HAL)
go_to_sleep_asm.S (Libraries:HAL)
lsad.c (Libraries:HAL)
power_modes.c (Libraries:HAL)
rffe.c (Libraries:HAL)
sassert.c (Libraries:HAL)
trim.c (Libraries:HAL)
uart.c (Libraries:HAL)
Driver_GPIO.h (Libraries:GPIO)
RTE_Device.h (Startup:Startup)
gpio_driver.c (Libraries:GPIO)
gpio_driver.h (Libraries:GPIO)
sections.sct (Startup:Startup)
startup.s (Startup:Startup)
system_rsl15.c (Startup:Startup)
Keil uVision
Project Explorer
blinky
include
RTE
Device
RSL15
acs_write_asm.S [ONSemiconductor::Device.Libraries.HAL.source]
clock.c [ONSemiconductor::Device.Libraries.HAL.source]
CRCSign.exe
Driver_GPIO.h [ONSemiconductor::Device.Libraries.GPIO.source]
flash_copier.c [ONSemiconductor::Device.Libraries.HAL.source]
go_to_sleep_asm.S [ONSemiconductor::Device.Libraries.HAL.source]
gpio_driver.c [ONSemiconductor::Device.Libraries.GPIO.source]
gpio_driver.h [ONSemiconductor::Device.Libraries.GPIO.source]
lsad.c [ONSemiconductor::Device.Libraries.HAL.source]
power_modes.c [ONSemiconductor::Device.Libraries.HAL.source]
rffe.c [ONSemiconductor::Device.Libraries.HAL.source]
RTE_Device.h [ONSemiconductor::Device.Startup.Startup.source]
sassert.c [ONSemiconductor::Device.Libraries.HAL.source]
sections.icf [ONSemiconductor::Device.Startup.Startup.source]
startup.s [ONSemiconductor::Device.Startup.Startup.source]
system_rsl15.c [ONSemiconductor::Device.Startup.Startup.source]
trim.c [ONSemiconductor::Device.Libraries.HAL.source]
uart.c [ONSemiconductor::Device.Libraries.HAL.source]
RTE_Components.h
app.c
blinky.dep
blinky.ewd
blinky.ewp
blinky.ewt
blinky.rteconfig
readme_blinky.md
IAR EW for Arm

Figure 3. Display of a Software Component’s Files in Different Development Tools

The "Software Components for the Blinky Sample Application" figure (Figure 4) shows the software components for the *blinky* sample project, as have been illustrated in the "Display of a Software Component’s Files in Different Development Tools" figure (Figure 3).



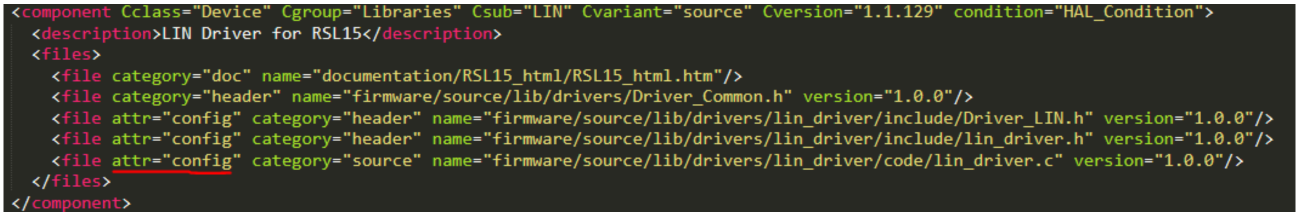
Software Component	Sel.	Variant	Version	Description
CMSIS				Cortex Microcontroller Software Interface Components
CMSIS Driver				Unified Device Drivers compliant to CMSIS-Driver Specifications
Device				Startup, System Setup
Bluetooth Core (API)			1.0.0	BLE Core implementation
Bluetooth Profiles				
Libraries				
Calibrate	☐	source	1.1.129	Calibrate Source
DMA	☐	source	1.1.129	DMA Driver for RSL15
Flash	☐	source	1.1.129	Flash Source
GPIO	☑	source	1.1.129	GPIO Driver for RSL15
HAL	☑	source	1.1.129	HAL Source
LIN	☐	source	1.1.129	LIN Driver for RSL15
PWM	☐	source	1.1.129	PWM Driver for RSL15
SwmTrace	☐	swmTrace_RTT_B_S	1.1.129	SwmTrace RTT Blocking Source (libswmTrace)
Timer	☐	source	1.1.129	Timer Driver for RSL15
trustzone_secure_gateway_ve...	☐	release	1.1.129	Trustzone Secure Gateway Veneer Library (libsecure_gateway_veneer)
Startup				
Startup	☑	source	1.1.129	System Startup for ON Semiconductor RSL15
cc312				

Figure 4. Software Components for the Blinky Sample Application

3.1.3 Modifying Files Without an Attribute

For most cases, you do not need to modify the linked files that do not have an attribute, as any modifications are applied to all applications that use the files. However, if you do need to modify the linked file for a particular application without impacting the other applications, you can do so by following these steps:

1. Open the *ONSemiconductor.RSL15.pdsc* file located at *%LOCALAPPDATA%\Arm\Packs\ONSemiconductor\RSL15\*`<version>` (for onsemi IDE and Keil IDE), or *C:\Users\*`<user_name>`*\IAR-CMSIS-Packs\ONSemiconductor\RSL15\*`<version>` (for IAR IDE).
2. Search for the file you want to modify in the `<component>` section, and add `attr = "config"` to the file. (See the "Modifying a File Without an Attribute" figure (Figure 5).)



```
<component Cclass="Device" Cgroup="Libraries" Csub="LIN" Cvariant="source" Cversion="1.1.129" condition="HAL_Condition">
  <description>LIN Driver for RSL15</description>
  <files>
    <file category="doc" name="documentation/RSL15_html/RSL15_html.htm"/>
    <file category="header" name="firmware/source/lib/drivers/Driver_Common.h" version="1.0.0"/>
    <file attr="config" category="header" name="firmware/source/lib/drivers/lin_driver/include/Driver_LIN.h" version="1.0.0"/>
    <file attr="config" category="header" name="firmware/source/lib/drivers/lin_driver/include/lin_driver.h" version="1.0.0"/>
    <file attr="config" category="source" name="firmware/source/lib/drivers/lin_driver/code/lin_driver.c" version="1.0.0"/>
  </files>
</component>
```

Figure 5. Modifying a File Without an Attribute

3. Save the *ONSemiconductor.RSL15.pdsc* file.

   NOTE: For the Keil IDE, you need to open the editor in administrator mode; otherwise you cannot save the file.

4. Restart the IDE.

3.2 MODIFYING LINKED SYSTEM-LEVEL FILES

For most use cases, there is no need to modify the linked system-level files. However, if such modification is necessary, it can be done in either of two ways:

1. Modify the files directly in the RSL15 CMSIS-Pack. Be aware that any other project linking to the modified files is affected by your changes as well.
2. Copy the file to the workspace, and break the link to the CMSIS-Pack.

IMPORTANT: Any modification to the CMSIS-Pack structure will require manual maintenance when you update to a new CMSIS-Pack version. Consider this before modifying the linked system-level files.

CHAPTER 4

How to Build Your Own RSL15 Application

This group of topics outlines two different methods for building your own RSL15 application using the onsemi IDE.

4.1 Option 1: Using the CMSIS-Pack Features to Build Your Own Application

The easiest way to build your first RSL15 application is to modify one of the existing sample applications that are part of the RSL15 CMSIS-Pack.

While this method has many steps, each step is simple and makes use of the CMSIS-Pack features to allow for ease development and ease of updating when new CMSIS-Packs are released by onsemi. This is the applicable method for most users. Advanced users can consider using an alternate option, described in Section 4.1 “Option 1: Using the CMSIS-Pack Features to Build Your Own Application” on page 13.

4.1.1 Copying Application Code Files

In the onsemi IDE, use the CMSIS-Pack Manager to copy an example that closely resembles your end application. The example used in this topic is the *ble_peripheral_server_sleep* application, as shown in the "Copying the Sample Application" figure (Figure 6), but you can use the sample application that suits your purposes best. This process copies the application code to your workspace. (For more information about using the IDE and its Pack Manager, see the *RSL15 Getting Started Guide*.)

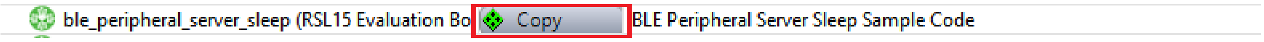
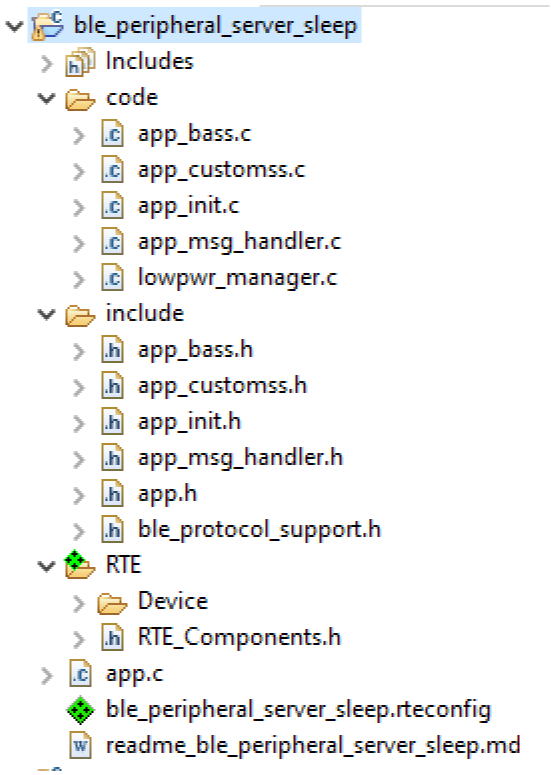

ble_peripheral_server_sleep (RSL15 Evaluation Bo
Copy
BLE Peripheral Server Sleep Sample Code

Figure 6. Copying the Sample Application

Once you have copied the application code files to your workspace, they can be seen in the project root directory, and the *code*, *include*, and *RTE* folders in the project explorer, as shown in the "Application Code Files" figure (Figure 7) (using the *ble_peripheral_server_sleep* sample project as the example). These are the applications files; you can edit them, and/or add new files, to build your own RSL15 application.

IMPORTANT: When you start building your own application, give it a new unique name, so that you can later copy updated sample projects from future CMSIS-Packs into the same workspace for comparison. It is not recommended that you keep the same name as the sample project your application is based on.

For this example, *ble_peripheral_server_sleep* is renamed *ble_peripheral_server_sleep_my_app.*

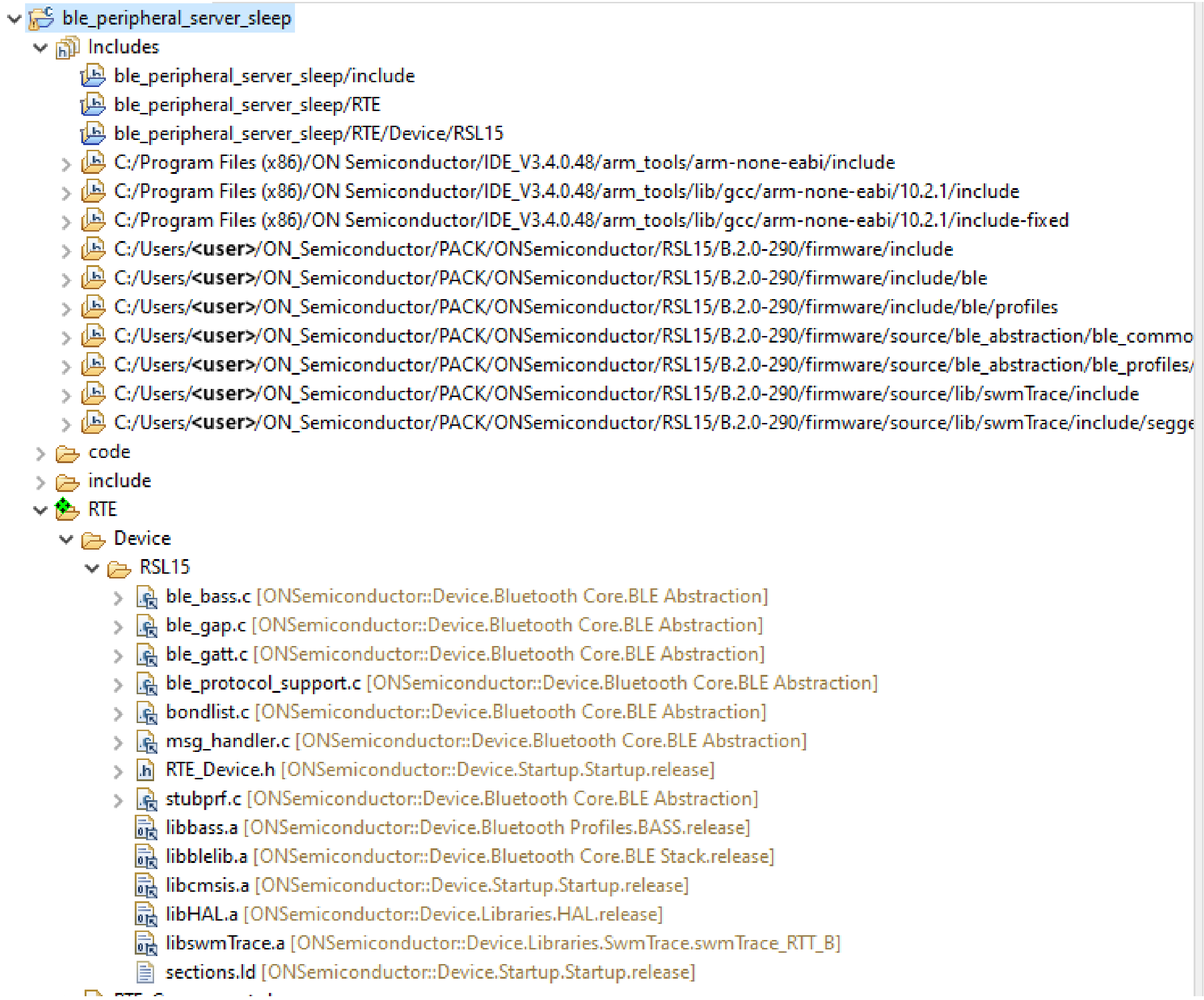


- ble_peripheral_server_sleep
  - Includes
  - code
    - app_bass.c
    - app_customss.c
    - app_init.c
    - app_msg_handler.c
    - lowpwr_manager.c
  - include
    - app_bass.h
    - app_customss.h
    - app_init.h
    - app_msg_handler.h
    - app.h
    - ble_protocol_support.h
  - RTE
    - Device
    - RTE_Components.h
  - app.c
  - ble_peripheral_server_sleep.rteconfig
  - readme_ble_peripheral_server_sleep.md

Figure 7. Application Code Files

4.1.2 Library and System Files

The other software components remain in the CMSIS-Pack at the installed location (*C:\users\<user_name>\ON_Semiconductor\PACK\ONSemiconductor\RSL15\*`<version_number>`*\firmware\source\samples*) and are linked by your project. These links are visible in the "Library and System Files Linked by the Project" figure (Figure 8).



```
ble_peripheral_server_sleep
  Includes
    ble_peripheral_server_sleep/include
    ble_peripheral_server_sleep/RTE
    ble_peripheral_server_sleep/RTE/Device/RSL15
    C:/Program Files (x86)/ON Semiconductor/IDE_V3.4.0.48/arm_tools/arm-none-eabi/include
    C:/Program Files (x86)/ON Semiconductor/IDE_V3.4.0.48/arm_tools/lib/gcc/arm-none-eabi/10.2.1/include
    C:/Program Files (x86)/ON Semiconductor/IDE_V3.4.0.48/arm_tools/lib/gcc/arm-none-eabi/10.2.1/include-fixed
    C:/Users/<user>/ON_Semiconductor/PACK/ONSemiconductor/RSL15/B.2.0-290/firmware/include
    C:/Users/<user>/ON_Semiconductor/PACK/ONSemiconductor/RSL15/B.2.0-290/firmware/include/ble
    C:/Users/<user>/ON_Semiconductor/PACK/ONSemiconductor/RSL15/B.2.0-290/firmware/include/ble/profiles
    C:/Users/<user>/ON_Semiconductor/PACK/ONSemiconductor/RSL15/B.2.0-290/firmware/source/ble_abstraction/ble_commo
    C:/Users/<user>/ON_Semiconductor/PACK/ONSemiconductor/RSL15/B.2.0-290/firmware/source/ble_abstraction/ble_profiles/
    C:/Users/<user>/ON_Semiconductor/PACK/ONSemiconductor/RSL15/B.2.0-290/firmware/source/lib/swmTrace/include
    C:/Users/<user>/ON_Semiconductor/PACK/ONSemiconductor/RSL15/B.2.0-290/firmware/source/lib/swmTrace/include/segge
  code
  include
  RTE
    Device
      RSL15
        ble_bass.c [ONSemiconductor::Device.Bluetooth Core.BLE Abstraction]
        ble_gap.c [ONSemiconductor::Device.Bluetooth Core.BLE Abstraction]
        ble_gatt.c [ONSemiconductor::Device.Bluetooth Core.BLE Abstraction]
        ble_protocol_support.c [ONSemiconductor::Device.Bluetooth Core.BLE Abstraction]
        bondlist.c [ONSemiconductor::Device.Bluetooth Core.BLE Abstraction]
        msg_handler.c [ONSemiconductor::Device.Bluetooth Core.BLE Abstraction]
        RTE_Device.h [ONSemiconductor::Device.Startup.Startup.release]
        stubprf.c [ONSemiconductor::Device.Bluetooth Core.BLE Abstraction]
        libbass.a [ONSemiconductor::Device.Bluetooth Profiles.BASS.release]
        libblelib.a [ONSemiconductor::Device.Bluetooth Core.BLE Stack.release]
        libcmsis.a [ONSemiconductor::Device.Startup.Startup.release]
        libHAL.a [ONSemiconductor::Device.Libraries.HAL.release]
        libswmTrace.a [ONSemiconductor::Device.Libraries.SwmTrace.swmTrace_RTT_B]
        sections.ld [ONSemiconductor::Device.Startup.Startup.release]
```

Figure 8. Library and System Files Linked by the Project

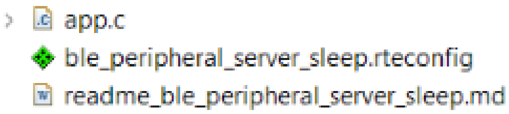
In most cases, these software components do not need to be modified by the user, so linking to the CMSIS-Pack is more efficient than copying these software component files to your workspace.

If you really do need to modify any of these software component files, you can go into the CMSIS-Pack and modify them directly, but be aware that his changes the files for all the sample code that links to those files.

The *rteconfig* file can be used to add software components to your project. The components can then be configured in the *RTE_device.h* file, using the CMSIS Configuration Wizard.

To add the software components, perform the following steps:

1. Double click `<project_name>`*.rteconfig*, as shown in the "Selecting the Project's .rteconfig File" figure (Figure 9).

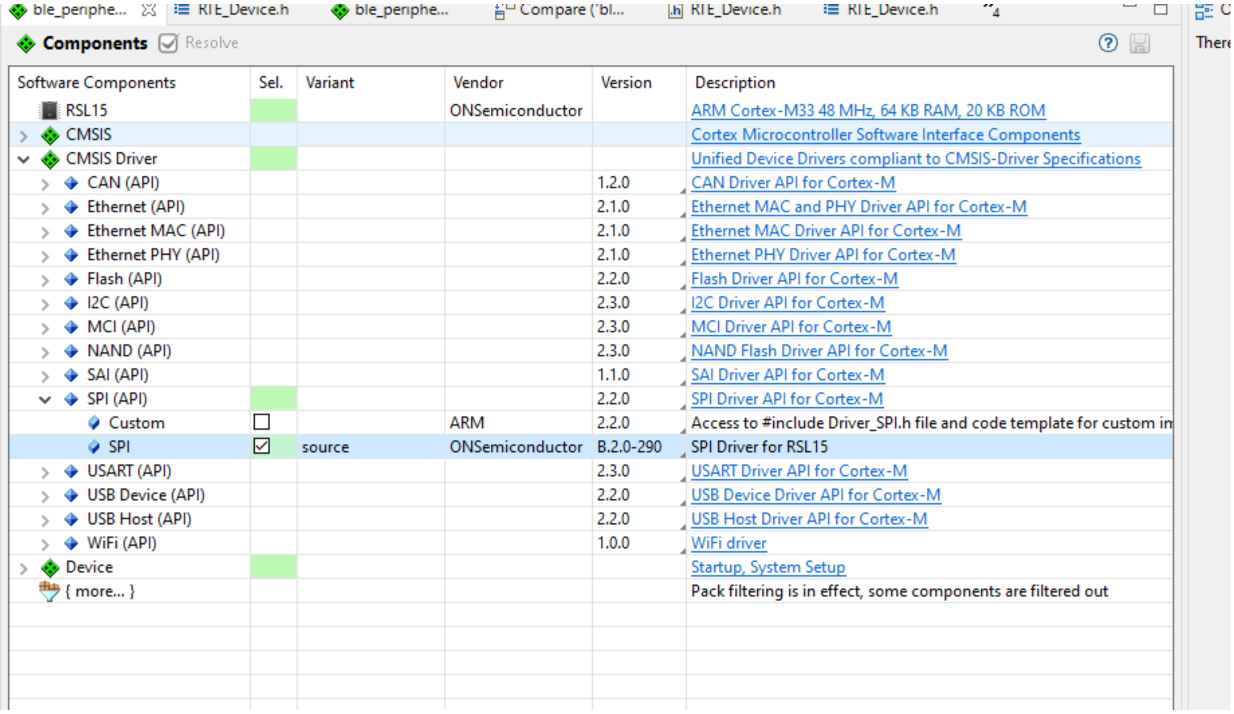


app.c
ble_peripheral_server_sleep.rteconfig
readme_ble_peripheral_server_sleep.md

Figure 9. Selecting the Project’s .rteconfig File

2. Select the desired components (for example, SPI), as shown in the "Selecting the Components" figure (Figure 10). Save the changes.

NOTE: Only software components whose **Vendor** is listed as **ONSemiconductor** are developed and provided by onsemi.

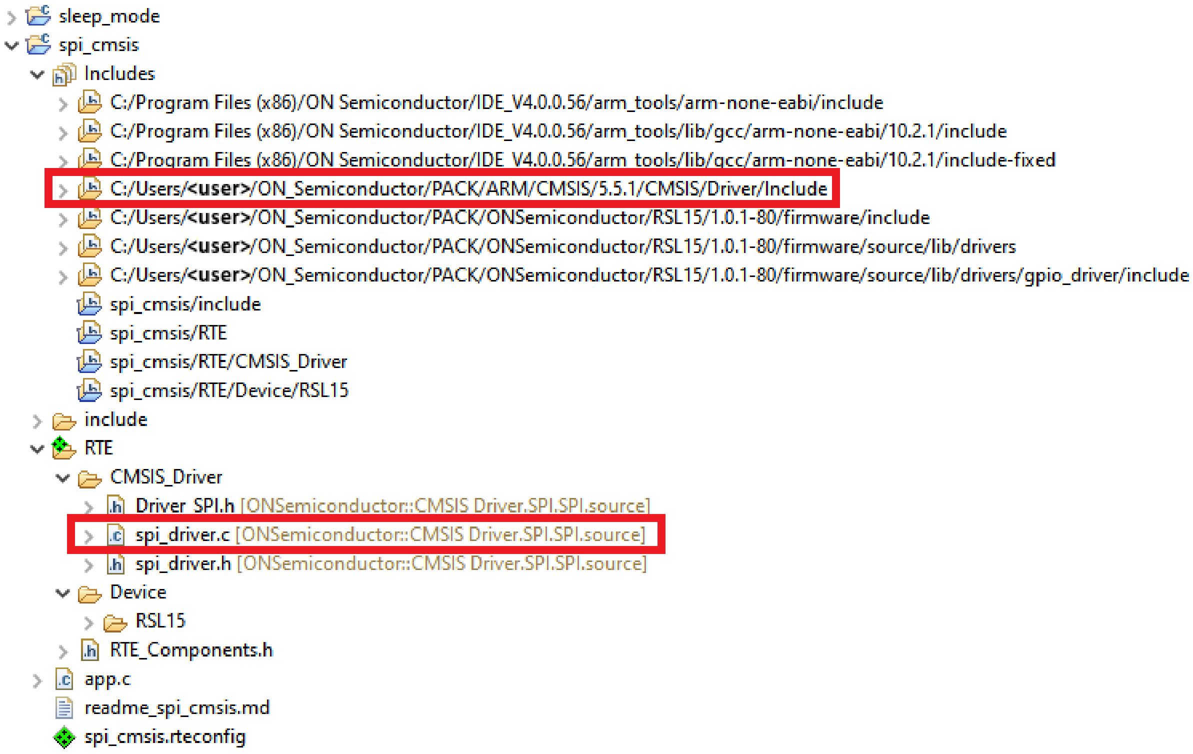


Software Components	Sel.	Variant	Vendor	Version	Description
RSL15	🟩		ONSemiconductor		ARM Cortex-M33 48 MHz, 64 KB RAM, 20 KB ROM
CMSIS					Cortex Microcontroller Software Interface Components
CMSIS Driver	🟩				Unified Device Drivers compliant to CMSIS-Driver Specifications
CAN (API)				1.2.0	CAN Driver API for Cortex-M
Ethernet (API)				2.1.0	Ethernet MAC and PHY Driver API for Cortex-M
Ethernet MAC (API)				2.1.0	Ethernet MAC Driver API for Cortex-M
Ethernet PHY (API)				2.1.0	Ethernet PHY Driver API for Cortex-M
Flash (API)				2.2.0	Flash Driver API for Cortex-M
I2C (API)				2.3.0	I2C Driver API for Cortex-M
MCI (API)				2.3.0	MCI Driver API for Cortex-M
NAND (API)				2.3.0	NAND Flash Driver API for Cortex-M
SAI (API)				1.1.0	SAI Driver API for Cortex-M
SPI (API)	🟩			2.2.0	SPI Driver API for Cortex-M
Custom	☐		ARM	2.2.0	Access to #include Driver_SPI.h file and code template for custom in
SPI	☑	source	ONSemiconductor	B.2.0-290	SPI Driver for RSL15
USART (API)				2.3.0	USART Driver API for Cortex-M
USB Device (API)				2.2.0	USB Device Driver API for Cortex-M
USB Host (API)				2.2.0	USB Host Driver API for Cortex-M
WiFi (API)				1.0.0	WiFi driver
Device	🟩				Startup, System Setup
{ more... }					Pack filtering is in effect, some components are filtered out

Figure 10. Selecting the Components

This adds the linked driver source and header files, as shown in the "Driver Source and Header Files" figure (Figure 11).

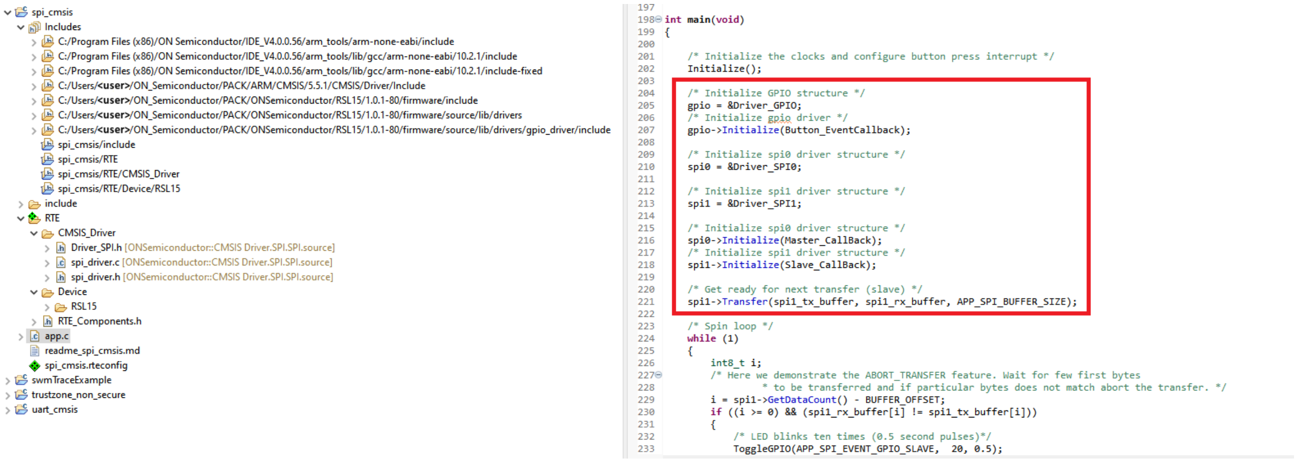
NOTE: You might need to manually refresh the project view. To do this, right-click the project name and click **Refresh** to see the changes.



```
> sleep_mode
v spi_cmsis
  v Includes
    > C:/Program Files (x86)/ON Semiconductor/IDE_V4.0.0.56/arm_tools/arm-none-eabi/include
    > C:/Program Files (x86)/ON Semiconductor/IDE_V4.0.0.56/arm_tools/lib/gcc/arm-none-eabi/10.2.1/include
    > C:/Program Files (x86)/ON Semiconductor/IDE_V4.0.0.56/arm_tools/lib/gcc/arm-none-eabi/10.2.1/include-fixed
    > C:/Users/<user>/ON_Semiconductor/PACK/ARM/CMSIS/5.5.1/CMSIS/Driver/Include
    > C:/Users/<user>/ON_Semiconductor/PACK/ONSemiconductor/RSL15/1.0.1-80/firmware/include
    > C:/Users/<user>/ON_Semiconductor/PACK/ONSemiconductor/RSL15/1.0.1-80/firmware/source/lib/drivers
    > C:/Users/<user>/ON_Semiconductor/PACK/ONSemiconductor/RSL15/1.0.1-80/firmware/source/lib/drivers/gpio_driver/include
      spi_cmsis/include
      spi_cmsis/RTE
      spi_cmsis/RTE/CMSIS_Driver
      spi_cmsis/RTE/Device/RSL15
  > include
  v RTE
    v CMSIS_Driver
      > Driver_SPI.h [ONSemiconductor::CMSIS Driver.SPI.SPI.source]
      > spi_driver.c [ONSemiconductor::CMSIS Driver.SPI.SPI.source]
      > spi_driver.h [ONSemiconductor::CMSIS Driver.SPI.SPI.source]
    v Device
      > RSL15
    > RTE_Components.h
  > app.c
    readme_spi_cmsis.md
    spi_cmsis.rteconfig
```

Figure 11. Driver Source and Header Files

Now the project includes your selected components — in the example seen in the "Initialization Code to Copy Manually" figure (Figure 12), this is the SPI driver. The initialization code must be manually copied from each component's sample code into your project. Some header includes might need to be added to your application as well; for instance, *#include* **`<spi_driver>`***.h* for the SPI driver. You can also copy the implementation from the component's code sample, or develop your own.

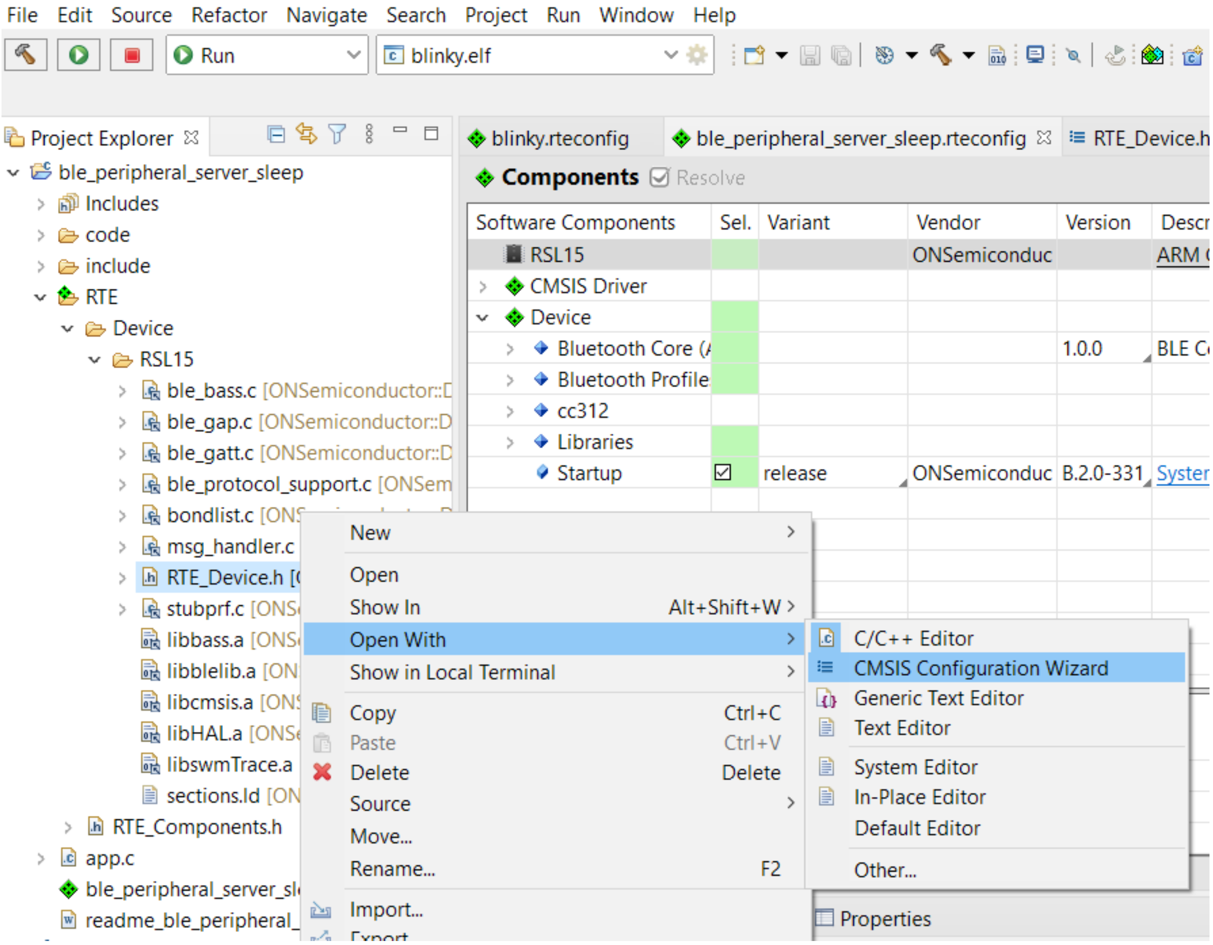


```
v spi_cmsis
  v Includes
    > C:/Program Files (x86)/ON Semiconductor/IDE_V4.0.0.56/arm_tools/arm-none-eabi/include
    > C:/Program Files (x86)/ON Semiconductor/IDE_V4.0.0.56/arm_tools/lib/gcc/arm-none-eabi/10.2.1/include
    > C:/Program Files (x86)/ON Semiconductor/IDE_V4.0.0.56/arm_tools/lib/gcc/arm-none-eabi/10.2.1/include-fixed
    > C:/Users/<user>/ON_Semiconductor/PACK/ARM/CMSIS/5.5.1/CMSIS/Driver/Include
    > C:/Users/<user>/ON_Semiconductor/PACK/ONSemiconductor/RSL15/1.0.1-80/firmware/include
    > C:/Users/<user>/ON_Semiconductor/PACK/ONSemiconductor/RSL15/1.0.1-80/firmware/source/lib/drivers
    > C:/Users/<user>/ON_Semiconductor/PACK/ONSemiconductor/RSL15/1.0.1-80/firmware/source/lib/drivers/gpio_driver/include
      spi_cmsis/include
      spi_cmsis/RTE
      spi_cmsis/RTE/CMSIS_Driver
      spi_cmsis/RTE/Device/RSL15
  > include
  v RTE
    v CMSIS_Driver
      > Driver_SPI.h [ONSemiconductor::CMSIS Driver.SPI.SPI.source]
      > spi_driver.c [ONSemiconductor::CMSIS Driver.SPI.SPI.source]
      > spi_driver.h [ONSemiconductor::CMSIS Driver.SPI.SPI.source]
    v Device
      > RSL15
    > RTE_Components.h
  > app.c
    readme_spi_cmsis.md
    spi_cmsis.rteconfig
> swmTraceExample
> trustzone_non_secure
> uart_cmsis
```

```c

int main(void)
{

    /* Initialize the clocks and configure button press interrupt */
    Initialize();

    /* Initialize GPIO structure */
    gpio = &Driver_GPIO;
    /* Initialize gpio driver */
    gpio->Initialize(Button_EventCallback);

    /* Initialize spi0 driver structure */
    spi0 = &Driver_SPI0;

    /* Initialize spi1 driver structure */
    spi1 = &Driver_SPI1;

    /* Initialize spi0 driver structure */
    spi0->Initialize(Master_CallBack);
    /* Initialize spi1 driver structure */
    spi1->Initialize(Slave_CallBack);

    /* Get ready for next transfer (slave) */
    spi1->Transfer(spi1_tx_buffer, spi1_rx_buffer, APP_SPI_BUFFER_SIZE);

    /* Spin loop */
    while (1)
    {
        int8_t i;
        /* Here we demonstrate the ABORT_TRANSFER feature. Wait for few first bytes
                 * to be transferred and if particular bytes does not match abort the transfer. */
        i = spi1->GetDataCount() - BUFFER_OFFSET;
        if ((i >= 0) && (spi1_rx_buffer[i] != spi1_tx_buffer[i]))
        {
            /* LED blinks ten times (0.5 second pulses)*/
            ToggleGPIO(APP_SPI_EVENT_GPIO_SLAVE,  20, 0.5);
```

Figure 12. Initialization Code to Copy Manually

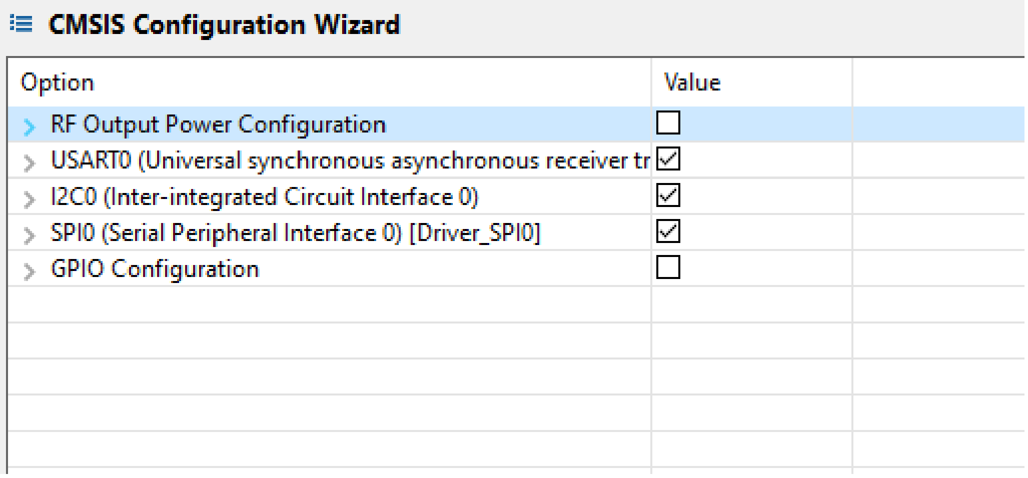
The components can be configured in the *RTE_device.h* file. In **RTE** > **Device** > **RSL15**, right click **RTE_device.h** and choose **Open With** > **CMSIS Configuration Wizard**, as shown in the "Opening the CMSIS Configuration Wizard" figure (Figure 13).



Software Components	Sel.	Variant	Vendor	Version	Descr
RSL15			ONSemiconduc		ARM (
CMSIS Driver					
Device					
Bluetooth Core (/				1.0.0	BLE C
Bluetooth Profile					
cc312					
Libraries					
Startup	☑	release	ONSemiconduc	B.2.0-331	Syster

Figure 13. Opening the CMSIS Configuration Wizard

This opens the CMSIS Configuration Wizard, which looks like the "CMSIS Configuration Wizard Window" figure (Figure 14):



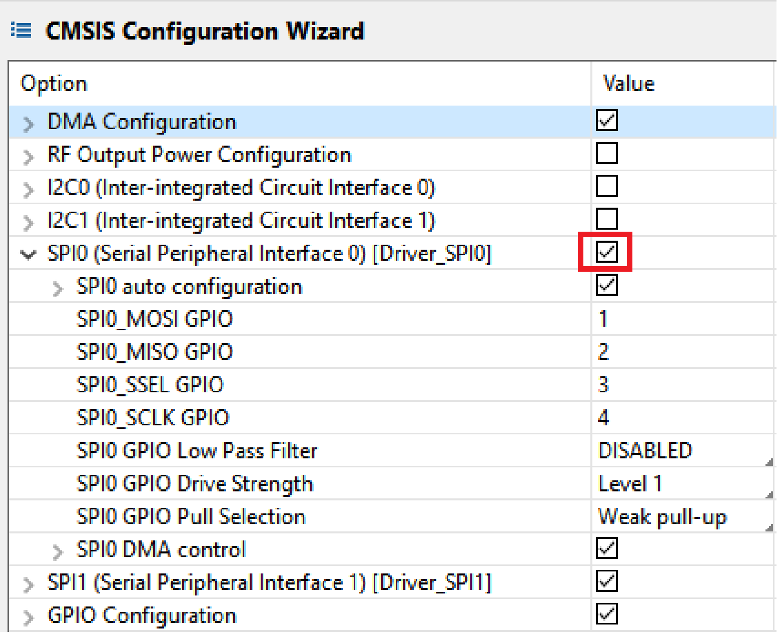
CMSIS Configuration Wizard

Option	Value
RF Output Power Configuration	☐
USART0 (Universal synchronous asynchronous receiver tr	☑
I2C0 (Inter-integrated Circuit Interface 0)	☑
SPI0 (Serial Peripheral Interface 0) [Driver_SPI0]	☑
GPIO Configuration	☐

Figure 14. CMSIS Configuration Wizard Window

This Wizard can configure some software components via a graphical interface, as shown in the "Applying Configuration with the CMSIS Configuration Wizard" figure (Figure 15).

NOTE: Selecting the component check box enables the configuration to be applied to the *RTE_device.h* file; however, it does not add the software component itself. Software components must be added using the *.rteconfig*.



CMSIS Configuration Wizard

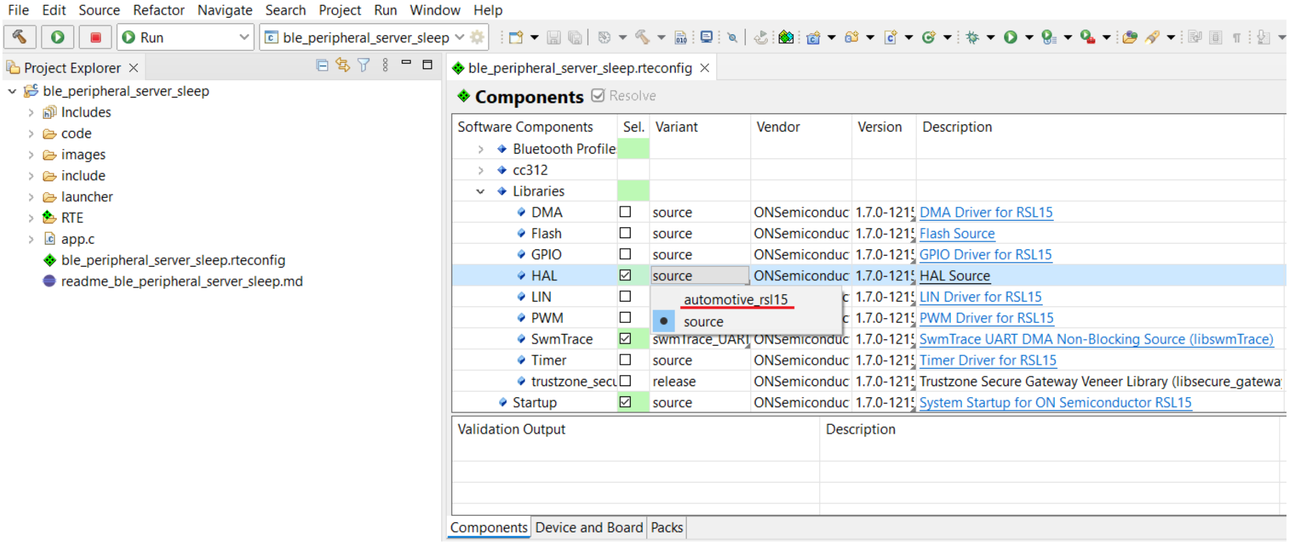
Option	Value
DMA Configuration	☑
RF Output Power Configuration	☐
I2C0 (Inter-integrated Circuit Interface 0)	☐
I2C1 (Inter-integrated Circuit Interface 1)	☐
SPI0 (Serial Peripheral Interface 0) [Driver_SPI0]	☑
SPI0 auto configuration	☑
SPI0_MOSI GPIO	1
SPI0_MISO GPIO	2
SPI0_SSEL GPIO	3
SPI0_SCLK GPIO	4
SPI0 GPIO Low Pass Filter	DISABLED
SPI0 GPIO Drive Strength	Level 1
SPI0 GPIO Pull Selection	Weak pull-up
SPI0 DMA control	☑
SPI1 (Serial Peripheral Interface 1) [Driver_SPI1]	☑
GPIO Configuration	☑

Figure 15. Applying Configuration with the CMSIS Configuration Wizard

From the CMSIS Configuration Wizard, your desired components (the SPI driver in the "Linked Files Update Automatically" figure (Figure 17)’s example) can be configured to your requirements.

4.1.3 NCV-RSL15 Automotive Product Selection

When using the NCV-RSL15 automotive product, select the **automotive_rsl15** variant for the HAL component in the *RTEConfig* file. This variant defines the **AUTOMOTIVE** definition, which selects the correct settings for the NCV-RSL15 automotive product. This is shown in the "NCV-RL15 Automotive Product HAL Variant" figure (Figure 16).


File Edit Source Refactor Navigate Search Project Run Window Help
Run
ble_peripheral_server_sleep
Project Explorer
ble_peripheral_server_sleep
Includes
code
images
include
launcher
RTE
app.c
ble_peripheral_server_sleep.rteconfig
readme_ble_peripheral_server_sleep.md
ble_peripheral_server_sleep.rteconfig
Components
Resolve
automotive_rsl15
source
Validation Output
Description
Components
Device and Board
Packs

Software Components	Sel.	Variant	Vendor	Version	Description
Bluetooth Profile					
cc312					
Libraries					
DMA	☐	source	ONSemiconduc	1.7.0-121	DMA Driver for RSL15
Flash	☐	source	ONSemiconduc	1.7.0-121	Flash Source
GPIO	☐	source	ONSemiconduc	1.7.0-121	GPIO Driver for RSL15
HAL	☑	source	ONSemiconduc	1.7.0-121	HAL Source
LIN	☐		c	1.7.0-121	LIN Driver for RSL15
PWM	☐		c	1.7.0-121	PWM Driver for RSL15
SwmTrace	☑	swmTrace_UART	ONSemiconduc	1.7.0-121	SwmTrace UART DMA Non-Blocking Source (libswmTrace)
Timer	☐	source	ONSemiconduc	1.7.0-121	Timer Driver for RSL15
trustzone_secu	☐	release	ONSemiconduc	1.7.0-121	Trustzone Secure Gateway Veneer Library (libsecure_gatewa
Startup	☑	source	ONSemiconduc	1.7.0-121	System Startup for ON Semiconductor RSL15

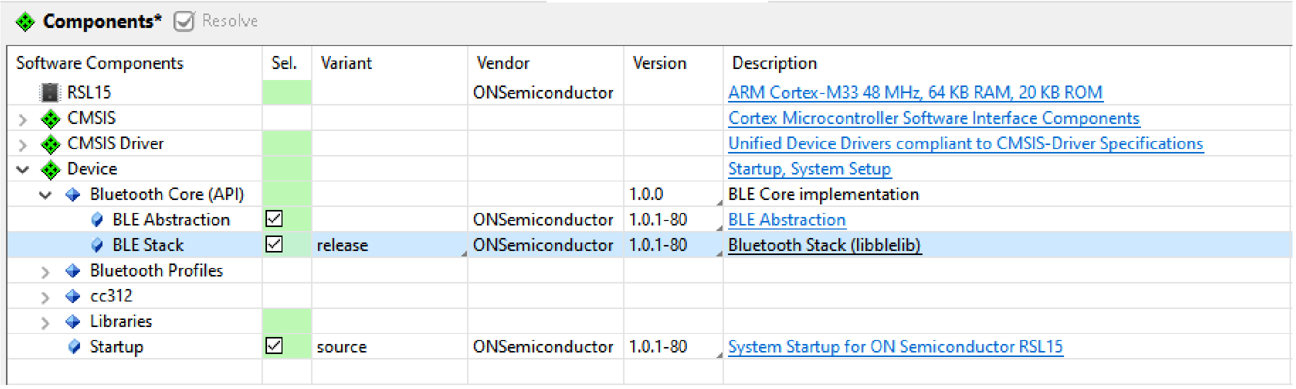
Figure 16. NCV-RL15 Automotive Product HAL Variant

4.1.4 Updating to a New CMSIS-Pack

From time to time, onsemi releases an updated CMSIS-Pack. You might wish to update your project to the latest CMSIS-Pack version. This process is made more convenient by the CMSIS-Pack linking feature. To update the CMSIS-Pack, use the general procedure guideline that follows, modified for your specific situation if necessary:

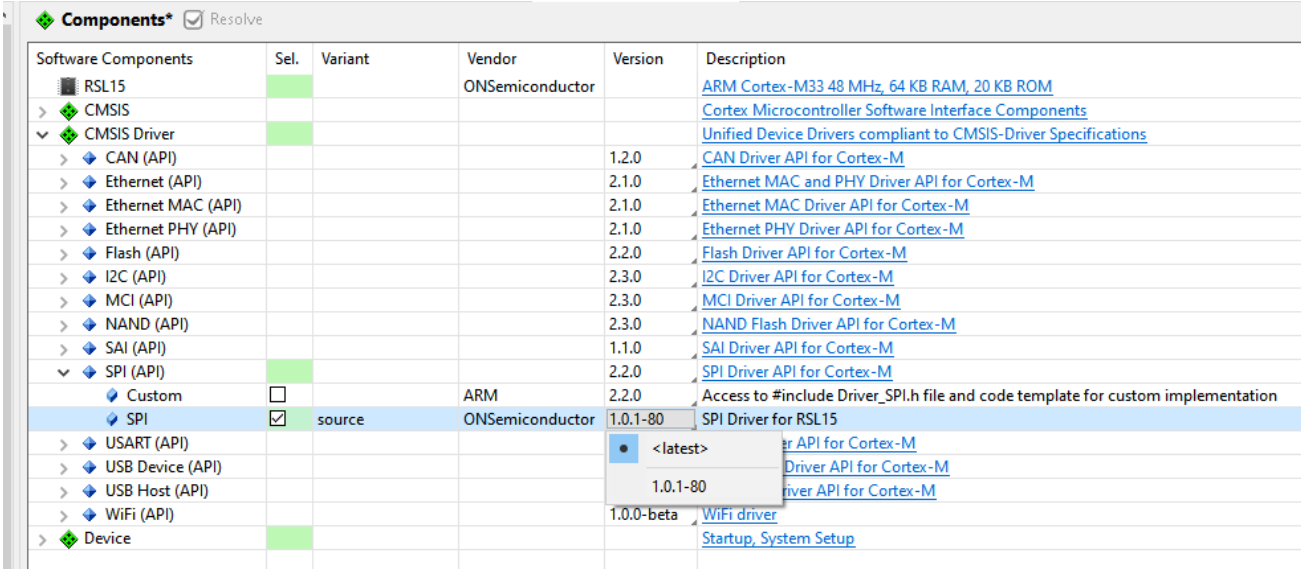
NOTE: Review the release notes from the CMSIS-Pack before you begin, as some additional steps might be required for a specific release.

1. Download and install the latest CMSIS-Pack, as instructed in the *RSL15 Getting Started Guide.*
2. Open your project. The CMSIS-Pack linked library and system files in your project are automatically updated to point to the latest installed CMSIS-Pack, as seen in the "Automatically Updating to the Latest Installed CMSIS-Pack" figure (Figure 18).



Software Components	Sel.	Variant	Vendor	Version	Description
RSL15	🟩		ONSemiconductor		ARM Cortex-M33 48 MHz, 64 KB RAM, 20 KB ROM
CMSIS					Cortex Microcontroller Software Interface Components
CMSIS Driver	🟩				Unified Device Drivers compliant to CMSIS-Driver Specifications
Device	🟩				Startup, System Setup
Bluetooth Core (API)	🟩			1.0.0	BLE Core implementation
BLE Abstraction	☑		ONSemiconductor	1.0.1-80	BLE Abstraction
BLE Stack	☑	release	ONSemiconductor	1.0.1-80	Bluetooth Stack (libblelib)
Bluetooth Profiles					
cc312					
Libraries	🟩				
Startup	☑	source	ONSemiconductor	1.0.1-80	System Startup for ON Semiconductor RSL15

Figure 17. Linked Files Update Automatically



Software Components	Sel.	Variant	Vendor	Version	Description
RSL15	🟩		ONSemiconductor		ARM Cortex-M33 48 MHz, 64 KB RAM, 20 KB ROM
CMSIS					Cortex Microcontroller Software Interface Components
CMSIS Driver	🟩				Unified Device Drivers compliant to CMSIS-Driver Specifications
CAN (API)				1.2.0	CAN Driver API for Cortex-M
Ethernet (API)				2.1.0	Ethernet MAC and PHY Driver API for Cortex-M
Ethernet MAC (API)				2.1.0	Ethernet MAC Driver API for Cortex-M
Ethernet PHY (API)				2.1.0	Ethernet PHY Driver API for Cortex-M
Flash (API)				2.2.0	Flash Driver API for Cortex-M
I2C (API)				2.3.0	I2C Driver API for Cortex-M
MCI (API)				2.3.0	MCI Driver API for Cortex-M
NAND (API)				2.3.0	NAND Flash Driver API for Cortex-M
SAI (API)				1.1.0	SAI Driver API for Cortex-M
SPI (API)	🟩			2.2.0	SPI Driver API for Cortex-M
Custom	☐		ARM	2.2.0	Access to #include Driver_SPI.h file and code template for custom implementation
SPI	☑	source	ONSemiconductor	1.0.1-80	SPI Driver for RSL15
USART (API)					r API for Cortex-M
USB Device (API)					Driver API for Cortex-M
USB Host (API)					iver API for Cortex-M
WiFi (API)				1.0.0-beta	WiFi driver
Device	🟩				Startup, System Setup

Figure 18. Automatically Updating to the Latest Installed CMSIS-Pack

5. If you want to update the application code from the latest CMSIS-Pack samples, use the CMSIS-Pack Manager to copy the examples from the latest CMSIS-Pack. Before coping the new project, rename the old project by appending the SDK version (such as *ble_peripheral_server_sleep_3_5*) to help keep track of the different sample versions.
6. Compare your application to the latest sample from the CMSIS-Pack using the Eclipse **Compare With** feature, as shown in the "Comparing Your Sample Application to the Latest Version" figure (Figure 19), or with your preferred comparison tool.

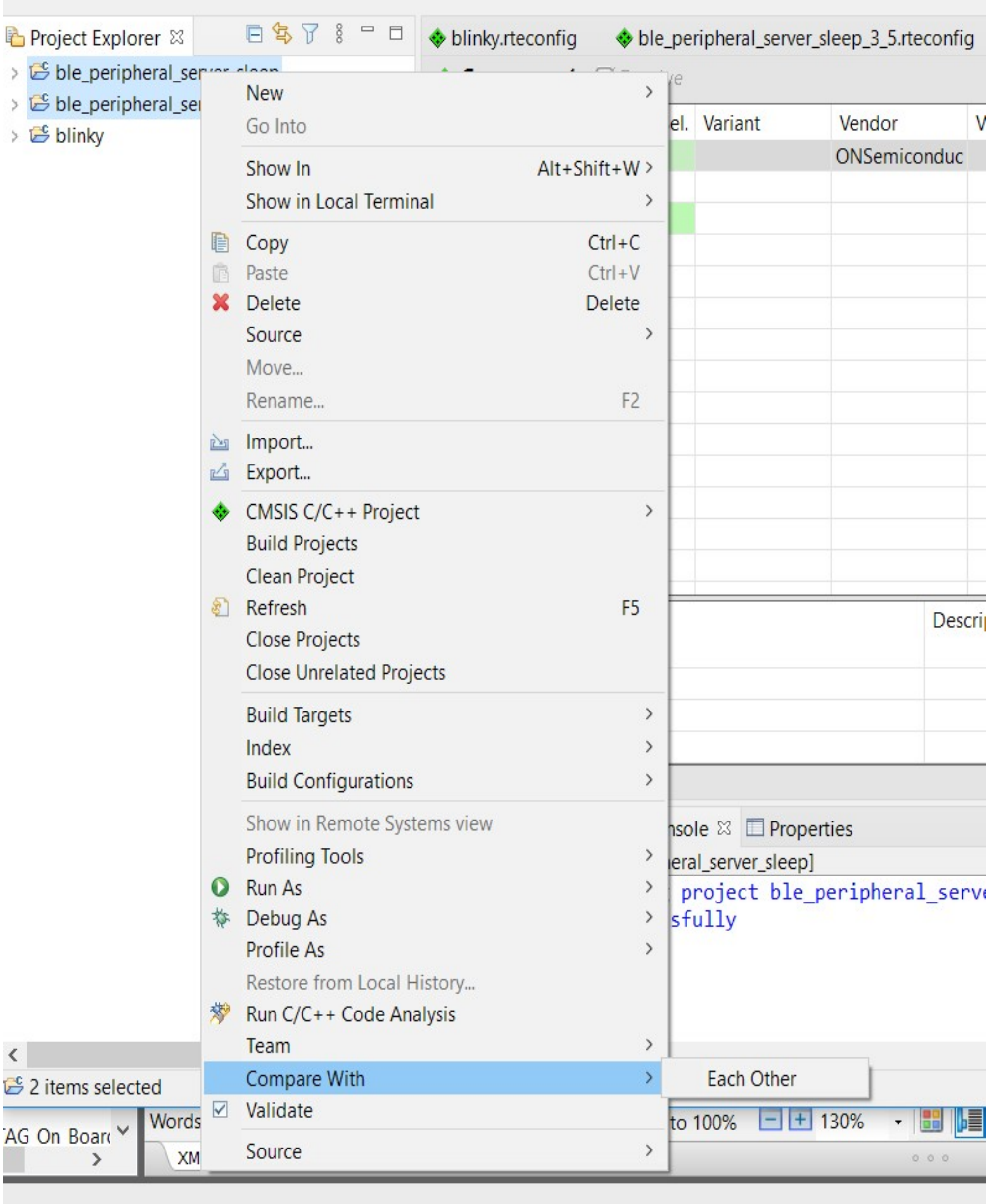
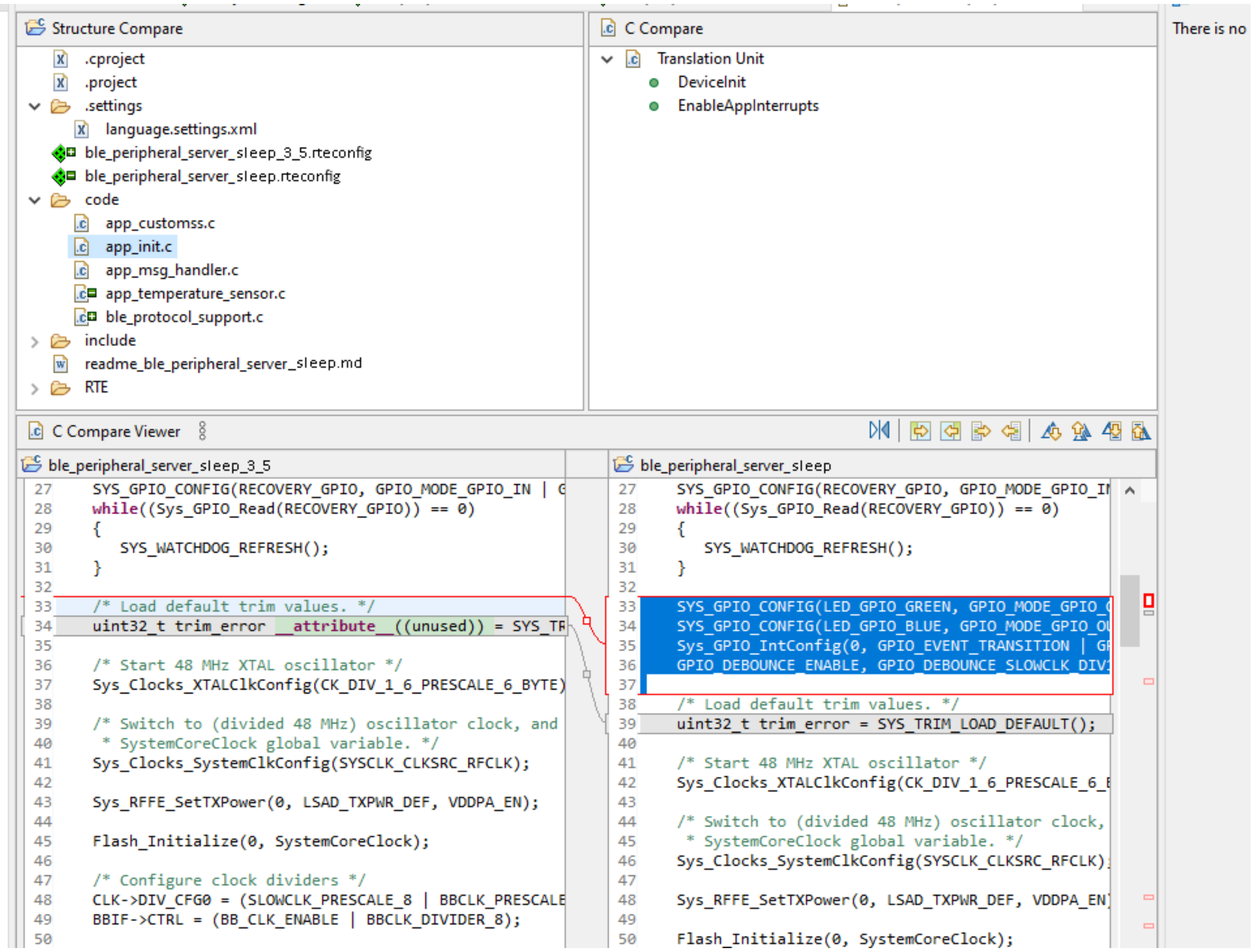


Figure 19. Comparing Your Sample Application to the Latest Version

7. The output shows all the files and the changes within the files, as seen in the "Changes Shown Between Application Versions" figure (Figure 20).



```
    SYS_GPIO_CONFIG(RECOVERY_GPIO, GPIO_MODE_GPIO_IN | G
    while((Sys_GPIO_Read(RECOVERY_GPIO)) == 0)
    {
        SYS_WATCHDOG_REFRESH();
    }

    /* Load default trim values. */
    uint32_t trim_error __attribute__((unused)) = SYS_TR

    /* Start 48 MHz XTAL oscillator */
    Sys_Clocks_XTALClkConfig(CK_DIV_1_6_PRESCALE_6_BYTE)

    /* Switch to (divided 48 MHz) oscillator clock, and
     * SystemCoreClock global variable. */
    Sys_Clocks_SystemClkConfig(SYSCLK_CLKSRC_RFCLK);

    Sys_RFFE_SetTXPower(0, LSAD_TXPWR_DEF, VDDPA_EN);

    Flash_Initialize(0, SystemCoreClock);

    /* Configure clock dividers */
    CLK->DIV_CFG0 = (SLOWCLK_PRESCALE_8 | BBCLK_PRESCALE
    BBIF->CTRL = (BB_CLK_ENABLE | BBCLK_DIVIDER_8);
```

```
    SYS_GPIO_CONFIG(RECOVERY_GPIO, GPIO_MODE_GPIO_IN
    while((Sys_GPIO_Read(RECOVERY_GPIO)) == 0)
    {
        SYS_WATCHDOG_REFRESH();
    }

    SYS_GPIO_CONFIG(LED_GPIO_GREEN, GPIO_MODE_GPIO_
    SYS_GPIO_CONFIG(LED_GPIO_BLUE, GPIO_MODE_GPIO_O
    Sys_GPIO_IntConfig(0, GPIO_EVENT_TRANSITION | GP
    GPIO_DEBOUNCE_ENABLE, GPIO_DEBOUNCE_SLOWCLK_DIV

    /* Load default trim values. */
    uint32_t trim_error = SYS_TRIM_LOAD_DEFAULT();

    /* Start 48 MHz XTAL oscillator */
    Sys_Clocks_XTALClkConfig(CK_DIV_1_6_PRESCALE_6_
    
    /* Switch to (divided 48 MHz) oscillator clock,
     * SystemCoreClock global variable. */
    Sys_Clocks_SystemClkConfig(SYSCLK_CLKSRC_RFCLK)

    Sys_RFFE_SetTXPower(0, LSAD_TXPWR_DEF, VDDPA_EN

    Flash_Initialize(0, SystemCoreClock);
```

Figure 20. Changes Shown Between Application Versions

8. Copy the changes from the latest CMSIS-Pack to your application as you see fit.

The linked library and system files in your project are automatically updated to point to the latest installed CMSIS-Pack, as seen in the "Linked Files Update Automatically" figure (Figure 17).

4.2 OPTION 2: UNZIP THE CMSIS-PACK (ADVANCED USERS)

Advanced users might need to modify the linked library and system files to suit their needs. If so, they can take advantage of the fact that the CMSIS-Pack is a zip file with an xml descriptor (*.pdsc* file). The CMSIS-Pack can simply be unzipped, independent of an IDE. Then the files can be taken directly from the unzipped output and modified as necessary. When new CMSIS-Packs become available from onsemi, the updates need to be manually copied/merged.

CHAPTER 5

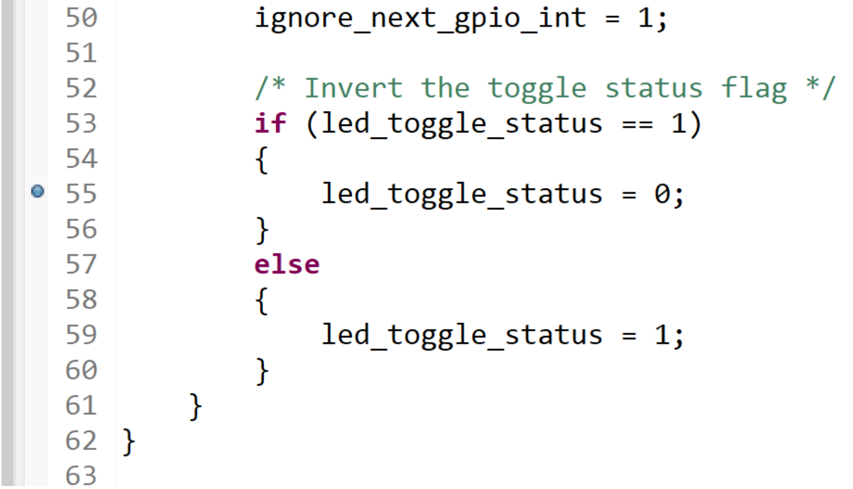
Diagnostic Strategies

This section explains some of the methods and tools available for testing, diagnosing and monitoring RSL15 applications. Good diagnostics and debugging generally starts with software best practices, such as modular code, coding standards and consistent well-placed comments. All attempts have been made to follow software best practices in the RSL15 SDK while offering other methods and tools to help quickly develop applications.

5.1 GENERAL DEBUGGING STRATEGIES

5.1.1 Breakpoints

A useful way to inspect code operation is to insert breakpoints in a Debug session, or resume a stopped Debug session. By doing so, you can have the program execute to a given point and then halt, at which point you can examine variables, memory or other diagnostic data. To insert a breakpoint, double-click to the left of the line number on which you want to stop. See the "Setting a Breakpoint" figure (Figure 21). Details on how to initiate a Debug session are provided in the *RSL15 Getting Started Guide*.



```
        ignore_next_gpio_int = 1;

        /* Invert the toggle status flag */
        if (led_toggle_status == 1)
        {
            led_toggle_status = 0;
        }
        else
        {
            led_toggle_status = 1;
        }
    }
}

```

Figure 21. Setting a Breakpoint

Breakpoints are a convenient method of checking that data at various stages is what you expect to see. With breakpoints, you can examine registers, variables and memory when debugging.

5.1.2 Debugging Best Practices

One bug can sometimes hide another, so do not ignore bugs you find that are unrelated to the one you are actively trying to fix. Remember, if you fix another bug, be sure to go back and try to reproduce the first one again.

When looking at problems with memory allocation, check that you did indeed allocate the right amount of memory, and check that you initialized it.

5.2 PRINTING TO A CONSOLE WITH SWMTRACE

We generally recommend establishing a method to view the internals of an application early on in development (before problems arise) to ensure that everything is working as expected. Comparing console outputs between successful and failed outcomes can help pinpoint problems. The swmTrace module provides data printing capability and is described in the *RSL15 Firmware Reference*. It supports the commonly known `printf()` function to allow seamless printing to a PC console and a series of logging functions whose output characteristics are based on the logging level selected. The source code for swmTrace is available in the installation folder under *firmware\source\lib\swmTrace*. There is also a sample application called *swmTrace_logger*, which demonstrates the usage of the swmTrace library.

5.2.1 onsemi IDE Setup Requirements

newlib-nano is an open-source C library enabled in all sample applications. We strongly recommend that you enable it in the onsemi IDE when using swmTrace, by performing the following steps:

1. Navigate in the onsemi IDE to **Properties** > **C/C++ Build** > **Settings** > **Cross ARM C Linker** > **Miscellaneous**.
2. Check the **Use newlib-nano** checkbox.

Enabling newlib-nano in the onsemi IDE also minimizes the flash and RAM requirements of applications.

5.2.2 Printing Methods

The swmTrace module supports two printing methods:

1. SEGGER Real Time Trace (RTT)
2. UART

SEGGER® RTT works over JTAG (SWD) and pairs with the onsemi IDE's RTT viewer plugin. This is a specific view found under in the **Window > Show View** menu under **onsemi**, and features a console with controls for clearing the console, connecting once, stopping a connection, and enabling automatic reconnection. It remains connected in Sleep Mode. You must have the J-Link software installed to use this mode. Additional settings are available in the **Window > Preferences** menu.

UART works simply by transmitting characters out of the UART interface. Only the UART transmit pin and a ground pin are required to receive the signal. The UART is especially helpful in later stages of product development when the SWD pins might not be accessible, and the UART interface is not needed for host controller interface test communications with the Bluetooth hardware. No standard PC viewers for UART data exist, so programs such as Hyperterminal or other terminal programs are commonly used. The UART never establishes a connection with the PC; thus it can transmit immediately after waking from sleep (with minimal initialization).

NOTE: UART support using GPIOs 5 and 6 is possible with the onboard J-Link interface on the RSL15 Evaluation and Development Board. See the Evaluation and Development Board manual for more information.

The functions used most frequently are the various `swmLog` functions, which allow you to output only trace messages if a particular log level has been selected. The log levels available are `VERBOSE`, `INFO`, `WARNING`, `ERROR`, and `FATAL`. For example, if you have some error messages being printed using `swmLogError`, some info messages using `swmLogInfo`, and some additional logging using `swmLogVerbose`, the amount of text output can be controlled by changing the selected log level, rather than actually commenting out portions of code.

Indicators for a test pass and fail are also available. The `swmTrace_printf` function always prints, regardless of the log level selected. Before using any swmTrace functions, the `swmTrace_init` function needs to be called.

5.2.3 Blocking Versus Non-Blocking Modes

In blocking mode, the `swmLog` function does not return until the output buffer is available. In low throughput scenarios, the buffer is almost always available, so little or no blocking occurs. In high throughput scenarios, the buffer might not always be available, so blocking might occur. Blocking mode is useful to ensure that the output data stream is not corrupted in high throughput scenarios; however, due to the swmLog function blocking, real-time performance of the system might be affected.

In non-blocking mode, the `swmLog` function always returns immediately, whether the buffer is available or not. In low throughput scenarios, the buffer is almost always available, so the data stream is usually not corrupted. In high throughput scenarios, the buffer might not always be available, so the data stream might be corrupted. Non-blocking mode is useful to help to maintain the real-time performance of the system; however, due to the swmLog function not blocking, the data stream might be corrupted.

Different versions of the library are available for each mode, so the appropriate one can be linked in to your application accordingly.

5.3 DEBUGGING WITH SLEEP MODE

When an RSL15 device enters Sleep Mode or another low power mode, the device typically disables all digital elements including the debug port to minimize the system current of the device.

IMPORTANT: When a debug port connection has been established, the debug port link remains active in low power modes if the Arm Cortex-M33-Debug power domain is not explicitly disabled. This supports debugging the functionality of an application that uses low power modes across a low power mode-wakeup cycle. For more information, see Section 1.1.1 "Keeping the Debugger Connected in Low Power Modes" on page 1.

NOTE: If the Arm Cortex-M33 processor processor is asleep but the overall system is not in Sleep Mode, the system can still transmit data or respond to a debugger. This common use case does not need special considerations.

If a problem or system being debugged requires current measurements as part of the debug process, the debug port needs to be disabled. If this is required, the following mechanisms can function as potential debug tools:

- The swmTrace module, described in section Section 5.2 "Printing to a Console with swmTrace" on page 24, can be used to print data up to the point of sleeping and immediately after wakeup.
- The UART interface can be used both with and without the swmTrace module, to communicate data while not in Sleep Mode.
- Simple techniques, such as setting a GPIO high while in Sleep Mode and low when exiting, can be used to quickly verify operation and whether the device has entered and exited Sleep Mode.

NOTE: UART and swmTrace transfers stop immediately and are truncated if a transfer is in progress when the device goes to Sleep Mode. If UART data truncation is a concern for your debug use case, care must be taken to ensure transfers are complete before going to Sleep Mode.

These debug tools and techniques for working with Sleep Mode are demonstrated in the *sleep_mode* sample application. See the *readme* file included in that sample for more information.

NOTE: Breakpoints are still active after waking from Sleep Mode. The breakpoint stops the core—but the debugger on the other side of the debug link might not be notified if the debug connection had not yet reconnected after wakeup.

5.4 TRUSTZONE APPLICATION DEVELOPMENT AND DEBUGGING

TrustZone is a security hardware extension that has been included with the Arm Cortex-M33 processor in the RSL15 device. This extension provides hardware isolation between secure and non-secure components, and can separate safe applications from unsafe ones, protecting underlying system components from being accessed or changed by untrusted code. For more information on the hardware support for TrustZone, see Section 1.1 “TrustZone” on page 1.

When using TrustZone, the application is typically divided into two components, containing the secure and the non-secure application code respectively. System execution starts in the secure application component, which executes the non-secure application when necessary.

NOTE: The main loop for the application can exist in either the secure or the non-secure application component.

- If the loop is in the non-secure application, no additional support is required.
- If the loop is in the secure application, the secure application must re-initialize and start execution of the non-secure application component from the reset handler each time the non-secure application component is expected to be executed.

5.4.1 Secure Application Components

Secure application components must include:

- All standard application items (stack pointer, vector table, standard interrupt and exception handlers, and system startup code)
- System configuration, including configuration of the execution environment for the non-secure application. This configuration includes:
  - The secure environment as defined by the Secure Attribution Unit (SAU) using the `SAU_*` registers, and the Implementation Defined Attribution Unit (IDAU) using the `SYSCTRL_NS_ACCESS_*` registers
  - The available peripherals, including the NVIC and other private peripherals
- A secure fault handler for catching faults that occur due to illegal accesses in the non-secure application
- Initialization and execution of the non-secure application component, including:
  - Configuration of the non-secure component's main stack pointer (`MSP_NS`)
  - Execution of the non-secure application component starting from its reset handler

The secure application component can provide an API for the non-secure application component. This API can provide access to secure hardware and firmware components that would otherwise be inaccessible to non-secure applications. All secure API functions that can be called from non-secure code must be declared as non-secure entry points, using the `cmse_nonsecure_entry` attribute.

The following provides an example API definition that supports toggling a GPIO that would not otherwise be accessible to non-secure code:

```
    void __attribute__ ((cmse_nonsecure_entry)) NSC_GPIO_Toggle(void)
```

5.4.2 Non-Secure Application Components

Non-secure application components must include:

- A non-secure specific stack pointer
- A non-secure specific vector table, including a reset handler and any other interrupt handlers that are needed in the non-secure application

IMPORTANT: The secure and non-secure application components must target different memory areas for code, data, and stack memory.

The non-secure application components must not include:

- Accesses to secure hardware components (including system configuration)
- Access to any peripherals that are not made accessible through the IDAU
- Calls into the secure application, except through function calls to API functions declared as non-secure entry points

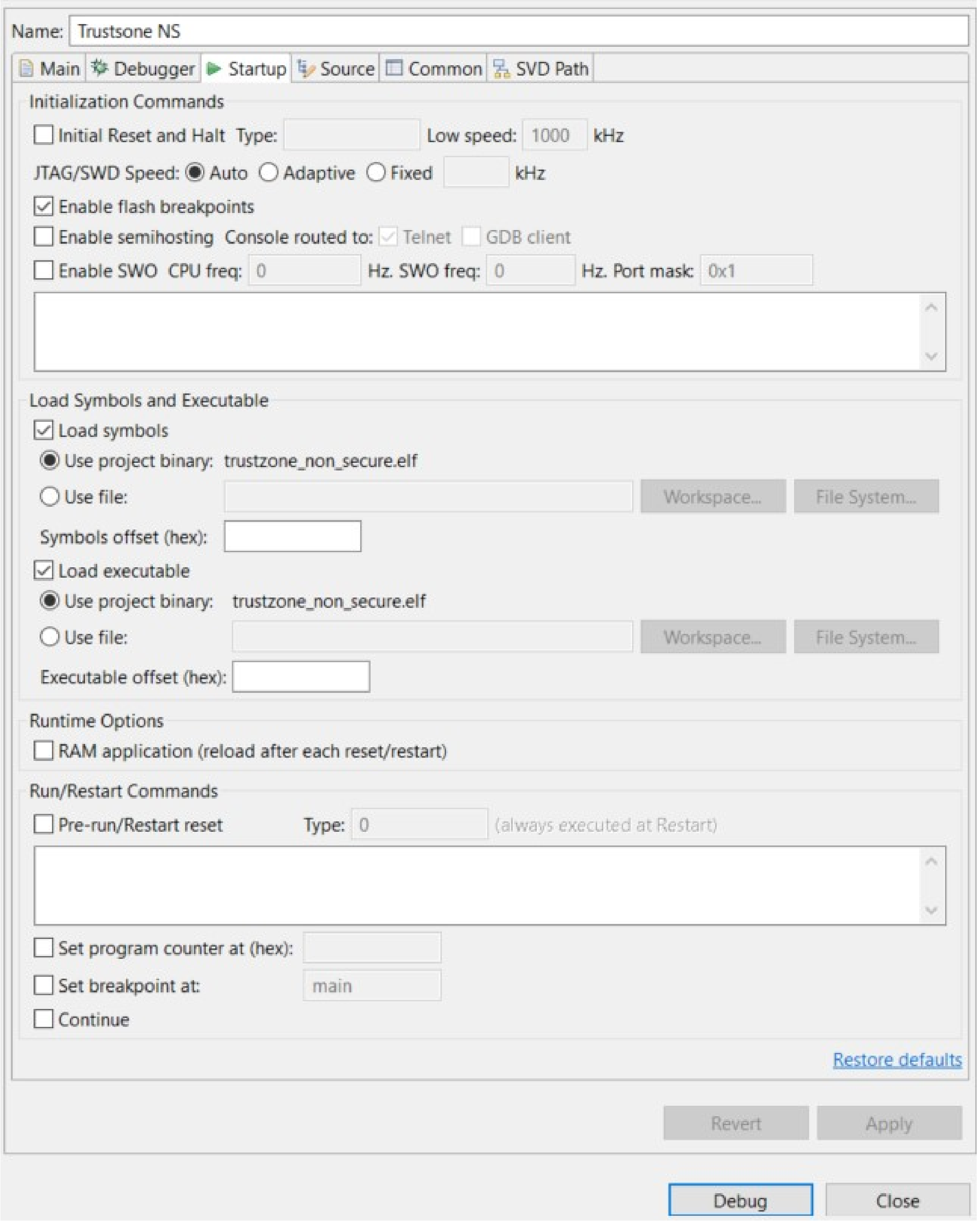
IMPORTANT: If an illegal access is included in the non-secure application, the secure application's secure fault handler is triggered and is used to handle the violation.

5.4.3 Debugging TrustZone Applications

When using TrustZone, extra steps beyond what is described in Chapter "Debugging the Sample Code" on page 1 are required. To debug the application components:

1. Load the non-secure application component to memory.
    - Loading this component can be implemented by setting up a debug or run launch configuration that loads the non-secure application component. It is best practice to clear the **Debug Configuration** > **Startup** > **Continue** setting when loading this component.

      An example debug configuration supporting loading a non-secure application is shown in "Loading non-secure application components" figure (Figure 22).



Name: Trustsone NS

Initialization Commands

☐ Initial Reset and Halt Type: Low speed: 1000 kHz

JTAG/SWD Speed: ● Auto ○ Adaptive ○ Fixed kHz

☑ Enable flash breakpoints

☐ Enable semihosting Console routed to: ☑ Telnet ☐ GDB client

☐ Enable SWO CPU freq: 0 Hz. SWO freq: 0 Hz. Port mask: 0x1

Load Symbols and Executable

☑ Load symbols

● Use project binary: trustzone_non_secure.elf

○ Use file:

Symbols offset (hex):

☑ Load executable

● Use project binary: trustzone_non_secure.elf

○ Use file:

Executable offset (hex):

Runtime Options

☐ RAM application (reload after each reset/restart)

Run/Restart Commands

☐ Pre-run/Restart reset Type: 0 (always executed at Restart)

☐ Set program counter at (hex):

☐ Set breakpoint at: main

☐ Continue

Figure 22. Loading non-secure application components

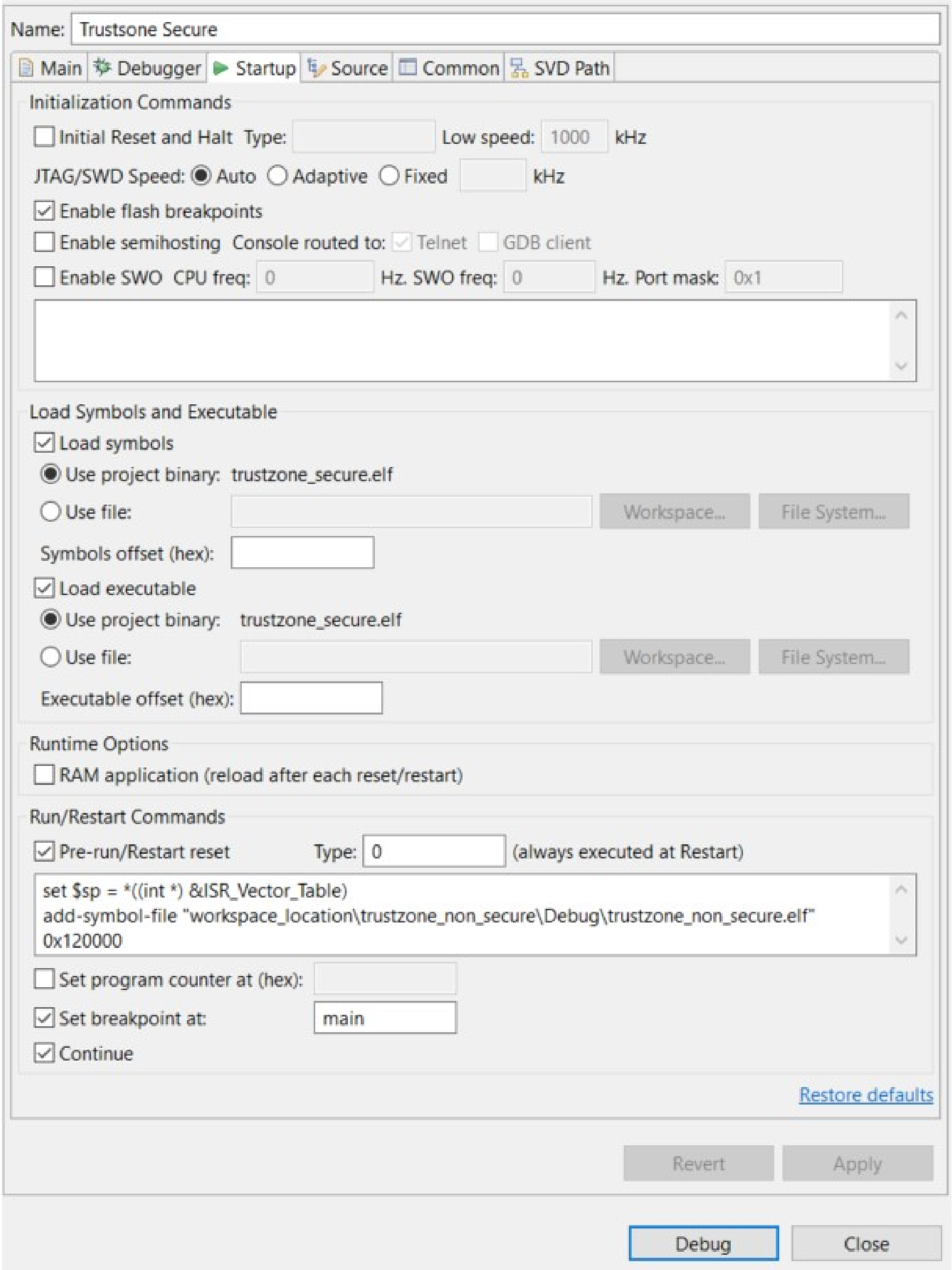
2. Load symbols from both application components and execute the secure application component.
    - The secure application component's symbols are loaded using **Debug Configuration** > **Startup** > **Load Symbols and Executable** > **Load symbols** > **Use project binary:** `<secure component elf file>`.

- To support debugging both application components, the following line must be added to the **Debug Configuration** > **Startup** > **Run/Restart Commands** box:

  ```
  add-symbol-file <non-secure component elf file> <non-secure component address>
  ```

  Where `<non-secure component address>` is the address of the non-secure component's vector table.

An example debug configuration, supporting the loading of a secure application and symbols for both the secure and non-secure application components, is shown in "Loading Secure Components and Debugging a Complete TrustZone Application" figure (Figure 23).



Name: Trustsone Secure

Initialization Commands

☐ Initial Reset and Halt Type: Low speed: 1000 kHz

JTAG/SWD Speed: ● Auto ○ Adaptive ○ Fixed kHz

☑ Enable flash breakpoints

☐ Enable semihosting Console routed to: ☑ Telnet ☐ GDB client

☐ Enable SWO CPU freq: 0 Hz. SWO freq: 0 Hz. Port mask: 0x1

Load Symbols and Executable

☑ Load symbols

● Use project binary: trustzone_secure.elf

○ Use file:

Symbols offset (hex):

☑ Load executable

● Use project binary: trustzone_secure.elf

○ Use file:

Executable offset (hex):

Runtime Options

☐ RAM application (reload after each reset/restart)

Run/Restart Commands

☑ Pre-run/Restart reset Type: 0 (always executed at Restart)

```
set $sp = *((int *) &ISR_Vector_Table)
add-symbol-file "workspace_location\trustzone_non_secure\Debug\trustzone_non_secure.elf"
0x120000
```

☐ Set program counter at (hex):

☑ Set breakpoint at: main

☑ Continue

Figure 23. Loading Secure Components and Debugging a Complete TrustZone Application

3. Load the secure application component to memory.

5.5 ACTIVITY COUNTERS

The Activity Counters are a set of hardware blocks used to accurately count cycles. The block can be used to profile system execution with the resolution of the system clock. It can easily be added to any code but keep in mind interrupts may vary the result. See the Activity Counters topic in the *RSL15 Hardware Reference* for more information.

5.6 DEBUGGING APPLICATIONS THAT ARE NOT STORED AT THE BASE ADDRESS OF FLASH

If you want to debug an application that does not start at the first address of the flash memory (0x00100000), adjust your Debug Configuration on the Startup tab. See the *RSL15 Getting Started Guide* for instructions on how to find this and the appropriate entry for the typical case of starting from the base of flash memory. The following example can be entered into the **Run/Restart Command**s field to execute an application at any location in memory:

```
set {int} &  VTOR = ISR_Vector_Table
set $sp = *((int *) &ISR_Vector_Table)
set $pc = *((int *) (&ISR_Vector_Table+4))
```

NOTE: The linker configuration file (*sections.ld*) must match the start address used in the debug configuration.

NOTE: All user applications must include a vector table, and the vector table must be aligned to a 512-byte boundary.

An example of when you might not want to start from the base address of flash is when you are using a bootloader that would typically be located at the base of flash. This is described in the readme file of the RSL15 sample bootloader application as well as in the RSL15 documentation set.

5.7 STRATEGIES FOR HANDLING APPLICATION ERRORS

When an application error occurs, the device can end up in one of a few error states:

- The device could end up in a state where the firmware is stuck and the watchdog is not being refreshed, triggering a watchdog interrupt.
- The system could enter a state where a reset occurs due to one or more power supply issues.
- The system could encounter faults in the program execution due to program errors, erroneous execution, or access to secured resources when in the wrong system state.

The behavior of user applications when any of these kinds of errors occurs is important for the stability of end products. The RSL15 device provides a variety of information and resources for determining the causes of system errors, and we can recommend some top-level best practices for handling errors. However, the details of how your user application handles errors and error recovery is dependent on your application use cases. The following subsections discuss general best practices, and the information that the system provides for handling and debugging watchdog interrupts, resets due to any cause, and faults.

5.7.1 Best Practices in Error Handling

Best practices for error handling in user applications requires the user applications to include:

1. A handler for the watchdog timer interrupt
2. Handlers for the hard fault and any other faults that may occur in the user application
3. Checks of the reset status registers (`ACS_RESET_STATUS` and `RESET_DIG_STATUS`) at startup to confirm the cause of the most recent system reset

If any of these error modes is detected, a device needs to perform a set of recovery actions. Possible steps include:

1. Checking the available status information to determine what can be done
2. Trigger a hardware reset of specific system blocks or even the whole device, if the system state indicates that a hardware reset is needed
3. Setting timeout flags to trigger an early exit from blocking loops and application level error handling
4. Signaling an external device in the system using a GPIO or interface for error handling at the extended system level

If these steps do not result in system recovery, more drastic steps can be taken to ensure that the application does not waste the system’s battery life.

For example, a user application could put the RSL15 device into Sleep Mode with a wakeup into an alternate error handling state. When using this kind of handling, best practice is for the error handling state to:

- Be different from any other Sleep Mode and Wakeup implemented by the application.
- Design this Sleep State to be in the lowest possible power state.
- Perform more rigorous checks of the device before switching to the normal state of operation.

This kind of error handling routine allows the application to retain limited state information, and can extend the overall system's battery life by keeping the RSL15 device in a lower power mode.

NOTE: Debugging the causes of watchdog interrupts, resets, and faults uses the same strategies and information as handling these errors.

5.7.2 Watchdog Interrupts

Expiry of the watchdog timer is often the first sign that application execution has failed. The watchdog timer interrupt triggers when the watchdog has not been refreshed within a defined number of system operating cycles. If another interrupt were to occur, the system would reset with the `RESET_DIG_STATUS` register indicating that a watchdog reset had occurred.

When a user application handles a watchdog interrupt, the system state has not been reset—which typically simplifies the identification of the causes of errors and improves debugging. Wherever possible, we recommend that an application use this interrupt to evaluate the state of the RSL15 device and to proceed appropriately. To assist in this handling, the watchdog timer interrupt handler can use:

- Application state variables
- Device register settings and status bits
- The system context stored onto the stack frame, including the core processor registers R0 to R3, R12, link register (LR), program counter (PC), and processor status register (PSR).

5.7.3 Resets

If a reset has occurred, the reset status registers (`ACS_RESET_STATUS` and `RESET_DIG_STATUS`) indicate what events or system state has triggered the reset. These registers and the possible reset causes are discussed in more detail in Section 1.1 “Resets” on page 1 from the *RSL15 Hardware Reference*.

NOTE: We recommend clearing all reset status flags in these registers at the start of application execution (after the reset source has been determined), to allow future executions to determine the cause of a reset or resets. To clear the status bits that indicate the source of a reset, the `RESET_DIG_STATUS` register must be cleared before the `ACS_RESET_STATUS` register.

5.7.4 Faults and Lockup

If the Arm Cortex-M33 processor encounters a fault condition, it will enter into a fault handler. Faults that can be detected include:

- Bus faults indicating an error in physically accessing a requested memory location
- Memory management faults indicating an error in memory management, such as issues during exception stacking, and accesses to memory that exists but cannot be accessed in the current system state (instruction or data access violations)
- Usage faults indicating an error in the code, such as:
  - Division by zero
  - Stack overflows
  - Unaligned data or code accesses
  - Invalid instructions
- Secure faults where the application violates the security requirements defined for the current processor state

The Arm Cortex-M33 processor is required to handle all faults, with faults being promoted to be handled by the Hard Fault handler if a specific fault handler is unavailable. A fault is escalated to the hard fault handler if:

- A fault handler causes the same kind of fault as the one it is servicing. This escalation to HardFault occurs because a fault handler cannot preempt itself; it must have the same priority as the current execution priority level.
- A fault handler causes a fault with the same or lower priority as the fault it is servicing. This is because the handler for the new fault cannot preempt the currently executing fault handler.
- An exception handler causes a fault for which the priority is the same as, or lower than, the currently executing exception.
- A fault occurs and the handler for that fault is not enabled.

If a fault occurs that cannot be handled (including faults in the NMI handler, or faults that occur while handling a hard fault), the Arm Cortex-M33 processor enters into a lockup state. The processor remains in this state until the core is reset or is halted by a debugger. In the lockup state, the program counter (PC) is forced to 0xEFFFFFFE.

If you encounter a fault, there are several items that can be used to figure out why the fault has occurred. The fault handling provides:

- The Configurable Fault Status Register (CFSR) that indicates the causes of bus, memory management, and usage faults

- The Bus Fault Address Register (BFAR) and Memory Management Address Register (MMAR) provide the address accessed that have caused the fault to occur, when a bus fault or memory management fault occurs at a known address (only valid when the corresponding bit in CFSR is set).
- The Secure Fault Status Register (SFSR) that indicates the causes of secure faults
- The Hard Fault Status Register (HFSR), which indicates if a hard fault has been triggered directly due to a debug event or failed vector fetch, or if it triggering is due to a fault that has been promoted to a hard fault.
- The system context stored onto the stack frame, including the core processor registers R0 to R3, R12, link register (LR), program counter (PC), and processor status register (PSR).

5.8 THIRD-PARTY TOOLS

SEGGER SystemView is a third-party tool available from SEGGER. It provides hooks for tracking and analyzing events that happen during runtime and can be used with RSL15 to provide diagnostics and visibility into your program execution. See https://www.segger.com/products/development-tools/systemview/ for details on its operation and to download the tool.

CHAPTER 6

RSL15 Flash Variants

RSL15 flash variants include devices with two different flash memory configurations. In both cases, the RSL15 flash is divided into two flash arrays that can both be read from with equal efficiency for any application use cases (including both application code and application data). These flash arrays have the following characteristics:

- The flash code array is organized into 2048-byte sectors, which make it ideal for storing larger items or items that are static.
- The flash data array is organized into 256-byte sectors, which make it ideal for storing smaller items or items that may be updated.

6.1 AVAILABLE FLASH VARIANTS WITH RSL15

The available RSL15 flash variants are:

- RSL15 (RSL15-512)
    - This variant provides 512 KB of flash memory, divided into a code flash array of 352 KB and a data flash array of 160 KB.
    - This is the default RSL15 variant, and all sample applications and device configuration is configured to work with this variant by default.
    - When debugging code running on this variant, specify **RSL15** as the device name on the debugger tab of the debug configuration.
- RSL15-284:
    - This variant provides 284 KB of flash memory, divided into a code flash array of 264 KB and a data flash array of 20 KB.
    - This variant uses different build configurations that are appropriate for the reduced memory layout. To configure the tools for use with this variant, see Section 6.1 “Available Flash Variants with RSL15”.
    - When debugging code running on this variant, specify **RSL15-284** as the device name on the debugger tab of the debug configuration.

NOTE: Debugging an application built and configured for RSL15-284 on an RSL15-512 device functions correctly. Debugging an application built and configured for an RSL15-512 on a RSL15-284 device fails if any memory accesses are attempted to the memories that are not instantiated on RSL15-284.

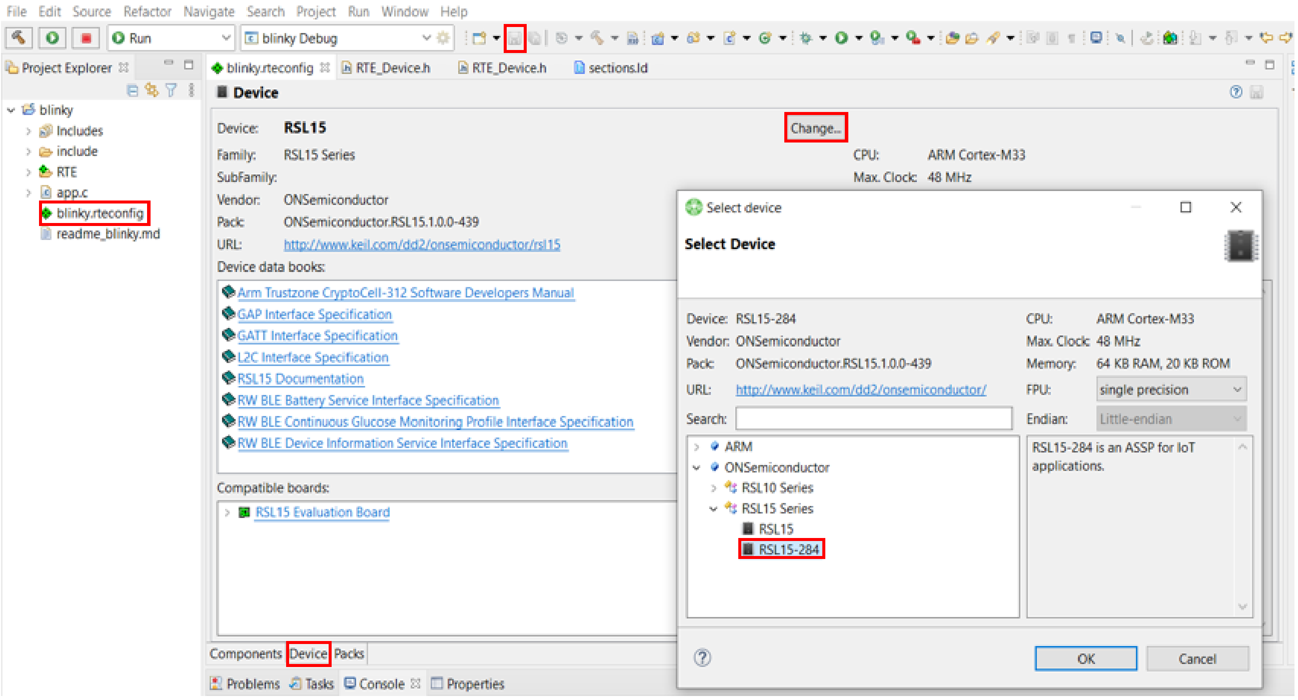
IMPORTANT: As the RSL15-284 variant has a reduced memory footprint, users must take extra care to ensure that their use case fits into memory. Use of this variant limits the size of user applications and combinations of key firmware components, including the Bluetooth stack, Firmware over the air (FOTA) update functionality, and security operations using the Arm CryptoCell-312 that can be supported.

6.2 BUILDING AN APPLICATION FOR AN RSL15-284 DEVICE

To build an application for an RSL15-284 device, a user needs to:

1. Select the RSL15-284 device.
2. Clone the *RTE_Device.h* information provided for RSL15-512 devices into the RSL15-284 information folder
3. Right click on the project, go to **Properties** > **C/C++ Build** > **Settings** > **Tool Settings** > **Cross C Linker** > **General** and change the path from **${workspace_loc:/${ProjName}}/RTE/Device/RSL15/sections.ld** to **${workspace_loc:/${ProjName}}/RTE/Device/RSL15-284/sections.ld**.

For the onsemi IDE, selecting the device can be done from the *<app>.rteconfig* file editor, **Device** tab, **Change** interface screen as shown in the "Selecting the RSL15-284 Device" figure (Figure 24), and cloning the *RTE_Device.h* information copies files from one directory to another as shown in the "Cloning the RTE_Device.h Configuration" figure (Figure 25).



Device

Device: RSL15
Family: RSL15 Series
SubFamily:
Vendor: ONSemiconductor
Pack: ONSemiconductor.RSL15.1.0.0-439
URL: http://www.keil.com/dd2/onsemiconductor/rsl15
CPU: ARM Cortex-M33
Max. Clock: 48 MHz

Device data books:
- Arm Trustzone CryptoCell-312 Software Developers Manual
- GAP Interface Specification
- GATT Interface Specification
- L2C Interface Specification
- RSL15 Documentation
- RW BLE Battery Service Interface Specification
- RW BLE Continuous Glucose Monitoring Profile Interface Specification
- RW BLE Device Information Service Interface Specification

Compatible boards:
- RSL15 Evaluation Board

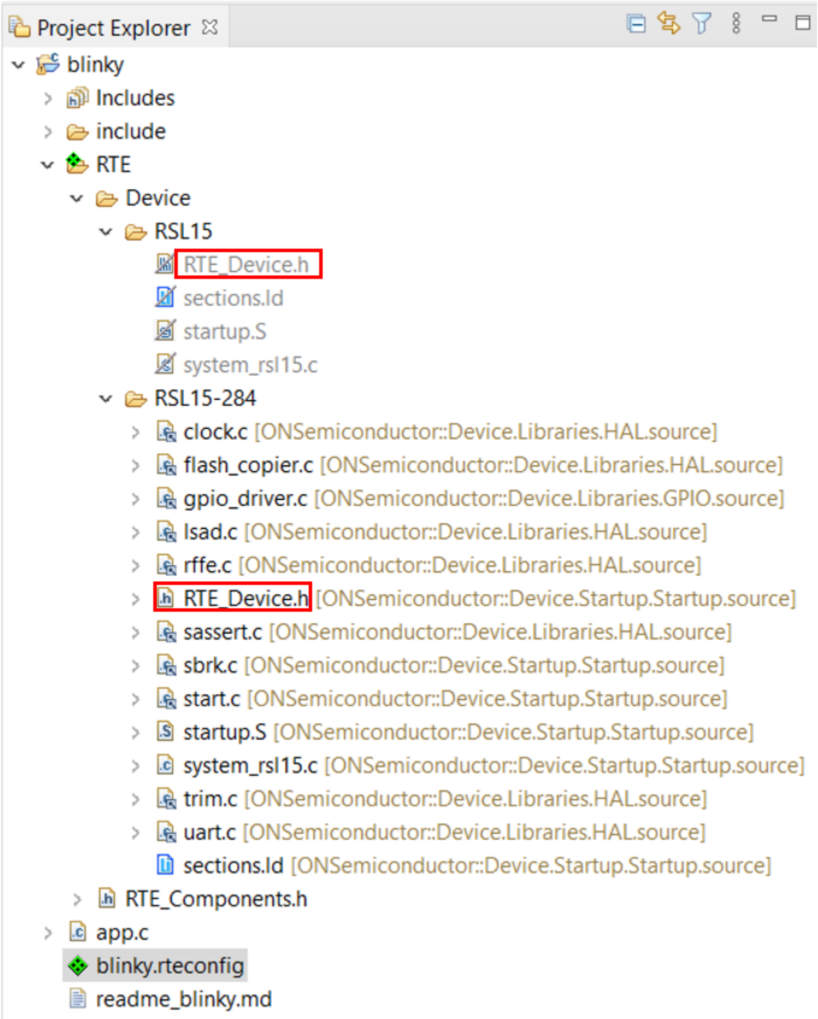
Select Device

Device: RSL15-284
Vendor: ONSemiconductor
Pack: ONSemiconductor.RSL15.1.0.0-439
URL: http://www.keil.com/dd2/onsemiconductor/
Search:
CPU: ARM Cortex-M33
Max. Clock: 48 MHz
Memory: 64 KB RAM, 20 KB ROM
FPU: single precision
Endian: Little-endian

- ARM
- ONSemiconductor
  - RSL10 Series
  - RSL15 Series
    - RSL15
    - RSL15-284

RSL15-284 is an ASSP for IoT applications.

Figure 24. Selecting the RSL15-284 Device



Project Explorer

- blinky
  - Includes
  - include
  - RTE
    - Device
      - RSL15
        - RTE_Device.h
        - sections.ld
        - startup.S
        - system_rsl15.c
      - RSL15-284
        - clock.c [ONSemiconductor::Device.Libraries.HAL.source]
        - flash_copier.c [ONSemiconductor::Device.Libraries.HAL.source]
        - gpio_driver.c [ONSemiconductor::Device.Libraries.GPIO.source]
        - lsad.c [ONSemiconductor::Device.Libraries.HAL.source]
        - rffe.c [ONSemiconductor::Device.Libraries.HAL.source]
        - RTE_Device.h [ONSemiconductor::Device.Startup.Startup.source]
        - sassert.c [ONSemiconductor::Device.Libraries.HAL.source]
        - sbrk.c [ONSemiconductor::Device.Startup.Startup.source]
        - start.c [ONSemiconductor::Device.Startup.Startup.source]
        - startup.S [ONSemiconductor::Device.Startup.Startup.source]
        - system_rsl15.c [ONSemiconductor::Device.Startup.Startup.source]
        - trim.c [ONSemiconductor::Device.Libraries.HAL.source]
        - uart.c [ONSemiconductor::Device.Libraries.HAL.source]
        - sections.ld [ONSemiconductor::Device.Startup.Startup.source]
    - RTE_Components.h
  - app.c
  - blinky.rteconfig
  - readme_blinky.md

Figure 25. Cloning the RTE_Device.h Configuration

For the Keil µVision IDE, selecting the device can be done from the *Device* tab of the *"Options for Target 'RSL15'..."* menu as shown in the "Opening "Options for Target 'RSL15'..."" figure (Figure 26) and the "Selecting the RSL15-284 Device" figure (Figure 27), and cloning the *RTE_Device.h* information from the project's *RTE→Device→RSL15* directory to the *RTE→Device→RSL15-284* directory. In addition, modify the location of the scatter file by selecting the **Linker** tab of the **Options for Target 'RSL15'...** menu and change the location from **.\RTE\Device\RSL15\sections.sct** to **.\RTE\Device\RSL15-284\sections.sct**.

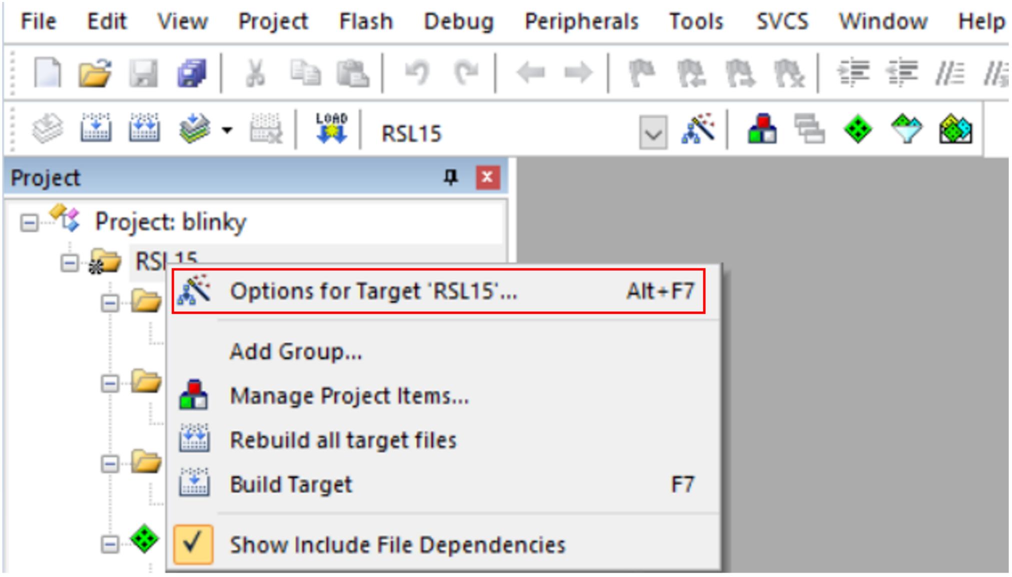
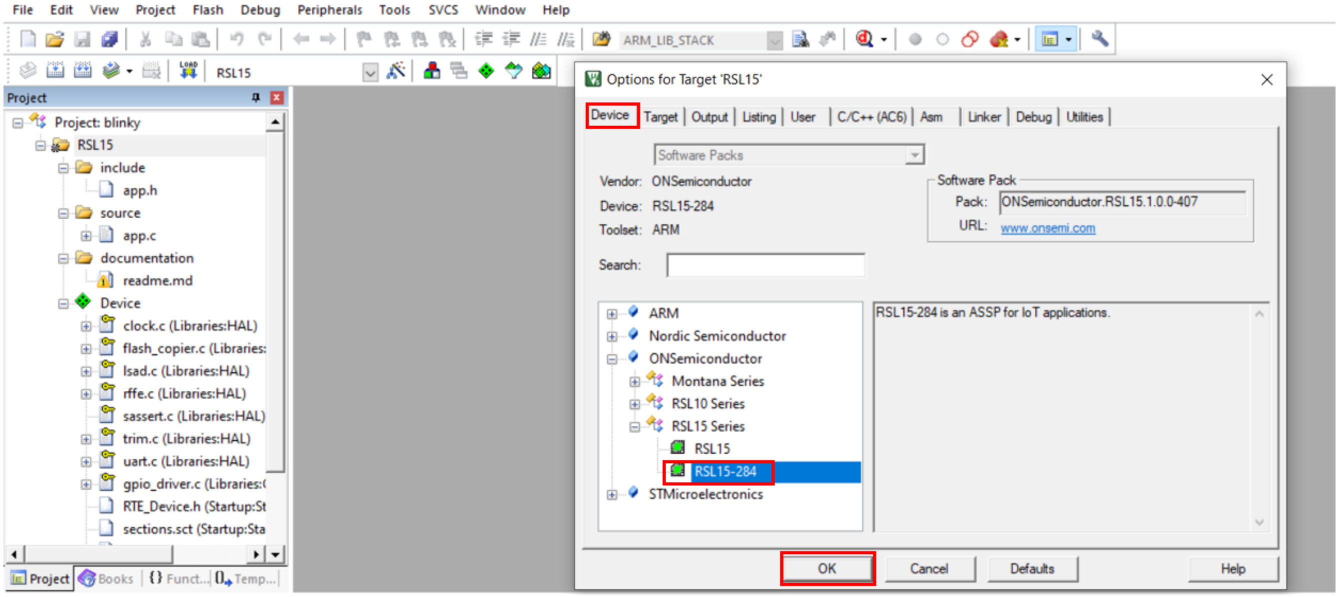


Figure 26. Opening "Options for Target 'RSL15'..."

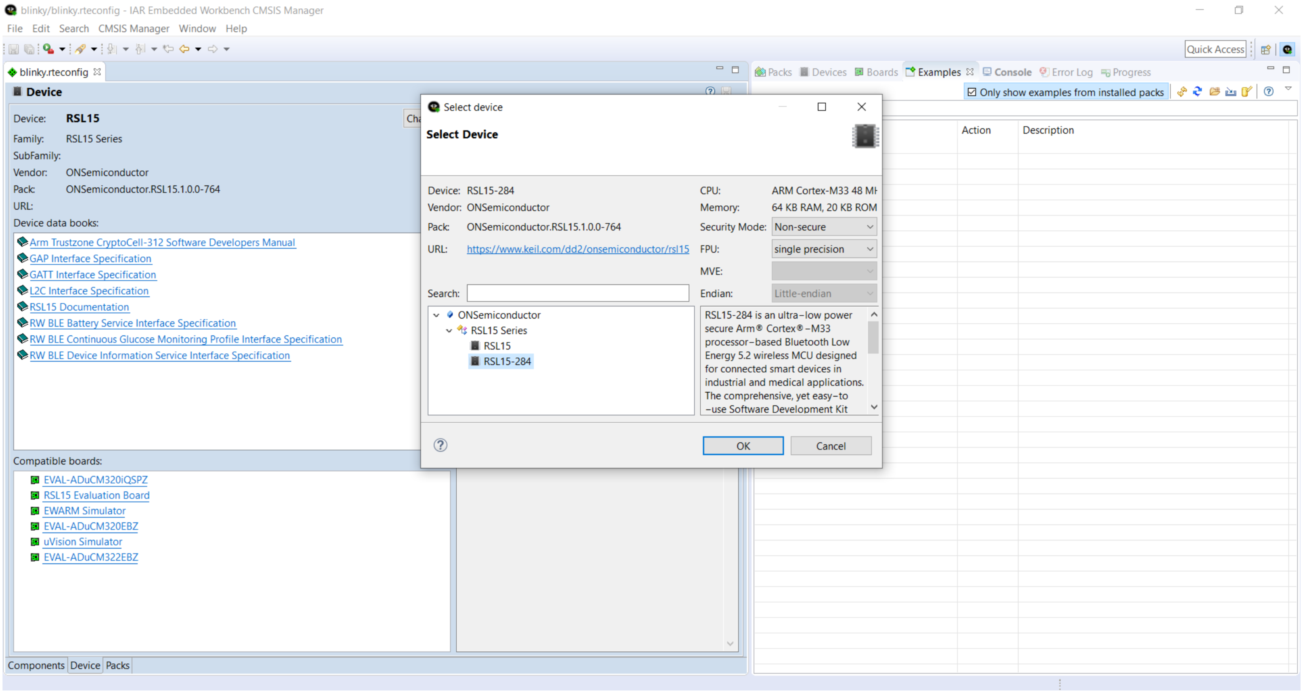


Vendor: ONSemiconductor
Device: RSL15-284
Toolset: ARM
Pack: ONSemiconductor.RSL15.1.0.0-407
URL: www.onsemi.com

RSL15-284 is an ASSP for IoT applications.

Figure 27. Selecting the RSL15-284 Device

For the IAR Embedded Workbench IDE, selecting the device can be done from the **<app>**.*rteconfig* file editor, **Device** tab, **Change** interface screen (as shown in the "Selecting the RSL15-284 Device in the IAR CMSIS Manager" figure (Figure 28)), and cloning the *RTE_Device.h* information from the project's **RTE** > **Device** > **RSL15** directory to the **RTE** > **Device** > **RSL15-284** directory. In addition, modify the location of the linker configuration file by selecting the **Linker** tab in the **Options,** and changing the Linker configuration file location from **$PROJ_DIR$\RTE\Device\RSL15\sections.icf** to **$PROJ_DIR$\RTE\Device\RSL15-284\sections.icf** (as shown in the "Modifying the Linker Configuration Location" figure (Figure 29)).



Device

Device: **RSL15**
Family: RSL15 Series
SubFamily:
Vendor: ONSemiconductor
Pack: ONSemiconductor.RSL15.1.0.0-764
URL:
Device data books:

- Arm Trustzone CryptoCell-312 Software Developers Manual
- GAP Interface Specification
- GATT Interface Specification
- L2C Interface Specification
- RSL15 Documentation
- RW BLE Battery Service Interface Specification
- RW BLE Continuous Glucose Monitoring Profile Interface Specification
- RW BLE Device Information Service Interface Specification

Compatible boards:

- EVAL-ADuCM320iQSPZ
- RSL15 Evaluation Board
- EWARM Simulator
- EVAL-ADuCM320EBZ
- uVision Simulator
- EVAL-ADuCM322EBZ

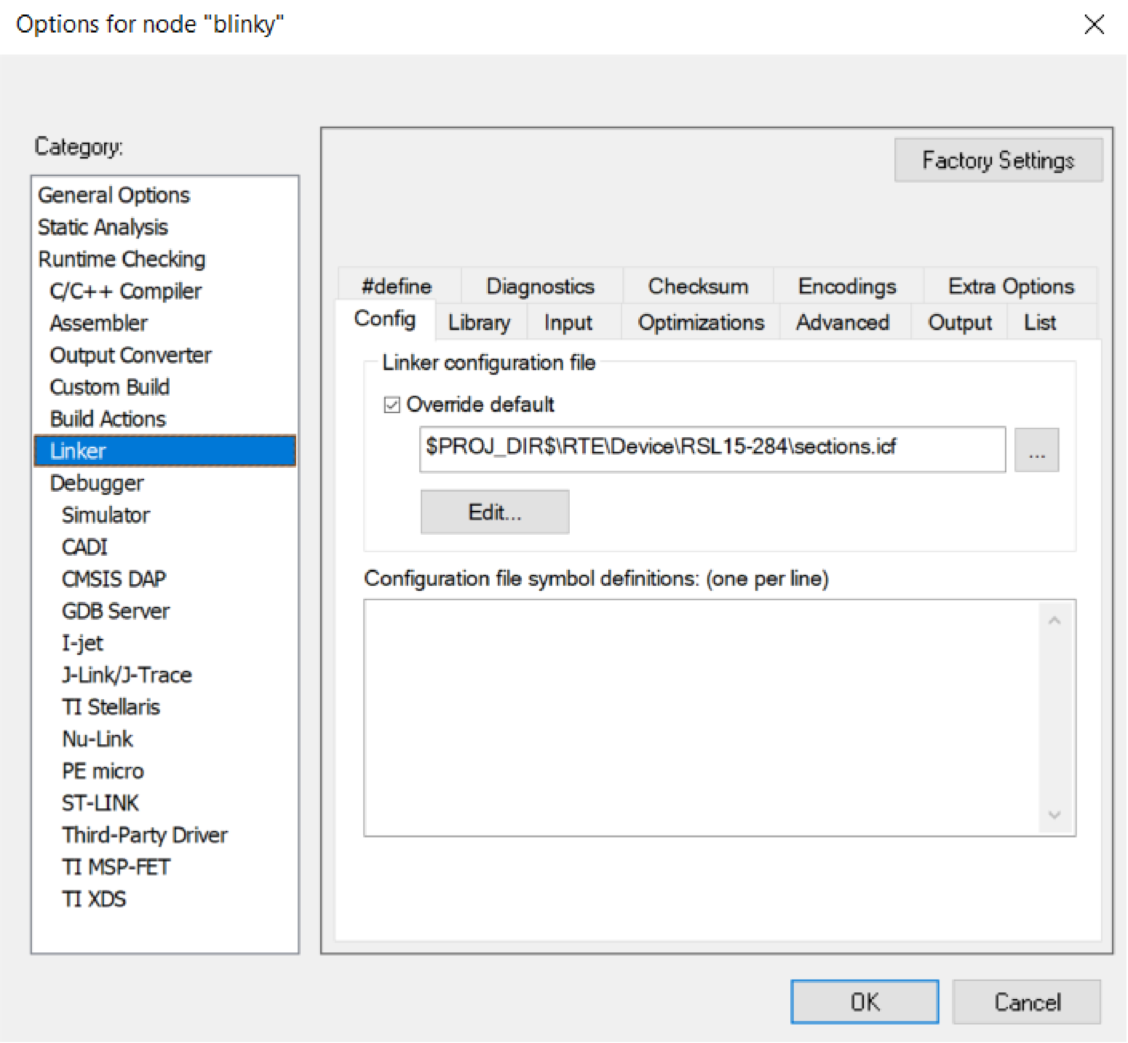
Select Device

Device: RSL15-284
Vendor: ONSemiconductor
Pack: ONSemiconductor.RSL15.1.0.0-764
URL: https://www.keil.com/dd2/onsemiconductor/rsl15
CPU: ARM Cortex-M33 48 MH
Memory: 64 KB RAM, 20 KB ROM
Security Mode: Non-secure
FPU: single precision
MVE:
Endian: Little-endian
Search:

- ONSemiconductor
  - RSL15 Series
    - RSL15
    - RSL15-284

RSL15-284 is an ultra−low power secure Arm® Cortex®−M33 processor−based Bluetooth Low Energy 5.2 wireless MCU designed for connected smart devices in industrial and medical applications. The comprehensive, yet easy−to −use Software Development Kit

Figure 28. Selecting the RSL15-284 Device in the IAR CMSIS Manager



Linker configuration file

☑ Override default

$PROJ_DIR$\RTE\Device\RSL15-284\sections.icf

Configuration file symbol definitions: (one per line)

Figure 29. Modifying the Linker Configuration Location

6.3 RSL15 PACKAGES

The RSL15 device contains 16 general purpose input/output (GPIO) pads. However, some RSL15 package variants do not have access to all 16 GPIO pads.

Refer to your product datasheet for information about which GPIOs are externally accessible.

CHAPTER 7

Running from Flash vs Running from RAM

Data that is meant to be written to flash cannot be read from flash while the flash programming interface is open. If an application loads a value from flash to a register, then completes a write on a single word and closes the flash programming interface before reading the next word, everything works fine. But in sequential writes, the system cannot close the flash programming interface between loads. The next value has to be loaded right away, and if that next value is located in flash, the process fails because the flash programming interface is open. Running from RAM is a way to keep this from happening.

7.1 FLASH VS. RAM IN SAMPLE APPLICATIONS

Several of the sample applications in the CMSIS-Pack have sequential writes and need to run from RAM. The ones that run from flash use the *sections.ld* linker script, as shown in Section 7.1.1 “Applications Running from Flash” on page 42. To run from RAM, the use of the *sections_ram.ld* linker script is required, as seen in Section 7.1.2 “Applications Running from RAM” on page 42.

7.1.1 Applications Running from Flash

The sample applications that run from flash automatically point to the *sections.ld* linker script. To see the pointer to the linker script, right-click on the Project folder or **Alt + Enter** > **Properties** > **C/C++ Build** > **Settings** > **Cross Arm C Linker** > **General** settings:

```
${workspace_loc:/${<project_name>}/RTE/Device/RSL15/sections.ld}
```

Alternatively, you can copy the *sections.ld* file from the *configuration* folder in the CMSIS-Pack to the project, as in this example:

```
${cmsis_pack_
root}/ONSemiconductor/RSL15/<version>/firmware/configuration/GCC/sections.ld
```

7.1.2 Applications Running from RAM

The sample applications that run from RAM point by default to the *sections_ram.ld* linker script. To see the pointer to the linker script, right-click on the Project folder or **Alt + Enter** > **Properties** > **C/C++ Build** > **Settings** > **Cross Arm C Linker** > **General** settings:

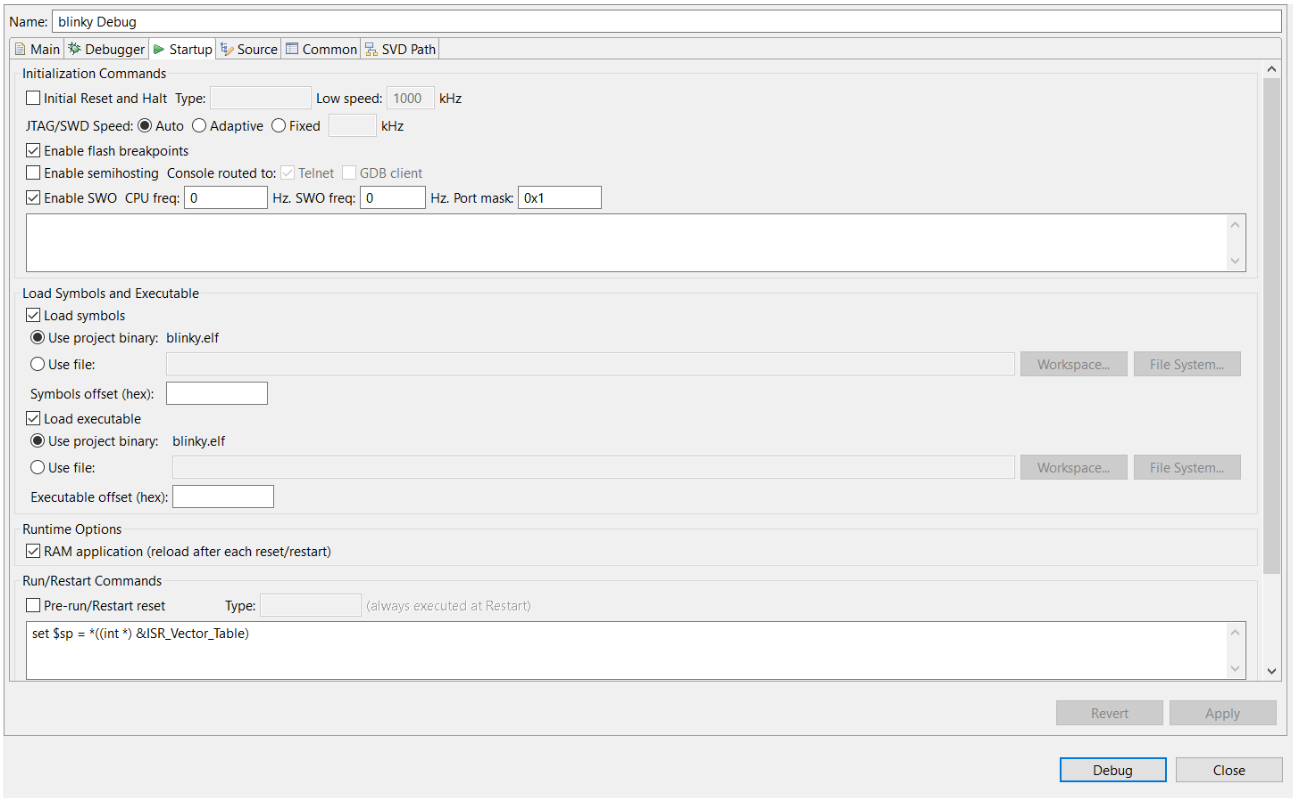
```
${workspace_loc:/${<project_name>}/RTE/Device/RSL15/sections_ram.ld}
```

Alternatively, you can copy the *sections_ram.ld* file from the *configuration* folder in the CMSIS-Pack to the project, as in this example:

```
${cmsis_pack_root}/ONSemiconductor/RSL15/<version>/firmware/configuration/GCC/sections_
ram.ld
```

Click the **Startup** tab of the Debug configuration, as shown in the "RAM Application Box in Debug Configuration Startup Tab" figure (Figure 30). Ensure that the **RAM application** box under **Runtime Options** is checked, and set the stack pointer (`sp`) in the **Run/Restart Commands box** as follows:

```
set $sp = *((int *) &ISR_Vector_Table)
```



Name: blinky Debug

Initialization Commands

☐ Initial Reset and Halt Type: Low speed: 1000 kHz

JTAG/SWD Speed: ● Auto ○ Adaptive ○ Fixed kHz

☑ Enable flash breakpoints

☐ Enable semihosting Console routed to: ☑ Telnet ☐ GDB client

☑ Enable SWO CPU freq: 0 Hz. SWO freq: 0 Hz. Port mask: 0x1

Load Symbols and Executable

☑ Load symbols

● Use project binary: blinky.elf

○ Use file:

Symbols offset (hex):

☑ Load executable

● Use project binary: blinky.elf

○ Use file:

Executable offset (hex):

Runtime Options

☑ RAM application (reload after each reset/restart)

Run/Restart Commands

☐ Pre-run/Restart reset Type: (always executed at Restart)

```
set $sp = *((int *) &ISR_Vector_Table)
```

Figure 30. RAM Application Box in Debug Configuration Startup Tab

CHAPTER 8

Arm Toolchain Support

There are several ways in which the onsemi IDE determines which Arm® GNU toolchain to use when building. Understanding how this works can help prevent confusion and frustration when the development machine has several versions of GNU toolchains installed.

8.1 BASIC INSTALLATION

The onsemi IDE supports the Arm toolchain by installing it in the *arm_tools* directory within the installed IDE location. The build tools `RM` and `Make` are also included with the toolchain, to allow for an easier building experience out of the box.

When the user starts the onsemi IDE with the *IDE.exe* program (whose shortcut is located in Windows menu items), the *arm_tools\bin* directory is added to the path, to give the onsemi IDE access to the toolchain installed with the RSL15 software tools.

Conflicts with toolchain versions can occur in the onsemi IDE, if an Arm-based toolchain has been installed elsewhere or already exists on the path, and the IDE selects that toolchain rather than the one included in *arm_tools*.

8.2 CONFIGURING THE ARM TOOLCHAIN IN THE ONSEMI IDE

All toolchain location options can be accessed by right clicking on the project in the Project Explorer view, selecting **Properties** at the bottom of the pop-up menu, and choosing the **C/C++Build** > **Settings** > **Toolchains** tab. The scope of the toolchain path support is described below.

Global Path:

This is the path used by all workspaces/projects. The global path can be set in the **Toolchains** tab of the project.

Workspace Path:

This is the path used by all projects in the current workspace.

Project Path:

This is the path used by the current project for its toolchain.

8.3 ADDITIONAL SETTINGS

Additional settings (other than the toolchain paths) are located within the MCU preference. These are:

- The Build Tools path (global, workspace, project-based) for tools such as `Make` and `RM`
- The SEGGER J-Link path (global, workspace, project-based) for the location of the SEGGER J-Link executables. This replaces the Run/Debug string substitutions for J-Link previously used.

8.4 FLOATING POINT UNIT (FPU) SUPPORT

The Arm Cortex-M33 processor has support for hardware floating point calculations. The SDK samples and libraries use floats. Therefore, the FPU is powered on whenever the device is in Run Mode.

In low power modes, power to the FPU is appropriately adjusted for each mode, as follows:

- In Sleep Mode when core retention is disabled by the application firmware, the FPU is not powered, so firmware does not need power down the FPU.

- In Sleep Mode when core retention is enabled, firmware powers down the FPU before entering sleep, and powers up the FPU again on wakeup.
- In Standby Mode, the power consumption of the FPU is negligible, so it is left powered on.

The evaluation board/kit (research and development board/kit) (hereinafter the "board") is not a finished product and is as such not available for sale to consumers. The board is only intended for research, development, demonstration and evaluation purposes and should as such only be used in laboratory/development areas by persons with an engineering/technical training and familiar with the risks associated with handling electrical/mechanical components, systems and subsystems. This person assumes full responsibility/liability for proper and safe handling. Any other use, resale or redistribution for any other purpose is strictly prohibited.

The board is delivered "AS IS" and without warranty of any kind including, but not limited to, that the board is production-worthy, that the functions contained in the board will meet your requirements, or that the operation of the board will be uninterrupted or error free. onsemi expressly disclaims all warranties, express, implied or otherwise, including without limitation, warranties of fitness for a particular purpose and non-infringement of intellectual property rights.

onsemi reserves the right to make changes without further notice to any board.

You are responsible for determining whether the board will be suitable for your intended use or application or will achieve your intended results. Prior to using or distributing any systems that have been evaluated, designed or tested using the board, you agree to test and validate your design to confirm the functionality for your application. Any technical, applications or design information or advice, quality characterization, reliability data or other services provided by onsemi shall not constitute any representation or warranty by onsemi, and no additional obligations or liabilities shall arise from onsemi having provided such information or services.

The boards are not designed, intended, or authorized for use in life support systems, or any FDA Class 3 medical devices or medical devices with a similar or equivalent classification in a foreign jurisdiction, or any devices intended for implantation in the human body. Should you purchase or use the board for any such unintended or unauthorized application, you shall indemnify and hold onsemi and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that onsemi was negligent regarding the design or manufacture of the board.

This evaluation board/kit does not fall within the scope of the European Union directives regarding electromagnetic compatibility, restricted substances (RoHS), recycling (WEEE), FCC, CE or UL, and may not meet the technical requirements of these or other related directives.

FCC WARNING – This evaluation board/kit is intended for use for engineering development, demonstration, or evaluation purposes only and is not considered by onsemi to be a finished end product fit for general consumer use. It may generate, use, or radiate radio frequency energy and has not been tested for compliance with the limits of computing devices pursuant to part 15 of FCC rules, which are designed to provide reasonable protection against radio frequency interference. Operation of this equipment may cause interference with radio communications, in which case the user shall be responsible, at its expense, to take whatever measures may be required to correct this interference.

onsemi does not convey any license under its patent rights nor the rights of others.

LIMITATIONS OF LIABILITY: onsemi shall not be liable for any special, consequential, incidental, indirect or punitive damages, including, but not limited to the costs of requalification, delay, loss of profits or goodwill, arising out of or in connection with the board, even if onsemi is advised of the possibility of such damages. In no event shall onsemi's aggregate liability from any obligation arising out of or in connection with the board, under any theory of liability, exceed the purchase price paid for the board, if any.

For more information and documentation, please visit www.onsemi.com.

PUBLICATION ORDERING INFORMATION

LITERATURE FULFILLMENT:
Literature Distribution Center for onsemi
19521 E. 32nd Pkwy, Aurora, Colorado 80011 USA
Phone: 303-675-2175 or 800-344-3860 Toll Free USA/Canada
Fax: 303-675-2176 or 800-344-3867 Toll Free USA/Canada
Email: orderlit@onsemi.com

N. American Technical Support:
800-282-9855 Toll Free USA/Canada
Europe, Middle East and Africa Technical Support: Phone: 421 33 790 2910

onsemi Website: www.onsemi.com

Order Literature: http://www.onsemi.com/orderlit

For additional information, please contact your local Sales Representative