

RSL10 Getting Started Guide

AND9697/D
November 2019, Rev. P7

© SCILLC, 2019
Previous Edition © 2018
"All Rights Reserved"



ON Semiconductor®

Table of Contents

	Page
1. Introduction	4
1.1 Overview	4
1.2 Intended Audience	4
1.3 Conventions	4
2. Setting Up the Hardware	5
2.1 Prerequisite Hardware	5
2.2 Connecting the Hardware	5
2.3 Preloaded Sample	5
3. Getting Started with the Eclipse-Based ON Semiconductor IDE	7
3.1 Software to Download	7
3.2 ON Semiconductor IDE and RSL10 CMSIS-Pack Installation Procedures	7
3.3 Building Your First Sample Application with the ON Semiconductor IDE	9
3.3.1 Launching the ON Semiconductor IDE	9
3.3.2 Importing the Sample Code	9
3.3.3 Build the Sample Code	11
3.4 Debugging the Sample Code	13
3.4.1 Debugging with the .elf File	13
3.4.2 Peripheral Registers View with the ON Semiconductor IDE	15
4. Getting Started with Keil	19
4.1 Prerequisite Software	19
4.2 RSL10 CMSIS-Pack Installation Procedure	19
4.3 Building Your First Sample Application with the Keil uVision IDE	20
4.3.1 Import the Sample Code	21
4.3.2 Build the Sample Code	22
4.3.3 Debugging the Sample Code	23
4.3.3.1 Preparing J-Link for Debugging	23
4.3.3.2 Debugging Applications	23
5. Getting Started with IAR	25
5.1 Prerequisite Software	25
5.2 RSL10 CMSIS-Pack Installation Procedure	25
5.3 Building Your First Sample Application with the IAR Embedded Workbench	26
5.3.1 Import the Sample Code	26
5.3.2 Building the Sample Code	28
5.3.3 Debugging the Sample Code	29
5.3.3.1 Debugging Applications	29
6. Resolving External CMSIS-Pack Dependencies	32
1 External CMSIS-Pack Dependencies	32
2 Resolving External Dependencies	32
7. Advanced Debugging	35
7.1 Printf Debug Capabilities	35
7.1.1 Adding Printf Debug Capabilities	35
7.2 Debugging Applications that Do Not Start at the Base Address of Flash	35
7.3 Arm Cortex-M3 Core Breakpoints	36

7.4 Debugging with Low Power Sleep Mode	36
7.4.1 Downloading Firmware in Sleep Mode	43
8. More Information	44
8.1 Folder Structure of the RSL10 CMSIS-Pack Installation	44
8.2 Documentation	45
8.2.1 Documentation Included with the CMSIS-Pack	45
8.2.2 Documentation in the RSL10 Documentation Package	49
A. Migrating to CMSIS-Pack	51
A.1 Migrating an Existing Eclipse Project to the CMSIS-Pack Method	51
A.2 Using the Latest RSL10 Firmware in a Previous Version of the Eclipse-Based IDE	52
B. Arm Toolchain Support	53
B.1 Basic Installation	53
B.2 Configuring the Arm Toolchain in the ON Semiconductor IDE	53
B.3 Additional Settings	53

CHAPTER 1

Introduction

1.1 OVERVIEW

RSL10 is a multi-protocol, Bluetooth® 5 certified, radio System on Chip (SoC), with the lowest power consumption in the industry. It is designed to be used in devices that require high performance and advanced wireless features, with minimal system size and maximized battery life. The RSL10 Software Development Kit (SDK) includes firmware, software, example projects, documentation, and development tools. The Eclipse-based ON Semiconductor Integrated Development Environment (IDE) is offered as a free download with optional support for Arm® Keil® µVision® and IAR Embedded Workbench®.

Software components, device and board support information are delivered using the CMSIS-Pack standard. Standard CMSIS-Drivers for peripheral interfaces and FreeRTOS sample applications are supported. With the CMSIS-Pack standard, you can easily go beyond what is included in our software package and have access to a variety of generic Cortex-M software components. If you have existing RSL10 projects and have not used the RSL10 CMSIS-Pack before, see Appendix A, “Migrating to CMSIS-Pack” on page 51 for more information.

The RSL10 SDK allows for rapid development of ultra-low power Bluetooth Low Energy applications. Convenient abstraction decouples user application code from system code, allowing for simple modular code design. Features such as FOTA (Firmware Over-the-Air) can easily be added to any application. Advanced debugging features such as support for SEGGER® RTT help developers monitor and debug code. Sample applications, from Blinky to ble_peripheral_server_bond and everything in between, help get software development moving quickly. An optional Bluetooth mesh networking CMSIS-Pack quickly enables mesh networking for any application. Android and iOS mobile apps are available on their respective app stores to demonstrate and explore RSL10 features.

This document helps you to get started with the RSL10 SDK. It guides you through the process of connecting your RSL10 Evaluation and Development Board, installing an IDE and the CMSIS-Pack, configuring your environment, and building and debugging your first RSL10 application.

NOTE: RSL10 contains a low power DSP processor core; see *RSL10 LPDSP32 Software Package.zip* for more information.

1.2 INTENDED AUDIENCE

This manual is for people who intend to develop applications for RSL10. It assumes that you are familiar with software development activities.

1.3 CONVENTIONS

The following conventions are used in this manual to signify particular types of information:

monospace Commands and their options, error messages, code samples and code snippets.

mono bold A placeholder for the specified information. For example, replace **filename** with the actual name of the file.

bold Graphical user interface labels, such as those for menus, menu items and buttons.

italics File names and path names, or any portion of them.

CHAPTER 2

Setting Up the Hardware

2.1 Prerequisite Hardware

The following items are needed before you can make connections:

- RSL10 Evaluation and Development Board and a micro USB cable
- A computer running Windows

2.2 CONNECTING THE HARDWARE

To connect the Evaluation and Development Board to a computer:

1. Check the jumper positions:

Ensure that the jumper CURRENT is connected and POWER OPTIONS is selected for USB. Also, connect the jumpers TMS, TCK and SWD. Finally, connect the headers P7, P8, P9 and P10 to 3.3 V, as highlighted in Figure 1.

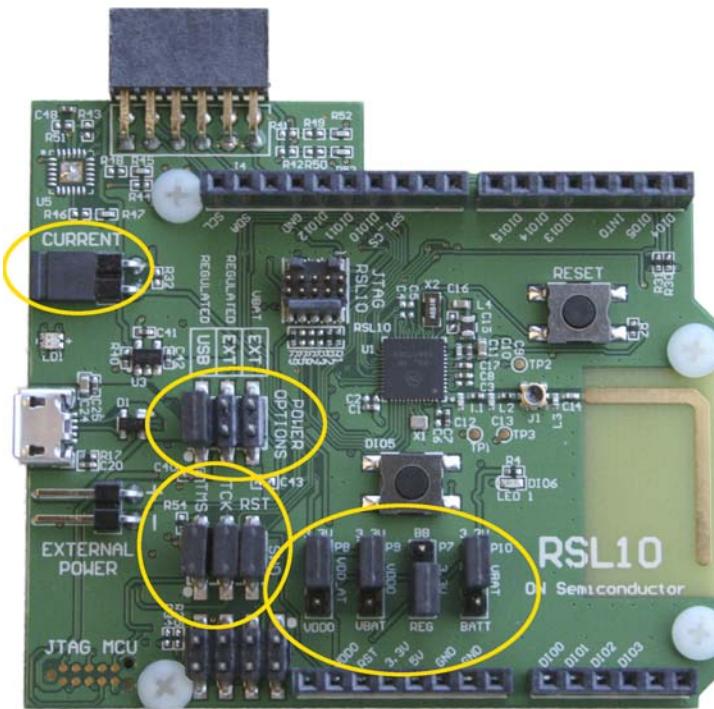


Figure 1. Evaluation and Development Board with Pins and Jumpers for Connection Highlighted

2. Once the jumpers are in the right positions, you can plug the micro USB cable into the socket on the board. The LED close to the USB connector flashes green during the first time plugging in, then turns a steady green once the process is finished.

2.3 PRELOADED SAMPLE

The Evaluation and Development Boards come with one of the following preloaded sample applications:

RSL10 Getting Started Guide

- “Peripheral Device with Sleep Mode” is on boards with a serial number lower than 1741xxxx.
- “Peripheral Device with Server” is on boards with a serial number higher than 1741xxxx.

For more information about sample applications, refer to the *RSL10 Sample Code User's Guide*.

CHAPTER 3

Getting Started with the Eclipse-Based ON Semiconductor IDE

3.1 SOFTWARE TO DOWNLOAD

1. Download the **ON Semiconductor IDE Installer** from www.onsemi.com/RSL10.
2. Download the **RSL10 Software Package** from www.onsemi.com/RSL10 and extract the RSL10 CMSIS-Pack (*ONSemiconductor:RSL10.<version>.pack*) to any temporary folder. (The temporary folder can be on any drive on your computer.)

3.2 ON SEMICONDUCTOR IDE AND RSL10 CMSIS-PACK INSTALLATION PROCEDURES

You can uninstall any previous version of the ON Semiconductor IDE by going to the Windows **Control Panel** and uninstalling **ON Semiconductor IDE** or **RSL10 Development Tools** (depending on how old your previous version is). Optionally, you can skip the uninstallation process and install the new version while keeping the old version(s) available for use.

Install your new ON Semiconductor IDE by running *ON_Semiconductor_IDE.msi*. The ON Semiconductor IDE is installed in this location by default: *C:\Program Files (x86)\ON Semiconductor\IDE_V<version>*.

- You are prompted to install SEGGER J-Link. You need the J-Link software to download and debug applications on the Evaluation and Development Board. The **J-Link Installation Check** screen will guide you through the process of installing J-Link if no valid J-Link installation is found.
- The release version and build number are stored in the *REVISION* text file at the root of the installed ON Semiconductor IDE.

To install the RSL10 CMSIS-Pack:

1. It is important to create a new workspace for each new version of the IDE to ensure compatibility. Create a new workspace at, for example, *c:\workspace* — using either Windows Explorer or the ON Semiconductor Launcher in step 2.
2. Open the ON Semiconductor IDE by going to the Windows Start menu and selecting **ON Semiconductor > ON Semiconductor IDE**. From the ON Semiconductor IDE Launcher screen, browse to your new workspace, select it, and click **Launch**.
3. On the top right corner of the Workbench perspective, click on the Open Perspective icon, select **CMSIS Pack Manager**, and click **Open** (see Figure 2).

NOTE: If you cannot see the **CMSIS-Pack Manager** item, re-install the IDE in your user folder (i.e., *C:\Users\<user_name>*).

RSL10 Getting Started Guide

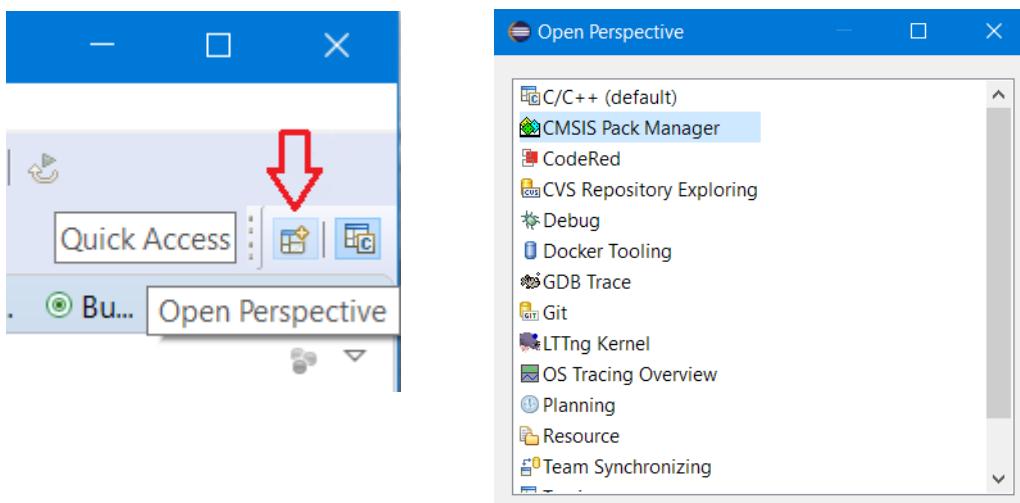


Figure 2. Opening the CMSIS Pack Manager Perspective

4. Click on the Import Existing Packs icon, select your pack file *ONSemiconductor:RSL10.<version>.pack*, where <version> is a number such as 3.1.575, and click Open (see Figure 3).

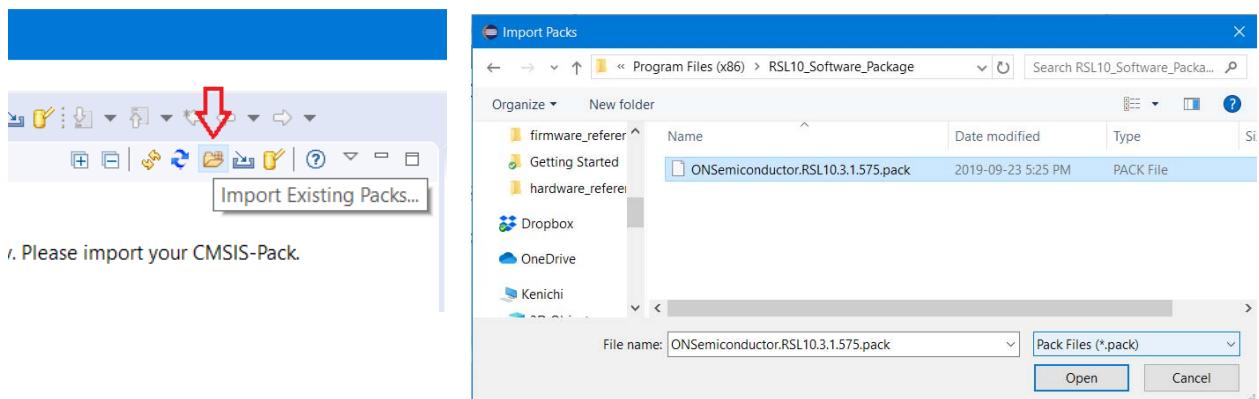


Figure 3. Installing the RSL10 CMSIS-Pack

5. The IDE prompts you to read and accept our license agreement, and then installs the RSL10 CMSIS-Pack in the specified pack root folder.
6. The RSL10 CMSIS-Pack now appears in the list of installed packs. In the **Devices** tab, if you expand **All Devices > ONSemiconductor > RSL10 Series** you can see RSL10 listed there. You can manage your installed packs in the **Packs** tab. Expanding **ONSemiconductor > RSL10** makes the **Pack Properties** tab display the details of the RSL10 CMSIS-Pack. Figure 4 illustrates what the Pack Manager perspective looks like after installation.

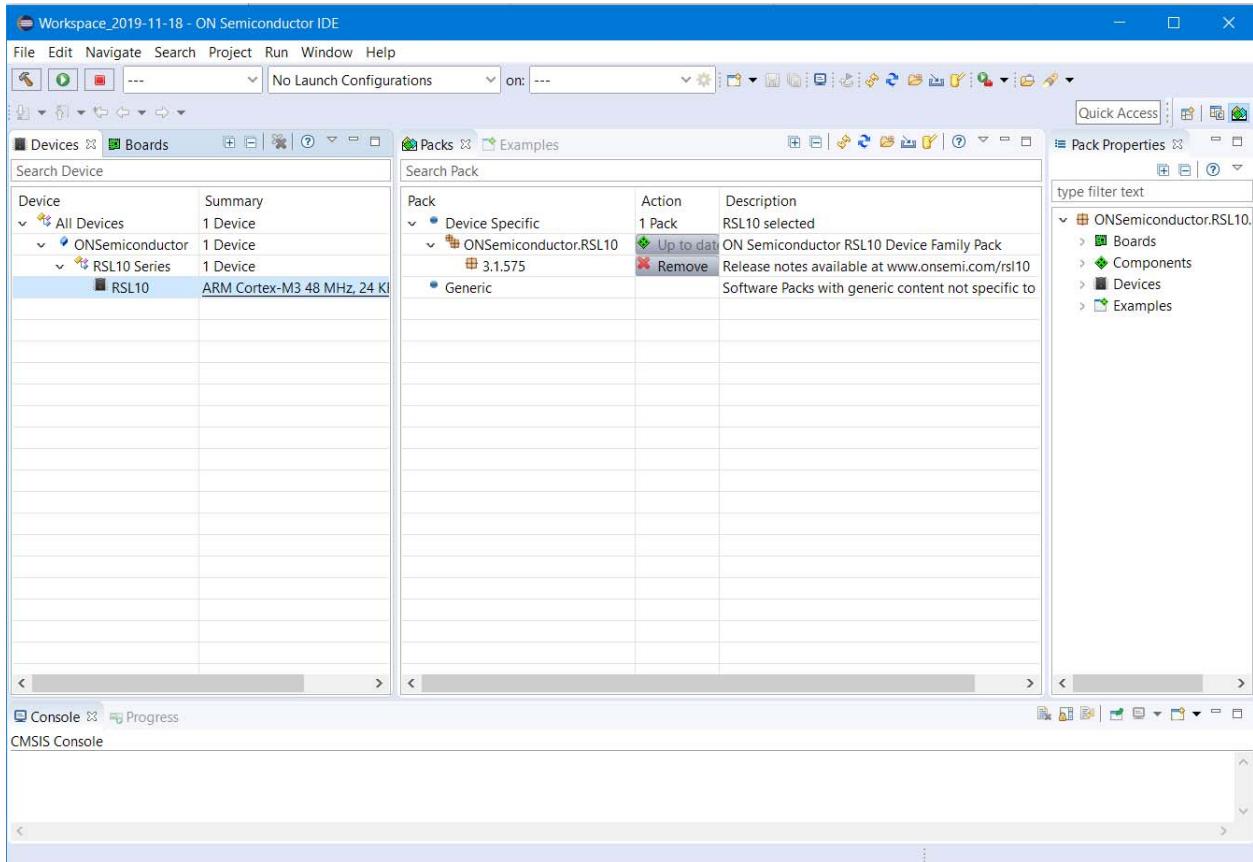


Figure 4. Pack Manager Perspective after RSL10 CMSIS-Pack is Installed

3.3 BUILDING YOUR FIRST SAMPLE APPLICATION WITH THE ON SEMICONDUCTOR IDE

This section guides you through importing and building your first sample application, named *blinky*. This application makes the LED (DIO6) blink on the Evaluation and Development Board.

For more information about the sample applications, see the *RSL10 Sample Code User's Guide*.

3.3.1 Launching the ON Semiconductor IDE

Open the ON Semiconductor IDE by going to the Windows Start menu and selecting **ON Semiconductor > ON Semiconductor IDE**.

3.3.2 Importing the Sample Code

Import the sample code as follows:

1. In the Pack Manager perspective, click on the **Examples** tab to list all the example projects included in the RSL10 CMSIS-Pack.
2. Choose the example project called *blinky*, and click the **Copy** button to import it into your workspace (see Figure 5).

RSL10 Getting Started Guide

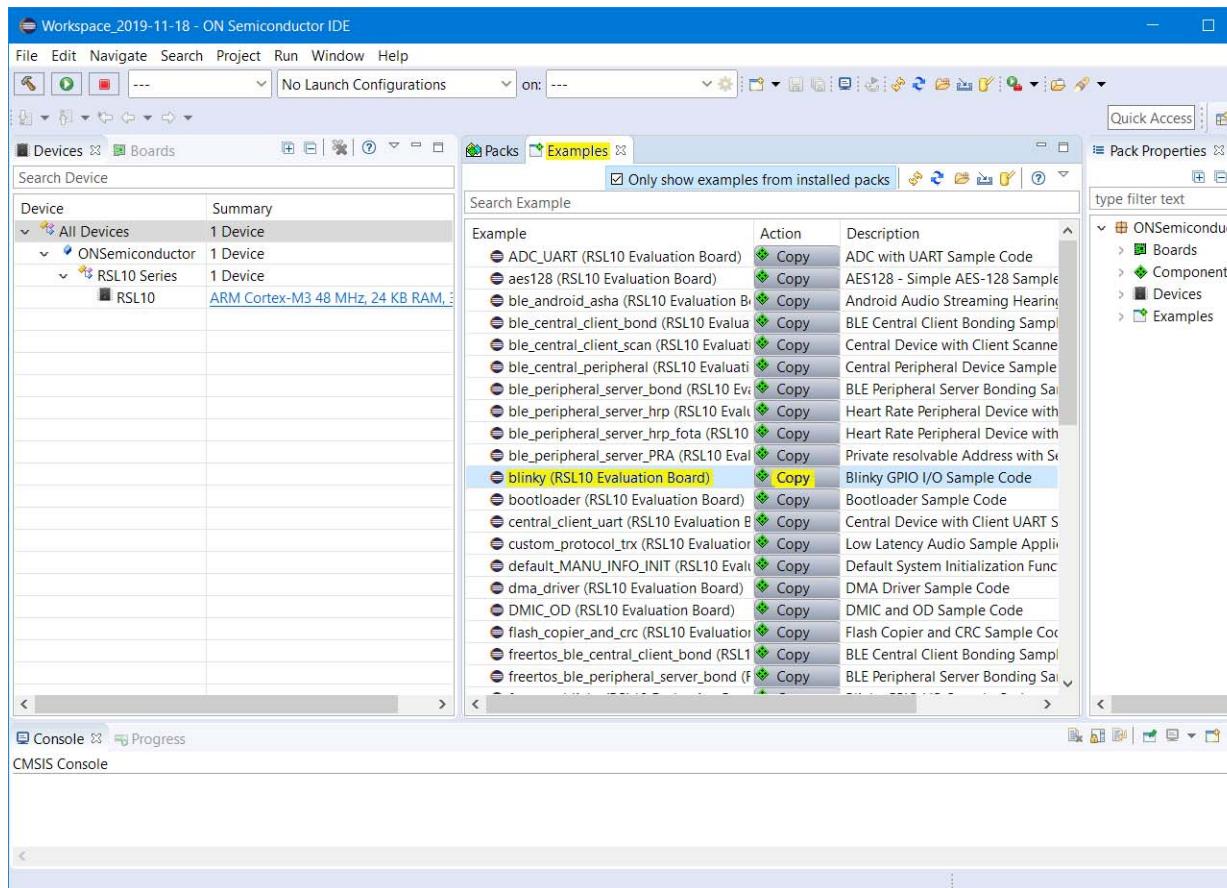


Figure 5. Pack Manager Perspective: Examples Tab

3. The C/C++ perspective opens and displays your newly copied project. In the Project Explorer panel, you can expand your project folder and explore the files inside your project. On the right side, the *blink.y.reconfig* file displays software components. If you expand **Device > Libraries**, you can see the **System library (libsystlib)** and the **Startup (libcmsis)** components selected for *blinky* (see Figure 6).

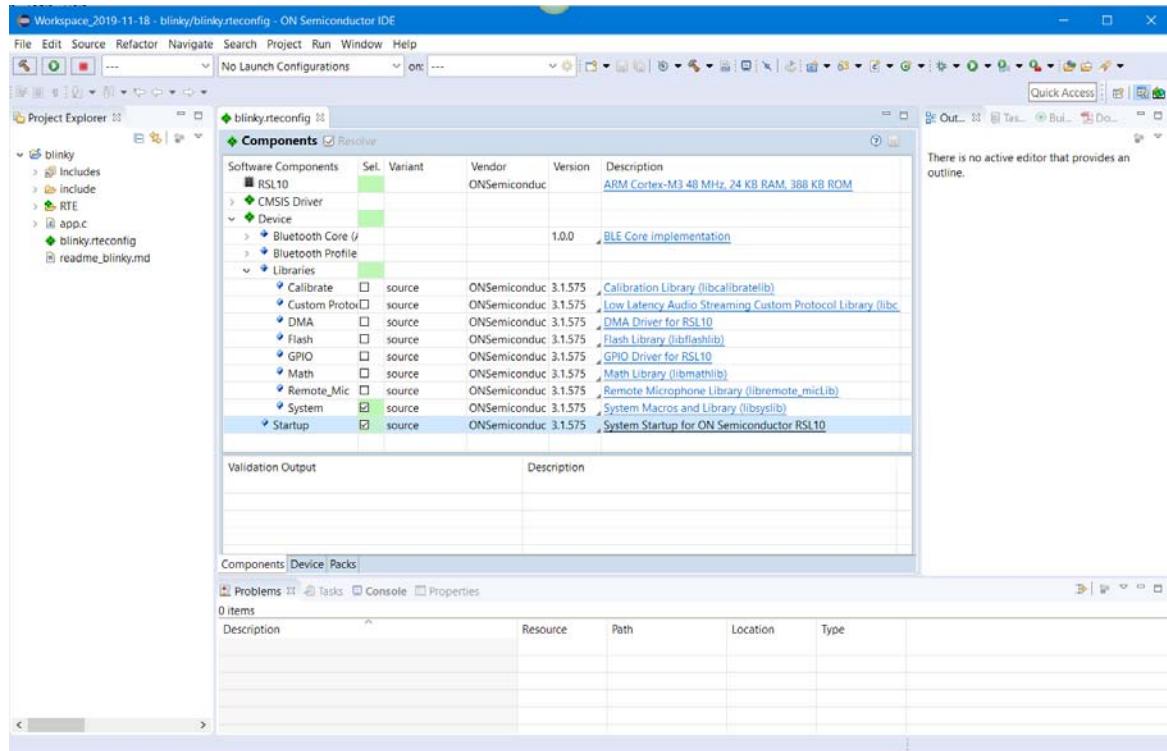


Figure 6. RTE Configuration for the Blinky Example Project in the ON Semiconductor IDE

3.3.3 Build the Sample Code

Follow these steps to build the sample code:

1. Right click on the folder for *blinky* and click **Build Project**. Alternatively, you can select the project and click the Build Project icon, which looks like a hammer, as shown in Figure 7.

RSL10 Getting Started Guide

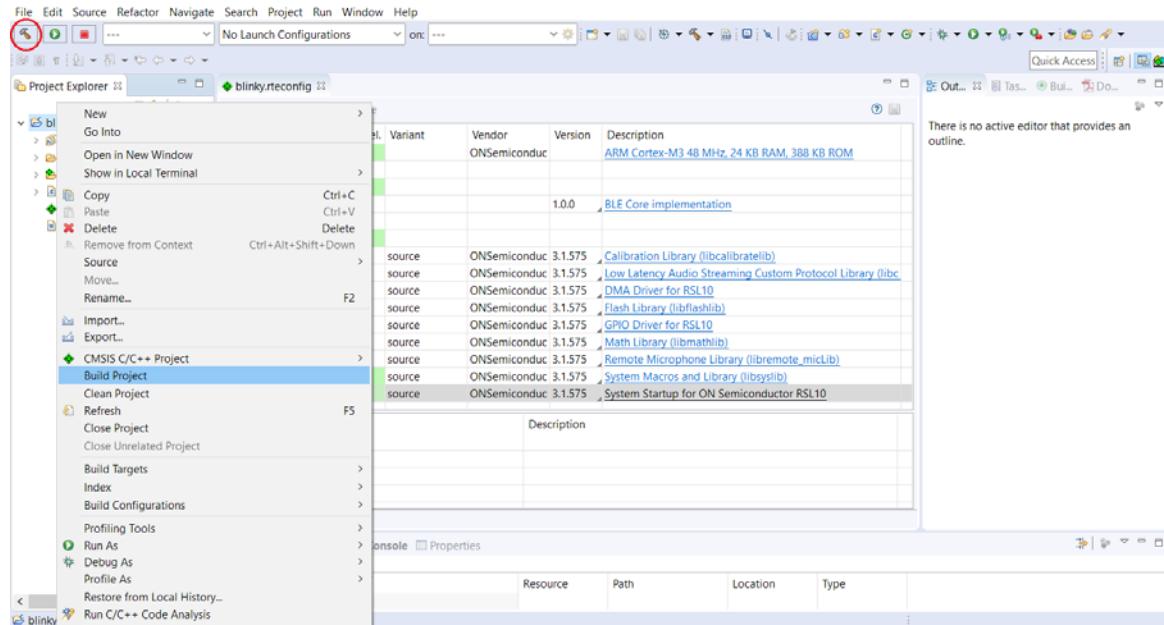


Figure 7. Starting to Build a Project in the ON Semiconductor IDE

2. When the build is running, the output of the build is shown in the ON Semiconductor IDE C/C++ Development Tooling (CDT) Build Console, as illustrated in Figure 8.

```
Validation Output
Description

Components Device Packs
Problems Tasks Console Properties
CDT Build Console [blinky]

Invoking: Cross ARM GNU Create Flash Image
arm-none-eabi-objcopy -O ihex "blinky.elf" "blinky.hex"
Finished building: blinky.hex

Invoking: Cross ARM GNU Print Size
arm-none-eabi-size --format-berkeley "blinky.elf"
text    data     bss   dec   hex filename
4204    100    1036   5340   14dc blinky.elf
Finished building: blinky.size

13:52:29 Build Finished. 0 errors, 0 warnings. (took 7s.970ms)
```

Figure 8. Example of Build Output

3. The key resulting output in Project Explorer, in the *Debug* folder, includes:
 - *blinky.hex*: HEX file for loading into Flash memory
 - *blinky.elf*: Arm® executable file, run from RAM, used for debugging
 - *blinky.map*: map file of the sections and memory usage

These files are shown in Figure 9.

NOTE: You might need to refresh the project to see the three built output files. To do so, right-click on the project name *blinky* and choose **Refresh** from the menu.

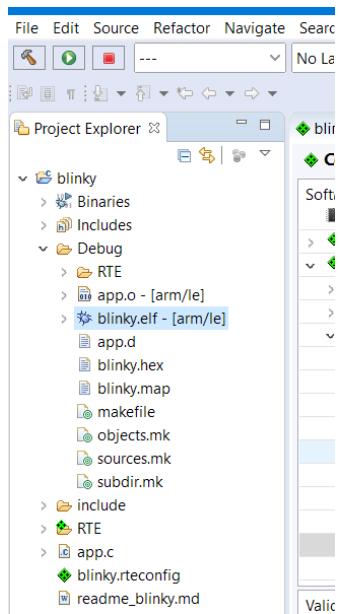


Figure 9. Output Files from Building a Sample Project

3.4 DEBUGGING THE SAMPLE CODE

3.4.1 Debugging with the .elf File

Debug the application using the *.elf* file as follows:

1. Within the **Project Explorer**, right-click on the *blinky.elf* file and select **Debug As > Debug Configurations...**
2. When the **Debug Configurations** dialog appears, right-click on **GDB SEGGER J-Link Debugging** and select **New Configuration**. A new configuration for *blinky* appears under the **GDB SEGGER** heading, with new configuration details in the right side panel.
3. Change to the **Debugger** tab, and enter RSL10 in the **Device Name** field. Ensure that **SWD** is selected as the target interface (as shown in Figure 10).

RSL10 Getting Started Guide

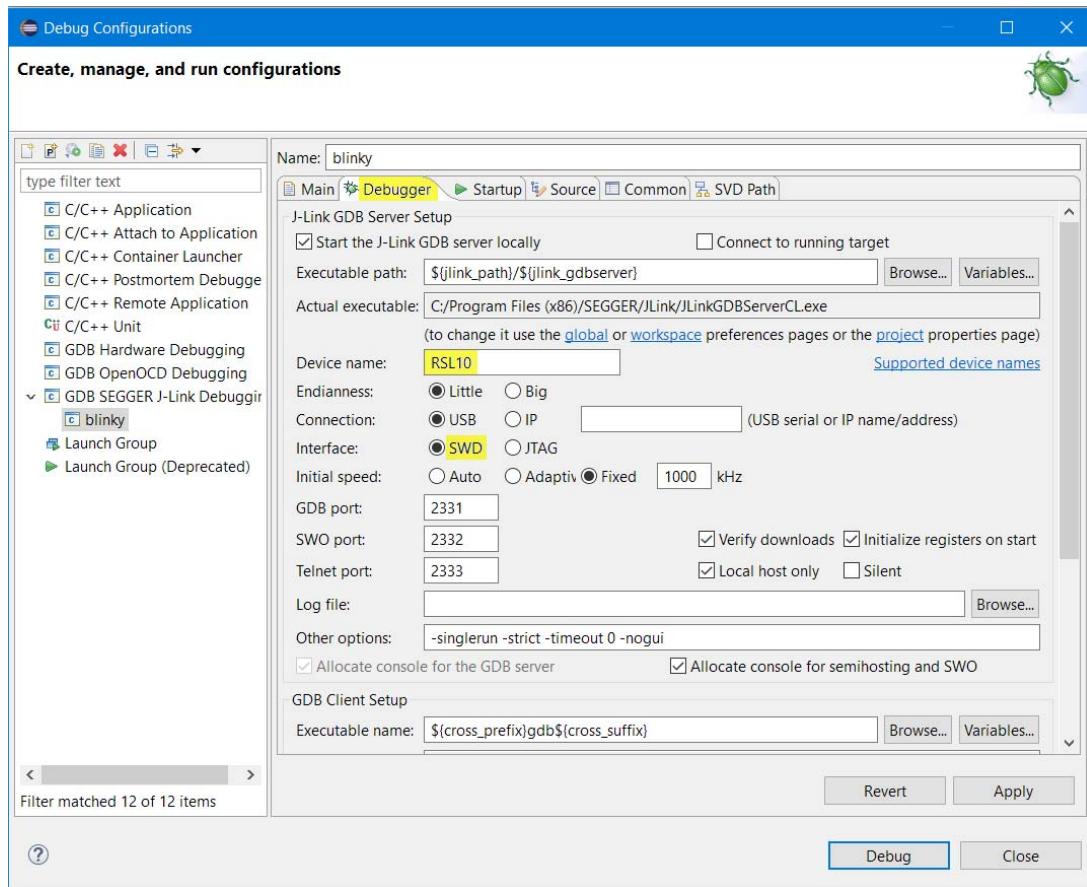


Figure 10. Setting Up a GDB Launch Configuration, Debugger Tab

NOTE: To debug an application that does not start at the first address of flash memory, see Chapter 7, “Advanced Debugging” on page 35.

4. Once the updates to the configuration are completed, make sure that the Evaluation and Development Board is connected to the PC via a micro USB cable, and click **Debug**. J-Link automatically downloads the *blinky* sample code to RSL10’s flash memory.

NOTE: If J-Link does not automatically write your program to RSL10’s flash memory, make sure you are using a compatible J-Link version (see Section 3.2, “ON Semiconductor IDE and RSL10 CMSIS-Pack Installation Procedures” on page 7).

If you are having trouble downloading firmware because an application with Sleep Mode is on the Evaluation and Development Board, see Section 7.4.1, “Downloading Firmware in Sleep Mode” on page 43.

5. You are prompted to switch to the debug perspective. Click **Switch**.
6. The Debug perspective opens and the application runs to *main*, as shown in Figure 11. You can press F6 multiple times to step through the code and observe that the LED changes its state when the application executes the function `Sys_GPIO_Toggle(LED_DIO)`.

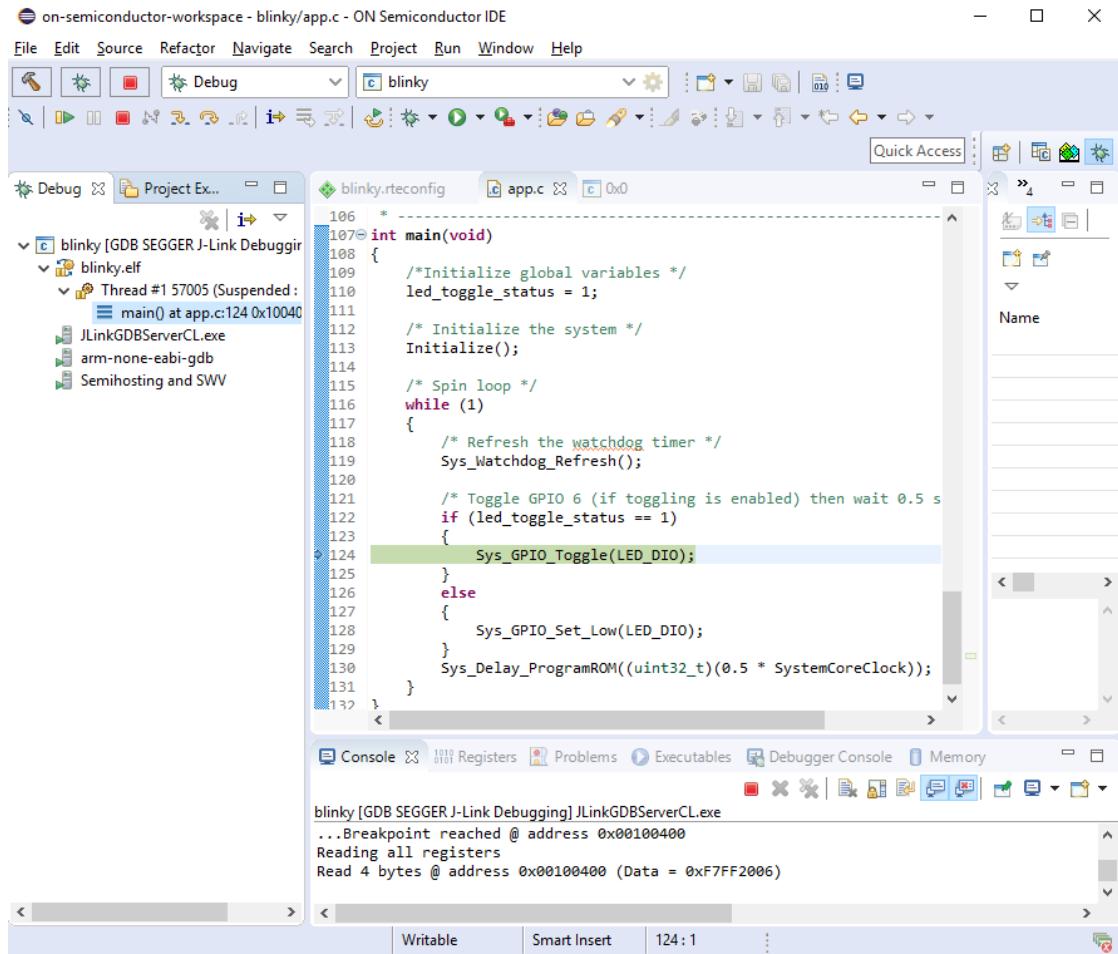


Figure 11. Debug Perspective

3.4.2 Peripheral Registers View with the ON Semiconductor IDE

The ON Semiconductor IDE includes a peripheral register view plugin that enables you to visualize and modify all of the RSL10 registers during a debug session. It can be configured by setting the path to the SVD file in the Debug session.

The following steps demonstrate how to configure and use the Peripheral Registers View with the *Blinky* application:

1. Right click on the *blink.yelf* file, select **Debug As > Debug Configurations**, and open your configuration details set, as described in Section 3.4.1, “Debugging with the .elf File” on page 13.
2. Change to the **SVD Path** tab, and set the path to the *rsl10.svd* file as *C:\Users\<user_id>\ON_Semiconductor\PACK\ONSemiconductor\RSL10\<version>\svd* (see Figure 12). Click **Debug**.

RSL10 Getting Started Guide

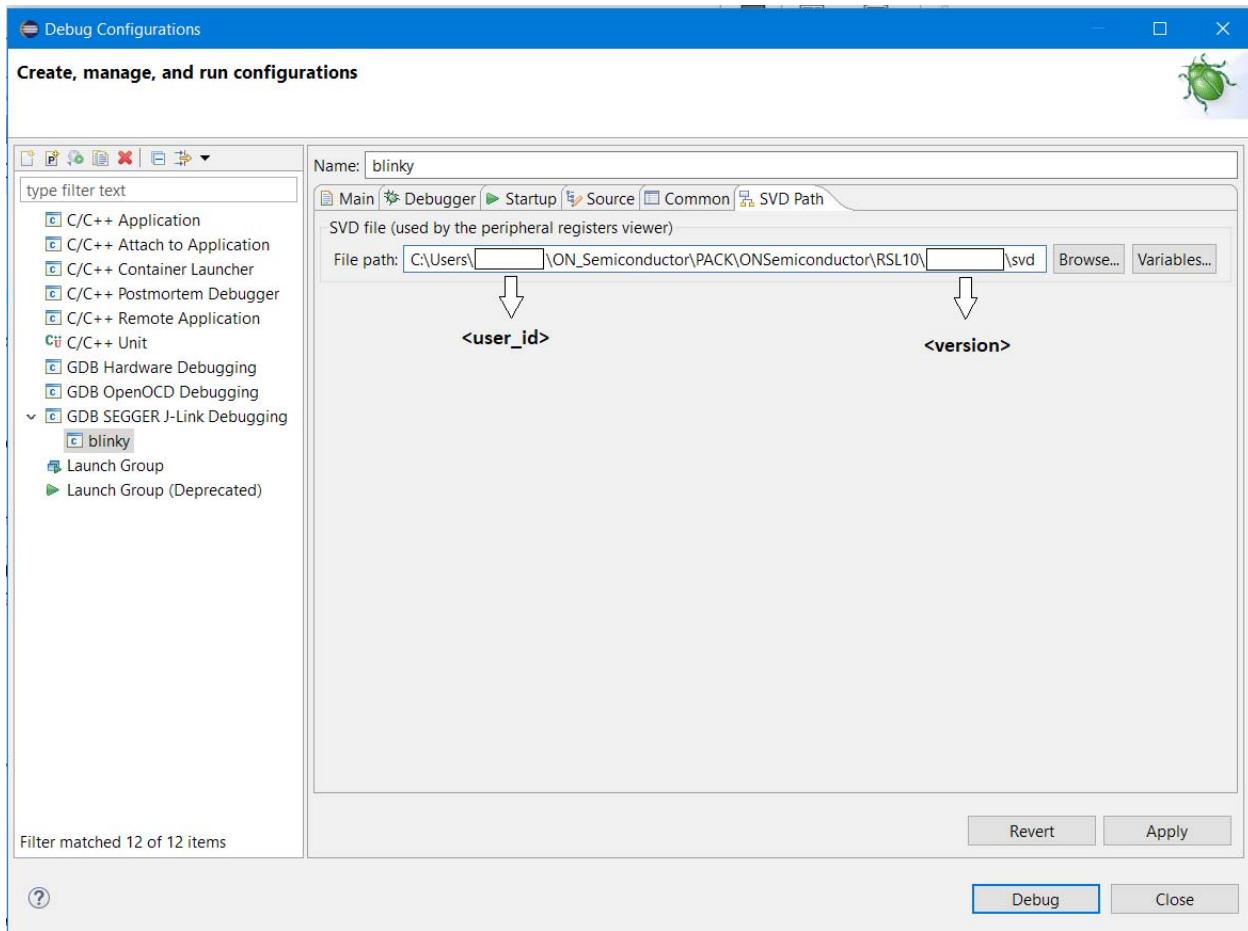


Figure 12. SVD Path Tab Perspective

3. In the **Debug** perspective, when the application runs up to the first breakpoint in *main*, open the Peripherals window view, by navigating to **Window > Show View > Other > Debug > Peripherals** and clicking **Open**. Now you can see all the RSL10 peripherals displayed.
4. In the Peripherals window, select **DIO**. Open the Memory window to monitor the RSL10 peripheral. Read only registers are highlighted in green. You might want to drag your Memory window and place it side-by-side with your source code view (see Figure 13) to prevent the console from switching focus away from the Memory window.
5. To see or change the DIO register status, choose **DIO** and expand the **DIO > DIO_DATA** register in the Memory window.
6. Press F6 to step through the code. You can observe that this register's bit 6 toggles its state when `Sys_GPIO_Toggle(LED_DIO)` is executed (in this case, from 0xF060 to 0xF020). The register turns yellow to indicate that you have activated real-time monitoring for it (see Figure 14 on page 17).

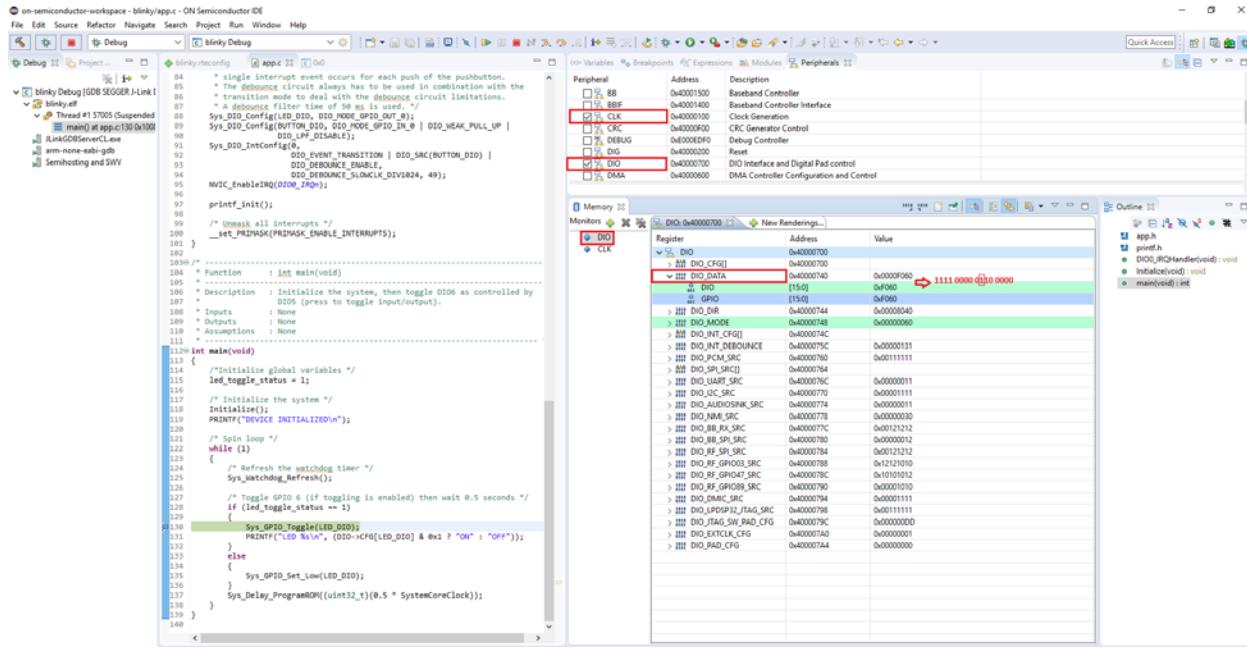


Figure 13. Peripheral Registers View Perspective in Debug Session After Setting SVD Path

- To manually change the register value, click on the **Value** field of the GPIO register to change the (**HIGH**/**LOW**) state of **GPIO6**. Figure 14 shows the view before making the change, and Figure 15 illustrates the view after making the change. You can observe that the LED (DIO6) on your board changes state.

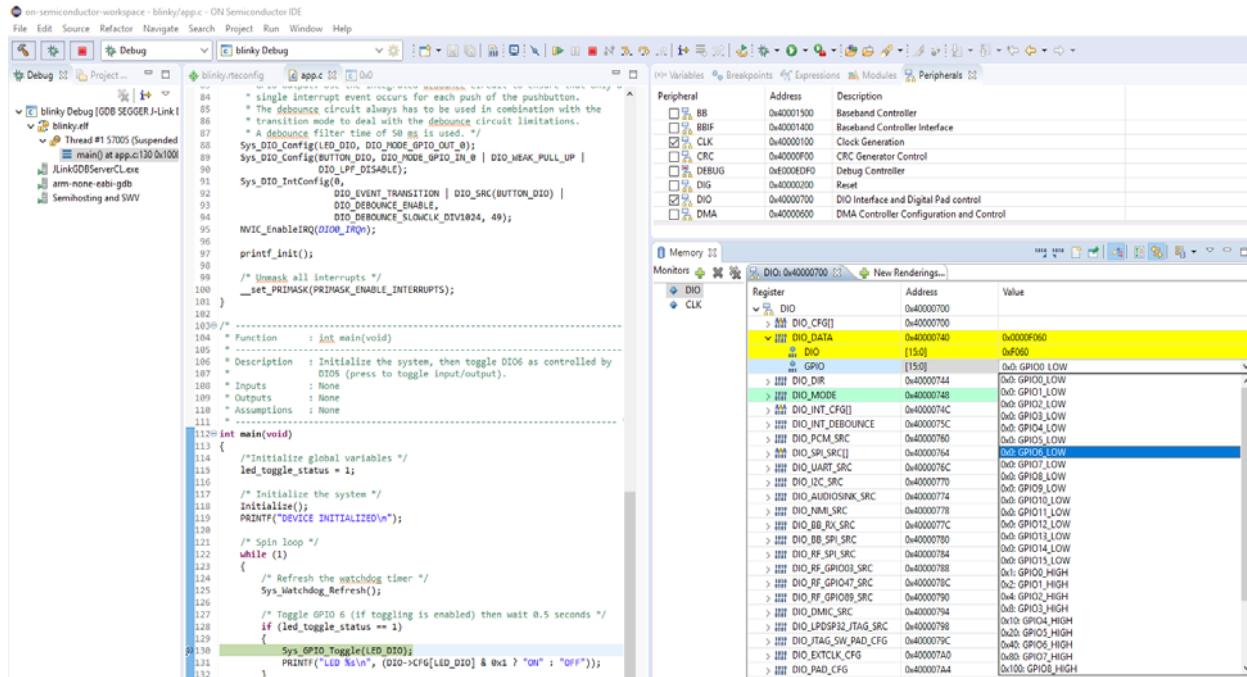


Figure 14. Toggling RSL10 DIO Using the Peripheral Registers View: Before

RSL10 Getting Started Guide

Figure 15. Toggling RSL10 DIO Using the Peripheral Registers View: After

CHAPTER 4

Getting Started with Keil

4.1 PREREQUISITE SOFTWARE

1. Download and install the Keil µVision IDE from the [Keil website](#), using the vendor's instructions.
2. Download the **RSL10 Software Package** from www.onsemi.com/RSL10 and extract the RSL10 CMSIS-Pack (*ONSemiconductor.RSL10.<version>.pack*) to any temporary folder.

4.2 RSL10 CMSIS-PACK INSTALLATION PROCEDURE

To install the RSL10 CMSIS-Pack:

1. Open the Keil µVision IDE and navigate to **Project > Manage > Pack Installer** or click on the icon shown in Figure 16.



Figure 16. Pack Installer Icon

2. Click on **File > Import**, select your pack file *ONSemiconductor.RSL10.<version>.pack*, and click **Open** (see Figure 17). <version> is the RSL10 version, such as 2.2.347.

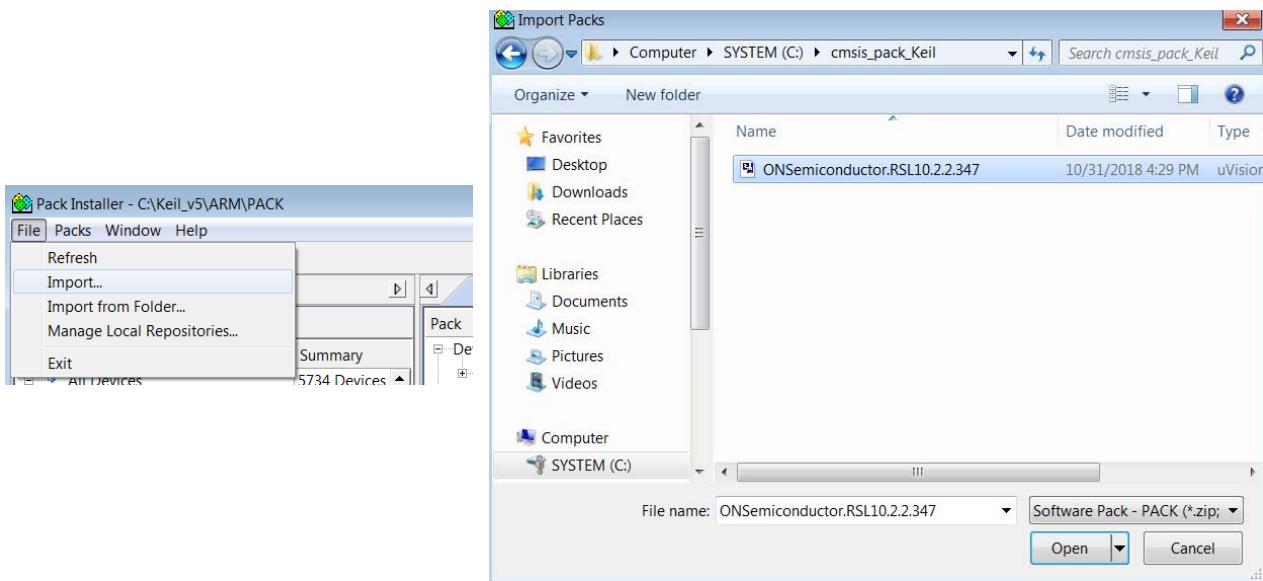


Figure 17. Installing the RSL10 CMSIS-Pack for the Keil µVision IDE

3. The IDE prompts you to read and accept our license agreement, then installs the RSL10 CMSIS-Pack in the `%LOCALAPPDATA%\Arm\Packs` folder.
4. After installation, use **File > Refresh** as shown in Figure 18 to update your pack properties.

RSL10 Getting Started Guide

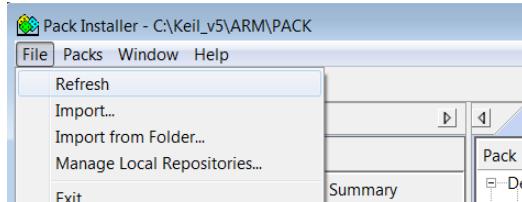


Figure 18. Refresh Pack after installation

- The RSL10 CMSIS-Pack now appears in the list of installed packs. In the **Devices** tab, if you expand **All Devices > ONSemiconductor > RSL10 Series**, you can see RSL10 listed there. You can manage your installed packs in the **Packs** tab. Expanding **ONSemiconductor > RSL10** makes the **Pack Properties** tab display the details of the RSL10 CMSIS-Pack. Figure 19 illustrates what the Pack Installer perspective looks like after installation.

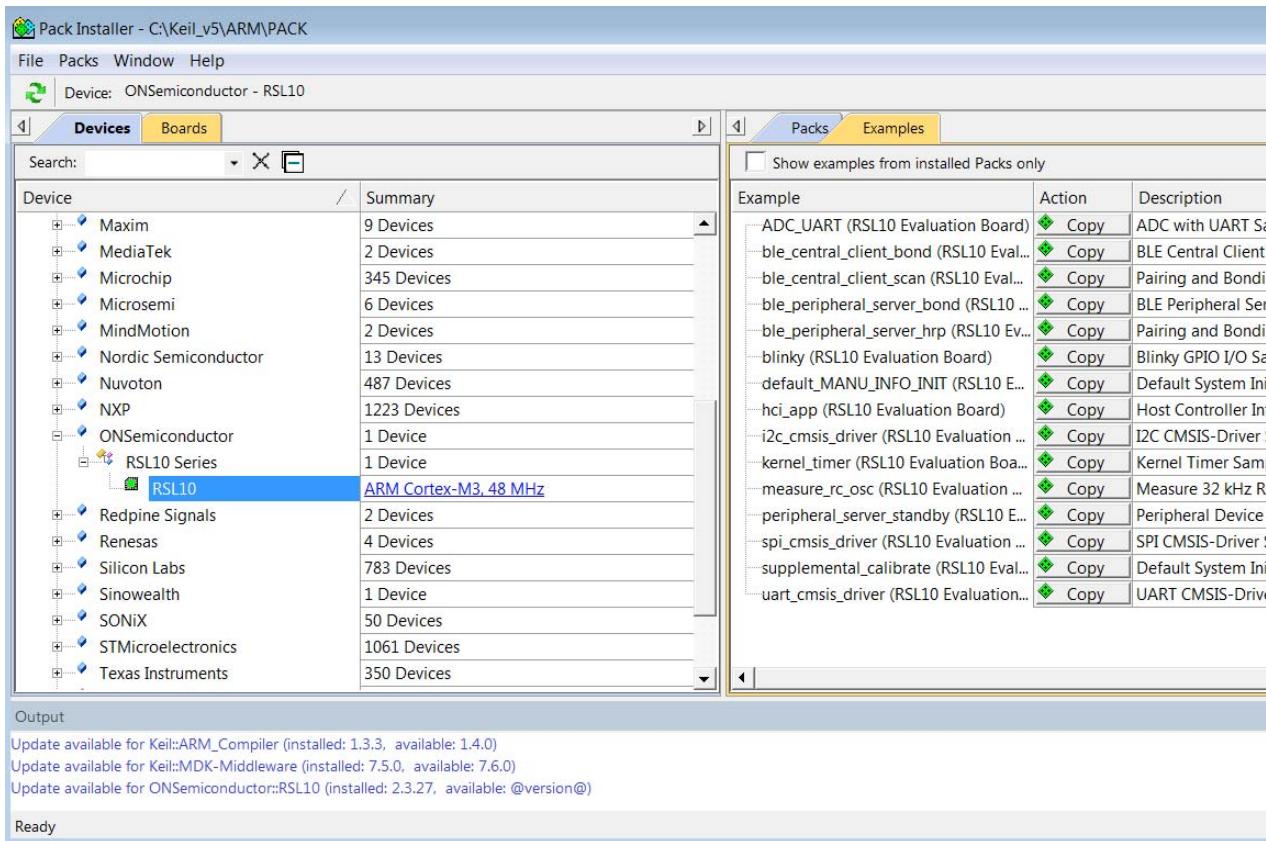


Figure 19. Pack Installer after RSL10 CMSIS-Pack is Installed in the Keil µVision IDE

4.3 BUILDING YOUR FIRST SAMPLE APPLICATION WITH THE KEIL µVISION IDE

This section guides you through importing and building your first sample application, named *blinky*. This application makes the LED (DIO6) blink on the Evaluation and Development Board.

For more information about the sample applications, see the *RSL10 Sample Code User's Guide*.

4.3.1 Import the Sample Code

To import the sample code:

1. In the Pack installer, click on the **Examples** tab to list all the example projects included in the RSL10 CMSIS-Pack.
2. Choose the example project called *blinky*, and click the **Copy** button to import it into your workspace (see Figure 20). Choose a destination folder for a copy of the sample code.

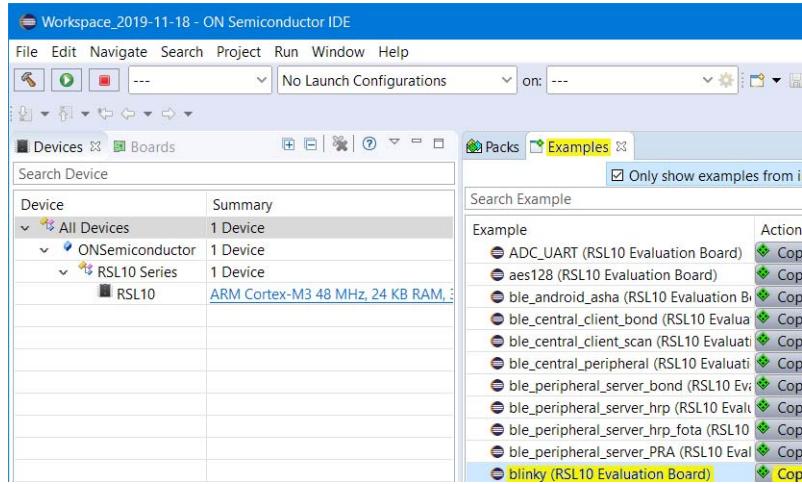


Figure 20. Pack Manager Perspective: Examples Tab

Sample projects are preconfigured with release versions of RSL10 libraries, which are distributed as object files. For Keil, System library (*libsyslib*) and Startup (*libcmsis*) are preconfigured with the source variant, so the source code of those libraries is included directly (see Figure 21).

RSL10 Getting Started Guide

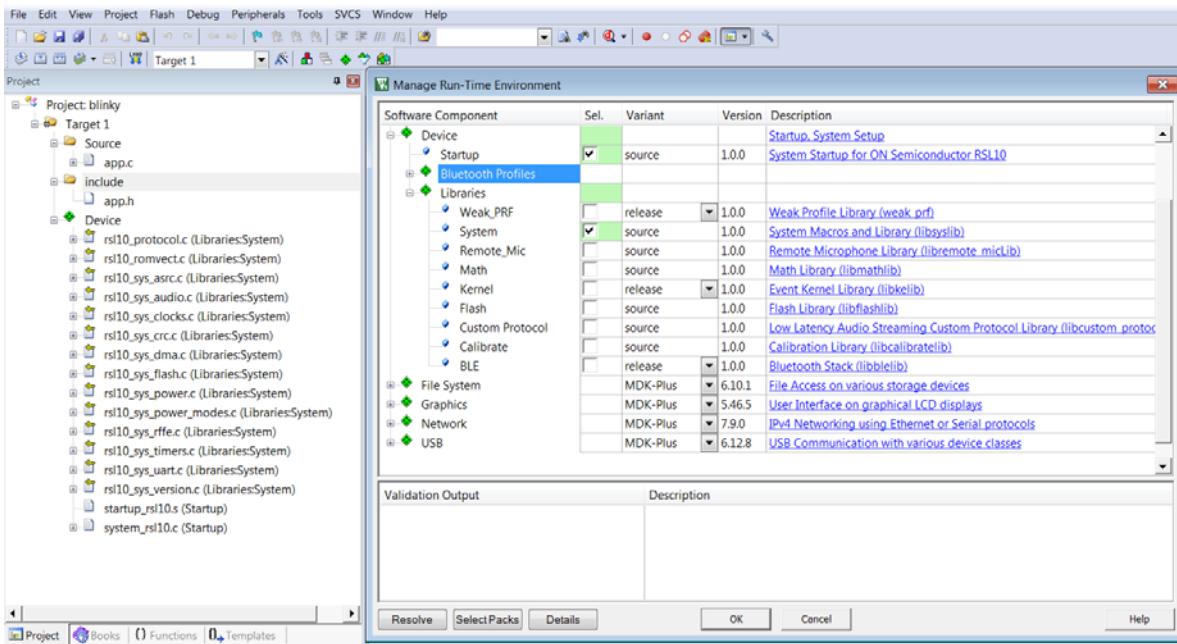


Figure 21. RTE Configuration for the Blinky Example Project in the Keil μ Vision IDE

4.3.2 Build the Sample Code

Build the sample code as follows:

1. Right click on **Target 1** and choose **Rebuild all target files**. Alternatively, you can use the icon shown in Figure 22.

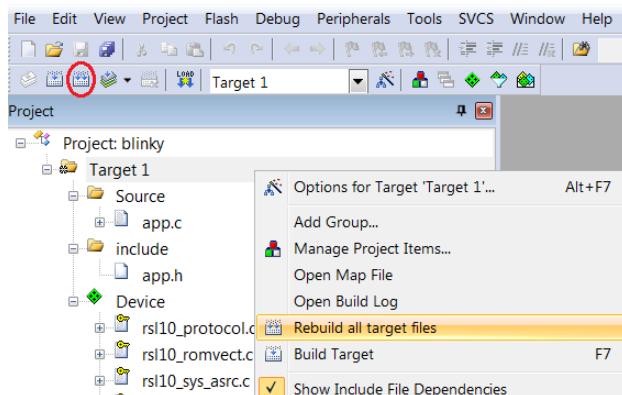


Figure 22. Starting to Build a Project in the Keil μ Vision IDE

2. When the build is running, the output of the build is shown in the Build Output view in the IDE, as illustrated in Figure 23.

```
Build Output
*** Using Compiler 'V5.06 update 6 (build 750)', folder: 'C:\Keil_v5\ARM\ARMCC\Bin'
Build target 'Target 1'
compiling app.c...
linking...
Program Size: Code=1508 RO-data=32 RW-data=4 ZI-data=3076
FromELF: creating hex file...
".\Objects\blinky.axf" - 0 Error(s), 0 Warning(s).
Build Time Elapsed: 00:00:02
```

Figure 23. Example of Build Output

3. The key resulting output in Project Explorer in the IDE includes:
 - *blinky.hex*: HEX file for loading into Flash memory
 - *blinky.axf*: Arm® executable file, run from RAM, used for debugging
 - *blinky.map*: map file of the sections and memory usage

4.3.3 Debugging the Sample Code

4.3.3.1 Preparing J-Link for Debugging

Before debugging with J-Link, go to *C:\Keil_v5\ARM\Segger* and make sure that the folder contains a *JL2CM3.dll* file. As well, make sure that you have installed a compatible version of J-Link.

4.3.3.2 Debugging Applications

The IDE's debug configurations are already set in the CMSIS-Pack. To debug an application:

1. Make sure the Evaluation and Development Board is connected to the PC via a micro USB cable.
2. Select **Debug > Start/Stop Debug Session** or click the icon shown in Figure 24.

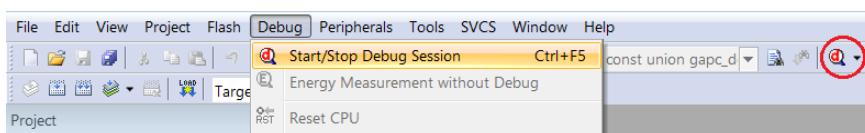


Figure 24. Start/Stop Debug Session Icon

If you are having trouble downloading firmware because an application with Sleep Mode is on the Evaluation and Development Board, see Section 7.4.1, “Downloading Firmware in Sleep Mode” on page 43.

3. The application runs up to the first breakpoint in *main*, as shown in Figure 25. You can press F11 multiple times to step through the code and observe that the LED changes its state when the application executes the function *Sys_GPIO_Toggle(LED_DIO)*.

RSL10 Getting Started Guide

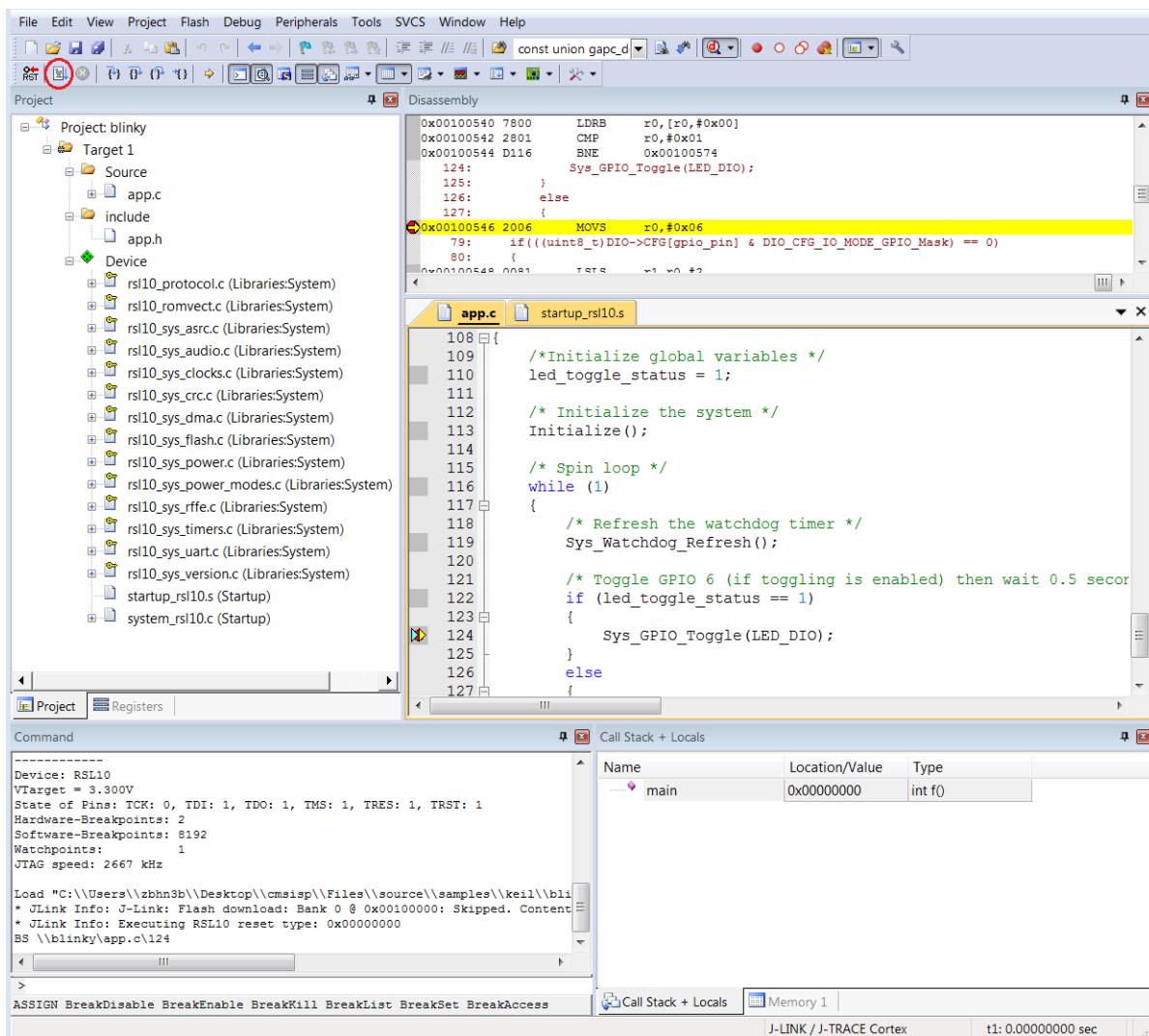


Figure 25. Debug Session in the Keil μ Vision IDE

NOTE: Debug configurations are preconfigured for the sample applications in the CMSIS-Pack. Flash downloading through the Download icon (Figure 26) or F8 is not supported for J-Link.



Figure 26. Download Button Not Supported for J-Link

CHAPTER 5

Getting Started with IAR

5.1 PREREQUISITE SOFTWARE

1. Download and install the IAR Embedded Workbench from the [IAR Website](#), using the vendor's instructions.
2. Download the **RSL10 Software Package** from www.onsemi.com/RSL10 and extract the RSL10 CMSIS-Pack (*ONSemiconductor.RSL10.<version>.pack*) to any temporary folder.

5.2 RSL10 CMSIS-PACK INSTALLATION PROCEDURE

To install the RSL10 CMSIS-Pack:

1. Open the IAR Embedded Workbench and expand **File > New Workspace** to open a new workspace, then go to **File > Save Workspace As** and choose the location for your workspace.
2. Navigate to **Project > CMSIS Pack Manager**, or click on the icon shown in Figure 27.



Figure 27. Pack Installer Icon

3. Click on **CMSIS Manager > Import Existing Packs**, select your pack file *ONSemiconductor.RSL10.<version>.pack*, and click **Open** (see Figure 28). <version> is the RSL10 version, such as 2.3.27.

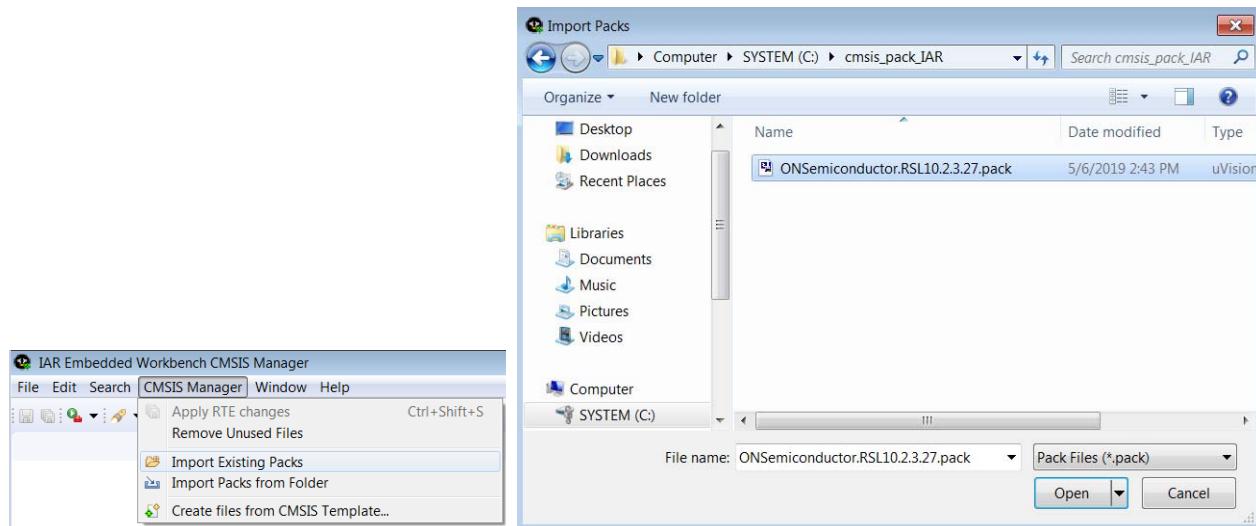


Figure 28. Installing the RSL10 CMSIS-Pack for the IAR Embedded Workbench IDE

4. The IDE prompts you to read and accept the license agreement, then installs the RSL10 CMSIS-Pack in the CMSIS-Pack root folder.
5. After installation, click on the refresh icon with yellow arrows, which shows the text **Reload Packs in the CMSIS Pack root folder** when you hover over it with your cursor, in the Packs tab (as shown in Figure 29), to update your pack properties.

RSL10 Getting Started Guide

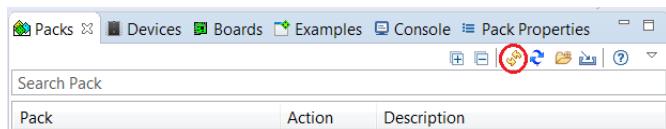


Figure 29. Refresh Pack after installation

6. In the **Devices** tab, expand **All Devices > ONSemiconductor > RSL10 Series**, and select **RSL10** from the list. The RSL10 CMSIS-Pack now appears in the list of installed packs in the **Packs** tab. Expanding **ONSemiconductor.RSL10** makes the **Pack Properties** tab display the details of the RSL10 CMSIS-Pack. Figure 30 on page 26 illustrates what the Pack Manager perspective looks like after installation.

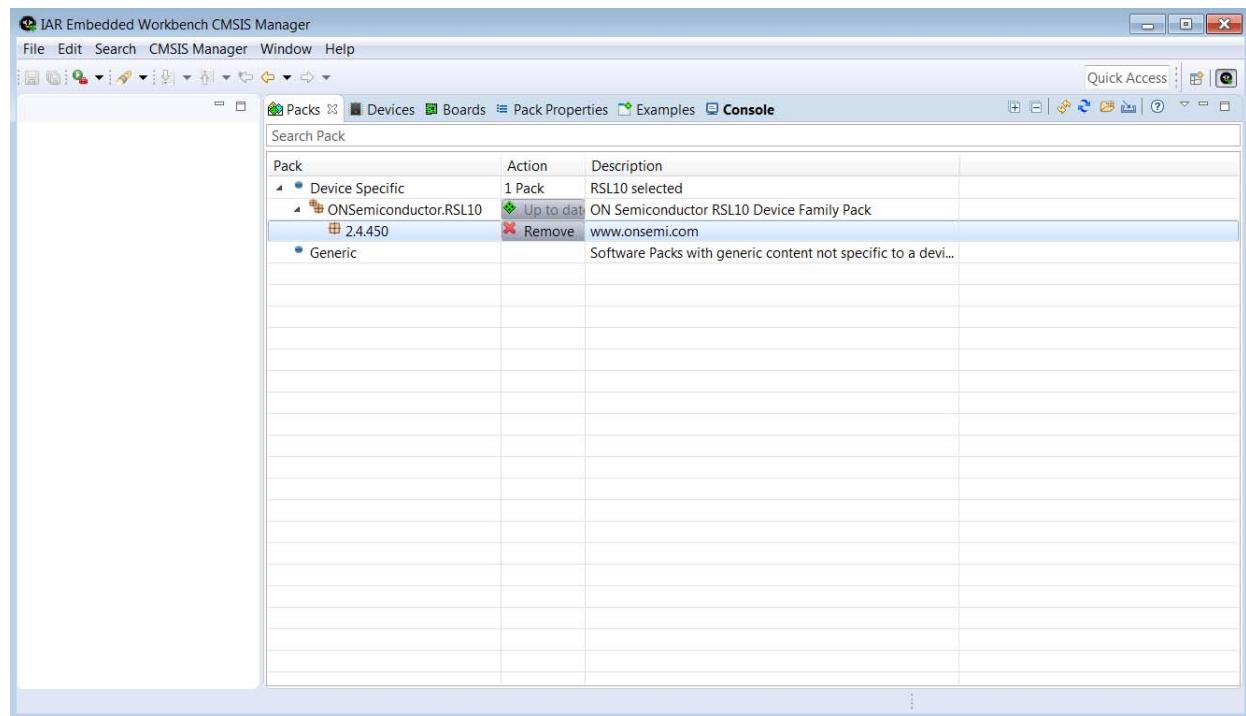


Figure 30. The IAR Embedded Workbench CMSIS Manager after RSL10 CMSIS-Pack is Installed

5.3 BUILDING YOUR FIRST SAMPLE APPLICATION WITH THE IAR EMBEDDED WORKBENCH

This section guides you through importing and building your first sample application, named *blinky*. This application makes the LED (DIO6) blink on the Evaluation and Development Board. The procedure described in this section assumes that you have installed the SDK.

For more information about the sample applications, see the *RSL10 Sample Code User's Guide*.

5.3.1 Import the Sample Code

Import the sample code to your workspace as follows:

1. In the IDE's **CMSIS Manager**, click on the **Examples** tab to list all the example projects included in the RSL10 CMSIS-Pack.

2. Choose the example project called *blinky*, and click the **Copy** button to import it into your workspace (see Figure 31 on page 27). Choose a destination folder for a copy of the sample code.

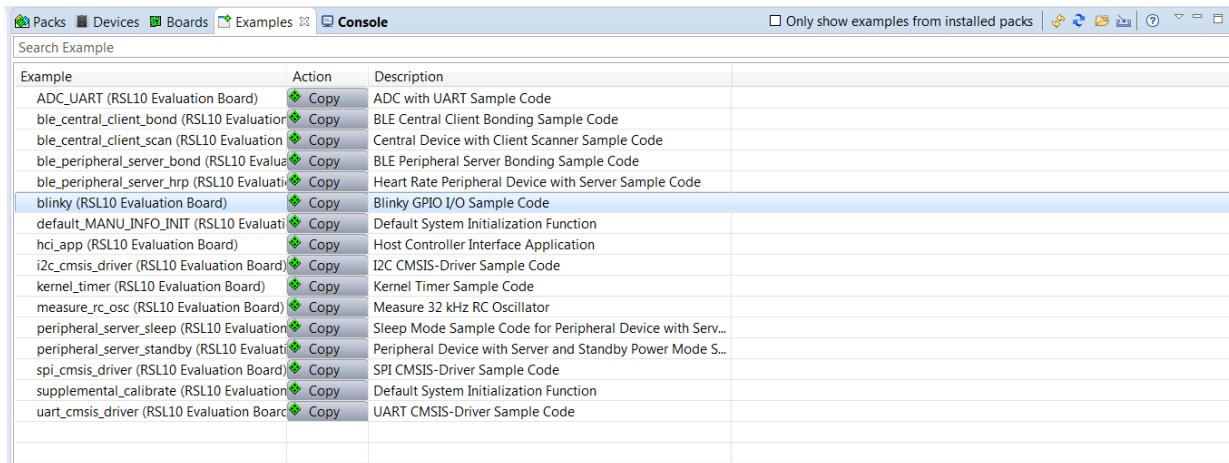


Figure 31. IAR Embedded Workbench CMSIS Manager: Examples Tab

Sample projects are preconfigured with release versions of RSL10 libraries, which are distributed as object files. For the IDE, System library (*libsystlib*) and Startup (*libcmsis*) are preconfigured with the source variant, so the source code of those libraries is included directly in both **CMSIS Manager** and **IAR Embedded Workbench IDE** windows (see Figure 32 on page 27 and Figure 33 on page 28).

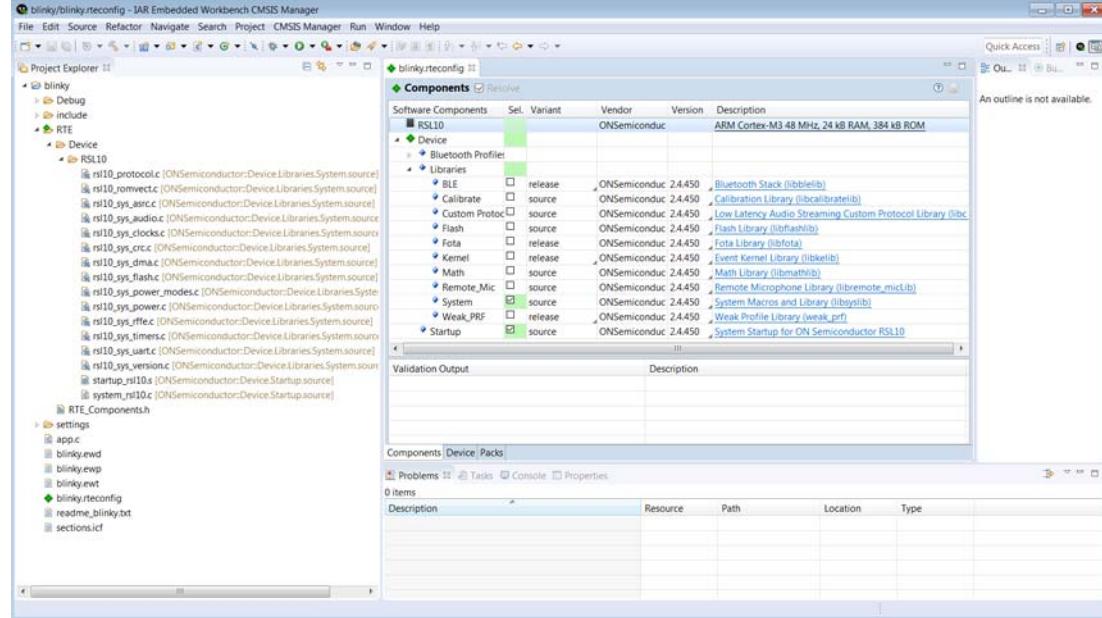


Figure 32. RTE Configuration for the Blinky Example Project in the IAR Embedded Workbench CMSIS Manager window

RSL10 Getting Started Guide

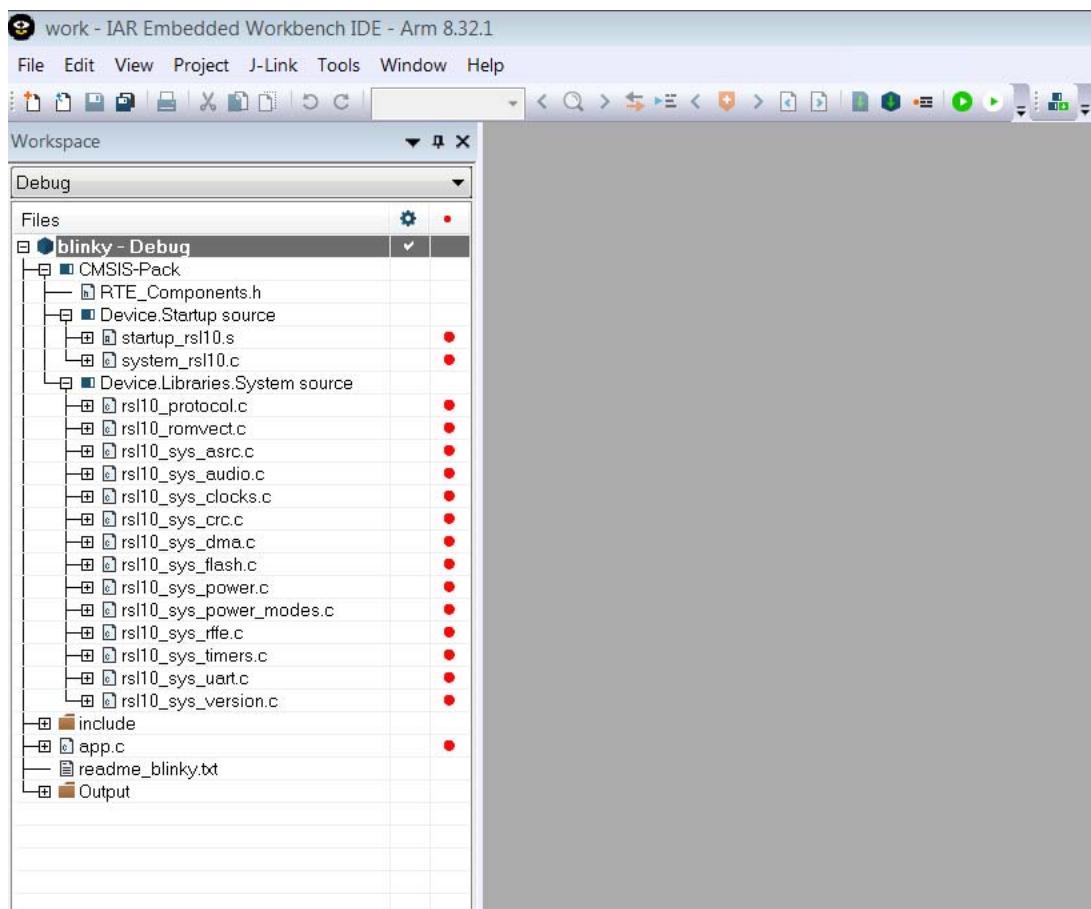


Figure 33. RTE Configuration for the Blinky Example Project in the IAR Embedded Workbench window

5.3.2 Building the Sample Code

To build the sample code:

1. Right click on the folder for blinky and choose **Rebuild All**. Alternatively, you can use the icon shown in Figure 34.

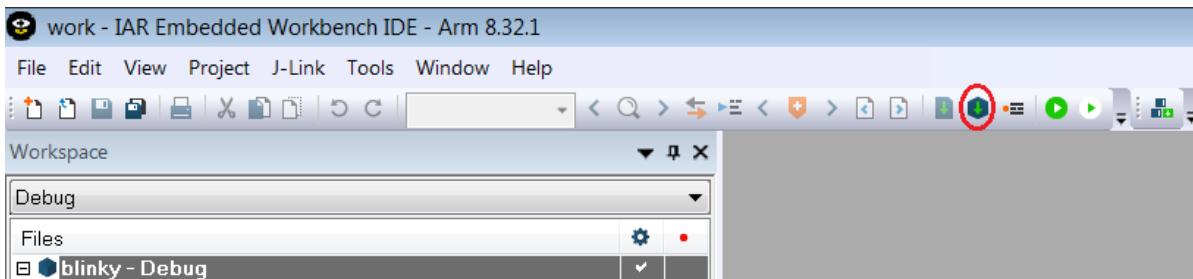


Figure 34. Starting to Build a Project in the IAR Embedded Workbench

2. When the build is running, the output of the build is displayed in the Build Output view in the IDE, as illustrated in Figure 35.

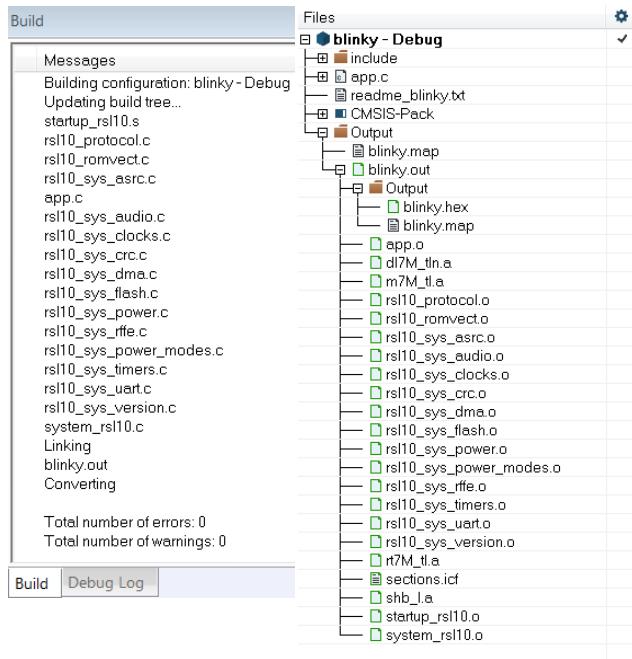


Figure 35. Example of Build Output

3. The key resulting output shown in Project Explorer in the IDE includes:
- *blinky.hex*: HEX file for loading into flash memory
 - *blinky.out*: Arm executable file, used for debugging
 - *blinky.map*: map file of the sections and memory usage

5.3.3 Debugging the Sample Code

5.3.3.1 Debugging Applications

IDE debug configurations are already set in the CMSIS pack. To debug an application:

1. Make sure the Evaluation and Development Board is connected to the PC via a micro USB cable.
2. Select **Project > Download and Debug**, or click the icon shown in Figure 36, then accept the J-Link pop-up dialog in order to use the flash breakpoints (as shown in Figure 37).



Figure 36. Start/Stop Debug Session Icon

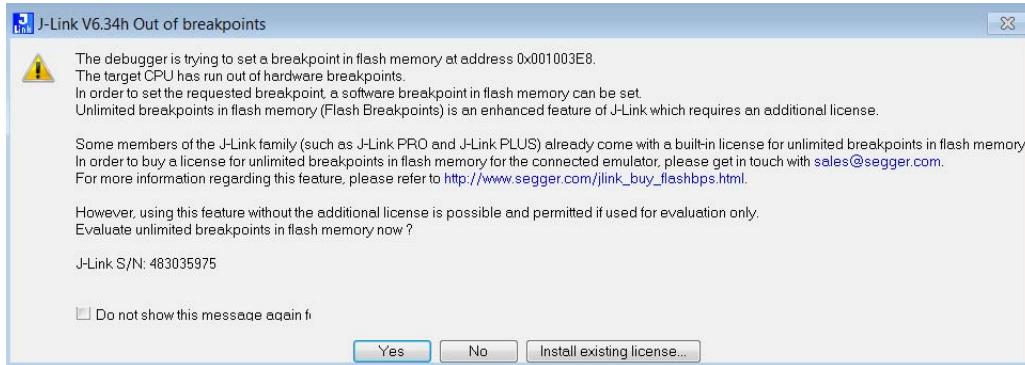


Figure 37. J-link “Out of breakpoints” pop-up dialog

If you are having trouble downloading firmware because an application with Sleep Mode is on the Evaluation and Development Board, see Section 7.4.1, “Downloading Firmware in Sleep Mode” on page 43.

3. The application runs up to the first breakpoint in *main*. You can press F5 or the Run icon (as shown in Figure 38) multiple times to step through the code and observe that the LED changes its state when the application executes the function `Sys_GPIO_Toggle(LED_DIO)`. To stop the debug session, press the Stop icon.

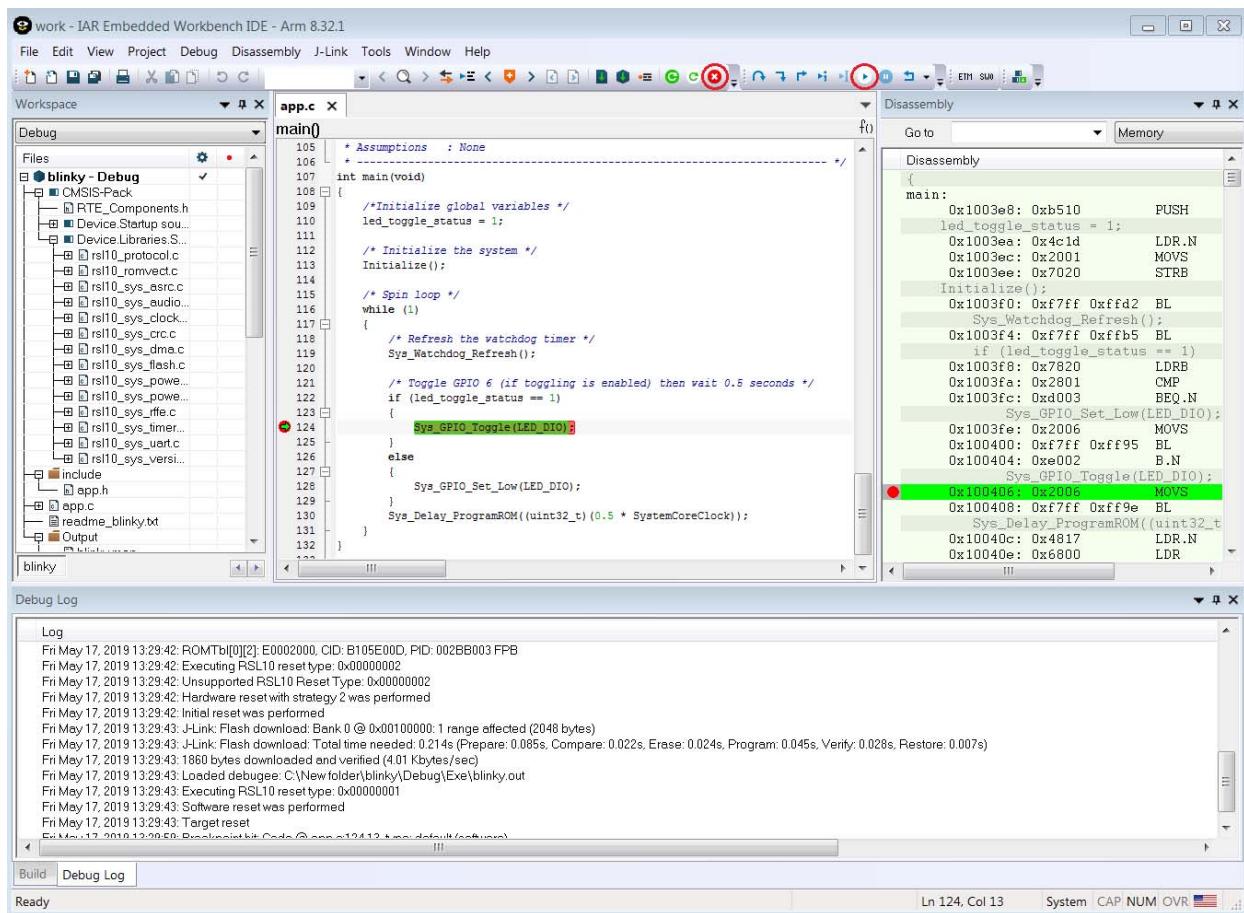


Figure 38. Debug Session in the IAR Embedded Workbench

CHAPTER 6

Resolving External CMSIS-Pack Dependencies

1. EXTERNAL CMSIS-PACK DEPENDENCIES

Some of the RSL10 sample applications depend on software components from external vendors. For example, applications that make use of CMSIS-Drivers or FreeRTOS depend on CMSIS-Packs provided by Arm®. The dependencies are displayed in the RTE Configuration (see Figure 39 for an example).

2. RESOLVING EXTERNAL DEPENDENCIES

The following instructions show how to easily identify and resolve external dependencies in RSL10 sample applications using the CMSIS-Pack manager.

Components		Resolve		
Software Components	Sel.	Variant	Vendor	Version
RSL10			ONSemiconduc	ARM Cortex-M3 48 MHz, 24 KB RAM, 388 KB ROM
↳ CMSIS				
↳ CMSIS Driver				
↳ Device				
↳ RTOS		FreeRTOS	ARM	

Validation Output	Description
✖ ARM::CMSIS.RTOS2.FreeRTOS	Component is missing. Pack is not installed: ARM.CMSIS-FreeRTOS
✖ ARM.FreeRTOS::RTOS.Config.CMSIS RTOS2	Component is missing. Pack is not installed: ARM.CMSIS-FreeRTOS
✖ ARM.FreeRTOS::RTOS.Core.Cortex-M	Component is missing. Pack is not installed: ARM.CMSIS-FreeRTOS
✖ ARM.FreeRTOS::RTOS.Event Groups	Component is missing. Pack is not installed: ARM.CMSIS-FreeRTOS
✖ ARM.FreeRTOS::RTOS.Heap.Heap_4	Component is missing. Pack is not installed: ARM.CMSIS-FreeRTOS
✖ ARM.FreeRTOS::RTOS.Timers	Component is missing. Pack is not installed: ARM.CMSIS-FreeRTOS

Figure 39. RTE Configuration Perspective Before Resolving Pack Dependencies

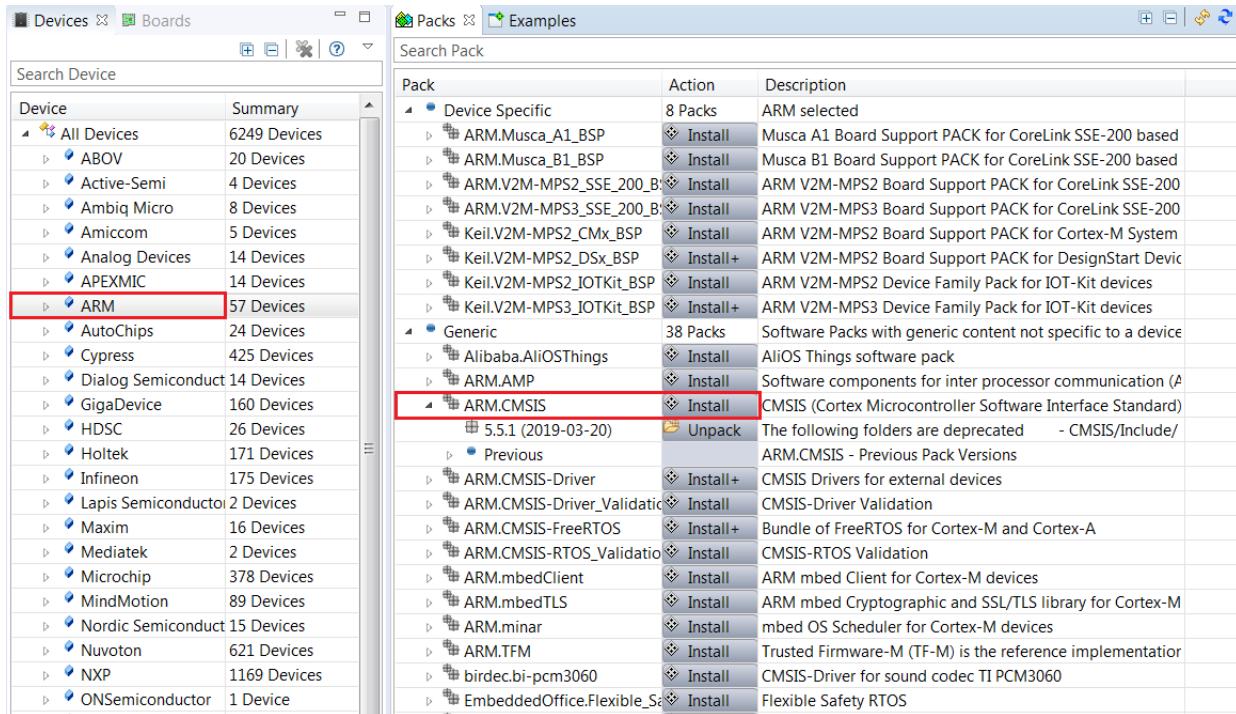
Figure 39, above, shows the RTE Configuration view when Pack dependencies are unresolved. To resolve Pack dependencies, follow these steps:

1. In the CMSIS-Pack Manager perspective, click on the **Check for Updates on Web** button (see Figure 40).

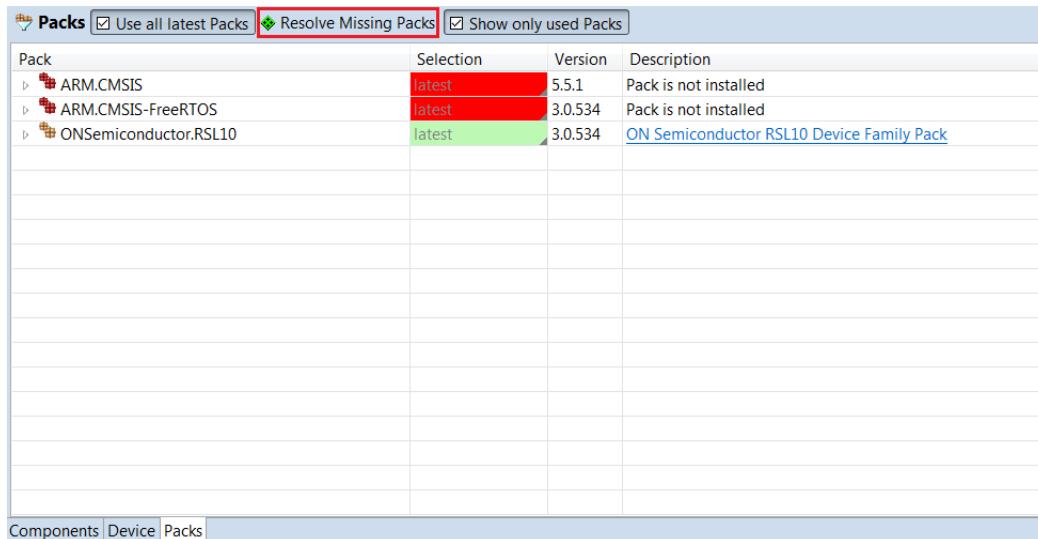


Figure 40. Check for Updates on Web Button

Figure 41, below, shows an example of the Packs tab after checking for updates.

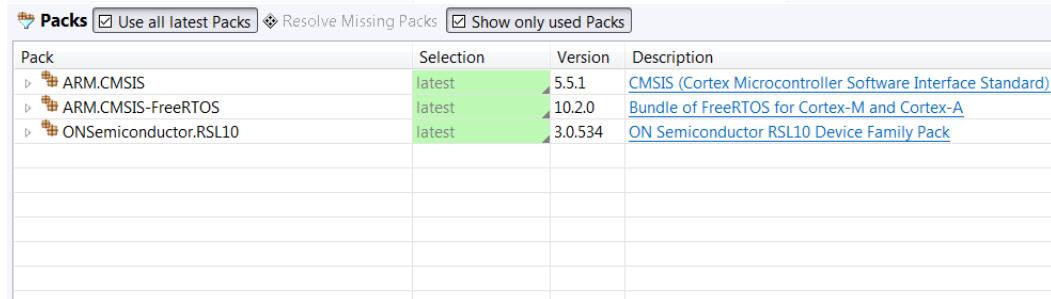
**Figure 41. Installing the Arm CMSIS-Pack**

- To manually install a CMSIS-Pack, select the **Packs** tab and search for the required CMSIS-Pack (in this example, we installed the *ARM.CMSIS* pack); click the **Install** button (shown in Figure 41). Alternatively, follow the next steps to automatically resolve any Pack dependencies that are missing.
- Open the **.rteconfig* file; in the Packs tab, select the **Resolve Missing Packs** button (see Figure 42).

**Figure 42. Resolve Missing Packs Icon**

RSL10 Getting Started Guide

4. The IDE prompts you to read and accept the license agreement, then installs the missing Packs. Figure 43 illustrates the RTE configuration after resolving missing Packs.



The screenshot shows the 'Packs' view in the RTE Configuration Perspective. The interface includes tabs for 'Packs' (selected), 'Use all latest Packs', 'Resolve Missing Packs', and 'Show only used Packs'. A table lists three packs: ARM.CMSIS, ARM.CMSIS-FreeRTOS, and ONSemiconductor.RSL10, each with its selection status, version, and description.

Pack	Selection	Version	Description
ARM.CMSIS	latest	5.5.1	CMSIS (Cortex Microcontroller Software Interface Standard)
ARM.CMSIS-FreeRTOS	latest	10.2.0	Bundle of FreeRTOS for Cortex-M and Cortex-A
ONSemiconductor.RSL10	latest	3.0.534	ON Semiconductor RSL10 Device Family Pack

Figure 43. RTE Configuration Perspective After Resolving Pack Dependencies

CHAPTER 7

Advanced Debugging

7.1 PRINTF DEBUG CAPABILITIES

The `PRINTF()` macro is used to provide `printf()` debug capability in RSL10 applications. The implementation of the `PRINTF()` macro is user selectable to allow for different types of debug interfaces. The functionality is accessed via the tracing API.

The tracing API supports two debug interfaces: UART and RTT. The implementation of the tracing functions can be found in the `app_trace.c` file. The developer can select the debug interface during the compilation process by setting the `RSL10_DEBUG` macro in the `app_trace.h` file. If the macro is set to `DBG_NO`, tracing is disabled. This is the default behavior in all sample applications.

NOTE: The files `app_trace.c` and `app_trace.h` need to be present in your sample application, and initialized using `TRACE_INIT()`, in order to use the `PRINTF()` feature. You can find these two required files in most Bluetooth Low Energy sample applications, such as `ble_peripheral_server_bond`.

To debug time critical applications, we recommend setting the tracing option to `DBG_RTT` option. With SEGGER RTT (Real Time Transfer), you can output information from the target MCU to the RTT Viewer application at a very high speed without compromising the target's real time behavior. More information about SEGGER RTT can be found in JLINK user manual, at www.segger.com.

7.1.1 Adding Printf Debug Capabilities

To add printf debug capabilities over UART, change the define in the `app_trace.h` file to `#define RSL10_DEBUG DBG_UART`, and set the `RSL10_DEBUG` macro to `DBG_UART`. A standard terminal program on a PC can be used to view the debug output.

To add RTT printf debug capabilities, change the define in the `app_trace.h` file to `#define RSL10_DEBUG DBG_RTT` and add the SEGGER RTT files to the application. The Segger RTT Viewer application on a PC can be used to view the debug output.

Samples for RTT are under `C:\Program Files (x86)\SEGGER\JLink_V640b\Samples\RTT`.

More information about the RTT API can be found in the JLINK manual, under `C:\Program Files (x86)\SEGGER\JLink_V640b\Doc\Manuals`.

NOTE: Note that these RTT sample and information files are for SEGGER JLink version 640b.

7.2 DEBUGGING APPLICATIONS THAT DO NOT START 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), read on. For example, you might be debugging an application in RAM, or a flash memory application that has been placed in a different address.

This procedure assumes you have performed the steps in Section 3.4.1, “Debugging with the .elf File” on page 13, and you are using the ON Semiconductor IDE:

1. In your Debug configuration, change to the **Startup** tab
2. Enter the following in the **Run/Restart Commands** field as illustrated in Figure 44:

RSL10 Getting Started Guide

```
set {int} &_VTOR = ISR_Vector_Table  
set $sp = *((int *) &ISR_Vector_Table)
```

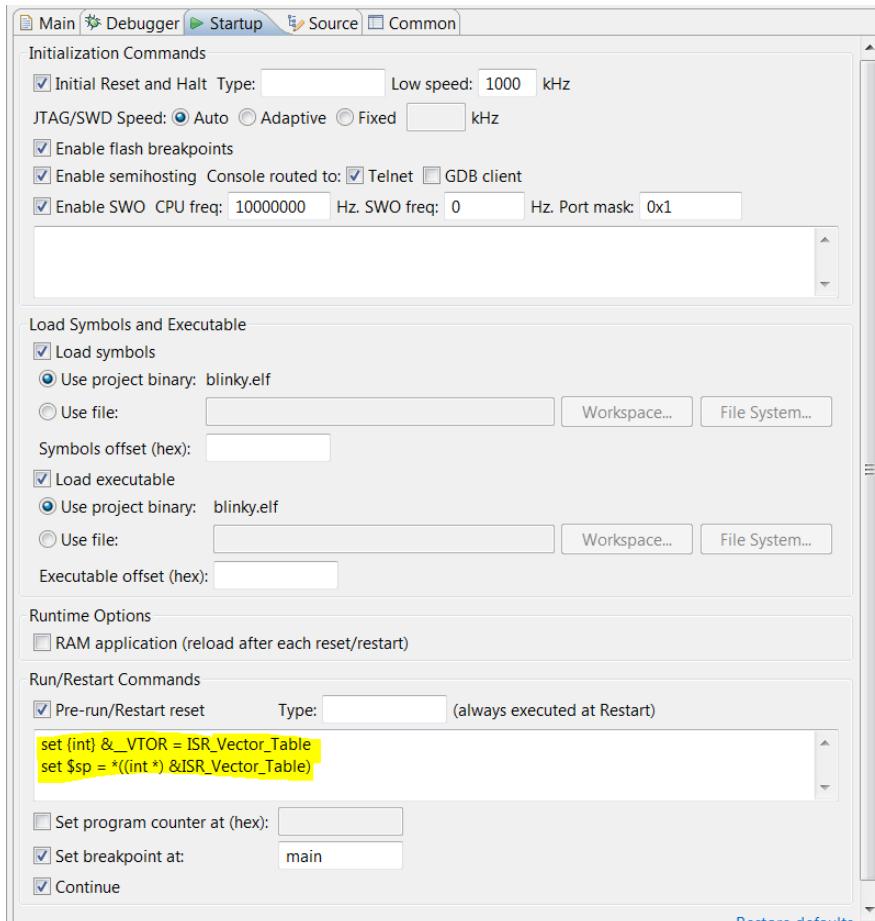


Figure 44. Setting Up a GDB Launch Configuration, Startup Tab

7.3 Arm Cortex-M3 Core Breakpoints

A maximum of two hardware breakpoints can be set at a given time. If you need more than two breakpoints, you can use the Unlimited Flash Breakpoints feature available through J-Link.

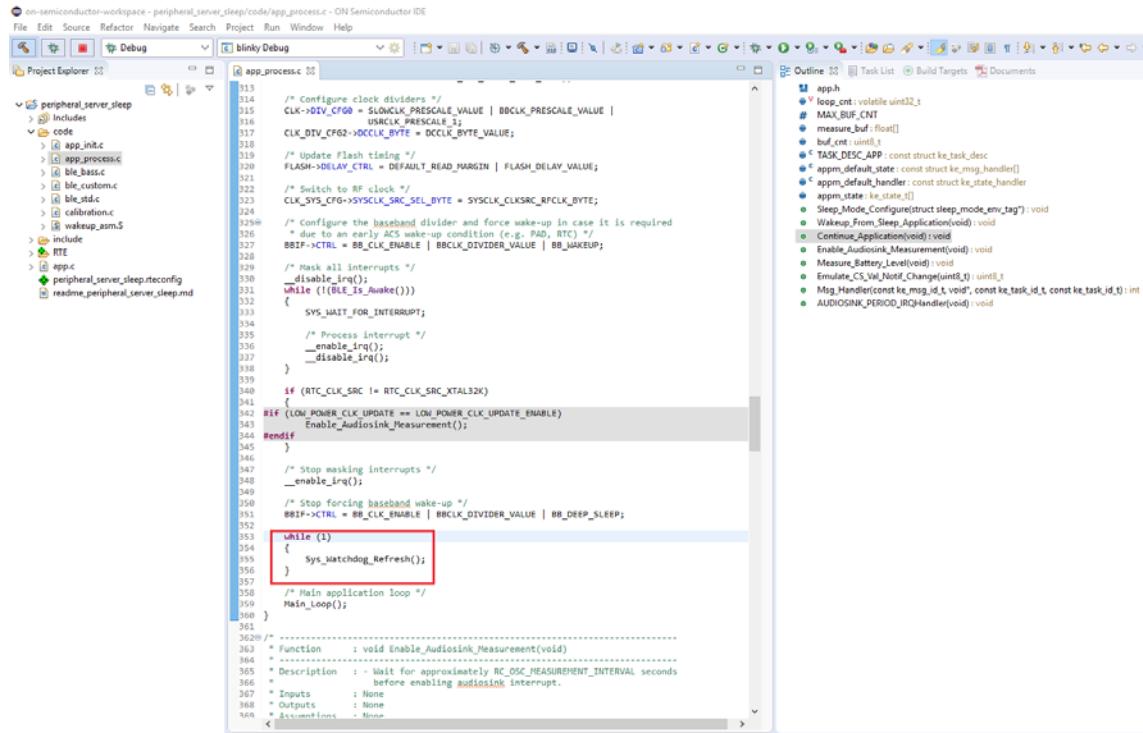
IMPORTANT: You can use hardware breakpoints when using the debugger with the Arm Cortex-M3 core, but software breakpoints cannot be used with the flash overlay. Writing to flash memory does not place breakpoints within the overlay, so any attempt to use software breakpoints would be ineffective.

7.4 DEBUGGING WITH LOW POWER SLEEP MODE

Debugging applications that use sleep mode is a challenging task because the hardware debug logic and system clocks are powered down when the device goes to sleep. Therefore, the debug session cannot be kept alive between sleep cycles.

Besides using GPIOs, UART, and other peripherals as tools to help debug your application, you can reattach the debugger after the device wakes up from sleep. To do so, you need to make sure that the device stays awake, and start a new debug session to connect to the running target, making sure a reset is not performed. The following instructions show an example of how to perform this on the *peripheral_server_sleep* sample application in the ON Semiconductor IDE, but you can also adapt it for other applications that use sleep mode, and for other IDEs.

1. Copy the *peripheral_server_sleep* application into your workspace and navigate to the *app_process.c* source file under the *code* folder.
2. Modify the function void *Continue_Application(void)* by adding a *while* loop before the *Main_Loop()* ; call, to make sure that the device stays awake in the infinite loop after waking up (see Figure 45). Save and compile your application.



```

314     /* Configure clock dividers */
315     CLK_DV_CFG0 = SLOWCLK_PRESCALE_VALUE | BBCLK_PRESCALE_VALUE |
316                 USCLK_PRESCALE_1;
317     CLK_DV_CFG2 = OCLK_BYTEx = OCLK_BYTE_VALUE;
318
319     /* Update flash timing */
320     FLASH->DELAY_CTRL = DEFAULT_READ_MARGIN | FLASH_DELAY_VALUE;
321
322     /* Switch to RF clock */
323     CLK_SYS_CFG = SYSCLK_SRC_SEL_BYTEx = SYSCLK_CLKSRC_RFCLK_BYTEx;
324
325     /* Configure the baseband divider and force wake-up in case it is required
326      * due to an early ACS wake-up condition (e.g. PAD, RTC) */
327     BBIF->CTRL = BB_CLK_ENABLE | BBCLK_DIVIDER_VALUE | BB_WAKEUP;
328
329     /* Mask all interrupts */
330     _disable_irq();
331     while (!BLE_Is_Awake())
332     {
333         SYS_WAIT_FOR_INTERRUPT;
334
335         /* Process interrupt */
336         _enable_irq();
337         _disable_irq();
338     }
339
340     #if (RTC_CLK_SRC != RTC_CLK_SRC_XTAL32K)
341
342     #if (LOW_POWER_CLK_UPDATE == LOW_POWER_CLK_UPDATE_ENABLE)
343         Enable_Audiosink_Measurement();
344     #endif
345
346
347     /* Stop masking interrupts */
348     _enable_irq();
349
350     /* Stop forcing baseband wake-up */
351     BBIF->CTRL = BB_CLK_ENABLE | BBCLK_DIVIDER_VALUE | BB_DEEP_SLEEP;
352
353     while (1)
354     {
355         Sys_Watchdog_Refresh();
356
357         /* Main application loop */
358         Main_Loop();
359
360     }
361
362     /*
363      * Function      : void Enable_Audiosink_Measurement(void)
364      * -----
365      * Description   : - wait for approximately RC_OSC_MEASUREMENT_INTERVAL seconds
366      *                  before enabling Audiosink interrupt.
367      * Inputs       : None
368      * Outputs      : None
369      * Assumptions  : None
370     */
371
372     /*
373      * -----
374      * Function      : void Main_Loop(void)
375      * -----
376      * Description   : Main application loop
377      * Inputs       : None
378      * Outputs      : None
379      * Assumptions  : None
380     */
381
382     /*
383      * -----
384      * Function      : void Sys_Watchdog_Refresh(void)
385      * -----
386      * Description   : Refresh the watchdog timer
387      * Inputs       : None
388      * Outputs      : None
389      * Assumptions  : None
390     */
391
392     /*
393      * -----
394      * Function      : void Main_Loop(void)
395      * -----
396      * Description   : Main application loop
397      * Inputs       : None
398      * Outputs      : None
399      * Assumptions  : None
400     */
401
402     /*
403      * -----
404      * Function      : void Sys_Watchdog_Refresh(void)
405      * -----
406      * Description   : Refresh the watchdog timer
407      * Inputs       : None
408      * Outputs      : None
409      * Assumptions  : None
410     */
411
412     /*
413      * -----
414      * Function      : void Main_Loop(void)
415      * -----
416      * Description   : Main application loop
417      * Inputs       : None
418      * Outputs      : None
419      * Assumptions  : None
420     */
421
422     /*
423      * -----
424      * Function      : void Sys_Watchdog_Refresh(void)
425      * -----
426      * Description   : Refresh the watchdog timer
427      * Inputs       : None
428      * Outputs      : None
429      * Assumptions  : None
430     */
431
432     /*
433      * -----
434      * Function      : void Main_Loop(void)
435      * -----
436      * Description   : Main application loop
437      * Inputs       : None
438      * Outputs      : None
439      * Assumptions  : None
440     */
441
442     /*
443      * -----
444      * Function      : void Sys_Watchdog_Refresh(void)
445      * -----
446      * Description   : Refresh the watchdog timer
447      * Inputs       : None
448      * Outputs      : None
449      * Assumptions  : None
450     */
451
452     /*
453      * -----
454      * Function      : void Main_Loop(void)
455      * -----
456      * Description   : Main application loop
457      * Inputs       : None
458      * Outputs      : None
459      * Assumptions  : None
460     */
461
462     /*
463      * -----
464      * Function      : void Sys_Watchdog_Refresh(void)
465      * -----
466      * Description   : Refresh the watchdog timer
467      * Inputs       : None
468      * Outputs      : None
469      * Assumptions  : None
470     */
471
472     /*
473      * -----
474      * Function      : void Main_Loop(void)
475      * -----
476      * Description   : Main application loop
477      * Inputs       : None
478      * Outputs      : None
479      * Assumptions  : None
480     */
481
482     /*
483      * -----
484      * Function      : void Sys_Watchdog_Refresh(void)
485      * -----
486      * Description   : Refresh the watchdog timer
487      * Inputs       : None
488      * Outputs      : None
489      * Assumptions  : None
490     */
491
492     /*
493      * -----
494      * Function      : void Main_Loop(void)
495      * -----
496      * Description   : Main application loop
497      * Inputs       : None
498      * Outputs      : None
499      * Assumptions  : None
500     */
501
502     /*
503      * -----
504      * Function      : void Sys_Watchdog_Refresh(void)
505      * -----
506      * Description   : Refresh the watchdog timer
507      * Inputs       : None
508      * Outputs      : None
509      * Assumptions  : None
510     */
511
512     /*
513      * -----
514      * Function      : void Main_Loop(void)
515      * -----
516      * Description   : Main application loop
517      * Inputs       : None
518      * Outputs      : None
519      * Assumptions  : None
520     */
521
522     /*
523      * -----
524      * Function      : void Sys_Watchdog_Refresh(void)
525      * -----
526      * Description   : Refresh the watchdog timer
527      * Inputs       : None
528      * Outputs      : None
529      * Assumptions  : None
530     */
531
532     /*
533      * -----
534      * Function      : void Main_Loop(void)
535      * -----
536      * Description   : Main application loop
537      * Inputs       : None
538      * Outputs      : None
539      * Assumptions  : None
540     */
541
542     /*
543      * -----
544      * Function      : void Sys_Watchdog_Refresh(void)
545      * -----
546      * Description   : Refresh the watchdog timer
547      * Inputs       : None
548      * Outputs      : None
549      * Assumptions  : None
550     */
551
552     /*
553      * -----
554      * Function      : void Main_Loop(void)
555      * -----
556      * Description   : Main application loop
557      * Inputs       : None
558      * Outputs      : None
559      * Assumptions  : None
560     */
561
562     /*
563      * -----
564      * Function      : void Sys_Watchdog_Refresh(void)
565      * -----
566      * Description   : Refresh the watchdog timer
567      * Inputs       : None
568      * Outputs      : None
569      * Assumptions  : None
570     */
571
572     /*
573      * -----
574      * Function      : void Main_Loop(void)
575      * -----
576      * Description   : Main application loop
577      * Inputs       : None
578      * Outputs      : None
579      * Assumptions  : None
580     */
581
582     /*
583      * -----
584      * Function      : void Sys_Watchdog_Refresh(void)
585      * -----
586      * Description   : Refresh the watchdog timer
587      * Inputs       : None
588      * Outputs      : None
589      * Assumptions  : None
590     */
591
592     /*
593      * -----
594      * Function      : void Main_Loop(void)
595      * -----
596      * Description   : Main application loop
597      * Inputs       : None
598      * Outputs      : None
599      * Assumptions  : None
600     */
601
602     /*
603      * -----
604      * Function      : void Sys_Watchdog_Refresh(void)
605      * -----
606      * Description   : Refresh the watchdog timer
607      * Inputs       : None
608      * Outputs      : None
609      * Assumptions  : None
610     */
611
612     /*
613      * -----
614      * Function      : void Main_Loop(void)
615      * -----
616      * Description   : Main application loop
617      * Inputs       : None
618      * Outputs      : None
619      * Assumptions  : None
620     */
621
622     /*
623      * -----
624      * Function      : void Sys_Watchdog_Refresh(void)
625      * -----
626      * Description   : Refresh the watchdog timer
627      * Inputs       : None
628      * Outputs      : None
629      * Assumptions  : None
630     */
631
632     /*
633      * -----
634      * Function      : void Main_Loop(void)
635      * -----
636      * Description   : Main application loop
637      * Inputs       : None
638      * Outputs      : None
639      * Assumptions  : None
640     */
641
642     /*
643      * -----
644      * Function      : void Sys_Watchdog_Refresh(void)
645      * -----
646      * Description   : Refresh the watchdog timer
647      * Inputs       : None
648      * Outputs      : None
649      * Assumptions  : None
650     */
651
652     /*
653      * -----
654      * Function      : void Main_Loop(void)
655      * -----
656      * Description   : Main application loop
657      * Inputs       : None
658      * Outputs      : None
659      * Assumptions  : None
660     */
661
662     /*
663      * -----
664      * Function      : void Sys_Watchdog_Refresh(void)
665      * -----
666      * Description   : Refresh the watchdog timer
667      * Inputs       : None
668      * Outputs      : None
669      * Assumptions  : None
670     */
671
672     /*
673      * -----
674      * Function      : void Main_Loop(void)
675      * -----
676      * Description   : Main application loop
677      * Inputs       : None
678      * Outputs      : None
679      * Assumptions  : None
680     */
681
682     /*
683      * -----
684      * Function      : void Sys_Watchdog_Refresh(void)
685      * -----
686      * Description   : Refresh the watchdog timer
687      * Inputs       : None
688      * Outputs      : None
689      * Assumptions  : None
690     */
691
692     /*
693      * -----
694      * Function      : void Main_Loop(void)
695      * -----
696      * Description   : Main application loop
697      * Inputs       : None
698      * Outputs      : None
699      * Assumptions  : None
700     */
701
702     /*
703      * -----
704      * Function      : void Sys_Watchdog_Refresh(void)
705      * -----
706      * Description   : Refresh the watchdog timer
707      * Inputs       : None
708      * Outputs      : None
709      * Assumptions  : None
710     */
711
712     /*
713      * -----
714      * Function      : void Main_Loop(void)
715      * -----
716      * Description   : Main application loop
717      * Inputs       : None
718      * Outputs      : None
719      * Assumptions  : None
720     */
721
722     /*
723      * -----
724      * Function      : void Sys_Watchdog_Refresh(void)
725      * -----
726      * Description   : Refresh the watchdog timer
727      * Inputs       : None
728      * Outputs      : None
729      * Assumptions  : None
730     */
731
732     /*
733      * -----
734      * Function      : void Main_Loop(void)
735      * -----
736      * Description   : Main application loop
737      * Inputs       : None
738      * Outputs      : None
739      * Assumptions  : None
740     */
741
742     /*
743      * -----
744      * Function      : void Sys_Watchdog_Refresh(void)
745      * -----
746      * Description   : Refresh the watchdog timer
747      * Inputs       : None
748      * Outputs      : None
749      * Assumptions  : None
750     */
751
752     /*
753      * -----
754      * Function      : void Main_Loop(void)
755      * -----
756      * Description   : Main application loop
757      * Inputs       : None
758      * Outputs      : None
759      * Assumptions  : None
760     */
761
762     /*
763      * -----
764      * Function      : void Sys_Watchdog_Refresh(void)
765      * -----
766      * Description   : Refresh the watchdog timer
767      * Inputs       : None
768      * Outputs      : None
769      * Assumptions  : None
770     */
771
772     /*
773      * -----
774      * Function      : void Main_Loop(void)
775      * -----
776      * Description   : Main application loop
777      * Inputs       : None
778      * Outputs      : None
779      * Assumptions  : None
780     */
781
782     /*
783      * -----
784      * Function      : void Sys_Watchdog_Refresh(void)
785      * -----
786      * Description   : Refresh the watchdog timer
787      * Inputs       : None
788      * Outputs      : None
789      * Assumptions  : None
790     */
791
792     /*
793      * -----
794      * Function      : void Main_Loop(void)
795      * -----
796      * Description   : Main application loop
797      * Inputs       : None
798      * Outputs      : None
799      * Assumptions  : None
800     */
801
802     /*
803      * -----
804      * Function      : void Sys_Watchdog_Refresh(void)
805      * -----
806      * Description   : Refresh the watchdog timer
807      * Inputs       : None
808      * Outputs      : None
809      * Assumptions  : None
810     */
811
812     /*
813      * -----
814      * Function      : void Main_Loop(void)
815      * -----
816      * Description   : Main application loop
817      * Inputs       : None
818      * Outputs      : None
819      * Assumptions  : None
820     */
821
822     /*
823      * -----
824      * Function      : void Sys_Watchdog_Refresh(void)
825      * -----
826      * Description   : Refresh the watchdog timer
827      * Inputs       : None
828      * Outputs      : None
829      * Assumptions  : None
830     */
831
832     /*
833      * -----
834      * Function      : void Main_Loop(void)
835      * -----
836      * Description   : Main application loop
837      * Inputs       : None
838      * Outputs      : None
839      * Assumptions  : None
840     */
841
842     /*
843      * -----
844      * Function      : void Sys_Watchdog_Refresh(void)
845      * -----
846      * Description   : Refresh the watchdog timer
847      * Inputs       : None
848      * Outputs      : None
849      * Assumptions  : None
850     */
851
852     /*
853      * -----
854      * Function      : void Main_Loop(void)
855      * -----
856      * Description   : Main application loop
857      * Inputs       : None
858      * Outputs      : None
859      * Assumptions  : None
860     */
861
862     /*
863      * -----
864      * Function      : void Sys_Watchdog_Refresh(void)
865      * -----
866      * Description   : Refresh the watchdog timer
867      * Inputs       : None
868      * Outputs      : None
869      * Assumptions  : None
870     */
871
872     /*
873      * -----
874      * Function      : void Main_Loop(void)
875      * -----
876      * Description   : Main application loop
877      * Inputs       : None
878      * Outputs      : None
879      * Assumptions  : None
880     */
881
882     /*
883      * -----
884      * Function      : void Sys_Watchdog_Refresh(void)
885      * -----
886      * Description   : Refresh the watchdog timer
887      * Inputs       : None
888      * Outputs      : None
889      * Assumptions  : None
890     */
891
892     /*
893      * -----
894      * Function      : void Main_Loop(void)
895      * -----
896      * Description   : Main application loop
897      * Inputs       : None
898      * Outputs      : None
899      * Assumptions  : None
900     */
901
902     /*
903      * -----
904      * Function      : void Sys_Watchdog_Refresh(void)
905      * -----
906      * Description   : Refresh the watchdog timer
907      * Inputs       : None
908      * Outputs      : None
909      * Assumptions  : None
910     */
911
912     /*
913      * -----
914      * Function      : void Main_Loop(void)
915      * -----
916      * Description   : Main application loop
917      * Inputs       : None
918      * Outputs      : None
919      * Assumptions  : None
920     */
921
922     /*
923      * -----
924      * Function      : void Sys_Watchdog_Refresh(void)
925      * -----
926      * Description   : Refresh the watchdog timer
927      * Inputs       : None
928      * Outputs      : None
929      * Assumptions  : None
930     */
931
932     /*
933      * -----
934      * Function      : void Main_Loop(void)
935      * -----
936      * Description   : Main application loop
937      * Inputs       : None
938      * Outputs      : None
939      * Assumptions  : None
940     */
941
942     /*
943      * -----
944      * Function      : void Sys_Watchdog_Refresh(void)
945      * -----
946      * Description   : Refresh the watchdog timer
947      * Inputs       : None
948      * Outputs      : None
949      * Assumptions  : None
950     */
951
952     /*
953      * -----
954      * Function      : void Main_Loop(void)
955      * -----
956      * Description   : Main application loop
957      * Inputs       : None
958      * Outputs      : None
959      * Assumptions  : None
960     */
961
962     /*
963      * -----
964      * Function      : void Sys_Watchdog_Refresh(void)
965      * -----
966      * Description   : Refresh the watchdog timer
967      * Inputs       : None
968      * Outputs      : None
969      * Assumptions  : None
970     */
971
972     /*
973      * -----
974      * Function      : void Main_Loop(void)
975      * -----
976      * Description   : Main application loop
977      * Inputs       : None
978      * Outputs      : None
979      * Assumptions  : None
980     */
981
982     /*
983      * -----
984      * Function      : void Sys_Watchdog_Refresh(void)
985      * -----
986      * Description   : Refresh the watchdog timer
987      * Inputs       : None
988      * Outputs      : None
989      * Assumptions  : None
990     */
991
992     /*
993      * -----
994      * Function      : void Main_Loop(void)
995      * -----
996      * Description   : Main application loop
997      * Inputs       : None
998      * Outputs      : None
999      * Assumptions  : None
1000    */
1001
1002     /*
1003      * -----
1004      * Function      : void Sys_Watchdog_Refresh(void)
1005      * -----
1006      * Description   : Refresh the watchdog timer
1007      * Inputs       : None
1008      * Outputs      : None
1009      * Assumptions  : None
1010     */
1011
1012     /*
1013      * -----
1014      * Function      : void Main_Loop(void)
1015      * -----
1016      * Description   : Main application loop
1017      * Inputs       : None
1018      * Outputs      : None
1019      * Assumptions  : None
1020     */
1021
1022     /*
1023      * -----
1024      * Function      : void Sys_Watchdog_Refresh(void)
1025      * -----
1026      * Description   : Refresh the watchdog timer
1027      * Inputs       : None
1028      * Outputs      : None
1029      * Assumptions  : None
1030     */
1031
1032     /*
1033      * -----
1034      * Function      : void Main_Loop(void)
1035      * -----
1036      * Description   : Main application loop
1037      * Inputs       : None
1038      * Outputs      : None
1039      * Assumptions  : None
1040     */
1041
1042     /*
1043      * -----
1044      * Function      : void Sys_Watchdog_Refresh(void)
1045      * -----
1046      * Description   : Refresh the watchdog timer
1047      * Inputs       : None
1048      * Outputs      : None
1049      * Assumptions  : None
1050     */
1051
1052     /*
1053      * -----
1054      * Function      : void Main_Loop(void)
1055      * -----
1056      * Description   : Main application loop
1057      * Inputs       : None
1058      * Outputs      : None
1059      * Assumptions  : None
1060     */
1061
1062     /*
1063      * -----
1064      * Function      : void Sys_Watchdog_Refresh(void)
1065      * -----
1066      * Description   : Refresh the watchdog timer
1067      * Inputs       : None
1068      * Outputs      : None
1069      * Assumptions  : None
1070     */
1071
1072     /*
1073      * -----
1074      * Function      : void Main_Loop(void)
1075      * -----
1076      * Description   : Main application loop
1077      * Inputs       : None
1078      * Outputs      : None
1079      * Assumptions  : None
1080     */
1081
1082     /*
1083      * -----
1084      * Function      : void Sys_Watchdog_Refresh(void)
1085      * -----
1086      * Description   : Refresh the watchdog timer
1087      * Inputs       : None
1088      * Outputs      : None
1089      * Assumptions  : None
1090     */
1091
1092     /*
1093      * -----
1094      * Function      : void Main_Loop(void)
1095      * -----
1096      * Description   : Main application loop
1097      * Inputs       : None
1098      * Outputs      : None
1099      * Assumptions  : None
1100     */
1101
1102     /*
1103      * -----
1104      * Function      : void Sys_Watchdog_Refresh(void)
1105      * -----
1106      * Description   : Refresh the watchdog timer
1107      * Inputs       : None
1108      * Outputs      : None
1109      * Assumptions  : None
1110     */
1111
1112     /*
1113      * -----
1114      * Function      : void Main_Loop(void)
1115      * -----
1116      * Description   : Main application loop
1117      * Inputs       : None
1118      * Outputs      : None
1119      * Assumptions  : None
1120     */
1121
1122     /*
1123      * -----
1124      * Function      : void Sys_Watchdog_Refresh(void)
1125      * -----
1126      * Description   : Refresh the watchdog timer
1127      * Inputs       : None
1128      * Outputs      : None
1129      * Assumptions  : None
1130     */
1131
1132     /*
1133      * -----
1134      * Function      : void Main_Loop(void)
1135      * -----
1136      * Description   : Main application loop
1137      * Inputs       : None
1138      * Outputs      : None
1139      * Assumptions  : None
1140     */
1141
1142     /*
1143      * -----
1144      * Function      : void Sys_Watchdog_Refresh(void)
1145      * -----
1146      * Description   : Refresh the watchdog timer
1147      * Inputs       : None
1148      * Outputs      : None
1149      * Assumptions  : None
1150     */
1151
1152     /*
1153      * -----
1154      * Function      : void Main_Loop(void)
1155      * -----
1156      * Description   : Main application loop
1157      * Inputs       : None
1158      * Outputs      : None
1159      * Assumptions  : None
1160     */
1161
1162     /*
1163      * -----
1164      * Function      : void Sys_Watchdog_Refresh(void)
1165      * -----
1166      * Description   : Refresh the watchdog timer
1167      * Inputs       : None
1168      * Outputs      : None
1169      * Assumptions  : None
1170     */
1171
1172     /*
1173      * -----
1174      * Function      : void Main_Loop(void)
1175      * -----
1176      * Description   : Main application loop
1177      * Inputs       : None
1178      * Outputs      : None
1179      * Assumptions  : None
1180     */
1181
1182     /*
1183      * -----
1184      * Function      : void Sys_Watchdog_Refresh(void)
1185      * -----
1186      * Description   : Refresh the watchdog timer
1187      * Inputs       : None
1188      * Outputs      : None
1189      * Assumptions  : None
1190     */
1191
1192     /*
1193      * -----
1194      * Function      : void Main_Loop(void)
1195      * -----
1196      * Description   : Main application loop
1197      * Inputs       : None
1198      * Outputs      : None
1199      * Assumptions  : None
1200     */
1201
1202     /*
1203      * -----
1204      * Function      : void Sys_Watchdog_Refresh(void)
1205      * -----
1206      * Description   : Refresh the watchdog timer
1207      * Inputs       : None
1208      * Outputs      : None
1209      * Assumptions  : None
1210     */
1211
1212     /*
1213      * -----
1214      * Function      : void Main_Loop(void)
1215      * -----
1216      * Description   : Main application loop
1217      * Inputs       : None
1218      * Outputs      : None
1219      * Assumptions  : None
1220     */
1221
1222     /*
1223      * -----
1224      * Function      : void Sys_Watchdog_Refresh(void)
1225      * -----
1226      * Description   : Refresh the watchdog timer
1227      * Inputs       : None
1228      * Outputs      : None
1229      * Assumptions  : None
1230     */
1231
1232     /*
1233      * -----
1234      * Function      : void Main_Loop(void)
1235      * -----
1236      * Description   : Main application loop
1237      * Inputs       : None
1238      * Outputs      : None
1239      * Assumptions  : None
1240     */
1241
1242     /*
1243      * -----
1244      * Function      : void Sys_Watchdog_Refresh(void)
1245      * -----
1246      * Description   : Refresh the watchdog timer
1247      * Inputs       : None
1248      * Outputs      : None
1249      * Assumptions  : None
1250     */
1251
1252     /*
1253      * -----
1254      * Function      : void Main_Loop(void)
1255      * -----
1256      * Description   : Main application loop
1257      * Inputs       : None
1258      * Outputs      : None
1259      * Assumptions  : None
1260     */
1261
1262     /*
1263      * -----
1264      * Function      : void Sys_Watchdog_Refresh(void)
1265      * -----
1266      * Description   : Refresh the watchdog timer
1267      * Inputs       : None
1268      * Outputs      : None
1269      * Assumptions  : None
1270     */
1271
1272     /*
1273      * -----
1274      * Function      : void Main_Loop(void)
1275      * -----
1276      * Description   : Main application loop
1277      * Inputs       : None
1278      * Outputs      : None
1279      * Assumptions  : None
1280     */
1281
1282     /*
1283      * -----
1284      * Function      : void Sys_Watchdog_Refresh(void)
1285      * -----
1286      * Description   : Refresh the watchdog timer
1287      * Inputs       : None
1288      * Outputs      : None
1289      * Assumptions  : None
1290     */
1291
1292     /*
1293      * -----
1294      * Function      : void Main_Loop(void)
1295      * -----
1296      * Description   : Main application loop
1297      * Inputs       : None
1298      * Outputs      : None
1299      * Assumptions  : None
1300     */
1301
1302     /*
1303      * -----
1304      * Function      : void Sys_Watchdog_Refresh(void)
1305      * -----
1306      * Description   : Refresh the watchdog timer
1307      * Inputs       : None
1308      * Outputs      : None
1309      * Assumptions  : None
1310     */
1311
1312     /*
1313      * -----
1314      * Function      : void Main_Loop(void)
1315      * -----
1316      * Description   : Main application loop
1317      * Inputs       : None
1318      * Outputs      : None
1319      * Assumptions  : None
1320     */
1321
1322     /*
1323      * -----
1324      * Function      : void Sys_Watchdog_Refresh(void)
1325      * -----
1326      * Description   : Refresh the watchdog timer
1327      * Inputs       : None
1328      * Outputs      : None
1329      * Assumptions  : None
1330     */
1331
1332     /*
1333      * -----
1334      * Function      : void Main_Loop(void)
1335      * -----
1336      * Description   : Main application loop
1337      * Inputs       : None
1338      * Outputs      : None
1339      * Assumptions  : None
1340     */
1341
1342     /*
1343      * -----
1344      * Function      : void Sys_Watchdog_Refresh(void)
1345      * -----
1346      * Description   : Refresh the watchdog timer
1347      * Inputs       : None
1348      * Outputs      : None
1349      * Assumptions  : None
1350     */
1351
1352     /*
1353      * -----
1354      * Function      : void Main_Loop(void)
1355      * -----
1356      * Description   : Main application loop
1357      * Inputs       : None
1358      * Outputs      : None
1359      * Assumptions  : None
1360     */
1361
1362     /*
1363      * -----
1364      * Function      : void Sys_Watchdog_Refresh(void)
1365      * -----
1366      * Description   : Refresh the watchdog timer
1367      * Inputs       : None
1368      * Outputs      : None
1369      * Assumptions  : None
1370     */
1371
1372     /*
1373      * -----
1374      * Function      : void Main_Loop(void)
1375      * -----
1376      * Description   : Main application loop
1377      * Inputs       : None
1378      * Outputs      : None
1379      * Assumptions  : None
1380     */
1381
1382     /*
1383      * -----
1384      * Function      : void Sys_Watchdog_Refresh(void)
1385      * -----
1386      * Description   : Refresh the watchdog timer
1387      * Inputs       : None
1388      * Outputs      : None
1389      * Assumptions  : None
1390     */
1391
1392     /*
1393      * -----
1394      * Function      : void Main_Loop(void)
1395      * -----
1396      * Description   : Main application loop
1397      * Inputs       : None
1398      * Outputs      : None
1399      * Assumptions  : None
1400     */
1401
1402     /*
1403      * -----
1404      * Function      : void Sys_Watchdog_Refresh(void)
1405      * -----
1406      * Description   : Refresh the watchdog timer
1407      * Inputs       : None
1408      * Outputs      : None
1409      * Assumptions  : None
1410     */
1411
1412     /*
1413      * -----
1414      * Function      : void Main_Loop(void)
1415      * -----
1416      * Description   : Main application loop
1417      * Inputs       : None
1418      * Outputs      : None
1419      * Assumptions  : None
1420     */
1421
1422     /*
1423      * -----
1424      * Function      : void Sys_Watchdog_Refresh(void)
1425      * -----
1426      * Description   : Refresh the watchdog timer
1427      * Inputs       : None
1428      * Outputs      : None
1429      * Assumptions  : None
1430     */
1431
1432     /*
1433      * -----
1434      * Function      : void Main_Loop(void)
1435      * -----
1436      * Description   : Main application loop
1437      * Inputs       : None
1438      * Outputs      : None
1439      * Assumptions  : None
1440     */
1441
1442     /*
1443      * -----
1444      * Function      : void Sys_Watchdog_Refresh(void)
1445      * -----
1446      * Description   : Refresh the watchdog timer
1447      * Inputs       : None
1448      * Outputs      : None
1449      * Assumptions  : None
1450     */
1451
1452     /*
1453      * -----
1454      * Function      : void Main_Loop(void)
1455      * -----
1456      * Description   : Main application loop
1457      * Inputs       : None
1458      * Outputs      : None
1459      * Assumptions  : None
1460     */
1461
1462     /*
1463      * -----
1464      * Function      : void Sys_Watchdog_Refresh(void)
1465      * -----
1466      * Description   : Refresh the watchdog timer
1467      * Inputs       : None
1468      * Outputs      : None
1469      * Assumptions  : None
1470     */
1471
1472     /*
1473      * -----
1474      * Function      : void Main_Loop(void)
1475      * -----
1476      * Description   : Main application loop
1477      * Inputs       : None
1478      * Outputs      : None
1479      * Assumptions  : None
1480     */
1481
1482     /*
1483      * -----
1484      * Function      : void Sys_Watchdog_Refresh(void)
1485      * -----
1486      * Description   : Refresh the watchdog timer
1487      * Inputs       : None
1488      * Outputs      : None
1489      * Assumptions  : None
1490     */
1491
1492     /*
1493      * -----
1494      * Function      : void Main_Loop(void)
1495      * -----
1496      * Description   : Main application loop
1497      * Inputs       : None
1498      * Outputs      : None
1499      * Assumptions  : None
1500     */
1501
1502     /*
1503      * -----
1504      * Function      : void Sys_Watchdog_Refresh(void)
1505      * -----
1506      * Description   : Refresh the watchdog timer
1507      * Inputs       : None
1508      * Outputs      : None
1509      * Assumptions  : None
1510     */
1511
1512     /*
1513      * -----
1514      * Function      : void Main_Loop(void)
1515      * -----
1516      * Description   : Main application loop
1517      * Inputs
```

RSL10 Getting Started Guide

the peripherals. If this does not work, see Section 7.4.1, “Downloading Firmware in Sleep Mode” on page 43.

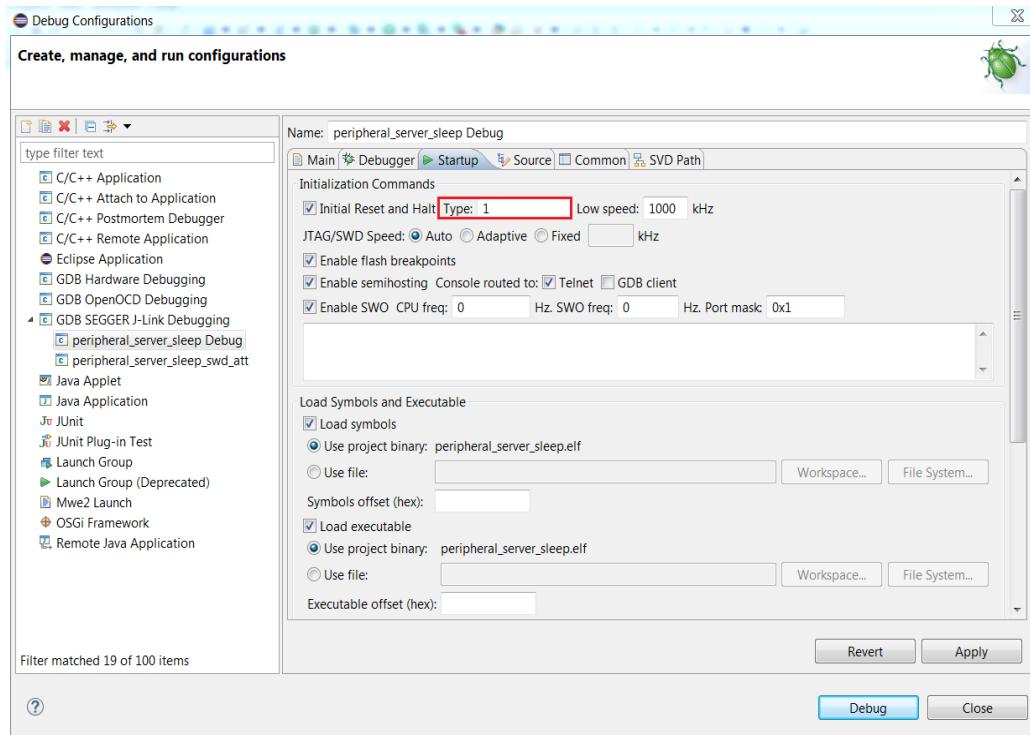


Figure 46. Setting Reset Type in the Debug Configuration Session

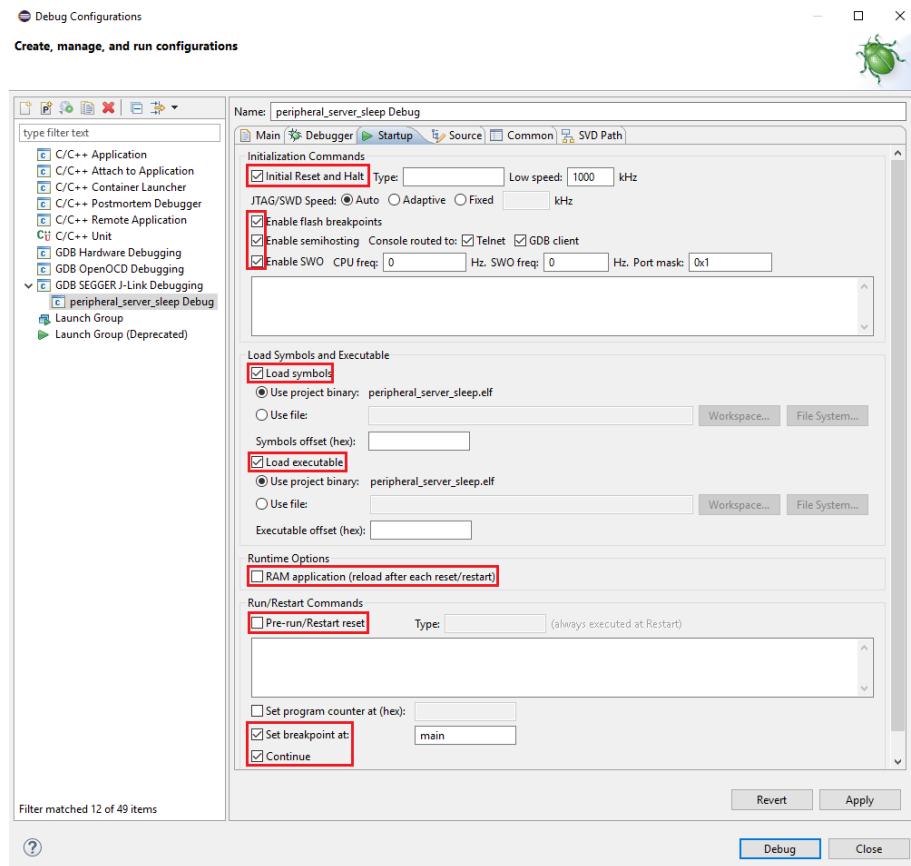


Figure 47. Startup Tab: Debug Session that Initiates Restart

- b. Debug session that connects to the running target:
 - i. Create another new debug configuration under the **GDB SEGGER** heading, with new configuration details in the right panel.
 - ii. Adjust the displayed values for your configuration then click on **Apply** (see Figure 48, and Figure 49 on page 41).

RSL10 Getting Started Guide

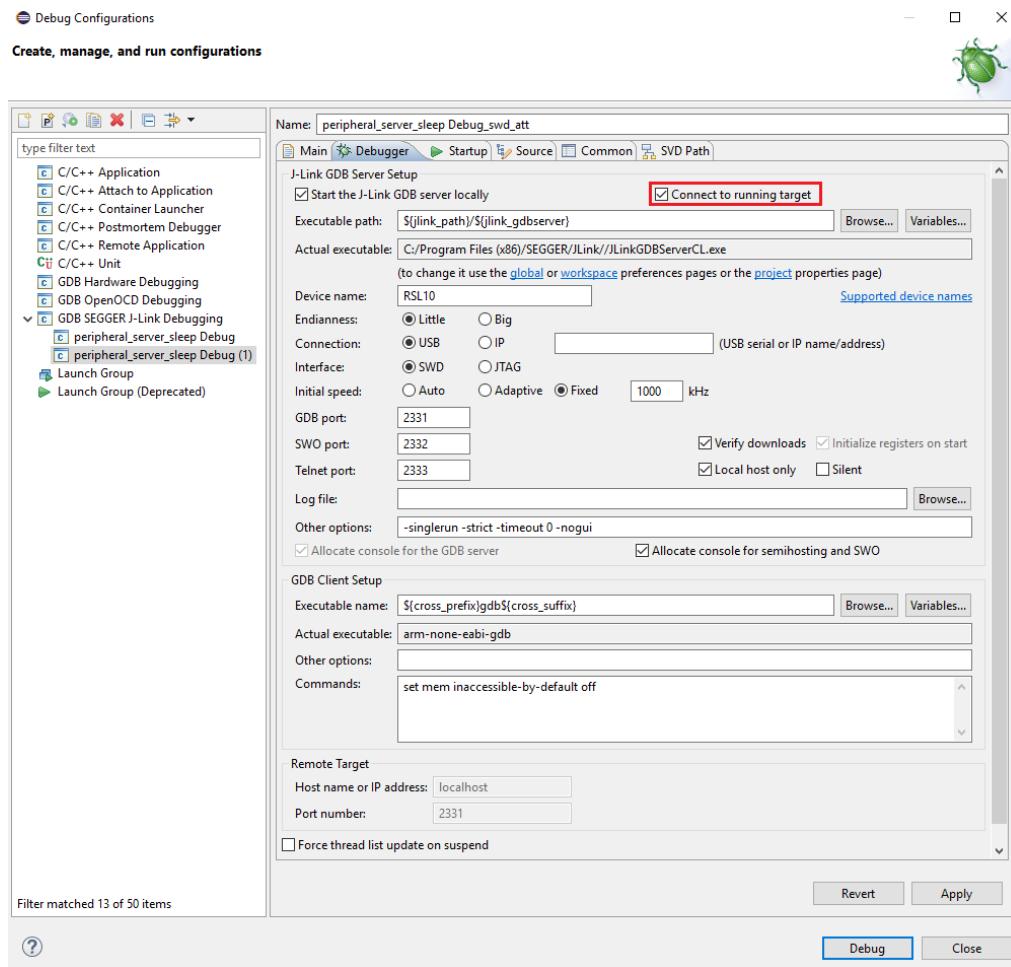


Figure 48. Debugger Tab: Debug Session that Connects to the Running Target

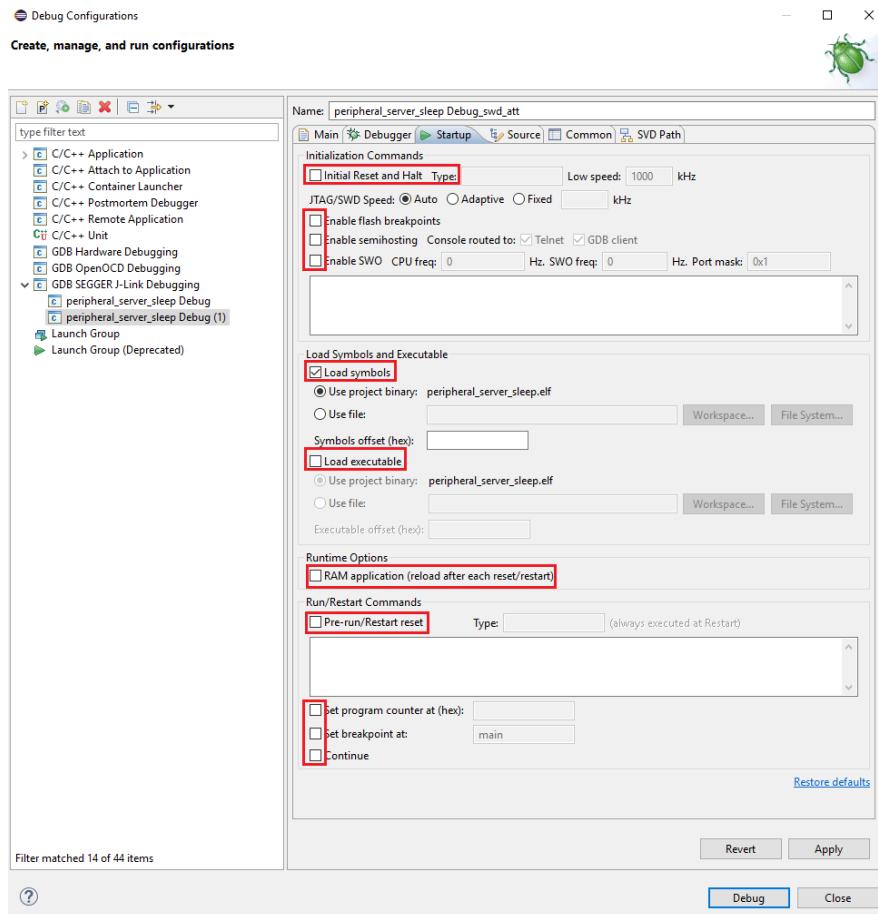


Figure 49. Startup Tab: Debug Session that Connects to the Running Target

5. Start the first debug session (which initiates target restart). Once the target is halted at `main`, resume the execution (see Figure 50).

RSL10 Getting Started Guide

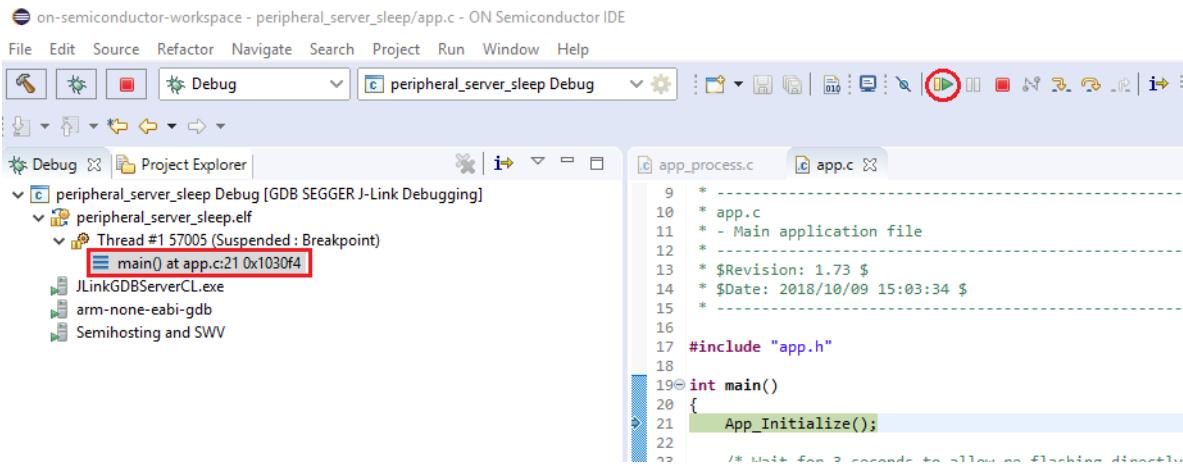


Figure 50. First Debug Session Perspective Before Starting Execution

6. Wait until the target enters Deep Sleep Mode. At this point the debug connection is lost; and even when the target is awake, it cannot establish a connection with JTAG. The following output is generated on the console (see Figure 51).

The screenshot shows the "Console" tab of the ON Semiconductor IDE. The title bar indicates the session is "peripheral_server_sleep Debug [GDB SEGGER J-Link Debugging] JLinkGDBServerCL.exe". The console output is as follows:

```
ERROR: Can not read register 2 (R2) while CPU is running
ERROR: Can not read register 3 (R3) while CPU is running
ERROR: Can not read register 4 (R4) while CPU is running
ERROR: Can not read register 5 (R5) while CPU is running
ERROR: Can not read register 6 (R6) while CPU is running
ERROR: Can not read register 7 (R7) while CPU is running
ERROR: Can not read register 8 (R8) while CPU is running
ERROR: Can not read register 9 (R9) while CPU is running
ERROR: Can not read register 10 (R10) while CPU is running
ERROR: Can not read register 11 (R11) while CPU is running
ERROR: Can not read register 12 (R12) while CPU is running
ERROR: Can not read register 13 (R13) while CPU is running
ERROR: Can not read register 14 (R14) while CPU is running
ERROR: Can not read register 15 (R15) while CPU is running
ERROR: Can not read register 16 (XPSR) while CPU is running
ERROR: Can not read register 17 (MSP) while CPU is running
ERROR: Can not read register 18 (PSP) while CPU is running
ERROR: Can not read register 24 (PRIMASK) while CPU is running
ERROR: Can not read register 25 (BASEPRI) while CPU is running
ERROR: Can not read register 26 (FAULTMASK) while CPU is running
ERROR: Can not read register 27 (CONTROL) while CPU is running
WARNING: Failed to read memory @ address 0xDEADBEEF
Starting target CPU...
ERROR: CPU is not halted
ERROR: Can not read register 15 (R15) while CPU is running
Reading all registers
ERROR: Failed to read memory @ address 0xDEADBEEF
```

Figure 51. Debug Session Perspective when Debug Connection is Lost

7. Stop the debug session and click on the Terminate icon to remove all terminated targets (see Figure 52).

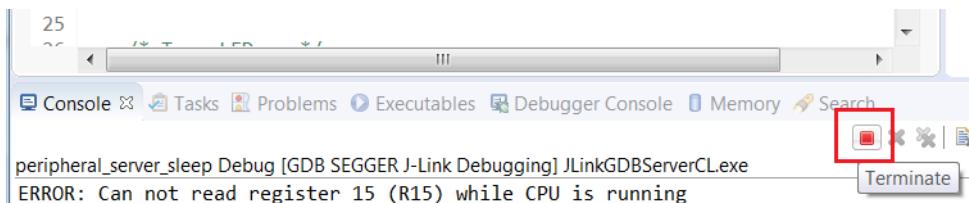


Figure 52. Terminate Targets Icon

- After the target exits Deep Sleep Mode, it is running in the infinite loop (step 1), and you can connect to the running target by starting the second debug session (see Figure 53). Note that the debugger is able to reattach to the running target and halt the processor after waking up from sleep.

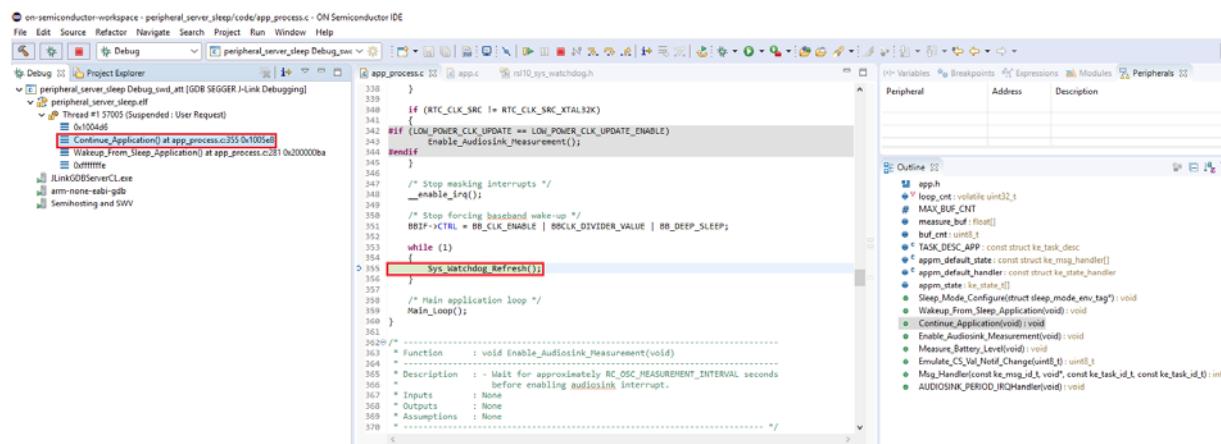


Figure 53. Second Debug Session Perspective After Connecting to the Running Target

7.4.1 Downloading Firmware in Sleep Mode

If an application with Sleep Mode is currently on your board, and changing the **Reset Type** to 1 as described in Section 7.4, “Debugging with Low Power Sleep Mode” is not working, try the following:

- Connect DIO12 to ground.
- Press the RESET button (this restarts the application, which pauses at the start of its initialization routine).
- Repeat step 2 above. After successfully downloading *blinky* to flash memory, disconnect DIO12 from ground, and press the RESET button so that the application works properly.

Alternatively, use the Stand-Alone Flash Loader (available with its own manual in the *RSL10.Utility_Apps.zip* file) to erase the application with Sleep Mode from the board’s flash memory.

CHAPTER 8

More Information

8.1 FOLDER STRUCTURE OF THE RSL10 CMSIS-PACK INSTALLATION

By default, the CMSIS-Pack contents are installed in the following location:

- If you are using the Eclipse-based ON Semiconductor IDE:
C:\Users\<user_id>\ON_Semiconductor\PACK\ONSemiconductor\RSL10\<version>.
- If you are using the Keil IDE: *%LOCALAPPDATA%\Arm\Packs*
- If you are using the IAR IDE:
C:\Users\<user_name>\IAR-CMSIS-Packs\ONSemiconductor\RSL10\<version>

Subfolders and files are described in Table 1 and Table 2.

Table 1. Installed Folders and Files - CMSIS-Pack

Folder	Contents	
<i>configuration</i>	J-Link flash loader files.	
<i>documentation</i>	Hardware, firmware and software documentation in PDF format. Also 3rd-party documentation from other companies besides ON Semiconductor. Available from the books tab in the IDE.	
<i>images</i>	Contains evaluation board pictures.	
<i>include</i>	Include files for the firmware components and libraries. Projects can point to this directory and sub-directories when including firmware header files.	
<i>lib</i>	Pre-built libraries which can be linked to by sample code or other source code. Project linker settings must point to this directory when linking with firmware libraries.	
<i>source</i>	<i>firmware</i>	The source of the provided support libraries.
	<i>samples/rslx</i> (for ON Semiconductor IDE)	Sample code sources as ready-to-build projects.
	<i>samples/uv</i> (for Keil IDE)	
<i>samples/iar</i> (for IAR IDE)		
<i>svd</i>	Contains the System View Description file used in the registers view during debugging.	
<i>ONSemiconductor.RSL10.pdsc</i>	A file that describes the dependencies to devices, processor, toolchains and other software components for the RSL10 CMSIS-Pack.	
<i>PACK_REVISION</i>	Identifies the revision of the RSL10 CMSIS-Pack.	
<i>Software_Use_Agreement.rtf</i>	ON Semiconductor license agreement.	

Table 2. Installed Folders and Files - ON Semiconductor IDE

Folder	Contents
<i>arm_tools</i>	The Arm Toolchain is installed here.
<i>eclipse</i>	Pre-built libraries which can be linked to by sample code or other source code. Project linker settings must point to this directory when linking with firmware libraries.
<i>jre*</i>	The included JAVA runtime environment.
<i>ide.exe</i>	Executable that opens the ON Semiconductor IDE.
<i>REVISION</i>	Identifies the revision of the ON Semiconductor IDE.
<i>Software_Use_Agreement.rtf</i>	ON Semiconductor license agreement.
<i>ThirdPartyLicenses.txt</i>	License agreements with third party software included in the IDE.

8.2 DOCUMENTATION

8.2.1 Documentation Included with the CMSIS-Pack

A set of documents is included with the CMSIS-Pack installation in `C:\Users\<user_id>\ON_Semiconductor\PACK\ONSemiconductor\RSL10\<version>\documentation` (where `<user_id>` is your profile name, and `<version>` is the version number, e.g., 3.0.521).

These documents are also accessible via any of the three IDEs:

- ON Semiconductor IDE: documentation is accessible through the C/C++ perspective by opening any RTE configuration file, such as `blinky.rteconfig`, and selecting the tab **Device** (see Figure 54, below).
- Keil µVision IDE: documentation is available in the **Books** tab, as shown in Figure 55.
- IAR Embedded Workbench: documentation is accessible through the **IAR Embedded Workbench CMSIS Manager** window, as shown in Figure 56 on page 47.

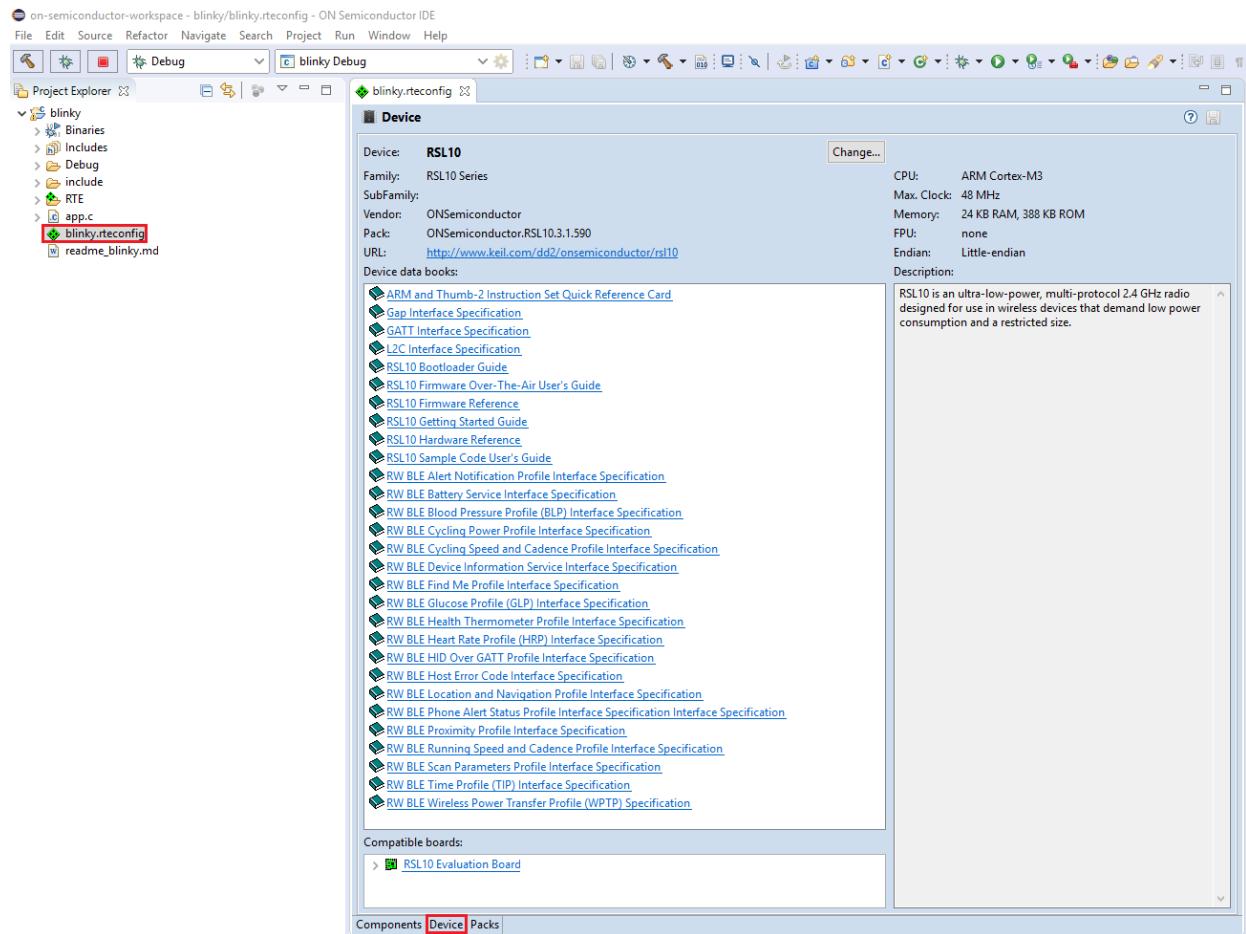


Figure 54. Accessing RSL10 Documentation from the ON Semiconductor IDE

RSL10 Getting Started Guide

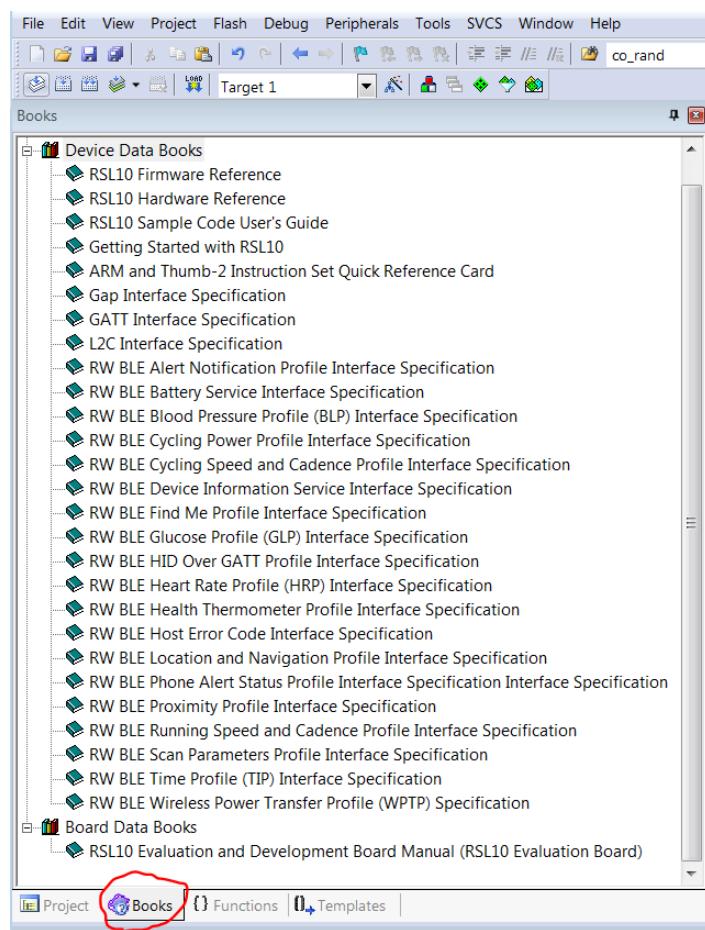


Figure 55. Accessing RSL10 Documentation from the Keil μVision IDE

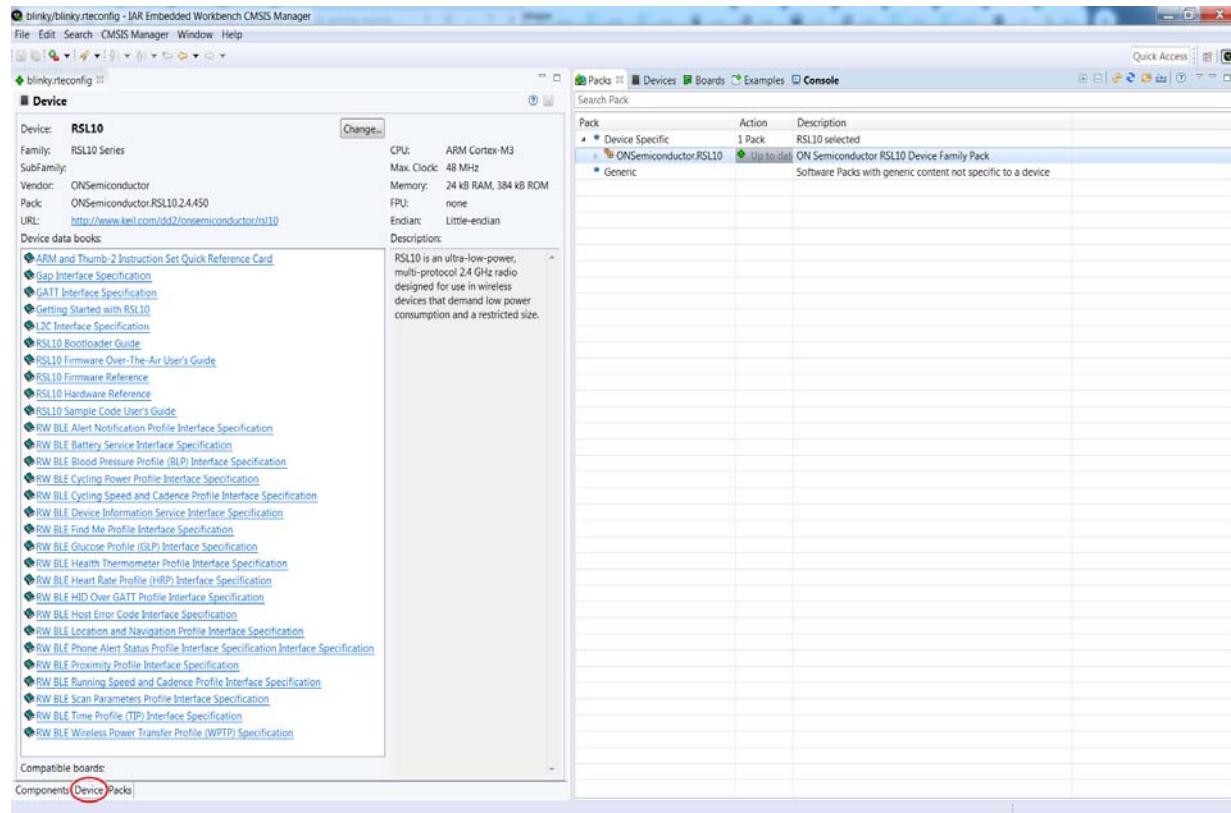


Figure 56. Accessing RSL10 documentation from the IAR Embedded Workbench

For more information, see the following:

Arm and Thumb®-2 Instruction Set Quick Reference Card

From the Arm company, this quick reference card provides a short-hand list of instructions for the Arm Cortex-M3 processor.

RSL10 Evaluation and Development Board Manual

This document actually contains a link to the manual that is stored elsewhere on the website. It is a reference manual that provides detailed information on the configuration and use of the RSL10 Evaluation and Development Board. When you use this board with the software development tools, you can test and measure the performance and capabilities of the RSL10 radio SoC.

RSL10 Firmware Reference

The system firmware provides functionality that isolates you from the hardware, and implements complex but common tasks, making it easier to support and maintain your code. The Bluetooth firmware provides an implementation of the Bluetooth host, controller, and profiles, supporting the standards-compliant use of these components within your application. This manual provides a reference to both sets of firmware features, and explains how they can assist with the development of your applications.

RSL10 Getting Started Guide

RSL10 Hardware Reference

Describes all the functional features provided by the RSL10 SoC, including how these features are configured and how they can be used. This manual is a good place to start when you are designing real-time implementations of your algorithms, or planning a product based on the RSL10 SoC.

RSL10 Sample Code User's Guide

Explains how to use the sample applications provided with the RSL10 software development tools. You learn about setting up your system, accessing code files, and how the sample applications work, using the Peripheral Device with Server sample code as the prime example.

RivieraWaves Interface Specifications (files in the ceva folder)

Interface Specifications from RivieraWaves provide a description of the API for the specified library:

- GAP Interface Specification
- GATT Interface Specification
- Host Error Code Interface Specification
- L2C Interface Specification
- RW BLE Alert Notification Profile Interface Specification
- RW BLE Battery Service Interface Specification
- RW BLE Blood Pressure Profile (BLP) Interface Specification
- RW BLE Cycling Power Profile Interface Specification
- RW BLE Cycling Speed and Cadence Profile Interface Specification
- RW BLE Device Information Service Interface Specification
- RW BLE Find Me Profile Interface Specification
- RW BLE Glucose Profile (GLP) Interface Specification
- RW BLE HID Over GATT Profile Interface Specification
- RW BLE Heart Rate Profile (HRP) Interface Specification
- RW BLE Health Thermometer Profile Interface Specification
- RW BLE Location and Navigation Profile Interface Specification
- RW BLE Phone Alert Status Profile Interface Specification
- RW BLE Proximity Profile Interface Specification
- RW BLE Running Speed and Cadence Profile Interface Specification
- RW BLE Scan Parameters Profile Interface Specification
- RW BLE Time Profile (TIP) Interface Specification
- RW BLE Wireless Power Transfer System Profile Interface Specification

LPDSP32 Documentation

The following documents are available in the *RSL10_LPDSP32_Support.zip* file:

- *RSL10 Getting Started with the LPDSP32 Processor*, which provides an overview of the techniques involved when writing and integrating code for the LPDSP32 processor that is on RSL10.
- *LPDSP32-V3 Block Diagram*, which provides a drawing of all the inputs, outputs, components and process blocks
- *LPDSP32-V3 Hardware Reference Manual*, which describes the hardware aspects of the LPDSP32-V3 core and its operations to provide an understanding of the core architecture and various kinds of supported operations.
- *LPDSP32-V3 Interrupt Support Manual*, which describes how interrupts are supported.

- *User Guide IP Programmers for LPDSP32-V3*, which describes the C application layer, the flow generally followed when any application is ported to LPDSP32, various tips for optimization to make the best use of the processor and compiler resources, and certain things the programmers should be aware of when porting applications. It also provides a few examples to show the usage of LPDSP32 intrinsic functions and to give an idea of how certain DSP functions can be ported to and optimized for LPDSP32.

RSL10 Release Notes

Lists new features in the latest release and known issues. This file is downloaded with the installer in a zip file, and is not in the *documentation* folder.

8.2.2 Documentation in the RSL10 Documentation Package

You can access documentation through the *RSL10 DOCUMENTATION PACKAGE.ZIP* file available with this release of RSL10. It contains all of the documents included with the CMSIS-Pack as well as the following:

Getting Started with RSL10 Bluetooth Low Energy Mesh

Helps you to get started with the RSL10 mesh package. It guides you through the process of installing the mesh package alongside the RSL10 SDK, configuring your environment, and building and debugging your first RSL10 mesh network.

RSL10 Bluetooth Low Energy Mesh Sample Code User's Guide

Shows you what the mesh sample application (*ble_mesh*) demonstrates, how to configure the project to set up different mesh network scenarios, and how to experiment with them to verify their features and operations.

Files in the mindtree folder (related to Bluetooth Low Energy Mesh networking)

- *EtherMind_Mesh_API.chm*
- *EtherMind_Mesh_Application_Developer's_Guide_Generic.pdf*
- *EtherMind_Mesh_CLI_User_Guide.pdf*

RSL10 Bootloader Guide

The RSL10 bootloader provides means of performing firmware updates using the UART interface, and is a required component for Firmware Over the Air (FOTA). The bootloader enables firmware updates without the use of the JTAG interface. Firmware can be loaded from a host microcontroller over UART or over the air from another wireless device using FOTA. The bootloader copies the firmware image to the designated location in flash memory. This document describes the bootloader firmware application and development tools.

RSL10 Firmware Over-The-Air User's Guide

This manual describes Firmware Over-The-Air (FOTA) with RSL10. It provides the prerequisites and instructions necessary to develop FOTA-ready firmware applications and to perform FOTA updates in the field.

RSL10 LPDSP32 Support Manual

Provides an overview of the techniques involved when writing and integrating code for the LPDSP32 processor included with the RSL10 radio System-on-Chip (SoC).

RSL10 Getting Started with the LPDSP32 Processor

Provides an overview of the techniques involved when writing and integrating code for the LPDSP32 processor that is on RSL10.

RSL10 Getting Started Guide

Manuals in the lpdsp32 folder:

- *LPDSP32-V3 Block Diagram:* provides a drawing of all the inputs, outputs, components and process blocks
- *LPDSP32-V3 Hardware Reference Manual:* Describes the hardware aspects of the LPDSP32-V3 core and its operations to provide an understanding of the core architecture and various kinds of supported operations
- *LPDSP32-V3 Interrupt Support Manual:* Describes how interrupts are supported
- *User Guide IP Programmers for LPDSP32-V3:* Describes the C application layer, the flow generally followed when any application is ported to LPDSP32, various tips for optimization to make the best use of the processor and compiler resources, and certain things the programmers should be aware of when porting applications. It also provides a few examples to show the usage of LPDSP32 intrinsic functions and to give an idea of how certain DSP functions can be ported to and optimized for LPDSP32.

RSL10 Stand Alone Flash Loader Manual

Provides the information that you need to use the stand-alone flash loader. It describes the operations that the flash loader can perform, and explains how to configure the flash loader to connect to an RSL10 radio IC. The stand-alone flash loader is used to program, erase and read flash memory in RSL10.

APPENDIX A

Migrating to CMSIS-Pack

If you have an existing project and have not used the RSL10 CMSIS-Pack before, this section is for you. Starting from SDK 3.0, the RSL10 firmware is no longer bundled with the Eclipse IDE. The RSL10 Eclipse IDE has been optimized and rebranded as the ON Semiconductor IDE, and the RSL10-specific firmware is now delivered exclusively as a separate CMSIS-Pack that can be imported into the IDE. For future RSL10 releases, you only need to download and import the updated CMSIS-Pack. There is no need to re-install the Eclipse IDE if it has not been updated.

Existing Eclipse project files from previous SDK releases are not compatible with the new ON Semiconductor IDE. Fortunately, migrating your existing project into the new IDE to take advantage of the CMSIS-Pack standard is a straightforward process, as shown in the next section.

A.1 MIGRATING AN EXISTING ECLIPSE PROJECT TO THE CMSIS-PACK METHOD

In order to tell whether your project is managed by CMSIS-Packs, check that a file with the *.rteconfig* extension is present in the project folder. If not, your project is not managed by CMSIS-Packs and needs to be migrated. The easiest way to migrate your existing Eclipse project to the new IDE is to start from one of the CMSIS-Pack RSL10 sample projects, and follow these steps:

NOTE: This section assumes you know how to import the CMSIS-Pack and a sample application, as shown in Chapter 3, “Getting Started with the Eclipse-Based ON Semiconductor IDE” on page 7.

1. Decide on which CMSIS-Pack sample project to import. It is best to import a CMSIS-Pack project that looks similar (in terms of libraries used) to the existing project you would like to migrate. For example, if your existing application uses the Heart Rate Profile, you might want to import the *ble_peripheral_server_hrp* sample application as a reference.
2. Right-click the project and rename it as you wish.
3. Remove the source code from the sample project.
4. Copy over the source and header files from your existing project into the new one.
5. Open the RTE Configuration Wizard by double-clicking the *.rteconfig* file, and make sure all the software components (libraries) required for your project are selected.
 - Pay special attention to the Bluetooth components, such as the Bluetooth Low Energy Stack, Kernel, and Profiles. Ensure that these components have the correct variants selected (such as *release*, *release_light*, or *release_hci*).
 - Some libraries might have been removed, such as the *weakprf.a*. This library has been replaced by the *stubprf.c* file that is automatically added together with the Bluetooth Low Energy Stack component, so you no longer need to explicitly reference it.
 - You can also remove (deselect) the software components that you do not need in your existing application.
 - If you change the *.rteconfig* file, make sure to save it, so that it can update your project settings automatically (such as the library paths, includes, etc.) to reflect the newly added or removed software components.
6. Navigate to your project settings and add or remove the preprocessor *symbol* or *include* folders from your existing project.
7. Build your application and make sure it builds correctly.
 - In case of build errors related to missing components, files, or preprocessor symbols, go back to steps 5 and 6 and review your configuration carefully.
 - If you encounter errors related to duplicated code, review the *RTE* folder in your application. Some files that were common to multiple sample applications have been transformed into software components, such as the BLE Abstraction, CMSIS-Drivers, etc.

- For errors related to deprecated code or API changes, review the latest RSL10 CMSIS-Pack release notes and check to see if there are any feature changes that could affect your project.

A.2 USING THE LATEST RSL10 FIRMWARE IN A PREVIOUS VERSION OF THE ECLIPSE-BASED IDE

We recommend always updating your installation to the latest version of the Eclipse-based ON Semiconductor IDE. However, if your circumstances are such that this is impractical, you can manually update the RSL10 firmware files in a previous version of the Eclipse-based IDE. If this is your case, try the following steps:

1. Download the **RSL10 Software Package** from www.onsemi.com/RSL10 and extract the RSL10 CMSIS-Pack (ONSemiconductor.RSL10.<version>.pack) to any temporary folder.
2. Use a compressing tool, such as 7-Zip, and extract the contents of the *ONSemiconductor.RSL10.<version>.pack* file.
3. Copy and replace the *lib* and *include* folders from the CMSIS-Pack into your existing RSL10 SDK Installation folder.
4. Clean and build your application. If the build has been successful, you can see that it now references the updated libraries and include files.

In case of build errors, make sure to review the latest release notes from the CMSIS-Pack and check to see if there are any features or bug fixes that affect your application.

APPENDIX B

Arm Toolchain Support

There are several ways in which the ON Semiconductor 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.

B.1 BASIC INSTALLATION

The ON Semiconductor IDE supports the Arm toolchain by installing it in the *arm_tools* directory within the installed RSL10 software tools 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 ON Semiconductor 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 ON Semiconductor IDE access to the toolchain installed with the RSL10 software tools.

Conflicts with toolchain versions can occur in the ON Semiconductor 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*.

B.2 CONFIGURING THE ARM TOOLCHAIN IN THE ON SEMICONDUCTOR 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 **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.

B.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.

RSL10 Getting Started Guide

Windows is a registered trademark of Microsoft Corporation. Arm, Cortex, Keil, and uVision are registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. IAR and IAR Embedded Workbench are trademarks or registered trademarks of IAR Systems AB. All other brand names and product names appearing in this document are trademarks of their respective holders.

ON Semiconductor and the ON Semiconductor logo are trademarks of Semiconductor Components Industries, LLC dba ON Semiconductor or its subsidiaries in the United States and/or other countries. ON Semiconductor owns the rights to a number of patents, trademarks, copyrights, trade secrets, and other intellectual property. A listing of ON Semiconductor's product/patent coverage may be accessed at www.onsemi.com/site/pdf/Patent-Marking.pdf. ON Semiconductor is an Equal Opportunity/Affirmative Action Employer. This literature is subject to all applicable copyright laws and is not for resale in any manner.

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. ON Semiconductor 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.

ON Semiconductor 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 ON Semiconductor shall not constitute any representation or warranty by ON Semiconductor, and no additional obligations or liabilities shall arise from ON Semiconductor 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 ON Semiconductor 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 ON Semiconductor 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 ON Semiconductor 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.

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

LIMITATIONS OF LIABILITY: ON Semiconductor 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 ON Semiconductor is advised of the possibility of such damages. In no event shall ON Semiconductor'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 ON Semiconductor
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

ON Semiconductor Website: www.onsemi.com

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

For additional information, please contact your local Sales Representative