RSL10 Getting Started Guide

AND9697/D
November 2019, Rev. P7

ON Semiconductor®

Table of Contents

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.

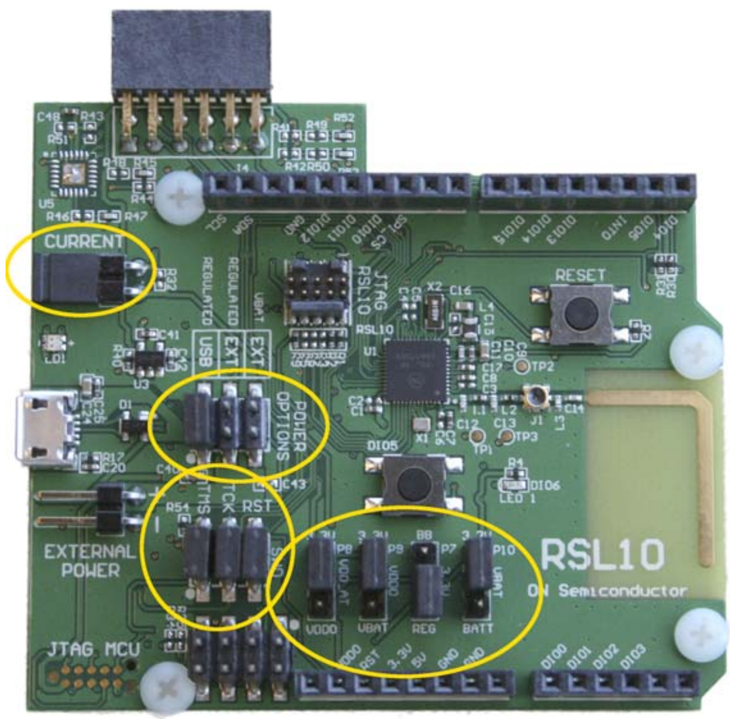

Figure 1. Evaluation and Development Board with Pins and Jumpers for Connection Highlighted

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:

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

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>`).

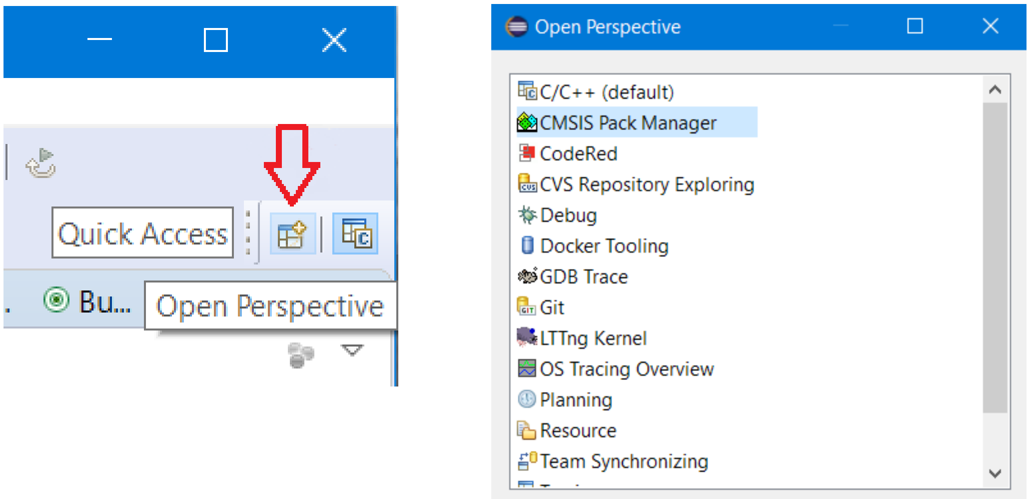

Figure 2. Opening the CMSIS Pack Manager Perspective

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).

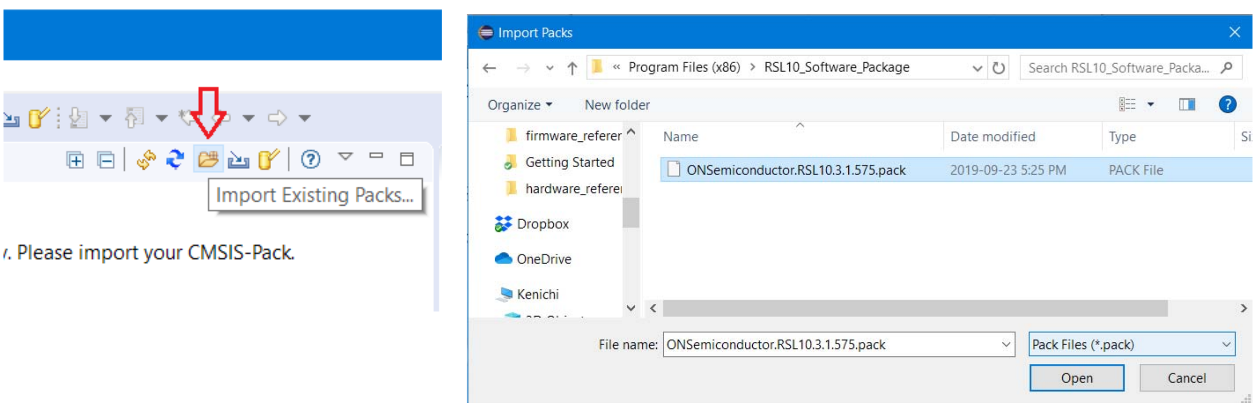

Figure 3. Installing the RSL10 CMSIS-Pack

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.

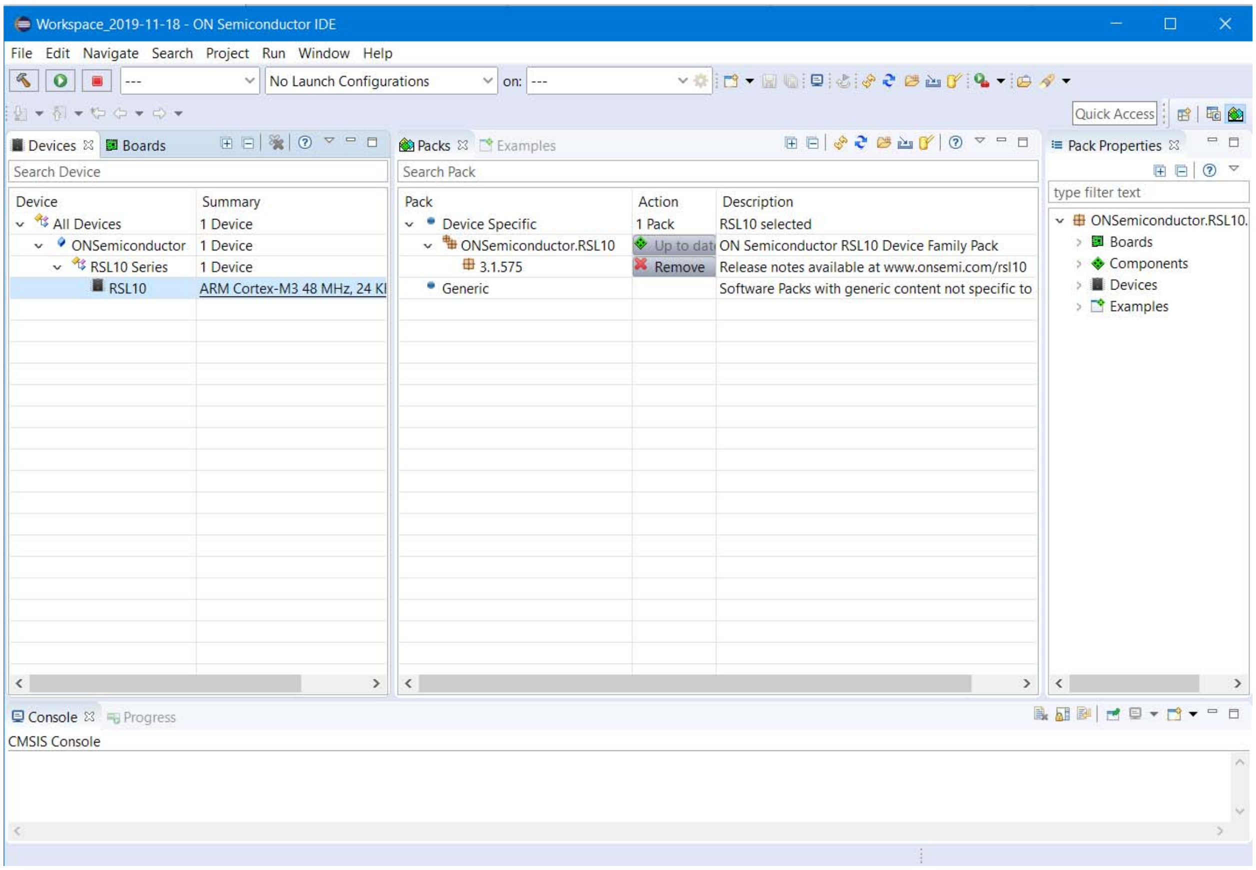

Figure 4. Pack Manager Perspective after RSL10 CMSIS-Pack is Installed

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).

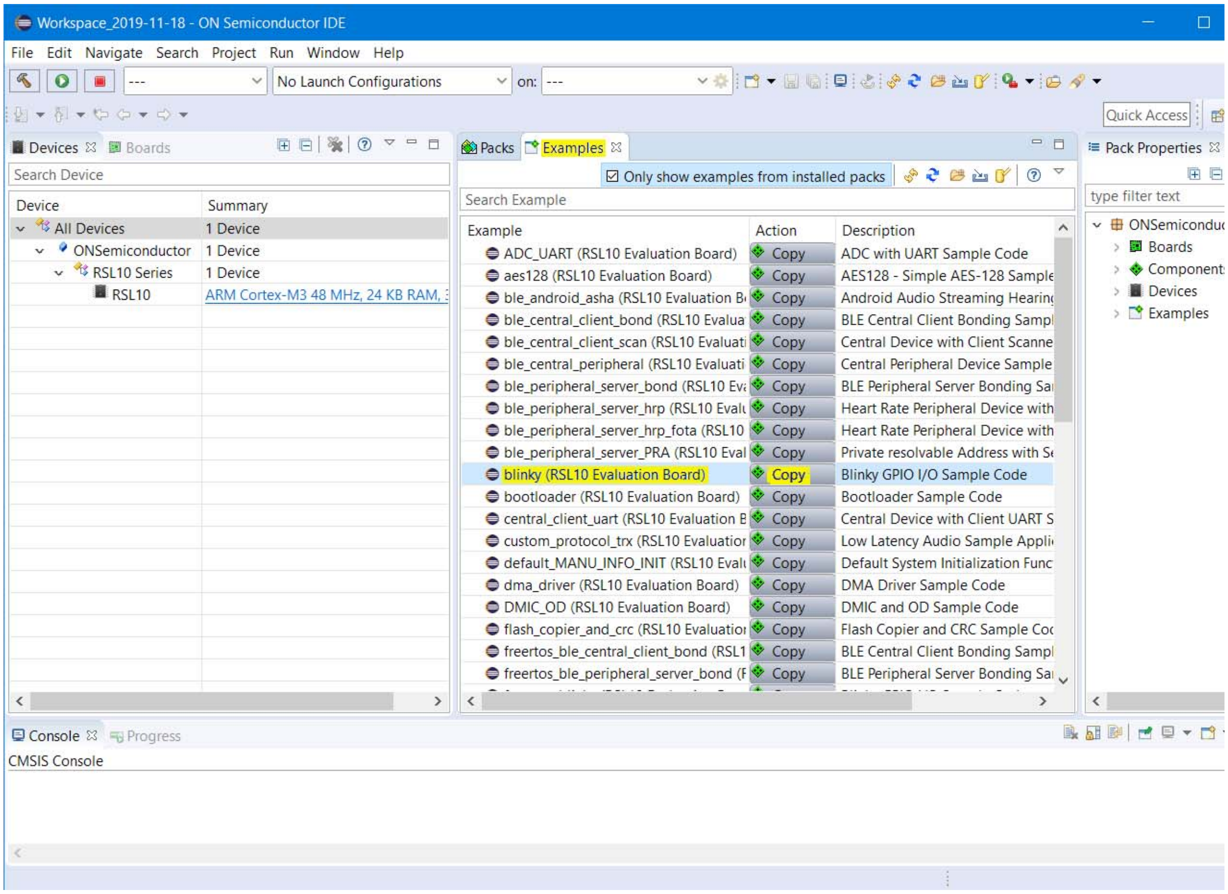

Figure 5. Pack Manager Perspective: Examples Tab

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 *blinky.rteconfig* file displays software components. If you expand **Device > Libraries**, you can see the **System library** (*libsyslib*) and the **Startup** (*libcmsis*) components selected for *blinky* (see Figure 6).

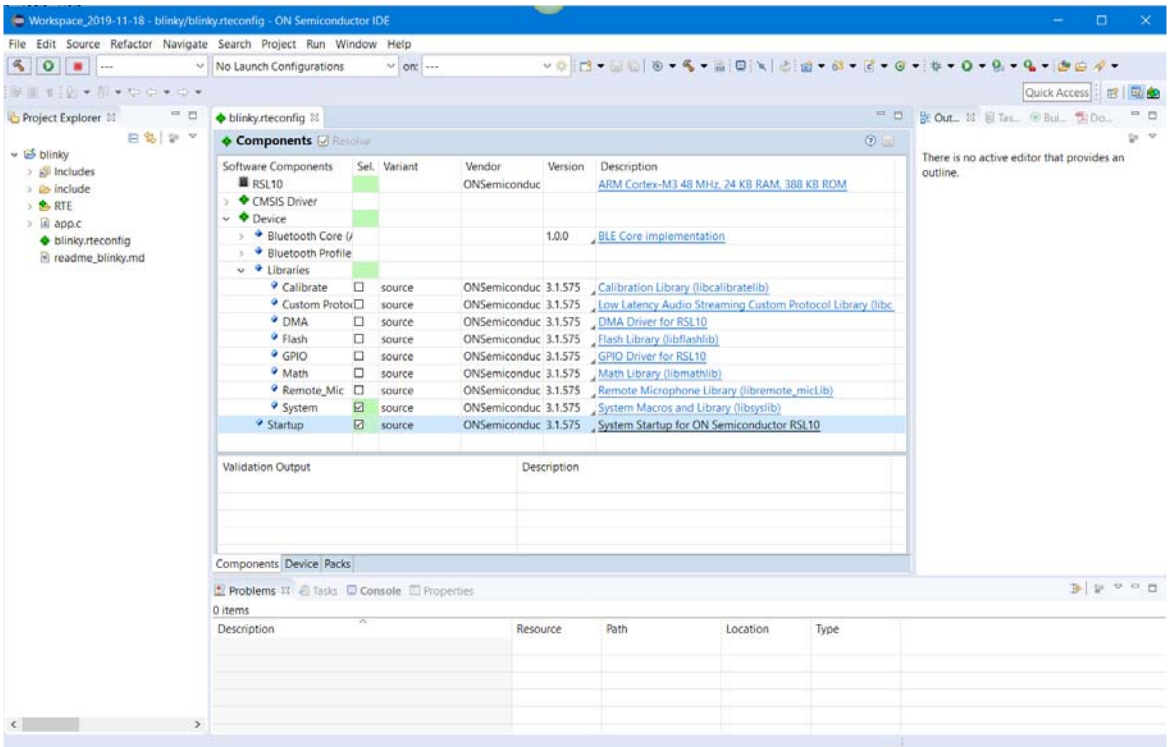

Figure 6. RTE Configuration for the Blinky Example Project in the ON Semiconductor IDE

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.

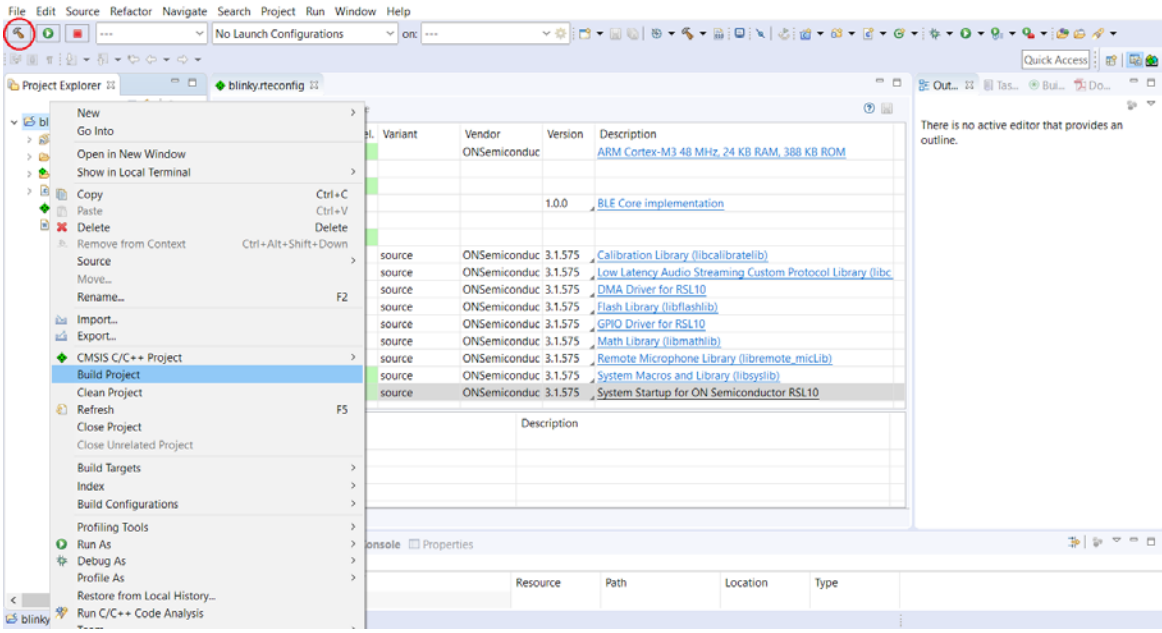

Figure 7. Starting to Build a Project in the ON Semiconductor IDE

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.

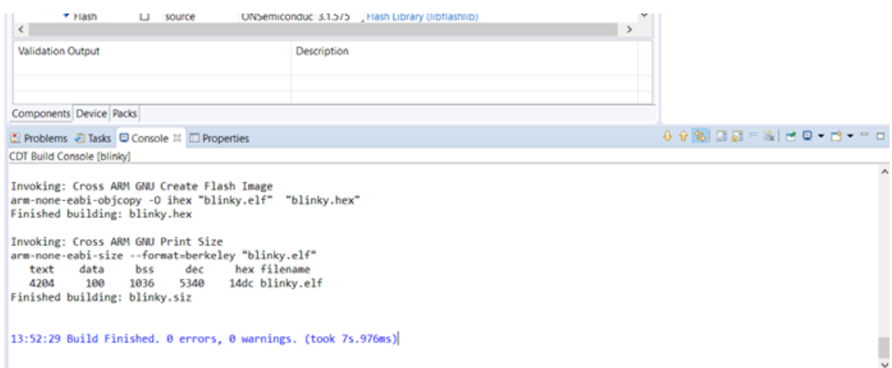

Figure 8. Example of Build Output

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.

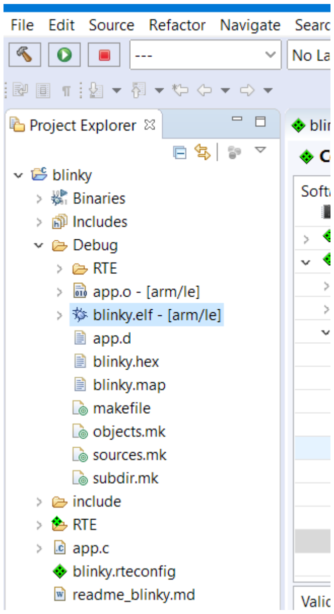

Figure 9. Output Files from Building a Sample Project

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 Explore**r, 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).

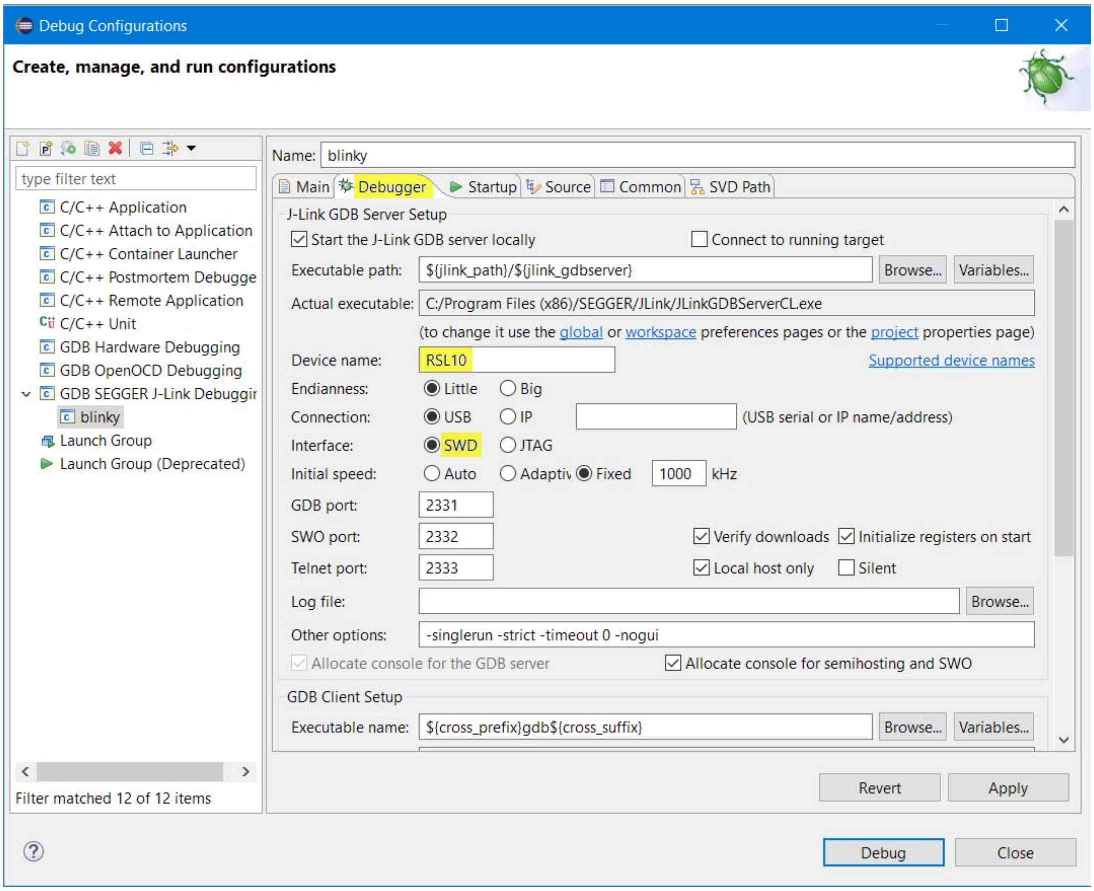

Figure 10. Setting Up a GDB Launch Configuration, Debugger Tab

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)`.

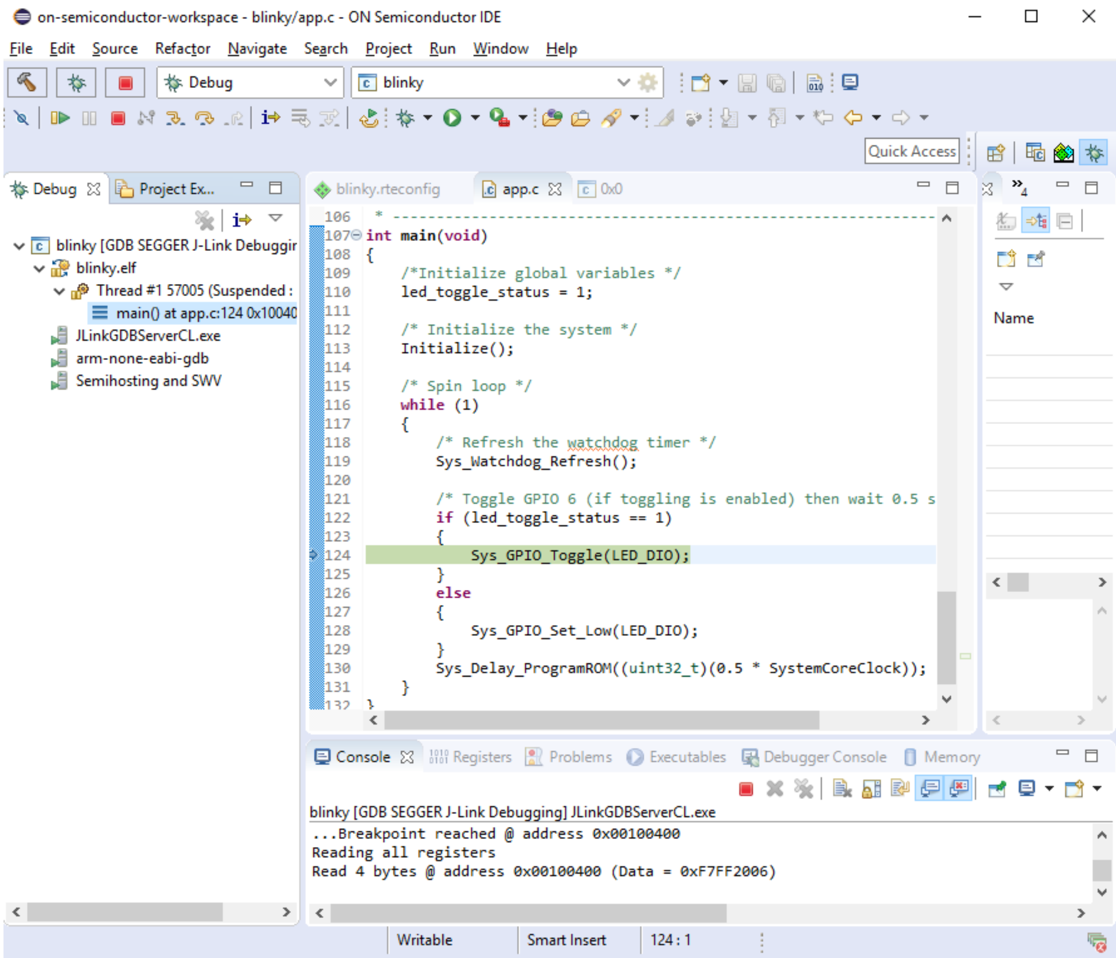

Figure 11. Debug Perspective

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 *blinky.elf* 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**.

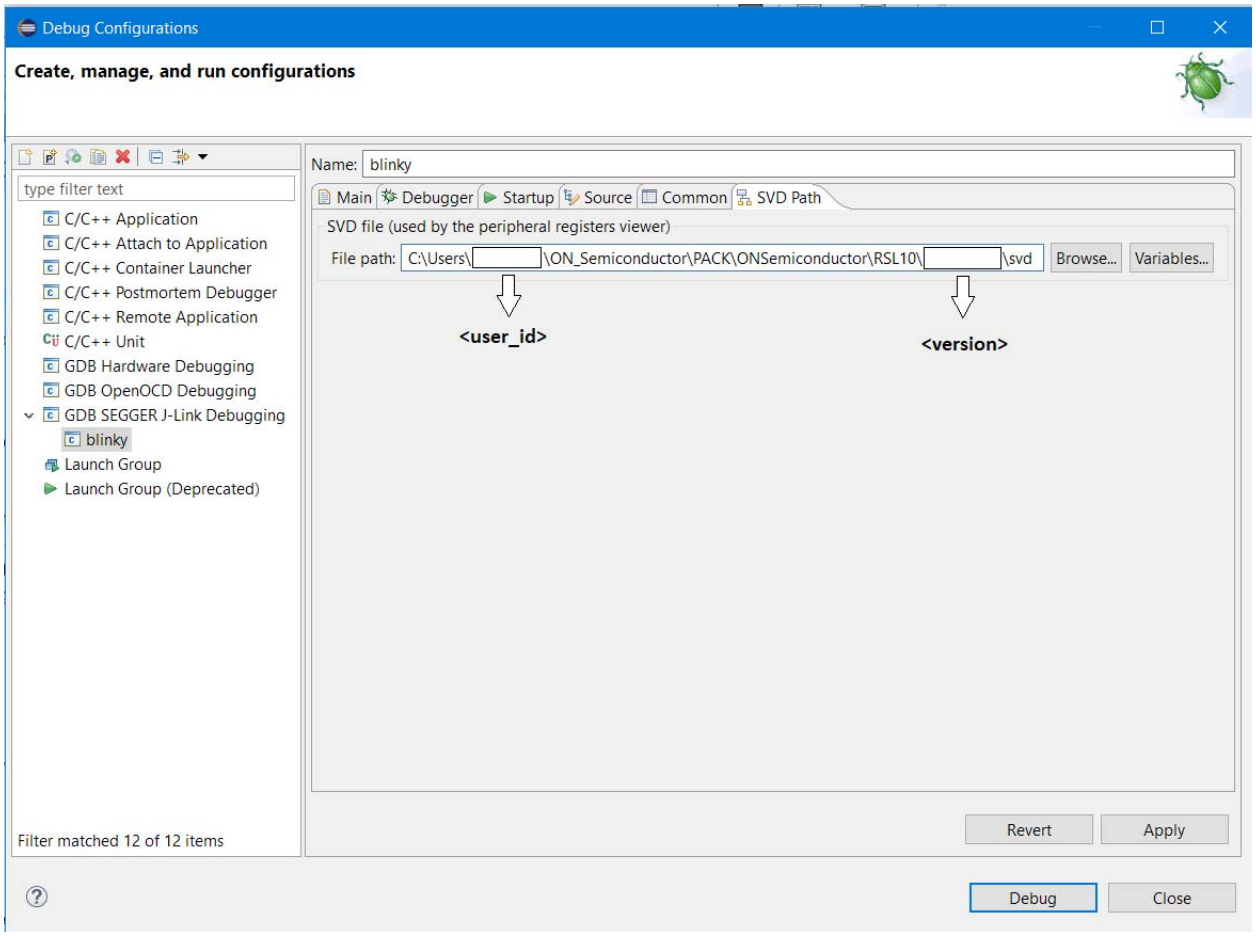

Figure 12. SVD Path Tab Perspective

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).

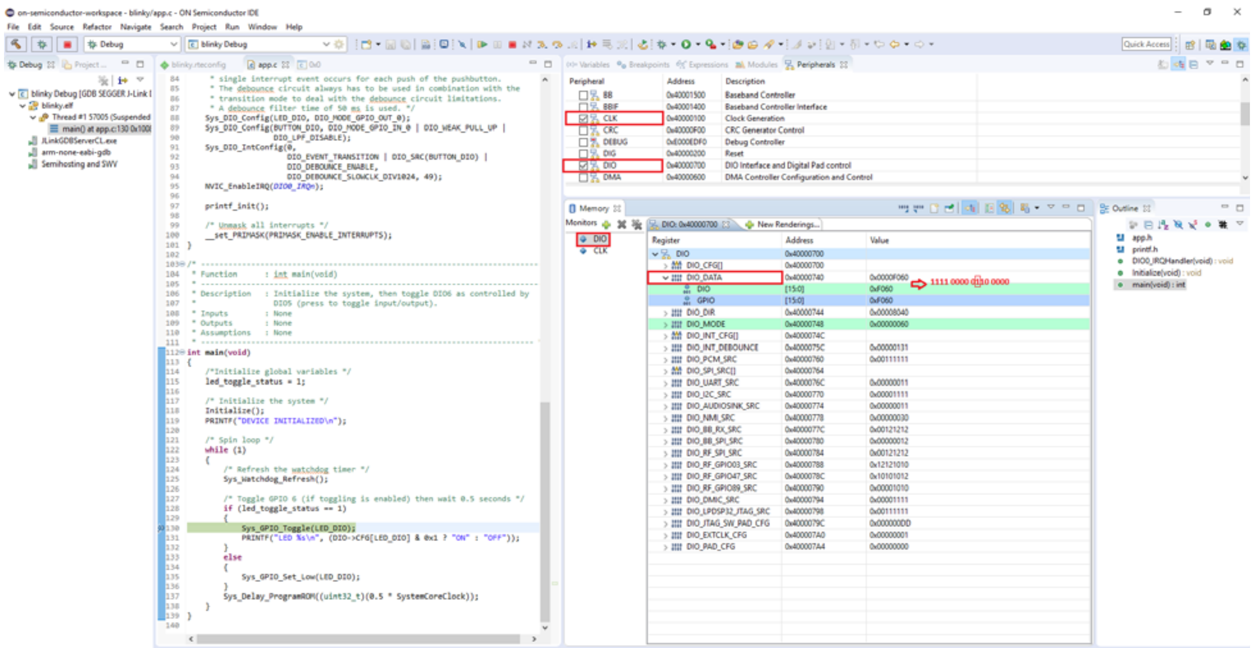

Figure 13. Peripheral Registers View Perspective in Debug Session After Setting SVD Path

7. 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.

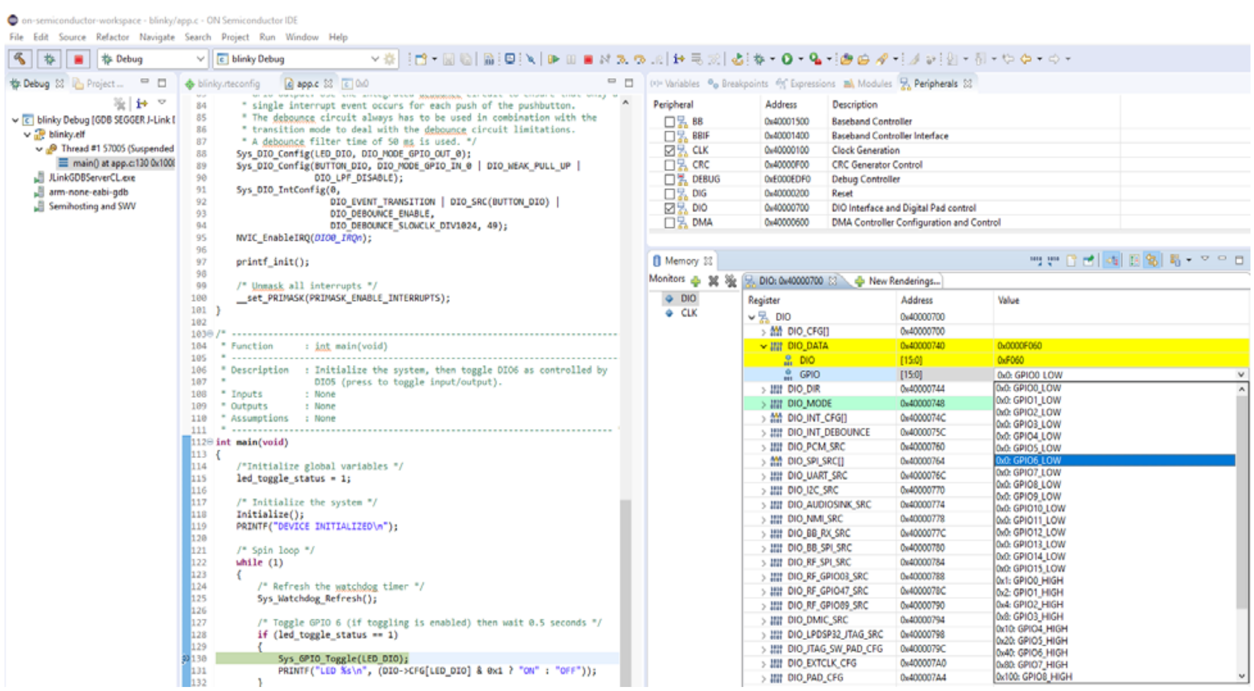

Figure 14. Toggling RSL10 DIO Using the Peripheral Registers View: Before

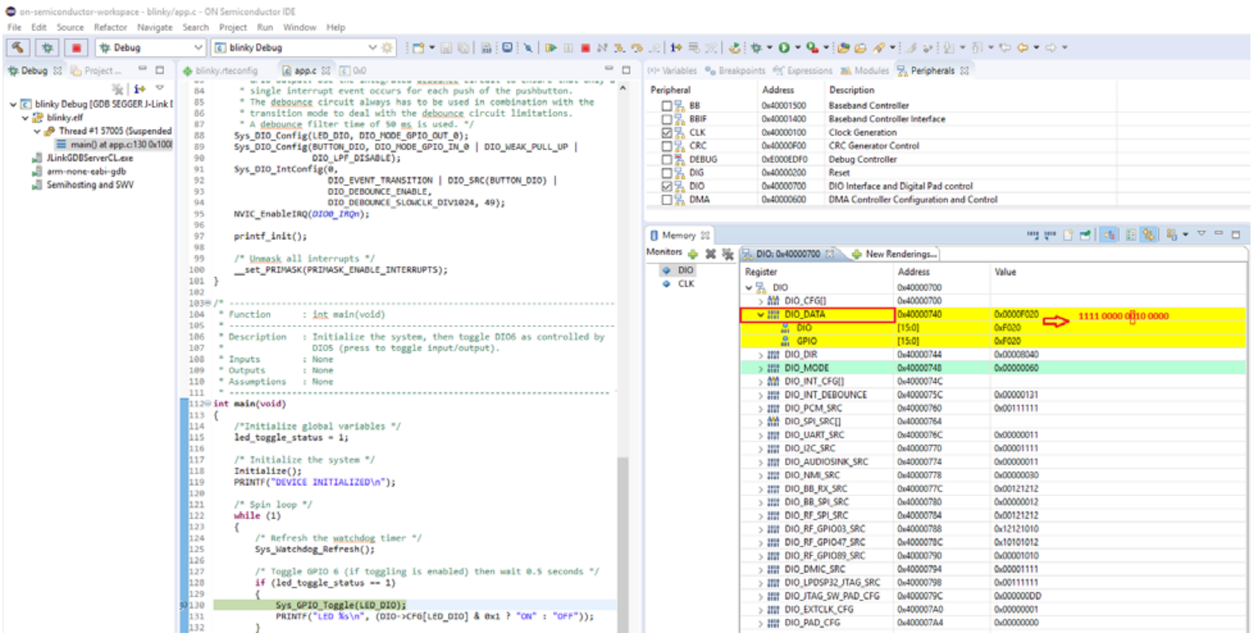

Figure 15. Toggling RSL10 DIO Using the Peripheral Registers View: After

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.

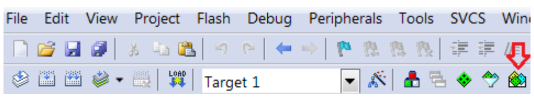

Figure 16. Pack Installer Icon

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.

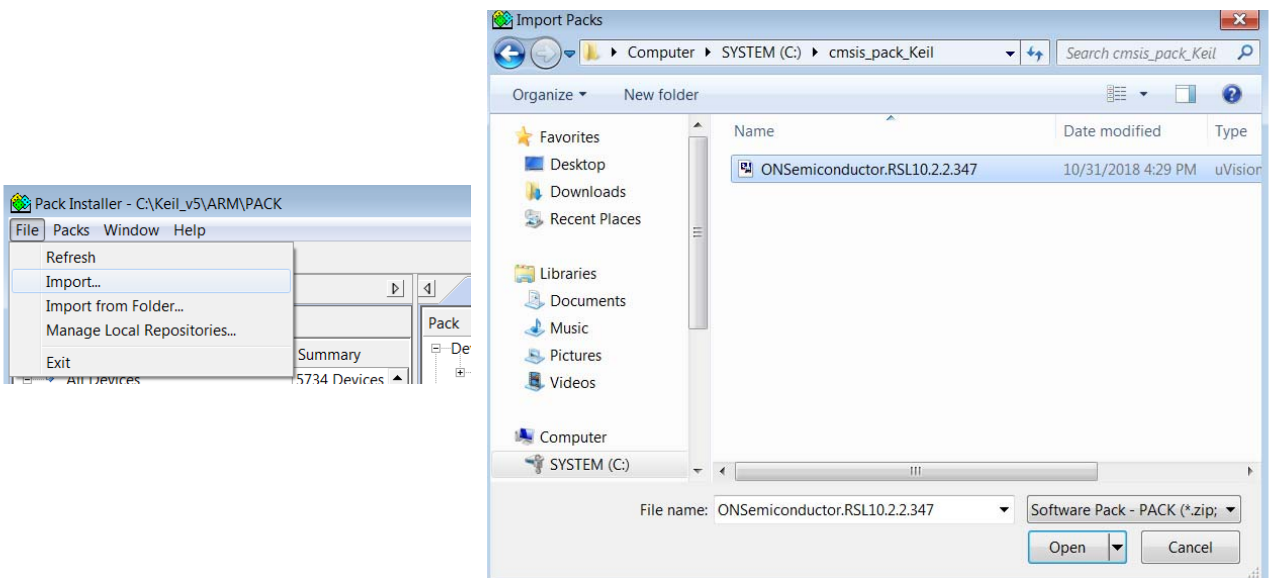

Figure 17. Installing the RSL10 **CMSIS-Pack for the Keil µVision IDE**

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 proprieties.

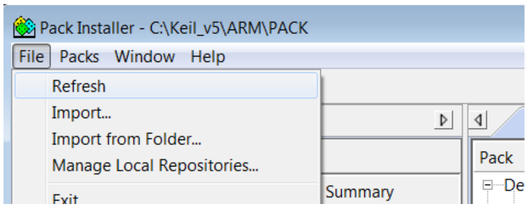

Figure 18. Refresh Pack after installation

5. 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.

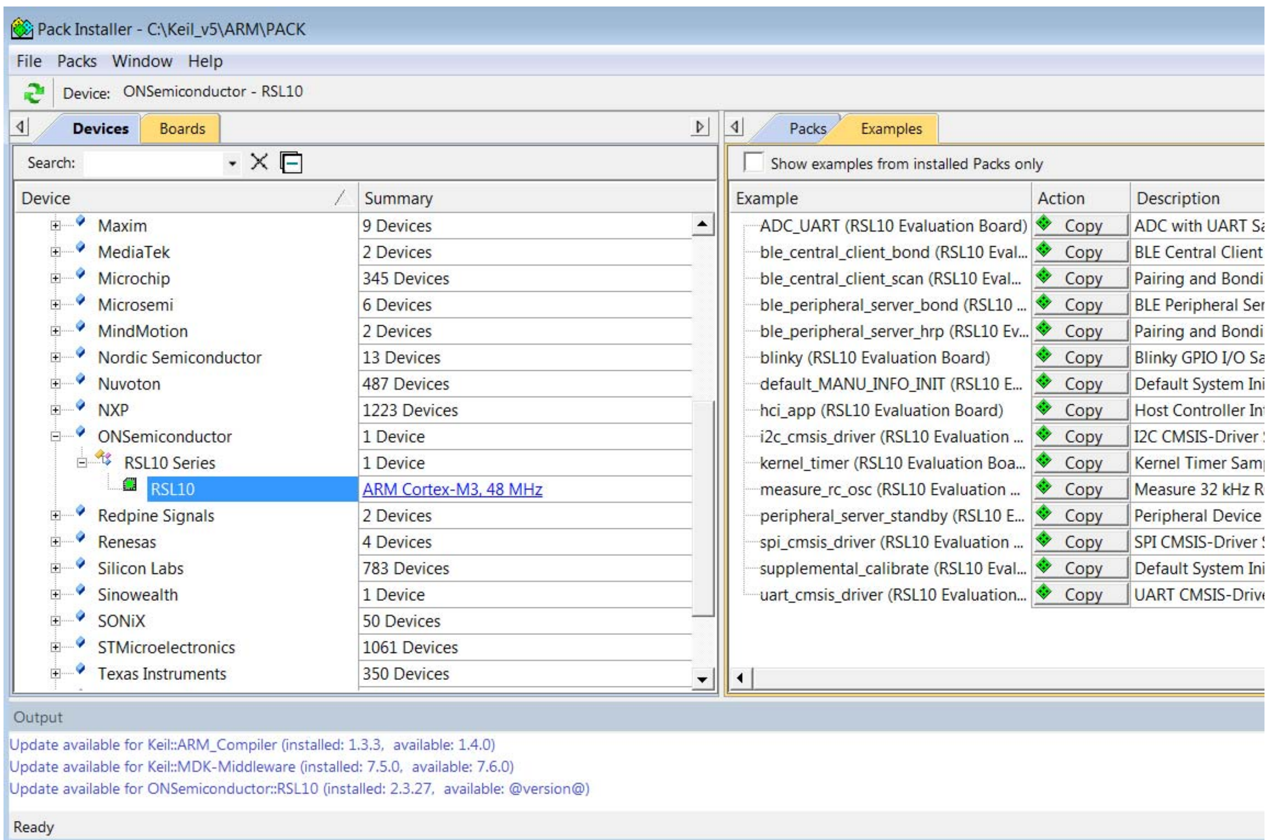

Figure 19. Pack Installer after RSL10 CMSIS-Pack is Installed in the Keil µVision IDE

4.3 Building Your First Sample Application with the Keil uVision 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.

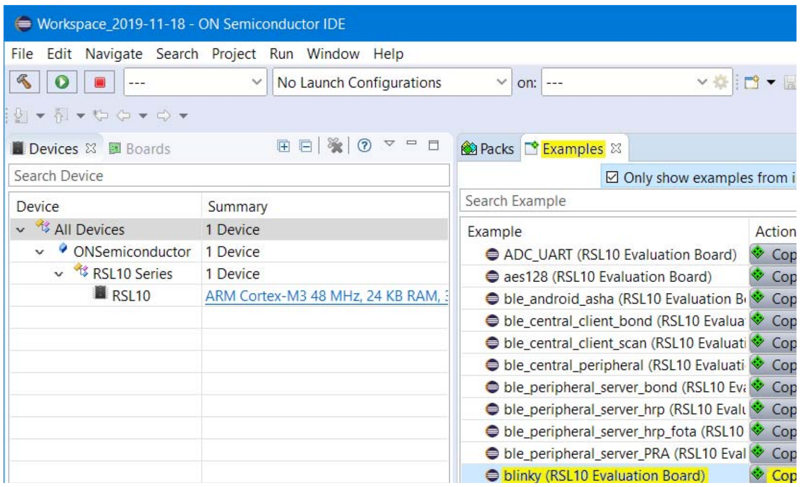

Figure 20. Pack Manager Perspective: Examples Tab

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).

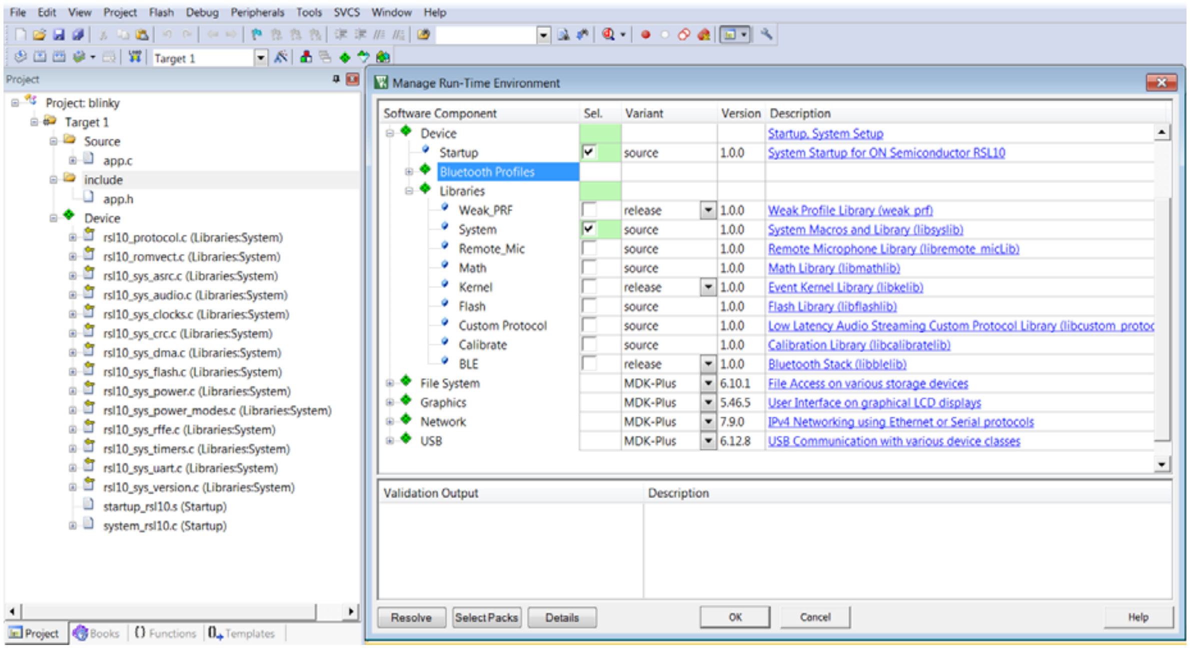

Figure 21. RTE Configuration for the Blinky Example Project in the Keil µVision IDE

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.

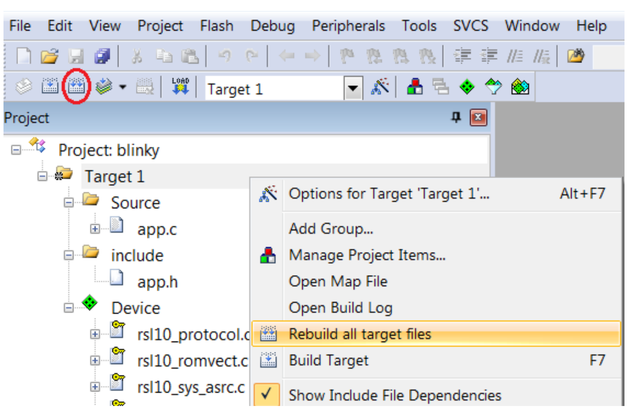

Figure 22. Starting to Build a Project in the Keil µVision IDE

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.

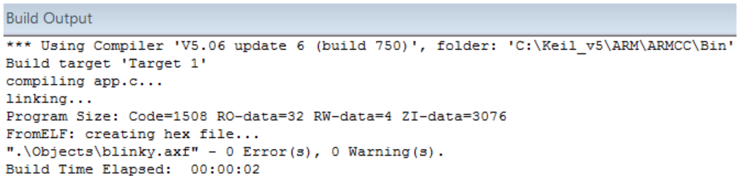

Figure 23. Example of Build Output

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.

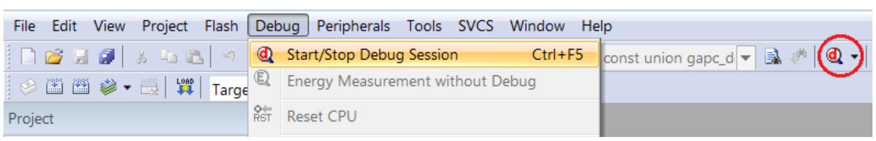

Figure 24. Start/Stop Debug Session Icon

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)`.

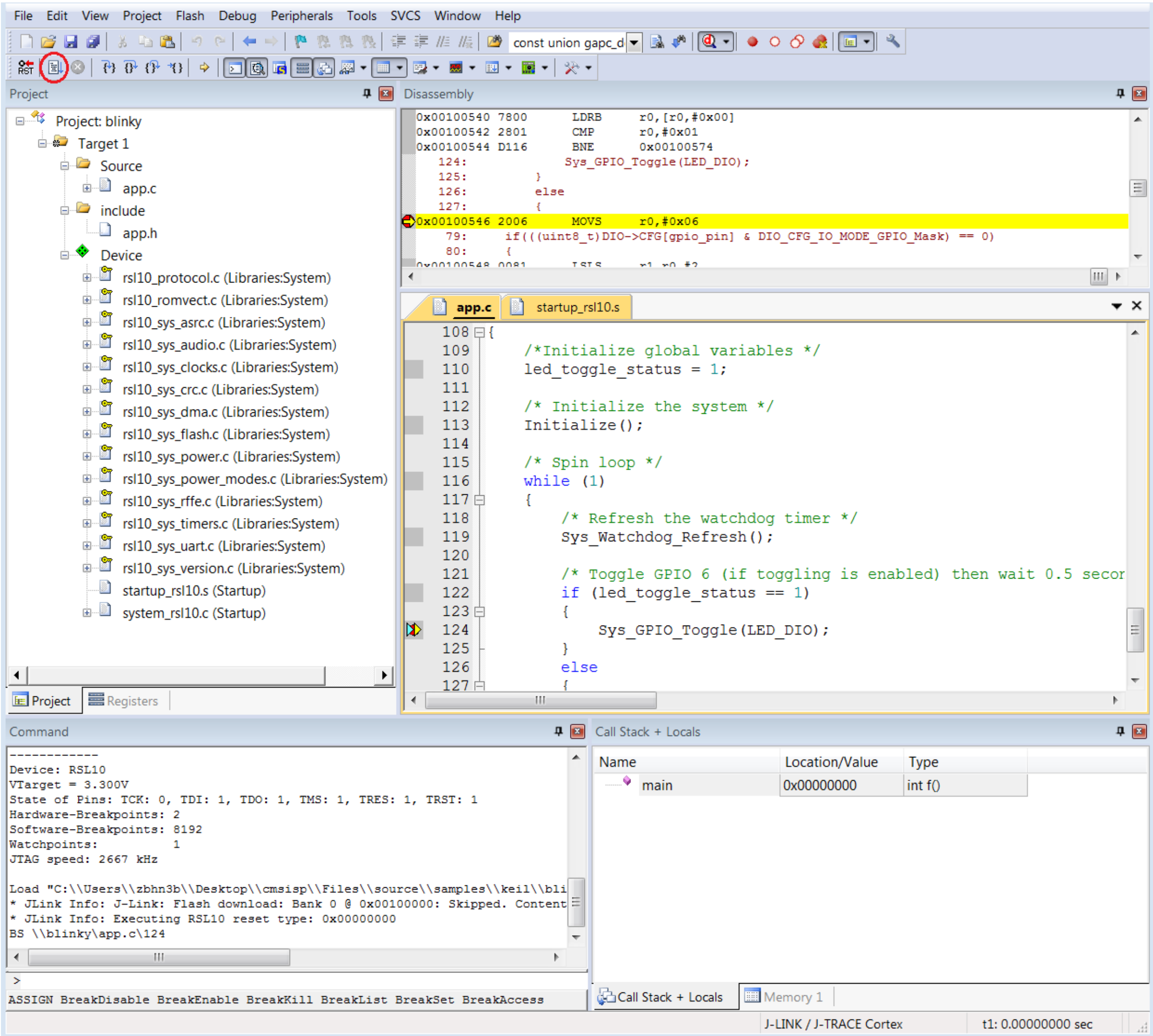

Figure 25. Debug Session in the Keil µVision IDE

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.

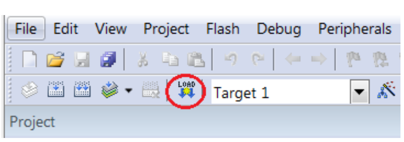

Figure 26. Download Button Not Supported for J-Link

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.

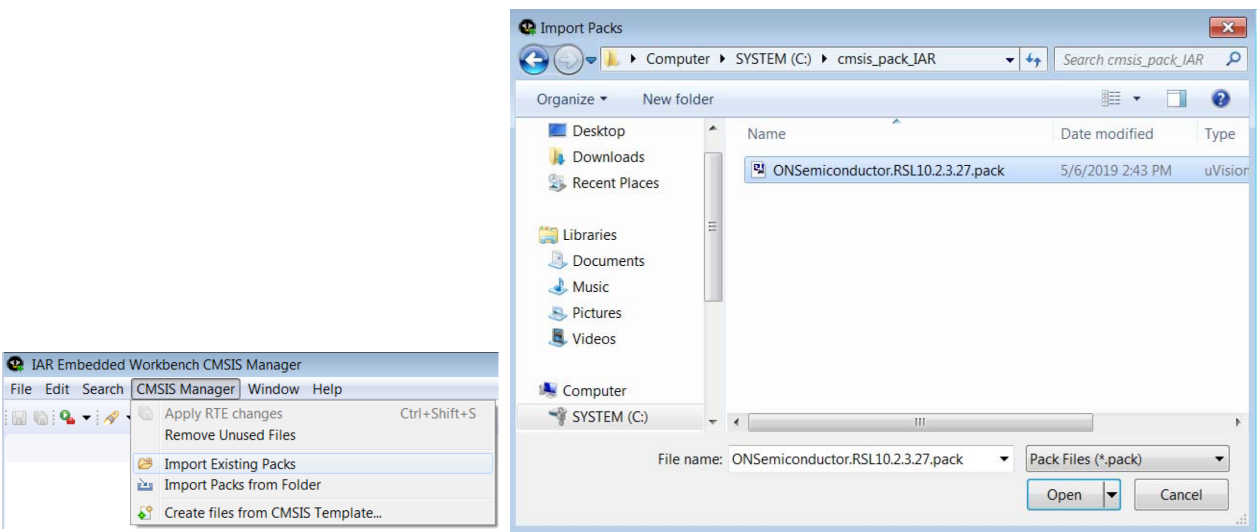

Figure 28. Installing the RSL10 CMSIS-Pack for the IAR Embedded Workbench IDE

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 proprieties.

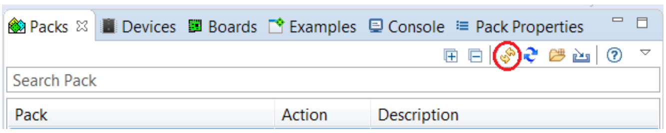

Figure 29. Refresh Pack after installation

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.

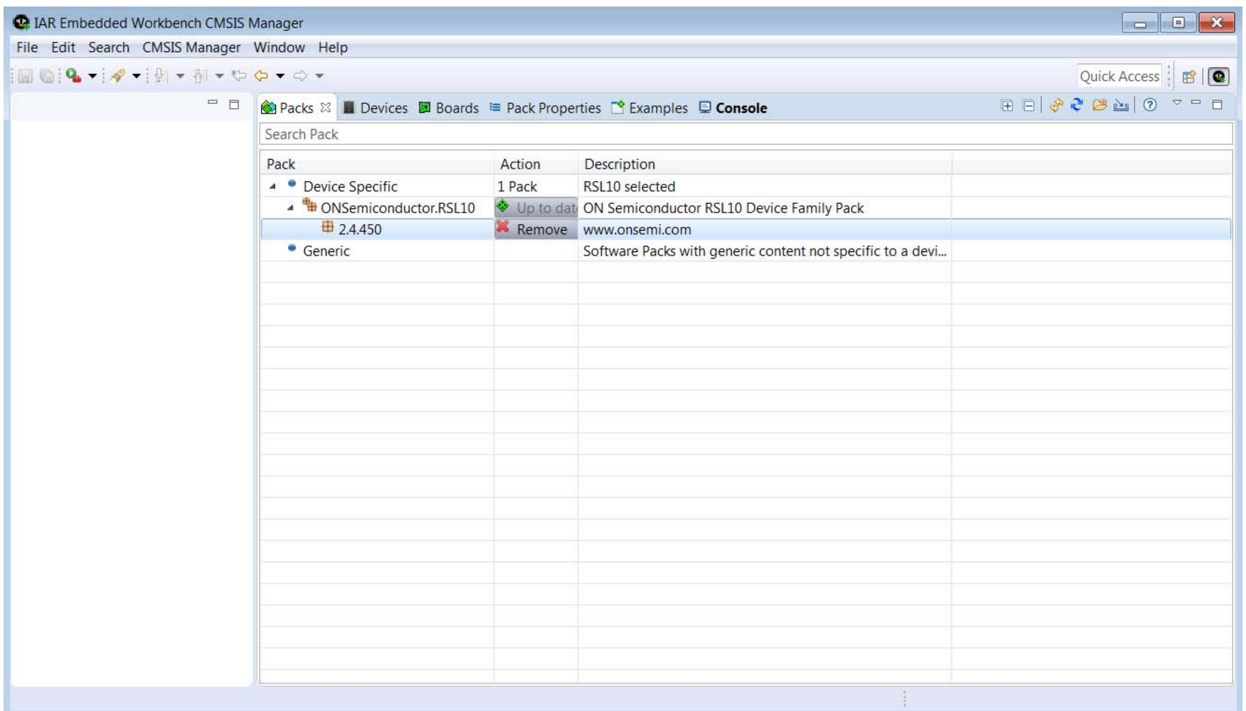

Figure 30. The IAR Embedded Workbench CMSIS Manager after RSL10 CMSIS-Pack is Installed

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.

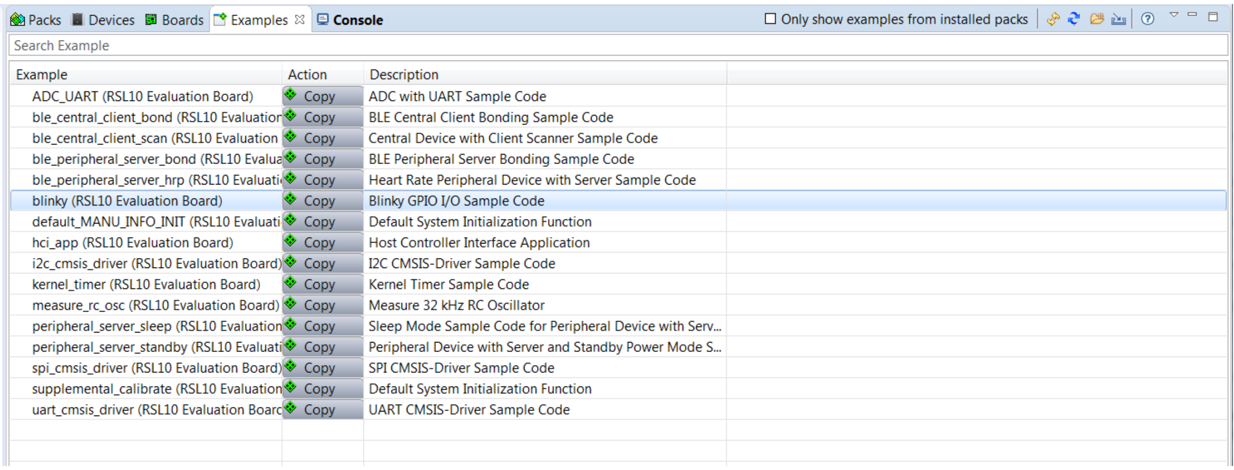

Figure 31. IAR Embedded Workbench CMSIS Manager: Examples Tab

Sample projects are preconfigured with release versions of RSL10 libraries, which are distributed as object files. For the IDE, System library (*libsyslib*) 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).

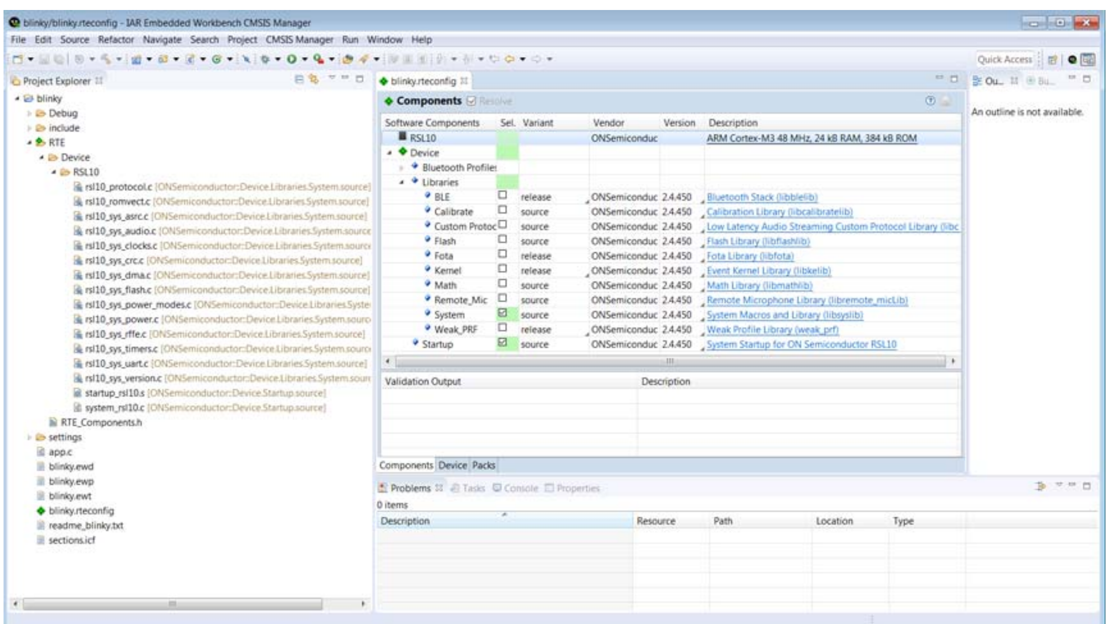

Figure 32. RTE Configuration for the Blinky Example Project in the IAR Embedded Workbench CMSIS Manager window

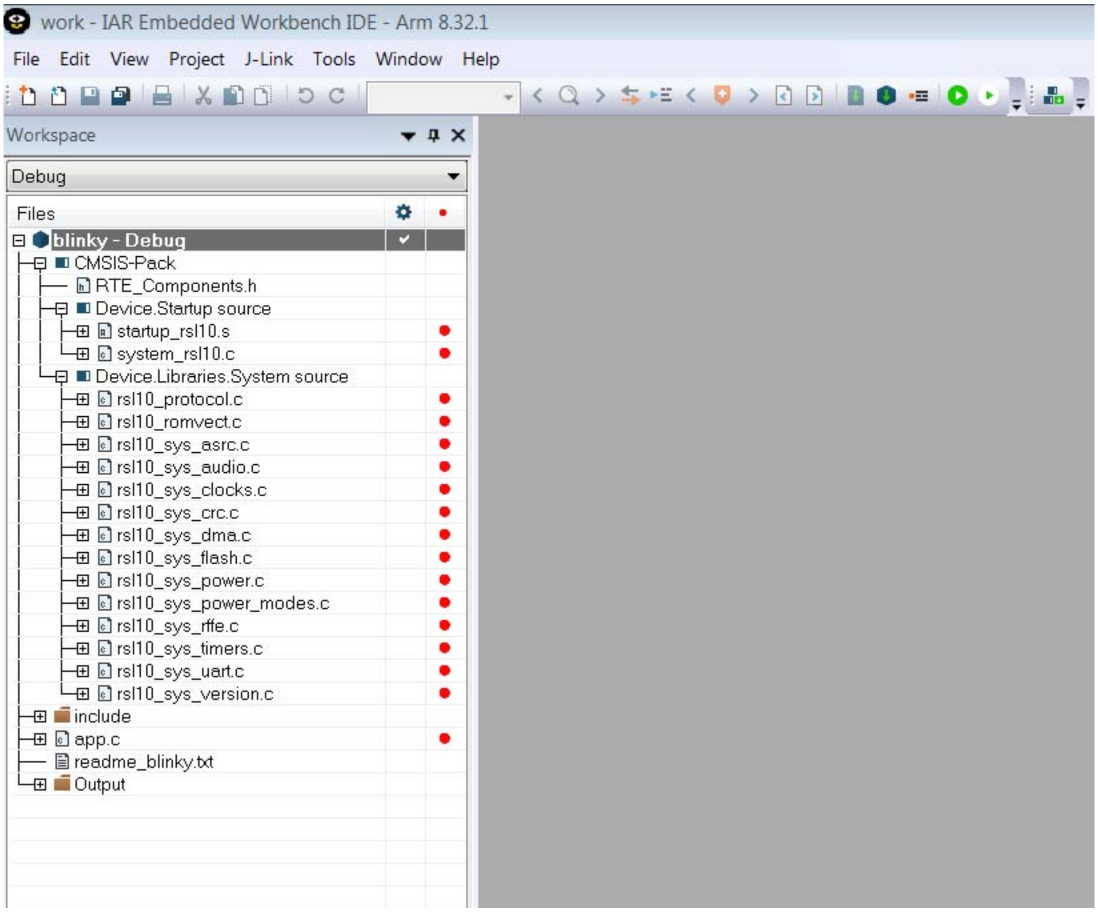

Figure 33. RTE Configuration for the Blinky Example Project in the IAR Embedded Workbench window

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.

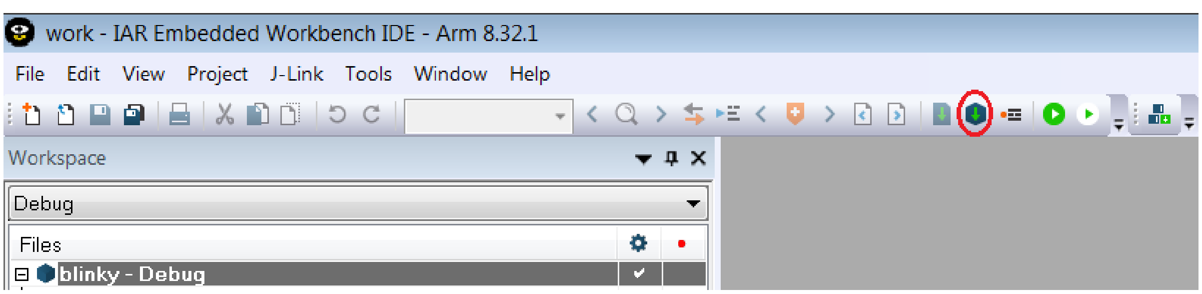

Figure 34. Starting to Build a Project in the IAR Embedded Workbench

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.

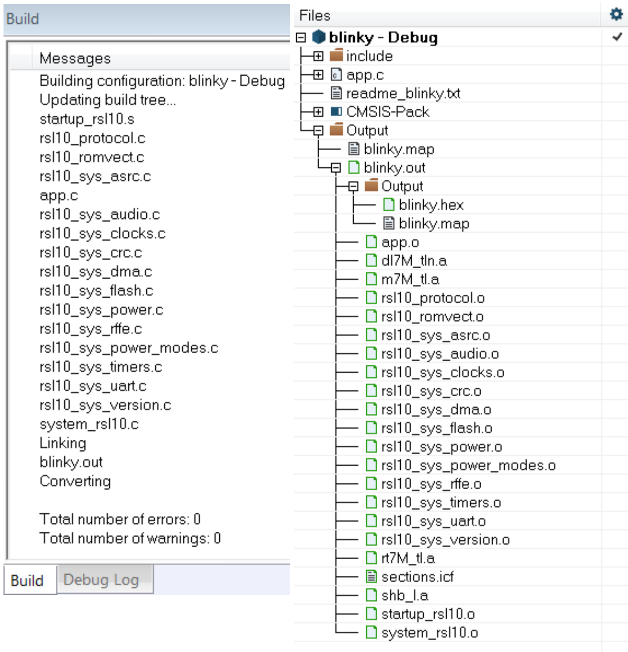

Figure 35. Example of Build Output

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

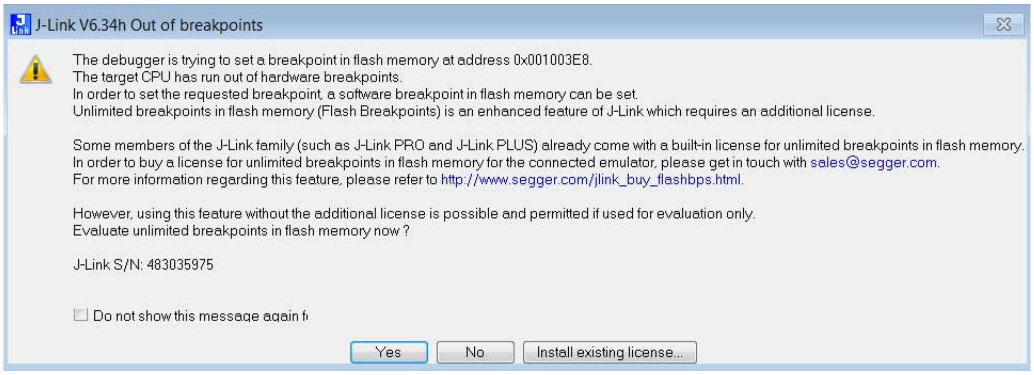

Figure 37. J-link “Out of breakpoints” pop-up dialog

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.

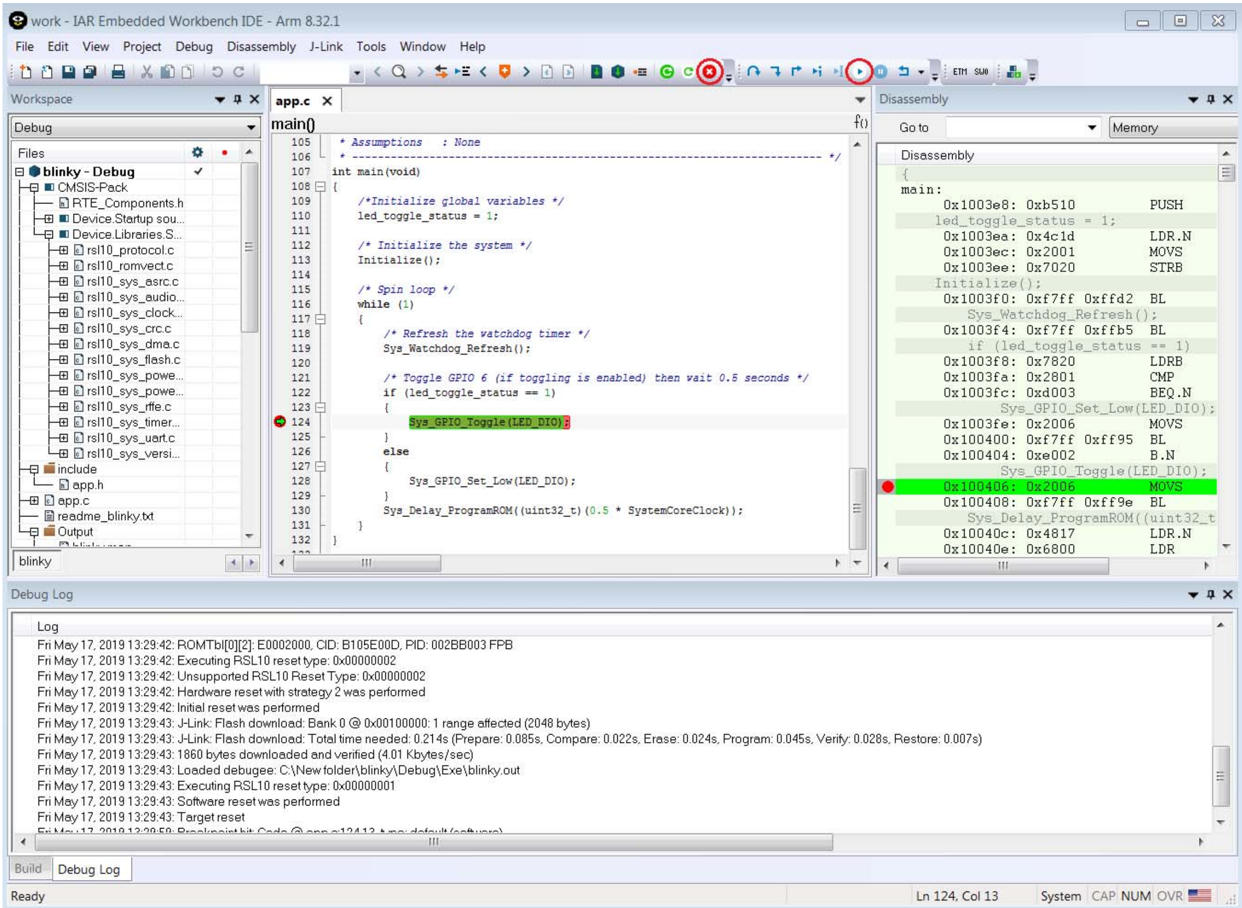

Figure 38. Debug Session in the IAR Embedded Workbench

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.

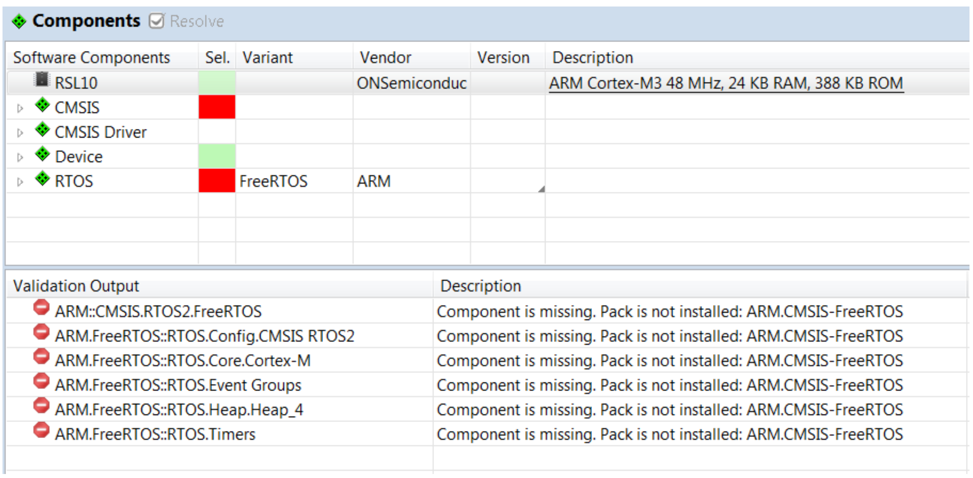

Figure 39. RTE Configuration Perspective Before Resolving Pack Dependencies

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).

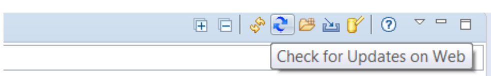

Figure 40. Check for Updates on Web Button

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

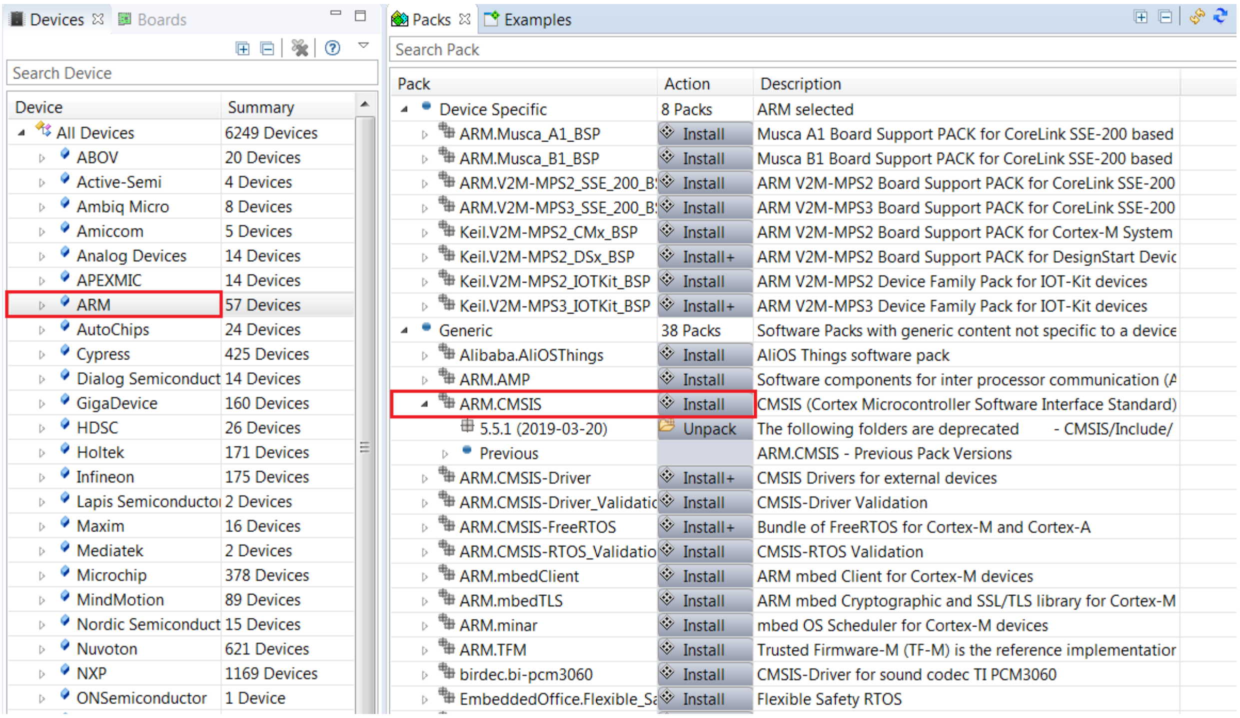

Figure 41. Installing the Arm CMSIS-Pack

2. 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.
3. Open the **.rteconfig* file; in the Packs tab, select the **Resolve Missing Packs** button (see Figure 42).

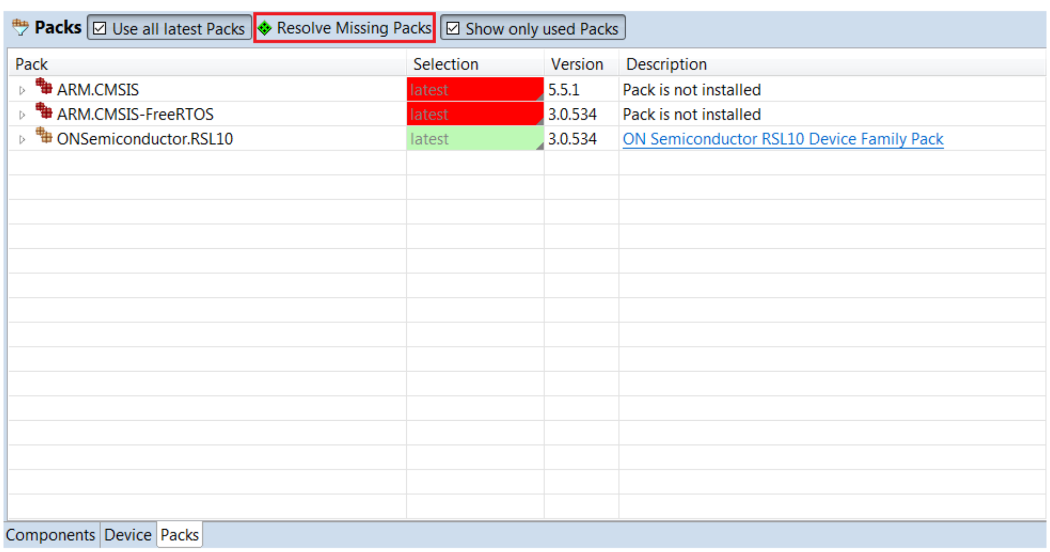

Figure 42. Resolve Missing Packs Icon

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.

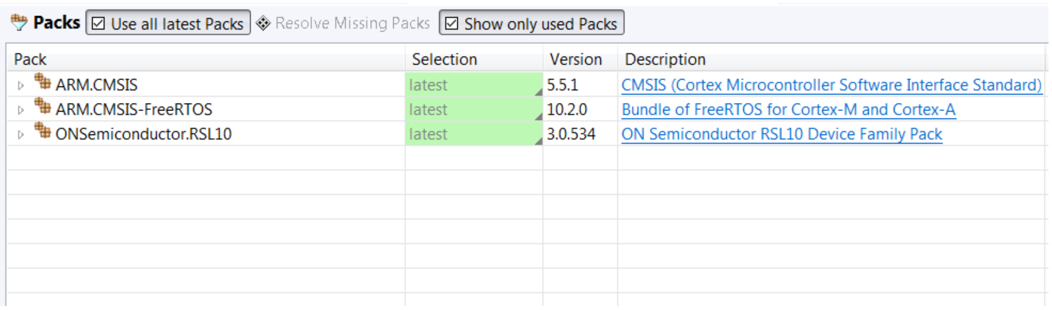

Figure 43. RTE Configuration Perspective After Resolving Pack Dependencies

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 for you 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:

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

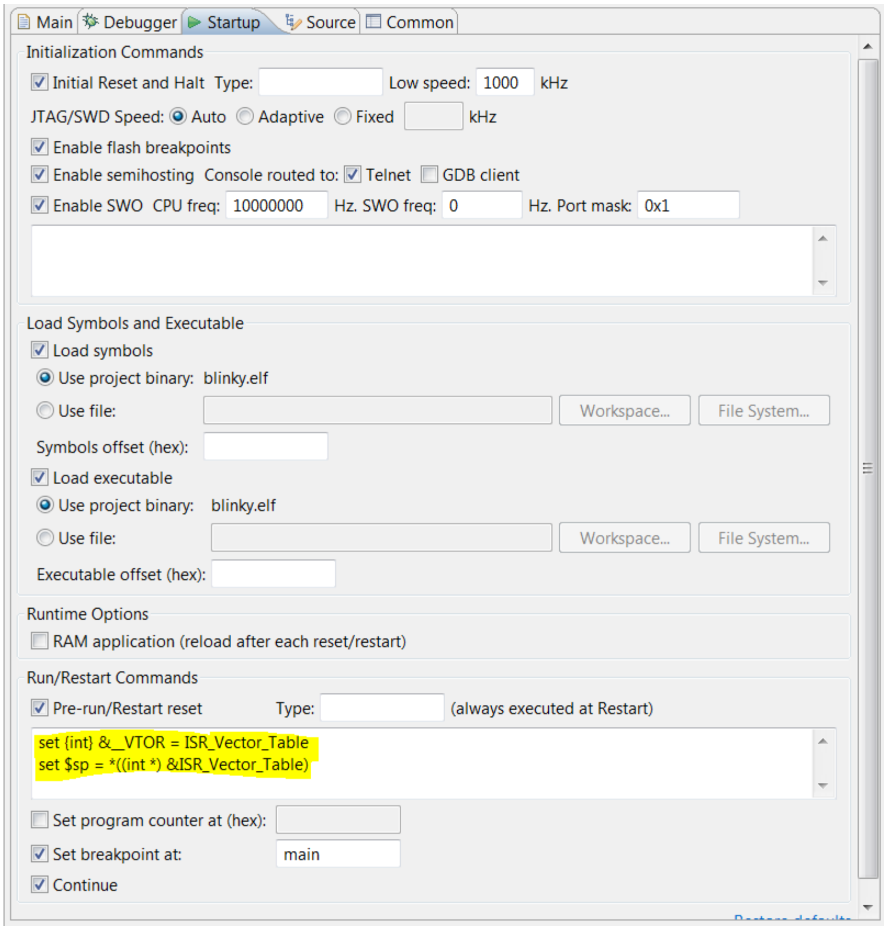

Figure 44. Setting Up a GDB Launch Configuration, Startup Tab

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.

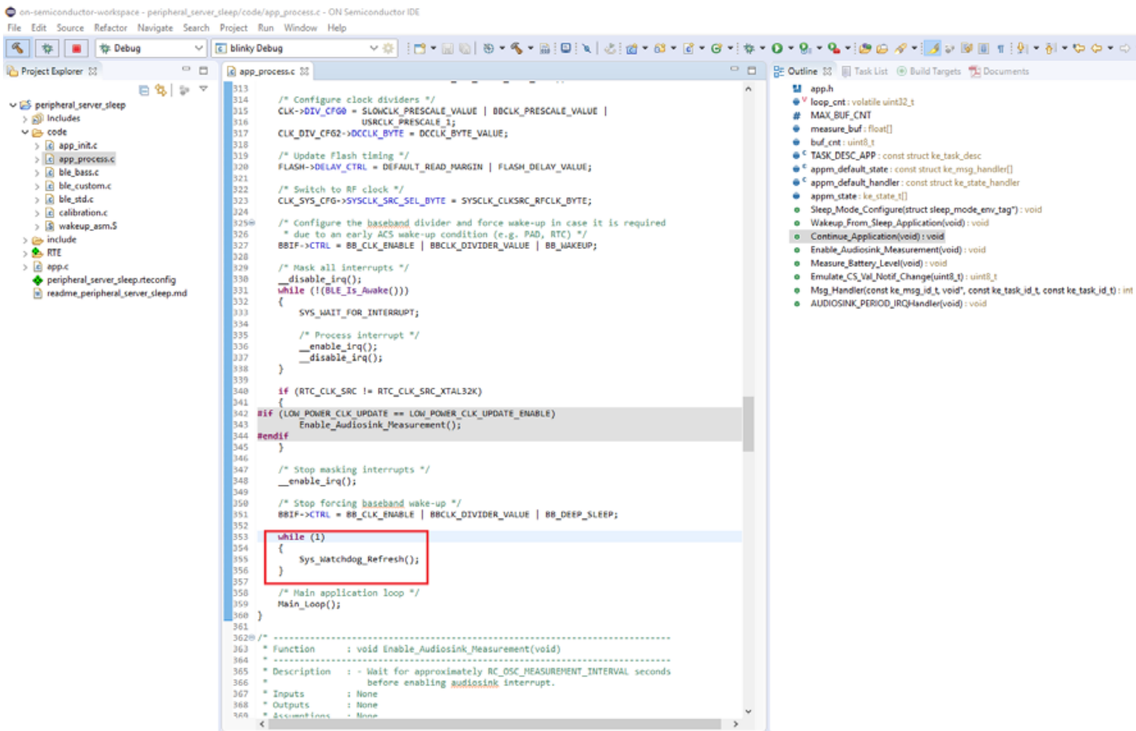

Figure 45. Continue_Application Function Perspective After Adding While Loop

3. Within the Project Explorer, right-click on the *.elf* file and select **Debug As** > **Debug Configurations**.
4. When the **Debug Configurations** dialog appears, create two debug sessions:
   a. Debug session that initiates restart and halts the target:
      i. Right-click on **GDB SEGGER J-Link Debugging** and select **New**. A new configuration appears under the **GDB SEGGER** heading, with new configuration details in the right panel.
      ii. Adjust the displayed values for your configuration and click on **Apply** (see Figure 46, and Figure 47 on page 39).

NOTE: If you are having trouble downloading firmware to the device, in addition to using DIO12, you can also perform the software recovery by setting the **Reset Type** to 1 in the **Debug** session configuration (see Figure 46). The default **Reset Type** is 0, which only resets the Arm Cortex-M3 core while leaving the device/peripherals in a state where J-Link can't reconnect. Setting the **Reset Type** to 1 ensures that not only is the Arm Cortex-M3 core reset, but so are all the peripherals. If this does not work, see Section 7.4.1, “Downloading Firmware in Sleep Mode” on page 43.

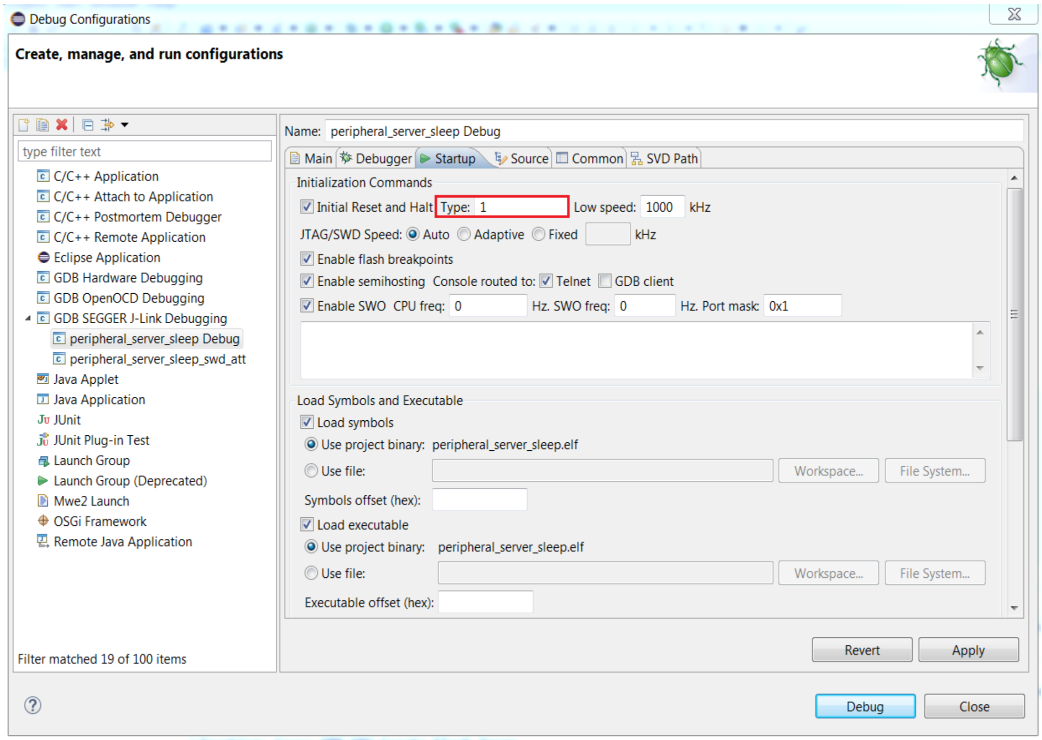

Figure 46. Setting Reset Type in the Debug Configuration Session

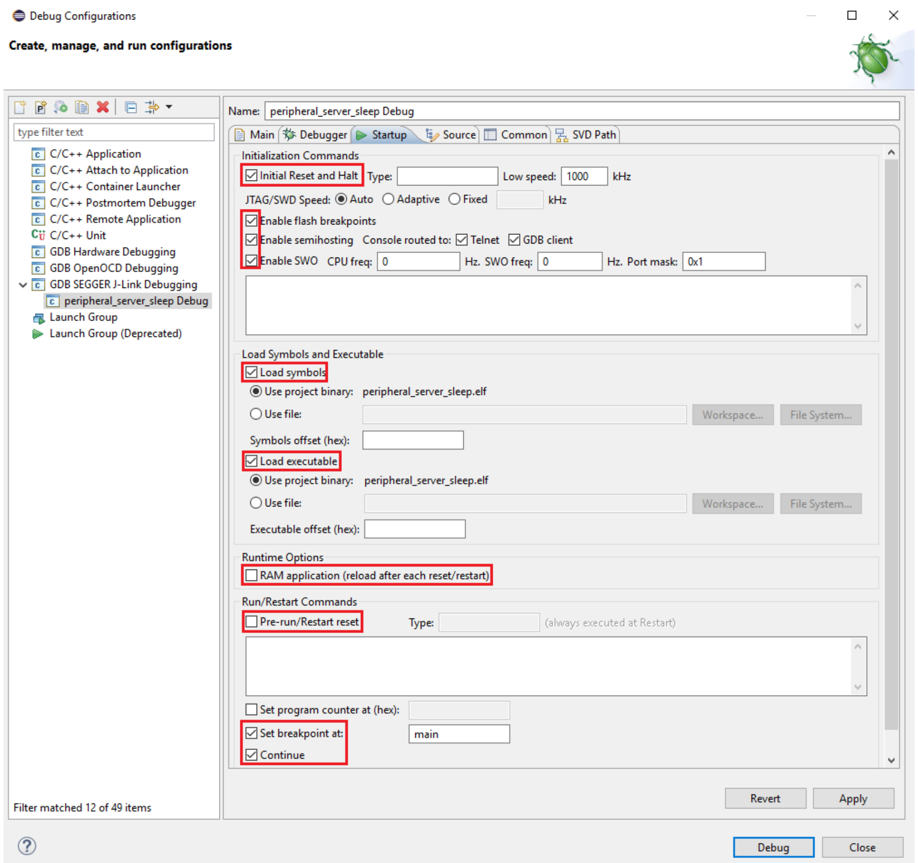

Figure 47. Startup Tab: Debug Session that Initiates Restart

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).

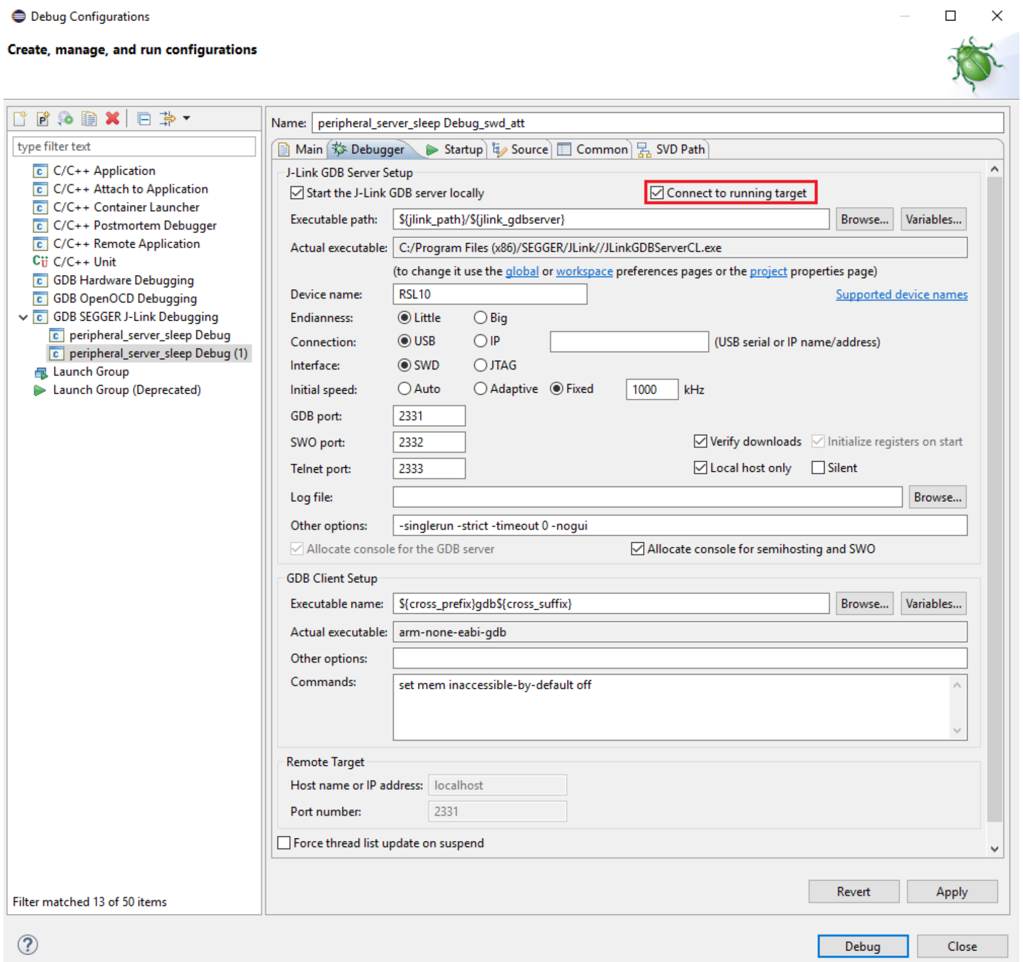

Figure 48. Debugger Tab: Debug Session that Connects to the Running Target

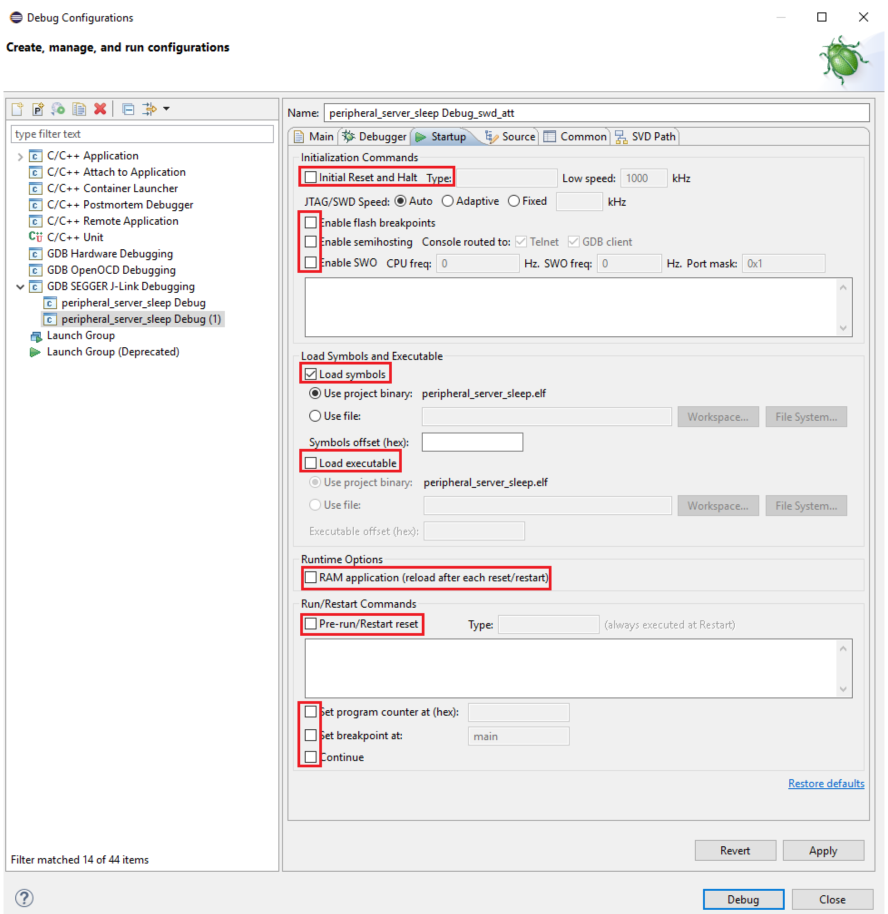

Figure 49. Startup Tab: Debug Session that Connects to the Running Target

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

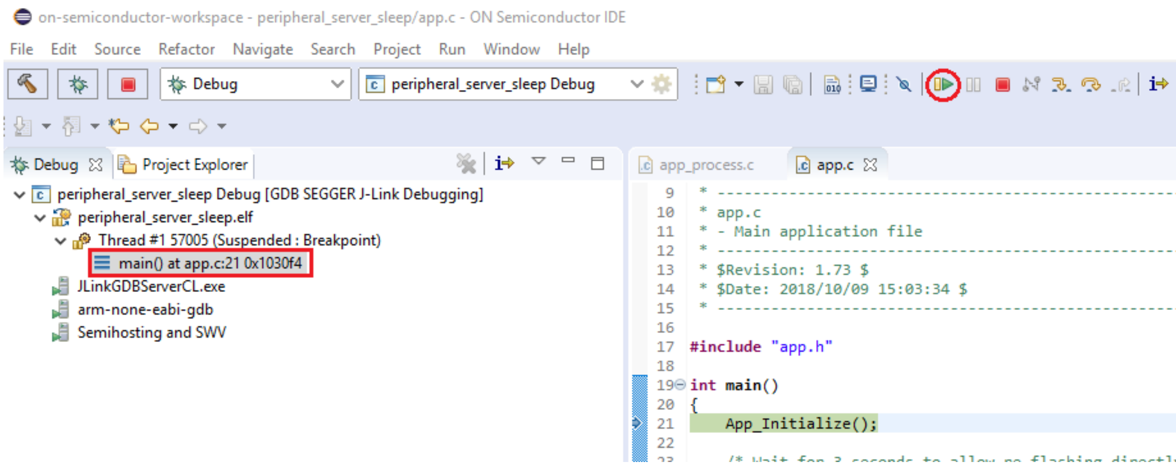

Figure 50. First Debug Session Perspective Before Starting Execution

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).

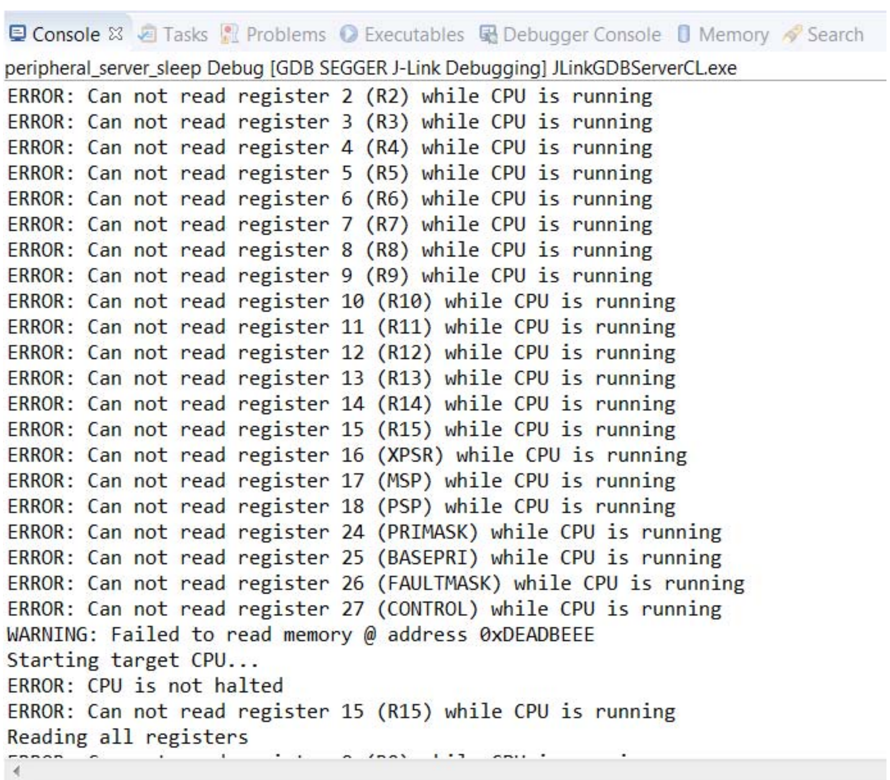

Figure 51. Debug Session Perspective when Debug Connection is Lost

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

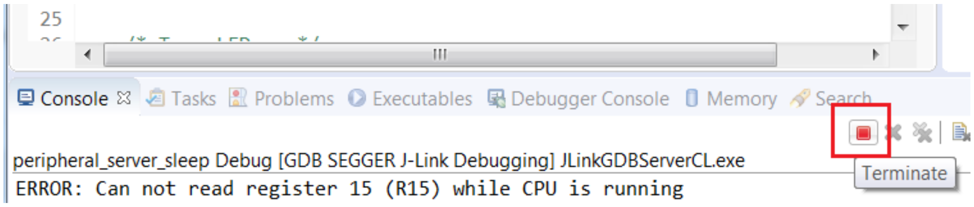

Figure 52. Terminate Targets Icon

8. 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.

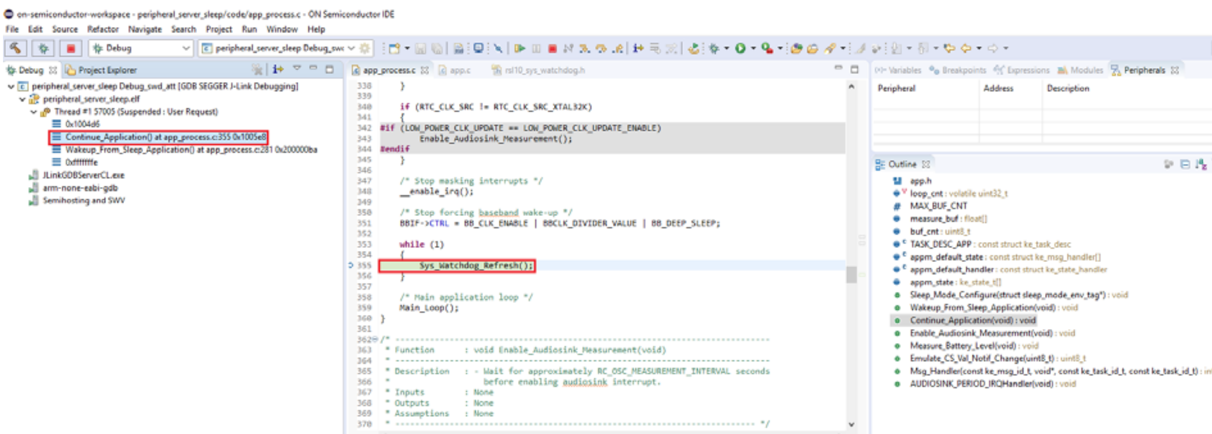

Figure 53. Second Debug Session Perspective After Connecting to the Running Target

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:

1. Connect DIO12 to ground.
2. Press the RESET button (this restarts the application, which pauses at the start of its initialization routine).
3. 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	
configuration	J-Link flash loader files.	
documentation	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.	
images	Contains evaluation board pictures.	
include	Include files for the firmware components and libraries. Projects can point to this directory and sub-directories when including firmware header files.	
lib	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.	
source	*firmware*	The source of the provided support libraries.
	samples/rslx (for ON Semiconductor IDE) *samples/uv* (for Keil IDE) *samples/iar* (for IAR IDE)	Sample code sources as ready-to-build projects.
svd	Contains the System View Description file used in the registers view during debugging.	
ONSemiconductor.RSL10.pdsc	A file that describes the dependencies to devices, processor, toolchains and other software components for the RSL10 CMSIS-Pack.	
PACK_REVISION	Identifies the revision of the RSL10 CMSIS-Pack.	
Software_Use_Agreement.rtf	ON Semiconductor license agreement.	

Table 2. Installed Folders and Files - ON Semiconductor IDE

Folder	Contents
arm_tools	The Arm Toolchain is installed here.
eclipse	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.
*jre**	The included JAVA runtime environment.
ide.exe	Executable that opens the ON Semiconductor IDE.
REVISION	Identifies the revision of the ON Semiconductor IDE.
Software_Use_Agreement.rtf	ON Semiconductor license agreement.
ThirdPartyLicenses.txt	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.

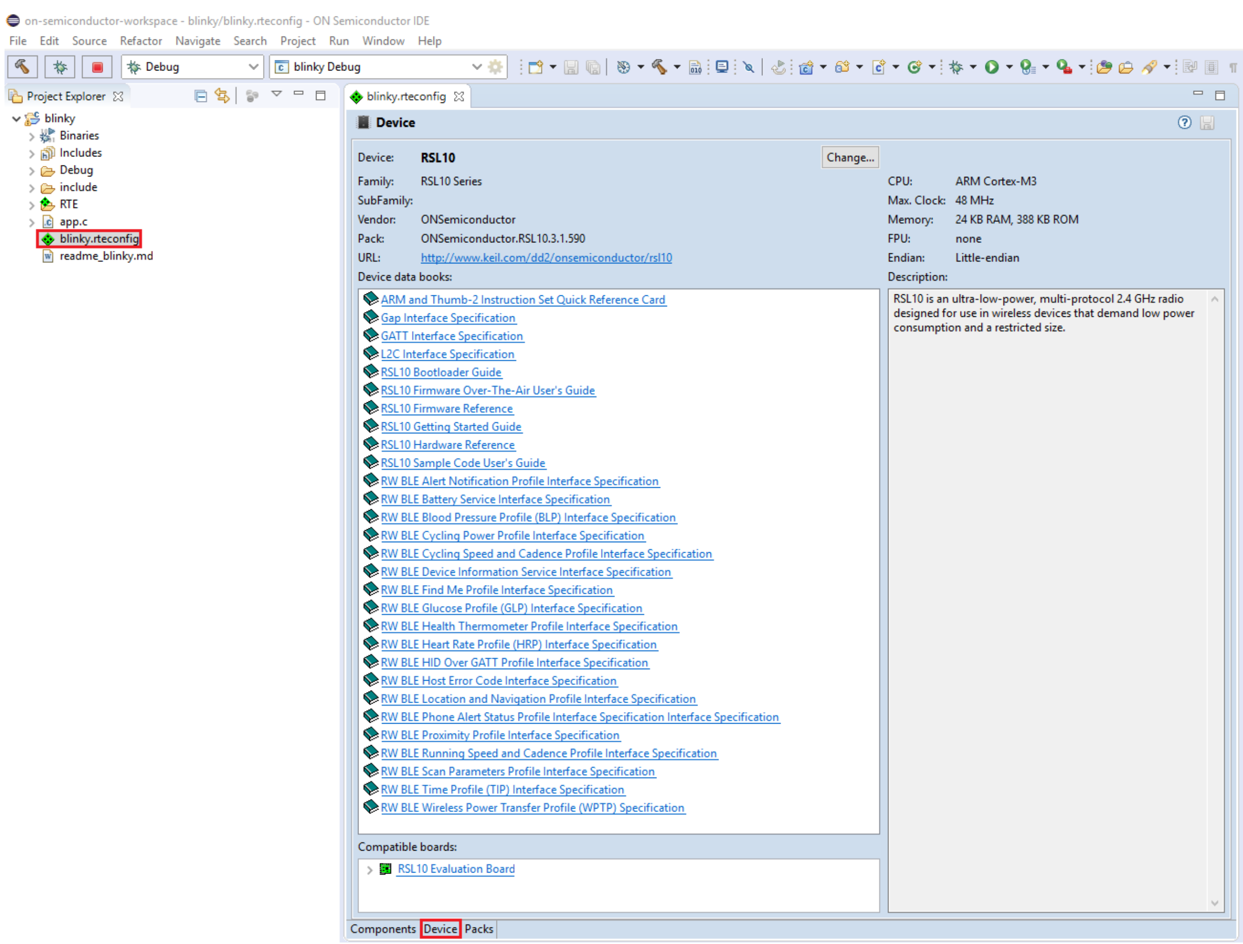

Figure 54. Accessing RSL10 Documentation from the ON Semiconductor IDE

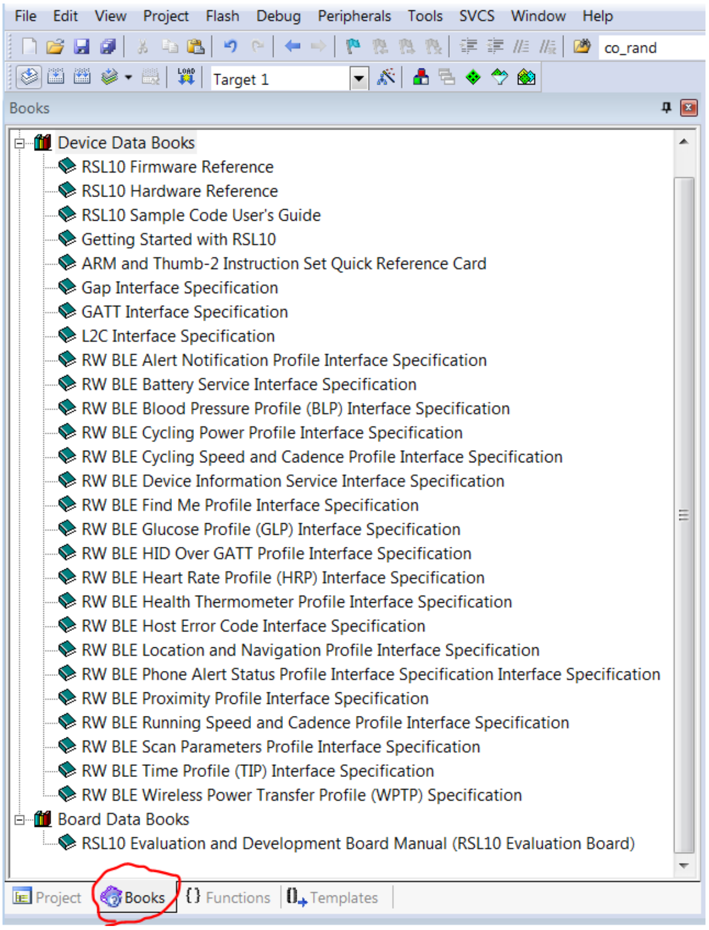

Figure 55. Accessing RSL10 Documentation from the Keil µVision IDE

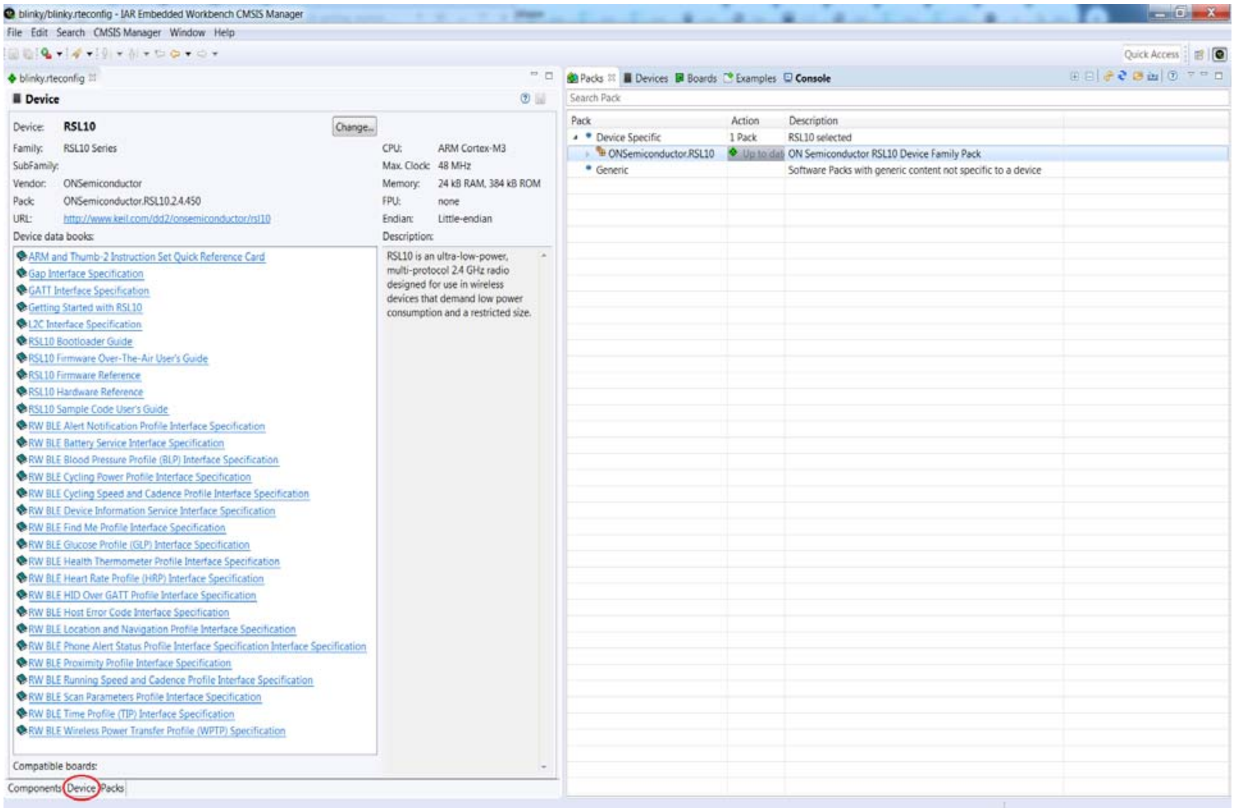

Figure 56. Accessing RSL10 documentation from the IAR Embedded Workbench

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 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.

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.

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