Getting Started with RSL10

AND9697/D
Mar. 2018, Rev. P1

ON Semiconductor®

Table of Contents

CHAPTER 1

Introduction

1.1 Purpose

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 advanced wireless features, with minimal system size and maximized battery life. The RSL10 Software Development Kit (SDK) includes firmware, software, example projects, and documentation; which are all incorporated into an Integrated Development Environment (IDE) to support the development of your applications.

Starting at version 2.0, the RSL10 IDE has been upgraded to the Eclipse Oxygen version with support for CMSIS-Packs. The new environment is backward compatible with projects from RSL10 SDK 1.4, so the migration to the CMSIS-Pack standard is not mandatory.

The RSL10 CMSIS-Pack is provided separately, and can be optionally imported into the RSL10 IDE 2.0. By using the CMSIS-Pack, you can enjoy new features:

- Future releases of the RSL10 CMSIS-Pack can be imported side-by-side on the RSL10 SDK 2.0, not requiring the re-installation of the full SDK.
- Documentation can be accessed from inside the IDE.
- Sample projects can be imported more easily, through the pack manager.
- The setup of new projects is simplified by the RTE configuration wizard, where RSL10 software components, such as libraries and source code, can be selected.

This document helps you to get started with the RSL10 SDK. It guides you through the process of connecting your hardware, installing the software with the option of applying the CMSIS-Pack, configuring your environment, and building and debugging your first RSL10 application.

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, file and path names, 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.

1.4 Manual Organization

Getting Started with RSL10 contains the following chapters and appendices:

- **Chapter 1: Introduction** describes the purpose of this guide and how the book is organized, and gives a list of suggested reading for more information.
- **Chapter 2: Connecting the Hardware** explains how to connect the RSL10 Evaluation and Development Board to a computer running Windows®.
- **Chapter 3: Installing the Software** describes the steps for installing the RSL10 SDK, including the steps for applying the CMSIS-Pack for Eclipse Oxygen.
- **Chapter 4: Building Your First Sample Application** shows how to import and build a sample application, with or without CMSIS-Pack, as a practical introduction to using the SDK.
- **Chapter 5: More Information** lists additional sources of information about using the RSL10 SDK, and additional tools.
- **Appendix A: Arm Toolchain Support** explains what to do if your IDE cannot find the correct toolchain because you have other versions of the GNU toolchain installed.

1.5 Further Reading

For more information, refer to the following documents:

- See Chapter 5, “More Information” on page 26 for a list of additional information sources.

CHAPTER 2

Connecting the Hardware

2.1 Prerequisites

The following items are needed before you can make connections:

- RSL10 Evaluation and Development Board and a USB to 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.

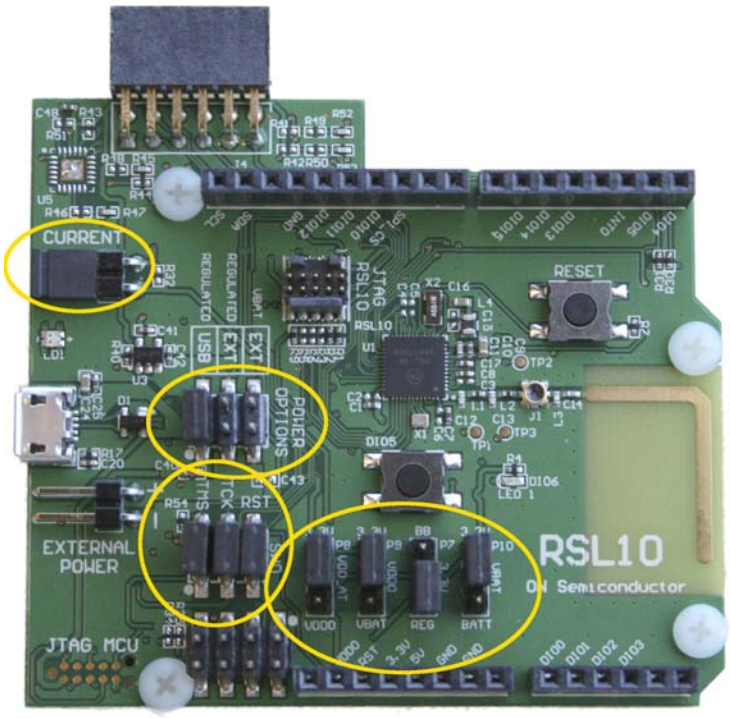

Figure 1. Evaluation and Development Board with Pins and Jumpers for Connection Highlighted

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.

2.2.1 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.

The serial number is on a sticker on the back of every board.

Another way to tell the difference is that LED1 blinks about 5 to 10 times per second with "Peripheral Device with Sleep Mode" loaded, and blinks about 2 times per second with "Peripheral Device with Server" loaded.

For more information about "Peripheral Device with Sleep Mode" or "Peripheral Device with Server", go to the *source/samples* folder and see the */peripheral_server_sleep/readme_peripheral_server_sleep.txt* file or */peripheral_server_sleep/readme_peripheral_server.txt* available in the RSL10 software development tools, or refer to the *RSL10 Sample Code User's Guide*. Alternatively, you can load a simpler sample application as described in Chapter 4, "Building Your First Sample Application" on page 11.

CHAPTER 3

Installing the Software

3.1 Prerequisite Software

- Install the 64-bit variant of the most recent Java version, choosing the Java Runtime Environment (JRE).
- Install J-Link version 6.20f or later. They are available from the SEGGER website: https://www.segger.com/downloads/jlink.

3.2 Installation Procedure

Install the RSL10 Software Development Kit (SDK) by running *RSL10_SDK_Installer.msi*. The RSL10 software development tools are installed in this location by default: *C:\Program Files (x86)\ON Semiconductor\RSL10 SDK*. If you have a previous RSL10 SDK version installed:

1. Uninstall it.
2. Check if the RSL10 SDK folder is still there; if it is, delete it.
3. Install your new RSL10 SDK.

The release version and build number are stored in the *REVISION* text file at the root of the installed RSL10 SDK software development tools.

If you are going to work in this environment, see Chapter 4, “Building Your First Sample Application” on page 11 for the next steps. To work in a CMSIS-Pack environment, see Section 3.3, “Optional RSL10 CMSIS-Pack Installation Procedure”.

3.3 Optional RSL10 CMSIS-Pack Installation Procedure

If you want to take advantage of supporting code written in different future releases of the RSL10 SDK, you will want to install the RSL10 CMSIS-Pack. It will allow you to develop and debug code written in different releases of RSL10 starting with Release 2.0 and going forward.

NOTE: You must have done the procedure in Section 3.2, “Installation Procedure” first.

1. Open your IDE and choose the desired location for your new workspace — for example, *c:\workspace* — and click **OK**.
2. Once the IDE is open, navigate to **Window** > **Preferences**. On the left panel, select **CMSIS Packs**. In the **CMSIS Pack root folder** field, type/choose a location where you would like your CMSIS-Packs to be installed — for example, *c:\cmsis_packs* — and click **Apply and Close** (Figure 2).

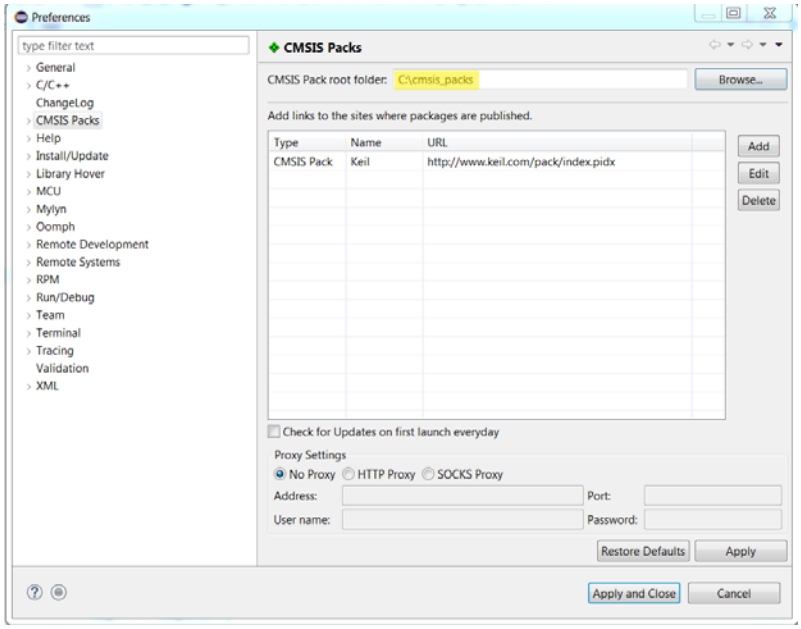

Figure 2. Configuring the CMSIS-Pack root folder

3. On the top right corner, click on the **Open Perspective** icon, select **CMSIS Pack Manager**, and click **OK** (see Figure 3).

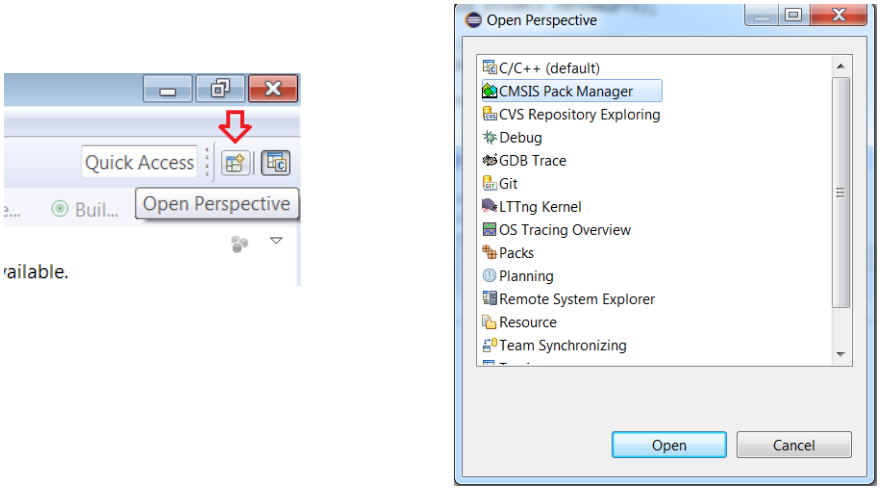

Figure 3. Opening the CMSIS-Pack Manager Perspective

4. Click on the **Import existing packs** icon, select your pack file *ONSemiconductor.RSL10.2.0.0.pack*, and click **Open** (see Figure 4). The pack can be found in the RSL10 Release 2.0 zip file, *RSL10_SDK_2.0.zip*.

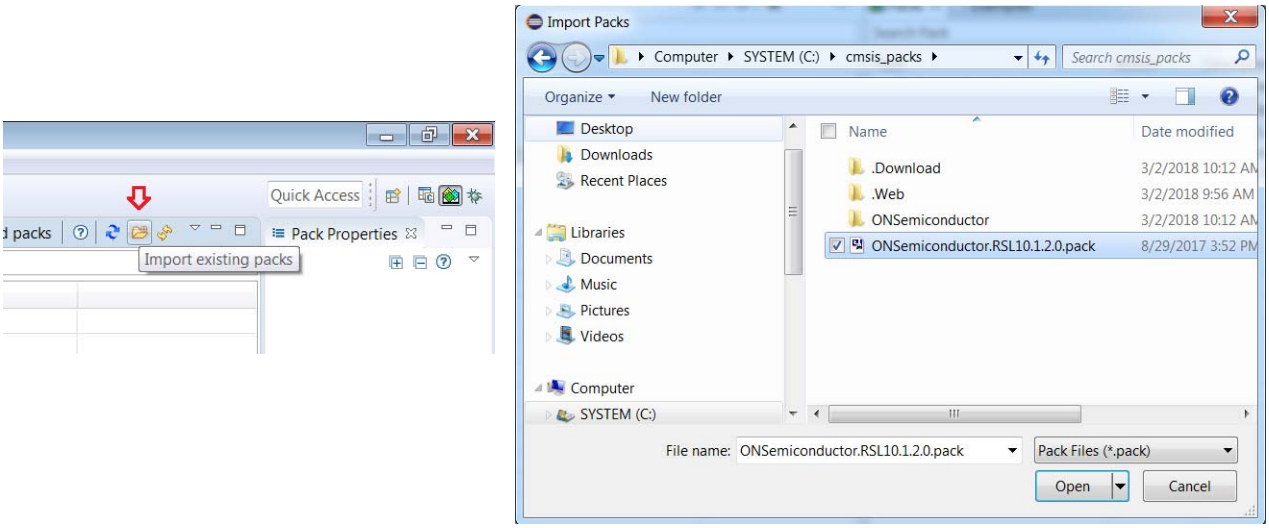

Figure 4. Installing the RSL10 CMSIS-Pack

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. After installation, click on the **Reload Packs** icon (see Figure 5) to update your list of installed packs.

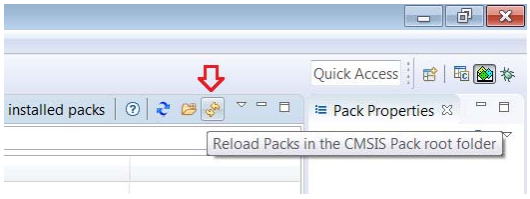

Figure 5. Reload Packs Icon

7. 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 6 on page 10 illustrates what the Pack Manager perspective looks like after installation.

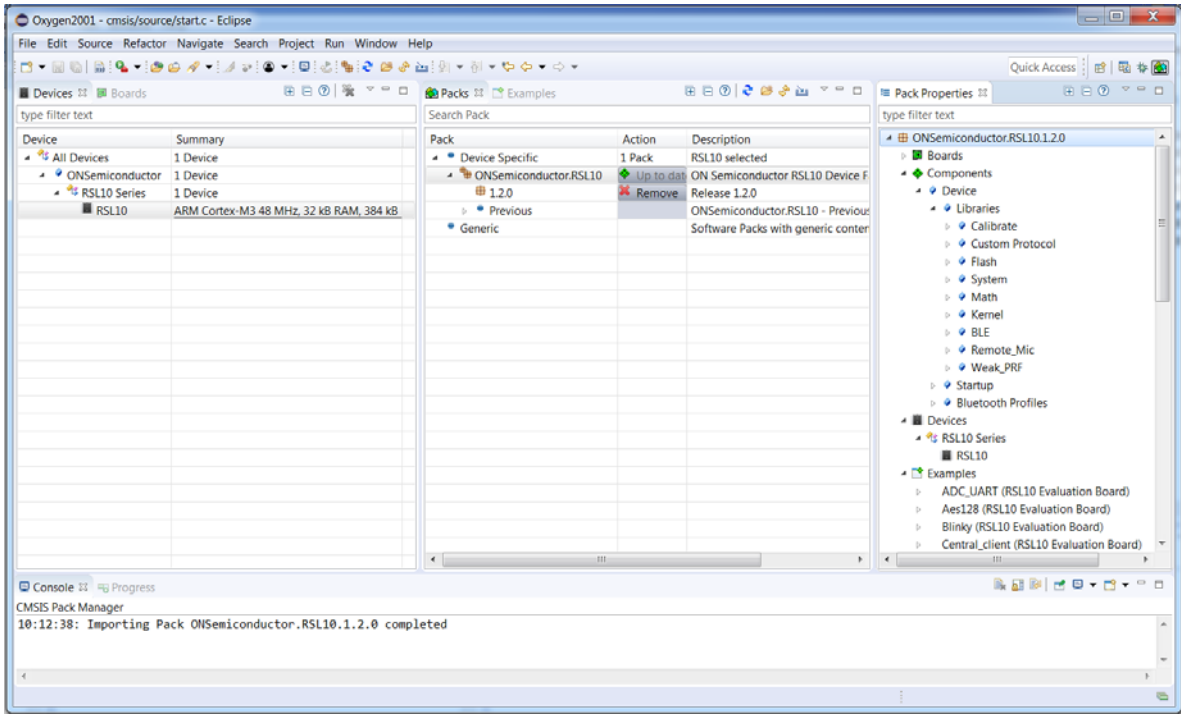

Figure 6. Pack Manager Perspective after RSL10 CMSIS-Pack is Installed

For the next steps of working with a sample application, see Chapter 4, “Building Your First Sample Application” on page 11.

CHAPTER 4

Building Your First Sample Application

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 following procedure assumes you have installed the SDK.

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

4.1 Launching the IDE

To use the IDE for the first time, follow the steps below:

1. To start the IDE, go to the Windows Start menu, and select **ON Semiconductor** > **RSL10 SDK IDE**.
2. When you open the IDE for the first time, you are prompted to select a workspace for the session. The workspace is the work area for all your IDE projects.

IMPORTANT: Create a new workspace for your version of the RSL10 IDE. Re-using an existing workspace originally created with other Eclipse-based IDE may not be compatible.

The procedures in this chapter are written on the assumption that you are looking at the Workbench perspective.

4.2 Import the Sample Code

Depending on whether you installed the CMSIS-Pack or not, choose one of the procedures in the following subsections. If you have chosen to install the CMSIS-Pack, jump to Section 4.2.2, "Importing Sample Code With CMSIS-Pack" on page 13 to learn how to import code in that environment. Otherwise, continue to Section 4.2.1, "Importing Sample Code Without CMSIS-Pack".

4.2.1 Importing Sample Code Without CMSIS-Pack

To import the Simple GPIO I/O sample code (affectionately known as *blinky*):

1. Click on the **File** menu item and select **Import**. An **Import** dialog opens.
2. Select the **General** folder and click on **Existing Projects into Workspace**, as shown in Figure 7.

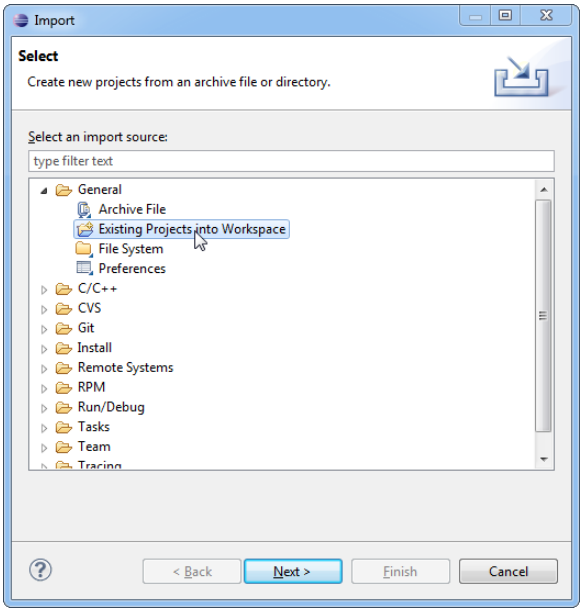

Figure 7. Importing an Existing Project

3. Click **Next**, and the **Import Projects** dialog appears. Select the **Copy projects into workspace** check box, as shown in Figure 8 on page 12.

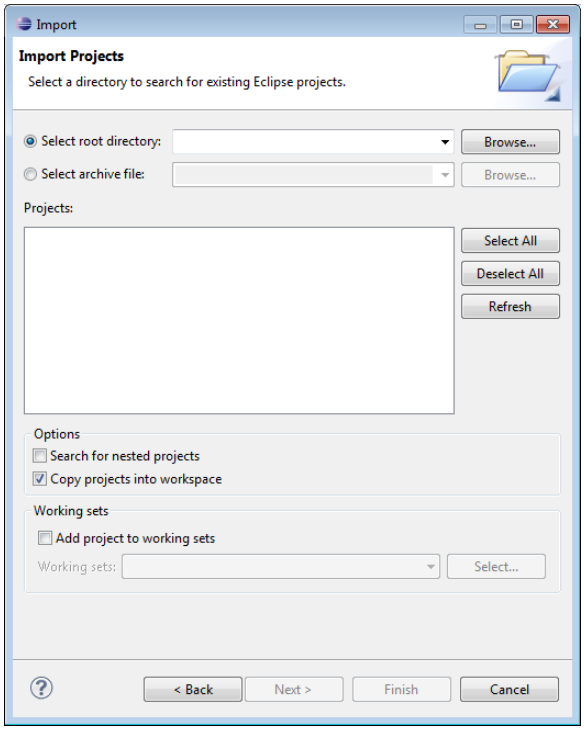

Figure 8. Import Projects Dialog Box

4. Browse to the folder *blinky*, as shown in Figure 9 on page 13, and click **OK**. The sample code is in the RSL10 installation directory (*C:\Program Files (x86)\ON Semiconductor\RSL10 SDK* by default) under *source\samples*

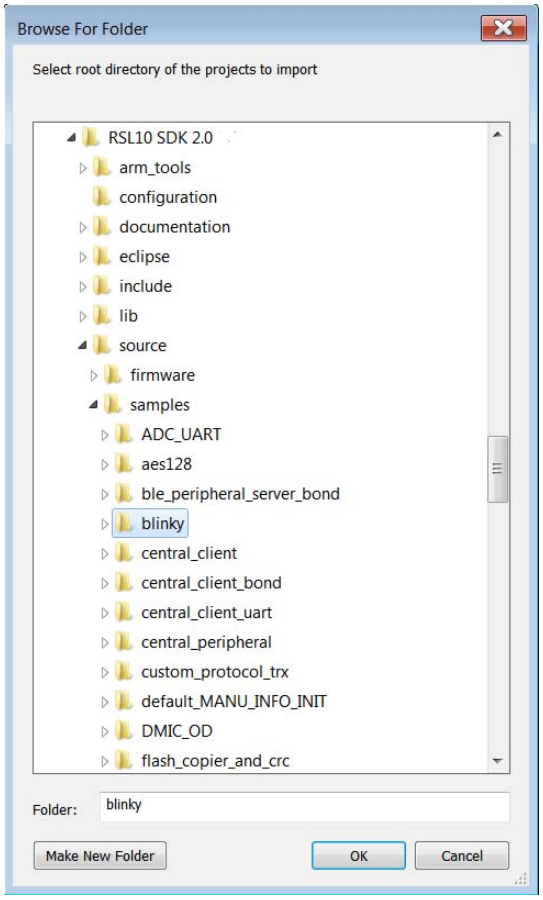

Figure 9. Select a Project to Import

5. On the **Import Projects** dialog, click **Finish**. This creates a new project in your workspace called *blinky*. The project appears in the left side of **Project Explorer**, as shown in Figure 10.

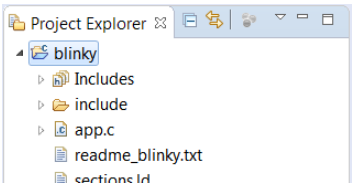

Figure 10. Newly Imported Project

4.2.2 Importing Sample Code With CMSIS-Pack

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 11 on page 14).

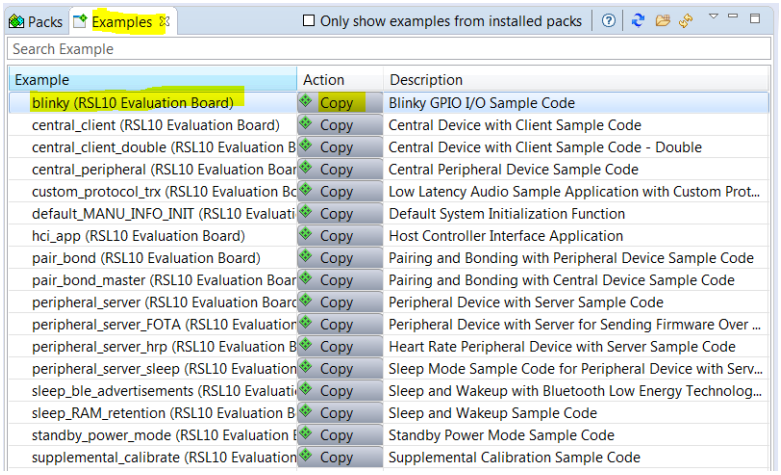

Figure 11. Pack Manager Perspective: Examples Tab

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 will see the **System library** (*libsyslib*) and the **Startup** (*libcmsis*) components selected for *blinky* (see Figure 12 on page 15).

NOTE: Sample projects are preconfigured with *Release* versions of RSL10 libraries, which are distributed as object files. In the RTE configuration, you can switch to the *Source* variant to include the source code of the library directly into your project (see Figure 12 on page 15).

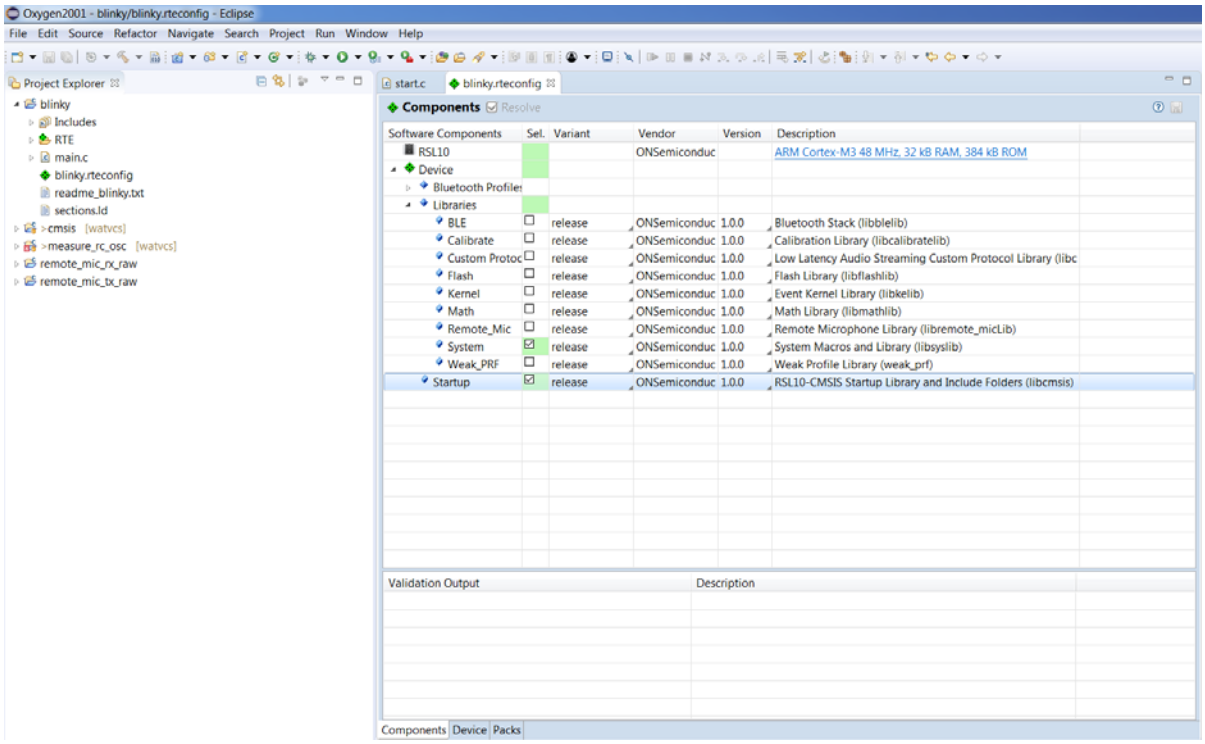

Figure 12. RTE Configuration For the Blinky Example Project

4.3 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 hammer icon shown in Figure 13 on page 16.

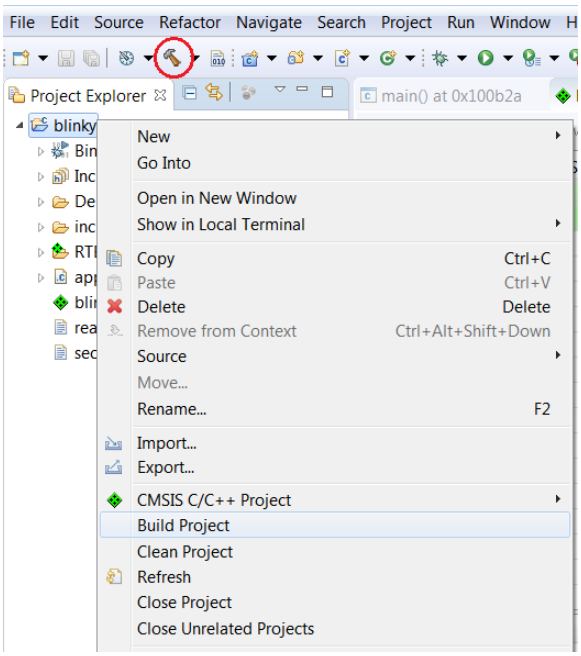

Figure 13. Starting to Build a Project

2. When the build is running, the output of the build is shown in the Eclipse C/C++ Development Tooling (CDT) Build Console, as illustrated in Figure 14.

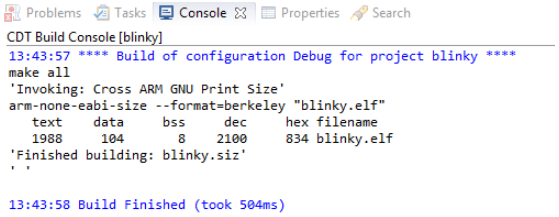

Figure 14. Example of Build Output

3. The key resulting output in Project Explorer 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 15 on page 17.

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.

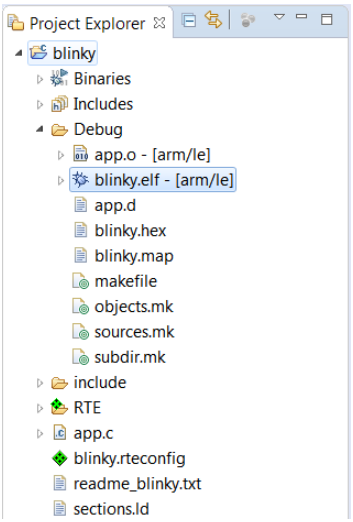

Figure 15. Output Files from Building a Sample Project

NOTE: If the IDE has trouble finding the GNU toolchain, it might be caused by having other GNU toolchains installed. See Appendix A, “Arm Toolchain Support” on page 29 for more information.

4.4 Debugging the Sample Code

4.4.1 Preparing J-Link for Debugging

Before you can debug with J-Link, configure the location of the J-Link GDB Server:

1. Under **Window > Preferences** go to **Run/Debug** and select **String Substitutions**.
2. Make sure the *Jlink_gdbserver* and *jlink_path* resemble the image in Figure 16.

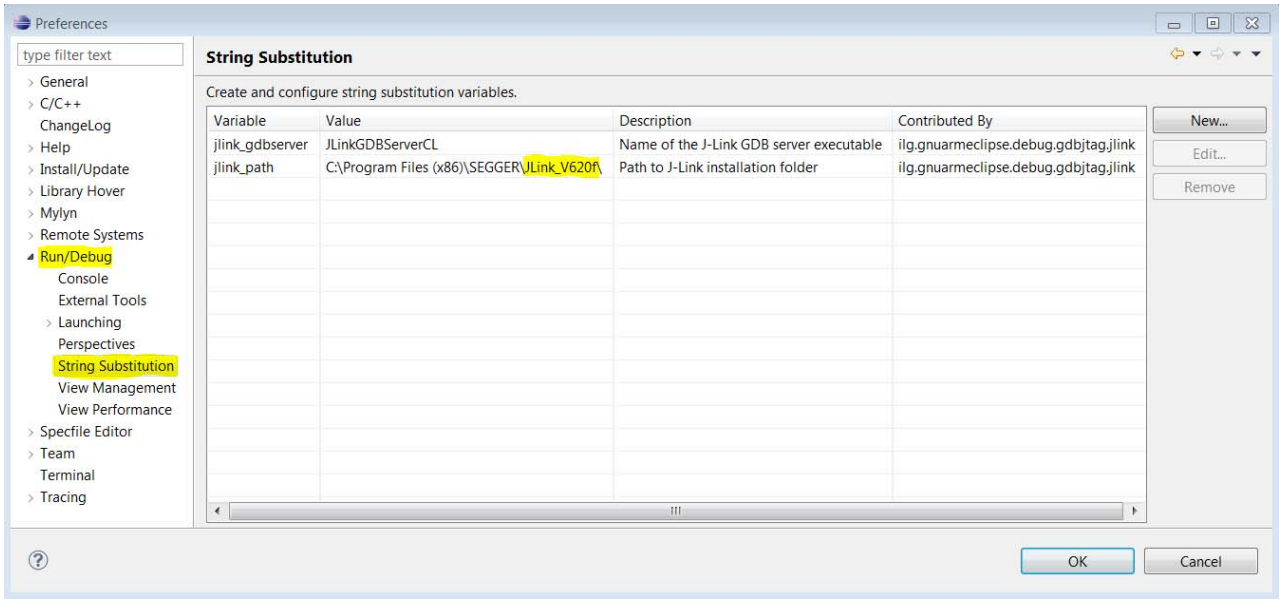

Figure 16. Configuring the J-Link GDB Server

4.4.2 Debugging with the .elf File

If you have performed the steps in the previous sections, you are ready to 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**. A new configuration for *blinky* appears under the **GDB SEGGER** heading, with new configuration details in the right side panel.
3. Adjust the following values for your configuration:
   a. Change to the **Debugger** tab, and enter `RSL10` in the **Device** field. Ensure that **SWD** is selected as the target interface (as shown in Figure 17).

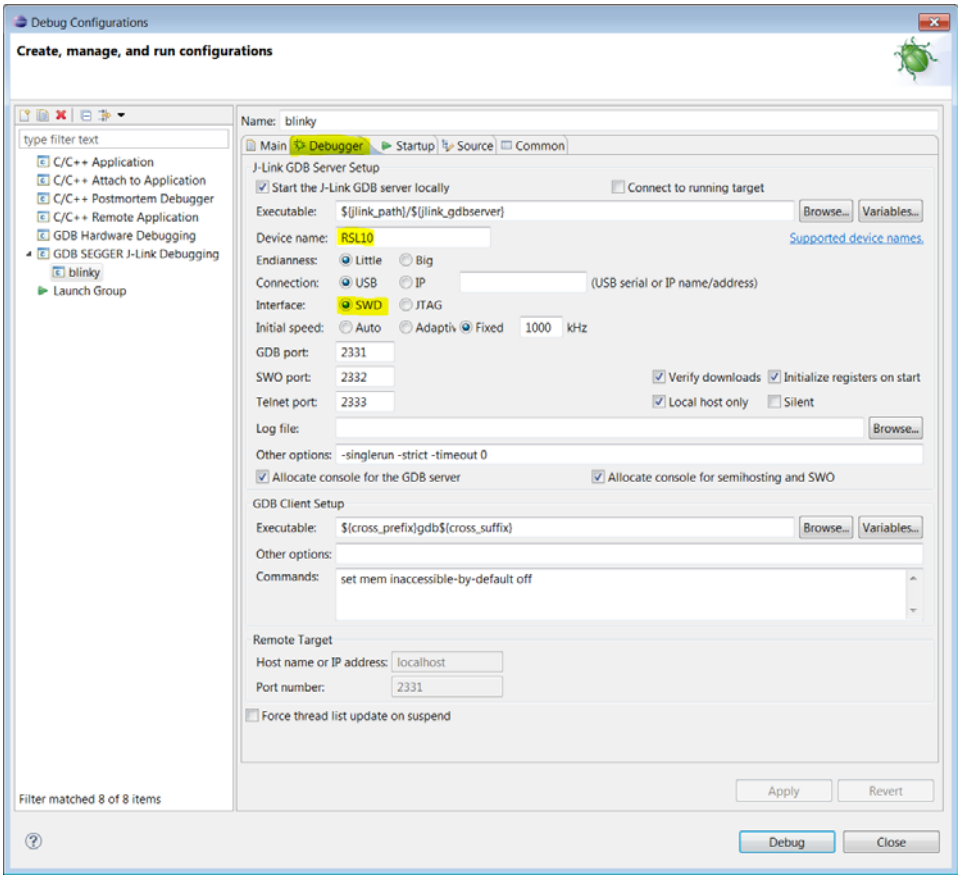

Figure 17. Setting Up a GDB Launch Configuration, Debugger Tab

b. Change to the **Startup** tab and enter the following in the **Run/Restart Commands** field as illustrated in Figure 18:

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

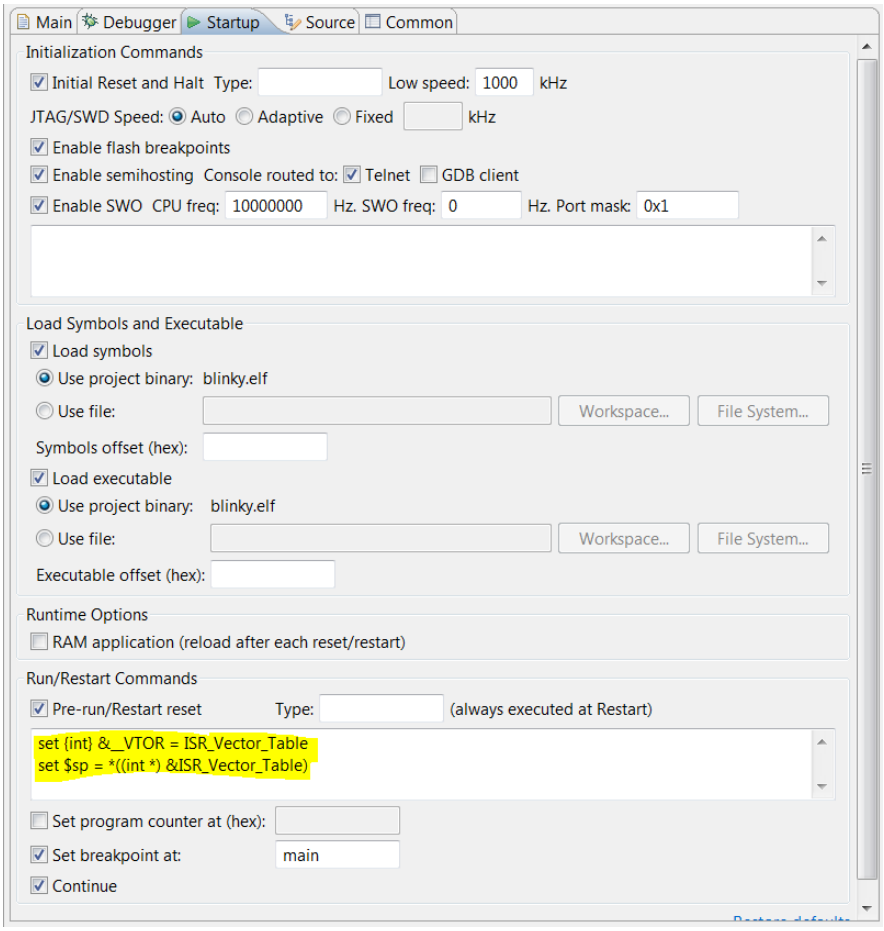

Figure 18. Setting Up a GDB Launch Configuration, Startup Tab

4. Once the updates to the configuration are completed, make sure 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 J-Link version 6.20f or later and that the RSL10 IDE points to the correct location, as

shown in Section 4.4.1, “Preparing J-Link for Debugging”.

IMPORTANT: If the preloaded sample “Peripheral Device with Sleep Mode” is currently on your board (see Section 2.2.1, “Preloaded Sample” on page 6 for help determining what your preloaded sample is), and you are having difficulty connecting to the board, do the following:

a. Connect DIO12 to ground.
b. Press the RESET button (this restarts the application, which will pause at the start of its initialization routine).
c. Repeat step 4 above. After successfully downloading *blinky* to flash memory, disconnect DIO12 from ground, and press the RESET button so that the application will work properly.

Alternatively, use the Stand-Alone Flash Loader (available with its own manual in the *RSL10_Utility_Apps.zip* file) to erase the “Peripheral Devices with Sleep Mode” application from the board’s flash memory.

5. Eclipse asks if you would like to open the Debug perspective. Answer **Yes,** and click on **Remember my decision** so that the question is not asked again.
6. The Debug perspective opens and the application runs up to the first breakpoint in *main*, as shown in Figure 19 on page 22. 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)`.

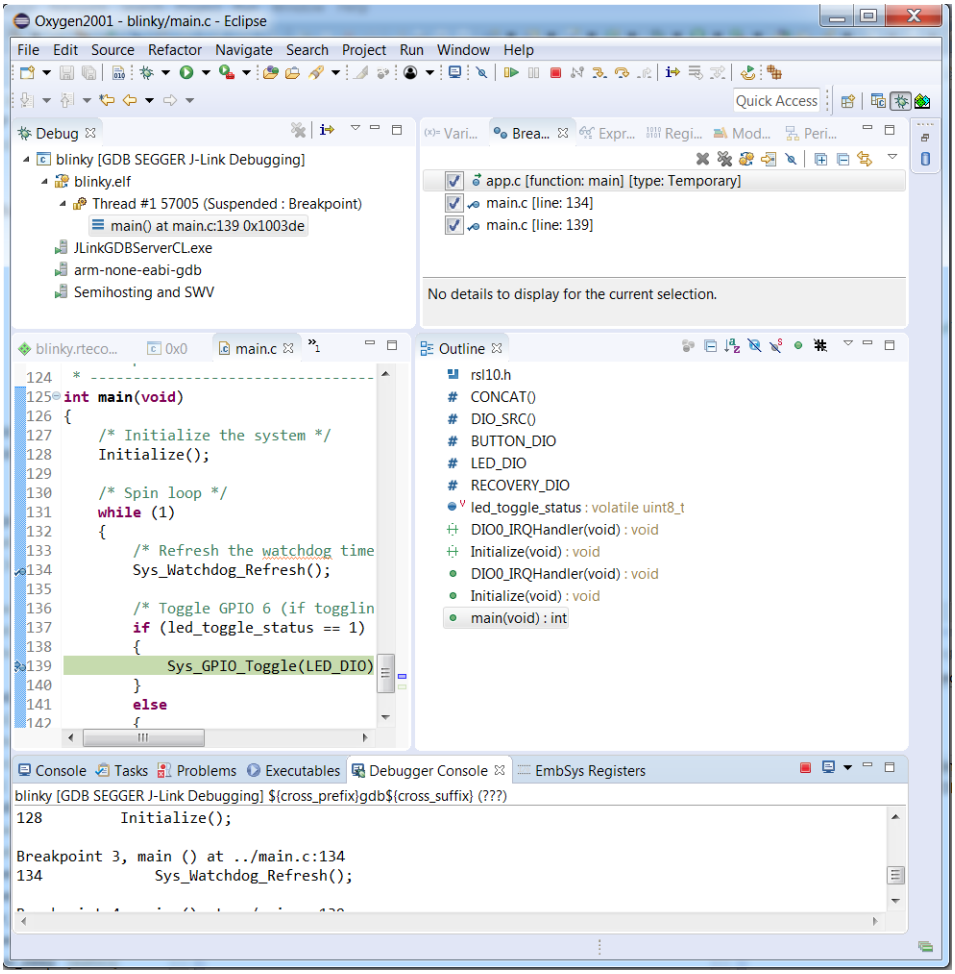

Figure 19. Debug Perspective

4.4.3 Peripheral Registers View with RSL10 IDE

1. The RSL10 IDE includes the EmbSysReg third party plugin, which can be used in a Debug session to visualize and modify RSL10 registers values. To configure the plugin, open **Window > Preferences**
2. On the left panel, select **C/C++ > Debug > EmbSys Register View** (see Figure 20 on page 23)
3. On the right panel, select **Architecture:** cortex-m3, **Vendor:** ONSemiconductor, **Chip:** RSL10 (see Figure 20 on page 23)

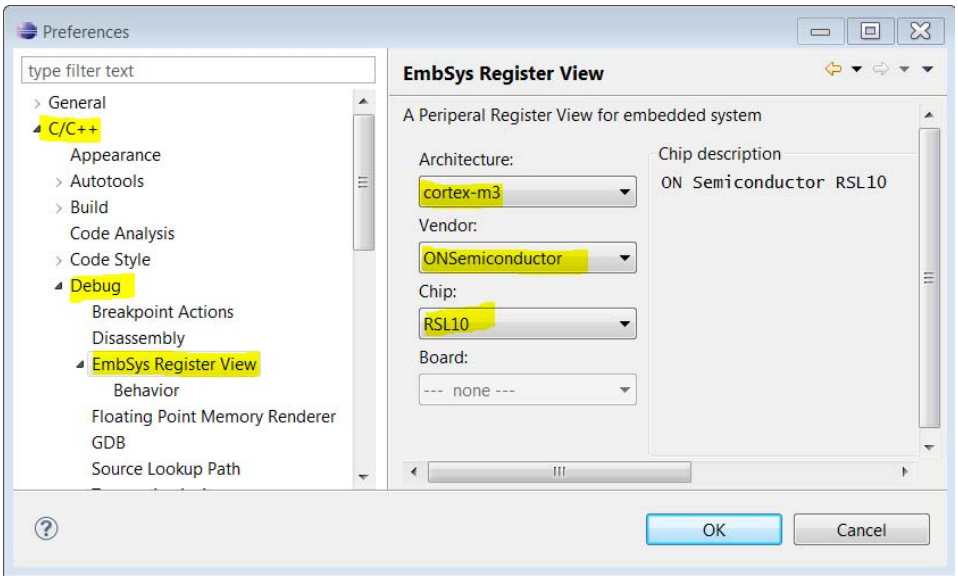

Figure 20. EmbSys Register View Configuration

4. Assuming you are running a Debug session for *blinky*, open **Window > Show View > Other...**
5. Under **Debug**, select **EmbSys Registers** and click **OK**.

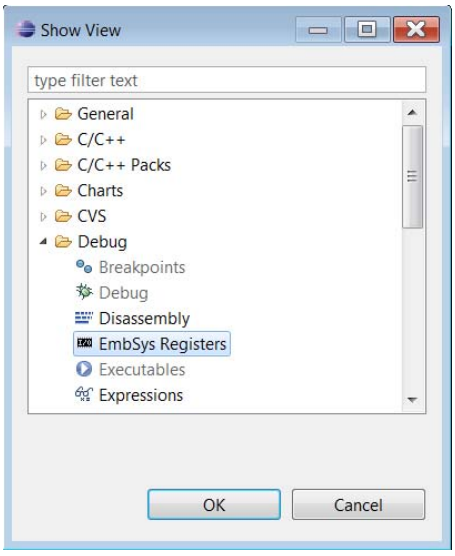

Figure 21. Opening the EmbSys Register View

6. The EmbSys Registers view will open and display all the RSL10 peripheral registers. If you wish, you can drag the EmbSys registers window and place it side-by-side with your source code view (see Figure 22 on page 24)
7. Expand and double click the **DIO > DIO > DIO_DATA** register. It will turn green to indicate you have activated real-time monitoring for this register
8. Press **F6** to step through the code and you can observe that this register bit 6 toggles its state when `Sys_GPIO_Toggle(LED_DIO)` is executed.

9. Double click the **Bin** column, which contains the current binary value of this register. It will display buttons for each bit. Click on the **Bit 6: GPIO** button to change its state and then click on **Set**, as shown in Figure 22 on page 24. You can observe that the LED (DIO6) on your board changes its state.

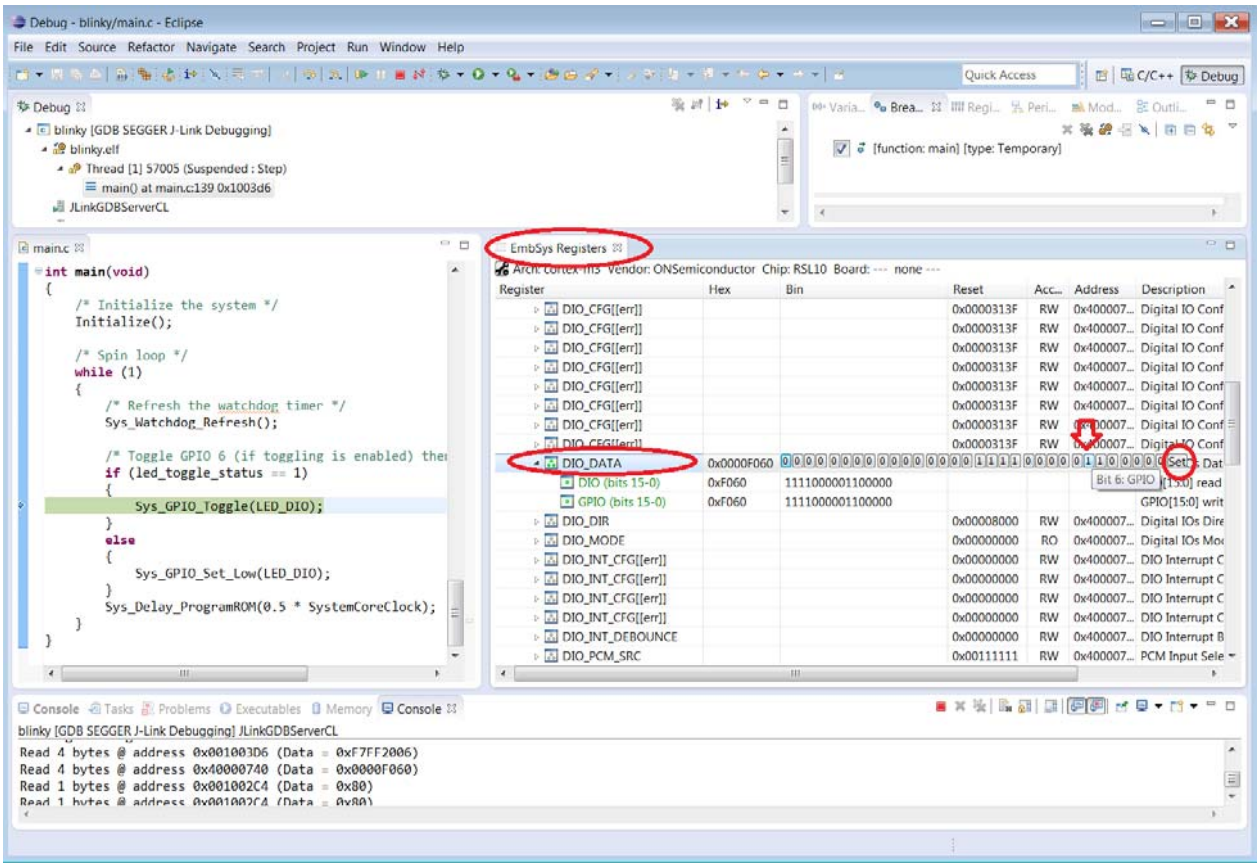

Figure 22. Toggling RSL10 DIO Using the EmbSys Registers View

4.4.4 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.

4.5 Folder Structure of the RSL10 SDK Installation

By default, all files are installed in *C:\Program Files (x86)\ON Semiconductor\RSL10 SDK*, but you can choose to install the SDK anywhere. The subfolders are described in Table 1.

Table 1. Installed Folders

Folder	Contents
Root installation folder	IDE application to start the SDK software license file and 3rd party license files.
arm_tools	Full installation of the GNU Arm Toolchain
configuration	Linker script file *sections.ld*. Linker settings must point to this location for the linker scripts.

Table 1. Installed Folders

Folder	Contents	
documentation	Hardware, firmware and software documentation in PDF format. Also 3rd-party documentation from other companies besides ON Semiconductor. Available from a link in the **Start** menu.	
eclipse	Full installation of Eclipse Oxygen version with GNU Arm plugins to support Arm Cortex-M3 processor processor development.	
include	Include files for firmware (BLE, Syslib, Flashlib etc.) Projects must 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	Sample code sources as ready-to-build projects.

CHAPTER 5

More Information

5.1 More Information

Documentation is available via a shortcut on your **Start** menu: **ON Semiconductor** > **RSL10 SDK Documentation**. You can also access documentation through the *documentation.zip* file downloaded with this release of RSL10.

If you added the RSL10 CMSIS-Pack, a set of documents is included. From your IDE, you can access the documents through the C/C++ perspective by opening any RTE configuration file, such as *blinky.rteconfig*, and selecting the tab **Device** (see Figure 23 on page 28).

In addition to this guide, the following documentation is included with the RSL10 SDK:

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.

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.

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 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.

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.

RivieraWaves Interface Specifications

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.

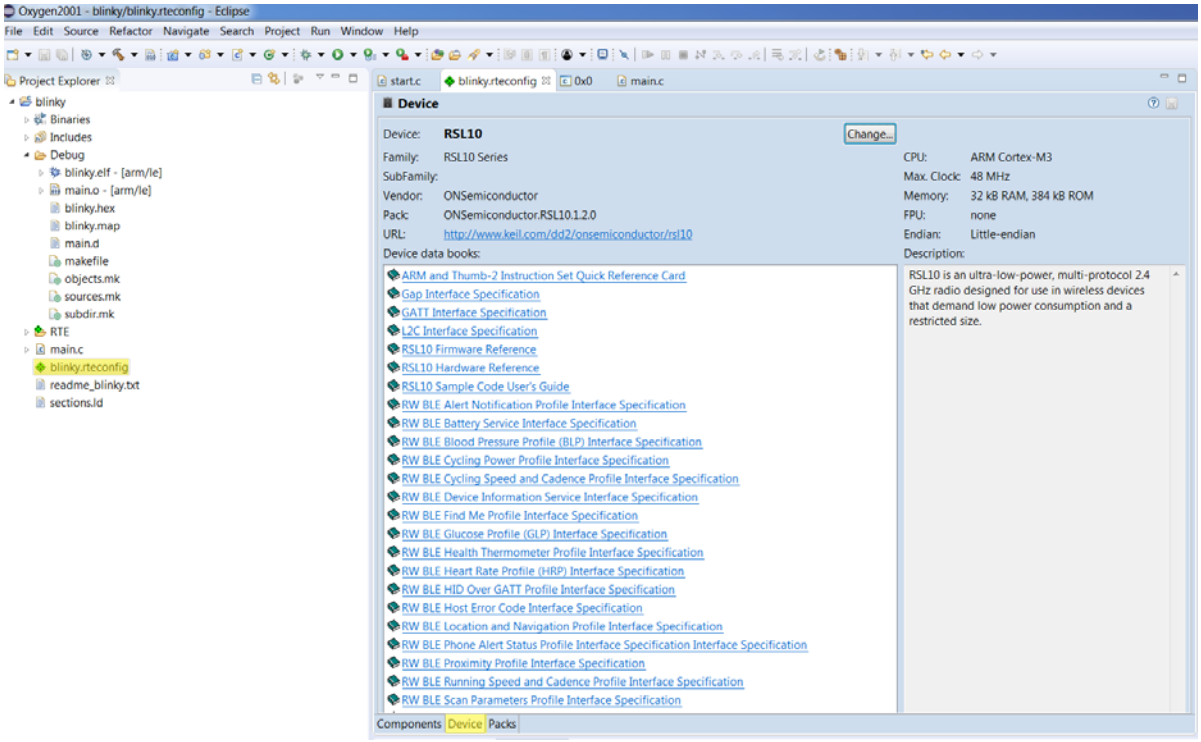

Figure 23. Accessing RSL10 Documentation

5.2 Other Relevant Documentation

The following manuals and tools are available directly from the ON Semiconductor RSL10 website:

RSL10 Evaluation and Development Board Manual

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 USB Dongle

The RSL10 USB Dongle acts as a generic central Bluetooth low energy device so that you can develop applications for peripherals. It is designed to be used with the Bluetooth Low Energy Explorer, which allows developers to become familiar with developing, testing, and evaluating Bluetooth low energy devices. Bluetooth Low Energy Explorer lets you scan for your device, read advertising data, connect, and discover services. You can then pair and bond to your device, read and write to characteristics, subscribe to notifications, and receive characteristics updates. The application also features a logging section, which displays the details of processes in the underlying structure, allowing for easier troubleshooting. For more information, see the manual that accompanies the application.

See the RSL10 product page for more information such as data sheets, application notes, and videos.

APPENDIX A

Arm Toolchain Support

There are several ways in which the RSL10 Eclipse 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.

A.1 Basic Installation

The RSL10 SDK supports the Arm toolchain, tested with the Oxygen versions of Eclipse, 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 RSL10 SDK 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 Eclipse access to the toolchain installed with the RSL10 software tools.

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

A.2 Configuring the Arm Toolchain in Eclipse

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.

A.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 JLink path (global, workspace, project-based) for the location of the Segger JLink executables. This replaces the Run/Debug string substitutions for JLink 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

AND9697/D