Integrating Silicon Labs SiMxxxxx Devices into the IAR Embedded Workbench® IDE

1. Introduction

This application note describes how to configure and use the IAR Embedded Workbench® for ARM (EWARM) Integrated Development Environment (IDE) with Silicon Laboratories Precision32™ 32-bit microcontrollers (SiMxxxxx).

2. Key Points

Key points described in this application note include:

- Generating a blank project in IAR Embedded Workbench
- Configuring an EWARM project for use with Silicon Laboratories SiMxxxxx devices
- Using the EWARM IDE to build, download, run, and debug a project
- Using the Register window and Terminal I/O Viewer

3. Creating a Project

A project is necessary in order to build an example and download the firmware image to the MCU. To create a project in EWARM:

1. Under the **Project** menu, select **Create New Project**. Select a project template and click **OK**. After naming the new project, click **OK**.
2. Under the **File** menu, select **Save Workspace**, and then name the workspace.

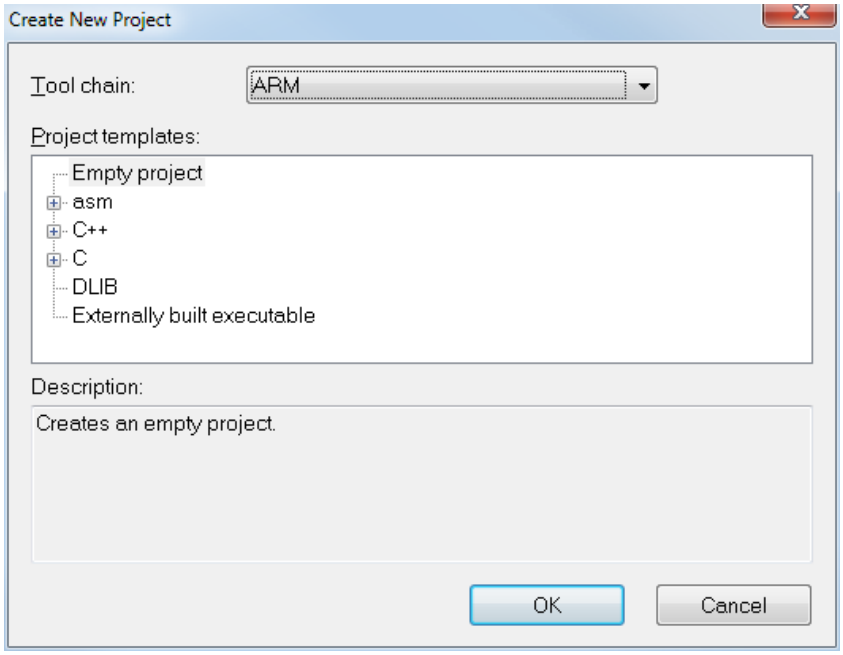


Figure 1. Selecting a Project Template

After creating the blank project, there will be an empty project in the Project Window. The next step is configuring the project options.

4. Configuring an IAR Workspace and Project

The MCU device type and some other specific configurations are required in order to communicate with the MCU using EWARM. Some of the options will be configured after selecting a device, but some modifications are required. This section describes the required settings in all of the configuration tabs within the **Project→Options** dialog. Any sections or tabs not mentioned can be left at the default setting.

4.1. General Options

In the **General Options** section, the **Target** and **Library Configuration** tabs need modifications from the default configuration.

4.1.1. Target

- Select **Device→SiliconLaboratories** and choose the appropriate MCU type.

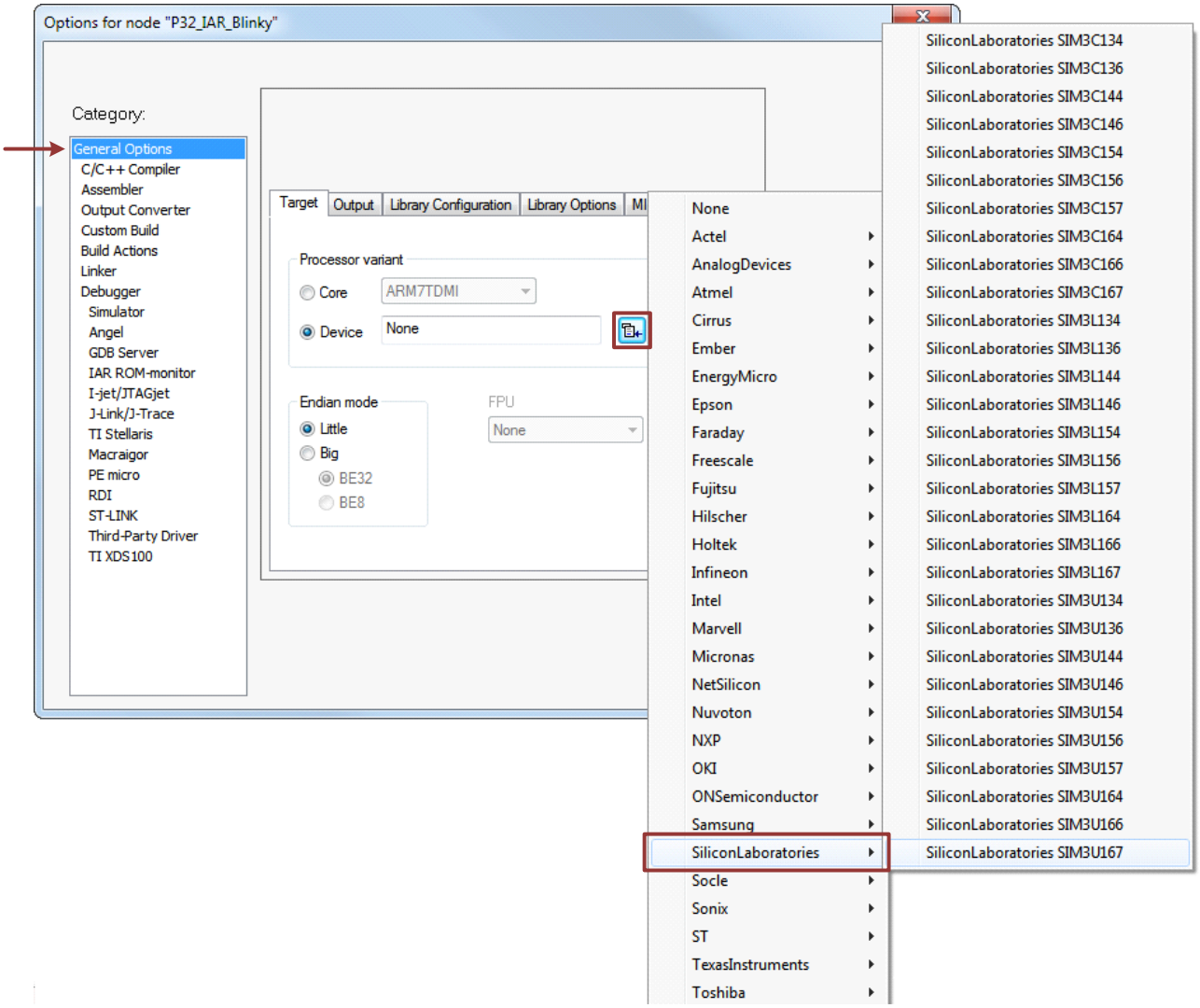


Figure 2. General Options→Target Tab

4.1.2. Library Configuration

- Select a library type according to the requirement.
- Select **Semihosted** for **Library low-level interface implementation**.
- Set **stdout/stderr** type to **Via SWO**. These settings redirect the low-level output (printf, for example) to the Serial Wire Viewer (SWV, or SWO: Serial Wire Output in EWARM) port and can be viewed in the **Terminal I/O** window.

Note: Only devices with the SWV pin can redirect output messages to the IDE.

- Check the **Use CMSIS** option.

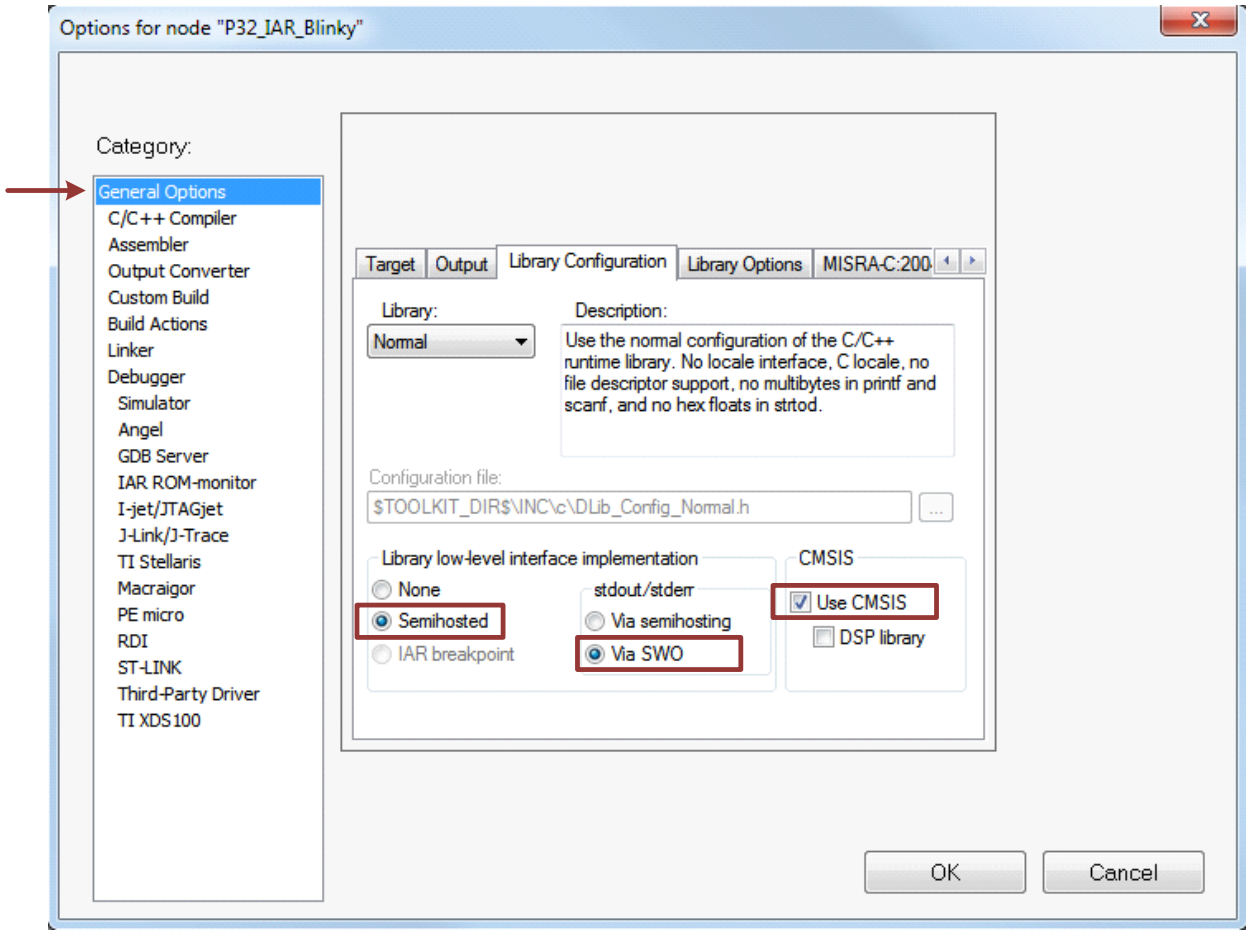


Figure 3. General Options→Library Configuration Tab

4.2. C/C++ Compiler

The **C/C++ Compiler** section contains settings for the compiler. Settings on the **Preprocessor** tab should be adjusted, but all other tabs can be left in their default state.

4.2.1. Preprocessor

- Add any local directories and the Silicon Labs HAL directories to the **Additional include directories** area.
  - C:\SiLabs\32bit\si32-**x**.**y**.**z**\si32Hal**device**
  - C:\SiLabs\32bit\si32-**x**.**y**.**z**\si32Hal\SI32_Modules

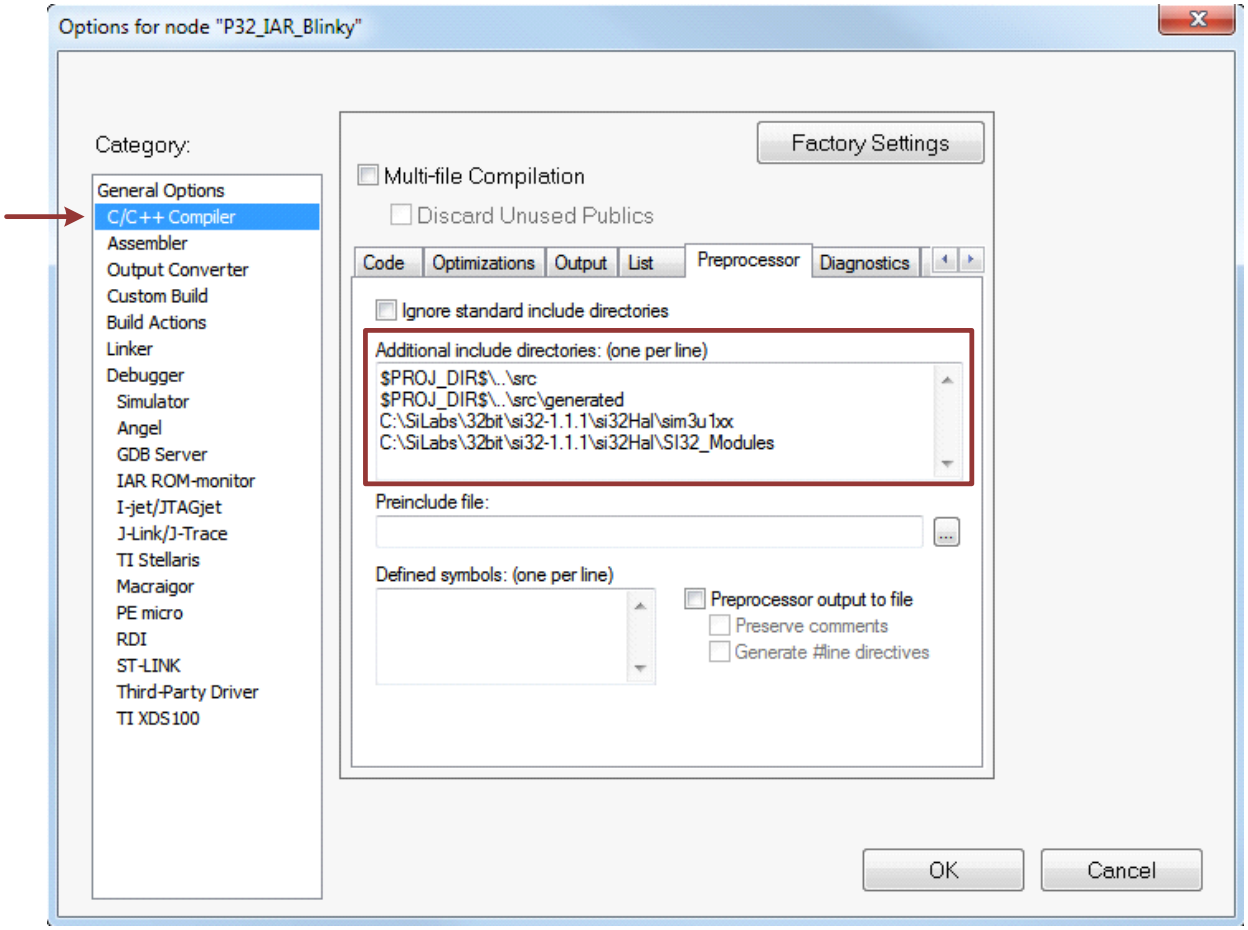


Figure 4. C/C++ Compiler→Preprocessor Tab

4.3. Output Converter

- Check the **Generate additional output** to generate a hex or bin file after a successful build.

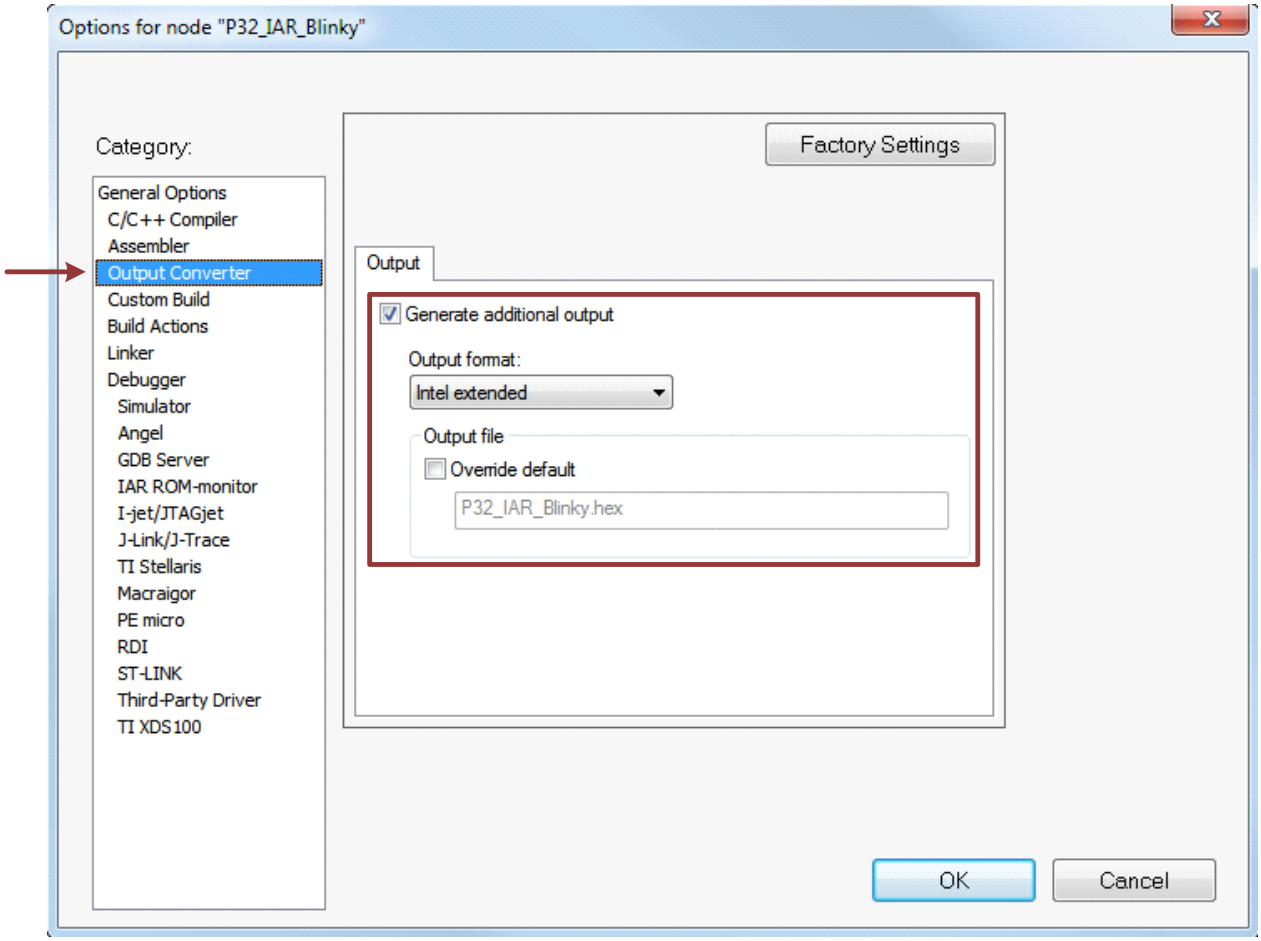


Figure 5. Output Converter→Output Tab

4.4. Linker

- To allocate code or variables to a specific address, check the **Override default** checkbox in the **Config** tab and modify the *.icf file.
- No other changes needed in this section.

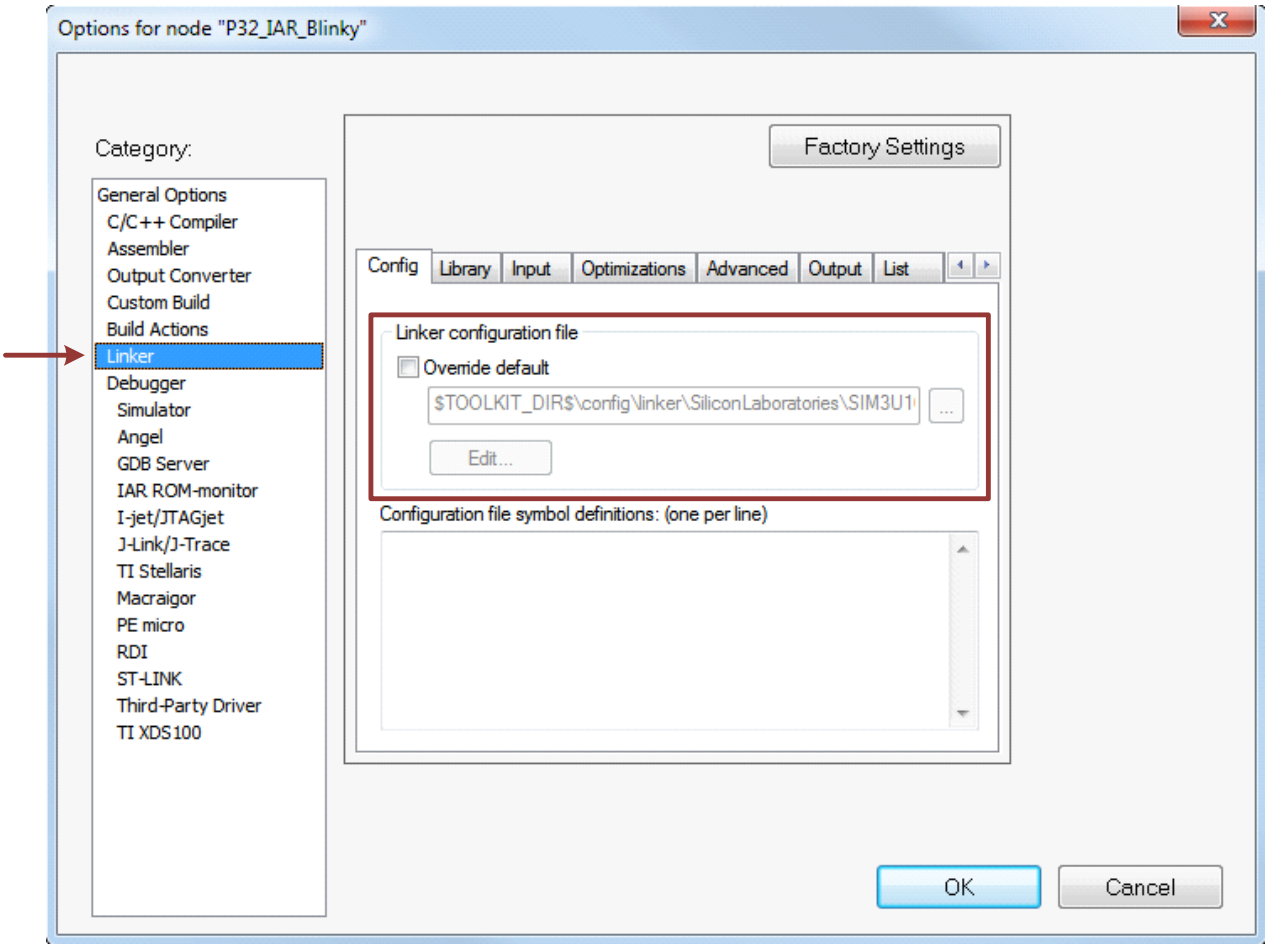


Figure 6. Linker→Config Tab

4.5. Debugger

The **Setup** and **Download Debugger** tabs require changes, but all other tabs in this section can be left in the default state.

4.5.1. Setup

- Select the **J-Link/J-Trace** driver in the **Driver** drop-down menu.
- For code to run to a specific function after reset, check the **Run to** checkbox and input the function name in the text box. If **Run to** is not enabled, the code will run to the reset IRQ handler.

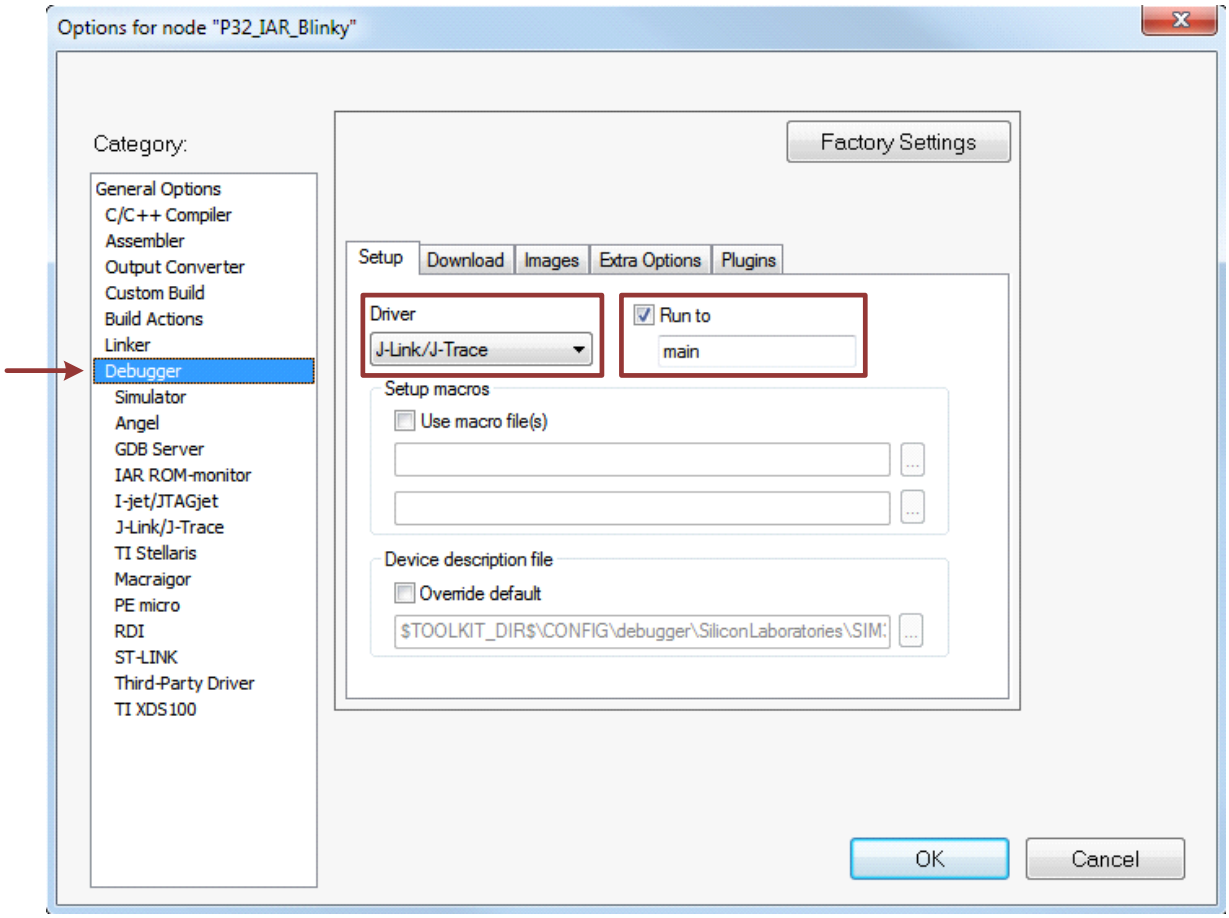


Figure 7. Debugger→Setup Tab

4.5.2. Download

- Check the **Use flash loader(s)** checkbox to download code to flash.

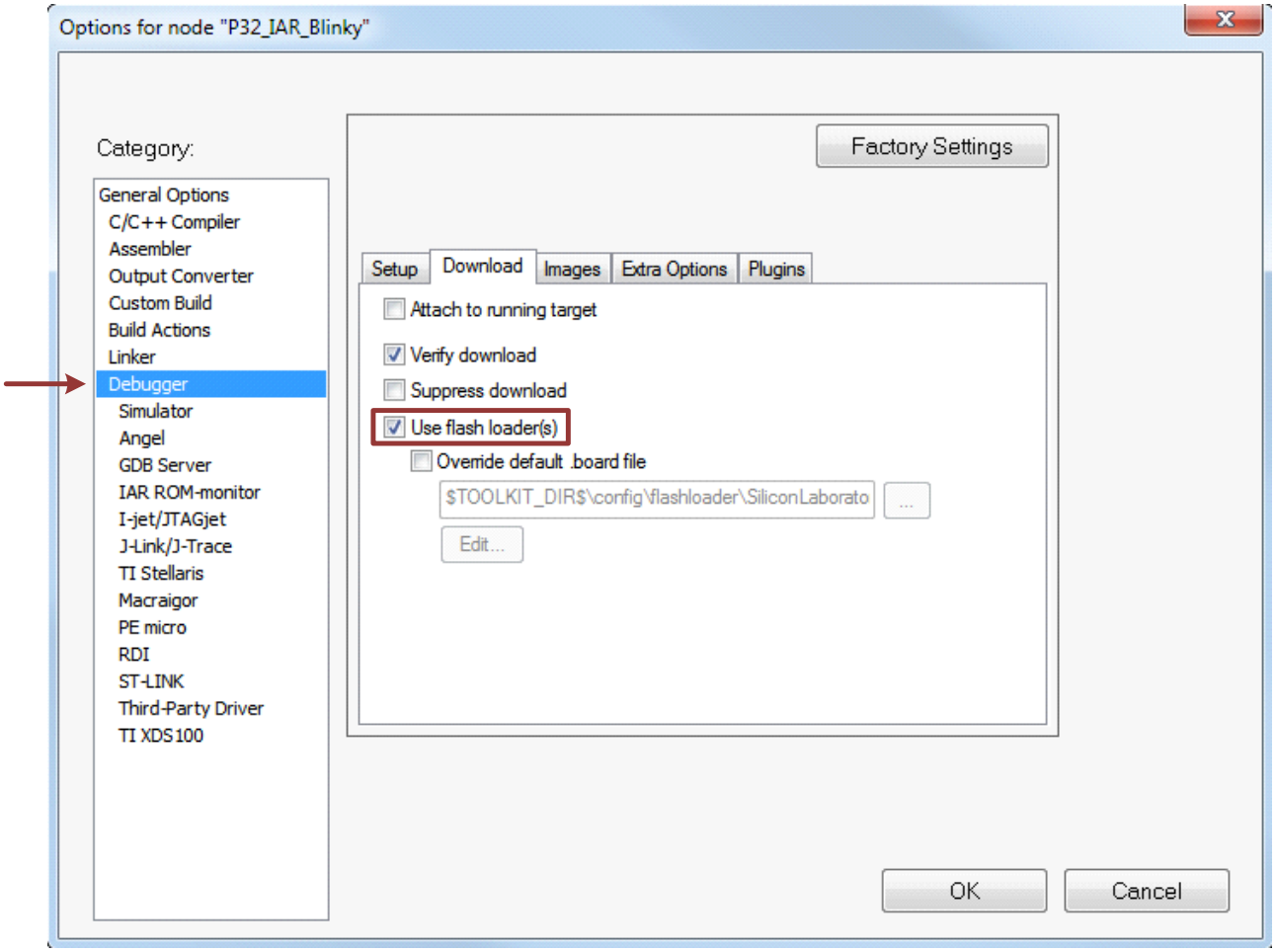


Figure 8. Debugger→Download Tab

4.6. J-Link/J-Trace

In the **Debugger J-Link/J-Trace** subcategory, the **Setup** tab requires some configuration. The **Connection** and **Breakpoints** tabs can retain their default settings.

4.6.1. Setup

- Set the **Reset** drop-down menu to **Core and peripherals**.
- Set the **JTAG/SWD speed** to **Auto**.
- Set the **CPU clock** text box to the AHB clock value to set the IDE clock divider for the desired SWO pin speed.
- Check the **SWO clock Auto** checkbox.

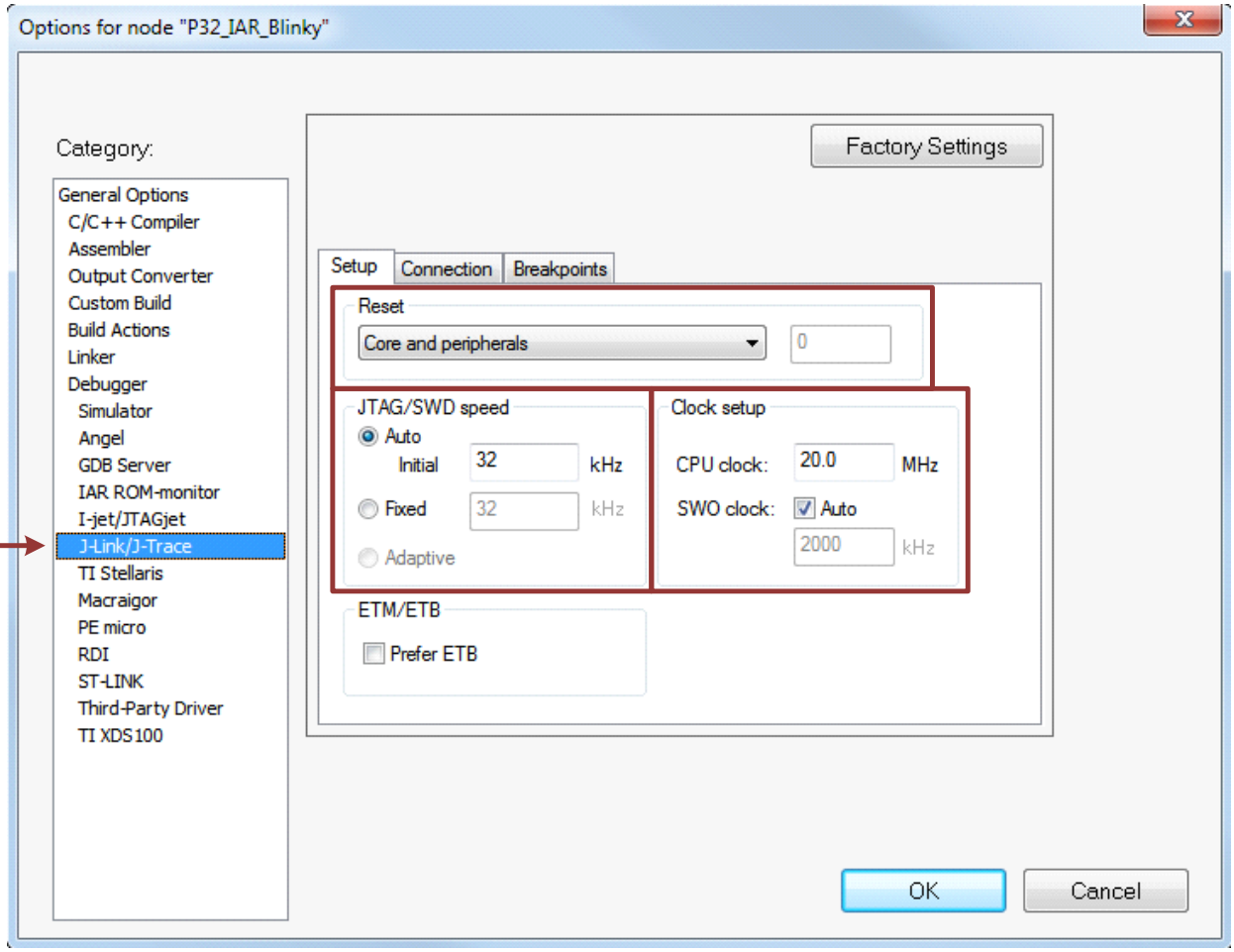


Figure 9. J-Link/J-Trace→Setup Tab

4.6.2. Connection

- No changes needed on this tab.
- The interface area selects SWD and is grayed out because the **stdout/stderr** option is set to **Via SWO** in the **Library Configuration** tab (“4.1.2. Library Configuration”).

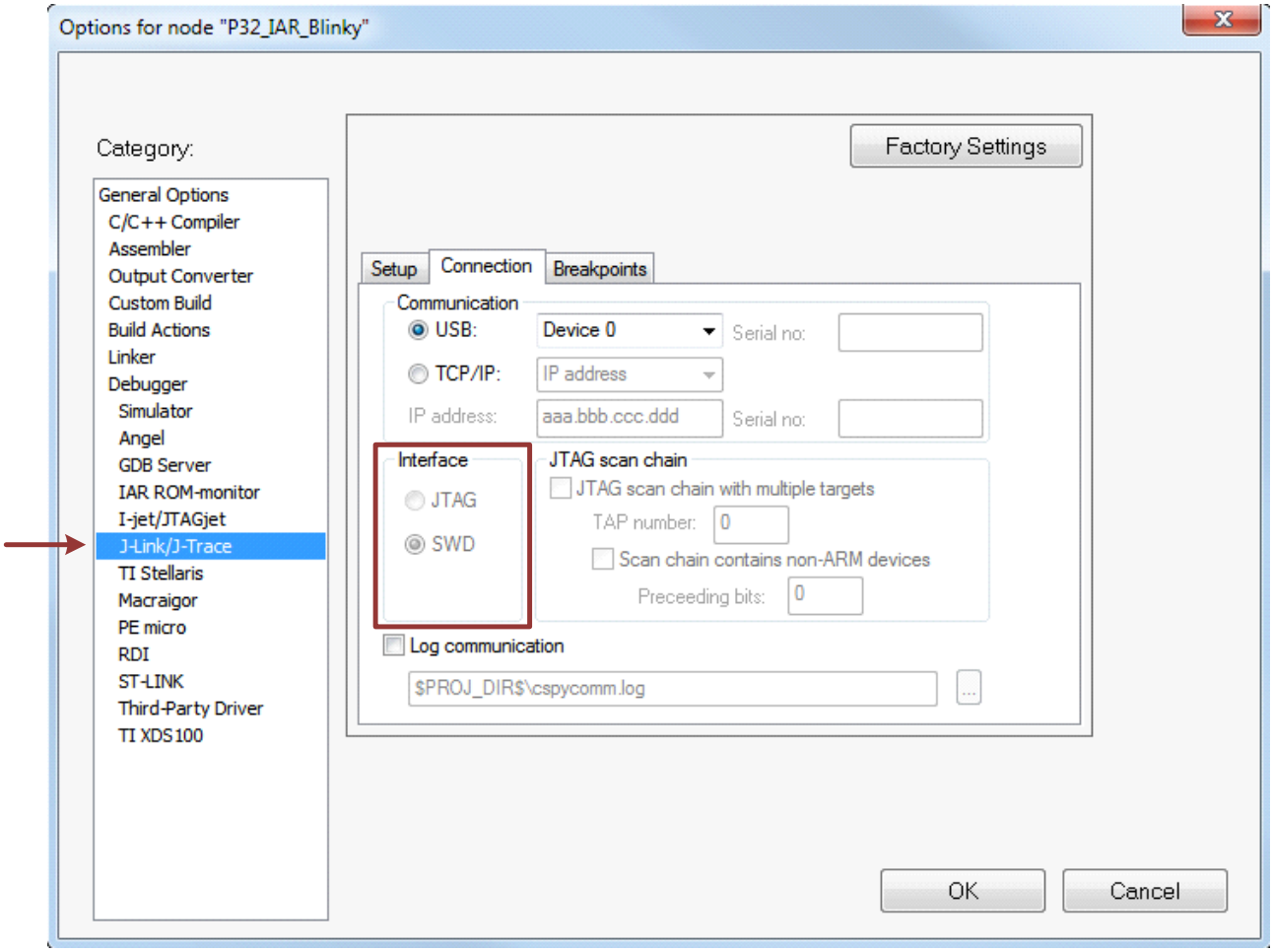


Figure 10. J-Link/J-Trace→Connection Tab

5. Managing Files Within the Project

To add files to a project, add a **Group** in the project tree structure by right-clicking on the root folder in the **Workspace** window and selecting **Add→Add Group…**. Add files to the Group by right-clicking on the Group and selecting **Add→Add Files…**.

5.1. Adding System Files to the Project

A startup file defines the interrupt vectors and is necessary for compatibility between any toolchain and the Silicon Labs MCU. The IAR template files (startup_**device**_iar.s) for different MCUs are in the SDK in the **...\si32-x.y.z\si32Hal\device** directory. For example, the template file for the SiM3U1xx the file is in the **C:\SiLabs\32bit\si32-1.1.1\si32Hal\sim3u1xx\startup_sim3u1xx_iar.s** directory by default for the v1.1.1 SDK.

6. Building the Project

Build a project by selecting **Project→Rebuild all** from the menu. To compile individual files, right-click the file in the **Workspace** window and select **Compile**. Any build-related errors or warnings will appear in the **Build Output** window. If the **Build Output** window is not visible, select **View→Messages-→Build**.

7. Running and Debugging the Code

After successfully building a project, clicking the **Download and Debug** button or selecting **Project→Download and Debug** will a download of the firmware image to the device and start debugging. Clicking the **Debug without Downloading** button or selecting **Project→Debug without Downloading** will start a debug session without downloading the firmware image. To download firmware to the MCU without debugging, select **Project→Download→Download active application**.

Note: If the flash is locked, the IAR IDE cannot communicate with the MCU. Erase the flash using the si32FlashUtility in the **C:\SiLabs\Precision32_vx.y.z\Utilities\FlashProgrammer** directory to restore the IDE communication with the device.

Figure 11. Download and Debug

During a debug session, the **Debug** toolbar or the **Debug** menu has the following options: **Reset**, **Break**, **Step Over**, **Step Into**, **Step Out**, **Next Statement**, **Run to Cursor**, and **Go**. To stop a debug session, click the **Stop Debugging** button or select **Debug→Stop Debugging**.

Figure 12. Debug Toolbar

7.1. Watching Variables

The EWARM IDE has four **Watch** windows and various **Memory** windows. Add variables to a **Watch** window by right-clicking on a variable and selecting **Add to Watch**. To add a variable to the **Memory** window, open the **Memory** window by going to **View→Memory**, copy the variable's name or address to the **Go to** text box, and press **Enter**. To view a specific memory location, open a memory window by selecting **View→Memory** and manually enter a location in the **Go to** text box.

Both methods of variable watching (**Watch** or **Memory** windows) are only available when the variable is in scope.

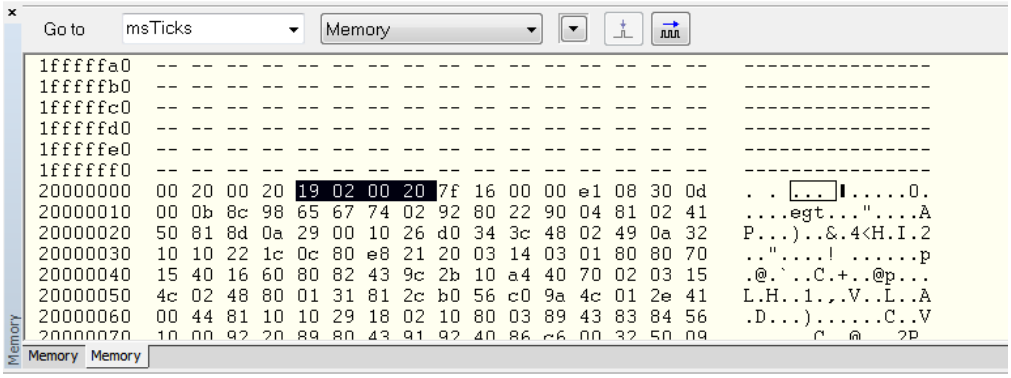


Figure 13. Memory Window

7.2. Viewing Registers

The **Register** window enables the viewing and modification of MCU registers. Hovering the mouse cursor over a field in the **Register** window opens a pop-up window with a description, which can also be helpful for debugging. Select **View→Register** to open all registers on the device. Select the desired peripheral from the drop-down box to view specific registers. Bit fields that changed while stepping through code will be highlighted in green in the **Register** window. Figure 14 shows an example of the PBSTD_2 Register window.

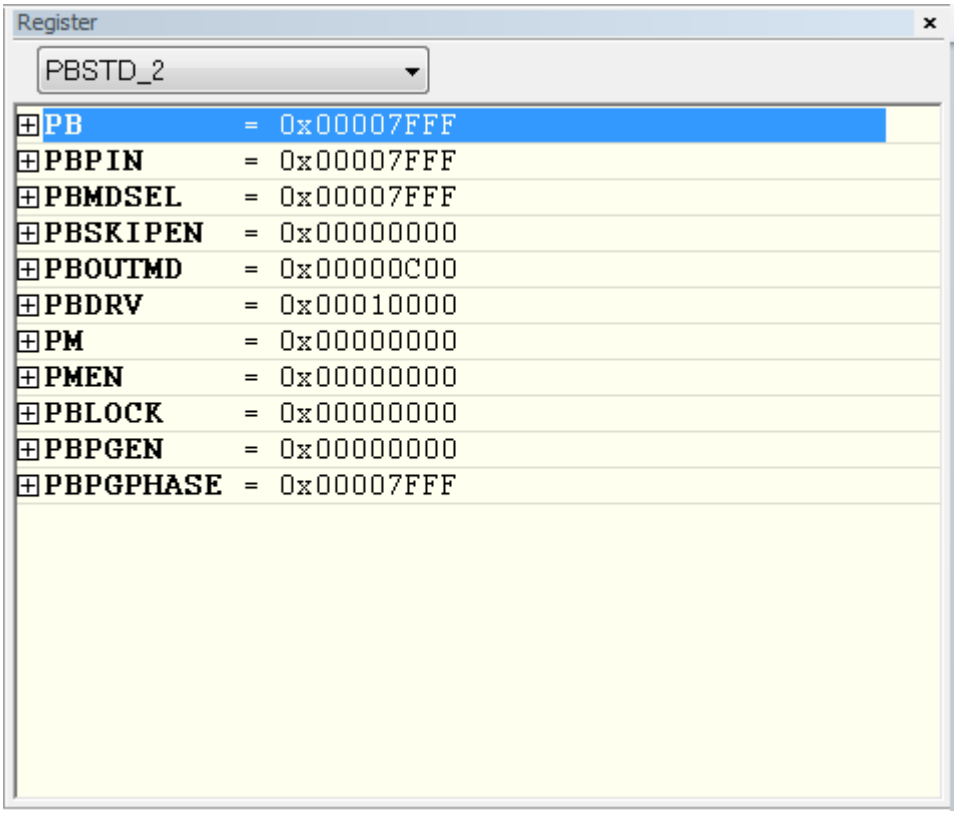


Figure 14. PBSTD_2 Register Window

Individual peripheral bit-fields cannot be modified through the register window if the APB clock to the peripheral is disabled. Enable the corresponding clock enable bit for the peripheral in the CLKCTRL_0 module.

7.3. Trace Configuration

Adding trace capability to projects enables redirecting low-level output functions (like printf) to the debug port, rather than a peripheral like a UART, adding additional visibility into the code without using an application resource.

Trace (SWO) configuration can be modified after starting to debug the project. Click on the SWO icon in the Trace Toolbar to open the **SWO Configuration** window.

Figure 15. Trace Toolbar

In the **SWO Configuration** window:

- The **CPU clock** setting must be set to the same value as the AHB clock in the firmware.
- Enable the **ITM Stimulus Port** channel 0.
- Enable channel 0 output to the **Terminal I/O** window.

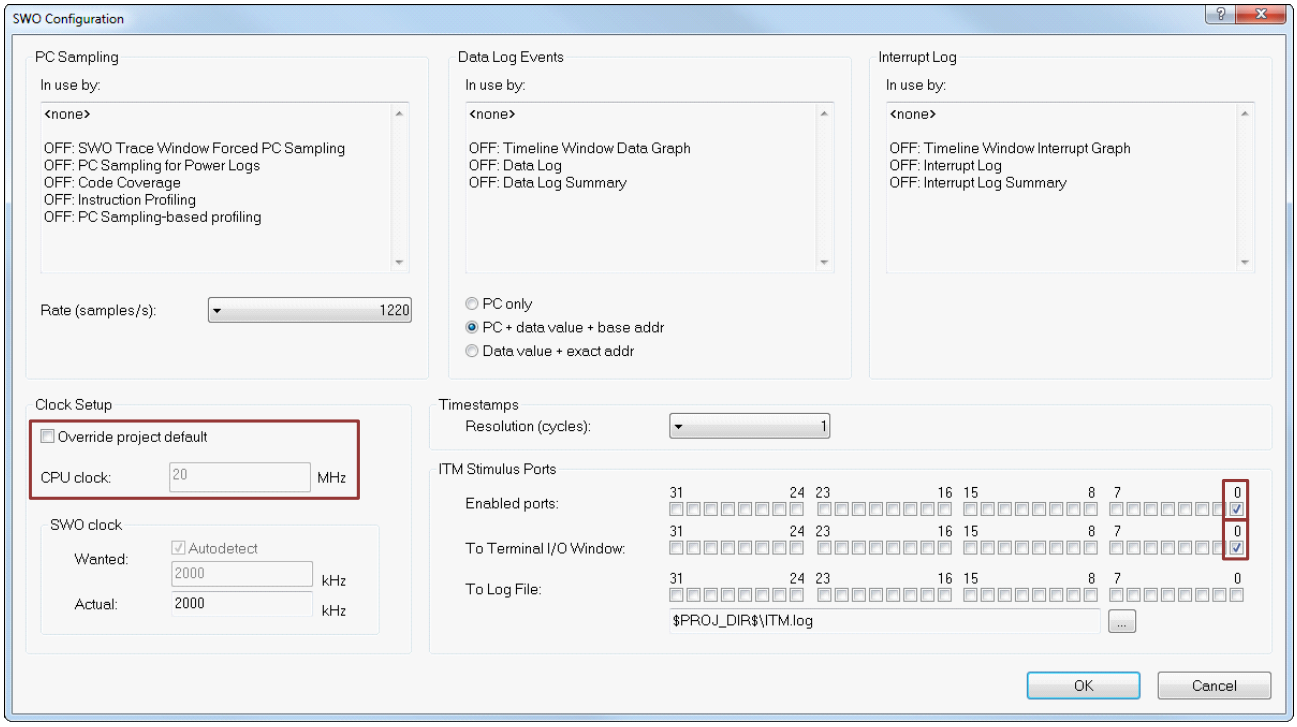


Figure 16. SWO Configuration for Trace

For more information on trace capability in IAR, refer to the “Using CoreSight Trace Techniques on Cortex-M3/M4” article on the IAR website (http://www.iar.com/Global/Resources/Developers_Toolbox/Building_and_debugging/Using_CoreSight_Trace_Techniques_on_Cortex-M3-M4/Using_CoreSight_Trace_Techniques_on_Cortex-M3-M4.pdf).

7.4. Terminal I/O

After setting a project to be **Semihosted** (described in “4.1.2. Library Configuration”), firmware can use printf() functions to print characters or strings to the **Terminal I/O** window. If the **stdout/stderr** type is set to **Via SWO**, the MCU uses the Serial Wire Output (SWO) pin to send data to the **Terminal I/O** window in the EWARM IDE. To output data to the **Terminal I/O** window using SWO:

- Configure the SWO port pin to push-pull mode (this is done in firmware).
- Set **stdout/stderr** to **Via SWO** (described in “4.1.2. Library Configuration”).
- Set the **CPU clock** to the firmware AHB clock value (described in “4.6.1. Setup”).
- Open the **Terminal I/O** window in the EWARM IDE by selecting **View→Terminal I/O**.

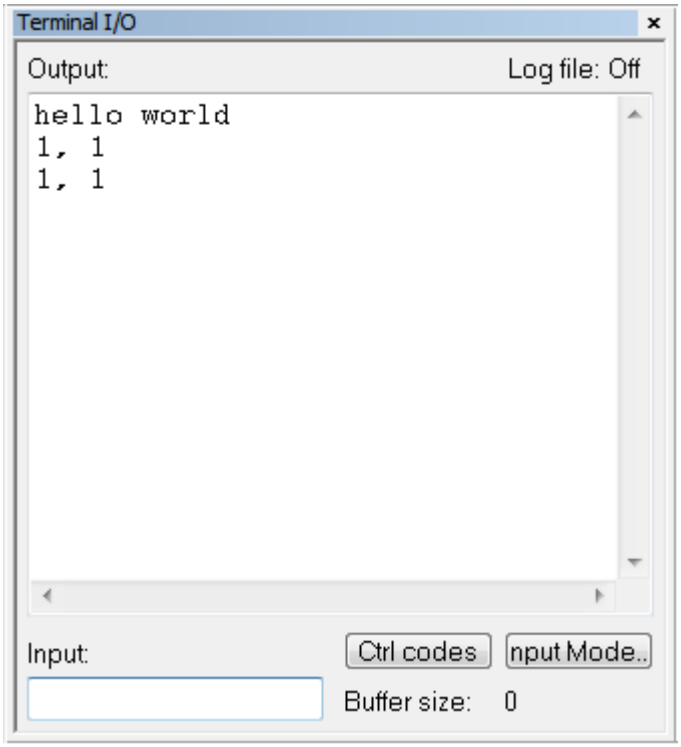


Figure 17. Terminal I/O Window

8. Code Examples and Source/Include Files

Examples and source/include files for each of the Silicon Labs devices are installed with the Precision32 IDE. The default location for these examples is **C:\Silabs\32bit\si32-x.y.z\Examples**, where **x** is the major revision number **y** is the minor revision number, and **z** is the trivial revision number. The **IAR** folder contains the EWARM project files, and the **src** folder contains the source files for each example.

The example included with this application note, **AN749_Example_3U_Blinky**, includes hard-coded paths to the v1.1.1 HAL. To use this example with another version of the HAL:

1. Open the workspace (**eww** file) and modify the **startup_sim3u1xx_iar.s** and **system_sim3u1xx.c** paths.
2. Modify the include path for compiler as described in “4.2.1. Preprocessor”.

9. Creating and Building an Example in IAR Embedded Workbench

To create a new project and build it in IAR Embedded Workbench using the SiM3U1xx Blinky example code included with the Silicon Labs SDK:

1. Create a blank project and workspace (“3. Creating a Project”).
2. Copy the **src** folder of the Blinky example in the SDK to the new IAR workspace. The default installation path for the example is **C:\SiLabs\32bit\si32-x.y.z\Examples\sim3u1xx\Blinky**, where x is the major SDK revision number, y is the minor revision number, and z is the trivial revision number.
3. In the **Workspace** window, right-click on the project name, select **Add→Add Files...**, and add the files in the **src** directory to the workspace.
4. Add a new **generated** group to the project by right-clicking in the **Workspace** window and selecting **Add→Add Group...**. Add the files in **src\generated** directory to the **generated** group.
5. Add **startup_sim3u1xx_iar.s** to the project (“5.1. Adding System Files to the Project”).
6. Add **system_sim3u1xx.c** and **system_sim3u1xx.h** to the project from the SDK (default directory is **C:\SiLabs\32bit\si32-x.y.z\si32Hal\sim3u1xx**). The IAR workspace should look as shown in Figure 18.

Note: The example included with this application note, **AN749_Example_3U_Blinky**, includes hard-coded paths to the v1.1.1 HAL. Follow the instructions in “8. Code Examples and Source/Include Files” to use this example with another version of the HAL.

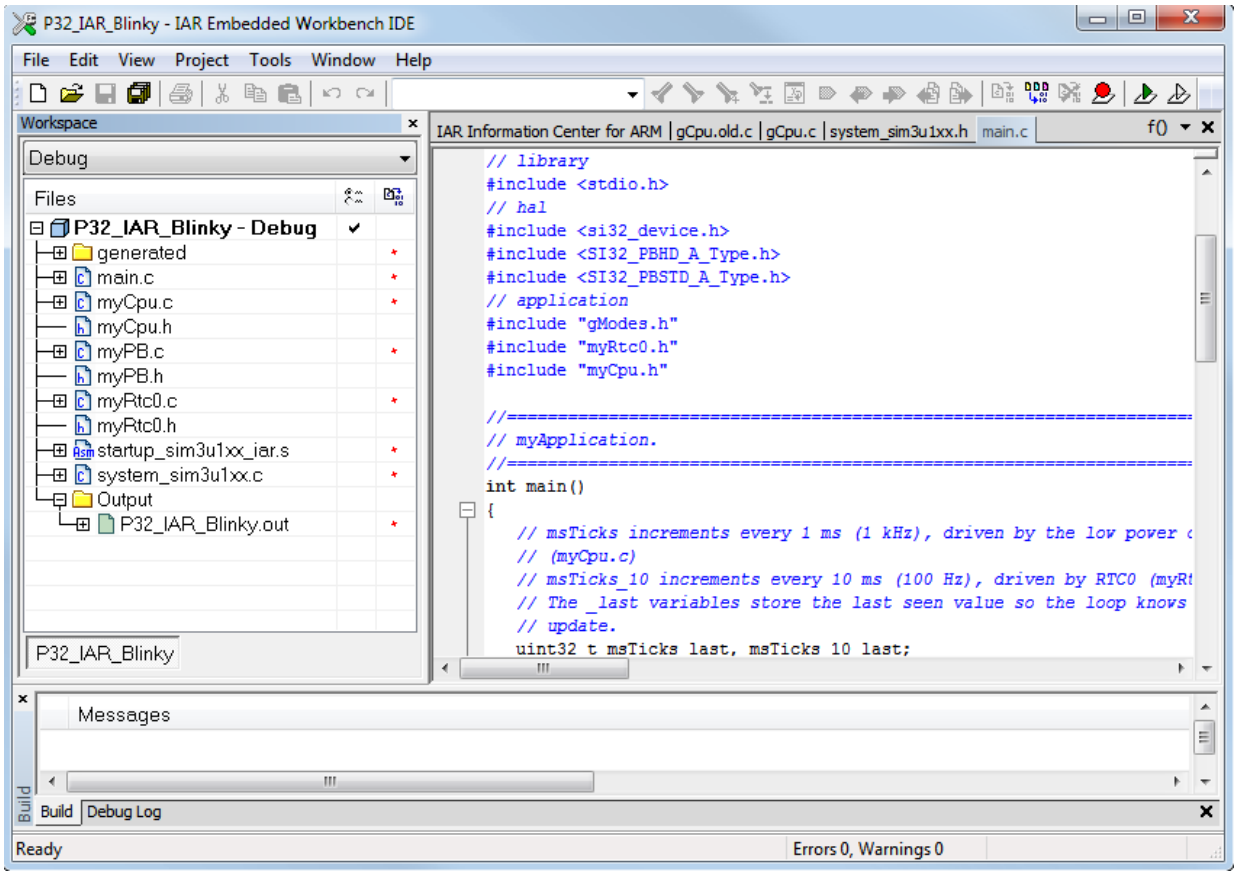


Figure 18. IAR Blinky Example

7. Set the options for this project as discussed in “4. Configuring an IAR Workspace and Project”.
8. Build the project by right-clicking on the project name in the **Workspace** window and selecting **Rebuild All**.

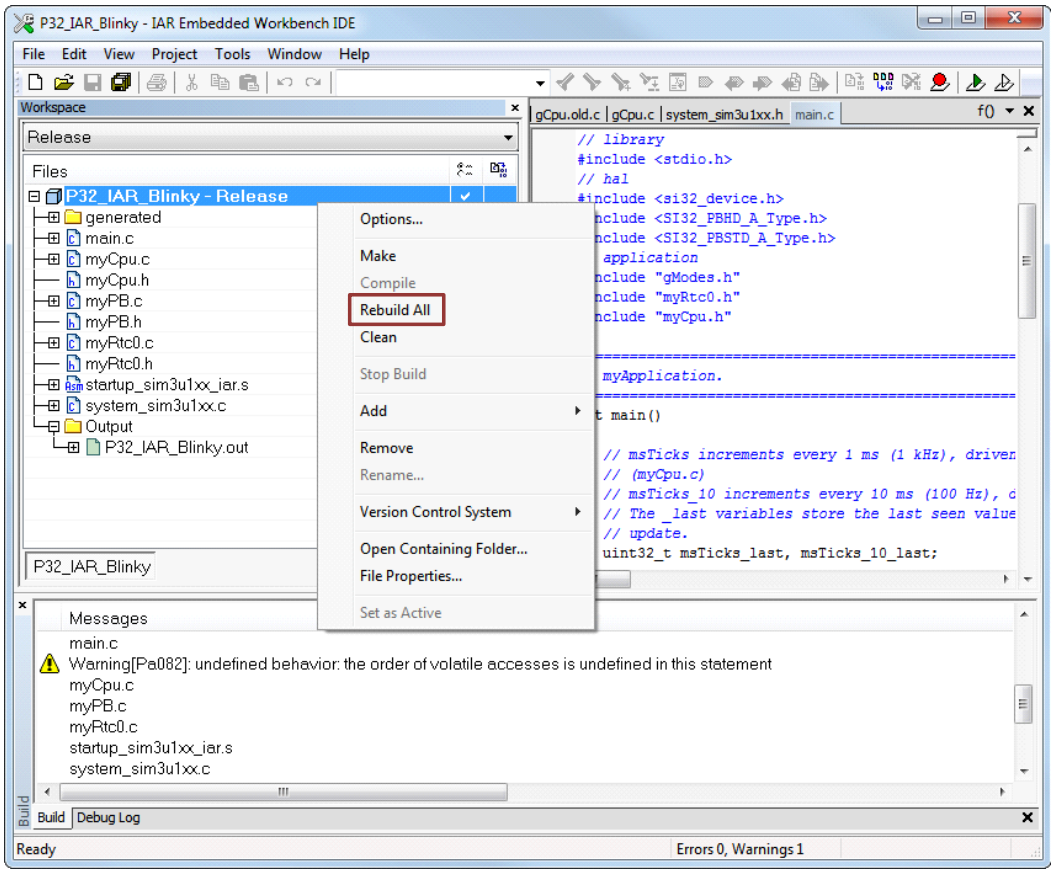
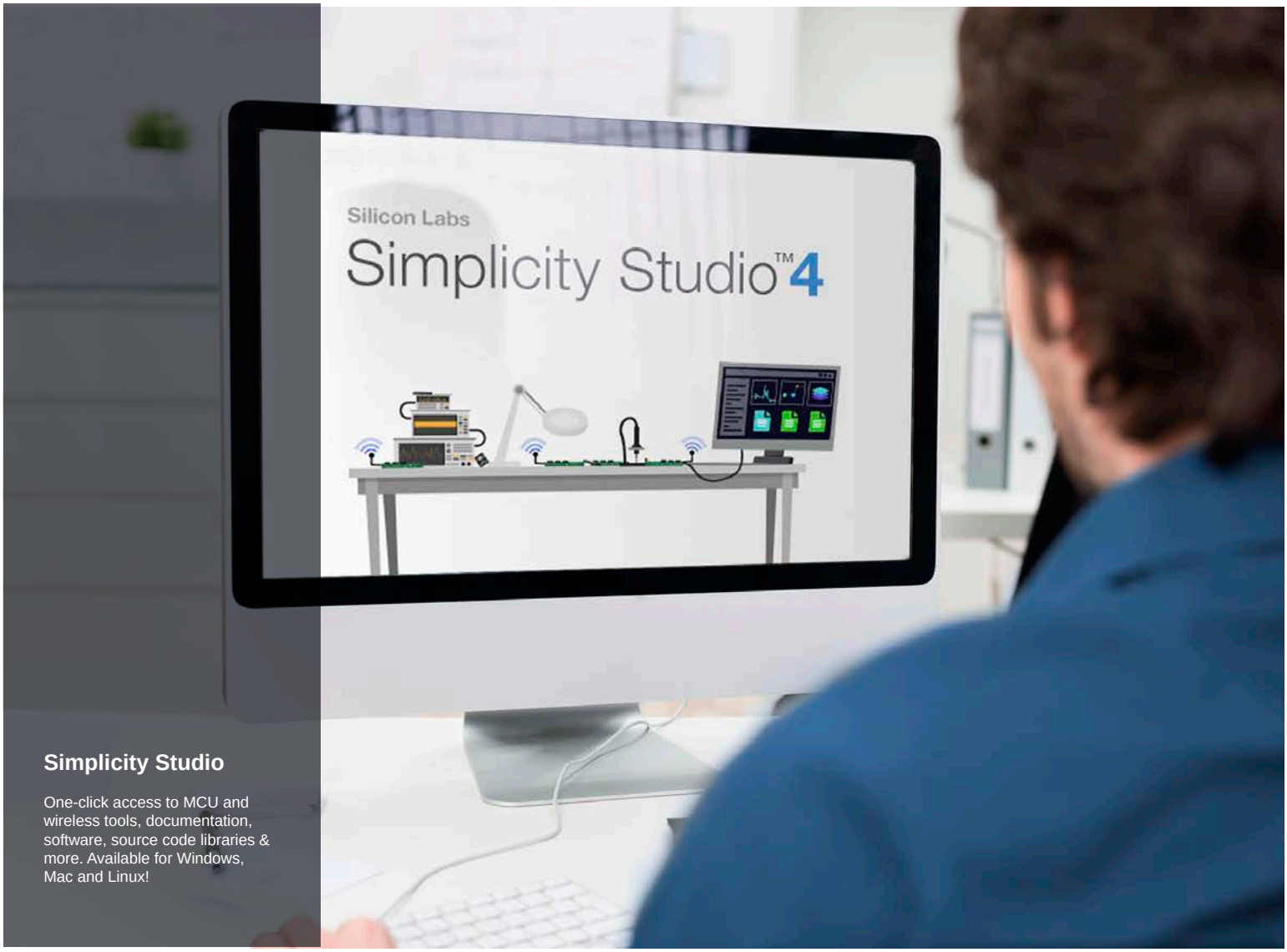


Figure 19. Building the IAR Blinky Example

9. Setup the hardware (SiM3U1xx MCU card):
   a. Connect the debug adapter to J31.
   b. Connect a USB mini cable to J13 to power the board.
   c. Move the SW5 **SYSTEM Power Select** switch to the top **USB** position. The blue power LED (**PWR LED** DS1) should turn on.
10. In the IDE, click the **Download and Debug** button or go to **Project→Download and Debug**.
11. Run the code and observe the blinking LEDs on the board. The code will also print information to the **Terminal I/O** window (**View→Terminal I/O**).



Simplicity Studio

One-click access to MCU and wireless tools, documentation, software, source code libraries & more. Available for Windows, Mac and Linux!

IoT Portfolio
www.silabs.com/IoT

SW/HW
www.silabs.com/simplicity

Quality
www.silabs.com/quality

Support and Community
community.silabs.com

SILICON LABS

Silicon Laboratories Inc.
400 West Cesar Chavez
Austin, TX 78701
USA

http://www.silabs.com