Microchip University

Lab Manual for:

Configuring an SmartFusion® 2 SoC FPGA and Developing Firmware Applications

Table of Contents

MICROCHIP
UNIVERSITY

Introduction:

This manual demonstrates how to configure the Microcontroller Subsystem (MSS) in a Microchip SmartFusion® 2 SoC FPGA, develop firmware applications and debug application using Microchip's SoftConsole tool.

Required Hardware and Software:

- Microchip Hello FPGA kit (M2S-HELLO-FPGA-KIT)
- Microchip FlashPro4, FlashPro5 or FlashPro6 programmer
- Libero® SoC v2021.3 or later
- SoftConsole v2021.3 or later
- System Builder requires the following IP Cores:
  - CoreAHBLSRAM v2.0.113
  - CoreI2C v7.0.102
  - CoreSPI v3.0.156
  - CoreGPIO v3.0.120
  - CoreTimer v1.1.101
  - CoreUARTapb v5.2.2
  - CorePWM v4.1.106

Libero SoC and SoftConsole are available for Windows 10 and Linux. Libero SoC requires a license. The free Silver license can be used to complete this lab. Refer to the course syllabus for additional information about Operating System support and information about obtaining the free Silver license.

The required System Builder cores are downloaded automatically when the host machine is connected to the internet. The cores are included in the IP_cores folder of the lab files. Refer to Appendix B for instructions on how to install the required cores manually if you do not have an internet connection while implementing the lab.

Refer to the course syllabus for links to the Libero SoC and SoftConsole software, the Hello FPGA kit and the FlashPro programmer.

Upon completion, you will:

- Create a Libero SoC project.
- Use System Builder to Configure the SmartFusion 2 MSS and add a FPGA fabric peripheral.
- Generate the design in SmartDesign and complete the FPGA design flow.
- Program the SmartFusion 2 SoC FPGA on the Microchip SmartFusion 2 Hello FPGA Kit and run the application.
- Develop and debug firmware applications that execute from the SmartFusion 2 eSRAM, eNVM and external DDR memory using SoftConsole.

Prerequisites:

The lab material assumes you have prior experience with:

- Libero SoC software
- C language programming
- Debugging Firmware applications

Lab 1 – Configuring the SmartFusion® 2 MSS

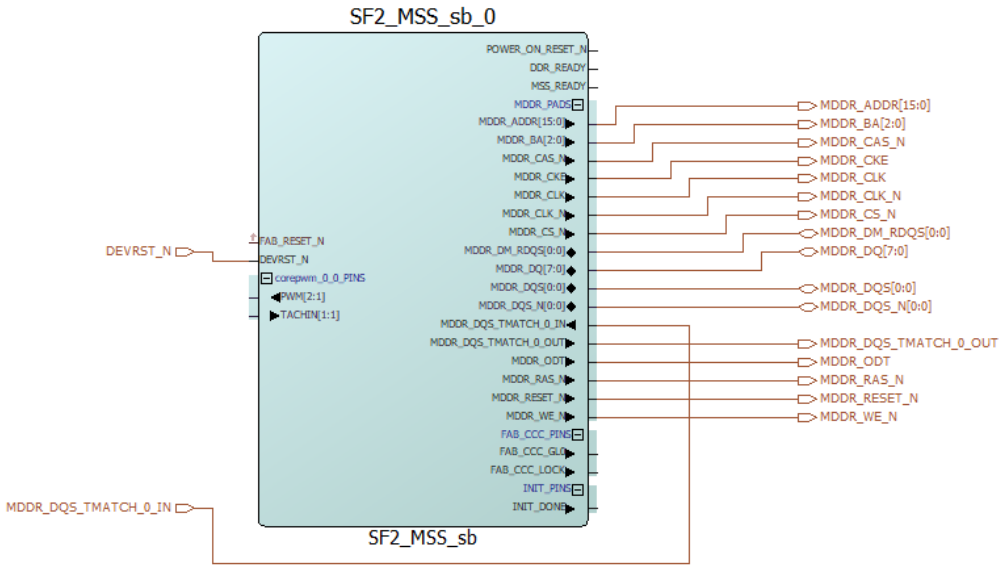
Purpose:

To learn how to create a Libero® SoC project and configure the SmartFusion 2 MSS with System Builder.

Overview:

The SmartFusion 2 Microcontroller Subsystem (MSS) is configured with the System Builder tool in Libero SoC. In this lab, you will create a Libero SoC project and use System Builder to configure the SmartFusion 2 Cortex®-M3, and peripherals.

Upon successful completion of this lab, you will see a component named SF2_MSS_sb_0 in the Libero SoC SmartDesign canvas.



Procedure:

STEP 1: Extract the lab source files

Extract Microchip_University_SF2_class.zip to obtain the files needed to complete this lab. Refer to the Appendix A for a description of the extracted files.

STEP 2: Checking and Installing the required IP cores

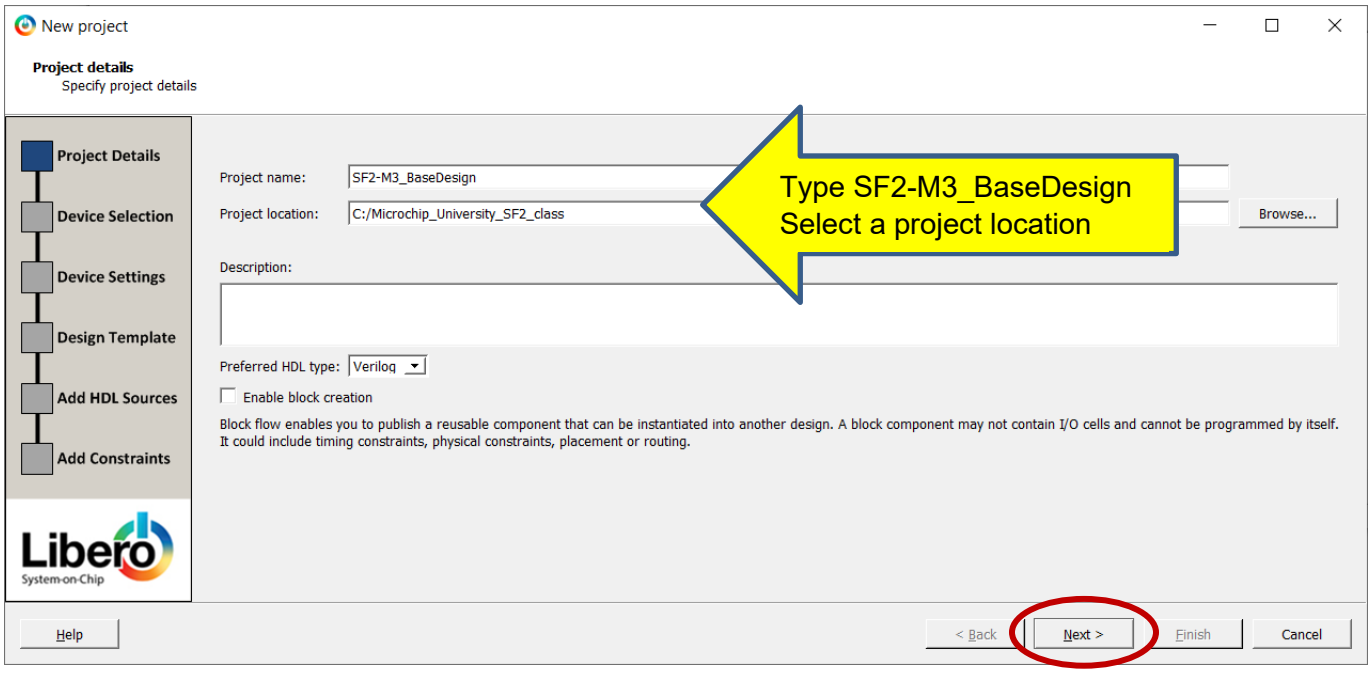
System Builder uses older versions of some IP cores that might not be in the IP vault when Libero SoC is installed. The IP cores that are required to complete this lab are provided in the lab source files. Refer to Appendix B for information about how to check for cores and install any missing cores.

Step 3: Creating a Libero® SoC project

Step 3a: Open the Libero SoC software by clicking the shortcut icon on the desktop. Skip to the next step if you opened Libero SoC to download cores as described in Appendix B.

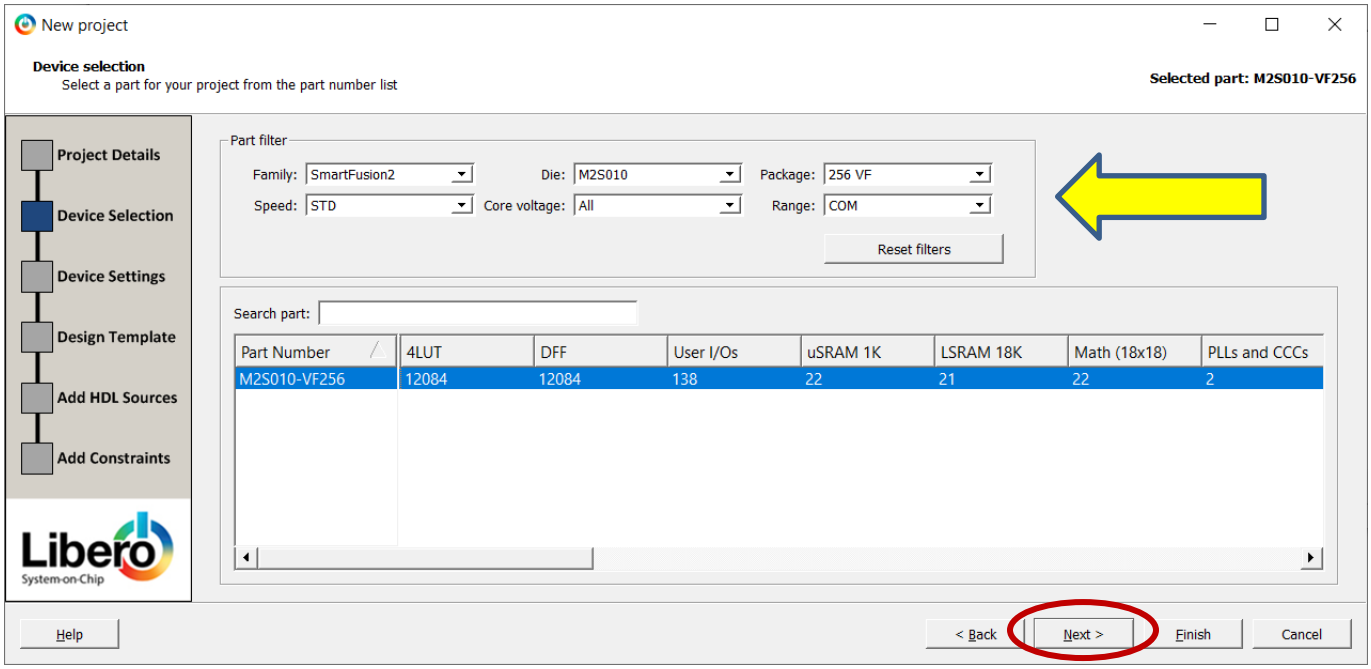
Step 3b: Create a new Libero SoC project by clicking **New** in the GUI (highlighted below).

Step 3c: Enter the project details shown below. Choose a folder without spaces in the name for the project location. Click **Next**.



Step 3d: Enter the information shown below in the Device Selection page dialog box then click **Next**.

- Family: SmartFusion 2
- Die: M2S010
- Package: 256 VF
- Speed: STD
- Core voltage: All
- Range: COM

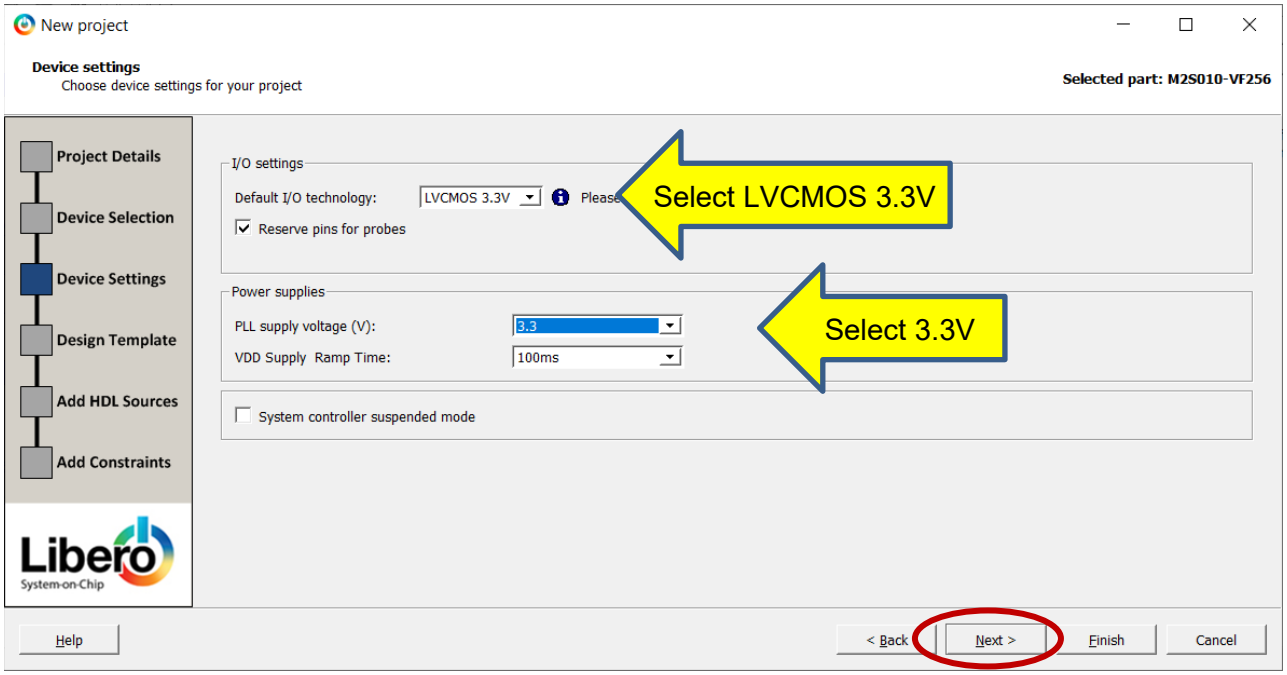


Step 3e: Enter the information shown below in the Device Settings page dialog box then click **Next**.

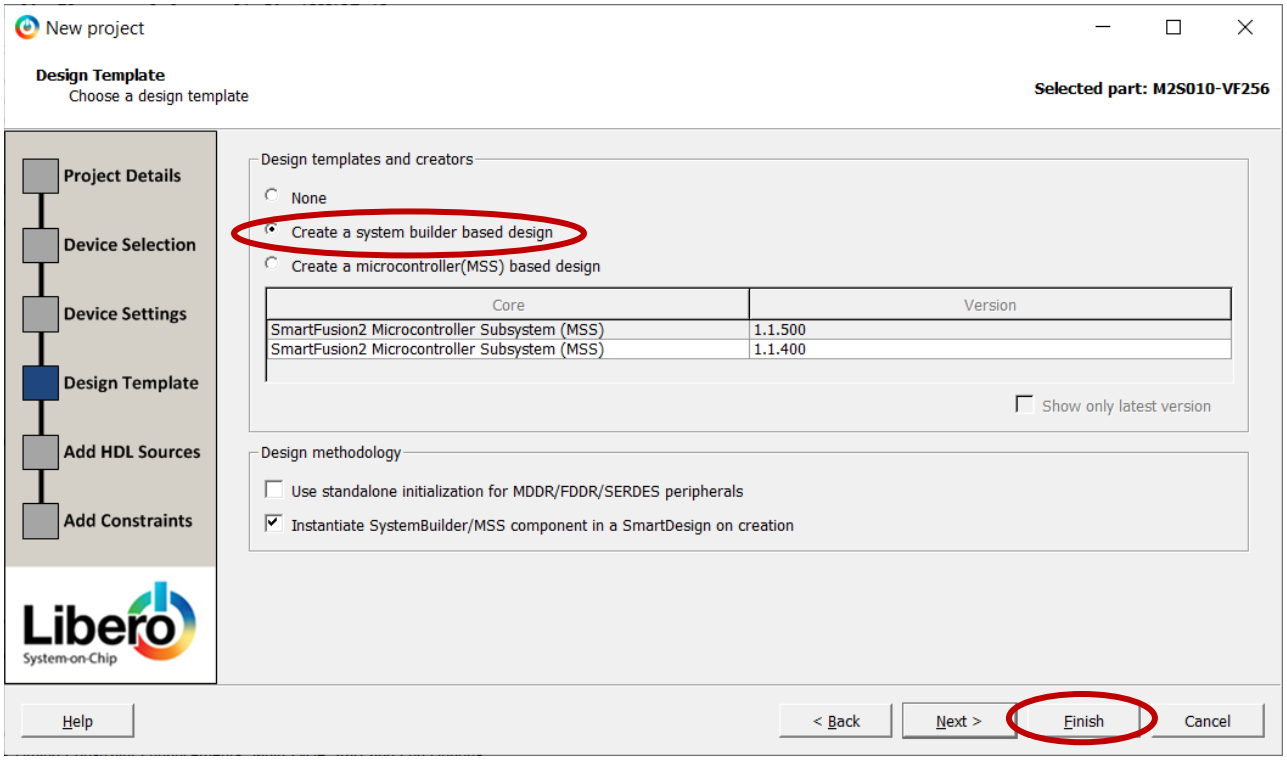
- Default I/O Technology: LVCMOS 3.3V
- Reserve pins for probes: checked (default)
- PLL Supply Voltage: 3.3V
- VDD Supply Ramp Time: 100ms (default)
- System Controller Suspend mode: un-checked (default)

SmartFusion® 2 I/Os support multiple I/O standards. The I/O standard for the I/Os in this lab are LVCMOS 3.3V.

The SmartFusion 2 PLL Supply voltage can be either 2.5V or 3.3V. The voltage setting in the New Project dialog box must match the PLL Analog Supply voltage on the board to ensure that the PLL works correctly. The PLL Analog Supply voltage is 3.3V on the Hello FPGA board.



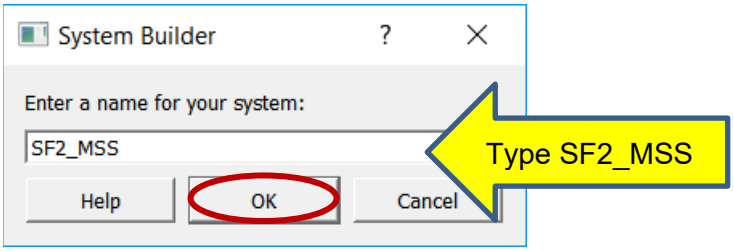
Step 3f: Choose Create a system builder-based design in the Design Template page then click **Finish**. This option will automatically open System Builder to configure the SmartFusion® 2 MSS upon clicking Finish.



STEP 4: Configuring the SmartFusion® 2 MSS

System Builder is a graphical tool for configuring the SmartFusion 2 Microcontroller Subsystem (MSS). Help is available for each System Builder page.

Step 4a: Enter SF2_MSS when prompted for the name of the system and then click **OK**.

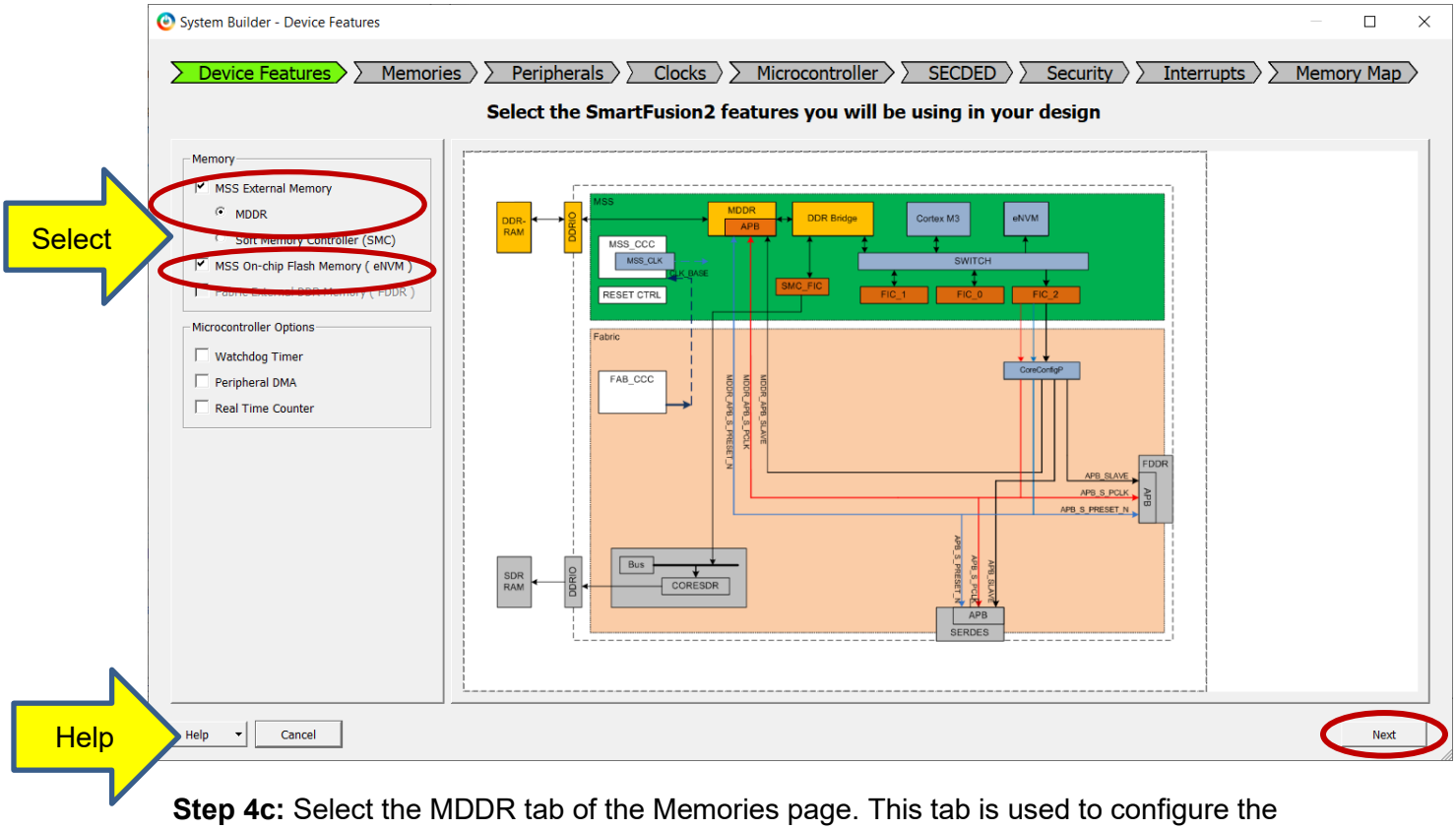


The System Builder wizard will open. There might be a slight delay if missing IP cores need to be downloaded.

Step 4b: Enter the following on the Device Features page then click **Next**.

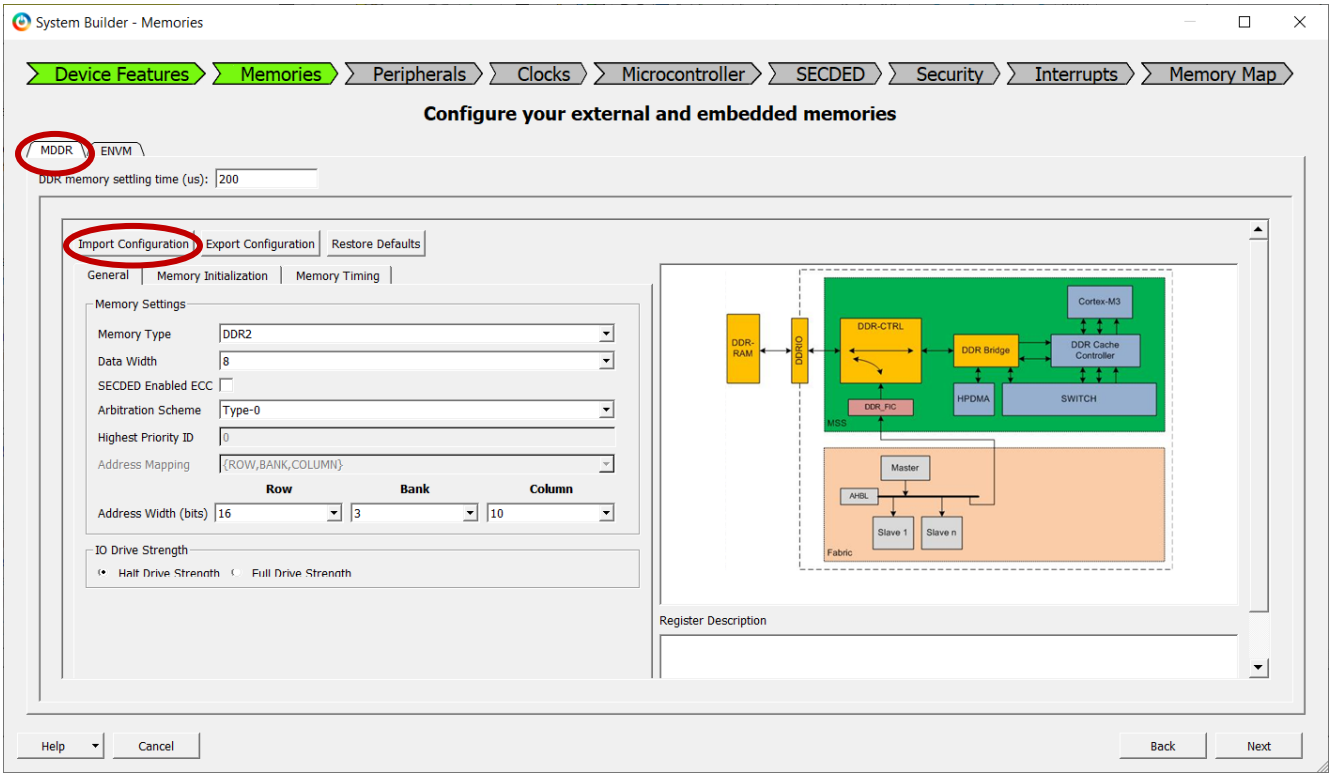
- Memory
  - MSS External Memory — checked; MDDR selected
  - MSS On-chip Flash Memory (eNVM) — checked
- Microcontroller Options
  - Watchdog Timer — un-checked (default)
  - Peripheral DMA — un-checked (default)
  - Real Time Counter — un-checked (default)

These selections will enable the MSS DDR memory controller and the embedded flash memory (eNVM). The Watchdog Timer, Peripheral DMA and Real Time Counter peripherals are not used in this lab. Selected features are highlighted in the figure.

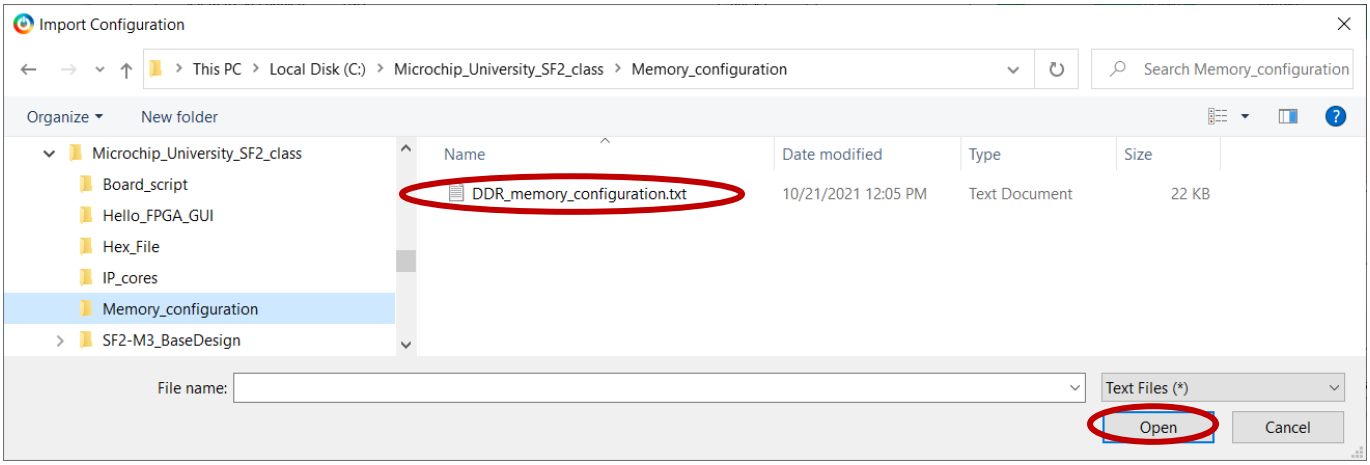


Step 4c: Select the MDDR tab of the Memories page. This tab is used to configure the SmartFusion® 2 DDR memory controller. This DDR controller is a hard block in the Microcontroller Subsystem rather than a soft DDR controller built with FPGA logic in the fabric.

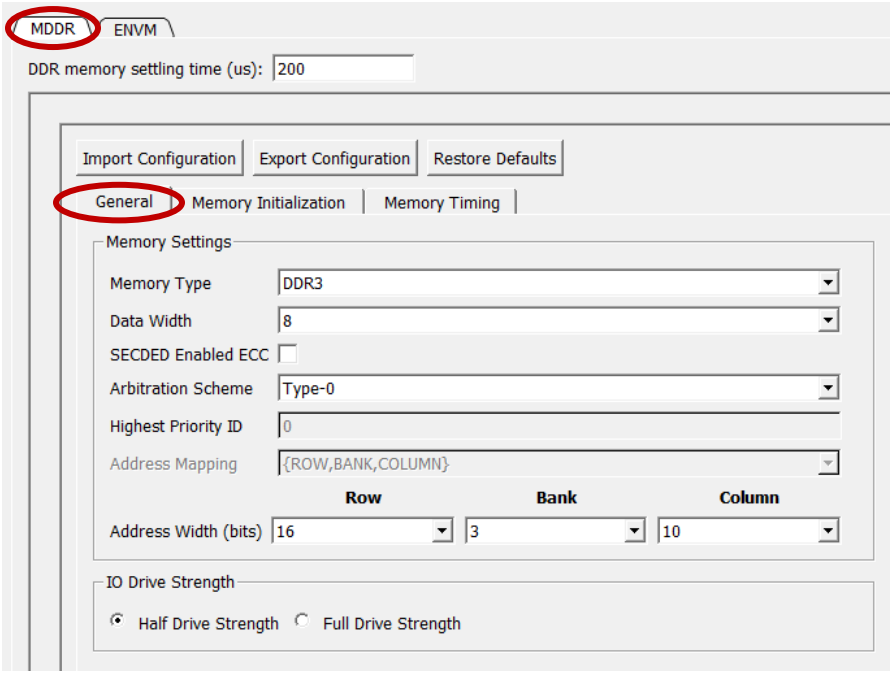
This Page has tabs for configuring the type of DDR memory, the DDR memory initialization settings, and the DDR memory timing. Values can be entered manually, or they can be imported from a file. Settings edited manually can be exported for future use.



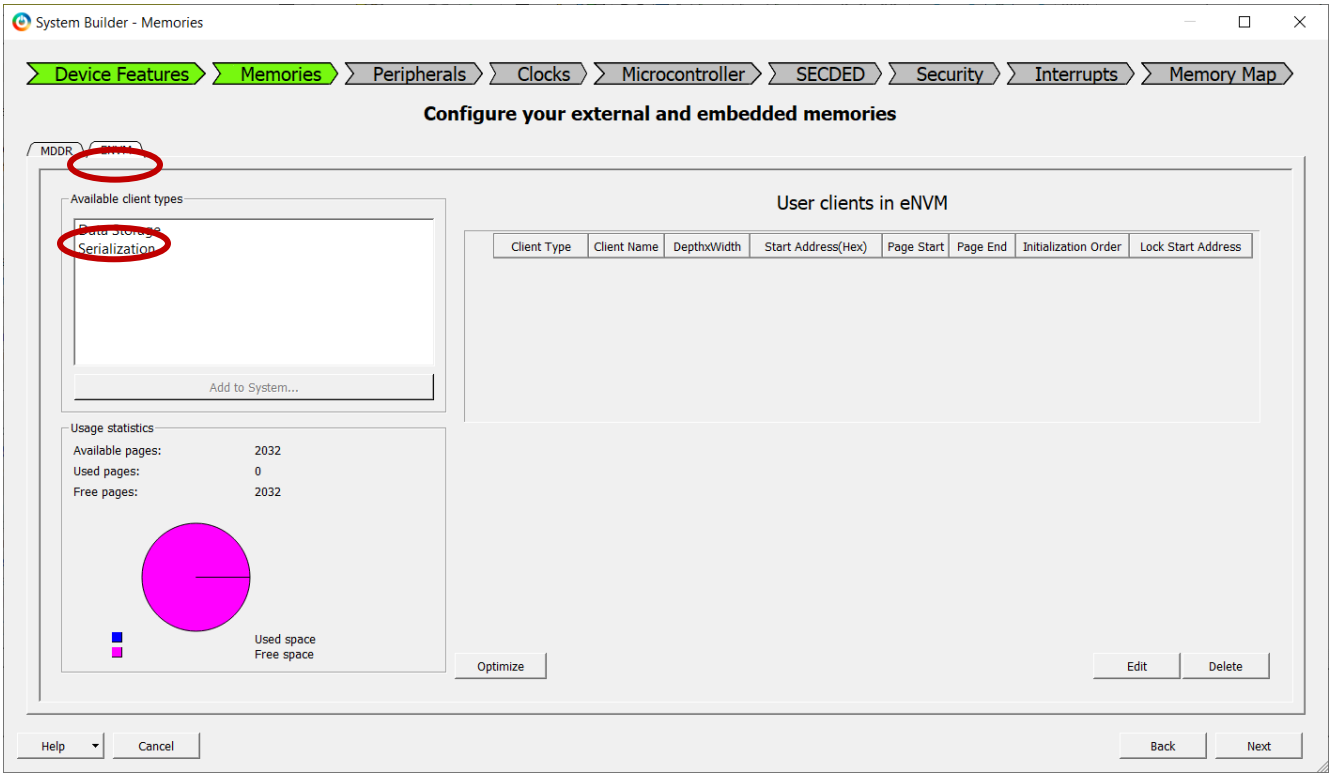
Step 4d: A memory configuration file that configures the DDR controller to match the DDR3 memory on the Hello FPGA kit is included in the lab files. Click the **Import Configuration** button. Navigate to the .\Microchip_University_SF2_class\Memory_configuration folder and select DDR_Memory_configuration.txt, then click **Open**.



Step 4e: Select the MDDR General tab. Confirm the settings match the figure below.



Step 4f: Select the ENVM tab of the Memories page. This tab is used to create SmartFusion® 2 Embedded Non-volatile memory (eNVM) partitions. Here you will create a partition to store the Cortex®-M3 application program.



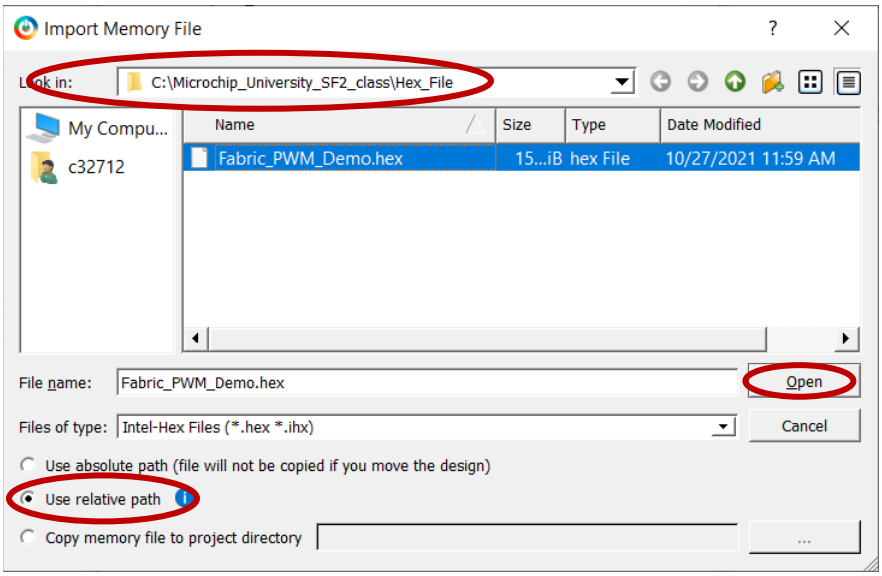
Step 4g: Double-click Data Storage under Available client types (highlighted in the figure above). The Add Data Storage Client dialog box will open.

Step 4h: Enter the following in the Add Data Storage Client dialog box:

- Client name: PGM_store
- Content from file: Click the browse button. The Import Memory File dialog box will open.

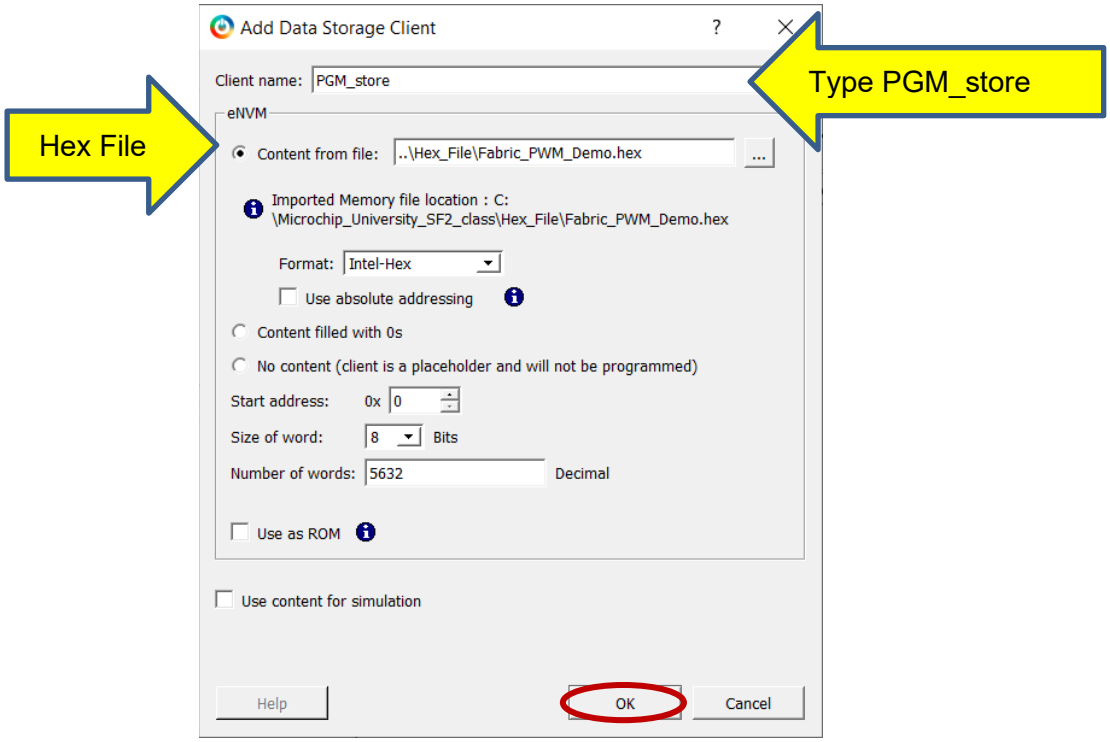
Step 4i: Enter the following in the Import Memory File dialog box:

- Look in: .\Microchip_University_SF2_class\Hex_File folder
- Select the following:
  - Fabric_PWM_Demo.hex This file configures a fabric PWM that drives LEDs on the Hello FPGA board.
  - Use relative path

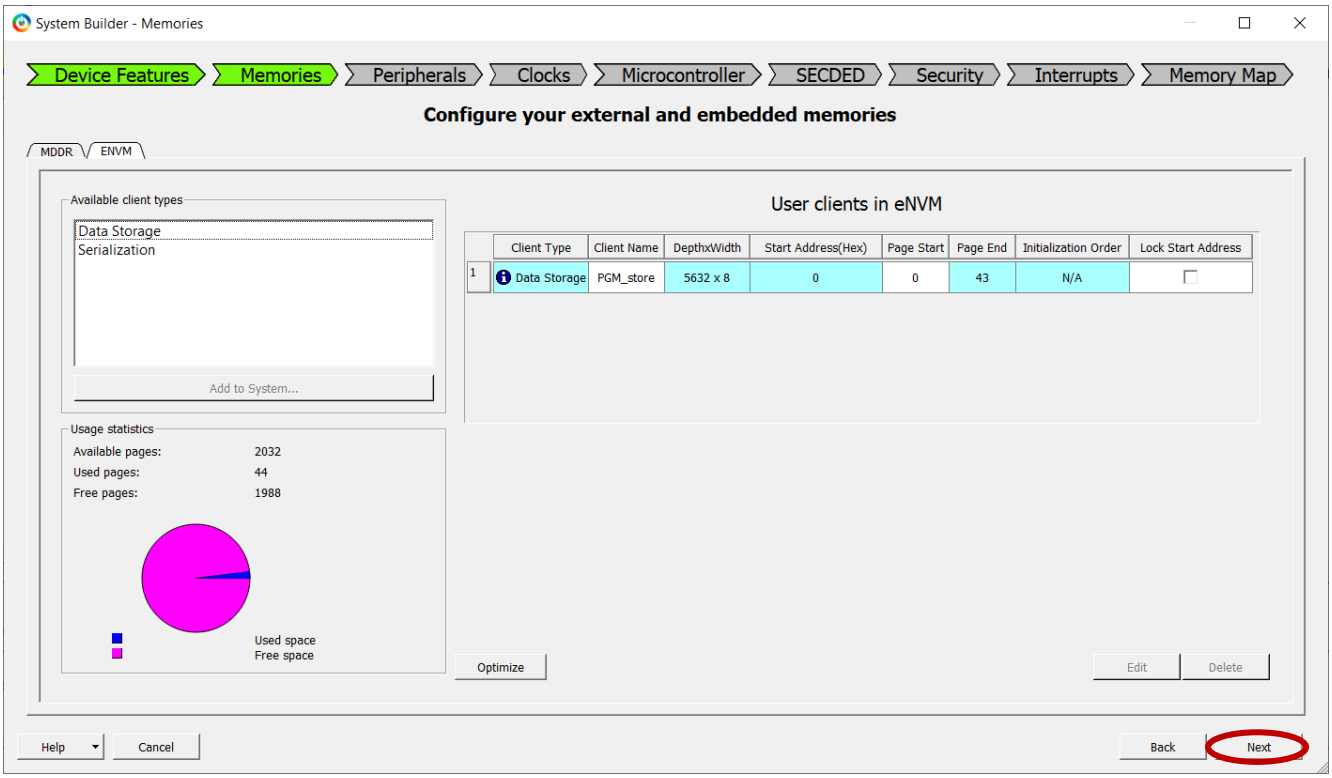


Step 4j: Click **Open** in the Import Memory File dialog box.

Confirm that the Add Data Storage Client dialog box appears as shown below.



Step 4k: Click **OK** to close the Add Data Storage dialog box. The eNVM client will be visible in the Memories page.

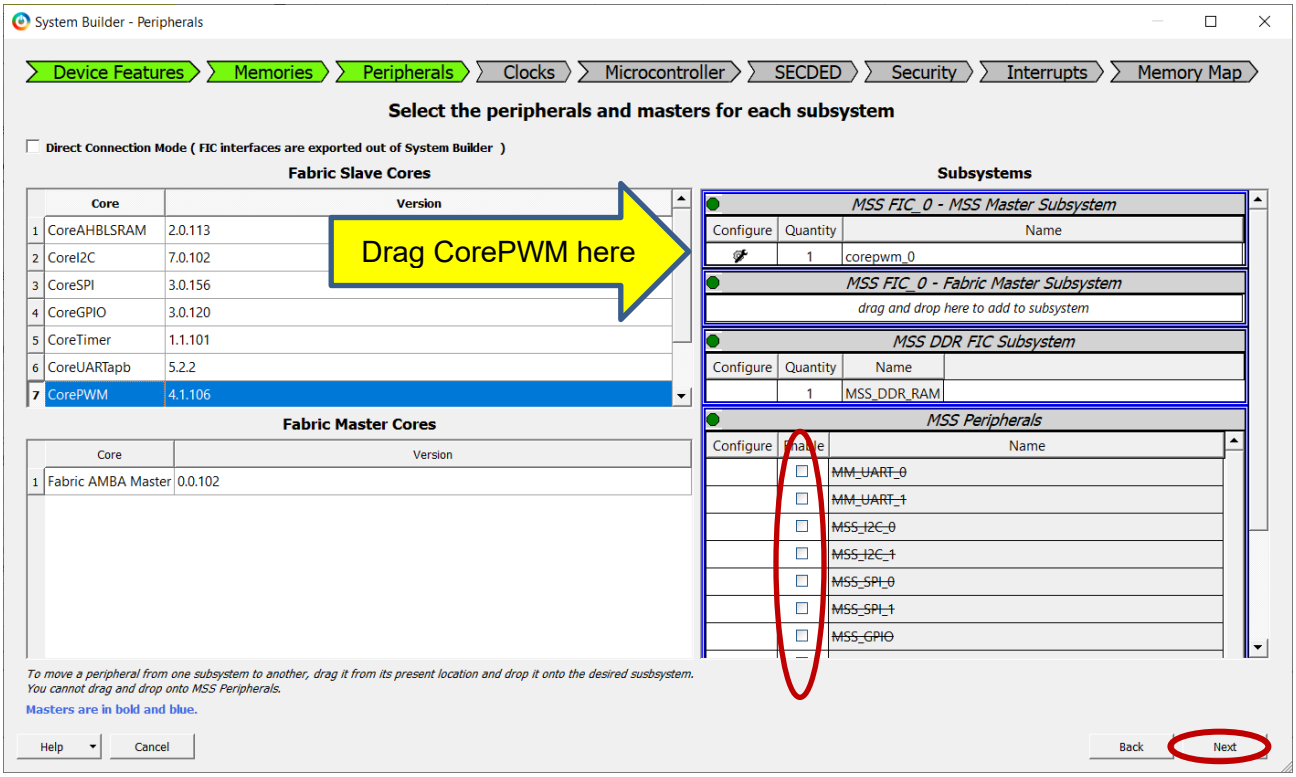


Step 4l: Click **Next**.

The Peripherals page will open. Here you can enable or disable MSS peripherals and add fabric peripherals. Configure peripherals by clicking the wrench symbol ( ).

Step 4m: Disable the MM_UART_0, MM_UART_1, MSS_I2C_0, MSS_I2C_1, MSS_SPI_0 and MSS_SPI_1 peripherals by un-checking the box in the Enable column. These peripherals are not used in the lab.

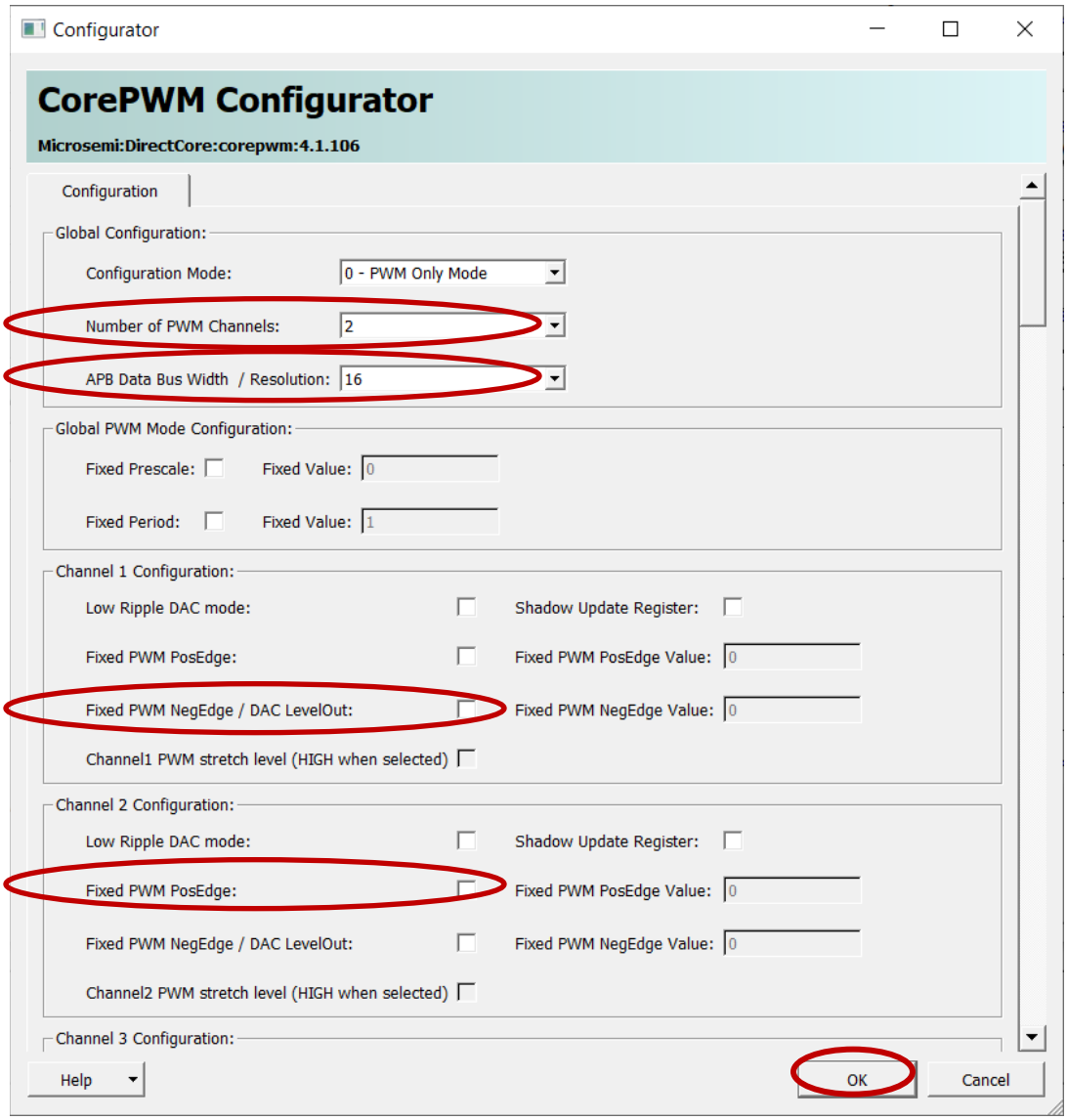
Step 4n: Drag an instance of CorePWM v4.1.106 from the Fabric Slave Cores to the MSS FIC_0 – MSS Master Subsystem. This will add an instance of a soft PWM (CorePWM) in the FPGA fabric and map it into the Cortex®-M3 memory space. CorePWM will be used to drive LEDs on the Hello FPGA board. Note that CorePWM v4.1.106 must be used with System Builder. The core is provided in the \Microchip_University_SF2_class\IP_cores folder.



Step 4o: Configure the soft PWM core to have two outputs with a 16-bit resolution. Click the wrench symbol ( ) next to corepwm_0 in the MSS FIC_0 – MSS Master Subsystem to open the CorePWM configurator. Enter the following in the CorePWM Configurator dialog box:

- Global Configuration:
  - Configuration Mode: 0 – PWM Only Mode (default)
  - Number of PWM Channels: 2
  - APB Data Bus Width / Resolution: 16
- Global PWM Mode Configuration:
  - Fixed Prescale: un-checked
  - Fixed Value: 0 (default)
  - Fixed Period: un-checked (default)
- Channel 1 / Channel 2 Configuration:
  - Fixed PWM PosEdge: un-checked

- Defaults for all other settings



Step 4p: Click **OK** to close the CorePWM Configurator.

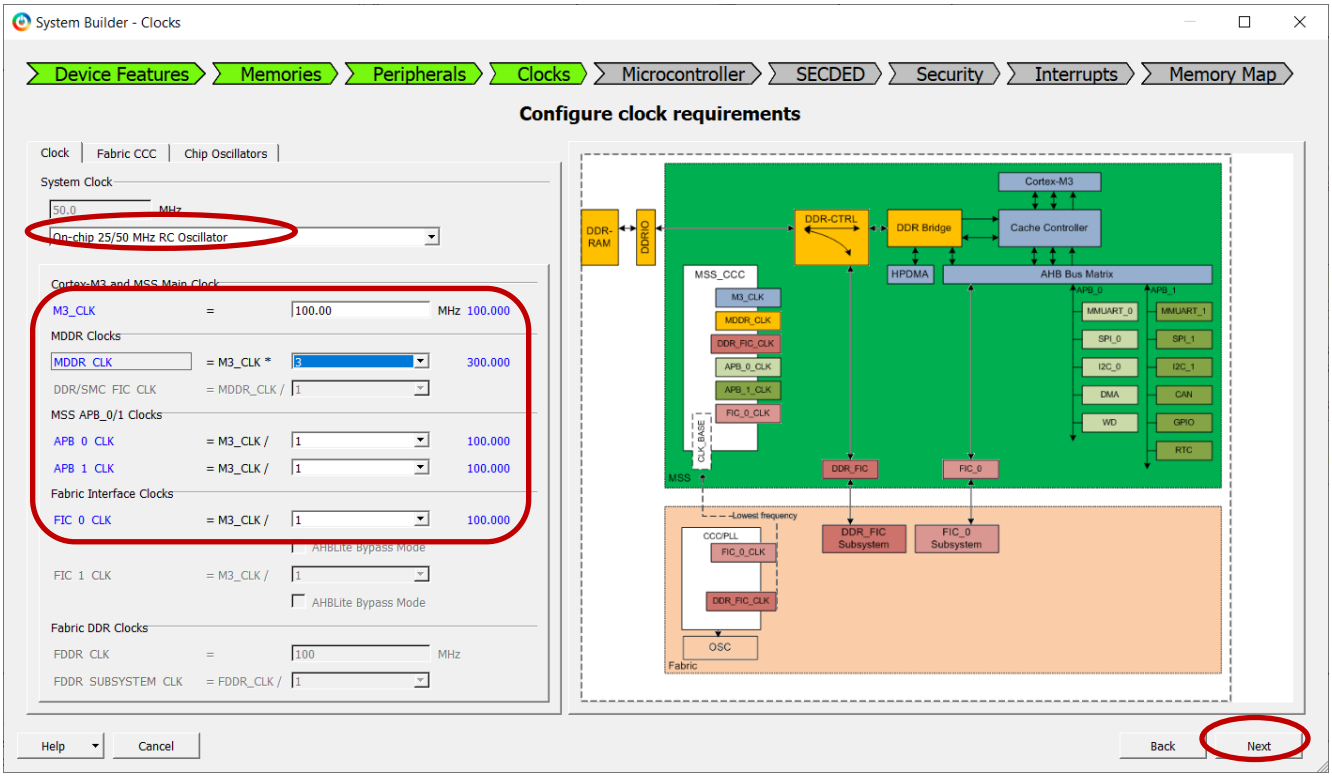
Step 4q: Click **Next** in the Peripherals page. The Clocks page will open. Use this page to specify the clock source and clock frequencies used in the design.

Step 4r: Enter the following in the Clocks page:

- System Clock: Select On-chip 25/50 MHz RC Oscillator from the pull-down menu. The internal 50 MHz oscillator will be the reference clock for the MSS PLL.
- M3_CLK: 100 MHz
- MDDR CLK[1]: M3_CLK*3 (300 MHz)
- APB_0 CLK: M3_CLK/1 (100 MHz) (default)
- APB_1 CLK: M3_CLK/1 (100 MHz) (default)
- FIC_0_CLK: M3_CLK/1 (100 MHz) (default)

[1]The MDDR clock is derived from the Cortex-M3 clock. Selecting 300 MHz supports 600 Mbps DDR memory operation.

Click the clock names in blue font in the Clocks page to highlight the clock source and logic it drives in the graphical view.



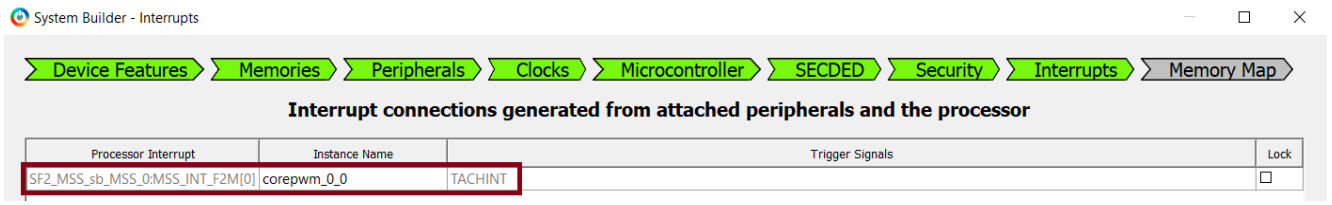
Step 4s: Click **Next**.

Step 4t: The Microcontroller page will open. Use this page to modify the Microcontroller configuration, enable and configure the instruction cache. Click **Next** to accept the default settings.

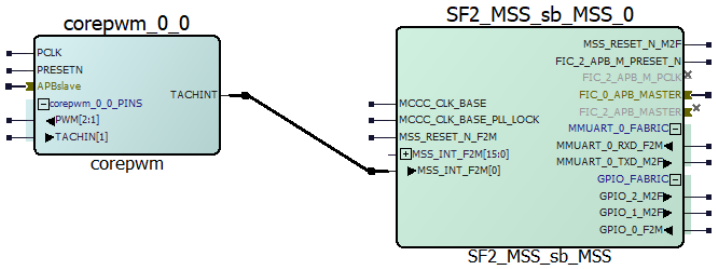
Step 4u: The SECDED page will open. Use this page to enable Error Correction for MSS memory blocks. Click **Next** to accept the default SECDED configuration (no Error Correction / Detection).

Step 4v: The Security page will open. The SmartFusion® 2 advanced security features are only available on “S” suffix parts. The Hello FPGA board does not use the “S” devices. Click **Next** to accept the default Security configuration.

Step 4w: The Interrupts page will open. This page displays the CorePWM TACHINT interrupt that connects to the Cortex®-M3 interrupt controller.

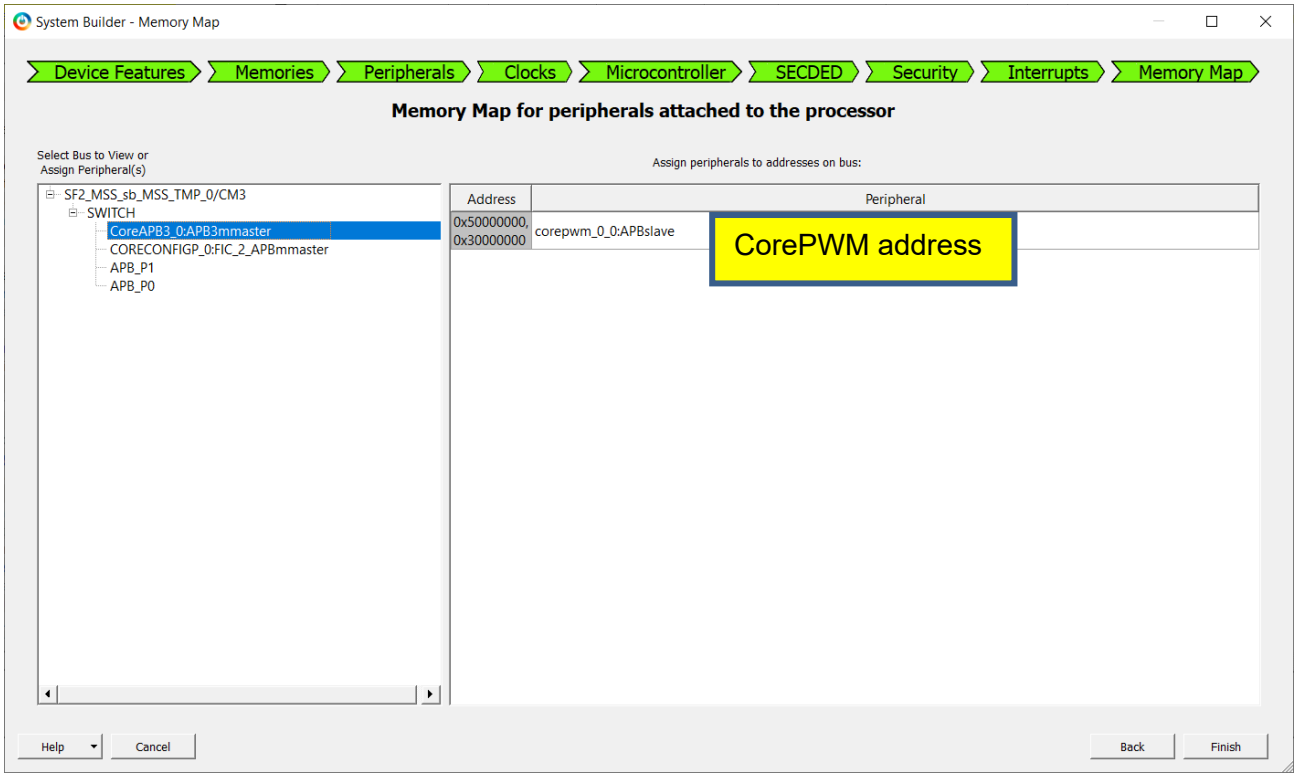


Note: the figure shown below shows the connection between the CorePWM TACHINT output and the SmartFusion2 MSS component. This figure does not appear in System Builder.

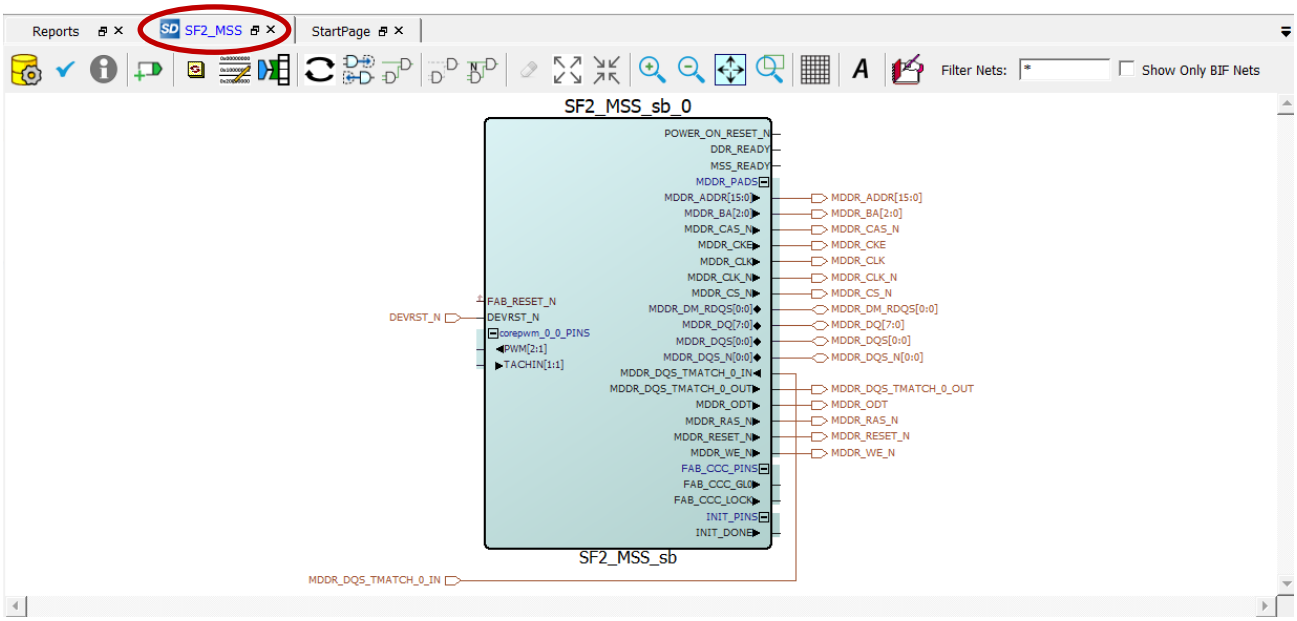


Step 4x: Click **Next**.

Step 4y: The Memory Map page will open. This page displays the addresses of the MSS peripherals and the memory mapped FPGA fabric peripherals. The corepwm is a target that will be configured by the CoreAPB3 initiator. Selecting the CoreAPB3_0:APB3mmaster will show the address for the fabric corepwm fabric peripheral.



Step 4z: Click **Finish** to create the system. The SmartDesign canvas will open showing a component named SF2_MSS_sb_0.



Step 4aa: Save the project (**Project > Save**). Do not close the Libero® SoC project.

Note: a zip file (SF2-M3_BaseDesign_Lab1.zip) containing the complete Lab 1 project is provided in the \Microchip_University_SF2_class\Solution folder.

Results:

If everything worked as expected, the SmartDesign canvas will appear as shown in the figure above.

If the ports for the SF2_MSS_sb_0 component do not appear as shown in the figure on the previous page, open System Builder and confirm the settings match the instructions in this lab guide.

Summary:

In this lab you used System Builder to configure the SmartFusion® 2 Microcontroller Subsystem. The configuration included configuring the DDR controller, creating a partition in the SmartFusion 2 eNVM and configuring the MSS clocking. You also added a PWM peripheral in the SmartFusion 2 FPGA fabric. System Builder automatically adds all the required logic and bus interfaces to configure the MSS per your specification. With this skill/knowledge, you can configure other MSS implementations and add additional fabric peripherals, including your own fabric peripherals.

Lab 2 – Completing the design and running the application

Purpose:

In this lab you will learn how to:

- Use SmartDesign to make connections to SF2_MSS component ports in the design.
- Constrain the design for synthesis, and place and route.
- Generate a programming bitstream and program the SmartFusion® 2 silicon.

Overview:

After configuring the SmartFusion 2 MSS with System Builder, some port connections are required. After making the port connections, the next steps are generating timing constraints, synthesis, generating IO constraints, running place and route and programming**.**

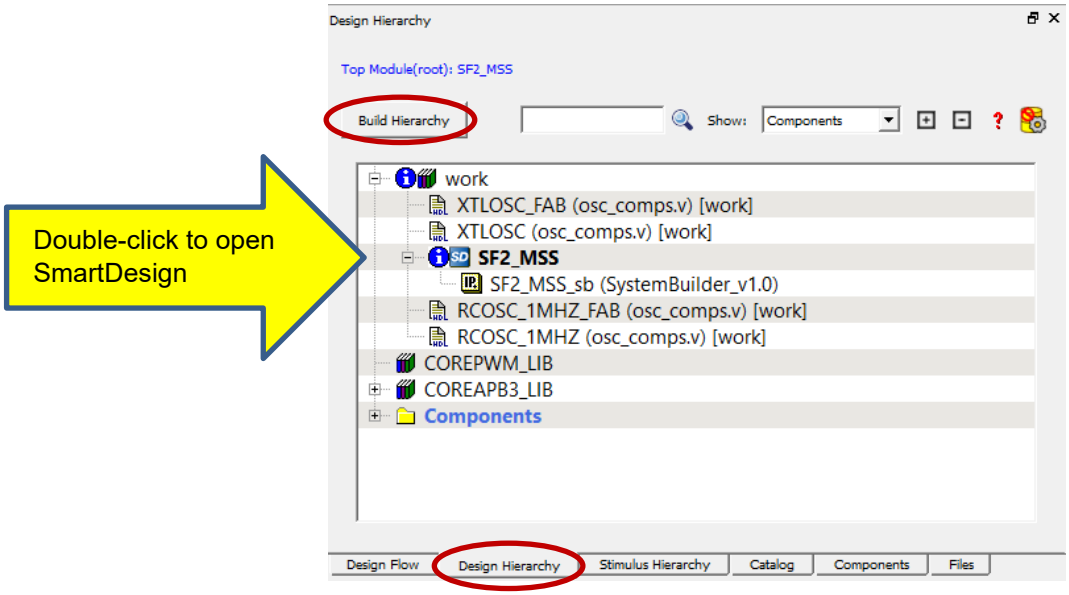
Procedure:

STEP 1: Making connections in SmartDesign

Step 1a: Select Libero® SoC. Open the project if necessary (**Project > Open**). Navigate to the project location (\Microchip_University_SF2_class\SF2-M3_BaseDesign) and select SF2-M3_BaseDesign.prjx to open the project.

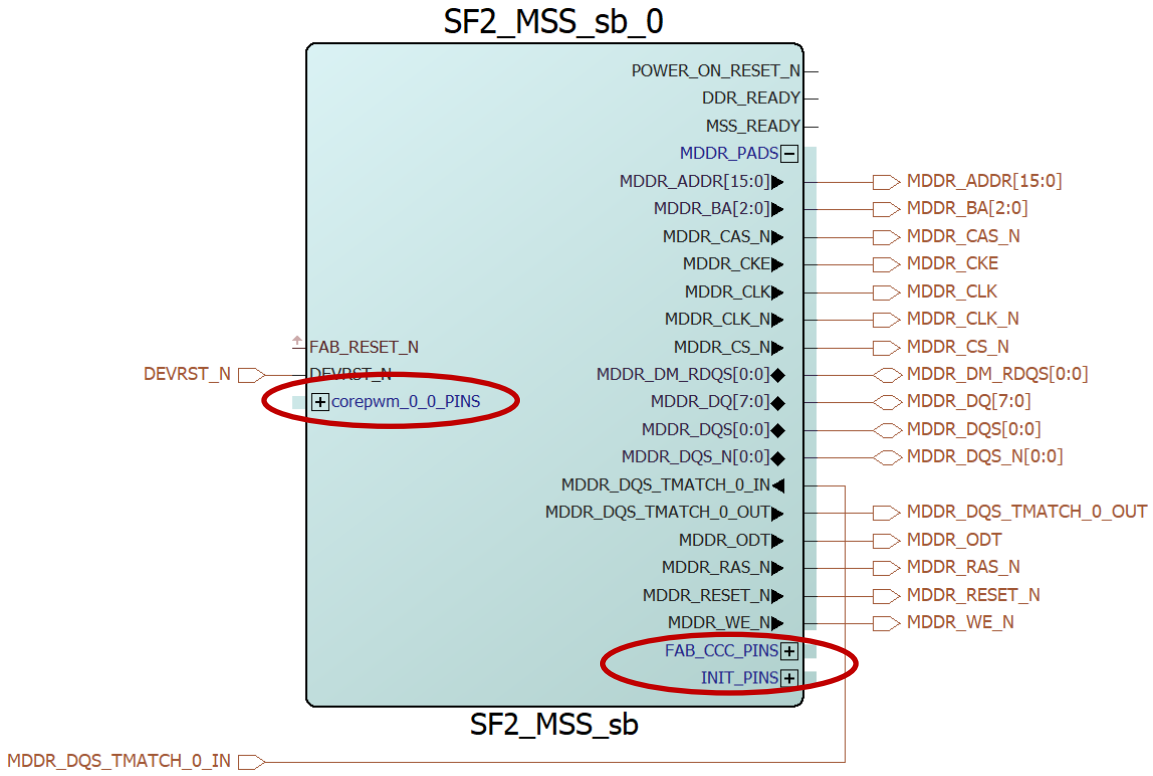
Step 1b: Select the SF2_MSS tab in the Libero SoC GUI (highlighted in the figure on the previous page). If the tab is not open, double click SF2_MSS on the Libero SoC Design Hierarchy tab.

Step 1c: Click the Build Hierarchy button.

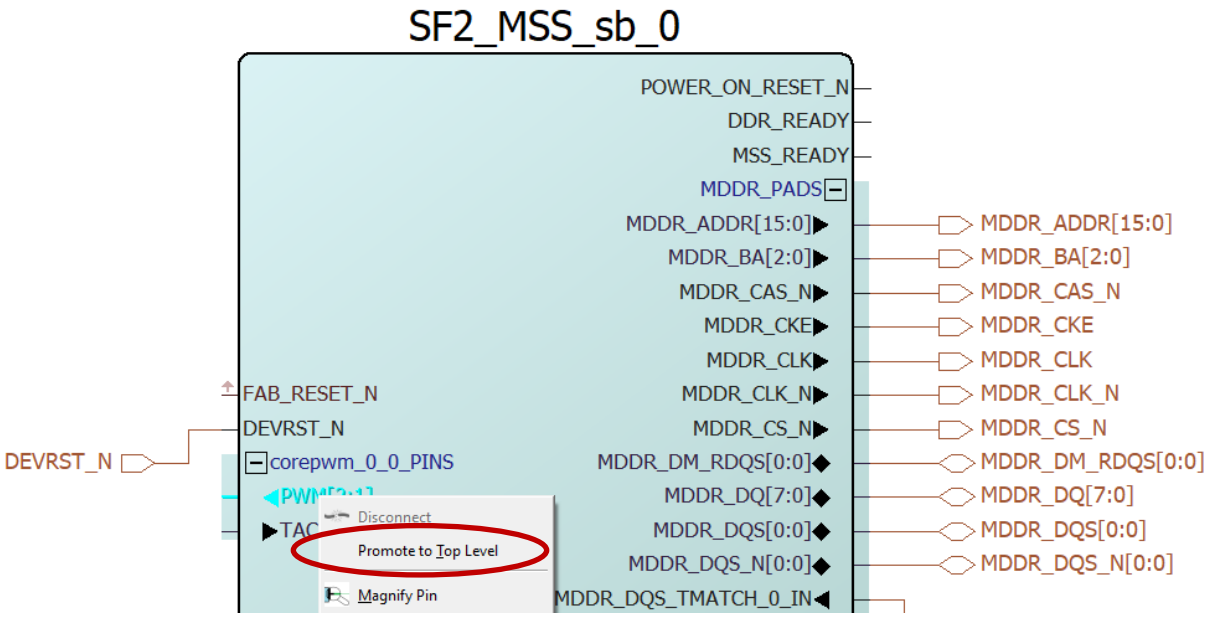


Step 1d: Maximize the SmartDesign working area by selecting **View > Maximize Work Area**.

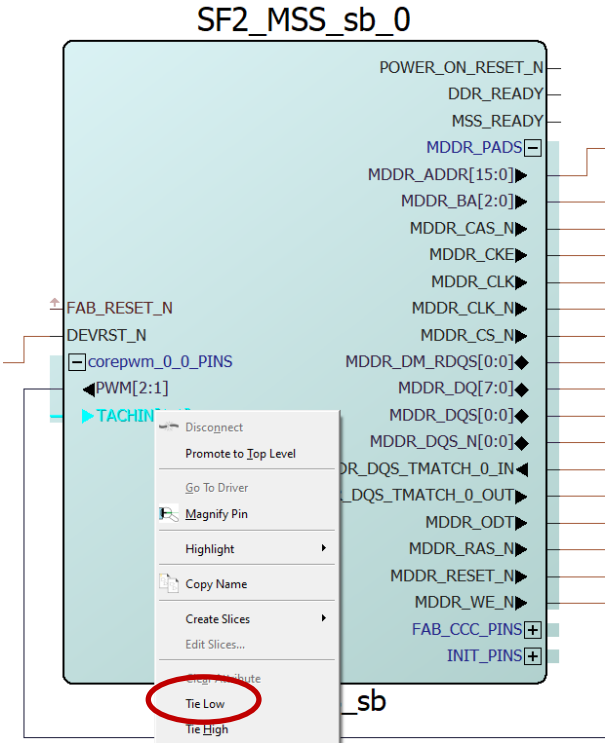
Step 1e: Click the + sign next to the corepwm_0_0_PINS, FAB_CCC_PINS and INIT_PINS to expand the pin groups if they are not already expanded.



Step 1f: Promote the PWM[2:1] port to the top level by selecting the port, right-clicking and selecting **Promote to Top Level**. These ports are the PWM outputs. They will be connected to pins on the SmartFusion® 2 device to drive LEDs on the Hello FPGA board.

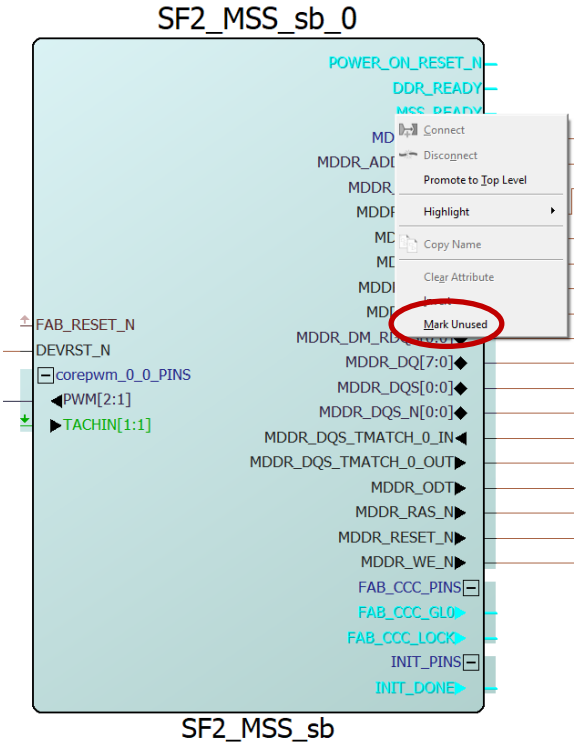


Step 1g: Tie the TACHIN[1] port low by selecting the port, right-clicking and selecting **Tie Low**.

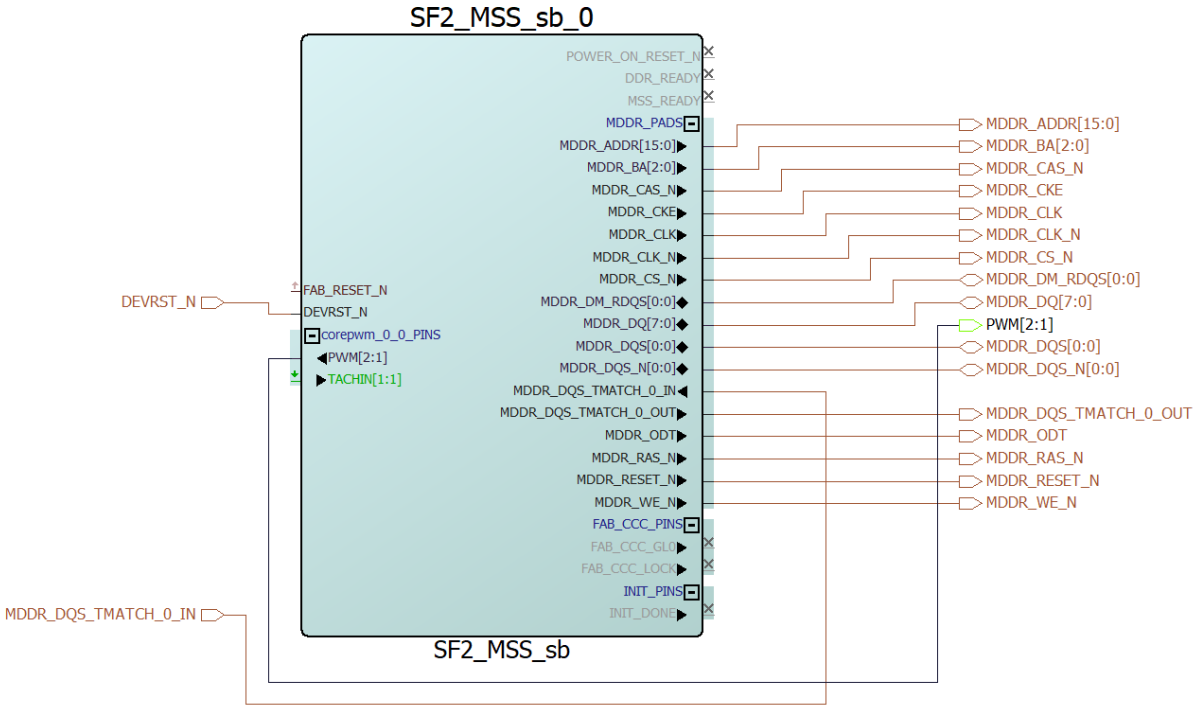


Step 1h: Mark the output ports POWER_ON_RESET_N, DDR_READY, MSS_READY, FAB_CCC_GL0, FAB_CCC_LOCK and INIT_DONE unused by selecting the port, right-clicking and selecting **Mark Unused**. These ports are not used in the design. Adding the Unused property to unused ports in the design suppresses Warning messages when the design is generated.

Tip: hold the CTRL key and select the ports before right-clicking.



After making the pin connections the SF2_MSS_sb_0 component will look like the figure below.

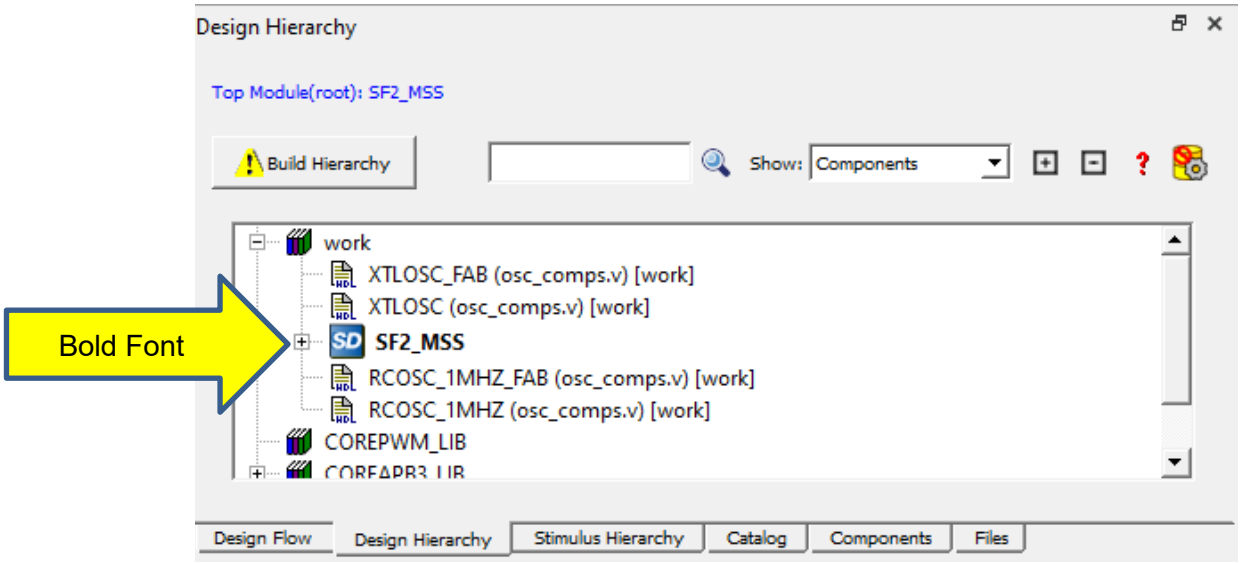


Step 1i: Restore the work area (**View > Restore Work Area**). Generate the design by clicking **SmartDesign > Generate Component** or by clicking the Generate Component icon on the SmartDesign toolbar ( ).

Select the Libero® SoC Log tab. The message “Successfully generated component ‘SF2_MSS’.” Will appear indicating the design was generated without any errors.

Step 1j: Close SmartDesign (**File > Close SF2_MSS**).

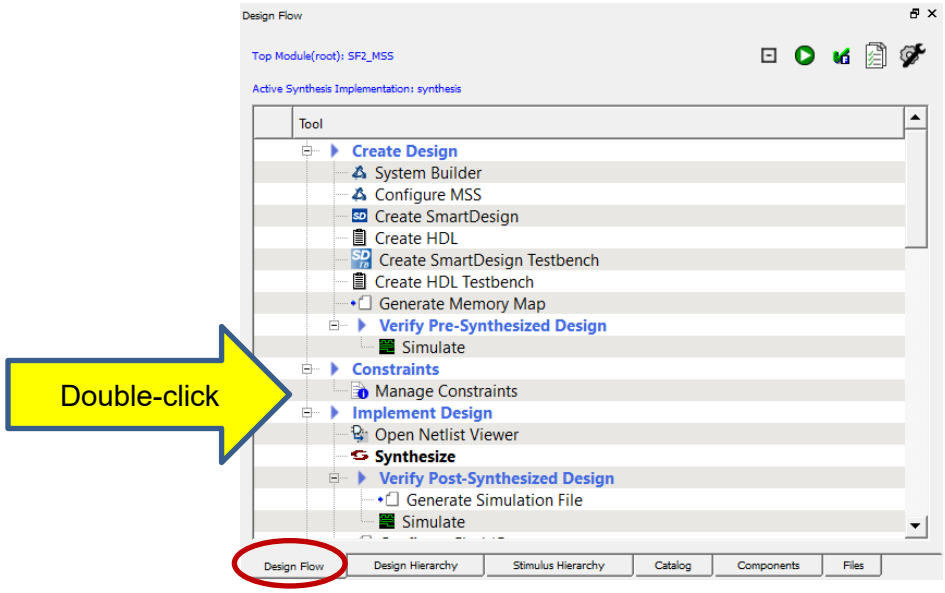
Step 1k: Confirm that SF2_MSS appears in bold font on the Libero SoC Design Hierarchy tab. If it does not, select SF2_MSS, right-click and select **Set As Root**.


Design Hierarchy
Top Module(root): SF2_MSS
Build Hierarchy
Show:
Components
work
XTLOSC_FAB (osc_comps.v) [work]
XTLOSC (osc_comps.v) [work]
SF2_MSS
RCOSC_1MHZ_FAB (osc_comps.v) [work]
RCOSC_1MHZ (osc_comps.v) [work]
COREPWM_LIB
Design Flow
Design Hierarchy
Stimulus Hierarchy
Catalog
Components
Files
Bold Font

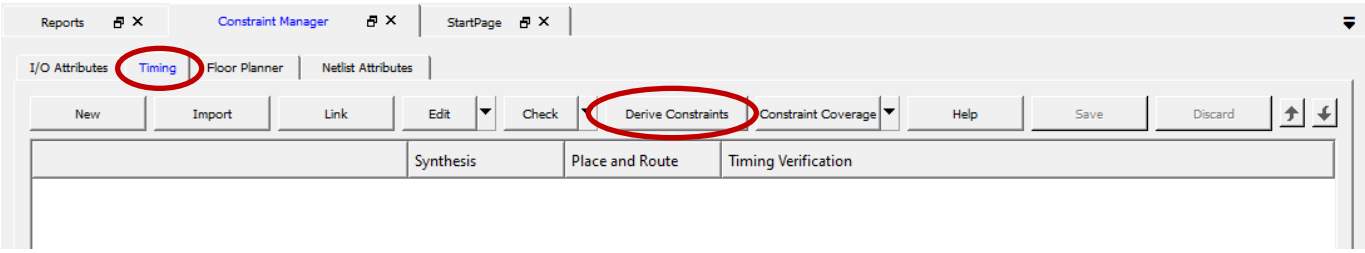
STEP 2: Generating Timing Constraints

The next step is to generate timing constraints for synthesis, Place and Route and Timing Verification.

Step 2a: Select the Libero® SoC Design Flow tab. Expand Constraints and double-click Manage Constraints.

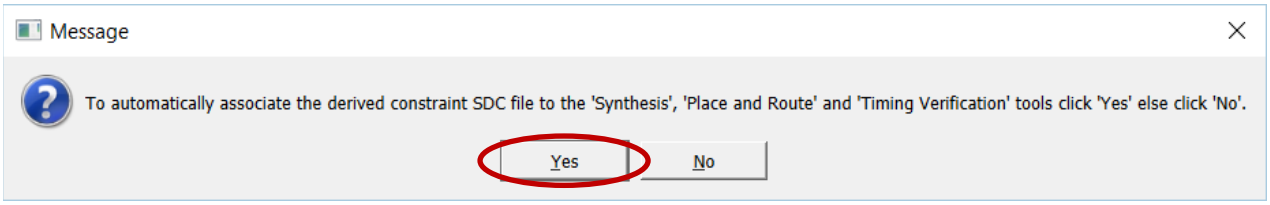


Step 2b: Select the Constraint Manager Timing tab.

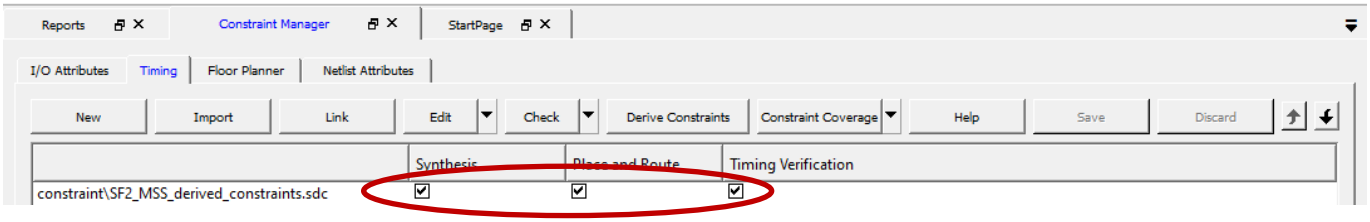


Libero SoC automatically creates timing constraints for known blocks such as Oscillators and PLL outputs. This step will create timing constraints for the SmartFusion® 2 50 MHz oscillator and the clocks that were configured on the System Builder Clocks page.

Step 2c: Click **Derive Constraints** (highlighted above). Click on **Yes** in the Message window to automatically associate the derived constraints SDC file to the 'Synthesis', 'Place and Route' and 'Timing Verification'.



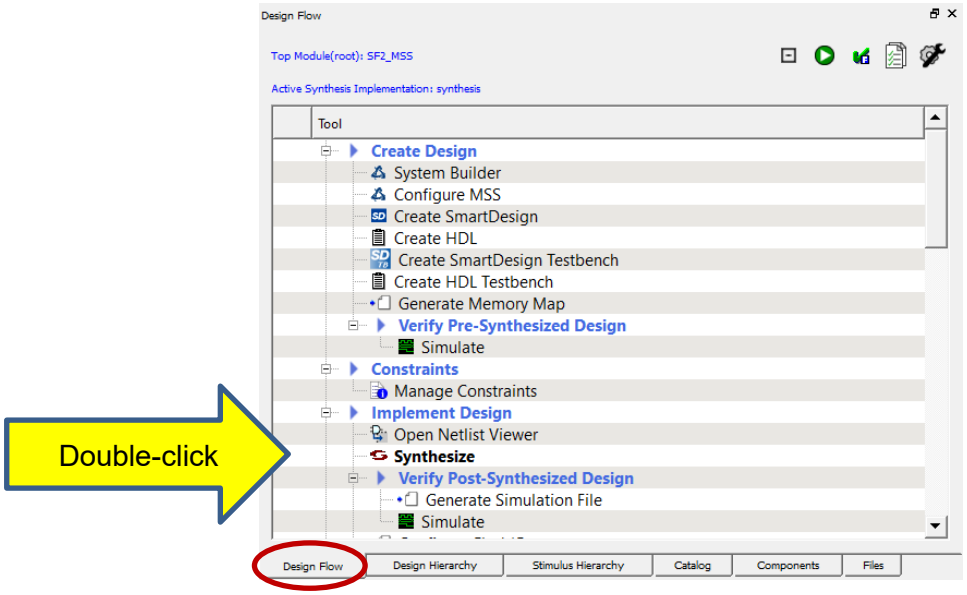
Step 2d: A constraint file named SF2_MSS_derived_constraints.sdc will be visible. Confirm that the file is checked for Synthesis, Place and Route and Timing Verification.



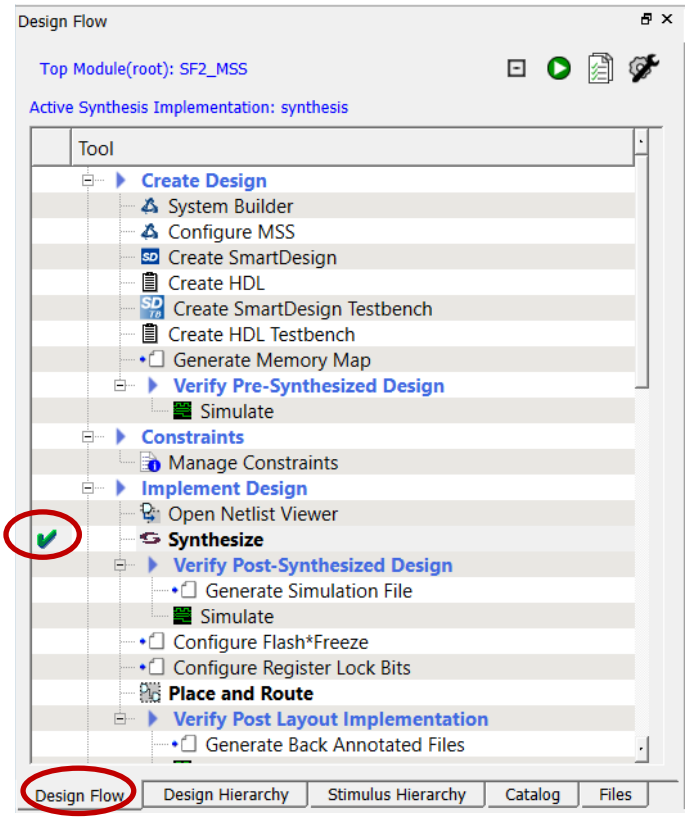
STEP 3: Running Synthesis

The next step is to synthesize the design with Synplify Pro® ME to generate a netlist for place and route.

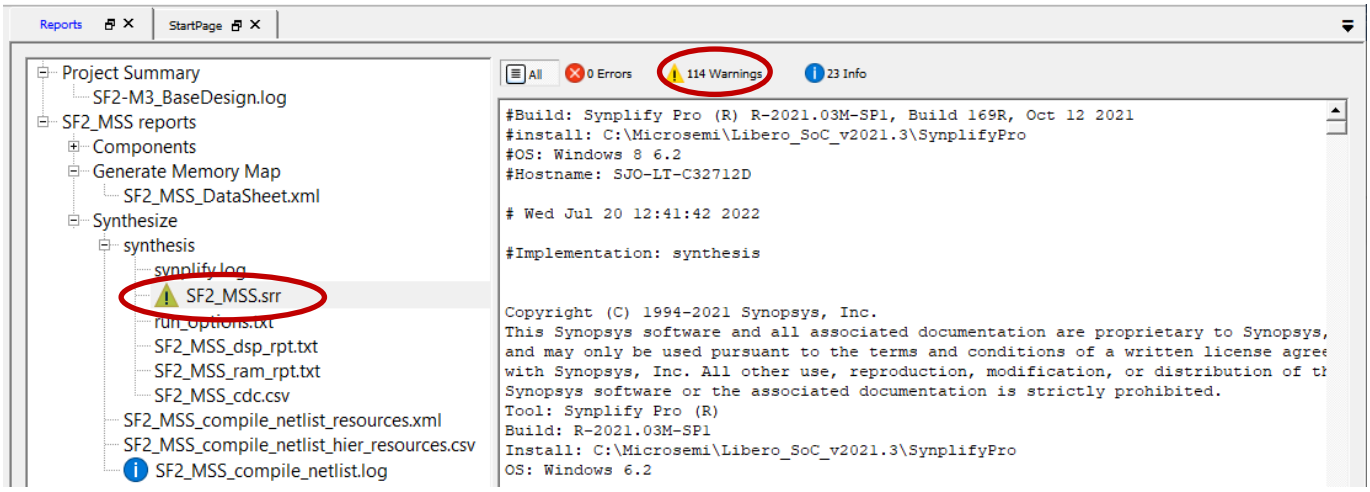
Step 3a: Double-click **Synthesize** on the Libero® SoC Design Flow tab to synthesize the design with Synplify Pro® ME.



After a few minutes, a green check mark will appear in the Design Flow tab indicating the synthesis step completed successfully.



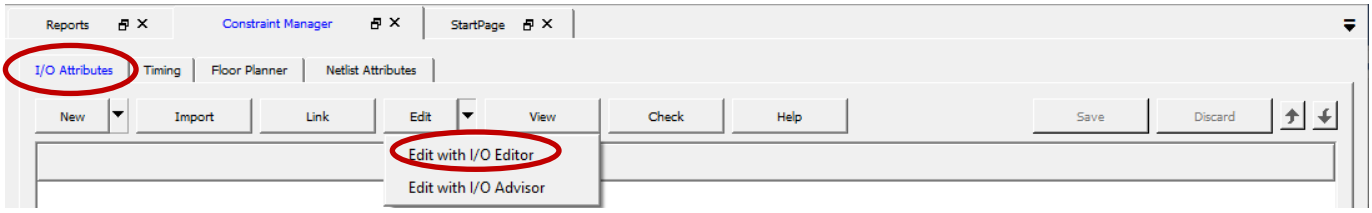
Note: synthesis warning messages can be ignored.



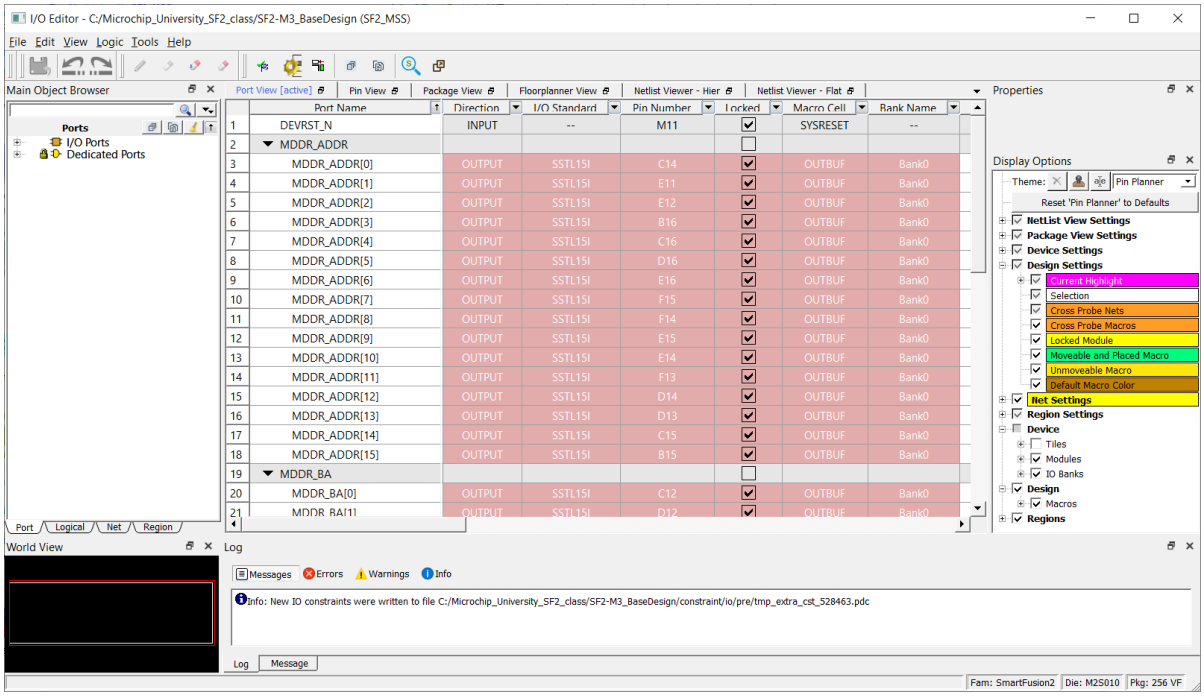
STEP 4: Entering I/O Constraints

Enter I/O constraints prior to running place and route. There are multiple ways to make I/O Assignments. In this lab, we will use the Libero® SoC I/O Constraint Editor.

Step 4a: Select the I/O Attributes tab in the Constraints Manager. If the Constraint Editor is not visible, double-click Manage Constraints on the Design Flow tab. Use the pull-down menu to open the I/O Constraint Editor.

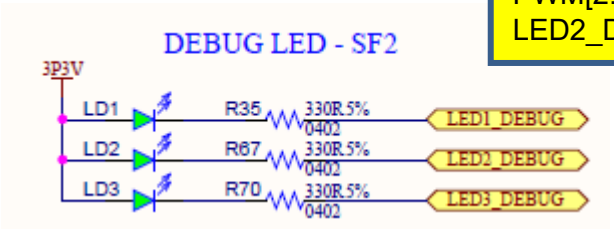


The Libero SoC I/O Editor will open.



The figure below is from the Hello FPGA board schematic and does not appear in Libero SoC.

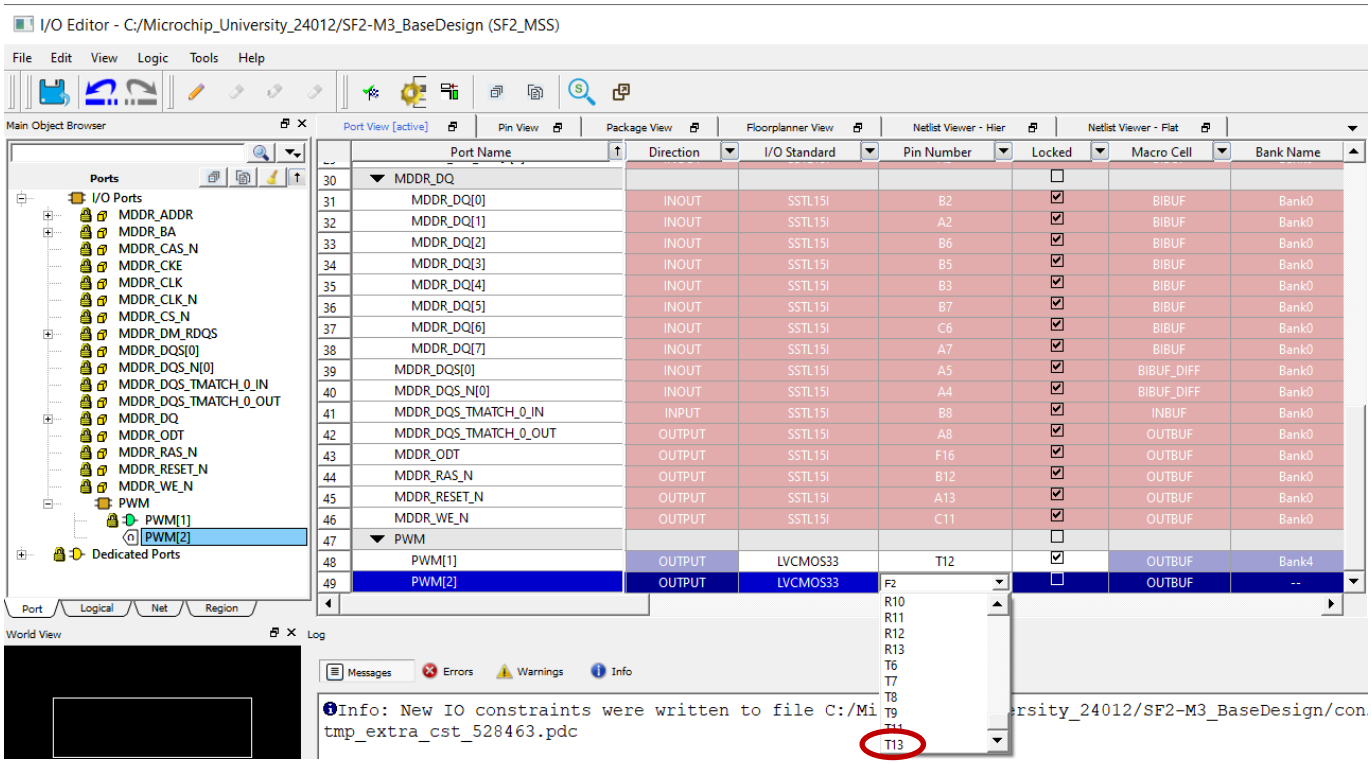
PWM[2:1] connect to LED1_DEBUG and LED2_DEBUG on the Hello FPGA board.



SmartFusion® 2 I/O banks support multiple I/O standards. The Libero SoC I/O Editor ensures that only I/Os with compatible standards can be assigned in the same I/O bank.

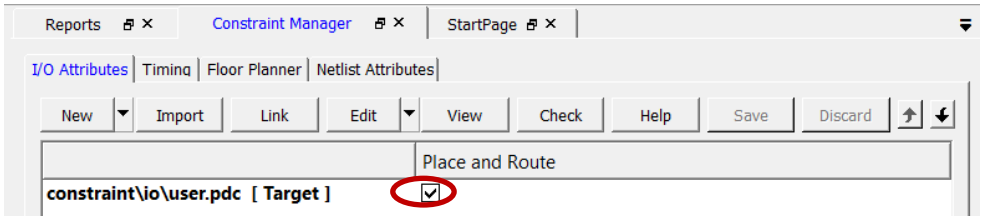
Step 4b: Select the Port View tab in the IO Editor. Scroll to the bottom of the Port list. Use the pull-down menu to make the following assignments. If the I/O standard is not LVCMOS33, change the I/O Standard before changing the Pin Number.

Port	**I/O Standard**	**Pin Number**
PWM[1]	LVCMOS33	T12
PWM[2]	LVCOMS33	T13

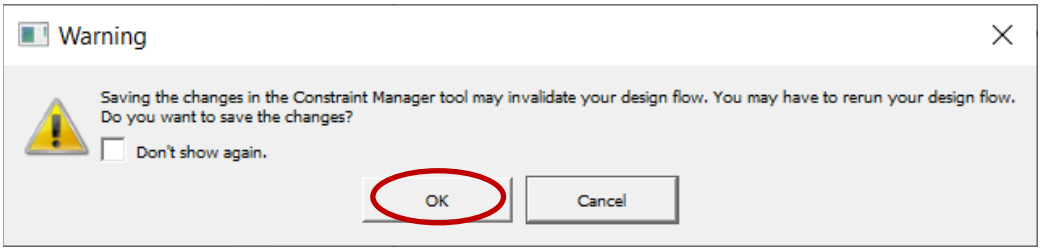


Step 4c: Save the changes (**File > Commit**) and close the I/O Attribute Editor (**File > Exit**).

Step 4d: An I/O constraint file named user.pdc will be visible on the Libero® SoC Constraint Manager I/O Attributes tab. Confirm that the file is checked for Place and Route. Check the box if it is not checked then click **Save**.

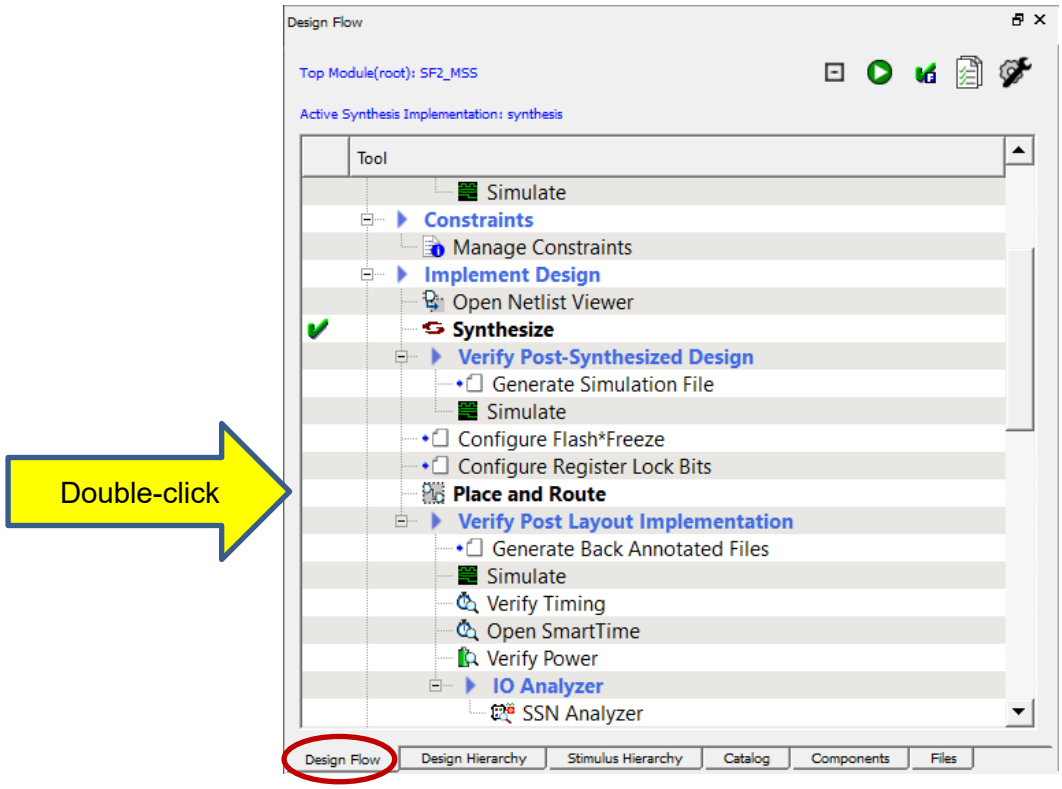


Step 4e: Click **OK** if the Warning dialog box opens when you select Save after checking user.pdc for Place and Route.

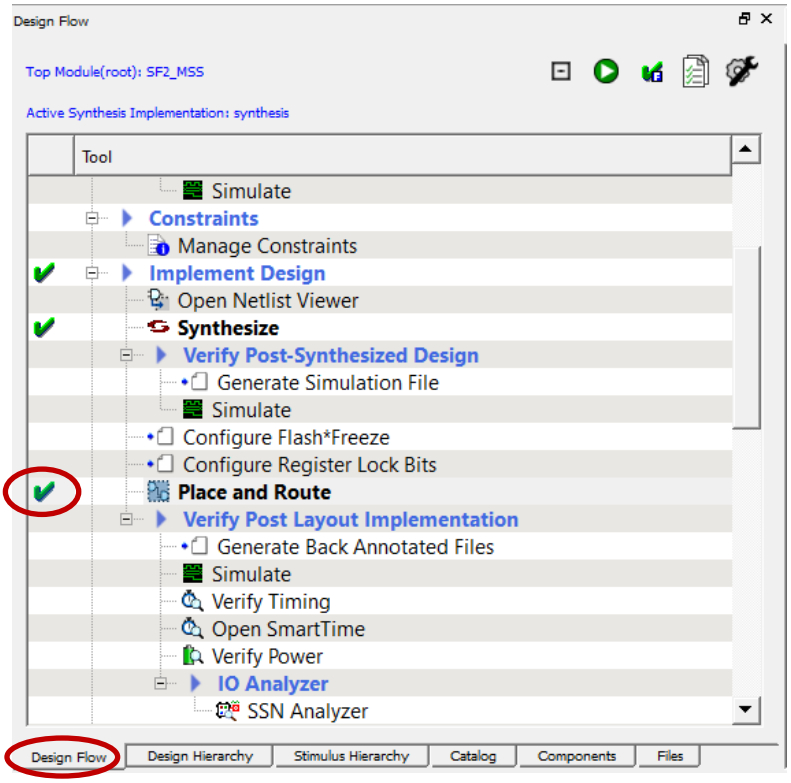


STEP 5: Running Place and Route

Step 5a: Double-click Place and Route in the Design Flow window to run place and route.



After a few minutes, a green check mark will appear on the Design Flow tab indicating the Place and Route step completed successfully.

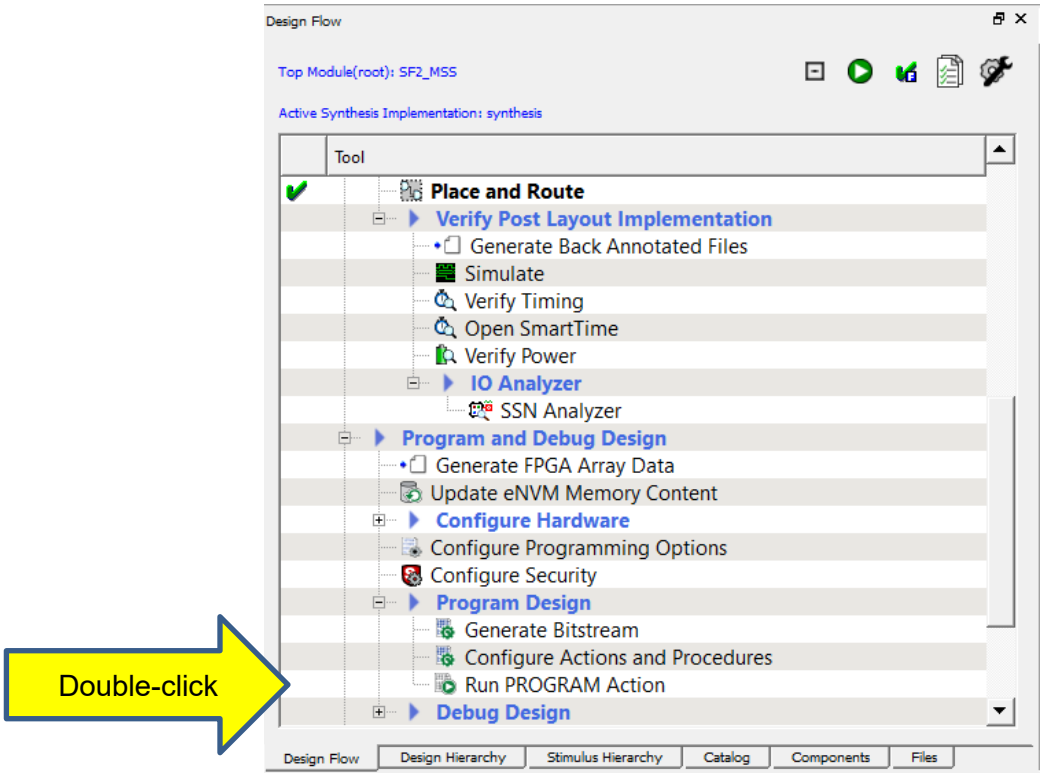


STEP 6: Bitstream Generation and Programming (Requires a FlashPro programmer)

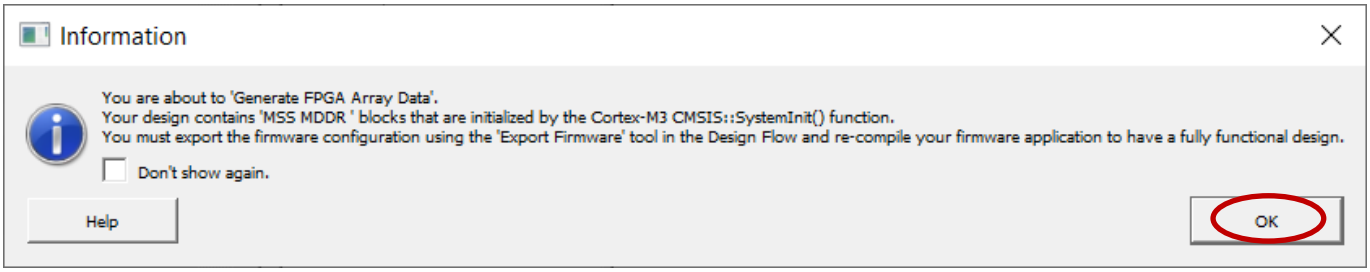
The next step is to generate the bitstream and program the SmartFusion® 2 FPGA on the Hello FPGA board.

Prior to programming, confirm the Hello FPGA board connections as described in Appendix C.

Step 6a: Double-click **Run PROGRAM Action** in the Design Flow window to generate the bitstream and program the SmartFusion 2 FPGA.

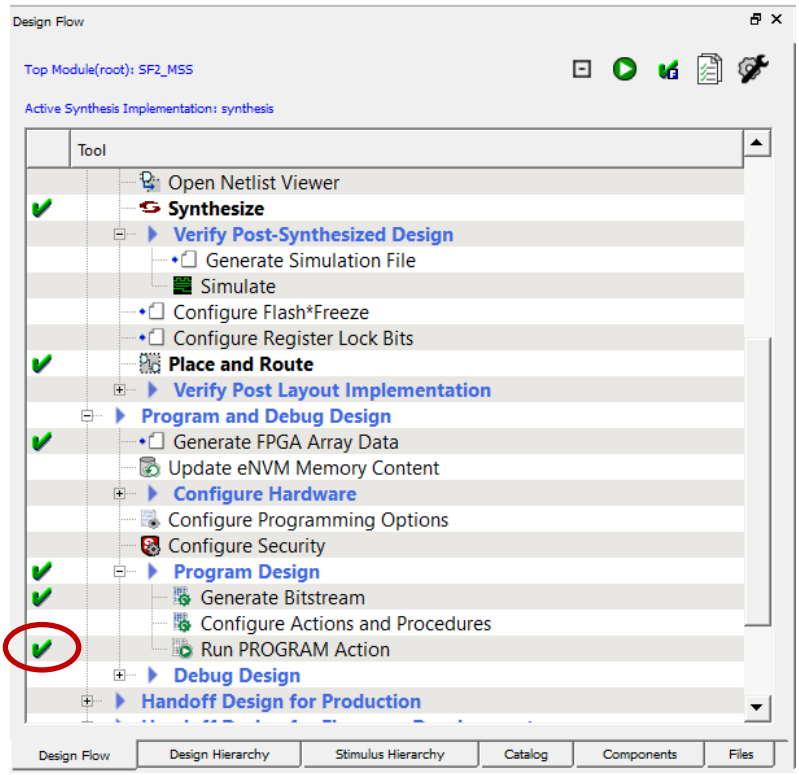


Step 6b: Click **OK** in the Information dialog box. The bitstream will be generated and the FPGA will be programmed.



Note: Do not interrupt the programming sequence.

A green check mark will appear next to Programming on the Design Flow tab indicating that programming was successful.



A message will appear in the Libero® SoC Log window indicating programming completed successfully. Note that the programmer number may differ.

```
Programmer 'E2001Q8YC9' : device 'M2GL010' : Executing action PROGRAM
PASSED.
```

Step 6c: Power cycle the board by removing then installing the USB cable at the USB Mini A connector on the Hello FPGA board.

Step 6d: Observe the Hello FPGA kit. LED1 and LED2 on the Hello FPGA board will toggle at different rates. The LEDs are driven by the PWM outputs.

Results:

After completing this lab, you have completed the FPGA design flow, generated a bitstream and programmed the SmartFusion® 2 FPGA on the Hello FPGA board.

If everything worked as expected, the LEDs should toggle as described above. If the LEDs are not toggling, go back to the Libero SoC project, open the IO Editor, and verify the pin assignments match the instructions on page 26. Correct the pin assignments if needed, then run place and route, generate the bitstream and program the SmartFusion 2 SoC FPGA.

Summary:

You have made connections to the ports of the SmartFusion 2 MSS, created timing and I/O constraints and completed the FPGA design flow. With this skill/knowledge, now you can create your own SmartFusion 2 project and implement a complete SmartFusion 2 design.

Lab 3 – Generating a Sample Firmware Project

Purpose:

In this lab step you will generate a sample firmware project and export the firmware configuration files for the design.

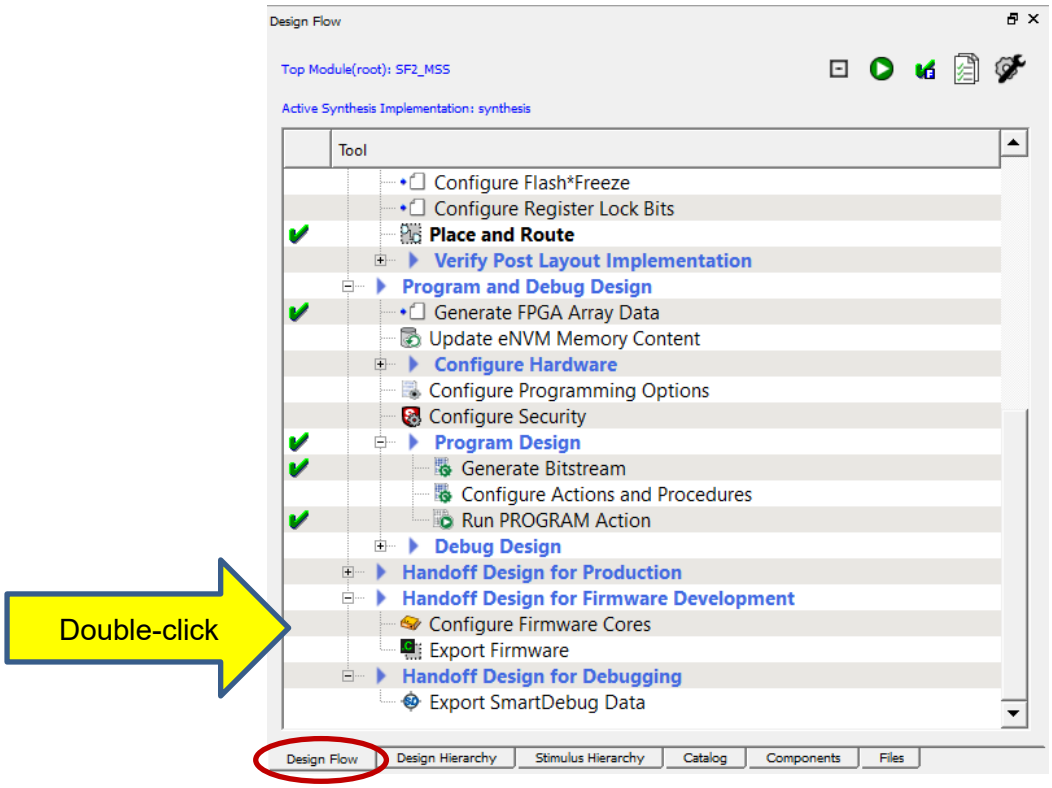
Overview:

After programming, the application runs from the SmartFusion® 2 Embedded Nonvolatile Memory (eNVM). Next you will generate a firmware project that will be opened in the Microchip SoftConsole software.

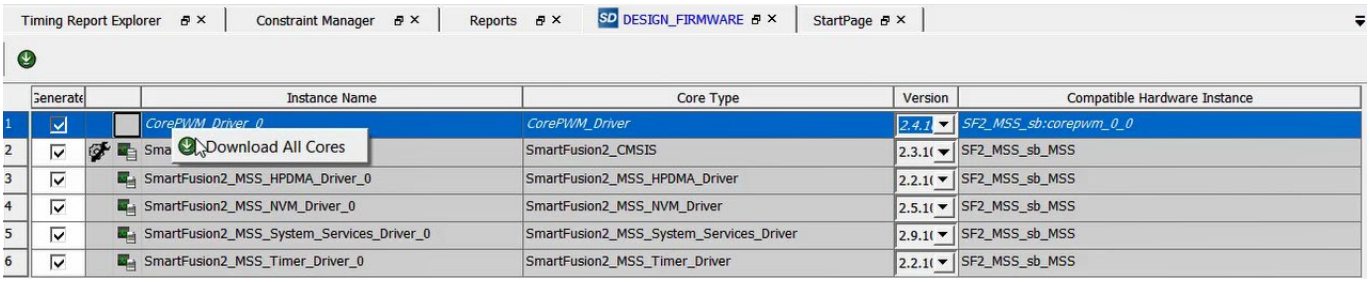
Procedure:

STEP 1: Generating a sample firmware project

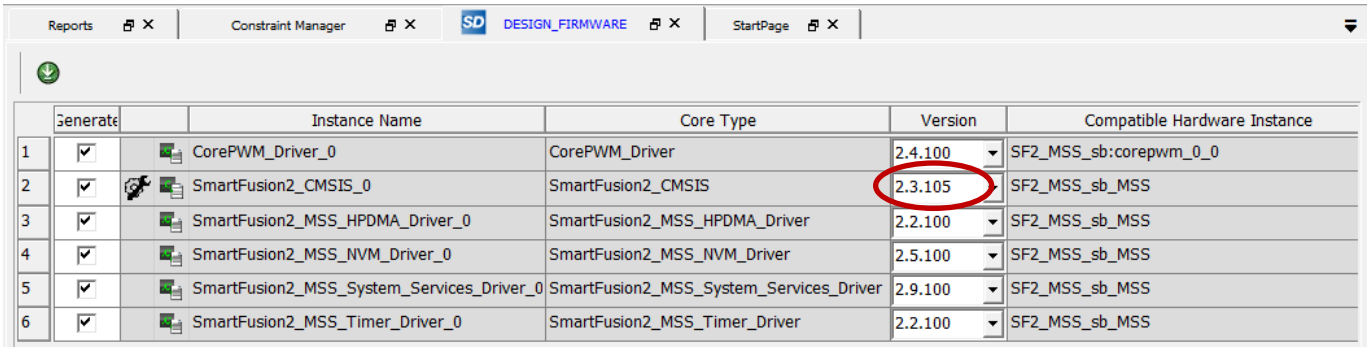
Step 1a: Expand Handoff Design for Firmware Development on the Libero® SoC Design Flow tab. Double-click Configure Firmware Cores to open the DESIGN FIRMWARE tab.



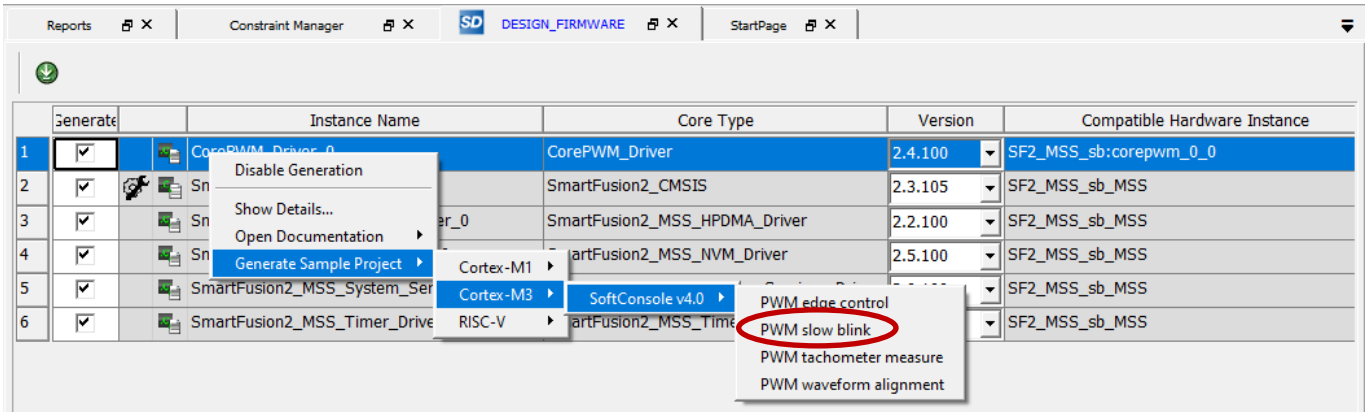
Step 1b: The DESIGN FIRMWARE tab will open. Confirm that none of the drivers appear in italics. If a driver appears in italics, select the core name, then right-click and select **Download All Cores**.



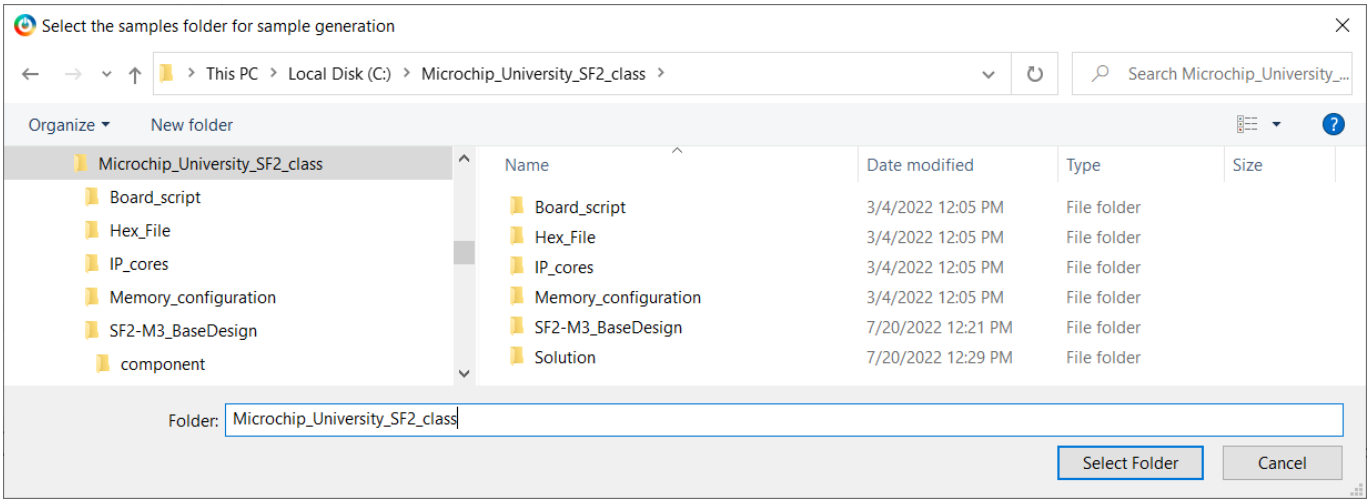
Step 1c: Use the pull-down menu in the SmartFusion2_CMSIS_0 row to select version 2.3.105 (or later) if it is not selected.



Step 1d: Create the CorePWM sample project by selecting CorePWM_Driver_0 on the DESIGN FIRMWARE tab, then right clicking and selecting **Generate Sample Project > Cortex-M3 > SoftConsole v4.0 > PWM slow blink**.

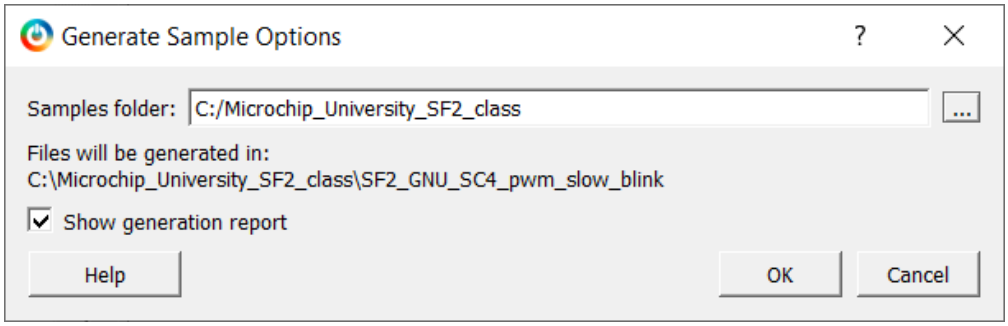


Step 1e: Navigate to the \Microchip_University_SF2_class folder in the Select the samples folder for sample generation dialog box and click **Select Folder**.

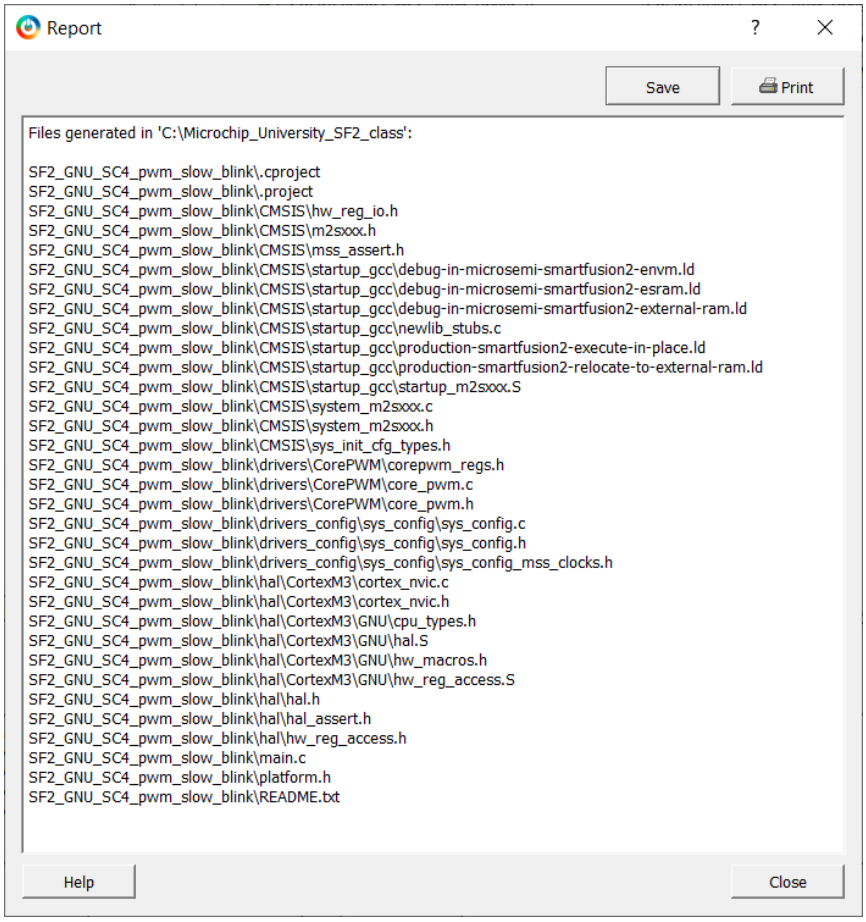


Confirm the following settings in the Generate Sample Options dialog box then click **OK**:

- Samples folder: /Microchip_University_SF2_class
- Show generation report: checked



The Report dialog box will list all the files generated and the location.

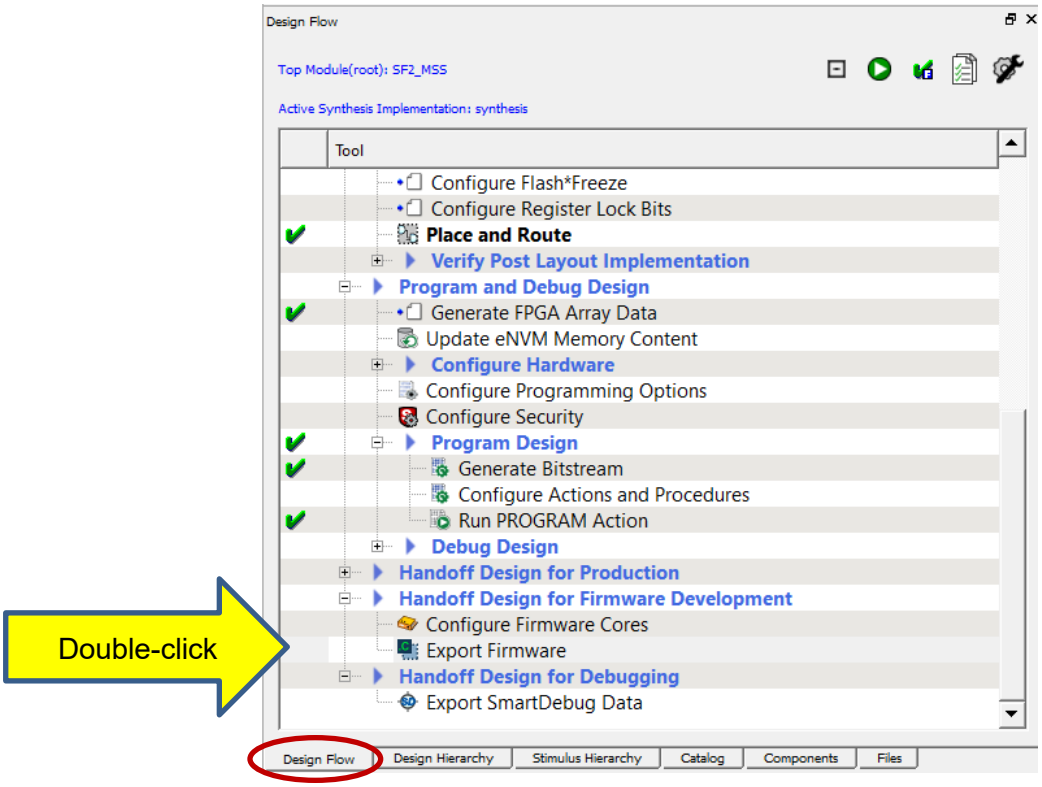


Step 1f: Click **Close** to close the Report window.

STEP 2: Exporting firmware configuration files and firmware drivers

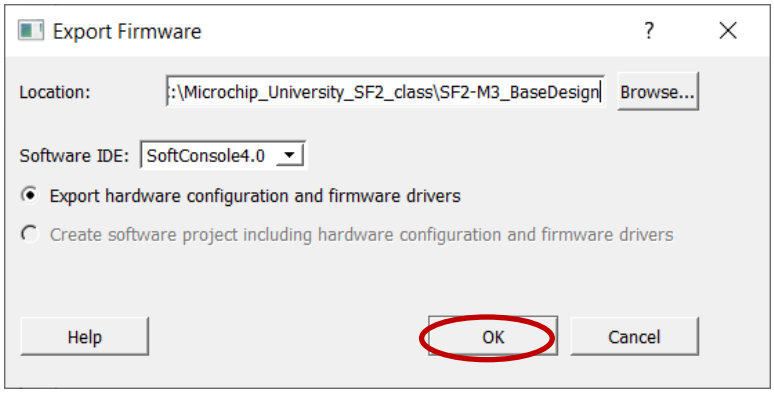
The firmware used in a SoftConsole project must match the target hardware configuration. The sample project created in the previous step contains generic firmware files that may not match the SmartFusion® 2 configuration. Libero® SoC generates specific firmware files that are required to ensure that the SoftConsole project matches the SmartFusion 2 configuration. In this step, you will export firmware configuration files that match the SmartFusion 2 configuration.

Step 2a: Double-click Export Firmware under Handoff Design for Firmware Development on the Libero SoC Design Flow tab to create the firmware drivers for the design.

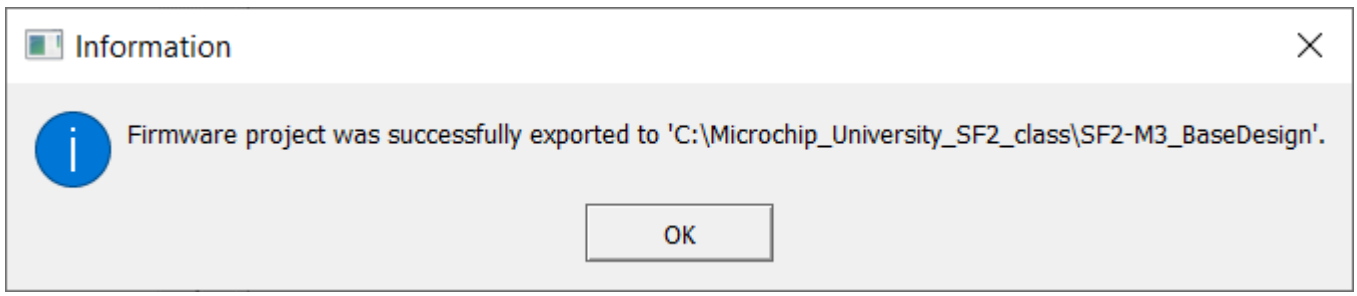


Step 2b: Confirm the following settings in the Generate Sample Options dialog box then click **OK**:

- Location: .\Microchip_University_SF2_class\SF2-M3_BaseDesign
- Software IDE: Select SoftConsole4.0 from the pull-down menu
- Export hardware configuration: Checked

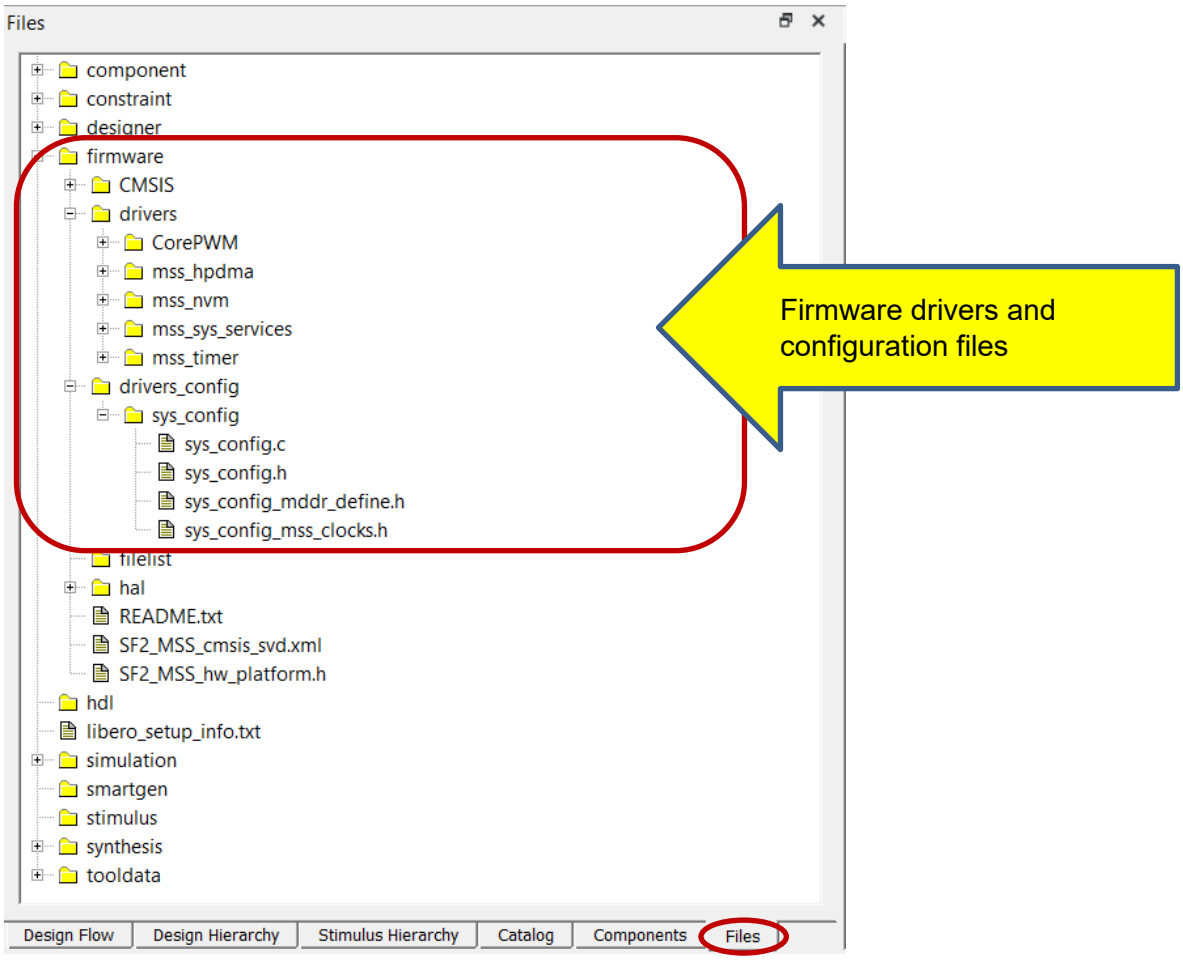


Step 2c: Click **OK** in the dialog box that indicates the location of the firmware cores.

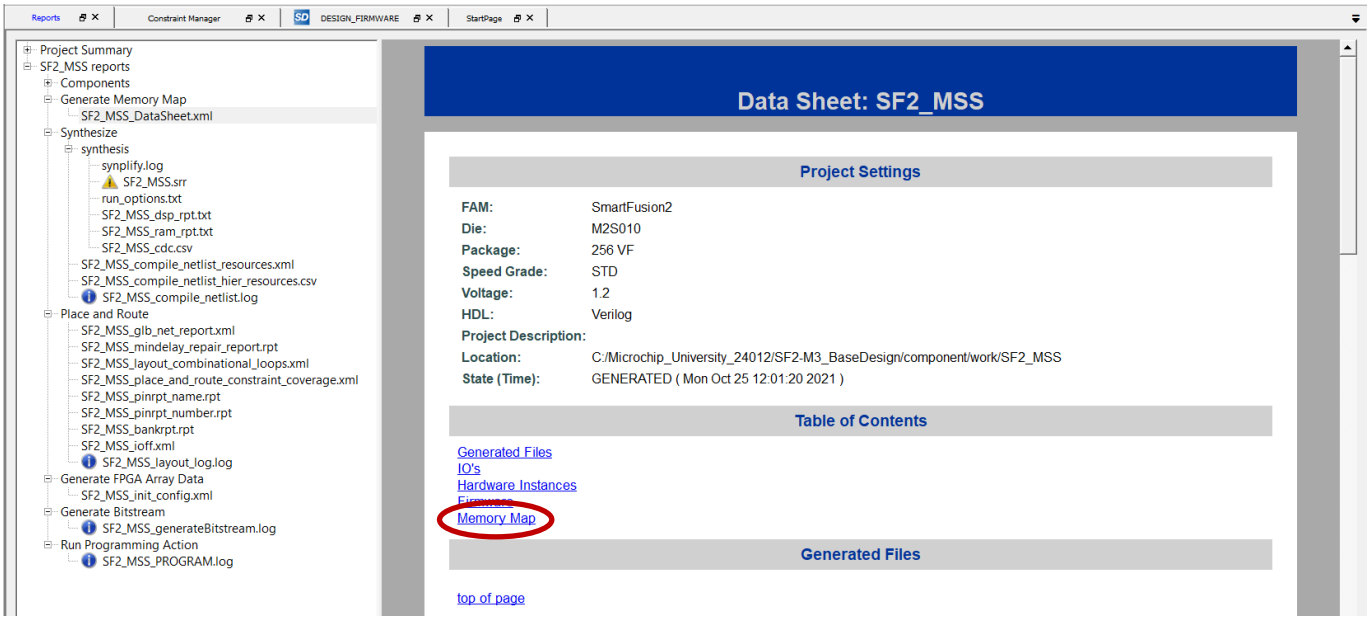


The firmware drivers will be visible on the Libero® SoC Files tab. If the projects are not visible,

select **View > Refresh Design Hierarchy** from the Libero® SoC menu.



The Libero SoC Reports tab will display the Datasheet for the design. The datasheet includes the memory map for the design.



Step 2d: Minimize Libero SoC.

Note: a zip file (SF2-M3_BaseDesign_Lab3.zip) containing the complete project for Lab 2 and Lab 3 is provided in the \Microchip_University_SF2_class\Solution folder.

Results:

After completing this lab, you will see the firmware sample project, the firmware drivers and configuration files on the Libero® SoC Files tab.

Summary:

This lab demonstrated how to generate the firmware drivers for the SmartFusion® 2 MSS peripherals and fabric peripherals that were enabled with System Builder, and how to generate a sample project that can be imported into the software development tool. Additional sample projects can be exported from the Libero SoC DESIGN_FIRMWARE tab to allow experimenting with additional SmartFusion 2 peripherals. The sample projects can also be used as a reference for creating a project from scratch.

Lab 4 – Debugging with SoftConsole

Purpose:

To learn how to import the firmware project into SoftConsole and debug applications running on the Hello FPGA kit.

Overview:

In this lab you will launch SoftConsole, import the CorePWM sample firmware project and run the application from the SmartFusion® 2 Embedded SRAM (eSRAM), the Embedded Non-Volatile Memory (eNVM) and the external DDR3 memory. After building the project you will launch the SoftConsole debugger to run the application.

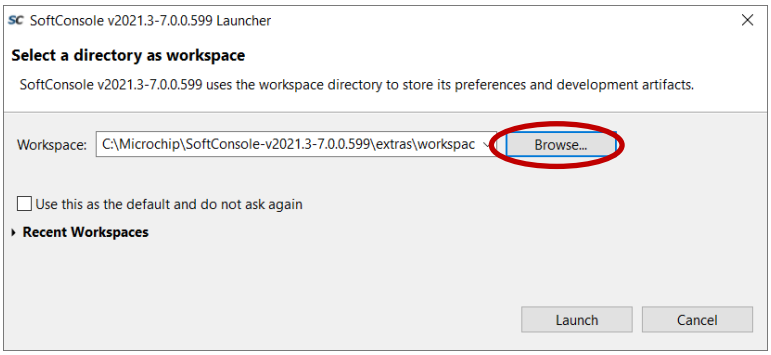
When you complete the lab, you will see the LEDs blinking on the Hello FPGA kit. You can modify the code to change the LED behavior.

Procedure:

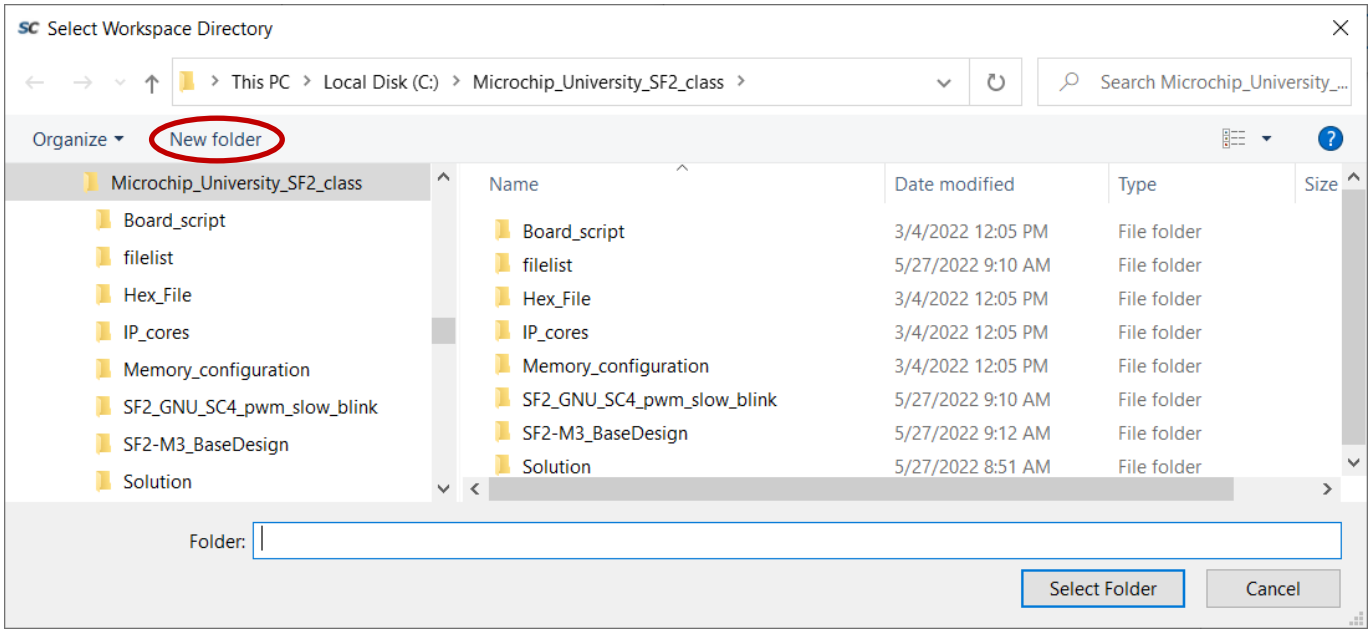
STEP 1: Launching SoftConsole

Step 1a: Open SoftConsole by clicking the shortcut icon on the desktop.

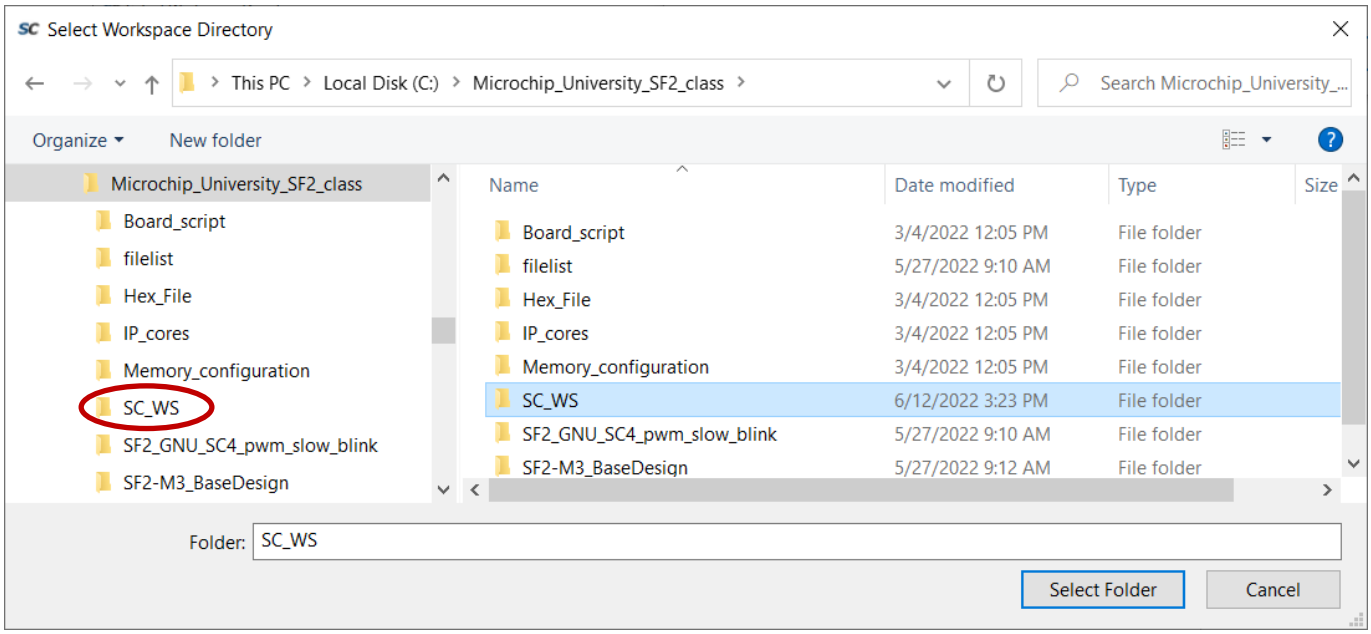
The SoftConsole Workspace Launcher may open.



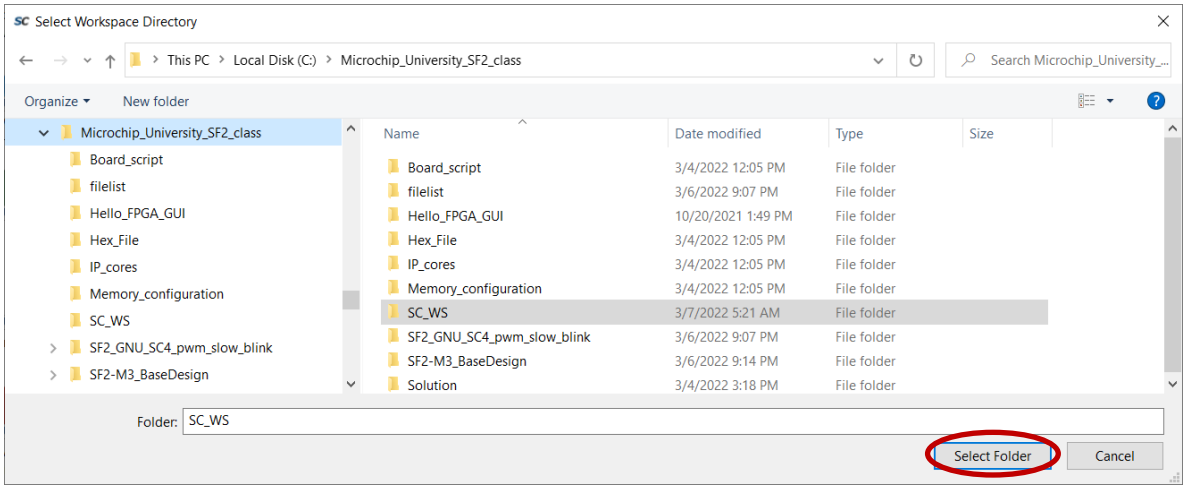
Step 1b: Click the Browse button and navigate to the \Microchip_University_SF2_class folder. Click **New Folder**.



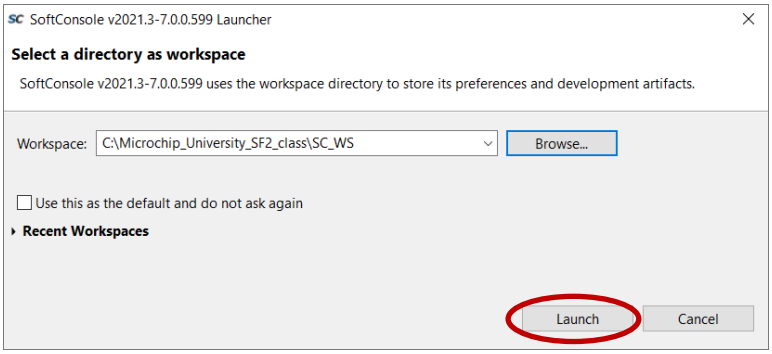
Step 1c: Create a folder named SC_WS.



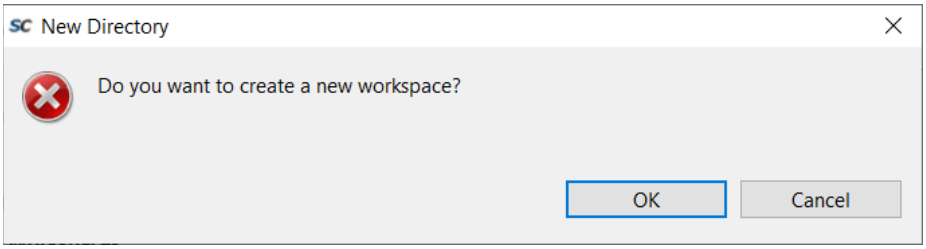
Step 1d: Select the SC_WS folder then click **Select Folder**.



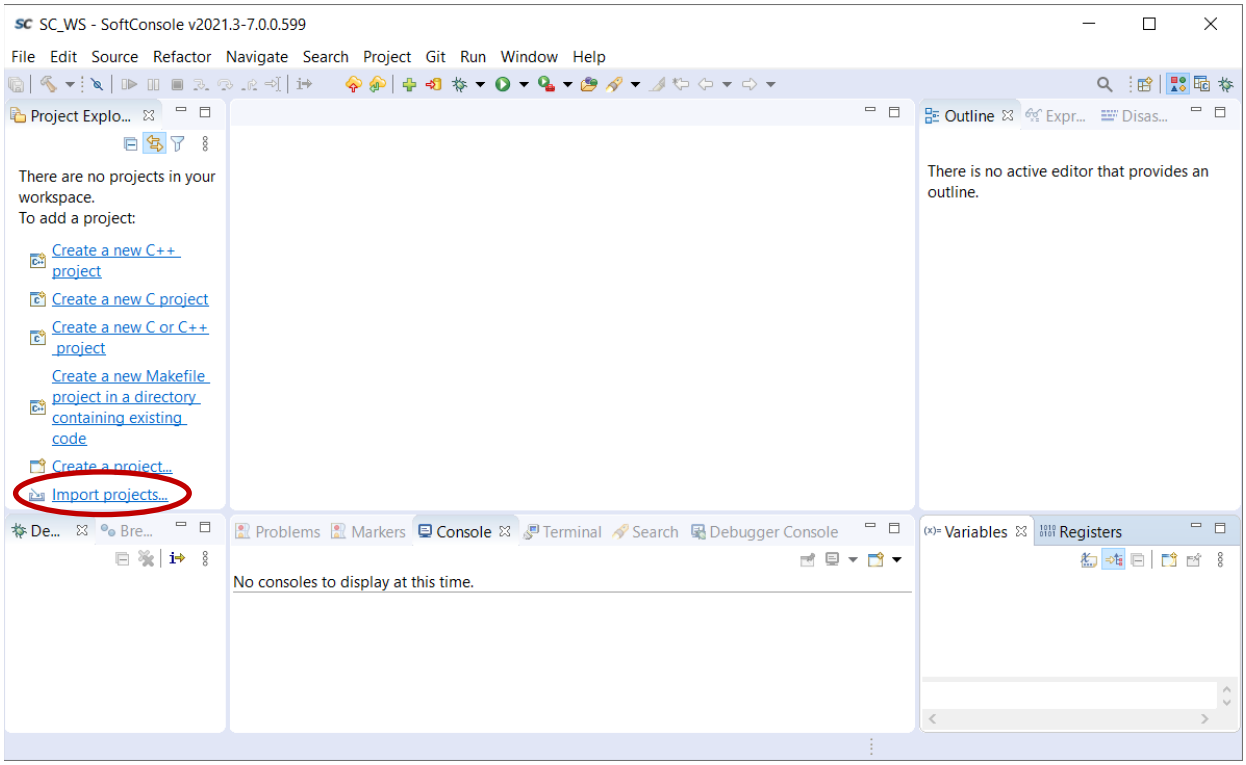
Step 1e: Click the Launch button in the SoftConsole Launcher.



Click **OK** when prompted about creating a new workspace.



The SoftConsole GUI will open.

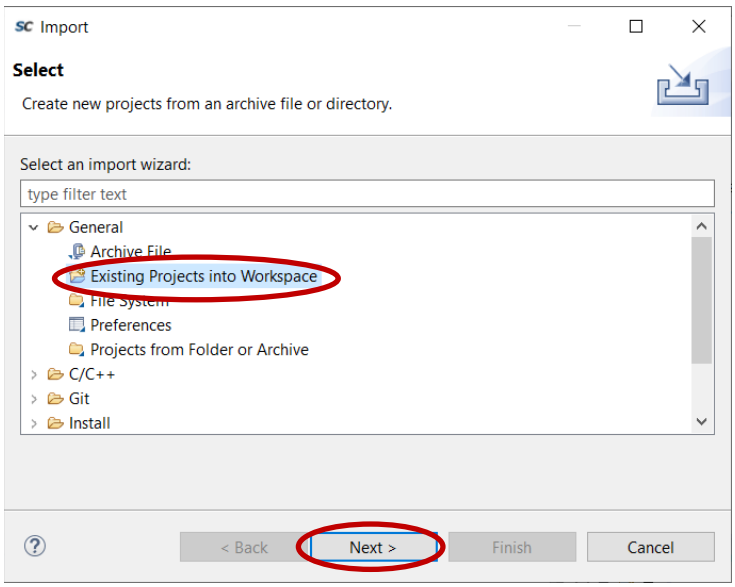


Note: If the workspace launcher did not open, select **File > Switch Workspace > Other** and navigate to the workspace location shown on the previous page.

STEP 2: Importing the CorePWM Slow Blink firmware project

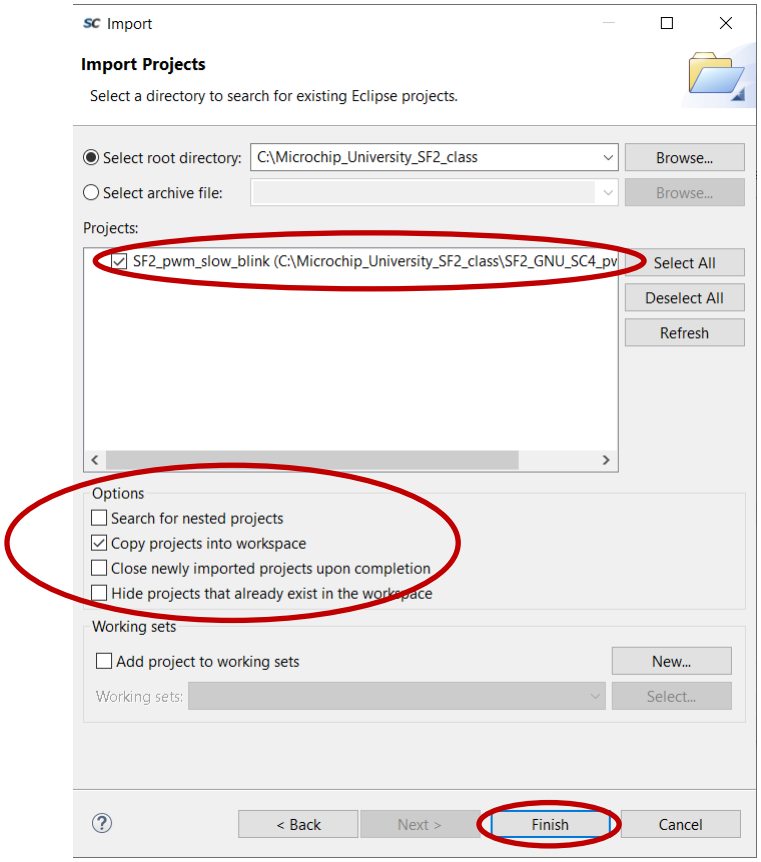
The next step is to import the PWM sample firmware project that was exported from the Libero® SoC project.

Step 2a: Click **Import projects** in the SoftConsole Project Explorer. The Import dialog box will open. Expand General and select **Existing Projects into Workspace** then click **Next**.

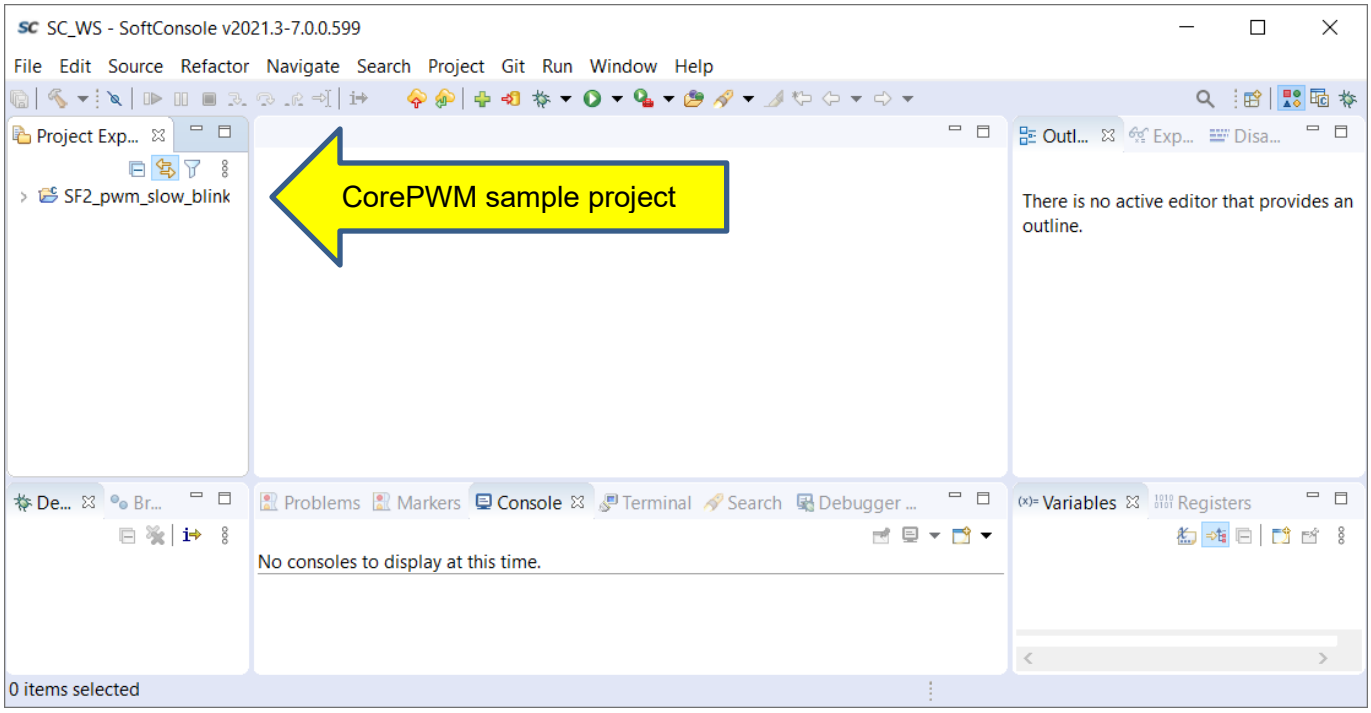


Step 2b: Enter the following in the Import Projects dialog box then click **Finish**:

- Select root directory: Browse to .\Microchip_University_SF2_class then click **Select Folder** in the Select Folder dialog box.
- Projects: SF2_pwm_slow_blink
- Options:
  - Search for nested projects: un-checked
  - Copy projects into workspace: checked
  - Close newly imported projects upon completion: un-checked
  - Hide projects that already exist in the workspace: un-checked



The SoftConsole project will be visible in the SoftConsole Project Explorer.

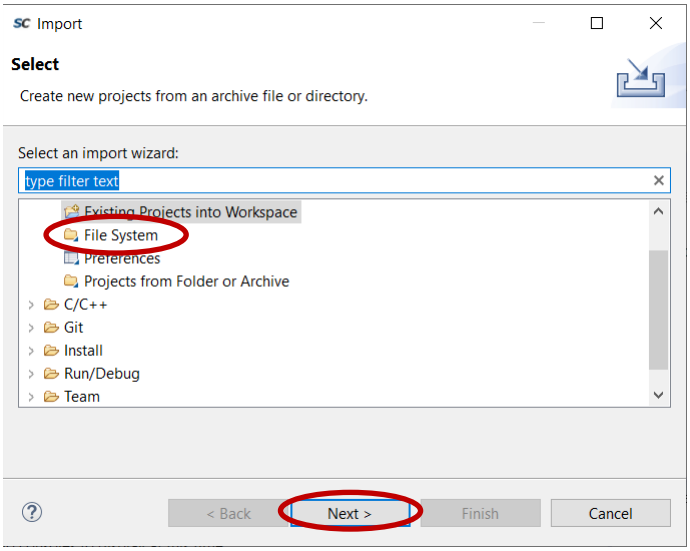


STEP 3: Importing the firmware configuration files

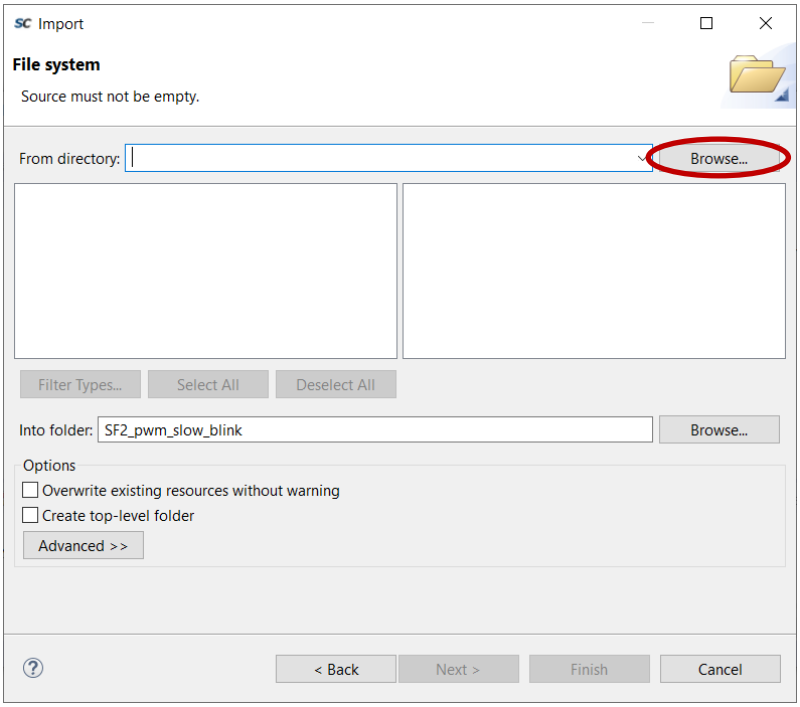
The firmware used in a SoftConsole project must match the SmartFusion® 2 configuration. The sample project created earlier contains generic firmware files that may not match the configuration in the design. Libero® SoC generates specific firmware files that match the SmartFusion 2 configuration. In this step, you will import firmware configuration files that were exported from the Libero SoC project.

Step 3a: Import the firmware configuration files that were exported from Libero SoC into the SF2_pwm_slow_blink project by selecting SF2_pwm_slow_blink in the Project Explorer then selecting **File > Import** from the SoftConsole menu. The Import dialog box will open.

Step 3b: Expand the General category in the Import dialog box and select **File System** and then click **Next**.

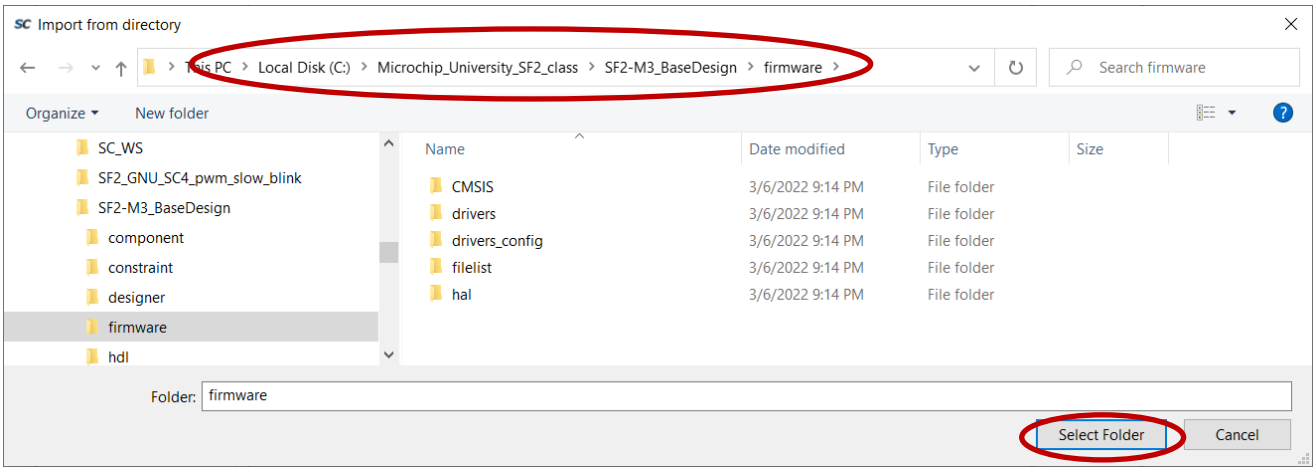


Step 3c: The File system page will open. Click the browse button.



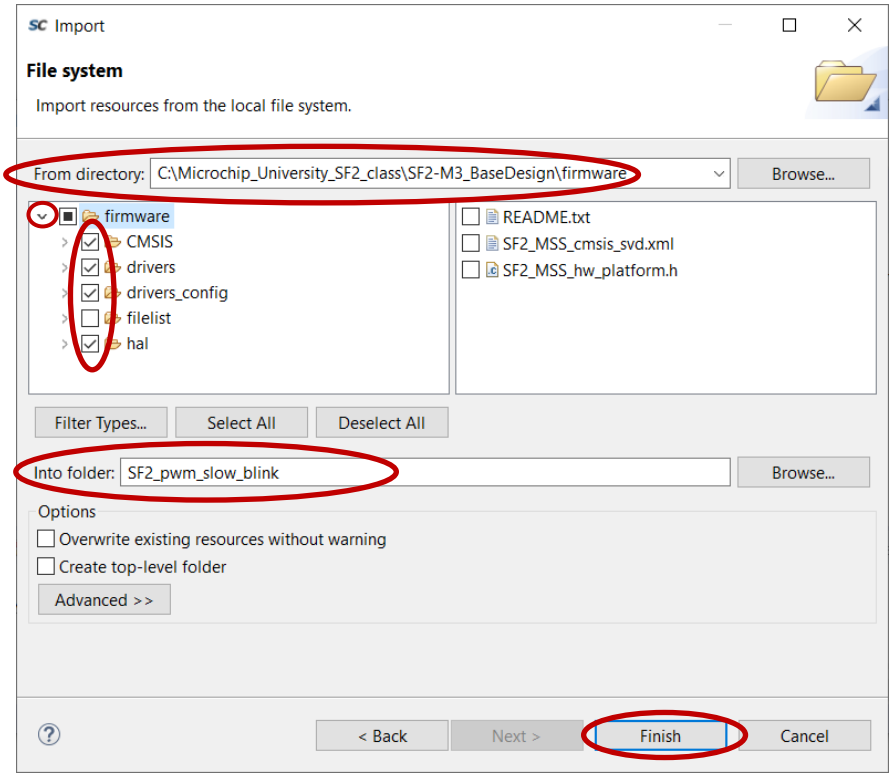
Step 3d: Enter the following in the Import from directory dialog box:

- From directory:
  Browse to .\Microchip_University_SF2_class\SF2-M3_BaseDesign\firmware then click **Select Folder** in the Import from directory dialog box.

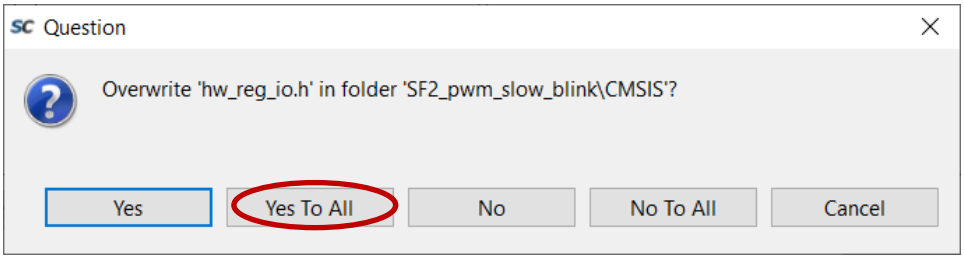


Step 3e: Click next to the firmware folder in the left windowpane to expand the folder (circled in the figure below).
Select the following:

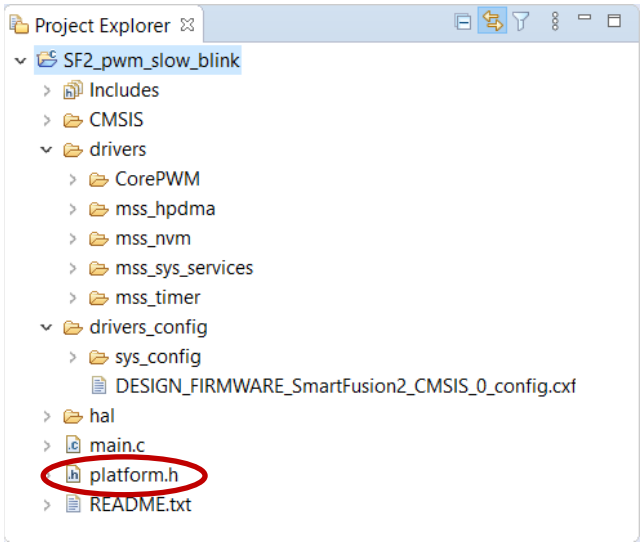
- CMSIS
- drivers
- drivers_config
- hal
- Into Folder: SF2_pwm_slow_blink



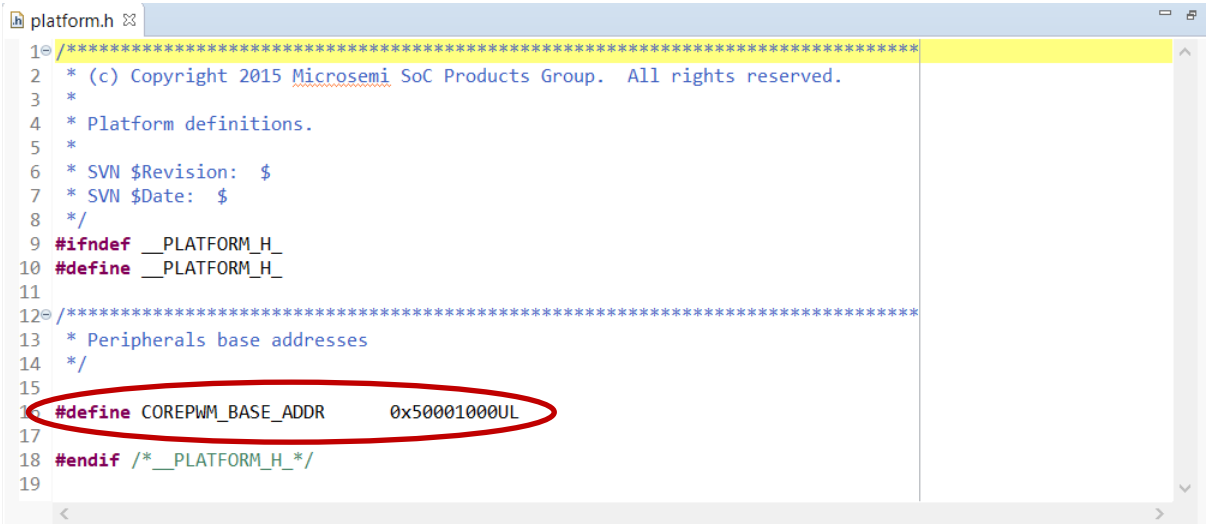
Step 3f: Click **Finish**. Click **Yes To All** in the Question dialog box when prompted about Overwriting files.



The Project Explorer will look like the figure below.



Step 3g: Double click platform.h in the Project Explorer window to open the file in the SoftConsole C/C++ editor.



Step 3h: Change the address of COREPWM_BASE_ADDR on line 16 to 0x50000000. This is the CorePWM address that is shown in the Address map of the design Datasheet in the Libero® SoC project. Save the file (**File > Save**) after making changes.

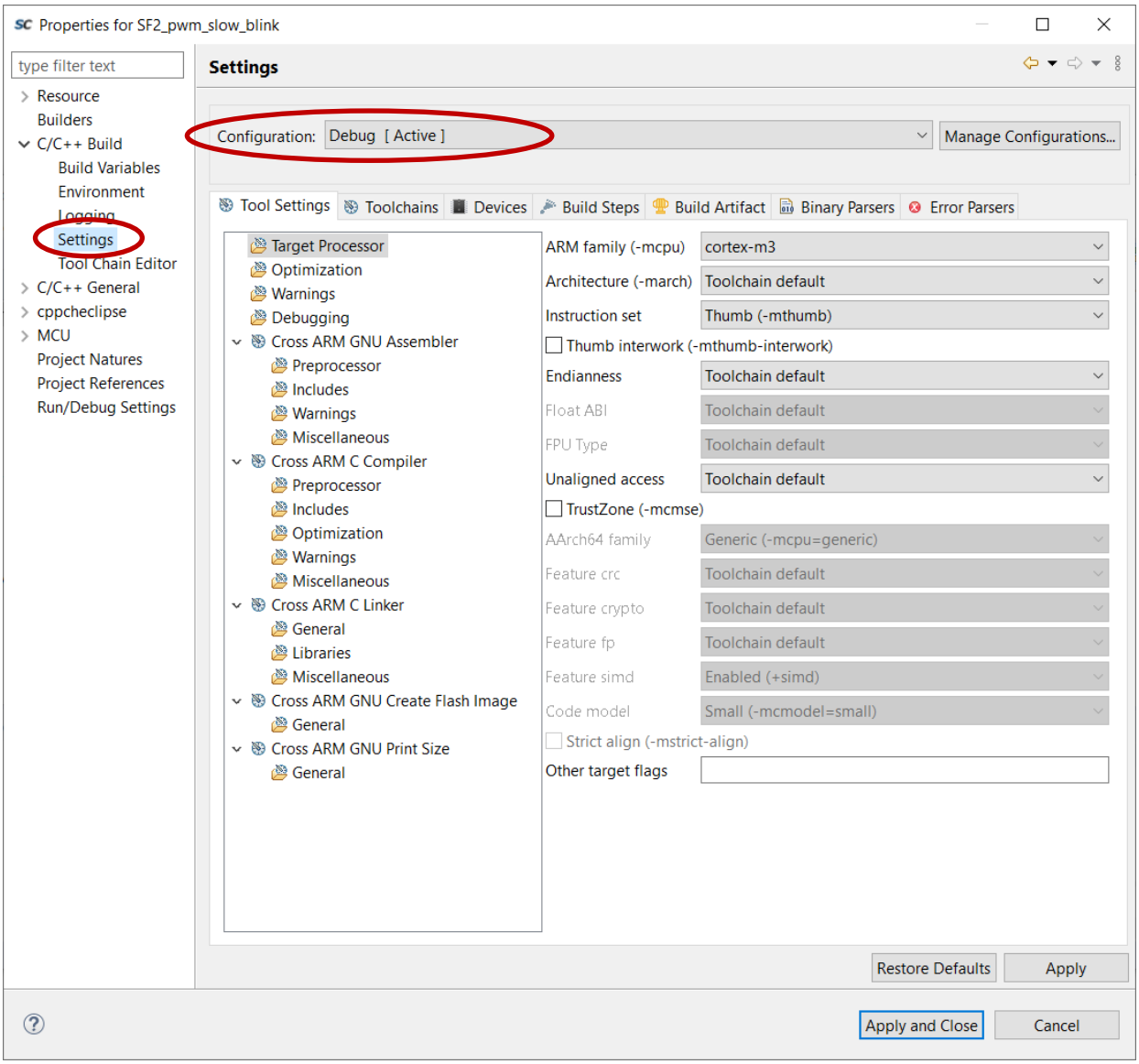
MSSFIC0TOFABRIC_APB_BRIDGE	0x5000_0000, 0x3000_0000	256MB	0x5FFF_FFFF, 0x3FFF_FFFF
SF2_MSS_sb_0/CoreAPB3_0:APB3mmaster			
SF2_MSS_sb_0/corepwm_0_0:APBslave	0x5000_0000, 0x3000_0000	4KB	0x5000_0FFF, 0x3000_0FFF

STEP 4: Confirming the SoftConsole Project Settings

The next step is to confirm the sample project settings prior to performing a build. SoftConsole projects contain two default build configurations named Debug and Release.

Step 4a: Open the Project Properties dialog box by clicking SF2_pwm_slow_blink in the Project Explorer and selecting **Project > Properties** from the SoftConsole menu.

Expand **C/C++ Build** and select **Settings** in the Properties for SF2_pwm_slow_blink dialog box.

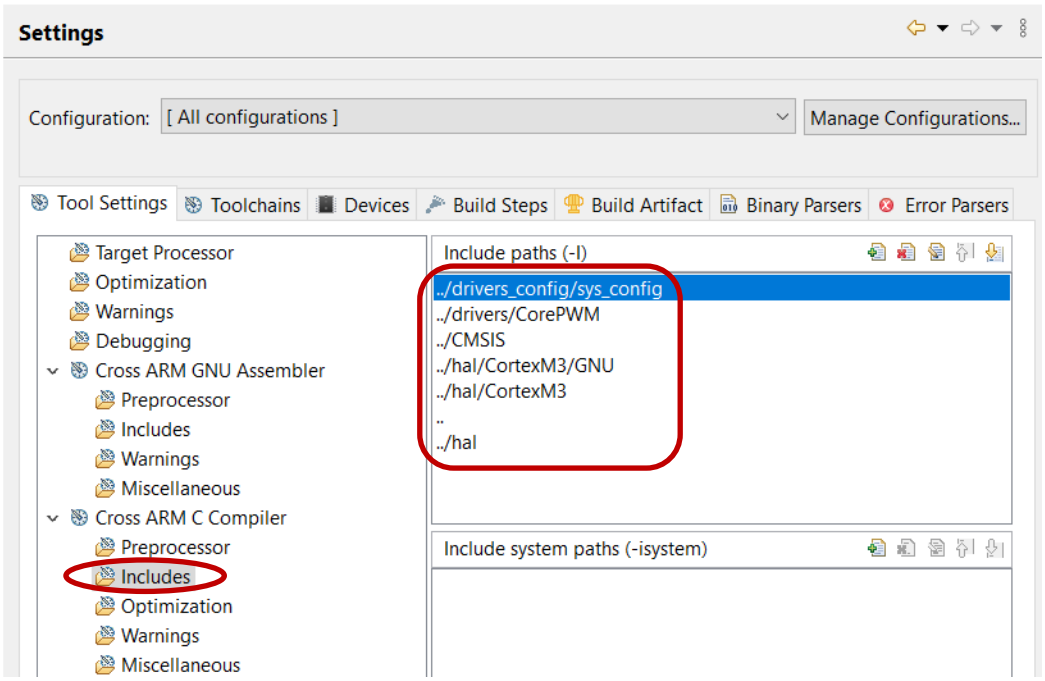


SoftConsole build configurations can be selected by using the pull-down menu in the Configuration field of the Project Properties dialog box (highlighted in the figure above). Some project settings are applicable to all build configurations while others are for a specific build configuration.

Step 4b: Select *Configuration = [All configurations]* to configure settings applicable to both the Debug and Release build targets.

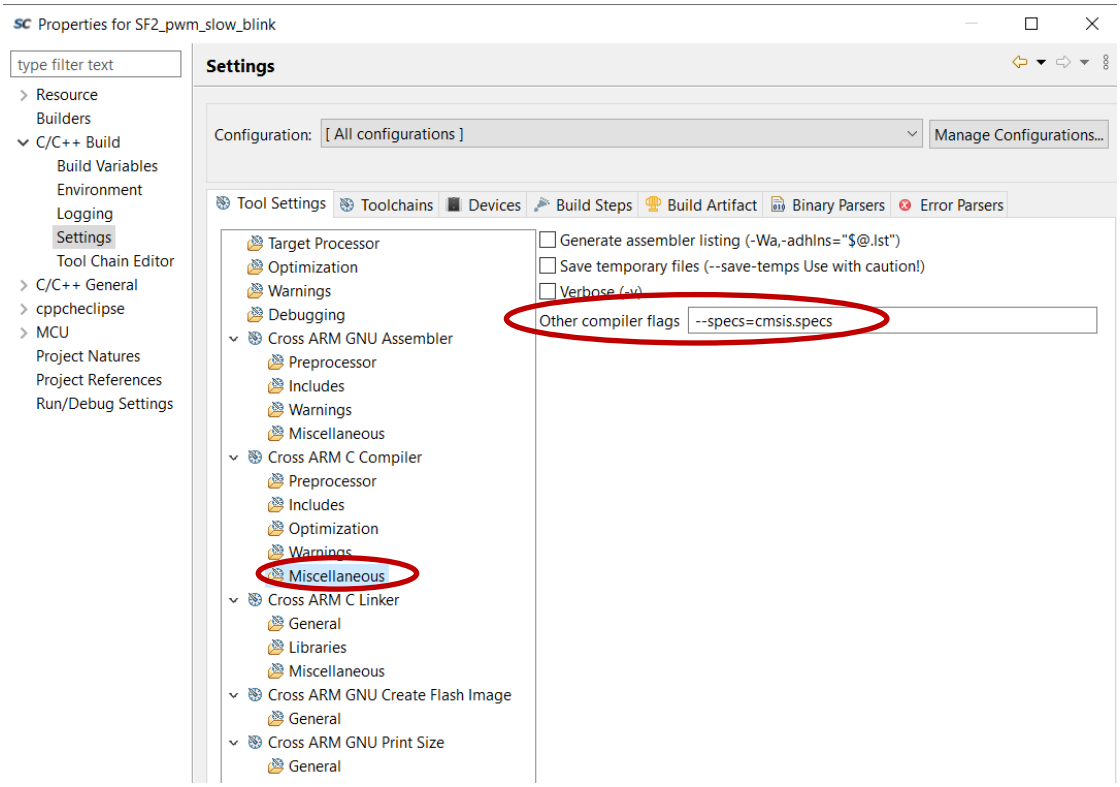
Step 4c: Select *Cross ARM C Compiler* > *Includes* on the Tool Settings tab. Confirm that the Include paths (-I) field contains the following:

- ../drivers_config/sys_config
- ../drivers/CorePWM
- ../CMSIS
- ../hal/CortexM3/GNU
- ../hal/CortexM3
- ..
- ../hal



If necessary, add or modify the Include paths by clicking the Add directory path button ( ) or the Edit directory path button ( ).

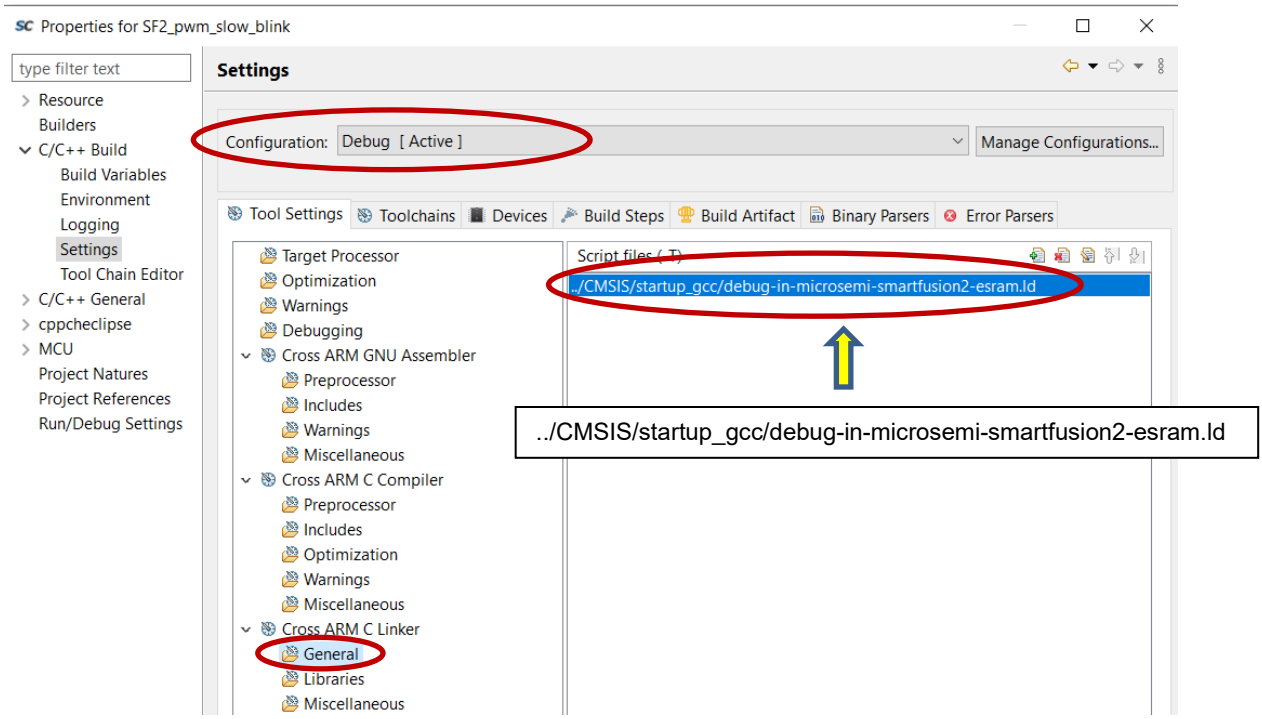
Step 4d: SmartFusion® 2 projects require a setting that allows the preprocessor to find the toolchain CMSIS header files. Select *Cross ARM C Compiler* > *Miscellaneous* on the Tool Settings tab. Confirm that the Other compiler flags field contains `-specs=cmsis.specs`.



The appropriate linker script must be configured for the project build configuration. The sample projects include linker scripts bundled with the CMSIS/HAL firmware core. Follow the steps below to confirm the linker script setting for the Debug and Release build configurations.

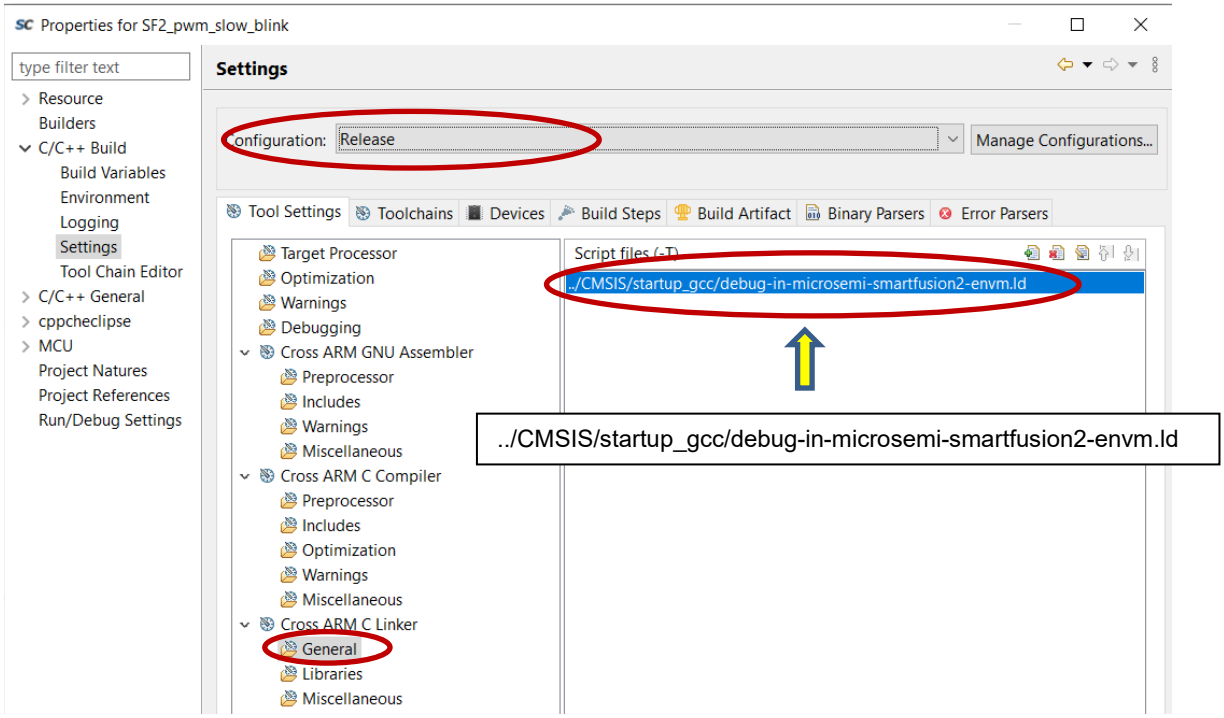
Step 4e: Select *Configuration = Debug [Active]* from the pull-down menu in the Configuration field of the Project Properties dialog box.

Step 4f: Select *Cross ARM C Linker > General* on the Tool Settings tab of the Project Properties dialog box. Confirm that the Script files (-T) contains **../CMSIS/startup_gcc/debug-in-microsemi-smartfusion2-esram.ld**. This linker script builds an application that runs from the SmartFusion® 2 eSRAM.



Step 4g: Select *Configuration = Release* from the pull-down menu in the Configuration field of the Project Properties dialog box.

Step 4h: Select *Cross ARM C Linker > General* on the Tool Settings tab of the Project Properties dialog box and confirm that the Script files (-T) field contains **../CMSIS/startup_gcc/debug-in-microsemi-smartfusion2-envm.ld**. This linker script builds an application that runs from the SmartFusion® 2 eNVM.

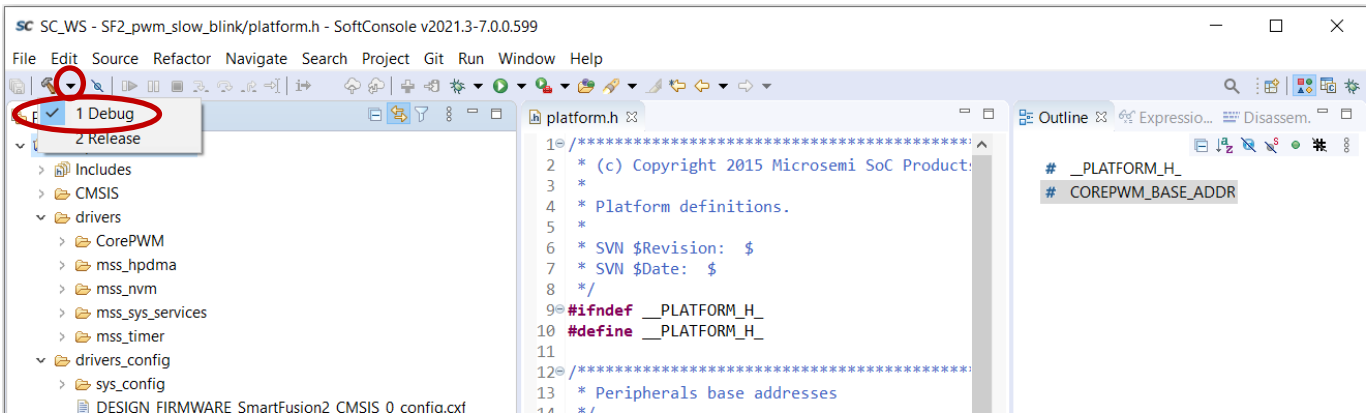


Step 4i: After confirming the settings listed in the previous steps, click **Apply and Close** in the Properties for SF2_pwm_slow_blink dialog box.

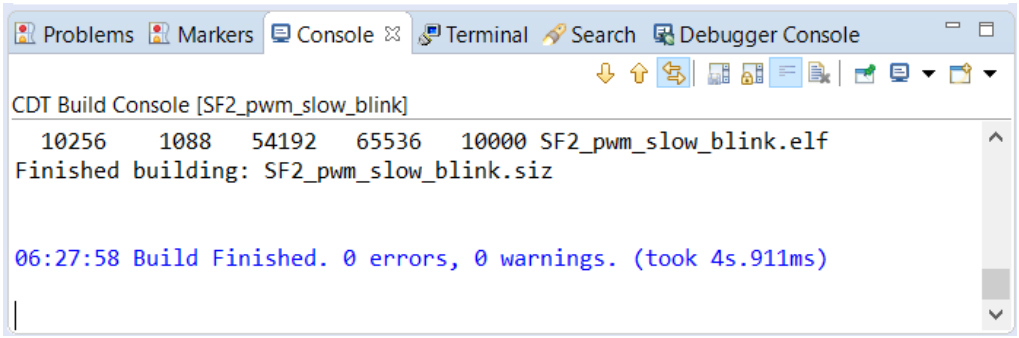
STEP 5: Building the Project

After configuring the project settings, the next step is to build the project using the Debug build configuration.

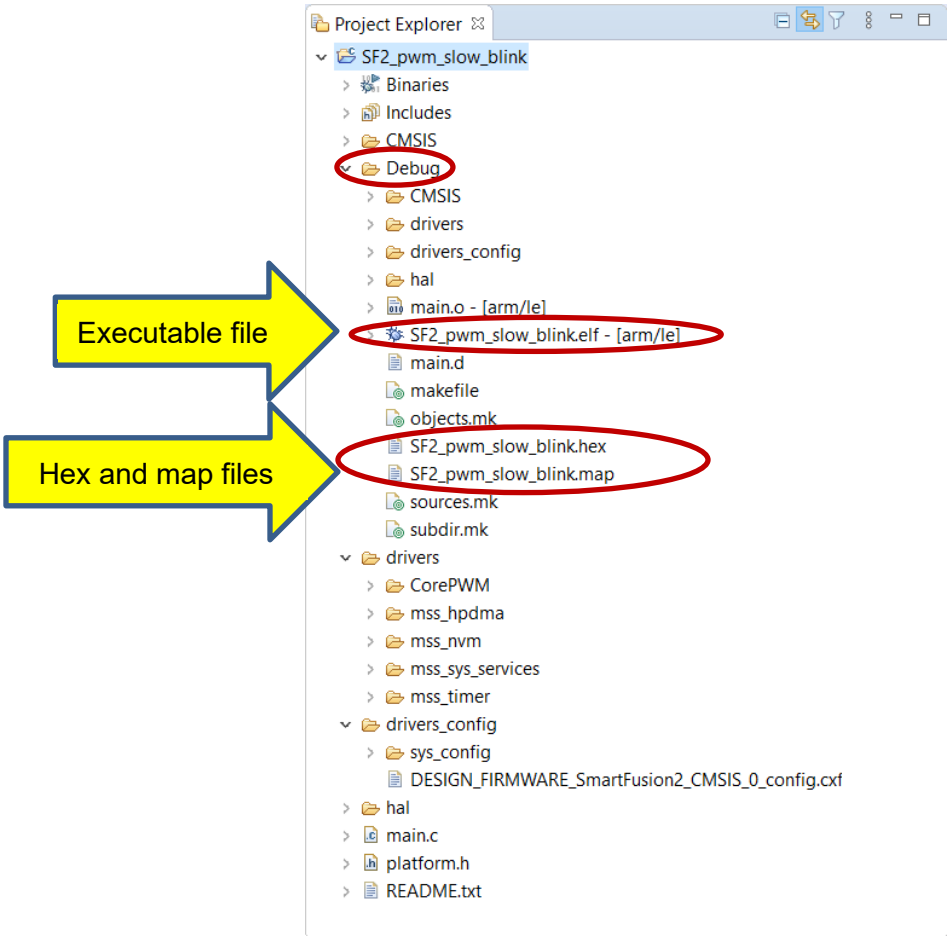
Step 5a: Select *SF2_pwm_slow_blink* in the Project Explorer then click the triangle next to the hammer icon. Click **Debug** to build the project using the Debug build configuration.



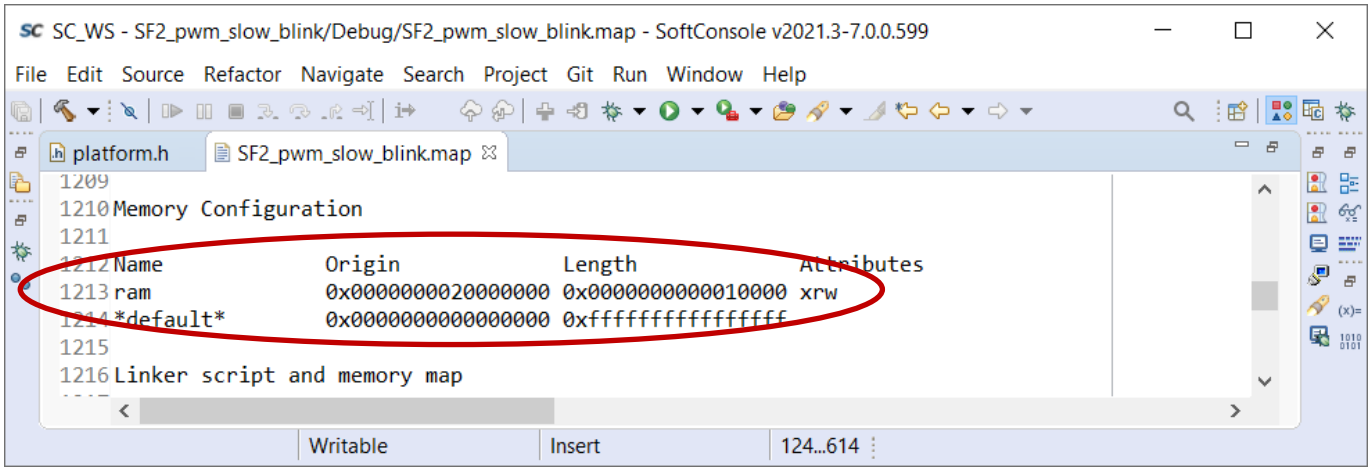
Confirm that there are no errors or warnings in the Console view. If errors occur, select the Problems view to get additional information. Correct the errors then build the project again.



A folder named Debug will be visible in the SoftConsole Project Explorer. This folder contains a file named SF2_pwm_slow_blink.elf which is the executable that will be downloaded using the SoftConsole debugger. The folder also includes a hex file and the map file for the build.



Double-click SF2_pwm_slow_blink.map in the Debug folder (highlighted in the figure above) to open the file in the SoftConsole editor. Scroll down to the Memory Configuration section. The section should appear as shown in the figure below.



The memory configuration for the Debug build includes a ram section at 0x20000000, which is the address of the SmartFusion® 2 eSRAM.

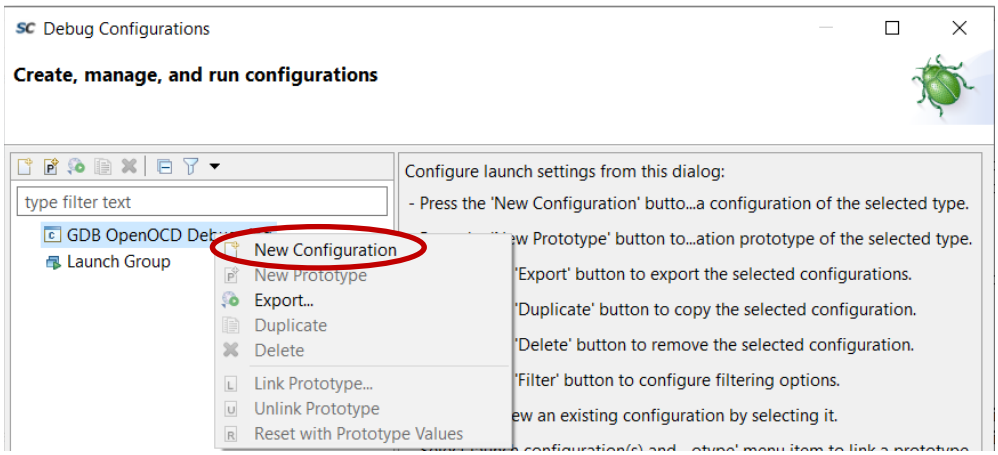
STEP 6: Debugging with SoftConsole

After building the project, the next step is to launch the SoftConsole debugger and download the application to the Hello FPGA board. Before using the On-chip Debugger (OCD), the debug launch configuration must be created.

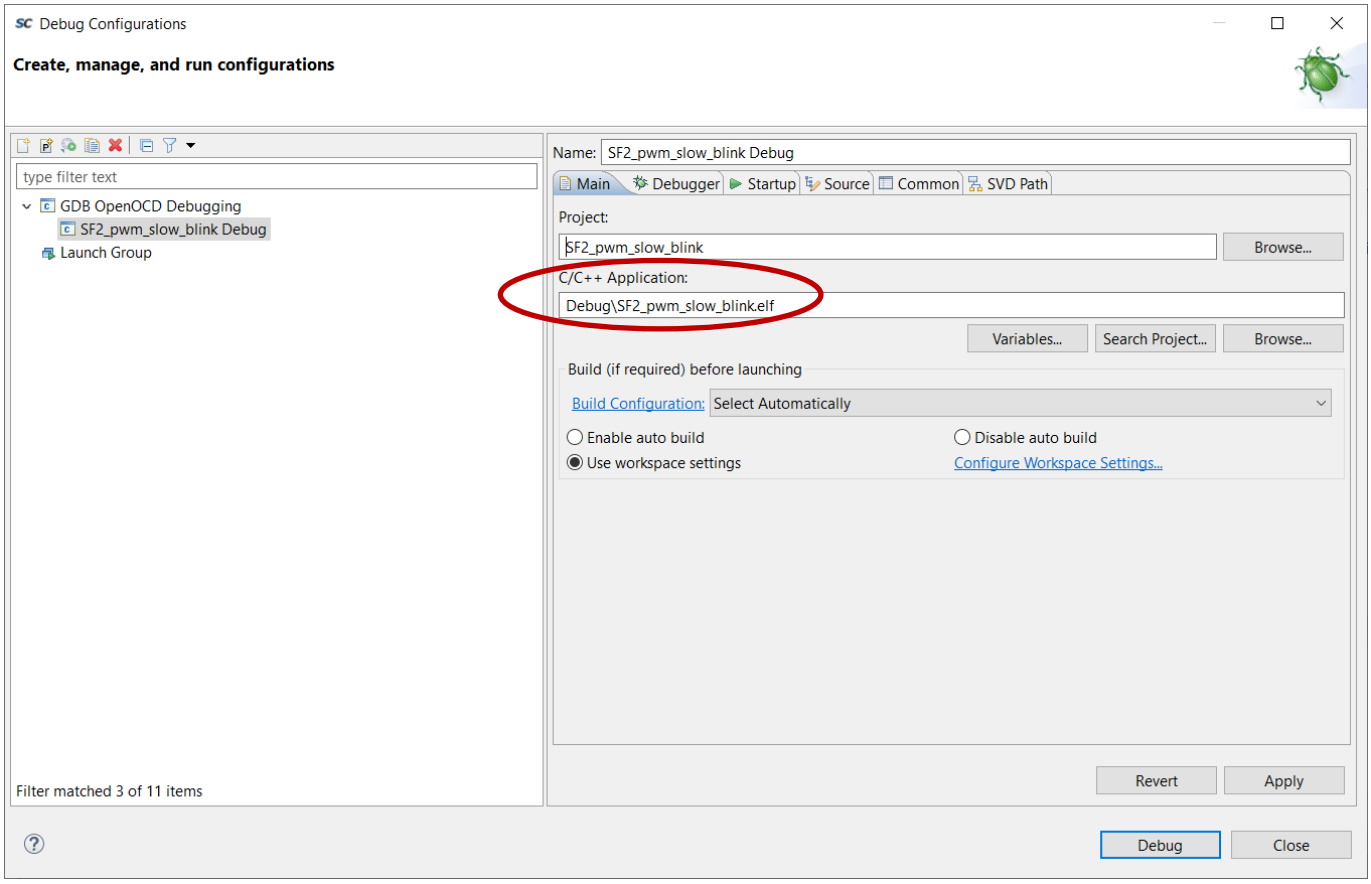
Step 6a: Creating a debug launch configuration

Open the Debug Configurations dialog box by clicking SF2_pwm_slow_blink in the Project Explorer then selecting **Run > Debug Configurations…** from the SoftConsole menu. The Debug Configurations dialog will open.

Step 6b: Select *GDB OpenOCD Debugging* in the Debug Configurations dialog box then right-click and select **New Configuration** to create a new debug launch configuration.



Step 6c: Confirm that the C/C++ Application field on the Main tab of the Debug Configurations dialog box contains Debug\SF2_pwm_slow_blink.elf. Click the Search Project button to search for the file if the box is blank.



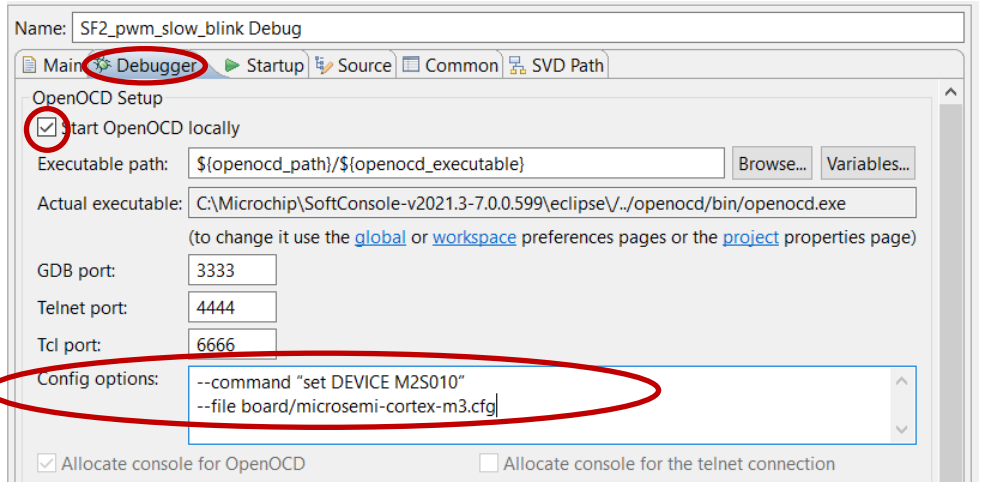
Step 6d: Select the Debugger tab of the Debug Configurations dialog box. Check Start OpenOCD locally.

Step 6e: The Config Options field must contain the correct command line options/script to be passed to OpenOCD. Several board script files are provided under the SoftConsole installation.

For Cortex®-M3 targets, the command line options/script are `--command "set DEVICE M2SXXX"` and `-file board/54icrosemi-cortex-m3.cfg`. `--command "set DEVICE M2SXXX"` specifies the target device. This line needs to be modified based on the target silicon.
`--file board/54icrosemi-cortex-m3.cfg` is a board script that supports SmartFusion® 2 programs that target eSRAM or eNVM. Additional board scripts are provided in the SoftConsole installation folder.

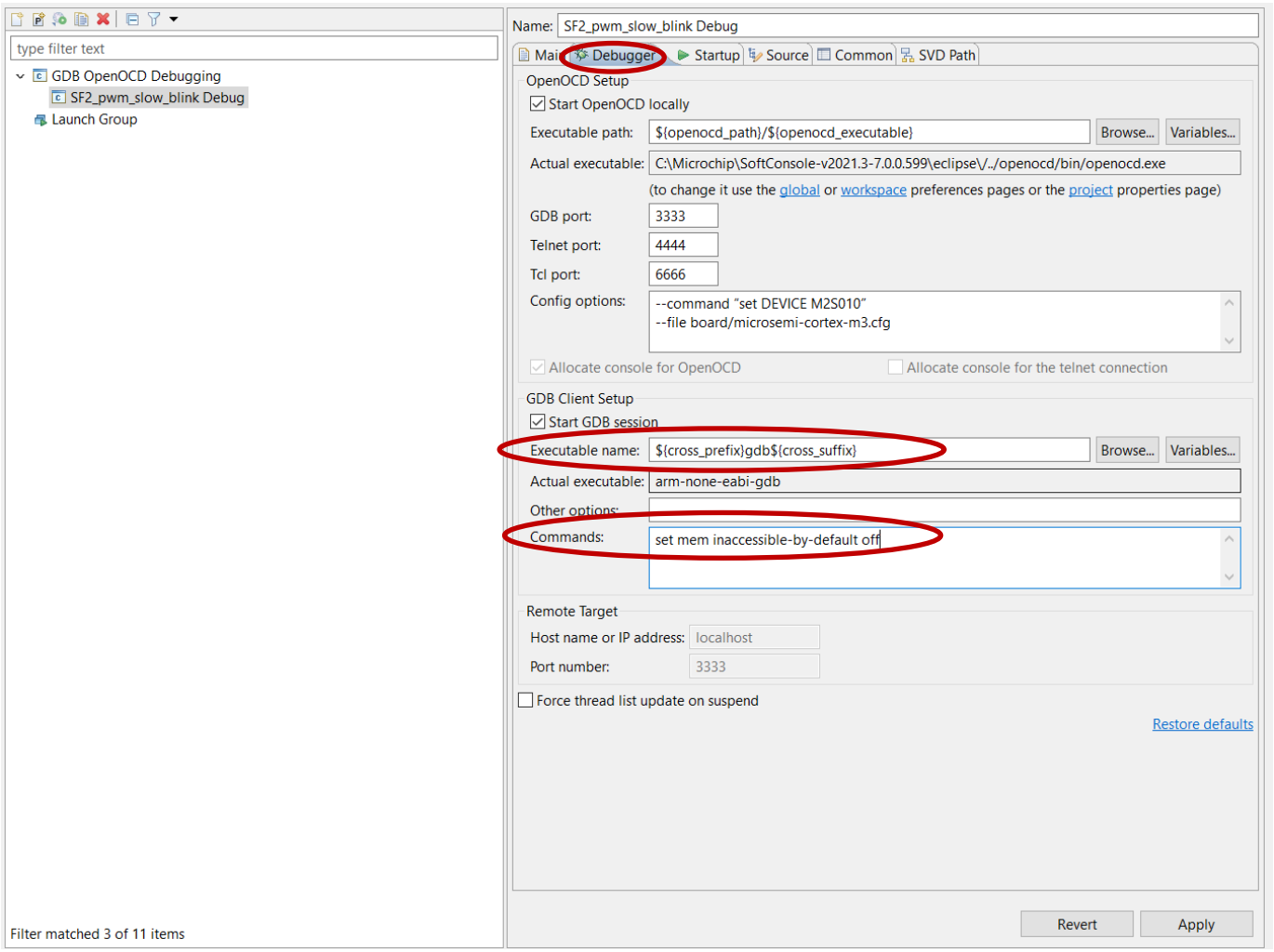
Modify the Config Options field as follows:

- Enter the "set DEVICE" command `--command "set DEVICE M2S010"`
- Enter the board script file: `--file board/54icrosemi-cortex-m3.cfg`

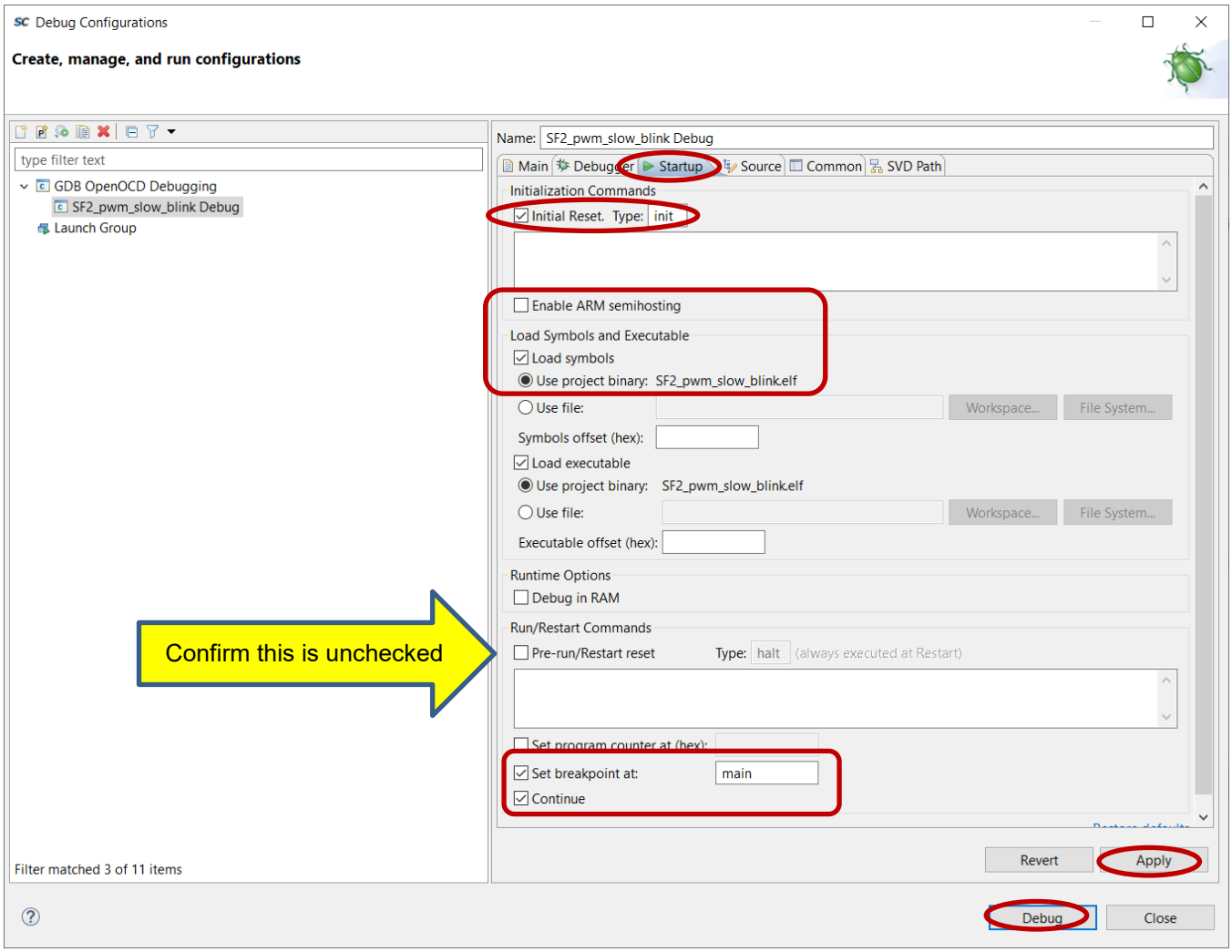


Step 6f: Scroll to the GDB Client Setup field on the Debugger tab. Confirm that the Executable name and Commands fields contain the following:

- Executable: `${cross_prefix}gdb${cross_suffix}`
- Commands: `set mem inaccessible-by-default off`



Step 6g: Select the Startup tab and confirm that the default settings are configured as shown in the figure below.

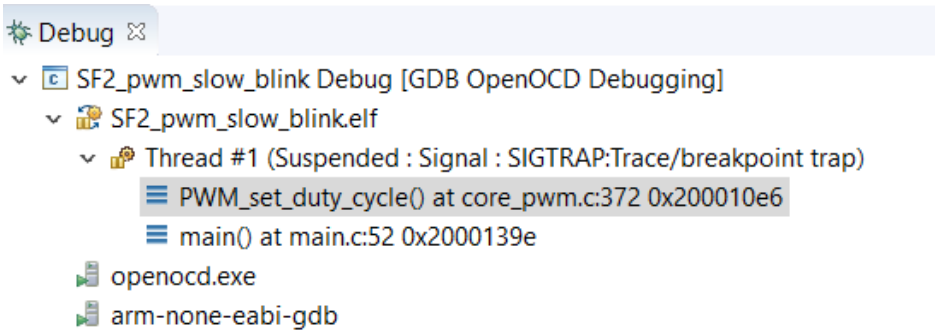


Step 6h: Click **Apply** to save the changes.

Step 6i: Click **Debug** to launch the SoftConsole Debugger.

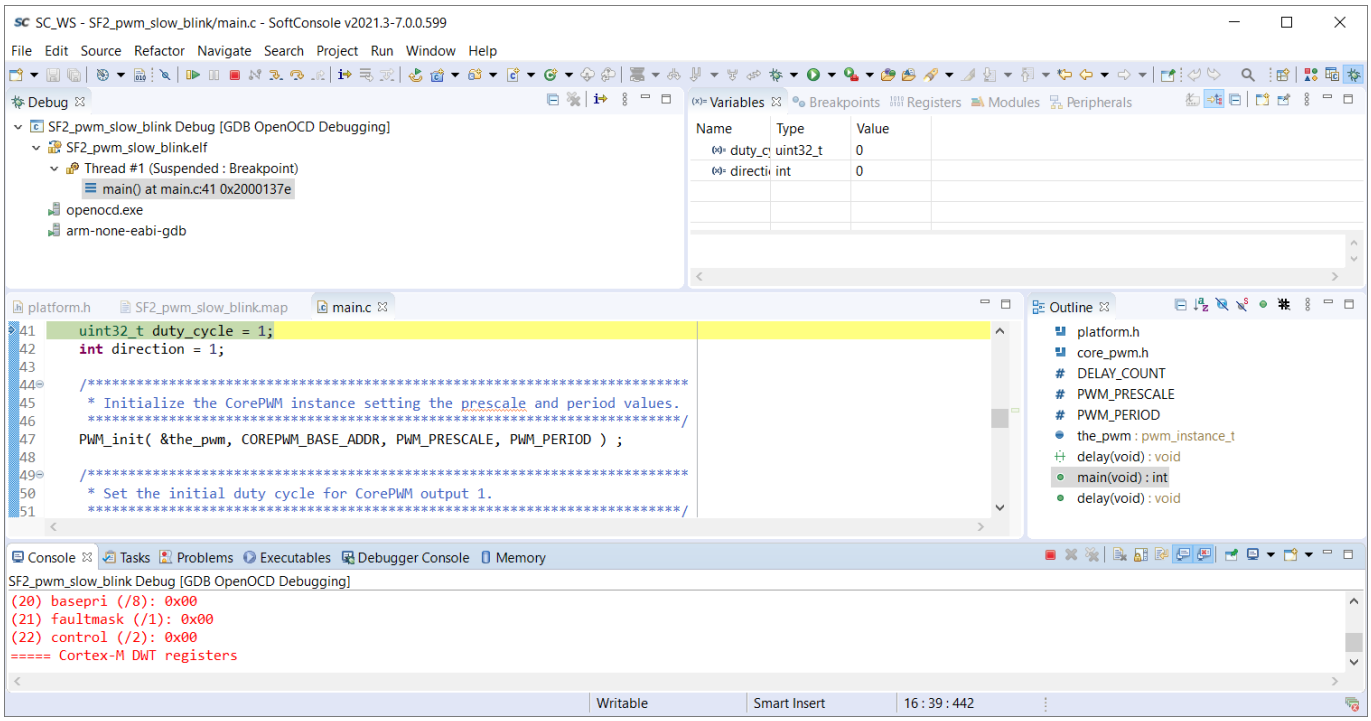
Note1: If an error occurs when launching the Debugger, open the Debug Configurations dialog box (**Run > Debug Configurations**) and confirm that the Config Options field settings match the instructions.

Note2: If the message `Thread #1 (Suspended: Signal: SIGTRAP:Trace/breakpoint trap)` appears in the Debug tab when the debugger launches (as shown below), confirm that the address for CorePWM was updated in the platform.h file as described in step 3h. This error occurs because the processor is attempting to access an unassigned address.



STEP 7: Running the pwm_slow_blink Application

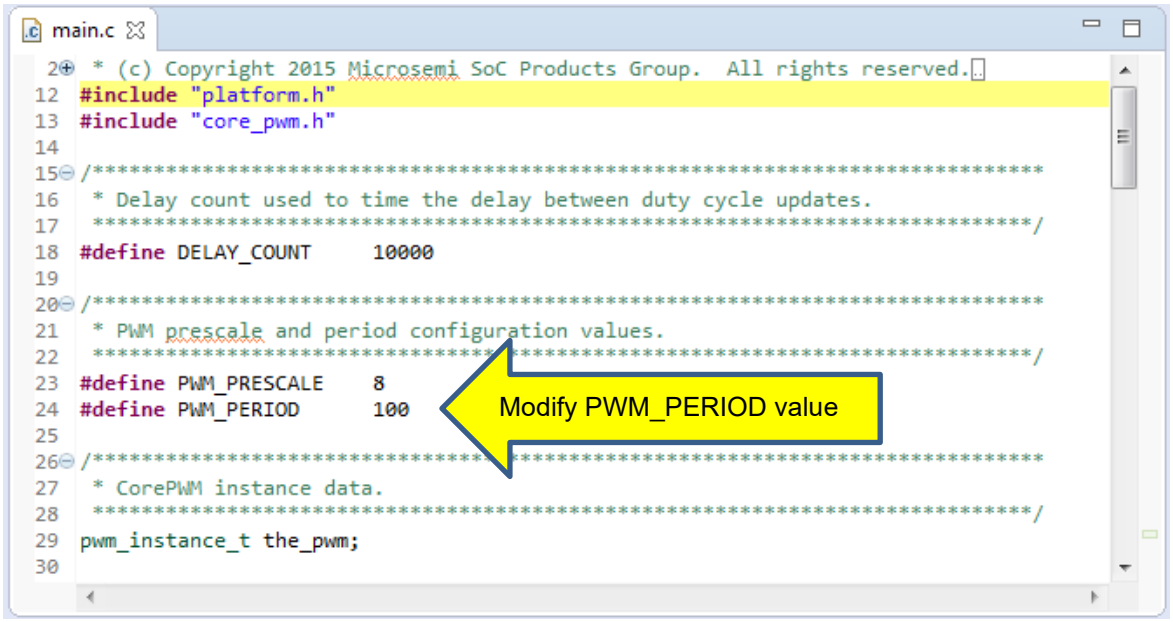
The executable will be automatically downloaded to the Hello FPGA board. Open the SoftConsole Debug Perspective by selecting **Window > Perspective > Open Perspective > Debug**. The program will be suspended at the first executable line of main() as shown in the figure below.



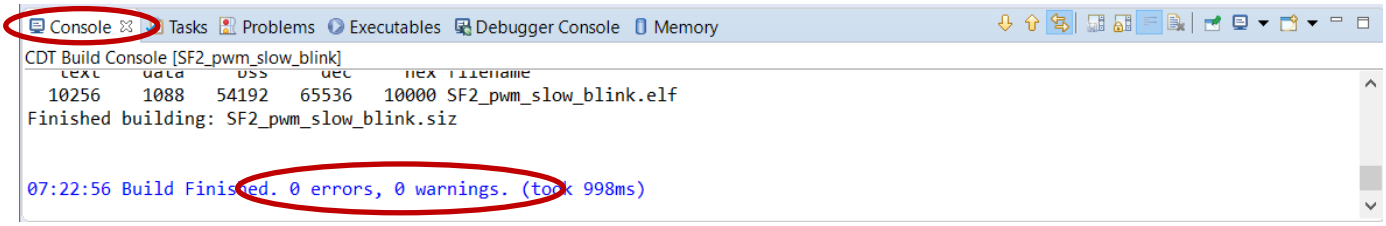
Step 7a: Run the Cortex®-M3 software application by clicking **Run > Resume** from the SoftConsole menu. LED1 will turn on and off. LED2 will be on.

Step 7b: Suspend the software application by clicking **Run > Suspend** from the SoftConsole menu or by clicking the Suspend icon ( ) on the SoftConsole Toolbar.

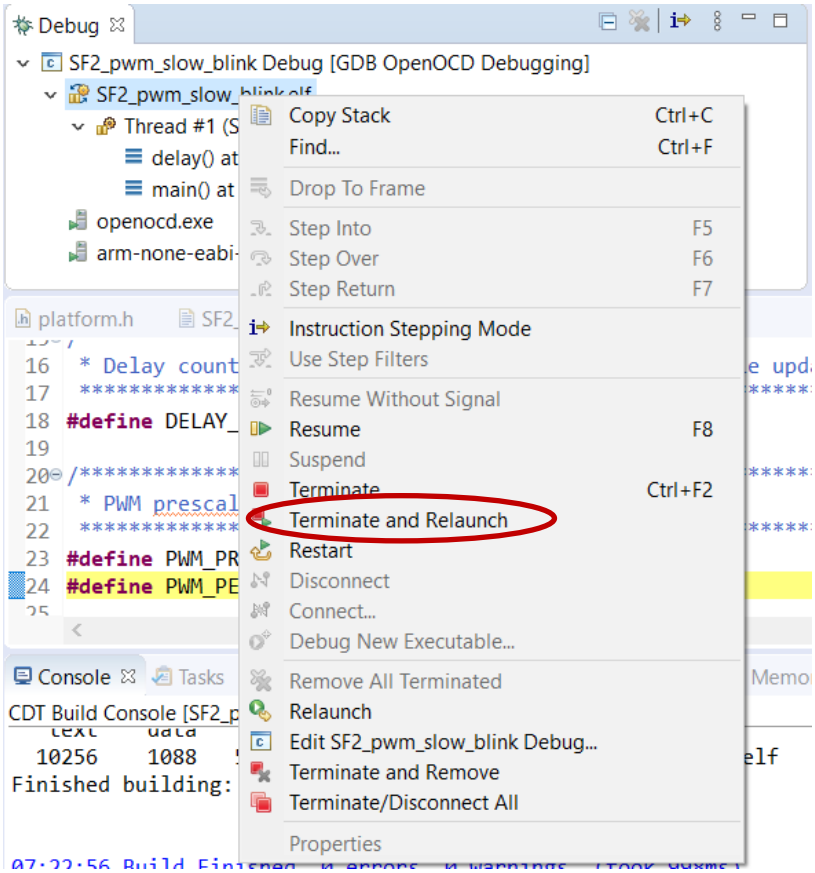
Step 7c: Scroll in **main.c** and locate PWM_PERIOD on line 24. Changing the value of PWM_PERIOD will change the blink rate of the LED. A larger value for the period will make the LED toggle more slowly. Change the value then save the file (**File > Save**).



Step 7d: Build the project (**Project > Build All**). Confirm that there are no errors reported in the Console view. If errors are reported, select the Problems view for more information. Correct any errors and rebuild the project.



Step 7e: Select SF2_pwm_slow_blink.elf under the Debug tab in the upper left corner of the SoftConsole Debug Perspective. Right-click and choose **Terminate and Relaunch** to stop the debugger and download the new executable.



Step 7f: Run the Cortex®-M3 software application by clicking **Run > Resume** from the SoftConsole menu. LED1 will blink at a different rate.

STEP 8: Driving two LEDs

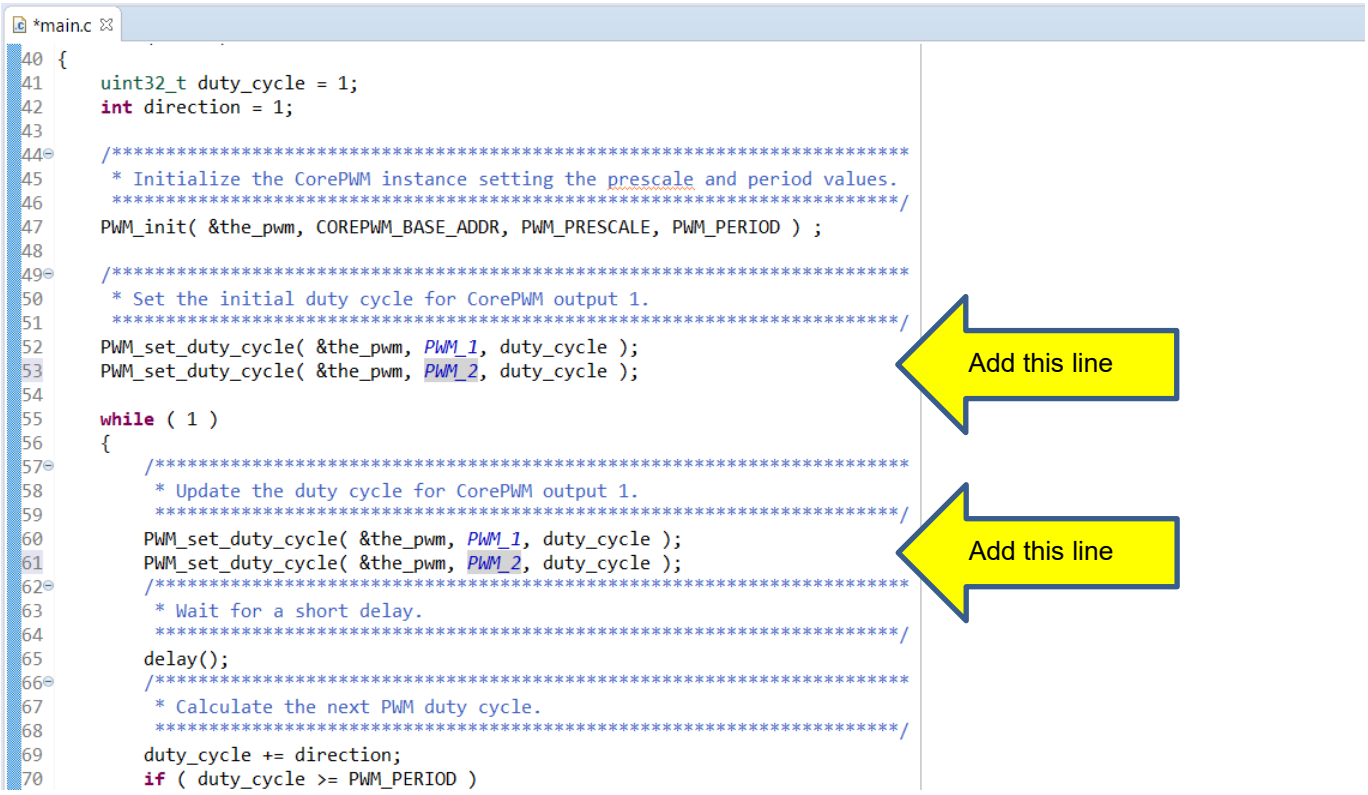
The code in the sample project only drives one LED, but CorePWM was configured to drive two outputs. Follow the steps below to modify the application to drive two LEDs.

Step 8a: Suspend the software application by clicking **Run > Suspend** from the SoftConsole menu clicking the Suspend icon ( ).

Step 8b: Scroll in main to the section with the comment "`* Set the initial duty cycle for CorePWM output 1.`" On line 50. This line configures one CorePWM output. Copy the line "`PWM_set_duty_cycle( &the_pwm, PWM_1, duty_cycle );`" and paste it below the existing line. Modify the line to drive PWM_2.

Step 8c: Scroll to the section with the comment "`* Update the duty cycle for CorePWM output 1.`" On line 58.

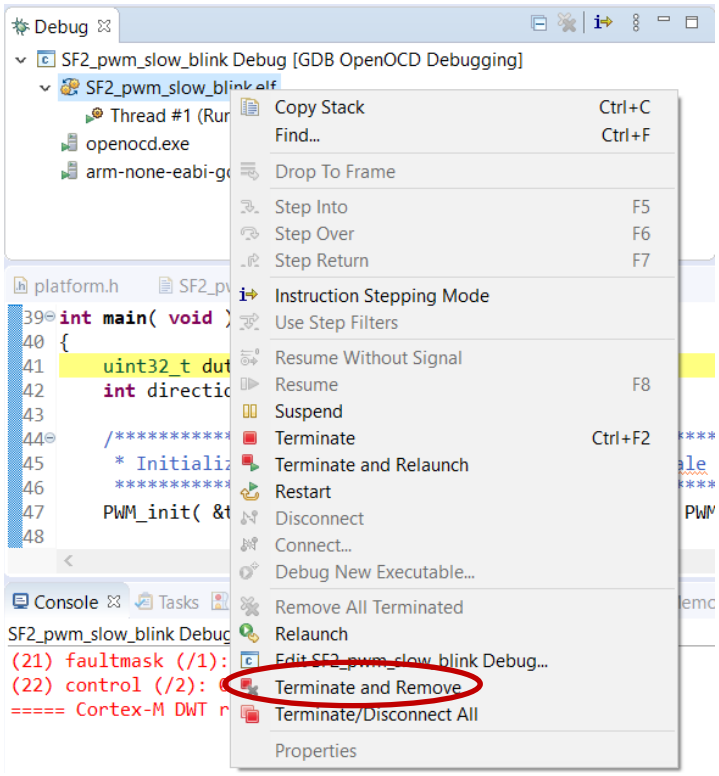
Copy the line "`PWM_set_duty_cycle( &the_pwm, PWM_1, duty_cycle );`"and paste it below the existing line. Modify the line to drive PWM_2. Save the file after editing.



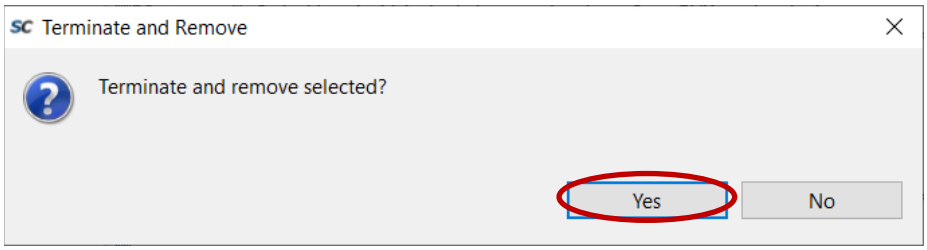

```
40 {
41     uint32_t duty_cycle = 1;
42     int direction = 1;
43
44     /*******************************************************************************
45      * Initialize the CorePWM instance setting the prescale and period values.
46      ******************************************************************************/
47     PWM_init( &the_pwm, COREPWM_BASE_ADDR, PWM_PRESCALE, PWM_PERIOD ) ;
48
49     /*******************************************************************************
50      * Set the initial duty cycle for CorePWM output 1.
51      ******************************************************************************/
52     PWM_set_duty_cycle( &the_pwm, PWM_1, duty_cycle );
53     PWM_set_duty_cycle( &the_pwm, PWM_2, duty_cycle );
54
55     while ( 1 )
56     {
57         /*******************************************************************************
58          * Update the duty cycle for CorePWM output 1.
59          ******************************************************************************/
60         PWM_set_duty_cycle( &the_pwm, PWM_1, duty_cycle );
61         PWM_set_duty_cycle( &the_pwm, PWM_2, duty_cycle );
62         /*******************************************************************************
63          * Wait for a short delay.
64          ******************************************************************************/
65         delay();
66         /*******************************************************************************
67          * Calculate the next PWM duty cycle.
68          ******************************************************************************/
69         duty_cycle += direction;
70         if ( duty_cycle >= PWM_PERIOD )
```

Step 8d: Repeat the previous steps to build the project. Terminate and re-launch the debugger. LED1 and LED2 on the Hello FPGA board will turn on and off together.

Step 8e: When finished, terminate the application by selecting SF2_pwm_slow_blink.elf under the Debug view in the SoftConsole Debug Perspective. Right-click and choose **Terminate and Remove** to stop the debugger.



Click **Yes** when prompted about Terminating and removing.

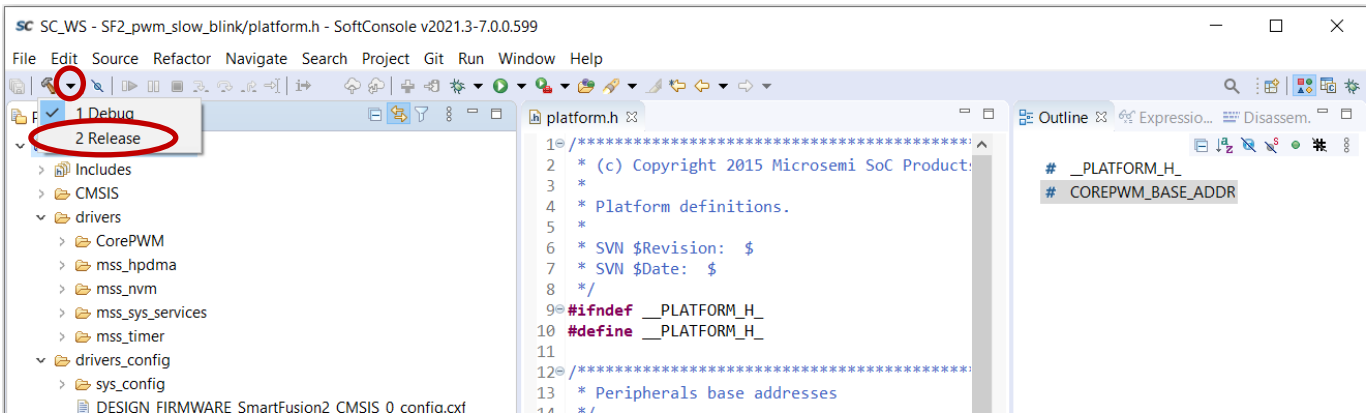


STEP 9: Executing code from the SmartFusion® 2 eNVM

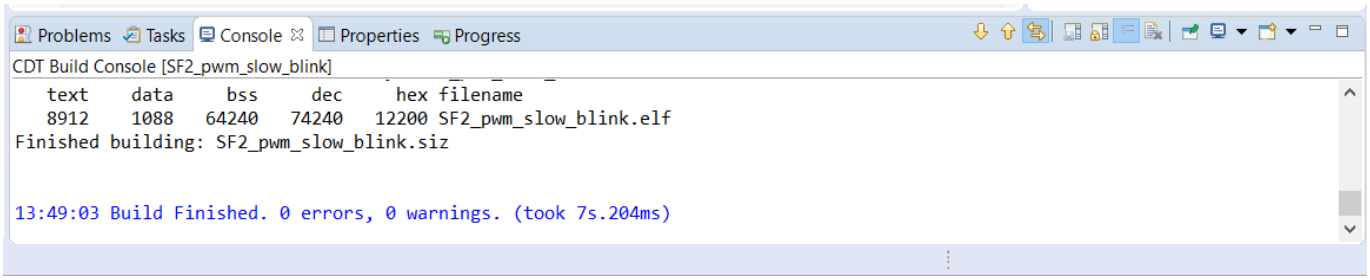
The application can also execute from the SmartFusion 2 eNVM. The Cortex®-M3 has six hardware breakpoints that can be used to debug code. This step describes how to execute code from the SmartFusion 2 eNVM.

Step 9a: Select the SoftConsole C/C++ Perspective (**Window > Perspective > Open Perspective > C/C++**) from the SoftConsole menu.

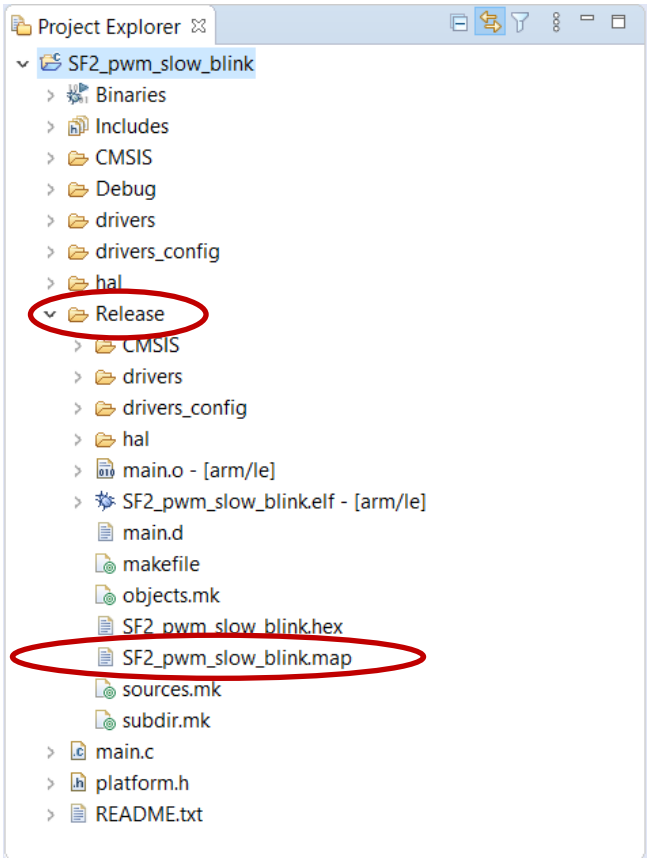
Step 9b: Build the project with the Release build configuration by selecting *SF2_pwm_slow_blink* in the Project Explorer then clicking the triangle next to the hammer icon. Click **Release** to build the project using the Release build configuration. The project Release build configuration settings use the debug-in-microsemi-smartfusion2-envm.ld linker script to build an executable that will run in the SmartFusion 2 eNVM.



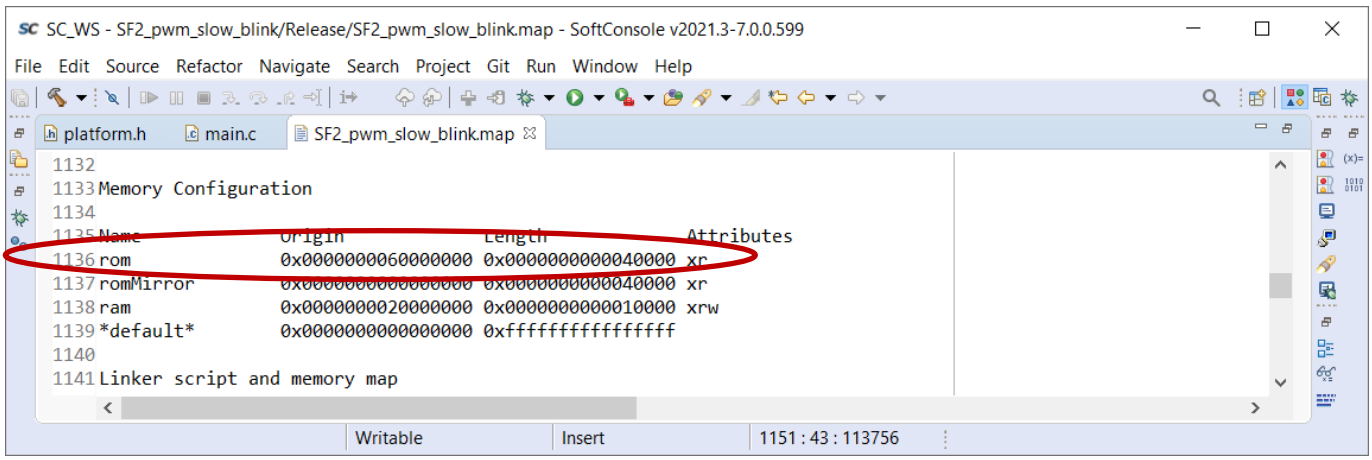
Step 9c: Confirm that there are no errors listed in the Console view. Select the Problems view for additional information on any errors that are reported. Correct the errors and rebuild the project.



A folder named Release containing the executable for the Release build configuration will be visible in the Project Explorer.



Step 9d: Double-click SF2_pwm_slow_blink.map in the Release folder (highlighted in the figure above) to open the file in the SoftConsole editor. Scroll down to the Memory Configuration section. The section should appear as shown in the figure below.

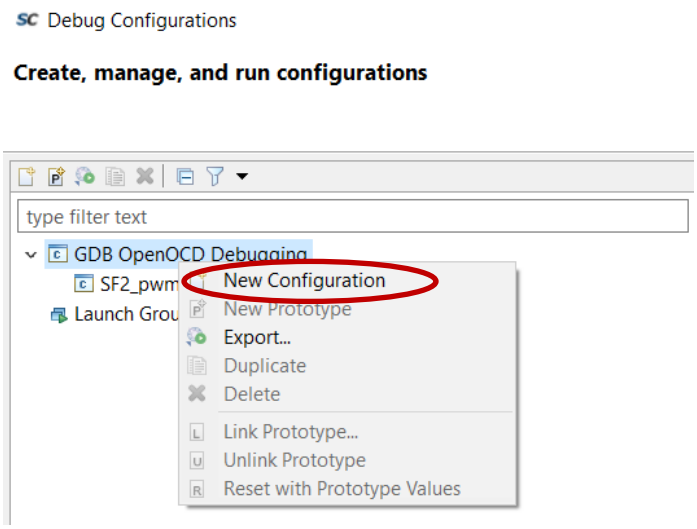


The memory configuration for the Release build includes a rom section at 0x60000000, which is the address of the SmartFusion® 2 eNVM.

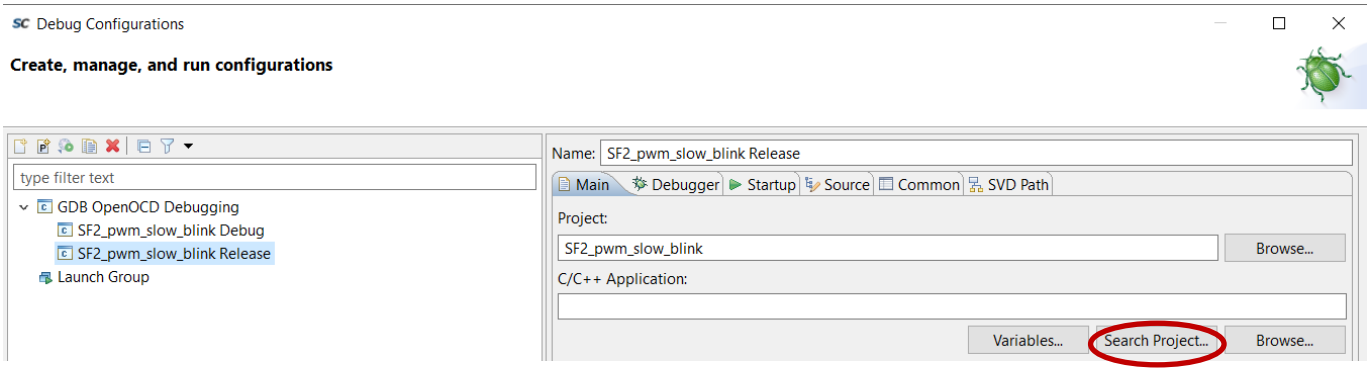
Step 9e: A debug launch configuration must be created for the Release build configuration.

Select SF2_pwm_slow_blink in the SoftConsole Project Explorer View then select **Run > Debug Configurations…** from the SoftConsole menu to open the Debug Configurations dialog box.

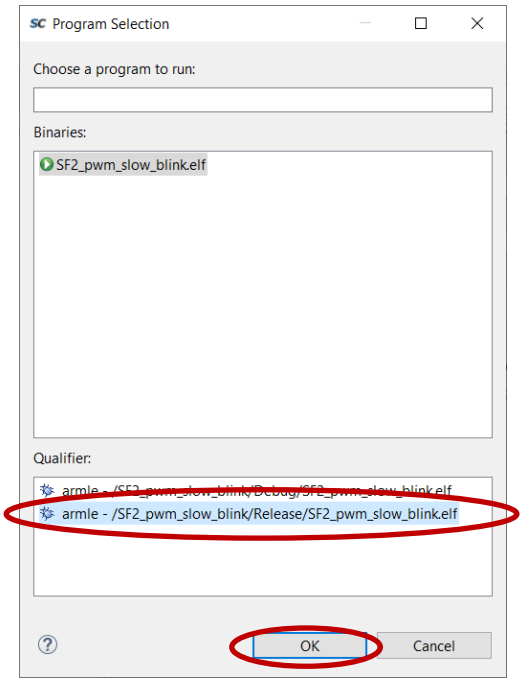
Step 9f: Select GDB OpenOCD Debugging in the Debug Configurations dialog box then right-click and select **New Configuration** to create a new debug launch configuration for the Release build configuration.



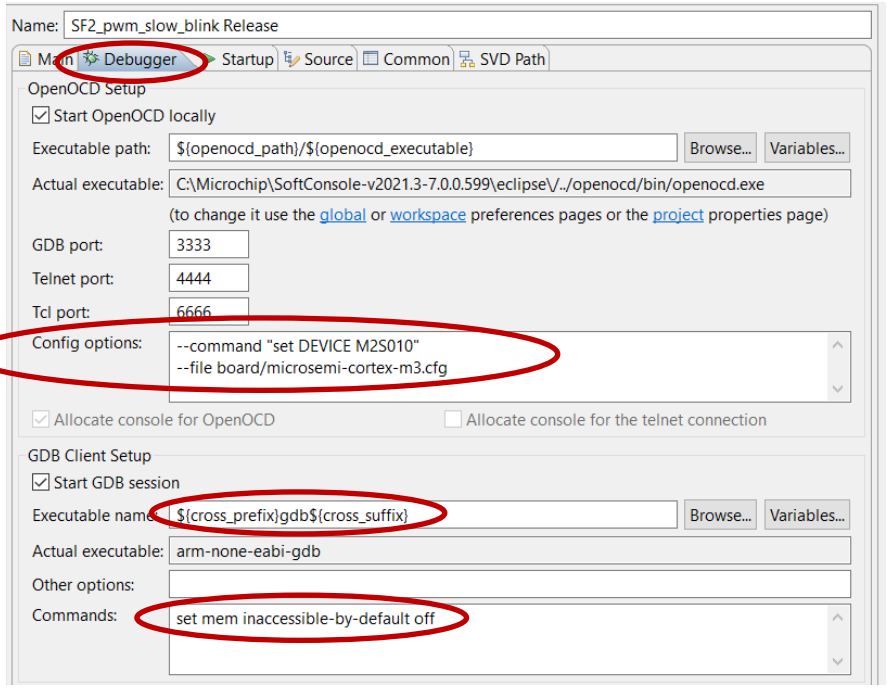
Step 9g: Click the Search Project button under the C/C++ Application field on the Main tab of the Debug Configurations dialog box.



Step 9h: The Program Selection dialog box will open.
Select armle - / SF2_pwm_slow_blink/Release/SC4_pwm_slow_blink.elf then click **OK**.

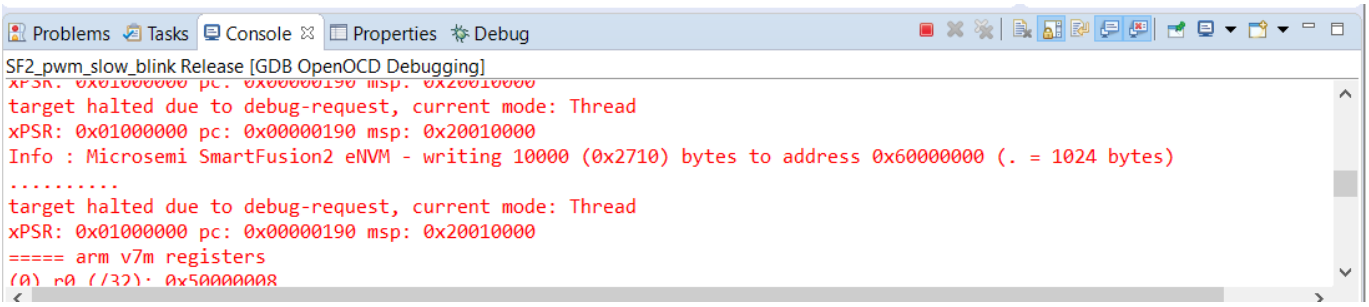


Step 9i: Select the Debugger tab of the Debug Configurations dialog box. Confirm that the Config options and GDB Client Setup fields contain the settings described previously for the silicon on the target board.



Step 9j: Click **Apply** to save the changes. Click **Debug** to launch the Debugger. The SoftConsole Debug perspective will open, and the code will be programmed to the SmartFusion® 2 eNVM.

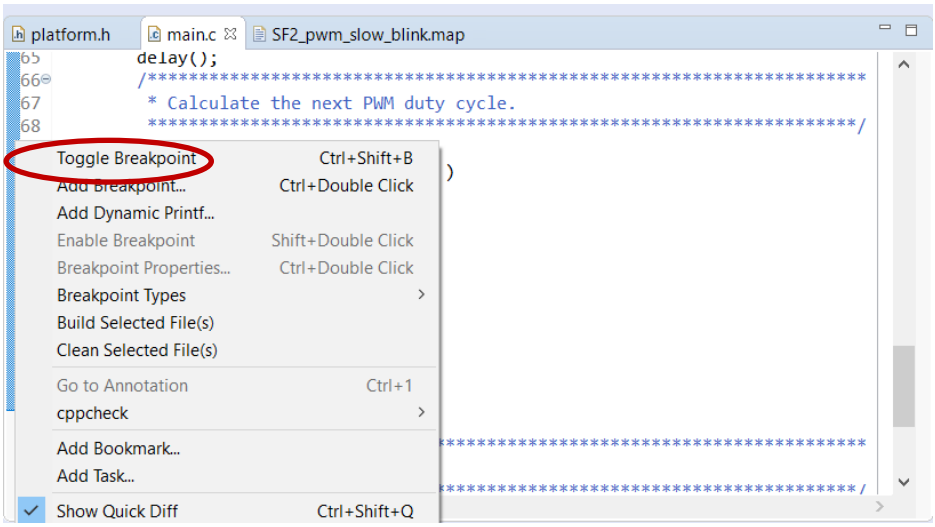
Messages will appear in the Console view while the code is being downloaded. When finished the program will be suspended at the first executable line of main().



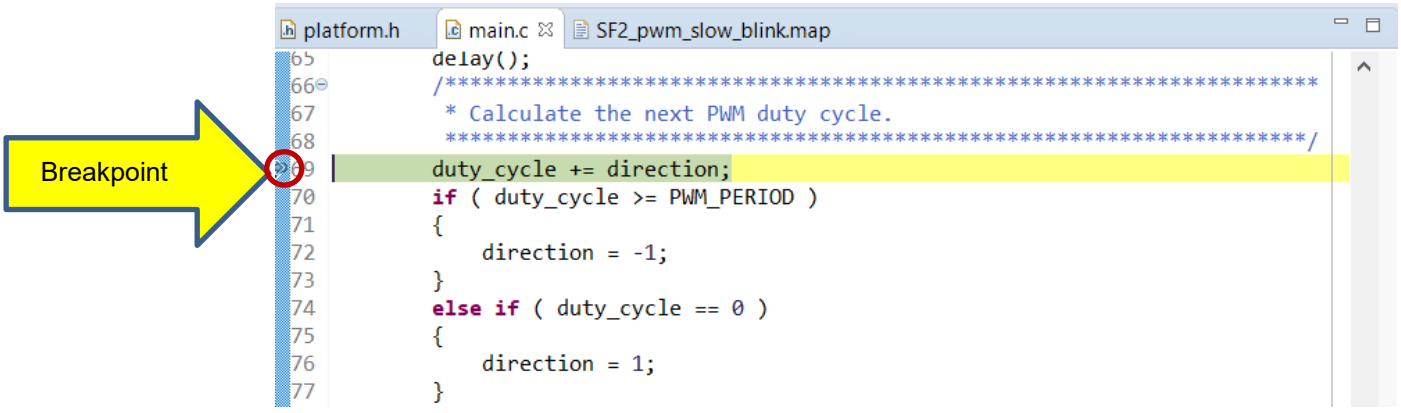
Note: If an error occurs when launching the Debugger, open the Debug Configurations dialog box (**Run > Debug Configurations**) and confirm that the Config Options field settings match the instructions.

Step 9k: Open the Debug Perspective (**Window > Perspective > Open Perspective > Debug**). Click the Resume icon to run the application. Note that the application runs the same as previously when it was executing from the SmartFusion® 2 eSRAM memory.

Step 9l: Try setting breakpoints while executing the code. To set a breakpoint, click next to a line number in main.c, then right click and select **Toggle Breakpoint**.



The execution will stop at the line with the breakpoint. An arrow will indicate the break location.



Step 9m: Click the Resume button () to resume code execution. Clear the breakpoint by moving the mouse to the breakpoint location, right-clicking and selecting **Toggle Breakpoint**.

Step 9n: When finished, terminate the debugger by selecting SF2_pwm_slow_blink.elf in the Debug window, then right clicking and selecting **Terminate and Remove**. Click **Yes** in the Terminate and Remove dialog box.

STEP 10: Creating an external DDR Memory Build Configuration

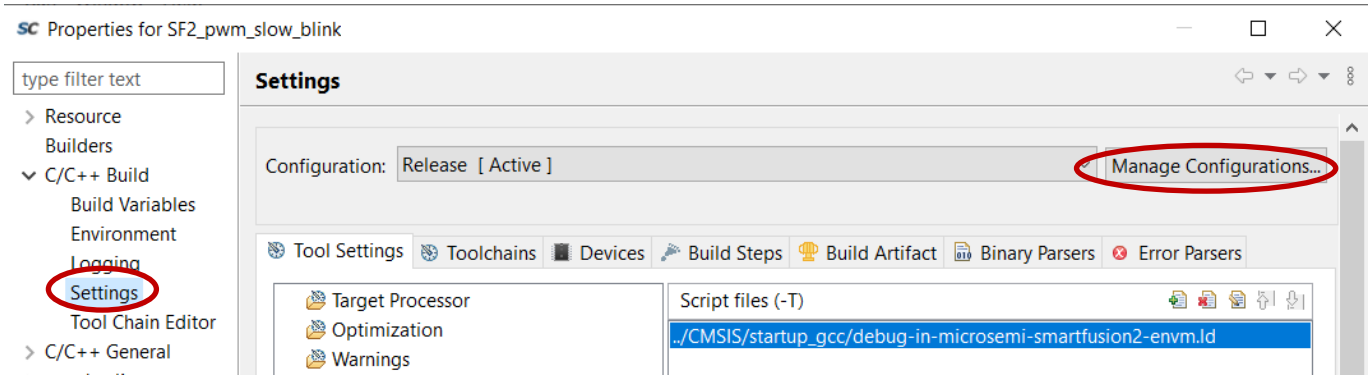
The Hello FPGA board has 1GB of DDR3 memory. The SmartFusion® 2 DDR memory controller was configured with System Builder in Lab 1. This step describes how to execute code from the DDR3 memory. The steps are:

1. Select the SoftConsole C/C++ Perspective.
2. Create a new SoftConsole build configuration that uses the linker script for the external DDR memory.
3. Create a SoftConsole Debug launch configuration for the DDR memory executable.
4. Launch the SoftConsole Debugger.

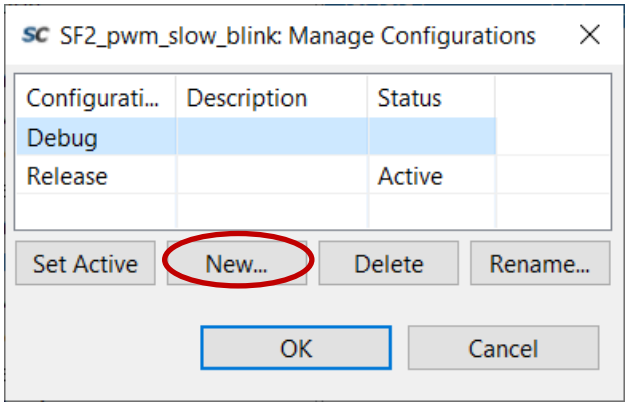
Step 10a: Select the SoftConsole C/C++ Perspective (**Window > Perspective > Open Perspective > C/C++**).

Step 10b: Select SF2_pwm_slow_blink in the SoftConsole Project Explorer and open the Properties dialog box (**Project > Properties**).

Step 10c: Expand **C/C++ Build** in the properties dialog box and select **Settings** in the Properties for SF2_pwm_slow_blink dialog box.

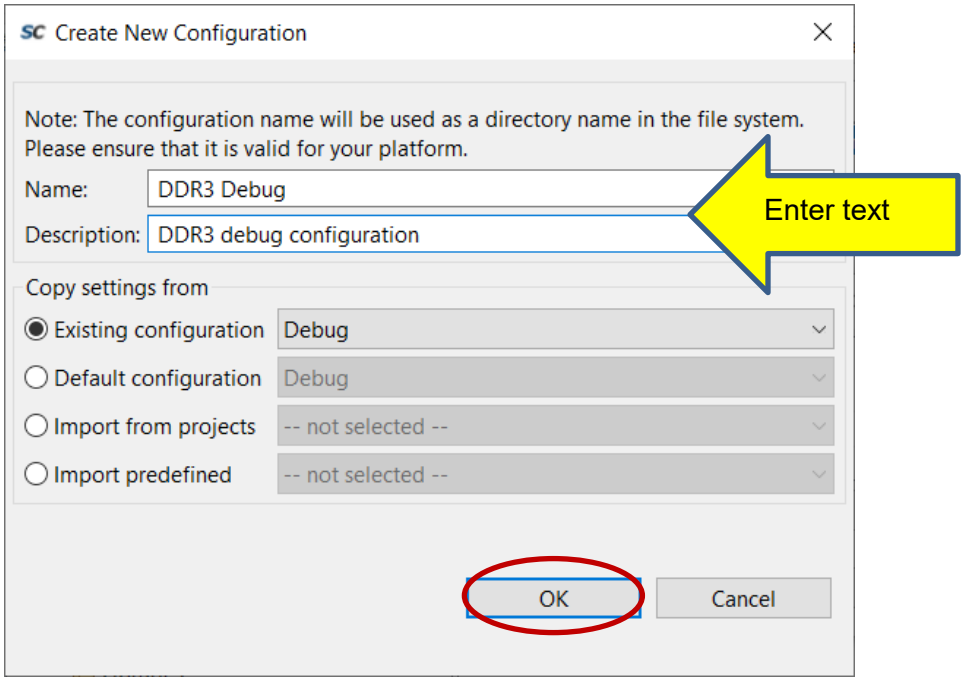


Step 10d: Click the **Manage Configurations** button (circled above). Click **New** in the SF2_pwm_slow_blink Manage Configurations dialog box.

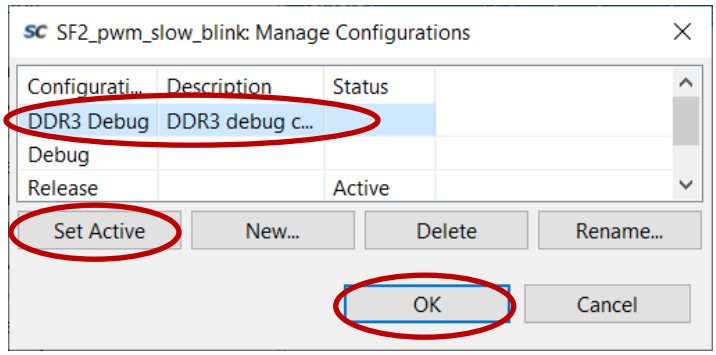


Step 10e: Enter the following in the Create New Configuration dialog box then click **OK**:

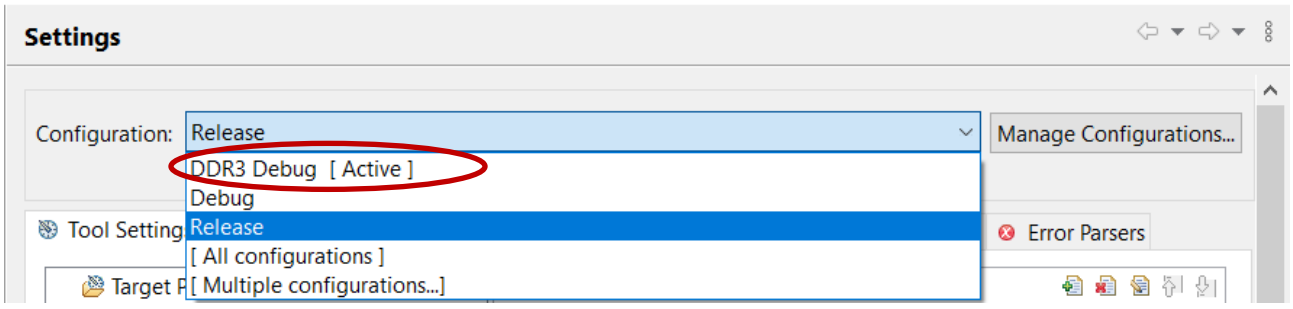
- Name: DDR3 Debug
- Description (optional): DDR3 debug configuration
- Copy settings from Existing configuration: Debug



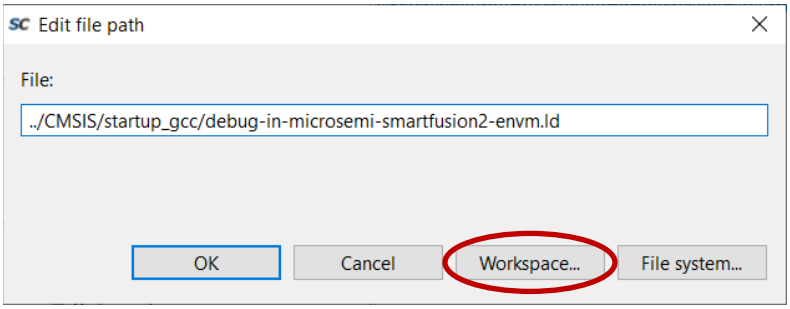
Step 10f: Select DDR3 Debug in the SF2_pwm_slow_blink Manage Configurations dialog box and click **Set Active** then **OK**.



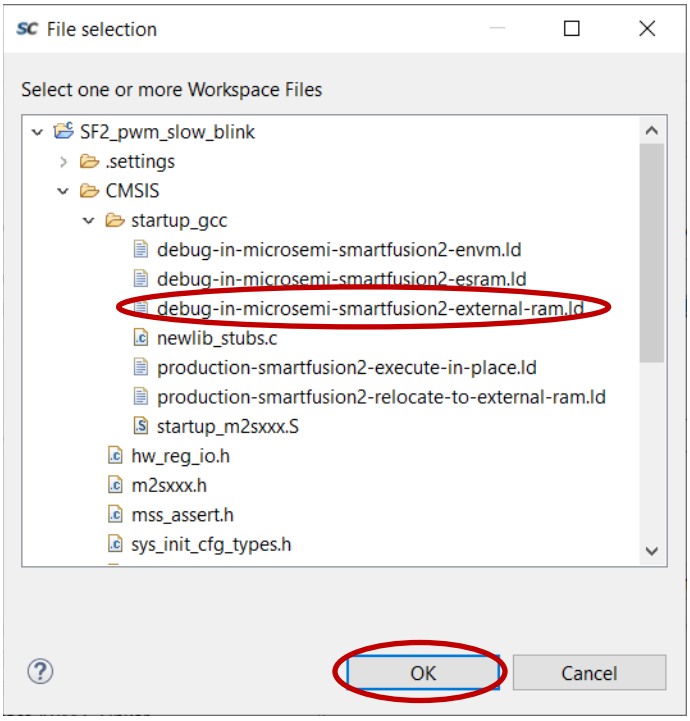
Step 10g: Use the pull-down menu in the Configuration field of the Settings dialog box to select the DDR3 Debug (circled in the figure below).



Step 10h: Select **General** under Cross ARM C Linker and click the edit icon ( ). The Edit file path dialog box will open.

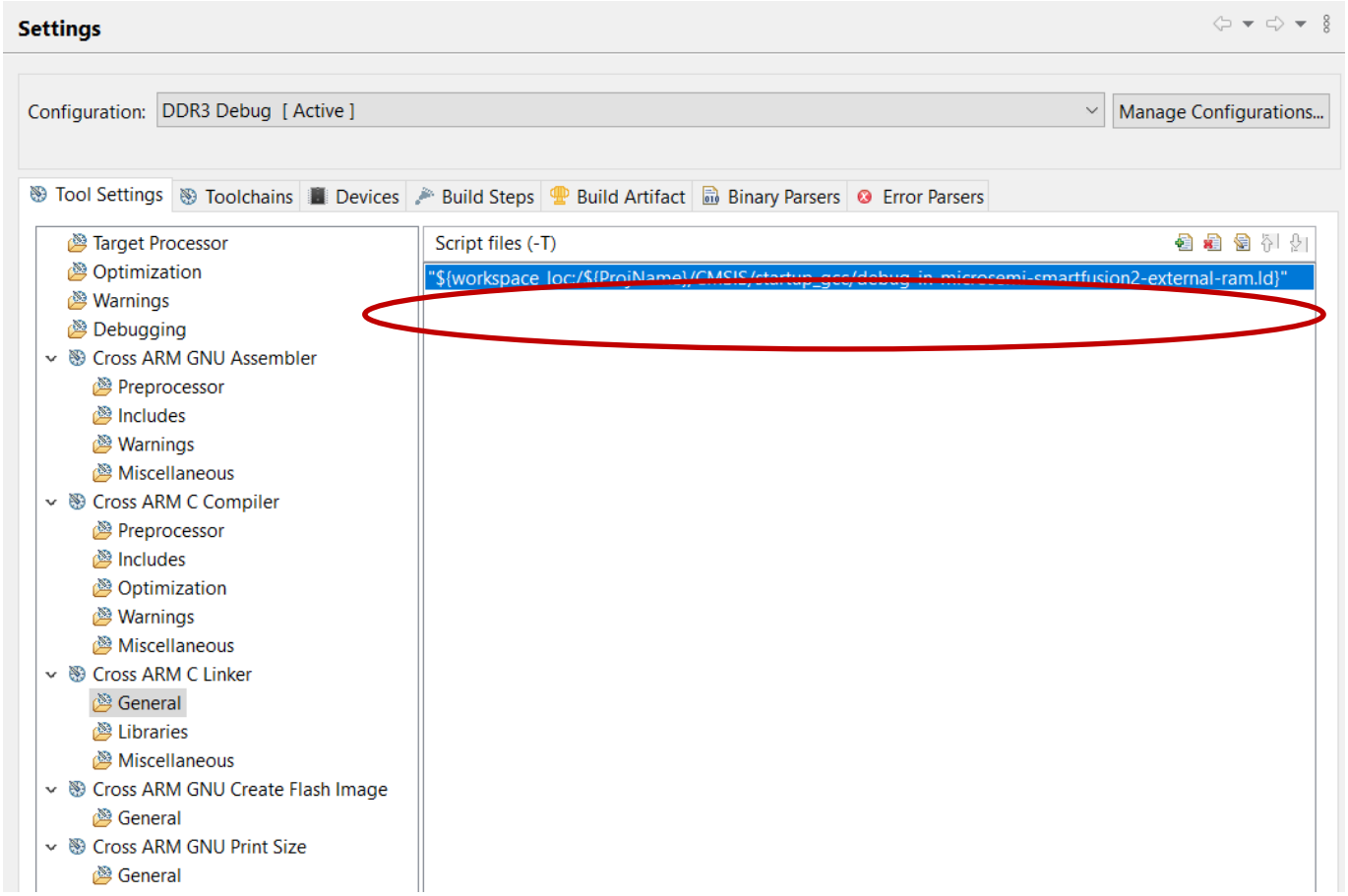


Step 10i: Click the Workspace button to open the File selection dialog box. Navigate to SF2_pwm_slow_blink > CMSIS > startup_gcc and select debug-in-microsemi-smartfusion2-external-ram.ld. This linker script builds an executable that will run from the external DDR3 memory.



Step 10j: Click **OK** in the File selection dialog box and in the Edit file path dialog box.

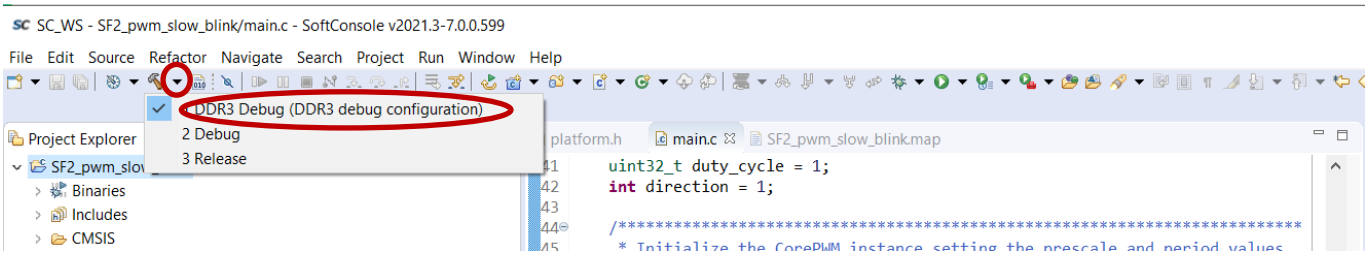
Confirm that the Script files field contains ${workspace_loc:/${ProjName}/CMSIS/startup_gcc/debug-in-microsemi-smartfusion2-external-ram.ld}.



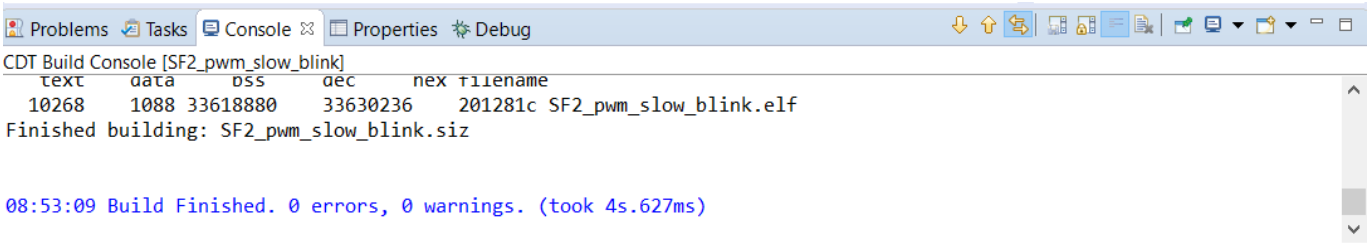
Click **Apply and Close** to close the Properties for SF2_pwm_slow_blink dialog box. Click **No** if prompted about building now.

Step 10k: After configuring the project settings, the next step is to build the project using the DDR3 Debug build configuration.

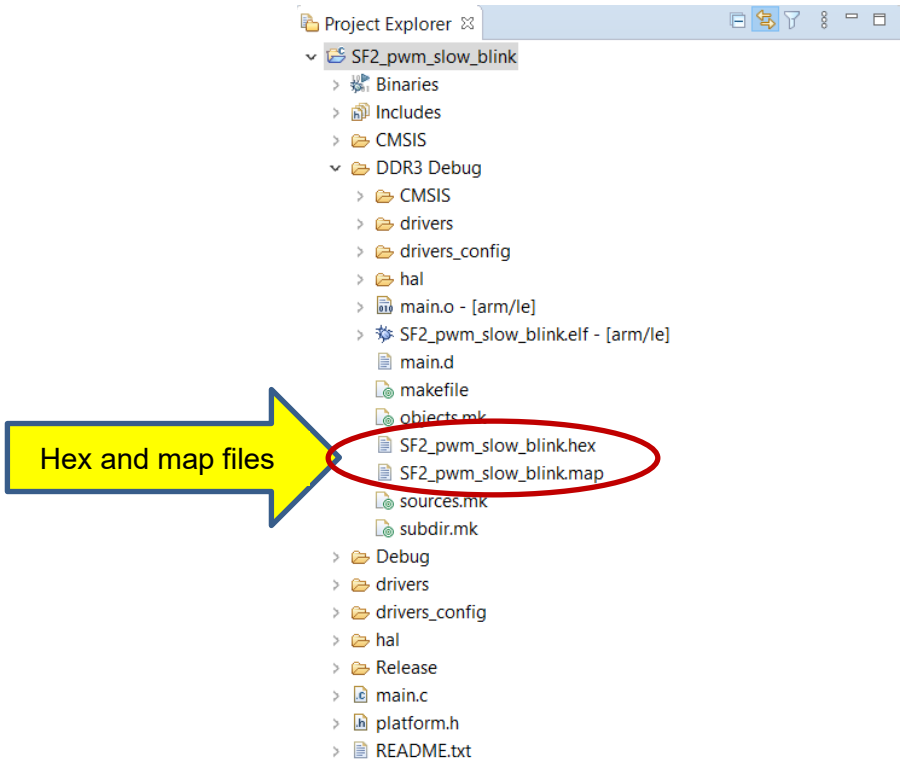
Step 10l: Select *SF2_pwm_slow_blink* in the Project Explorer then click the triangle next to the hammer icon. Click **DDR3 Debug** to build the project using the DDR3 Debug build configuration.



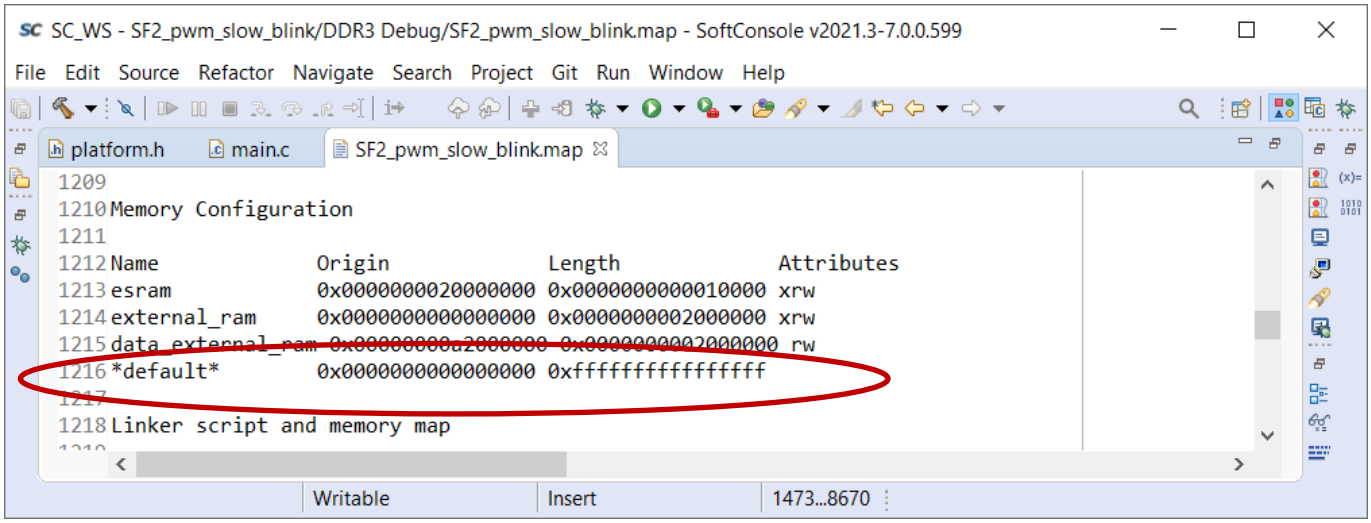
Confirm that there are no errors or warnings in the Console view. If errors occur, select the Problems view to get additional information. Correct the errors then build the project again.



A folder named DDR3 Debug will be visible in the SoftConsole Project Explorer. The folder will contain the hex file and map file for the DDR3 Debug build.



Step 10m: Double-click SF2_pwm_slow_blink.map in the DDR3 Debug folder to open the file in the SoftConsole editor. Scroll down to the Memory Configuration section. The section will appear as shown in the figure below.



The memory configuration for the DDR3 Debug build includes a data_external_ram section at 0xa2000000, which within the SmartFusion® 2 DDR_0_SPACE_0. This can be seen in the SF2_MSS data sheet which can be viewed in the Libero® SoC Project on the Report View.

SF2_MSS_sb_0/ SF2_MSS_sb_MSS_0/ CM3

Target	Offset Address	Range	High Address	DRC
DDR_0_SPACE_3	0xD000_0000	256MB	0xDFFF_FFFF	
DDR_0_SPACE_2	0xC000_0000	256MB	0xCFFF_FFFF	
DDR_0_SPACE_1	0xB000_0000	256MB	0xBFFF_FFFF	
DDR_0_SPACE_0	0xA000_0000	256MB	0xAFFF_FFFF	
AHB2ENVM1_REGISTERS	0x600C_0000	256KB	0x600F_FFFF	
AHB2ENVM0_REGISTERS	0x6008_0000	256KB	0x600B_FFFF	
ENTIRE_ENVM	0x6000_0000	256KB	0x6003_FFFF	

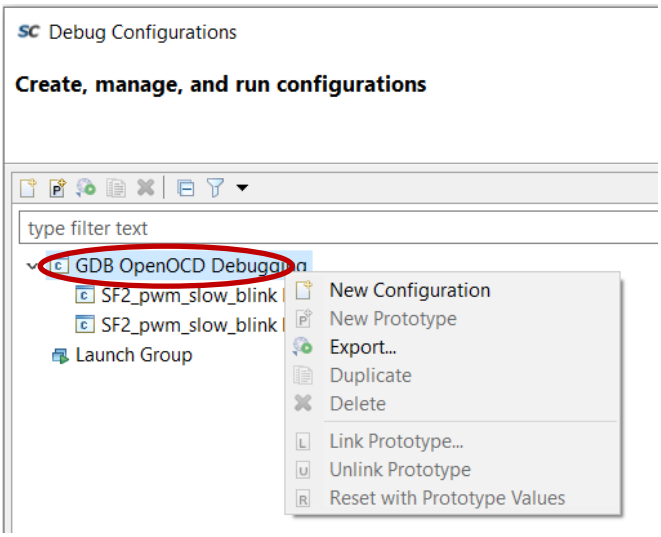
Note: If the address range displayed in the .map file for the DDR3 Debug build does not match the address shown above, confirm that the correct linker script was used to build the project.

STEP 11: Creating a debug launch configuration for external DDR Memory

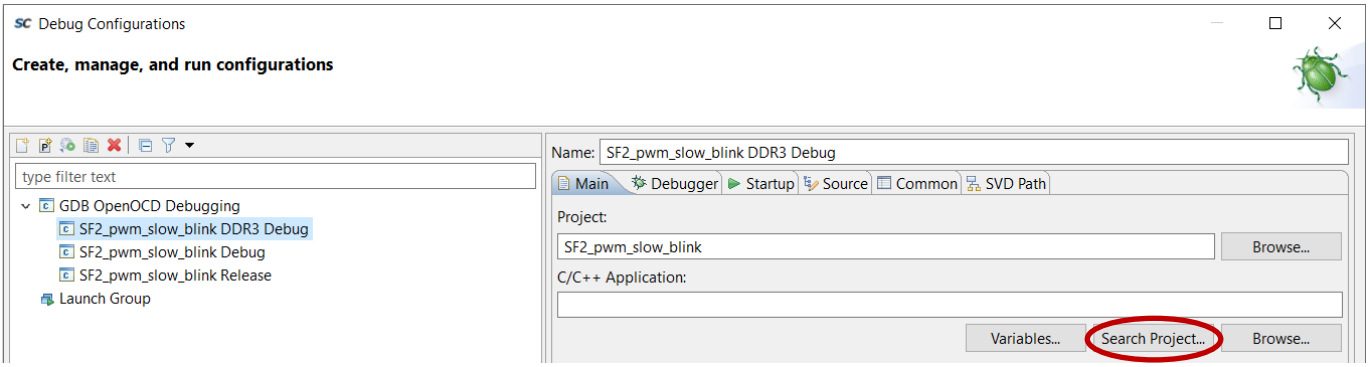
A debug launch configuration must be created for the DDR3 Debug build configuration.

Step 11a: Select SF2_pwm_slow_blink in the SoftConsole Project Explorer View then select **Run > Debug Configurations…** to open the Debug Configurations dialog box.

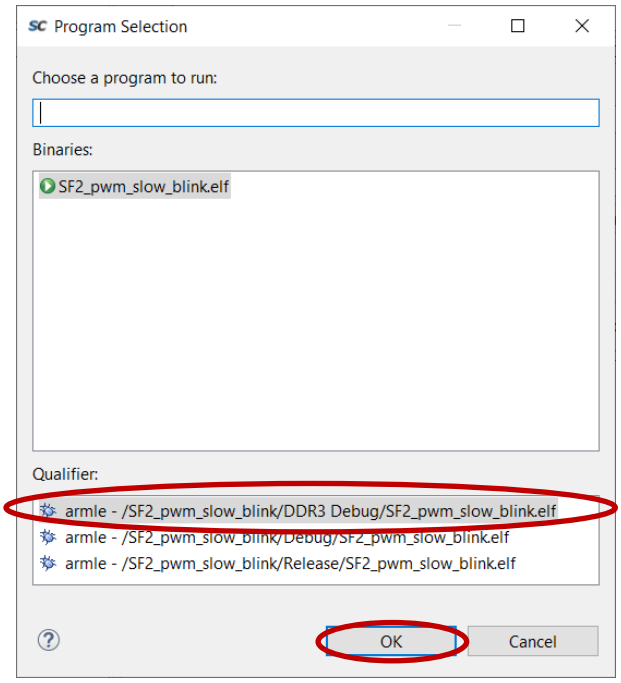
Step 11b: Select GDB OpenOCD Debugging in the Debug Configurations dialog box then right-click and select **New Configuration** to create a new debug launch configuration for the DDR3 build configuration.



Step 11c: Click the Search Project button under the C/C++ Application field on the Main tab of the Debug Configurations dialog box.



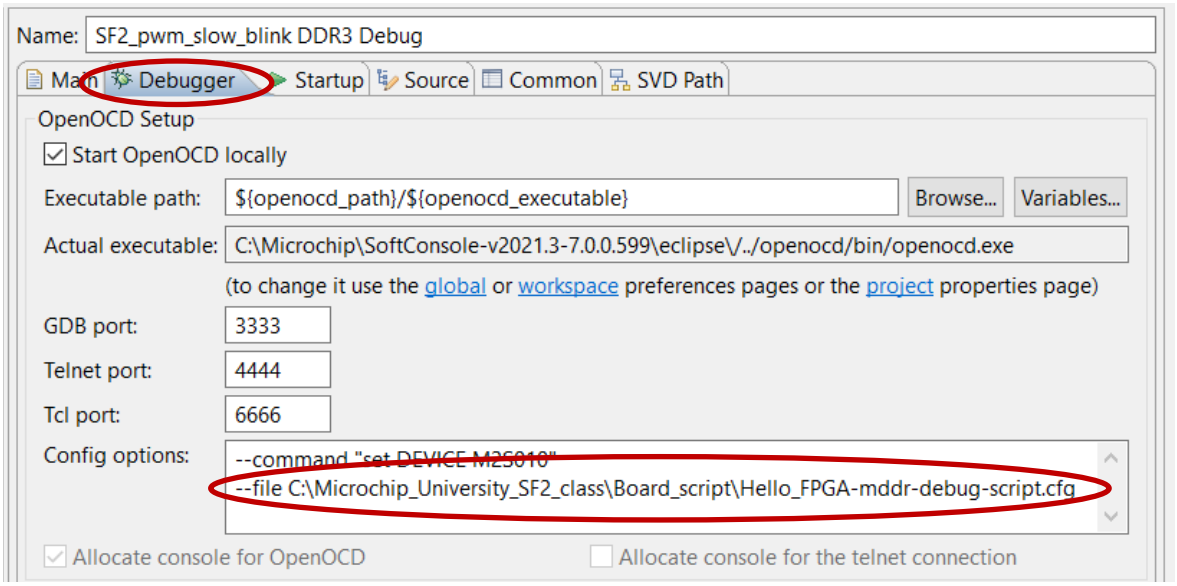
Step 11d: Select armle - /SF2_pwm_slow_blink/DDR3 Debug/SC4_pwm_slow_blink.elf in the Program Selection dialog box then click **OK**.



Step 11e: The Config Options field on the Debugger tab must contain information regarding the target device and a board script that matches the hardware configuration. A board script for the DDR3 configuration is provided in the \Microchip_University_SF2_class\Board_script folder.

Select the Debugger tab of the Debug Configurations dialog box. Modify the Config Options field as follows:

- Enter the "set DEVICE" command `--command "set DEVICE M2S010"`
- Enter the board script file[1]:
  `--file .\Microchip_University_SF2_class\Board_script\Hello_FPGA-mddr-debug-script.cfg`

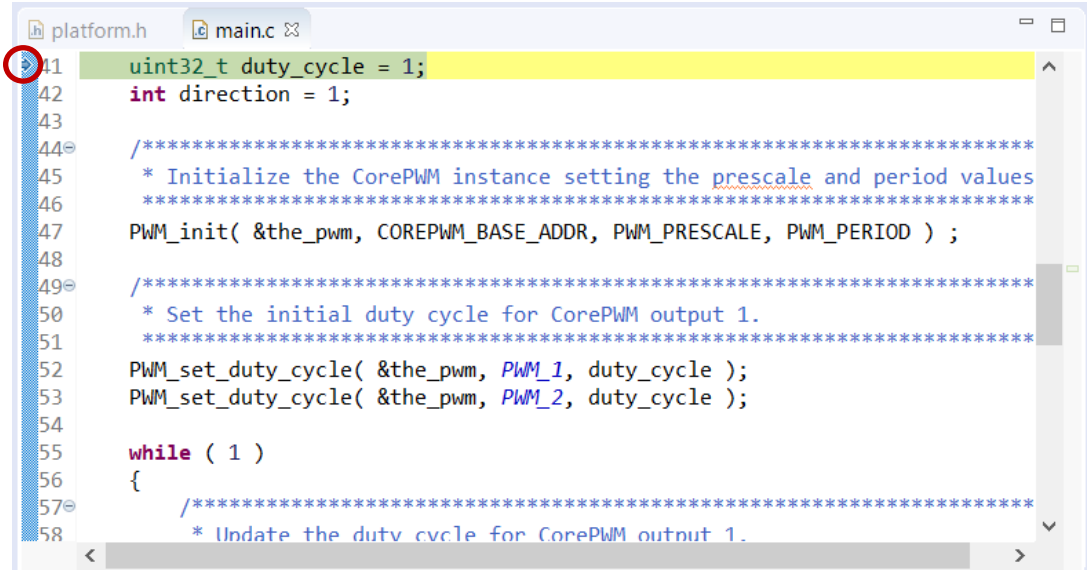


[1]The path shown in the figure on the previous page assumes the lab files were extracted to the C drive. Modify the path as needed if the files were extracted to a different folder.

Step 11f: Click **Apply** to save the changes. Click **Debug** to launch the Debugger. The code will be downloaded to the DDR3 memory on the Hello FPGA kit.

Step 11g: Open the SoftConsole Debug perspective (**Window > Perspective > Open Perspective > Debug**).

The execution will stop at the first executable line in main. An arrow will indicate the break location.



Note: If an error occurs when launching the Debugger, open the Debug Configurations dialog box (**Run > Debug Configurations**) and confirm that the Config Options field settings match the instructions.

Step 11h: Click the Resume button ( ) to resume code execution. Observe the operation of the LEDs.

Step 11i: Try setting a breakpoint while the code is running. Click the Resume button to resume code execution after hitting the breakpoint.

Step 11j: When finished, terminate the debugger by selecting SF2_pwm_slow_blink.elf in the Debug window, then right clicking and selecting **Terminate and Remove**. Click **Yes** in the Terminate and Remove dialog box.

Results:

LED1 and LED2 on the Hello FPGA kit should blink as described in the lab guide.

If errors appear in the Console View after building the project, select the Problems View for additional information. Confirm that there are no errors in the project settings, then build the project again.

Summary:

In this lab you created a SoftConsole workspace and imported the CorePWM sample firmware project then updated the firmware drivers and configuration files. You built executables to run in the SmartFusion® 2 eSRAM, SmartFusion 2 eNVM and the external DDR3 memory on the Hello FPGA board. You created debug launch configurations and ran the code from each of the three memory spaces.

Appendix A – Reference Information

Libero® SoC: https://www.microchip.com/en-us/products/fpgas-and-plds/fpga-and-soc-design-tools/fpga/libero-software-later-versions

SoftConsole: https://www.microchip.com/en-us/products/fpgas-and-plds/fpga-and-soc-design-tools/soc-fpga/softconsole#Download%20Software

Hello FPGA kit: https://www.microchip.com/en-us/development-tool/M2S-HELLO-FPGA-KIT

FlashPro4: https://www.microchip.com/en-us/development-tool/FLASHPRO4

FlashPro5: https://www.microchip.com/en-us/development-tool/FLASHPRO5

FlashPro6: https://www.microchip.com/en-us/development-tool/FLASHPRO6

Lab source files (Microchip_University_SF2_class.zip) contains the following:

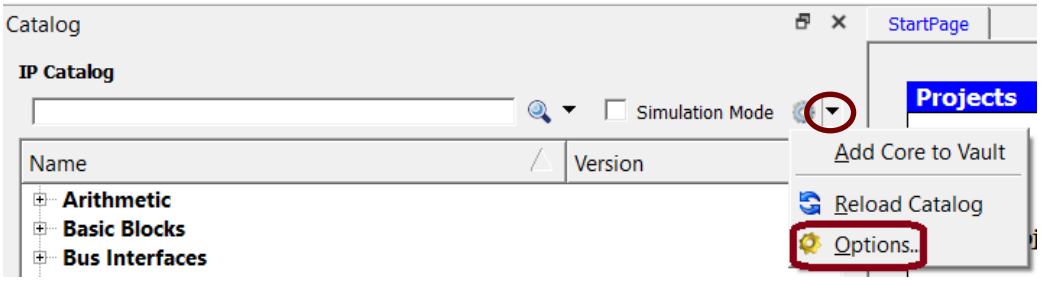
Microchip_University_SF2_class
|
+----- Board_script – contains a board script file for debugging from DDR3 memory
+----- Hex_File – contains a Cortex®-M3 hex file
+----- IP_cores – contains IP cores required to complete the lab
+----- Memory_configuration – contains a configuration file for the DDR3 memory on the Hello FPGA kit
+----- Solution – contains completed designs for the labs 1 and 3 (Note SF2-M3_BaseDesign_Lab3.zip includes Lab3 and Lab2 solutions)

Appendix B – Checking and Installing the IP cores

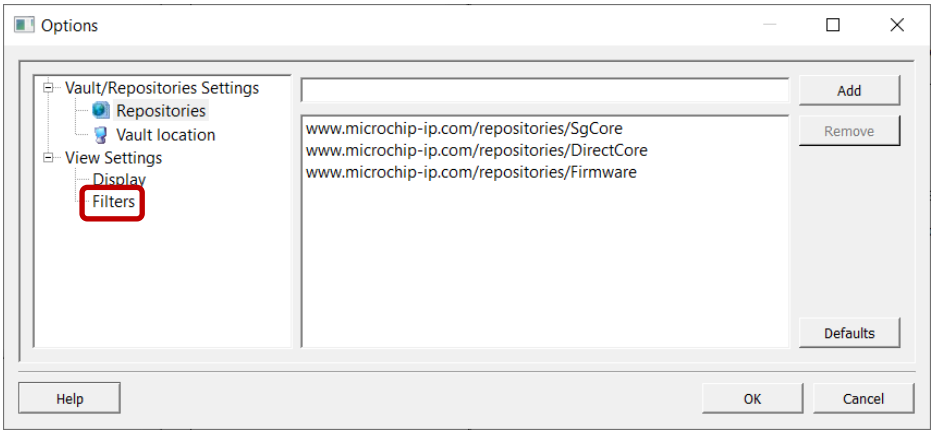
Step 1: Open the Libero SoC software by clicking the shortcut icon on the desktop**.**

Step 2: Open the Libero SoC IP catalog (**View > Windows > Catalog**).

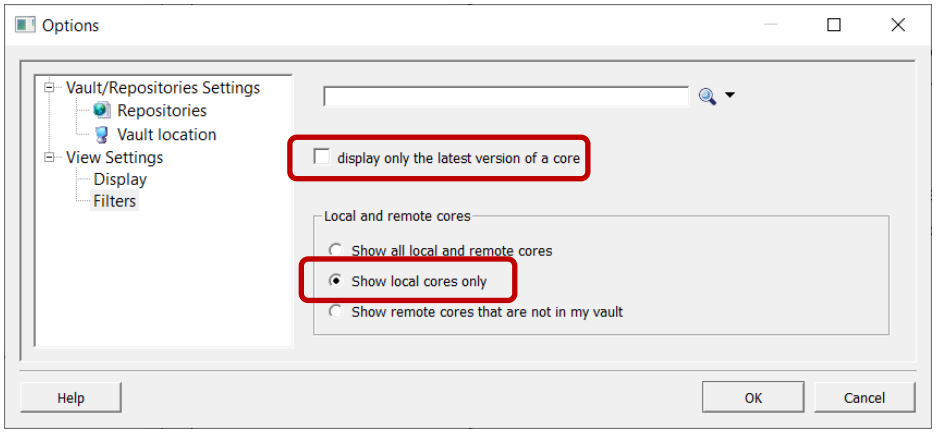
Step 3: Open the IP catalog options page by clicking next to the Gear symbol in the IP catalog. Select Options from the pull-down menu.



Step 4: Select **Filters** in the Options dialog box.

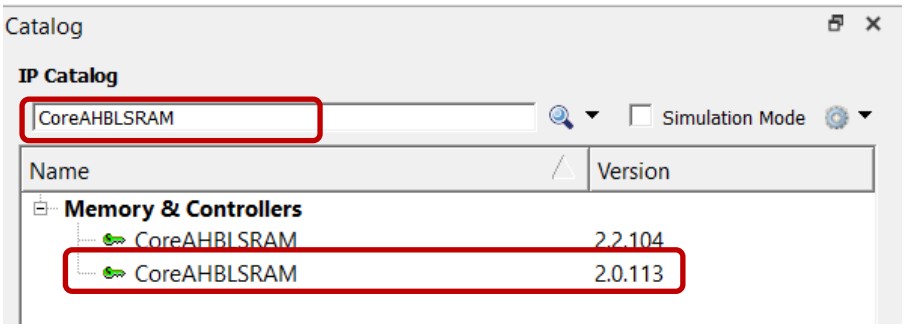


Step 5: Un-check display only the latest version of a core to display all versions of a core in the IP catalog. Check Show local cores only. Click **OK** to close the Options dialog box.

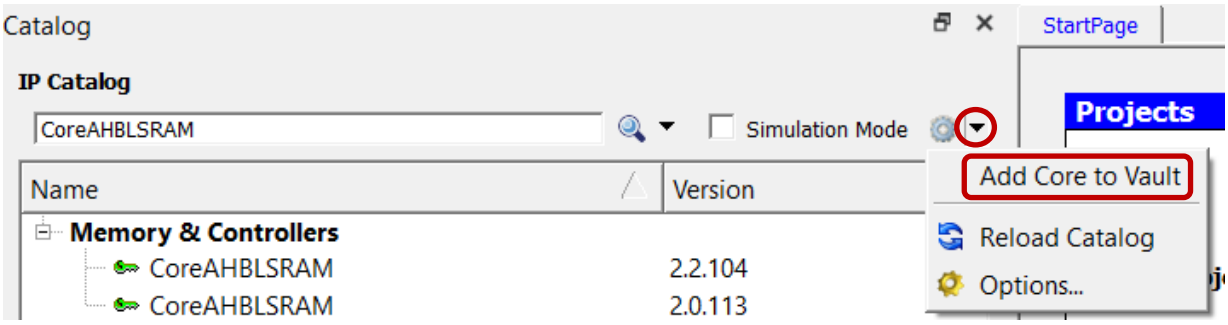


Step 6: Enter the name of the IP core in the IP catalog search field to search for the cores listed in the table below. Confirm the version shown is in the IP catalog.

Core Name	Core Version
CoreAHBLSRAM	2.0.113
CoreI2C	7.0.102
CoreSPI	3.0.156
CoreGPIO	3.0.120
CoreTimer	1.1.101
CoreUARTapb	5.2.2
CorePWM	4.1.106

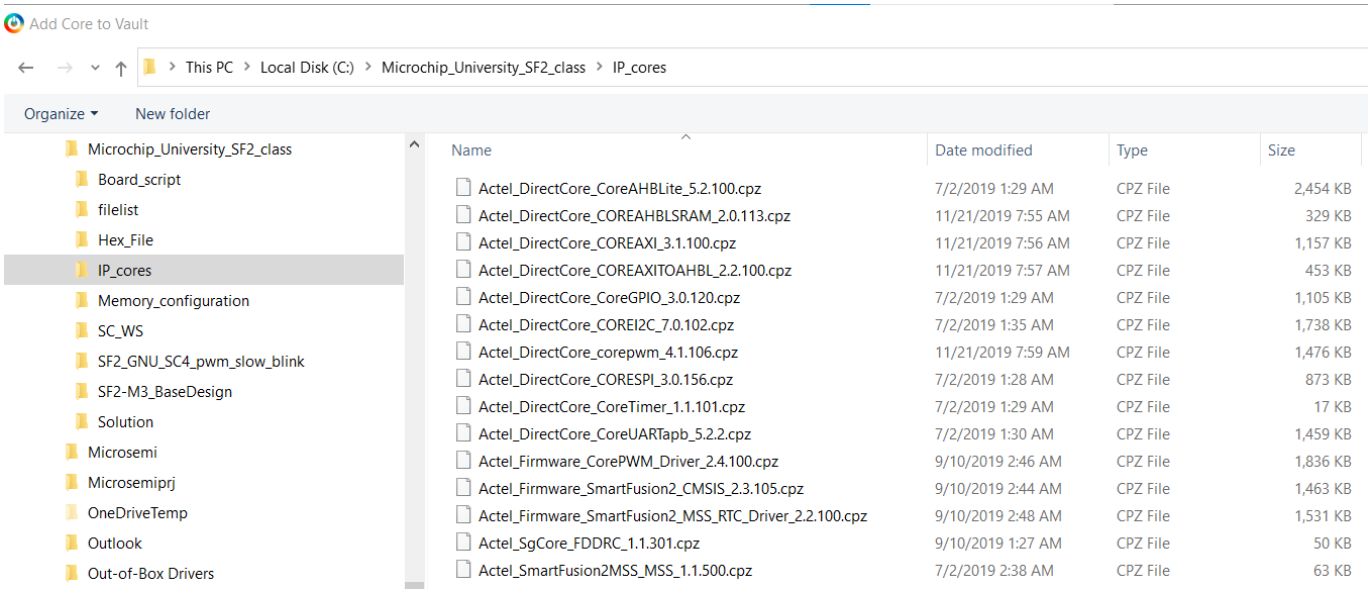


Step 7: Record any missing IP cores. Add missing cores by clicking next to the Gear symbol in the IP catalog. Select **Add Core to Vault** from the pull-down menu.



The Add Core to Vault dialog box will open.

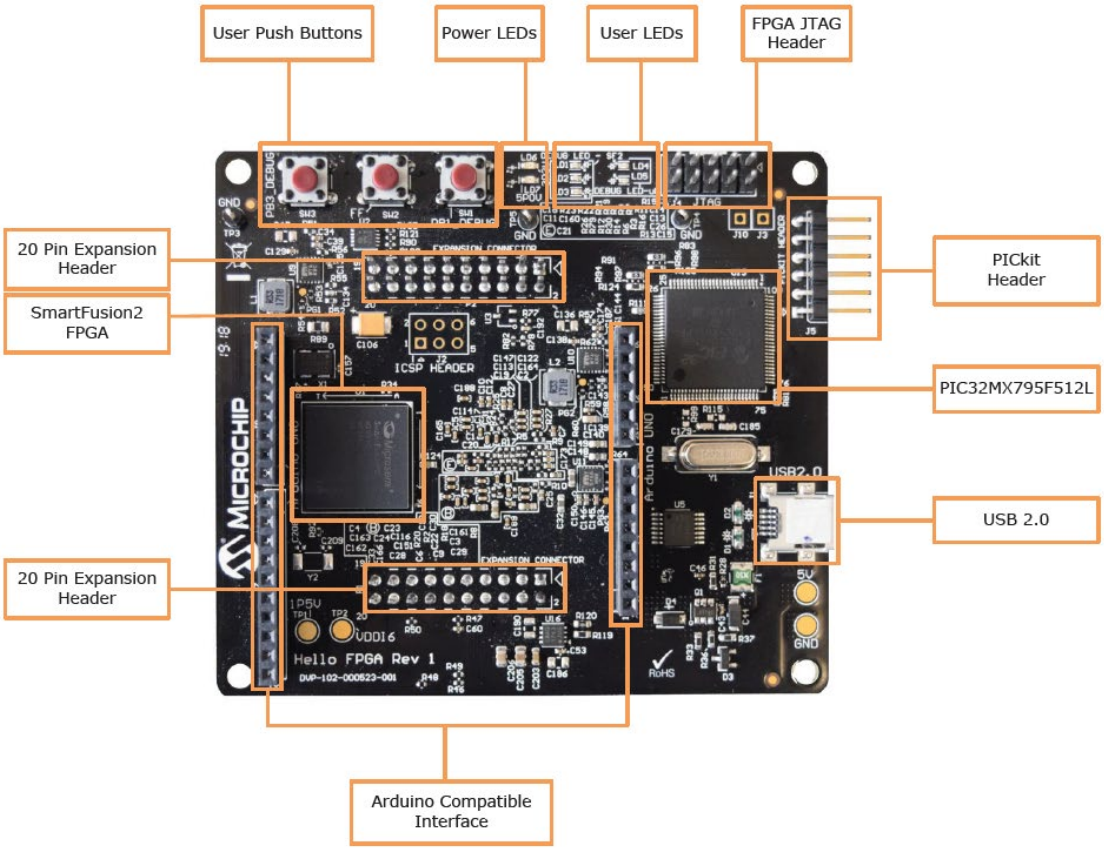
Navigate to the .\Microchip_University_SF2_class\IP_cores folder and select the missing core or cores. The cores will have a cpz extension. Select multiple cores by holding the CTRL key while clicking the core name.



Step 8: Click **OK** to close the Add Core to Vault dialog box.

Step 9: Go to “Creating a Libero SoC project” on page 4 after checking and installing the IP cores.

Appendix C - Hello FPGA kit Setup



Board Documentation, Hello FPGA GUI and Design Files: https://www.microchip.com/en-us/development-tool/M2S-HELLO-FPGA-KIT

Step 1: Prior to programming (and powering up) the Hello FPGA board, make the following connections:

a. Connect the USB cable from the host PC to the USB Mini A connector on the Hello FPGA board.
b. Connect the ribbon cable for a FlashPro programmer to the 10 pin FPGA JTAG Header. Note that pin 1 is located on the end closest to the PICKit header.
c. Connect a USB cable between the FlashPro programmer and the host PC.

Appendix D - Configure Linux to Detect and Use FlashPro5 Programmer Hardware

If you want a regular user (without root permission) to program the Flash based FPGA devices with FlashPro5 hardware, you must run the udev_install script on the Linux machine as root. The udev_install script helps you set up a udev rule file for the FLashPro5 hardware.
This udev rule authorizes the Linux user group (that you specify during execution of udev_install script) to access the FlashPro5 hardware without sudo or root permission.
Follow the steps below to configure Linux to detect and use FlashPro5 programmer hardware.

1. At the prompt, type su -.
2. Change directory to the <caeadmin> HOME directory: cd /home/<caeadmin>.
3. Execute the udev_install script with the following option: ./udev_install -t /tmp.
4. The previous command generates a template file called "70-microsemi.rules" in the /tmpdirectory
5. Modify the template file to match your group ID of the user "john" connecting the FlashPro5 hardware (assuming the user "john" will attach the FlashPro5 hardware):
   a. Open a terminal and run the command id as user john. The output should be like the following: text: uid=500(john) gid=500(john)….
   b. Return to your terminal with root access. Open the “70-microsemi.rules” in an ASCII editor.
   c. Replace the "" with your current group ID # in the following two lines:

```
BUS=="usb",SYSFS{idProduct}=="2008",SYSFS{idVendor}=="
1514",MODE="0660",GROUP="",SYMLINK+="FlashPro5"
BUS=="usb",SYSFS{idProduct}=="6001",SYSFS{idVendor}=="
0403",MODE="0660",GROUP="",SYMLINK+="FTDI232"
```

   Assuming the user john has "uid=500" and "gid=500", the line after editing should look like this:

```
BUS=="usb",SYSFS{idProduct}=="2008",SYSFS{idVendor}=="1514",MODE="0660"\
,GROUP="500",SYMLINK+="FlashPro5"
BUS=="usb",SYSFS{idProduct}=="6001",SYSFS{idVendor}=="0403",MODE="0660"\
,GROUP="500",SYMLINK+="FTDI232"
```

6. Move the "70-microsemi.rules" file to the proper location:

```
% mv /tmp/70-microsemi.rules /etc/udev/rules.d/
```

   If when running the "udev_install" script to set up FlashPro on Linux, the script fails with error message:

```
% ./udev_install
/bin/sh^M: bad interpreter: No such file or directory
```

   Problem: The script uses Windows CR/LF line termination instead of Unix/Linux LF only line termination and, as such, is not a valid shell script for Linux.

   Workaround: Users must run dos2unix command on the script "udev_install" to convert Windows CR/ LF line termination to Linux LF only line termination and run the script again. At the Linux shell prompt:

```
% dos2unix udev_install
% ./udev_install
```

   Note: If dos2unix is not available, run the following command to install dos2unix, then run dos2unix:

```
% sudo yum install dos2unix
```