PolarFire FPGA Building a Mi-V Processor Subsystem Application Note

AN4997

Introduction (Ask a Question)

Microchip offers the Mi-V processor IP and software toolchain free of cost to develop RISC-V processor-based designs. RISC-V, a standard open Instruction Set Architecture (ISA) under the governance of the RISC-V foundation, offers numerous benefits, which include enabling the open source community to test and improve cores at a faster pace than closed ISAs.

PolarFire® FPGAs support Mi-V soft processors to run user applications. The objective of the application notes is to build a Mi-V processor subsystem that can execute an application from the designated fabric RAMs initialized from the sNVM/SPI Flash. The application notes also describes how to build a RISC-V application using SoftConsole and run it on a PolarFire Evaluation Board.

Design Requirements (Ask a Question)

The following table lists the application notes requirements for building a Mi-V processor subsystem.

Table 1. Design Requirements

Requirement	Version
Hardware	
Host PC	Windows® 10 and 11
POLARFIRE-EVAL-KIT (MPF300TS-1FCG1152I) – 12V/5A AC power adapter and cord – USB 2.0A to mini-B cable	Rev D or Rev F[1]
Software	
Libero® SoC Design Suite	Refer the `readme.txt` file provided in the design files for all software versions needed to create this reference design.
SoftConsole	See the `readme.txt` file provided in the design files for all software versions needed to create this reference design.
PuTTY (serial terminal emulation program)	—

Important:

1. Rev F Kit has a different on-board DDR part. For more information, see PolarFire Evaluation Kit Quick Start Guide.

Prerequisites (Ask a Question)

Before you start, perform the following steps:

1. Download the design files from www.microchip.com/en-us/application-notes/AN4997
   The design files folder contains the following subfolders:
   - Programming_files: Two programming files (`.job`) one each for Rev D (`top_RevD.job`) and Rev F (`top_RevF.job`) Kit are provided.
   - HW: Contains the Tcl scripts for the design.
   - FW: Contains Softconsole project for this design.

2. Download and install Libero® SoC from Libero SoC Documentation.
3. Download and install SoftConsole from SoftConsole.
4. From the Libero Catalog, download the latest versions of the IP cores from the warning pop-up as shown in the following figure.

Figure 1. Download New Cores Option

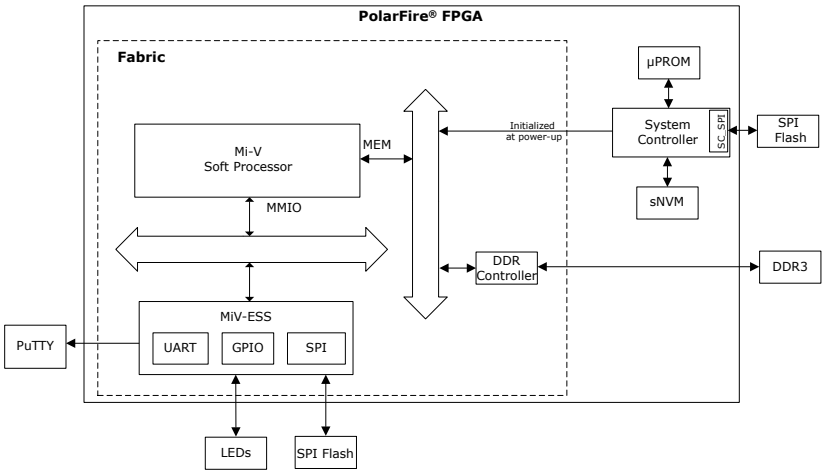
Table of Contents

1. Design Description (Ask a Question)

The application notes describes how to create a Mi-V subsystem for executing user applications. The user application can be stored in µPROM, sNVM, or an external SPI flash. At device power-up, the PolarFire System Controller initializes the designated TCM with the user application and releases the system reset. If the user application is stored in SPI Flash, the System Controller uses the SC_SPI interface for reading the user application from SPI Flash. The given user application prints the UART message “Hello World!” and blinks user LEDs on the board.

The following figure shows the top-level block diagram of the design.

Figure 1-1. Block Diagram



1.1 Fabric RAMs Initialization (Ask a Question)

Each logical RAM instance in the design is initialized from a different source– sNVM, µPROM, or SPI-Flash. The initialization client storage location is configurable. Generate the initialization data to add the initialization clients to the chosen non-volatile memories and program the device. Program SPI-Flash, if chosen as storage location for initialization data. For more information, see Configure Design Initialization Data and Memories.

Important: Libero SmartDesign and configuration screen shots shown in this application notes are for illustration purpose only. Open the Libero project to see the latest updates and IP versions.

2. Creating a Mi-V Processor Subsystem (Ask a Question)

Creating a Mi-V processor subsystem involves:

- Creating a Libero Project
- Creating a New SmartDesign Component
- Instantiating IP Cores in SmartDesign
- Connecting IP Instances in SmartDesign
- Generating SmartDesign Component
- Managing Timing Constraints
- Running the Libero Design Flow

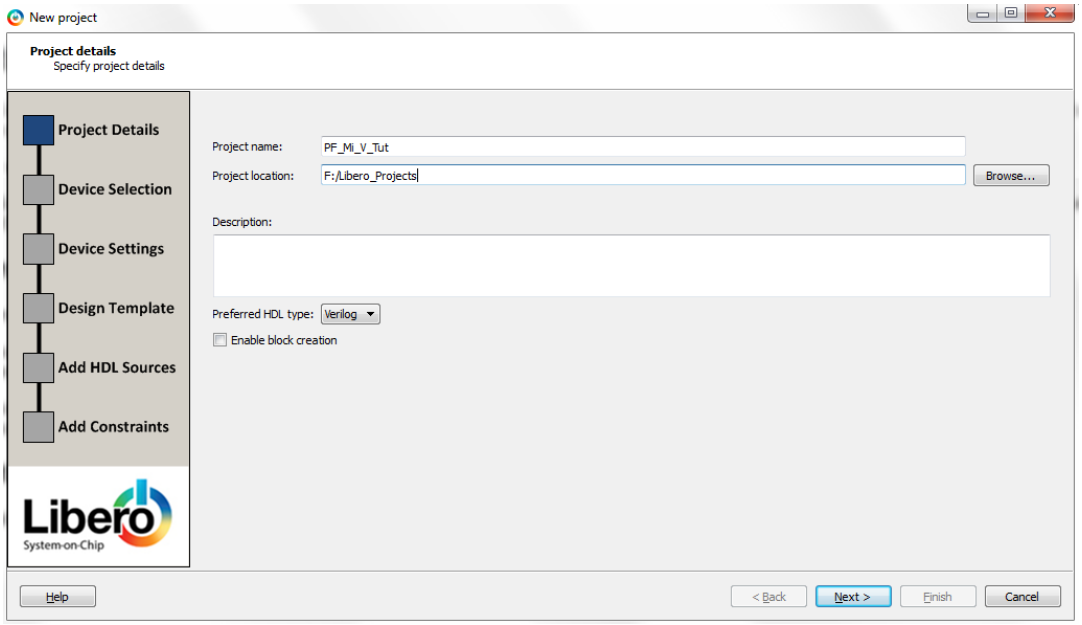
This section describes all of the steps required to create a Mi-V processor subsystem on a new SmartDesign canvas.

2.1 Creating a Libero Project (Ask a Question)

To create a Libero project, follow these steps:

1. On the Libero menu bar, click **Project** > **New Project**
2. Enter the following details, and click **Next**.
   - Project name: PF_Mi_V_Tut
   - Project location: For example, `F:/Libero_Projects`
   - Preferred HDL type: Verilog

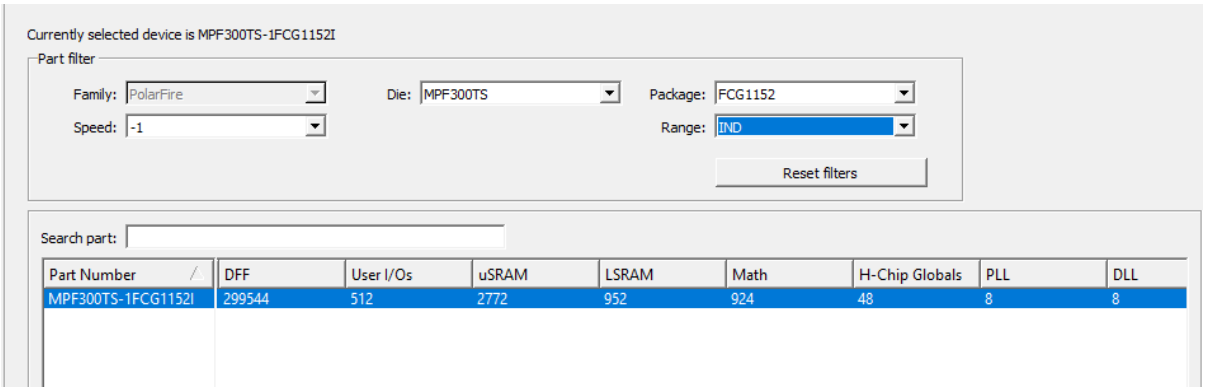
   Figure 2-1. New Project Details



3. To choose the PolarFire® device present on the PolarFire Evaluation Board, select the following settings in the **Device Selection** window, and click **Next**.
   - Family: PolarFire
   - Die: MPF300TS
   - Package: FCG1152

- Speed: -1
- Range: IND
- Part Number: MPF300TS-1FCG1152I

Figure 2-2. Device Selection



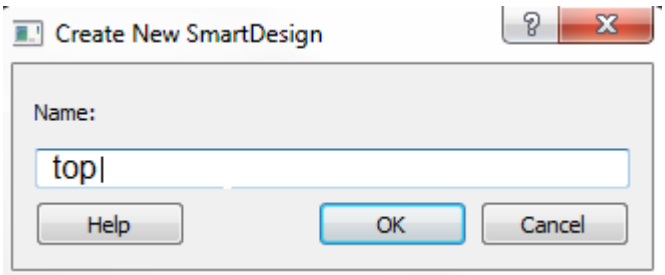
4. In the **Device Settings** window, click **Next** to retain the default core voltage and I/O settings.
5. In the **Add HDL Sources** window, click **Next** to retain the default settings.
6. In the **Add constraints** window, click **Import file** to import the I/O constraint file.
7. In the **Import files** window, locate the `io_constraints.pdc` file in the `DesignFiles_directory\HW\src\constraints` folder, and double-click it.
8. Click **Finish**.
   The Log pane displays a message indicating that the PF_Mi_V_Tut project was created.

2.2 Creating a New SmartDesign Component (Ask a Question)

To create a new SmartDesign component:

1. In Libero, select **File** > **New** > **SmartDesign**.
2. In the **Create New SmartDesign** dialog box, enter **top** as the name of the new SmartDesign project, as shown in the following figure.

Figure 2-3. Create New SmartDesign



3. Click **OK**.
   The **top SmartDesign** component is created.

2.3 Instantiating IP Cores in SmartDesign (Ask a Question)

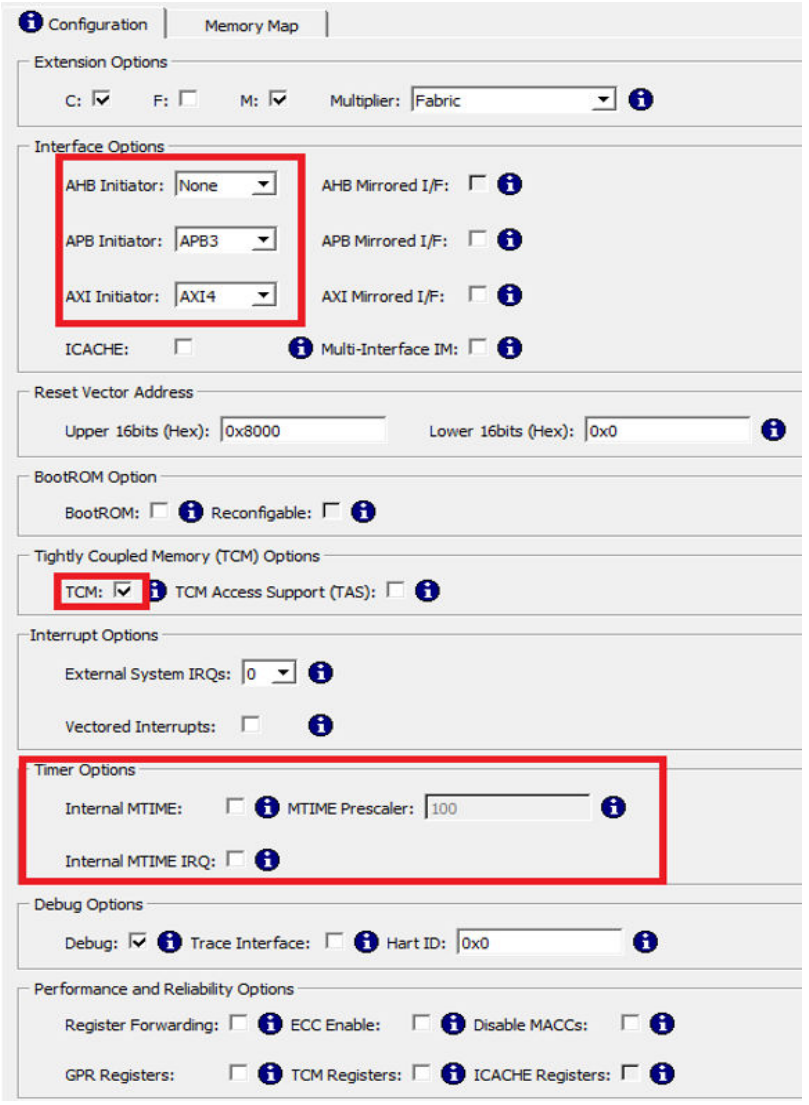
When an IP core is dragged from the Catalog to SmartDesign, Libero prompts you to name the component, and if applicable, to configure the IP core. After the core is configured, Libero generates the component for that core and instantiates it in SmartDesign.

2.3.1 Instantiating Mi-V Processor IP (Ask a Question)

To instantiate the Mi-V Processor IP, perform the following steps:

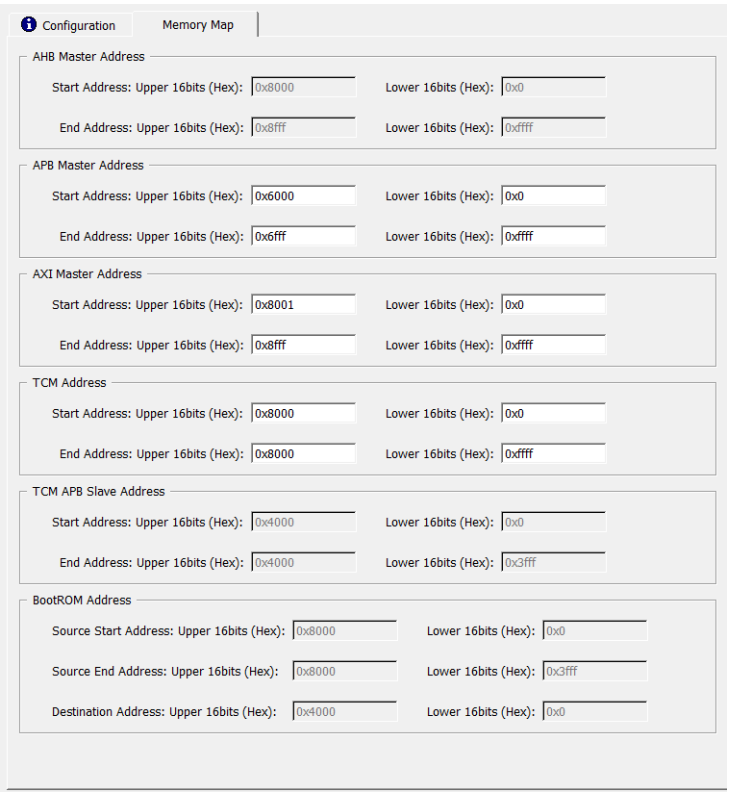
1. From the **Catalog**, drag the **MIV_RV32** to **SmartDesign**.
2. In the **Create Component** dialog box, enter **MiV_RV32_C0** as the component name, and then click **OK**.
3. In the **Configurator** window, under the **Configuration** tab, set the following configuration:
   - Set **Reset Vector Address** > **Upper 16bits (Hex)** to 0x8000 and retain the default setting for **Lower 16bits (Hex)** as shown in the following figure. This is the address the processor will start executing from after a reset.
   - Ensure that the interface options for **AHB initiator** is set to none and AXI4 for **AXI initiator**, as shown in the following figure.
   - Under **Tightly Coupled Memory (TCM) Options**, enable **TCM** option.
   - Disable **Timer Options**.

Figure 2-4. Mi-V Configuration



4. Click the **Memory Map** tab to review the memory map settings of TCM, DDR3, and APB peripherals, as shown in the following figure.

Figure 2-5. TCM and DDR3 Memory Map

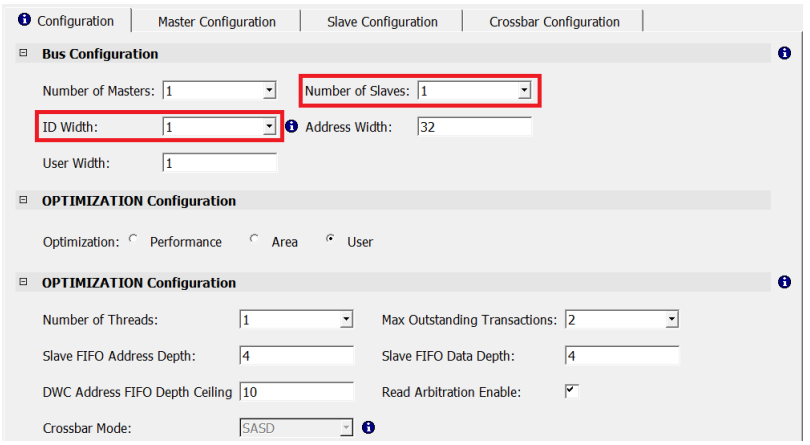


2.3.2 Instantiating AXI Interconnect Bus IP (Ask a Question)

The AXI interconnect bus must be configured to connect the Mi-V core with memory. Also, the AXI4Interconnect is needed for converting the Mi-V processor's AXI4 32-bit data to the DDR3 AXI4 64-bit data, and also for bridging the Mi-V processor's AXI4 clock rate of 83.3 MHz to the DDR3 AXI4 clock rate of 166.66 MHz.

1. From the **Catalog**, drag the **CoreAXI4Interconnect** IP core to SmartDesign.
2. In the **Create Component** dialog box, enter **COREAXI4INTERCONNECT_C0** as the component name, and click **OK**. The **Configurator** window opens.
3. Under the **Configuration** tab, in the **Bus Configuration** section, configure the **COREAXI4INTERCONNECT** IP to have one slave with an ID width of 1, as shown in the following figure. Leave the rest as defaults.

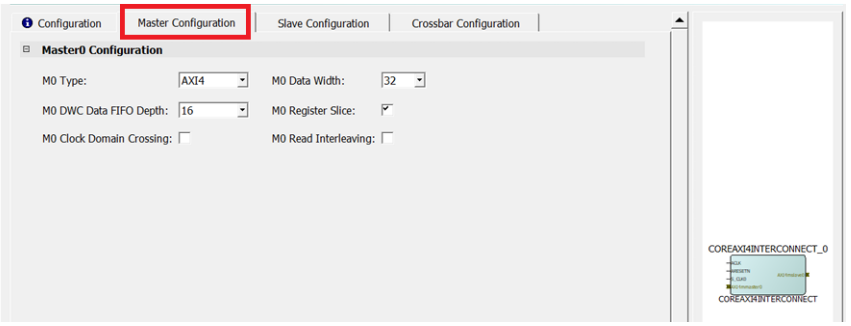
Figure 2-6. CoreAXI4Interconnect Configurator – Bus Configuration Section



4. In the **Master Configuration** tab, retain the following Master0 configuration default settings:
   - M0 Type: AXI4
   - M0 Data Width: 32 bits
   - M0 DWC Data FIFO Depth: 16
   - M0 Register Slice: Selected

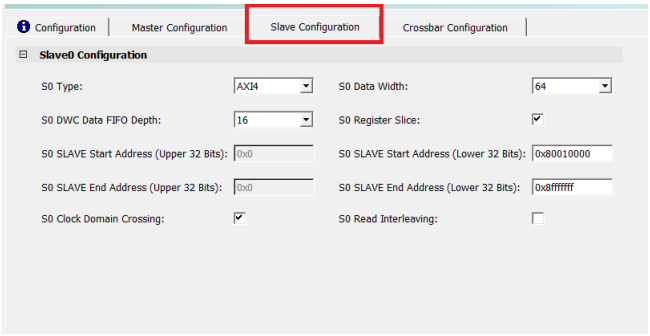
   The following figure shows the Master0 configuration.

Figure 2-7. CoreAXI4Interconnect - Master0 Configuration



5. In the **Slave Configuration** tab, configure the Slave0 port as follows:
   - S0 SLAVE Start Address (Lower 32 bits): 0x80010000
   - S0 SLAVE End Address (Lower 32 bits): 0x8FFFFFFF
   - S0 Clock Domain Crossing: Enabled
   - Leave the rest as defaults

Figure 2-8. CoreAXI4Interconnect Configurator – Slave0 Configuration 6.



6. In the **Crossbar Configuration** tab, ensure that the following options are set:
   - Under **Enable Master Write Access**, **M0 access to S0** is enabled, and under **Enable Master Read Access**, **M0 access to S0** is enabled.
   - Leave the rest as defaults.

Figure 2-9. Crossbar Configuration and Enabling Master Write Access Settings

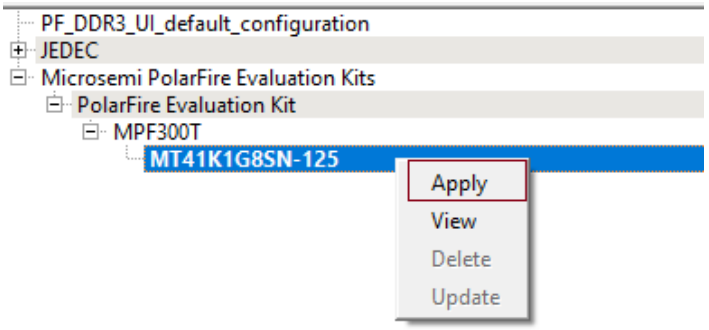
2.3.3 Instantiating DDR3 Memory Controller (Ask a Question)

This application notes demonstrates how to build and debug an application from DDR3 memory. Executing an application from DDR3 memory in the release mode requires a bootloader. The bootloader use case isn't in the scope of this application notes.

If you are using the Rev D Kit, configure the DDR IP as follows. (If you are using Rev F Kit, see Appendix 3 - References)

1. From the Catalog, drag the **PolarFire DDR3** IP core to SmartDesign.
2. In the **Create Component** dialog box, enter **PF_DDR3_C0** as the component name, and then click **OK**.
3. In the left pane of the **Configurator**, expand **Microchip PolarFire Evaluation Kits** > **PolarFire Evaluation Kit** > **MPF300T**.
4. Left-click **MT41K1G8SN-125**, and then click **Apply**, as shown in the following figure.

Figure 2-10. Apply Option for MPF300T



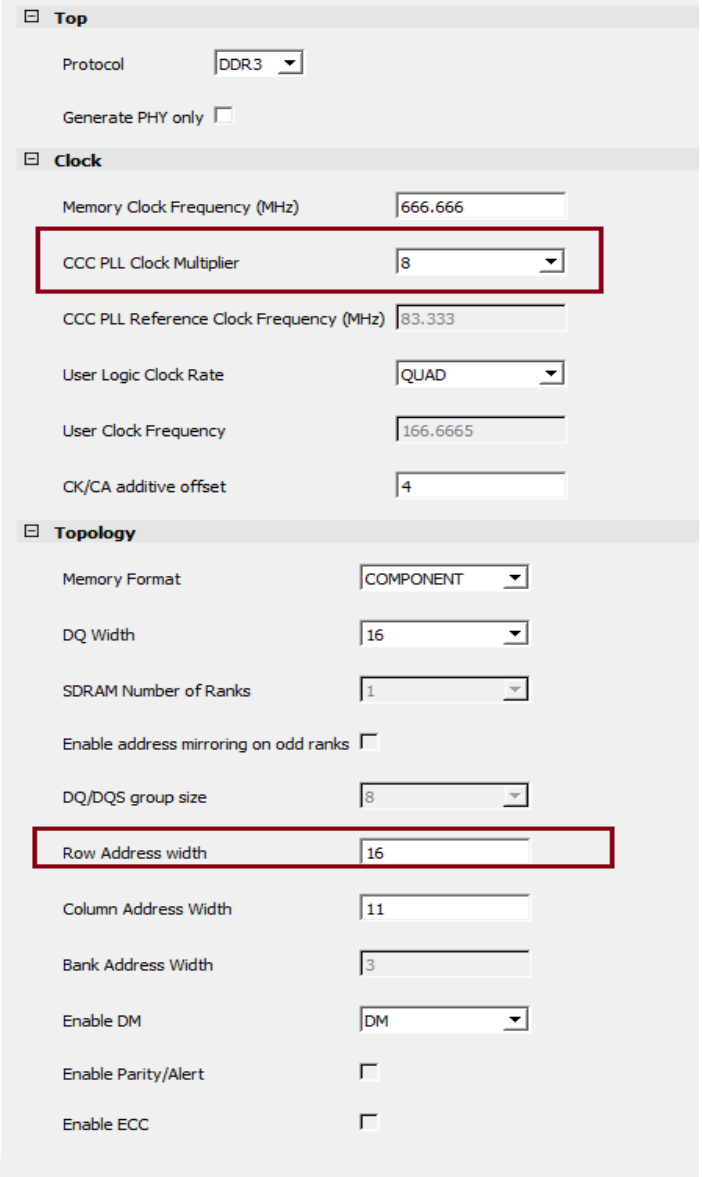
This configures the DDR3 controller with the initialization and timing parameters of the DDR3 memory (MT41K1G8SN-125) present on the PolarFire Evaluation kit.

5. In the right pane of the **Configurator**, click on the **General** tab, set the **CCC PLL Clock Multiplier** to **8**, and the **DQ Width** to **16**, as shown in the following figure.

   The clock multiplier value of 8 sets the CCC PLL reference clock frequency to **83.333** MHz. A reference clock of this frequency is required for the PLL present inside the DDR3 subsystem. The PLL generates a 666.666 MHz DDR3 memory clock frequency and a 166.666 MHz DDR3 AXI clock frequency.

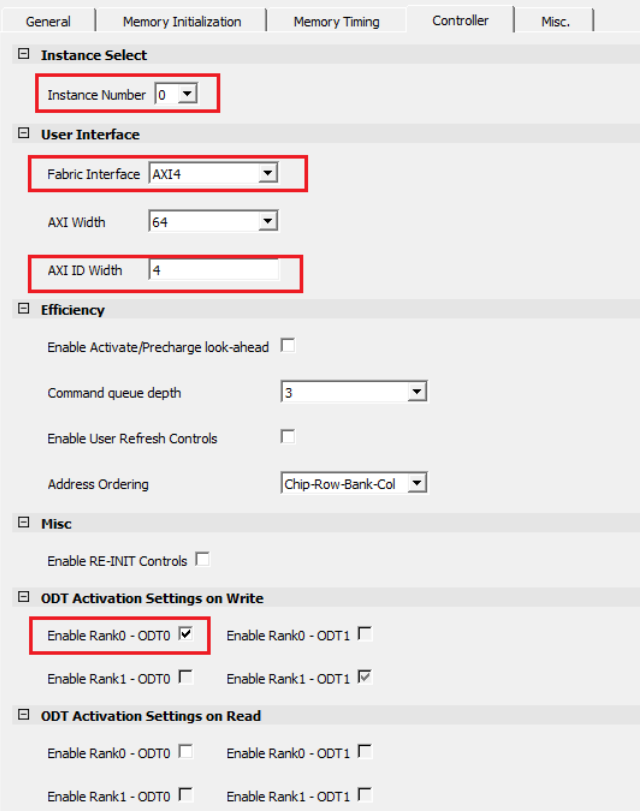
   The **DQ Width** is set to **16** to match the width of the DDR3 memory present on the board.

Figure 2-11. DDR3 General Configuration



6. On the **Controller** tab, ensure that the settings are as follows:
   - Instance Number: 0
   - Fabric Interface: AXI4
   - AXI ID Width: 4
   - Enable Rank0 - ODT0 check box: Selected

Figure 2-12. DDR3 Controller Configuration



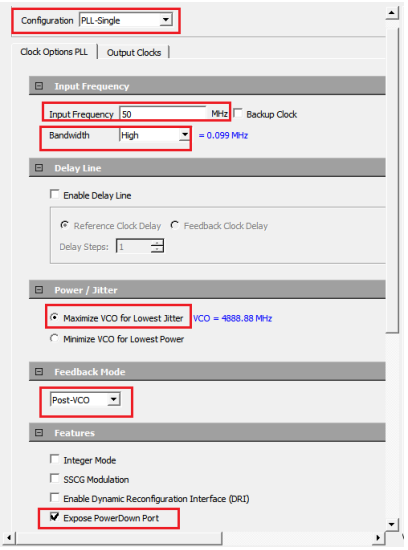
7. Retain the default settings for other tabs, and then click **OK**.

2.3.4 Instantiating PolarFire Clock Conditioning Circuitry (CCC) (Ask a Question)

The PolarFire Clock Conditioning Circuitry (CCC) block generates a 83.333 MHz clock to the processor subsystem, which is used as a reference clock to the PF_DDR3_C0_0 PLL. To instantiate the CCC block, perform the following steps:

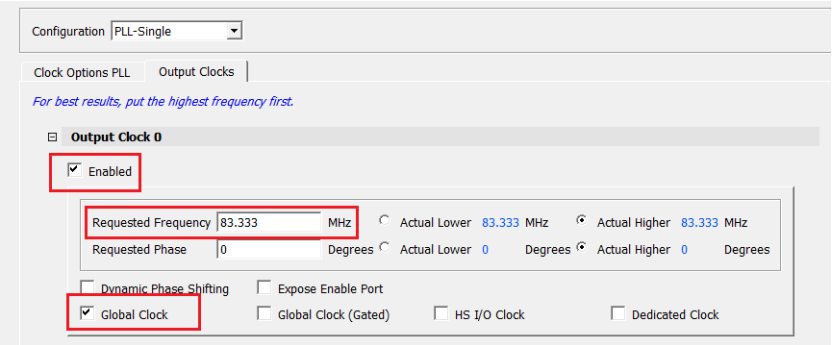
1. From the Catalog, drag the **Clock Conditioning Circuitry (CCC)** core to SmartDesign.
2. In the **Create Component** dialog box, enter **PF_CCC_C0** as the component name, and click **OK**.
3. In the **Configurator** screen, set the configuration to PLL-Single.
4. In the **Clock Options PLL** tab, perform the following.
   - Set the **Input Frequency** to 50 MHz.
   - Under **Power/Jitter**, select Maximize VCO for Lowest Jitter.
   - Set the feedback mode to Post-VCO.
   - Set the Bandwidth to High.

Figure 2-13. CCC Configurator Clock Options PLL Tab



5. In the **Output Clocks** tab, under the **Output Clock 0** section, perform the following.
   - Select the **Enabled** check box to enable PLL output 0.
   - Set the **Requested frequency** to 83.333 MHz.
   - Select the **Global Clock** checkbox.

Figure 2-14. CCC Configurator Output Clocks Tab



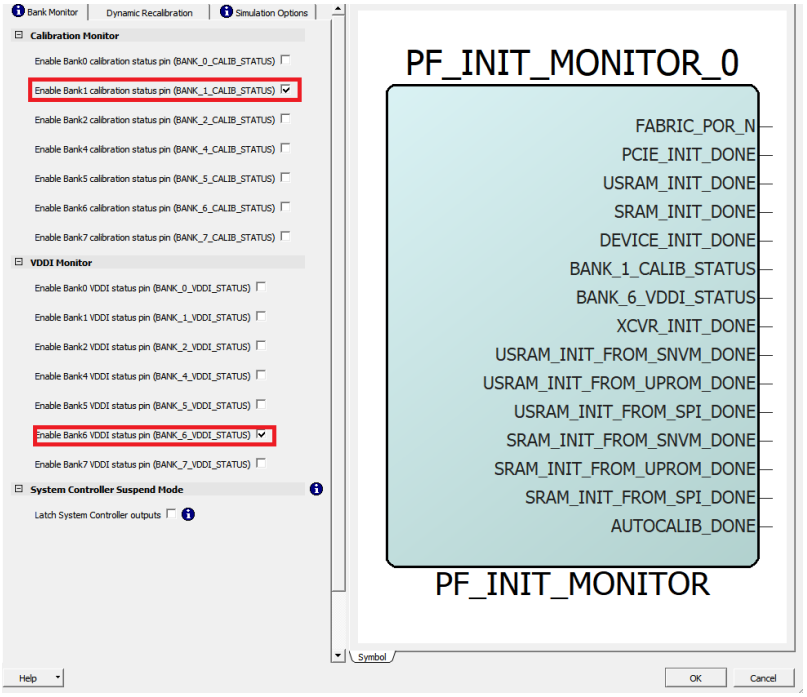
6. Click **OK**.

2.3.5 Instantiating PolarFire Initialization Monitor (Ask a Question)

The PolarFire Initialization Monitor is used to get the status of device initialization including the TCM initialization. To instantiate the PolarFire Initialization Monitor:

1. From the **Catalog**, drag the **PolarFire Initialization Monitor** core to **SmartDesign**.
2. In the **Create Component** dialog box, enter **INIT_Monitor** as the component name, and then click **OK**.
3. In the **INIT_MONITOR Configurator** > **Bank Monitor** tab, clear all the check boxes under **Calibration Monitor** except for **BANK_1_CALIB_STATUS**.
4. In the **INIT_MONITOR Configurator** > **Bank Monitor** tab, clear all the check boxes under **VDDI Monitor** except for **BANK_6_VDDI_STATUS**, and then click **OK**.

Figure 2-15. INIT_MONITOR Configuration



2.3.6 Instantiating CORERESET_PF (Ask a Question)

Two instances of the CORERESET_PF IP are required in this design.

1. From the **Catalog**, drag the **CORERESET_PF** IP.
2. In the **Component Name** dialog box, enter **CORERESET_PF_C0** as the name of this component, and then click **OK**.
3. Retain the default configuration for this IP and click **OK**.
4. Similarly, instantiate another instance with **CORERESET_PF_C1** as its name.

2.3.7 Instantiating CoreJTAGDebug (Ask a Question)

The CoreJTAGDebug IP connects the Mi-V soft processor to the JTAG header for debugging. To instantiate CoreJTAGDebug:

1. From the **Catalog**, drag the **CoreJTAGDebug** IP core to SmartDesign.
2. In the **Create Component** window, enter **COREJTAGDEBUG_C0** as the component name, and click **OK**.
3. In the **Configurator** window, retain the default configuration, and click **OK**.

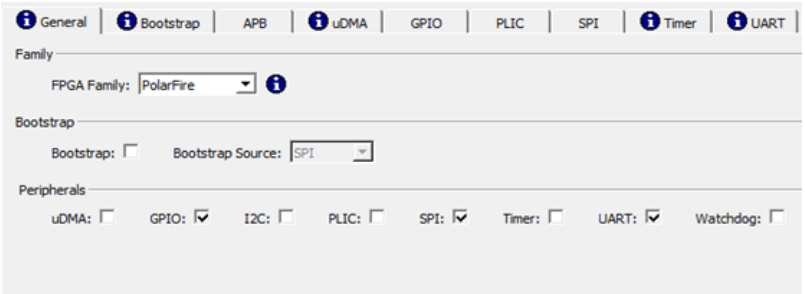
2.3.8 Instantiating MiV ESS Core (Ask a Question)

To instantiate the MiV ESS core, perform the following steps:

1. From the **Catalog**, drag the **MiV_ESS IP** core to **SmartDesign**.
2. In the **Create Component** dialog box, enter **MIV_ESS_C1** as the component name, and then click **OK**.
3. In the **MiV ESS Configurator** screen, perform the following configurations:
   a. Navigate to **General** tab, and make sure that the configurations are same as shown in the following figure.
      i. Deselect **Bootstrap**.

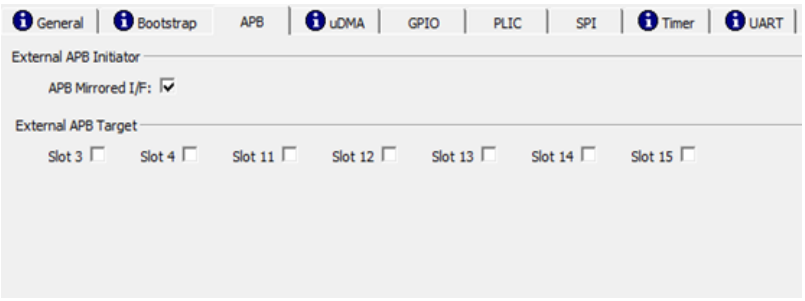
ii. Select the following peripherals: GPIO, SPI, and UART, and deselect all others.

Figure 2-16. General Configurator



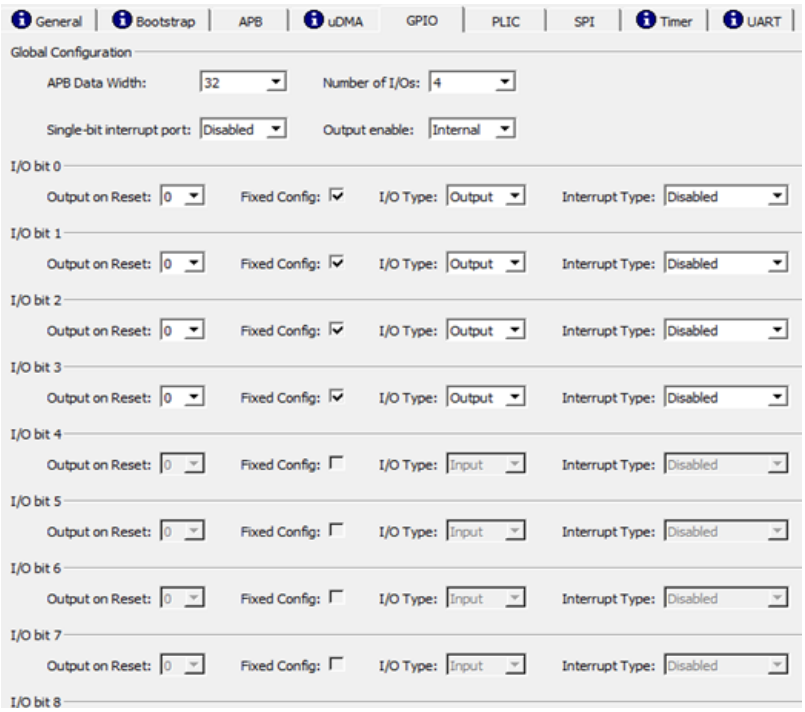
b. Navigate to **APB** tab, and enable **APB Mirrored I/F** as shown in the following figure.

Figure 2-17. APB Configurator



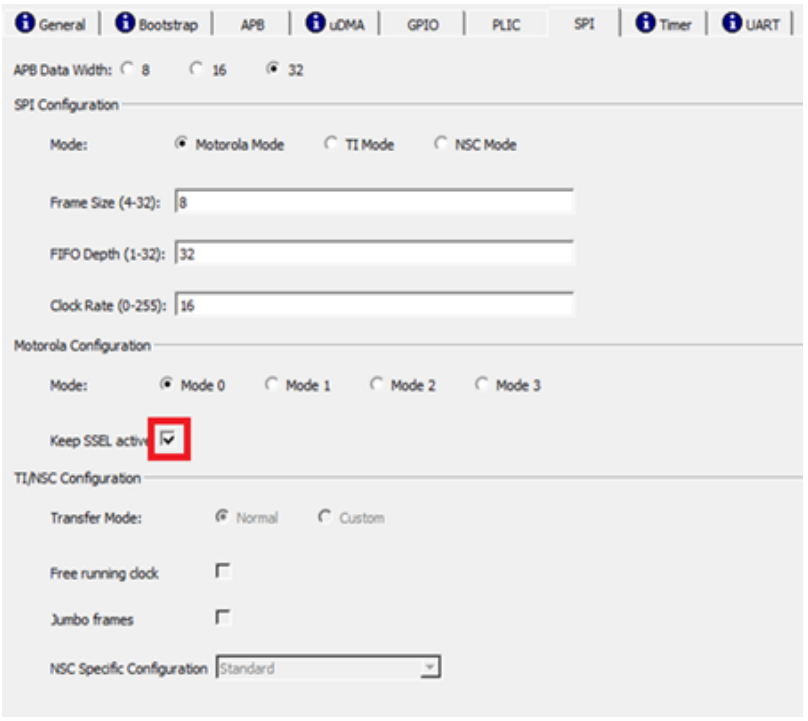
c. Navigate to **GPIO** tab, and make sure that the configurations are same as shown in the following figure.

Figure 2-18. GPIO Configurator



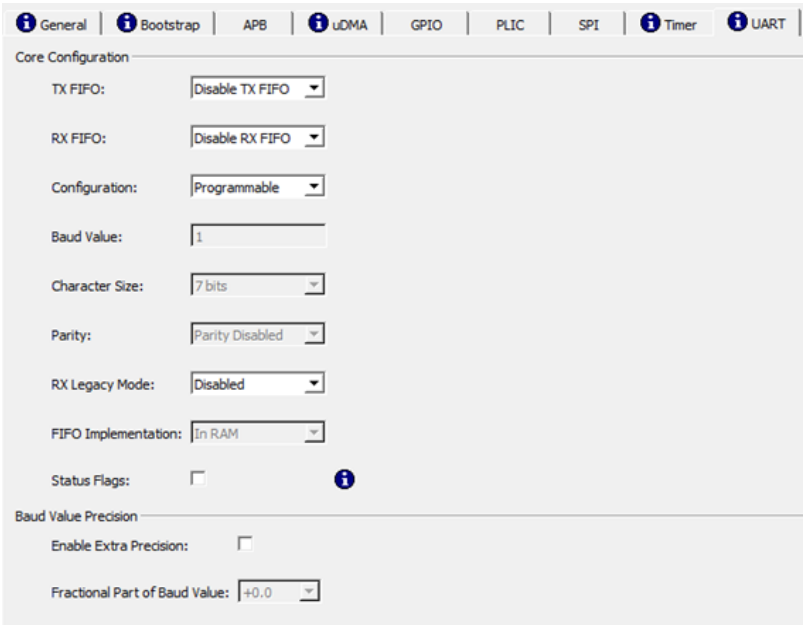
d. Navigate to **SPI** tab, and check the option of **Keep SSEL active**. Make sure that the configurations are same as shown in the following figure.

Figure 2-19. SPI Configurator



e. Navigate to **UART** tab, and make sure that the configurations are same as shown in the following figure.

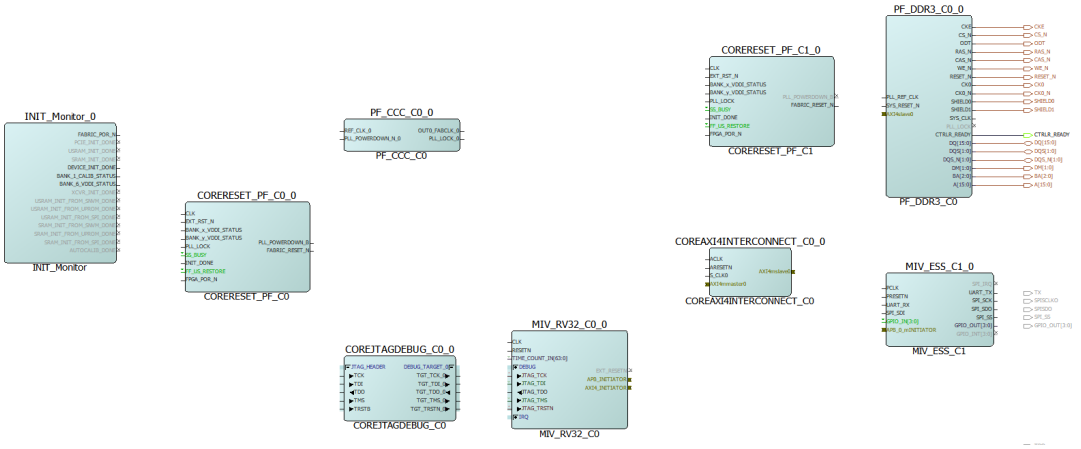
Figure 2-20. UART Configurator



4. To generate the component, click **OK**.

The following figure shows the top SmartDesign after all the components are instantiated.

Figure 2-21. Top SmartDesign with All Components Instantiated



2.4 Connecting IP Instances in SmartDesign (Ask a Question)

Connect the IP blocks in SmartDesign using any of the following methods:

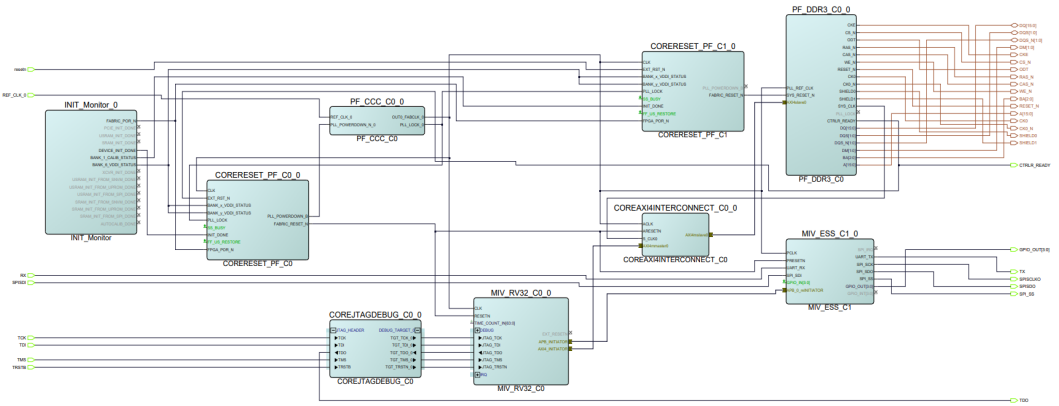
- Using the **Smart Search and Connect** icon: You can initiate the connection mode in SmartDesign by clicking the **Smart Search and Connect** icon in the SmartDesign toolbar, as shown in the following figure. You can search and select multiple ports and connect or disconnect them at once.

  Figure 2-22. Smart Search and Connect Method

- Using the **Connect** option in the Context menu: You can also connect pins by selecting the pins, and then selecting **Connect** from the context menu. To connect multiple pins, hold down the Ctrl key while selecting the pins. Right-click the input source signal, and select **Connect**. To disconnect signals, right-click the input source signal, and select **Disconnect**.
- You can select the source pin, click and drag the wire to the destination pin until you see a '+' sign. The pins will be automatically connected after you release the button of the mouse.
- Right-clicking on a pin provides a list of options like Mark Unused, Edit Slice, Tie Low, Promote to Top-Level, and Tie High. Use these options for individual pins settings.

The following figure shows the Mi-V subsystem in SmartDesign with all IP blocks connected and top-level I/Os.

Figure 2-23. Mi-V Subsystem Connected



Important: Grayed out pins are marked unused, green pins are tied Low, and red pins are tied High. Ensure that unused, tied Low, and tied High pins are strictly set as per preceding figure.

To connect the IP blocks, perform the following steps:

1. Set the pins as follows on **INIT_MONITOR_0**:
   a. Select PCIE_INIT_DONE, USRAM_INIT_DONE, SRAM_INIT_DONE, XCVR_INIT_DONE, USRAM_INIT_FROM_SNVM_DONE, USRAM_INIT_FROM_UPROM_DONE, USRAM_INIT_FROM_SPI_DONE, SRAM_INIT_FROM_SNVM_DONE, SRAM_INIT_FROM_UPROM_DONE, SRAM_INIT_FROM_SPI_DONE, and AUTOCALIB_DONE pins.
   b. Right-click the pins, and select **Mark Unused**.
   c. Connect the FABRIC_POR_N pin to FPGA_POR_N pin of **CORERESET_PF_C0_0** and **CORERESET_PF_C1_0**.
   d. Connect the DEVICE_INIT_DONE pin to CORERESET_PF_C0_0:INIT_DONE.
   e. Connect the BANK_1_CALIB_STATUS pin to CORERESET_PF_C1_0:INIT_DONE.
   f. Connect the BANK_6_VDDI_STATUS pin to CORERESET_PF_C0_0:BANK_x_VDDI_STATUS, CORERESET_PF_C0_0:BANK_y_VDDI_STATUS, CORERESET_PF_C1_0:BANK_x_VDDI_STATUS, and CORERESET_PF_C1_0:BANK_y_VDDI_STATUS.
2. Set the pins as follows on **PF_CCC_C0_0**:
   - Right-click the REF_CLK_0 pin, and select **Promote to Top Level**.
   - Connect the other pins as specified in the following table.

Table 2-1. PF_CCC_C0_0 Pin Connections

Connect From	Connect To
PLL_LOCK_0	CORERESET_PF_C0_0: PLL_LOCK and CORERESET_PF_C1_0: PLL_LOCK
OUT0_FABCLK_0	CORERESET_PF_C0_0: CLK and CORERESET_PF_C1_0:CLK
	MIV_RV32_C0_0: CLK
	PF_DDR3_C0_0: PLL_REF_CLK
	MIV_ESS_C1_0: PCLK
	COREAXI4INTERCONNECT_C0_0: ACLK
PLL_POWERDOWN_N_0	CORERESET_PF_C0_0: PLL_POWERDOWN_B

3. Set the pins of **CORERESET_PF_C0_0** as follows:

- Connect EXT_RST_N pin to PF_DDR3_C0_0:CTRLR_READY.
- Right-click SS_BUSY and FF_US_RESTORE pins and tie them low.

4. Connect the CORERESET_PF_C0_0: FABRIC_RESET_N to the following pins.
   - MIV_RV32_C0_0: RESETN
   - COREAXI4INTERCONNECT_C0_0: ARESETN
   - MIV_ESS_C1_0: PRESETN

> **Important:** As PF_DDR3_C0_0:CTRL_READY pin is connected to CORERESET_PF_C0_0:EXT_RST_N, the Mi-V processor is held in reset until the DDR3 controller is ready. The rest of the system is out of reset as soon as device initialization is done.

5. Set the pins of **CORERESET_PF_C1_0** as follows:
   - Right-click SS_BUSY and FF_US_RESTORE pins and tie them low using the Tie Low option.
   - Select the EXT_RST_N pin and promote it to top level and rename it to resetn.
   - Connect the FABRIC_RESET_N pin to PF_DDR3_C0_0: SYS_RESET_N.
   - Right-click the PLL_POWERDOWN_B pin and mark it unused.
6. Set the pins as follows on **COREJTAGDEBUG_C0_0**:
   - Expand **JTAG HEADER**.
   - Right-click the TDI, TCK, TMS, and TRSTB pins, and select Promote to Top Level.
   - Expand **JTAG HEADER**.
   - Right-click the TDO pin, and select **Promote to Top Level**.
   - Connect the other pins as specified in the following table.

Table 2-2. DEBUG_TARGET Pin Connections

Connect From	Connect to
COREJTAGDEBUG_C0_0:TGT_TCK_0	MIV_RV32_C0_0:JTAG_TCK
COREJTAGDEBUG_C0_0:TGT_TRSTN_0	MIV_RV32_C0_0:JTAG_TRSTN
COREJTAGDEBUG_C0_0:TGT_TMS_0	MIV_RV32_C0_0:JTAG_TMS
COREJTAGDEBUG_C0_0:TGT_TDI_0	MIV_RV32_C0_0:JTAG_TDI
COREJTAGDEBUG_C0_0:TGT_TDO_0	MIV_RV32_C0_0:JTAG_TDO

7. Set the pins as follows on **MIV_RV32_C0_0**:
   - Right-click EXT_RESETN pin, and select Mark Unused.
   - Right-click on TIME_COUNT_IN[63:0] and tie it low.
   - Connect the APB_INITIATOR to MIV_ESS_C1_0: APB_0_mINITIATOR.
   - Connect the AXI4_INITIATOR to COREAXI4INTERCONNECT_C0_0: AXI4mmaster0.
8. Connect the **COREAXI4INTERCONNECT_C0_0** pins as specified in the following table.

Table 2-3. COREAXI4INTERCONNECT_C0_0 Pin Connections

COREAXI4INTERCONNECT_C0_0 Pin Name	Connect To
S_CLK0	PF_DDR3_C0_0:SYS_CLK
AXI4mslave0	PF_DDR3_C0_0:AXI4slave0

9. Set the pins as follows on **PF_DDR3_C0_0**:
   - Right-click the PLL_LOCK output pin, and select **Mark Unused**.

- Right-click the CTRLR_READY pin, and select **Promote to Top Level** for debug purpose. The CTRLR_READY signal is used to monitor the status of the DDR controller.
- Ensure that the other pins are promoted to top level.

10. Set the pins as follows on **MIV_ESS_C1_0**:
    - Select UART_RX and SPI_SDI.
    - Right-click, and select **Promote to Top Level**.
    - Select UART_TX, SPI_SCK, SPI_SDO, SPI_SS, and GPIO_OUT[3:0].
    - Right-click, and select **Promote to Top Level**.
    - Right-click the SPI_IRQ, GPIO_INT[3:0], and select Mark Unused.
    - Select GPIO_IN[3:0] and select Tie low.
    - Right-click on the port "UART_TX", and rename it to "TX".
    - Similarly, rename "UART_RX" to "RX", "SPI_SCK" to "SPISCLKO", "SPI_SDO" to "SPISDO", and "SPI_SDI" to "SPISDI".
11. Right-click the top SmartDesign canvas, and select **Auto Arrange Layout**.
12. Click **File** > **Save top**.

The IP blocks are successfully connected.

2.5 Generating SmartDesign Component (Ask a Question)

To generate the SmartDesign component, perform the following steps:

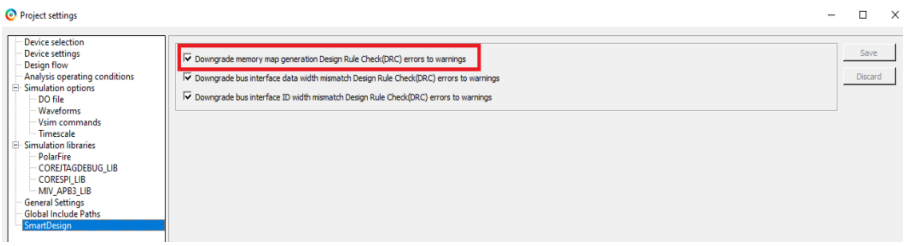
1. In **Design Hierarchy**, right-click top, and select **Set As Root**.
2. Save the project.
3. Click the **Generate Component** icon (shown in the following figure) on the SmartDesign toolbar.

Figure 2-24. Generate Component Icon

When the Mi-V component is generated, the **Message** window displays the message, “The top was generated successfully.”

Important: In case any error is observed related to address space access issue, ensure to perform the following step: Navigate to **Project** > **Project Settings** > **SmartDesign**, and then Enable "Downgrade memory map generation DRC errors to warnings" as shown in the following figure.

Figure 2-25. Project Settings



4. Select the **Build Hierarchy** option and save the project.

2.6 Managing Timing Constraints (Ask a Question)

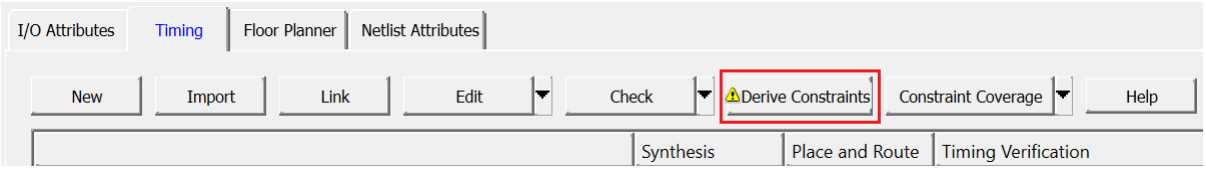
Before running the Libero design flow, you must derive the timing constraints and import the JTAG and asynchronous clocking constraints.

2.6.1 Deriving Constraints (Ask a Question)

To derive constraints, perform the following steps:

1. Double-click **Manage Constraints** on the **Design Flow** tab.
2. In the **Manage Constraints** window, select the **Timing** tab, and click **Derive Constraints**, as shown in the following figure.

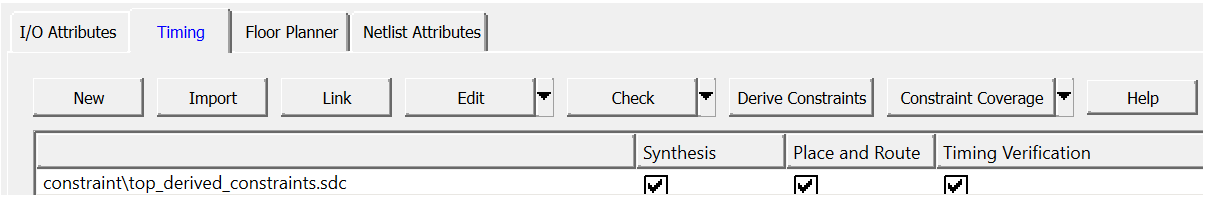
Figure 2-26. Derive Constraints Button



The design hierarchy is built, and the `top_derived_contraints.sdc` file is generated in the project folder.

In the dialog box that appears, click **Yes** to associate the SDC file to the **Synthesis**, **Place and Route**, and **Timing Verification** tools, as shown in the following figure.

Figure 2-27. Derived Constraints



3. Click Save the project.

2.6.2 Importing Other Constraint Files (Ask a Question)

The JTAG clock constraint and the asynchronous clocks constraint must be imported. These constraints (`.sdc`) files are available in the `DesignFiles_directory\HW\src\constraints` folder.

To import and map the constraint files, perform the following steps:

1. On the **Timing** tab, click **Import**.
2. Navigate to the `DesignFiles_directory\HW\src\constraints`, and select the `timing_user_constraints.sdc` file.
3. Select the **Synthesis**, **Place and Route**, and **Timing Verification** checkboxes next to the `timing_user_constraints.sdc` file.
   This constraint file defines that the PF_CCC_C0_0 output clock and PF_DDR3_C0_0 AXI clock are asynchronous clocks.
4. Click **Save**.

2.7 Running the Libero Design Flow (Ask a Question)

This section describes the Libero® design flow, which involves the following steps:

- Synthesis

- Place and Route
- Verify Timing
- Generate FPGA Array Data
- Configure Design Initialization Data and Memories
- Generate Design Initialization Data
- Generate Bitstream
- Run PROGRAM Action
- Generate SPI Flash Image
- Run PROGRAM_SPI_IMAGE Action

After each step is completed, a green tick mark appears next to the step on the **Design Flow** tab.

Important: To initialize the TCM in PolarFire using the system controller, a local parameter **l_cfg_hard_tcm0_en**, in the `miv_rv32_subsys_pkg.v` file must be changed to 1'b1 prior to synthesis. See the TCM section in the *MIV_RV32 Handbook*.

2.7.1 Synthesis (Ask a Question)

To synthesize the design, perform the following steps:

1. Right-click Synthesis, select **Configure Options** and disable the **Enable automatic compile point** checkbox.
2. Double-click **Synthesis** on the **Design Flow** tab. When the synthesis is complete, a green tick mark appears next to Synthesize.
3. Right-click **Synthesize** and select **View Report** to view the synthesis report in the **Reports** tab.

2.7.2 Place and Route (Ask a Question)

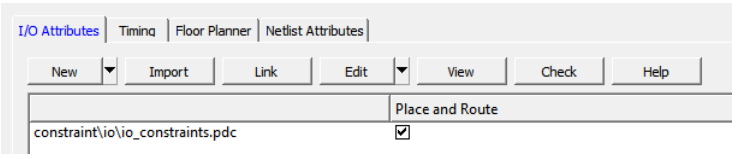
The place and route process requires the following steps to be completed:

- Selecting the already imported `io_constraints.pdc` file
- Placing the PF_DDR3_C0_0 block using the I/O Editor
- Ensuring all the I/Os are locked

To perform place and route, follow these steps:

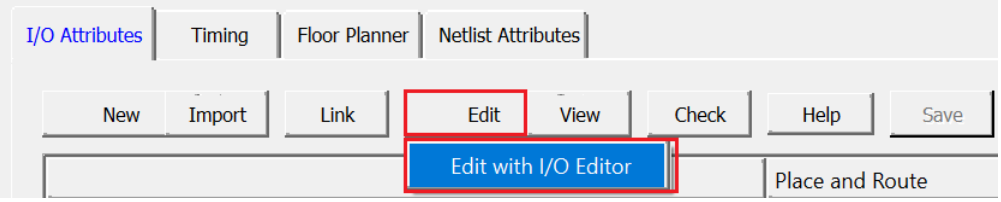
1. Double-click **Manage Constraints** on the **Design Flow** tab.
2. On the **I/O Attributes** tab, enable the check box next to the `io_constraints.pdc` file, as shown in the following figure. The `io_constraints.pdc` file contains the I/O assignment for reference clock, UART, GPIO, and SPI interfaces, and other top-level I/Os.

Figure 2-28. I/O Attributes



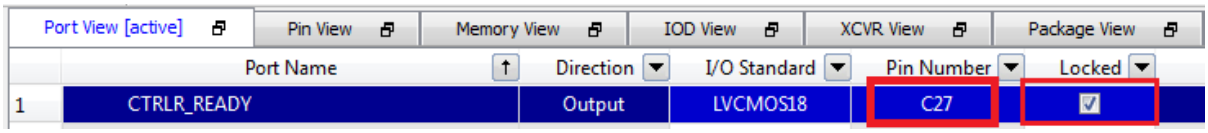
3. From the **Edit** drop-down list, select **Edit with I/O Editor**, as shown in the following figure.

Figure 2-29. Edit with I/O Editor Option



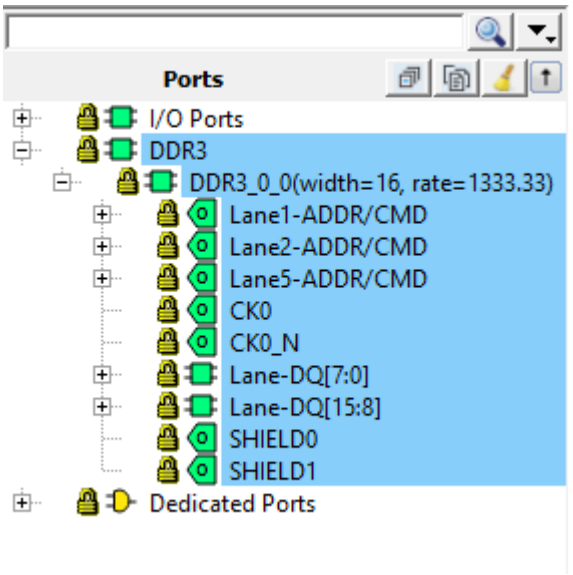
4. In the I/O Editor, click the **Port View [active]** tab, and lock the CTRLR_READY port to pin C27, as shown in the following figure. This ensures that the CTRLR_READY port is assigned to pin C27, which is connected to an user LED for debug purposes.

Figure 2-30. Port View



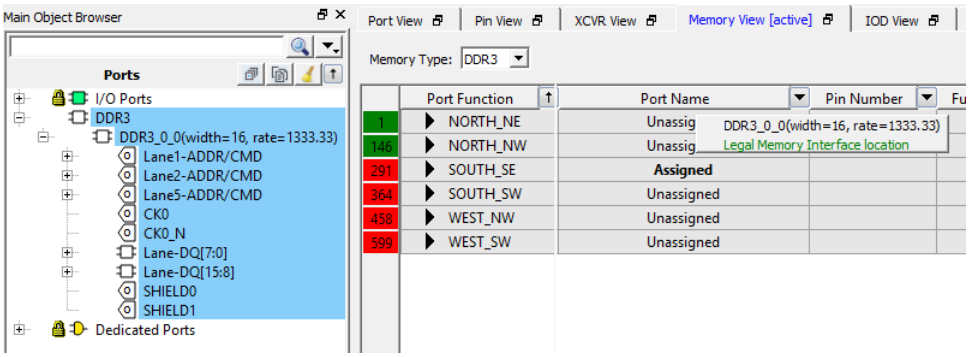
5. To place the DDR3 I/O lanes, in the I/O Editor Design View, click the Port tab in the left pane, and select **DDR3**, as shown in the following figure.

Figure 2-31. I/O Editor Design View – DDR3 Selection



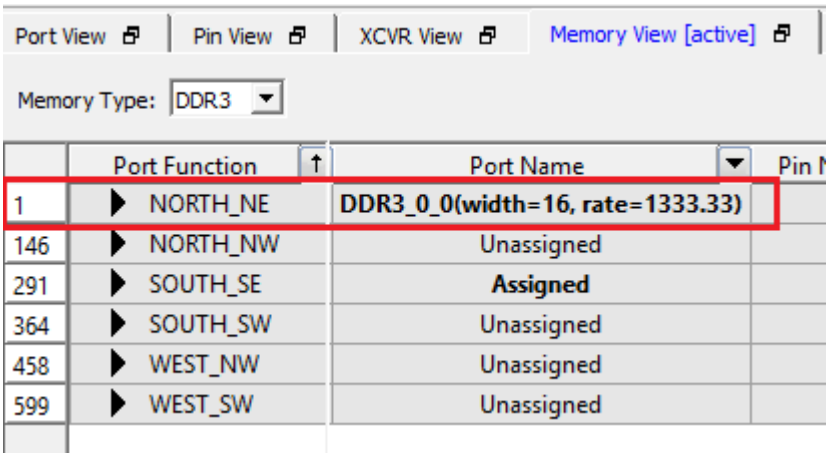
6. Drag and place the DDR3 subsystem on the **NORTH_NE** side, as shown in the following figure. The DDR3 memory on the board is connected to DDR I/Os present on the north-east side.

Figure 2-32. Memory View [active] Tab with DDR3 Subsystem Placement



The DDR3 subsystem is placed on the **NORTH_NE** side, as shown in the following figure.

Figure 2-33. PF_DDR3_C0_0 Placed



7. From **I/O Editor Port View** tab, check if there are any unlocked I/Os, and lock them as mapped in the `io_constraints.pdc` file available in the `Design_Files_Directory\HW\src\constraints` folder.
8. Click **Save**.
9. Close the I/O Editor.
   A `user.pdc` file is created for PF_DDR3_C0_0 block in the **Constraint Manager** > **I/O Attributes** and **Floor Planner** tabs.

> **Important:** PF_DDR3_C0_0 can also be placed using the `fp_constraints.pdc`. Import the `fp_constraints.pdc` from **Constraint Manager** > **Floor Planner** tab and select the place and route option after synthesis. This constraint file is available in the `Design_Files_Directory\HW\src\constraints` folder.
> When place and route is successful, a green tick mark appears next to **Place and Route**.

10. Double-click **Place and Route** from the **Design Flow** tab.

Important: The user has to enable the **High Effort Layout and Repair Minimum Delay Violation** option in the Place and Route settings to meet the timing requirements.

2.7.3 Verify Timing (Ask a Question)

To verify timing reports, perform the following steps:

1. Double-click **Verify Timing** on the **Design Flow** tab.
   When the design successfully meets the timing requirements, a green tick mark appears next to **Verify Timing**.
2. Right-click **Verify Timing** and select **View Report** to view the verify timing report in the **Reports** tab.

2.7.4 Generate FPGA Array Data (Ask a Question)

Double-click **Generate FPGA Array Data** on the **Design Flow** tab.

When the FPGA array data is generated, a green tick mark appears next to **Generate FPGA Array Data**.

2.7.5 Configure Design Initialization Data and Memories (Ask a Question)

The **Configure Design Initialization Data and Memories** step in the Libero design flow is used to configure the TCM initialization data and storage location. User can use µPROM, sNVM, or SPI Flash as storage location based on the size of the initialization data and design requirements. In this application notes, the SPI Flash memory is used to store the TCM initialization data.

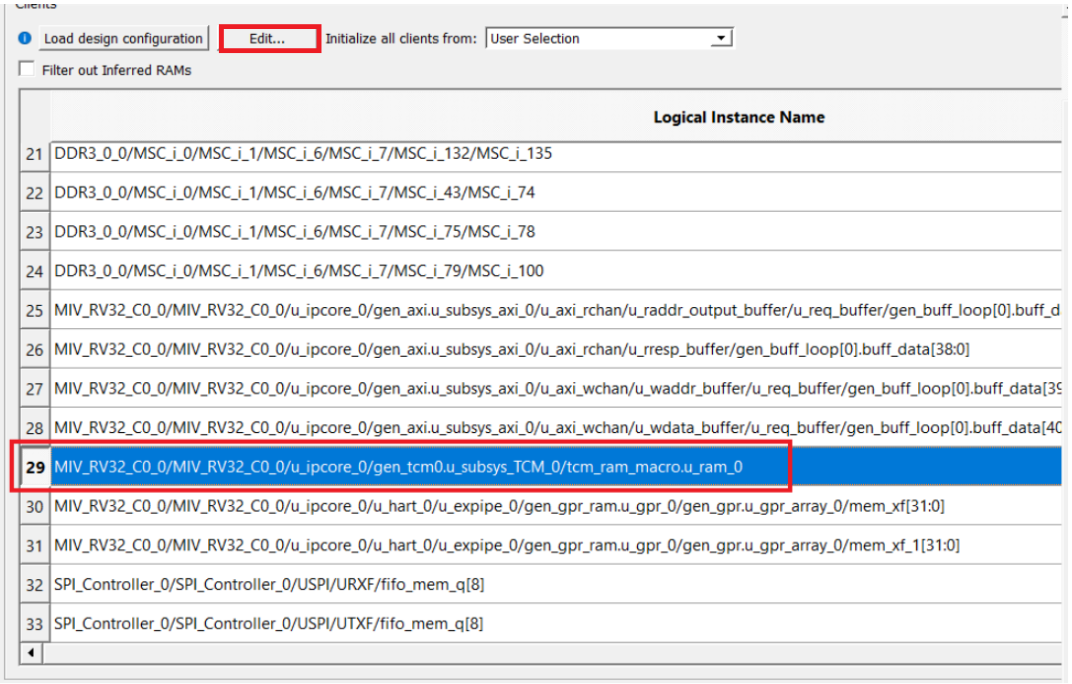
This process requires the user application executable file (HEX file) as input to initialize the TCM blocks after device power-up. The HEX file is provided with the design files. For more information about building the user application, see Building the User Application Using SoftConsole.

Important: The HEX file available in the `DesignFiles_Directory\HW\src\softconsole\MiV_uart_blinky.hex` folder is already modified to be compatible.

To generate an TCM initialization client and add it to an external SPI flash device:

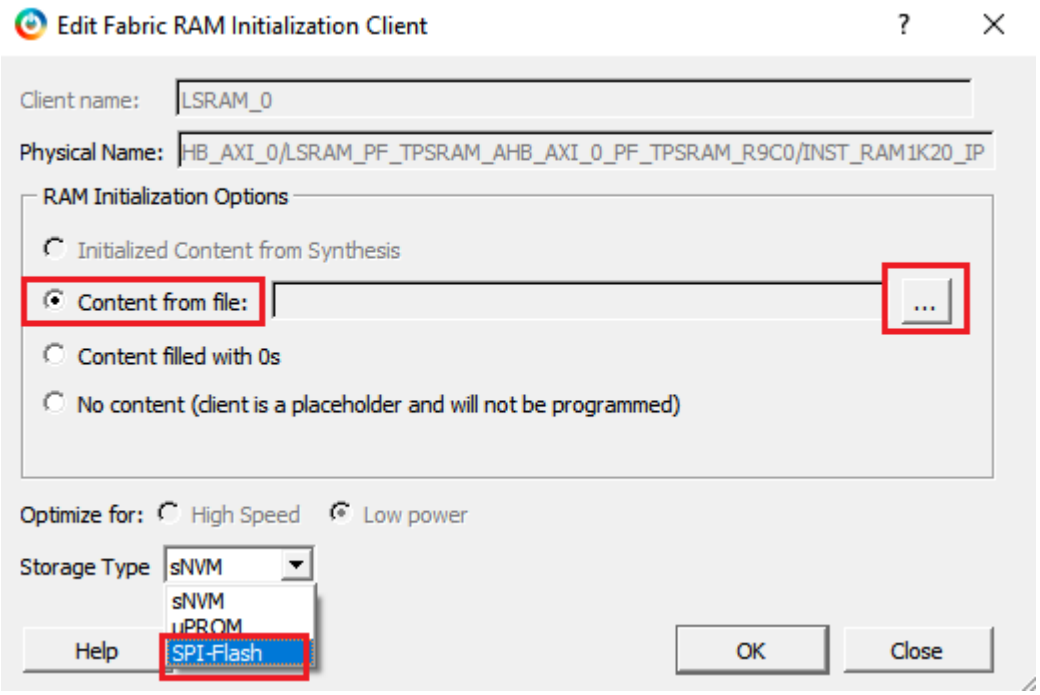
1. Double-click **Configure Design Initialization Data and Memories** on the **Design Flow** tab.
2. On the Fabric RAMs tab, select `top/MIV_RV32_C0_0/MIV_RV32_C0_0/u_ipcore_0/gen_tcm0.u_subsys_TCM_0/tcm_ram_macro.u_ram_0` from the list of logical instances, and click **Edit**, as shown in the following figure. The `top/MIV_RV32_C0_0/MIV_RV32_C0_0/u_ipcore_0/gen_tcm0.u_subsys_TCM_0/tcm_ram_macro.u_ram_0` instance is the MIV_RV32 processor's main memory. The System Controller initializes this instance with the imported client at power-up.

Figure 2-34. Fabric RAMs Tab



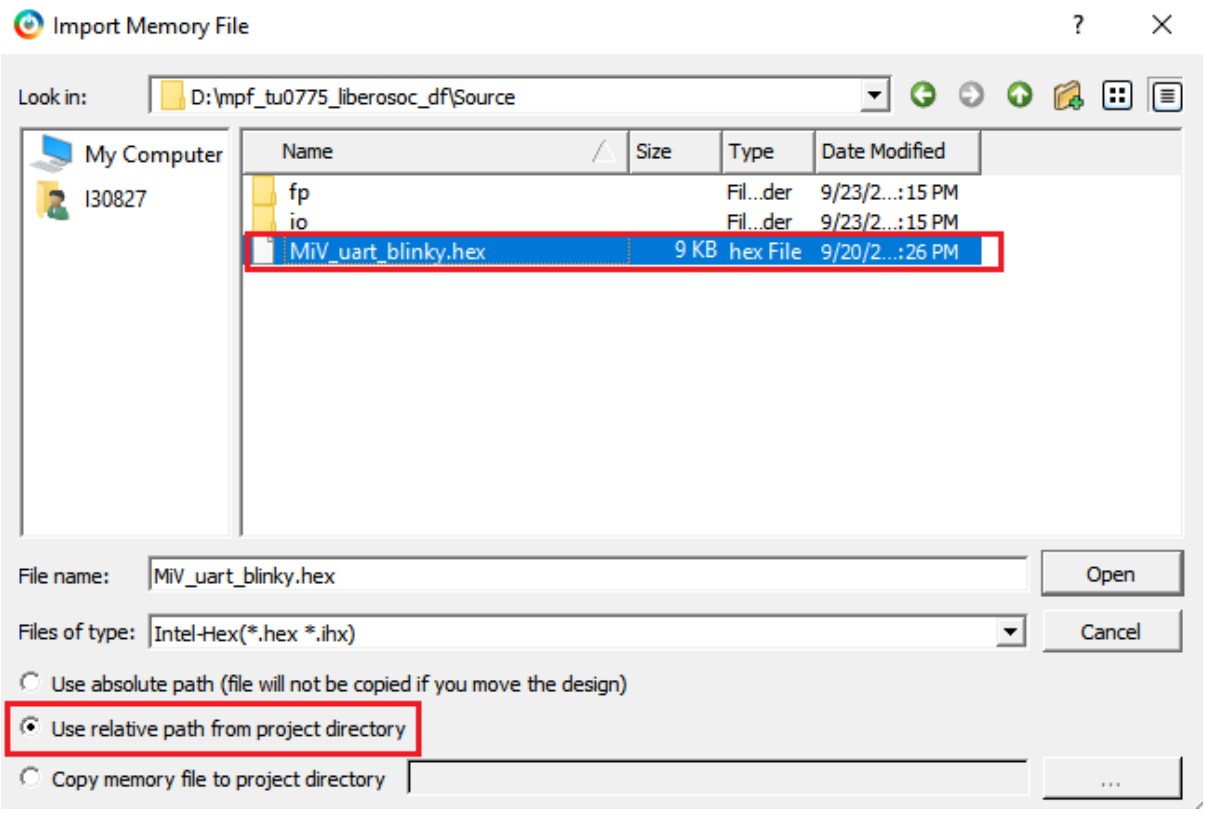
3. In the **Edit Fabric RAM Initialization Client** dialog box, set **Storage Type** to **SPI-Flash** and click the **Import** button next to **Content from file**, as shown in the following figure.

Figure 2-35. Edit Fabric RAM Initialization Client Dialog Box



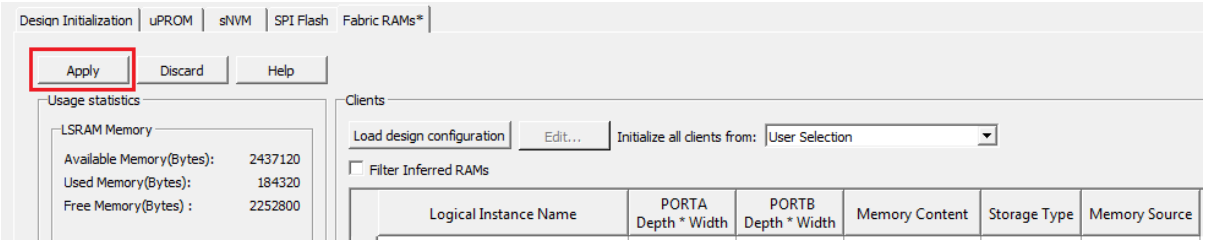
4. In the **Import Memory File** dialog box, locate the `MiV_uart_blinky.hex` file from `DesignFiles_directory\HW\src\softconsole` folder. Select the **Use relative path from project directory** option and click **Open**.

Figure 2-36. Import Memory File Dialog Box



5. In the **Edit Fabric RAM Initialization Client** window, click **OK**.
6. On the **Fabric RAMs** tab, click **Apply**, as shown in the following figure.

Figure 2-37. Fabric RAMs Tab - Apply Button

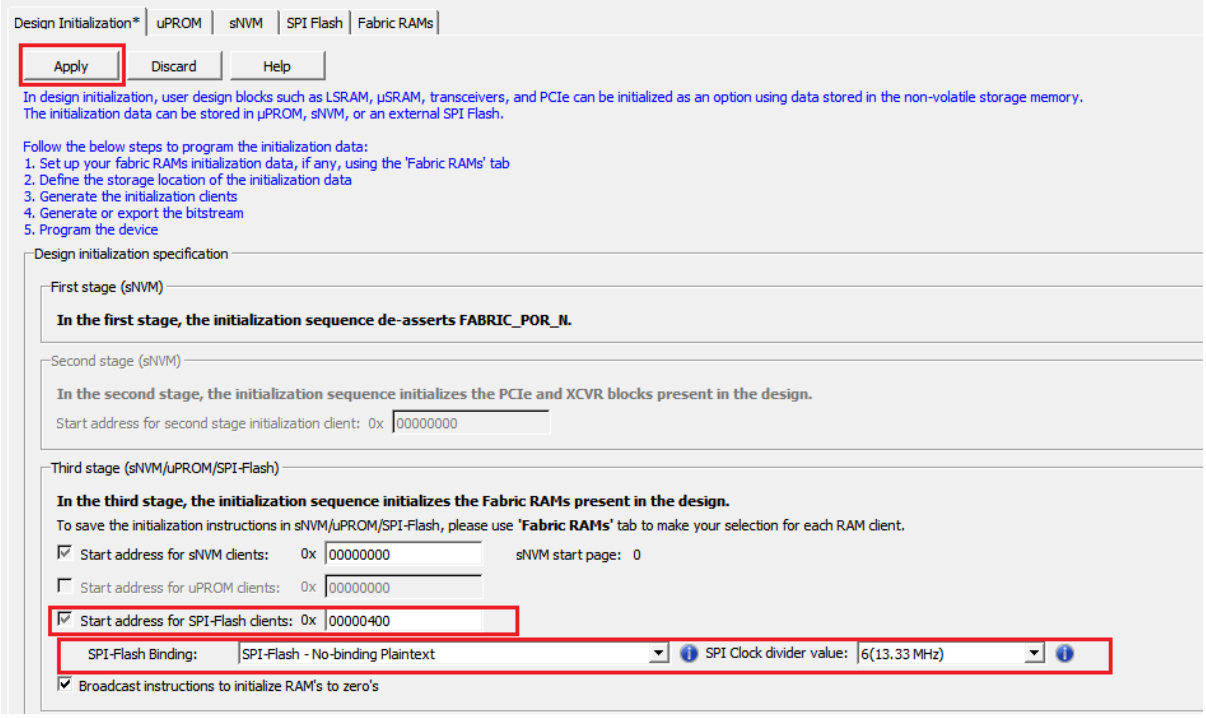


7. In the **Design Initialization** tab, under **Third stage (uPROM/sNVM/SPI-Flash)**, select the **SPI-Flash - No-binding Plaintext** option and ensure that the **SPI Clock divider value** is set to 6, as shown in the following figure. This means that the imported user application will be written to SPI-Flash without encryption and authentication.

> **Important:** The SPI Clock divider value specifies the required SPI SCK frequency to read the initialization data from SPI Flash. The SPI Clock divider value must be selected based on the external SPI Flash operating frequency range.

8. Click **Apply**.

Figure 2-38. Design Initialization Data



This concludes the configuring of the storage type and application file for the fabric RAMs initialization.

2.7.6 Generate Design Initialization Data (Ask a Question)

1. Double-click **Generate Design Initialization Data** on the **Design Flow** tab.
   When the design initialization data is generated successfully, a green tick mark appears next to Generate Design Initialization Data in the Libero Design flow, and the following messages appear in the Log window:

```
Info: 'Generate design initialization data' has completed successfully.
```

```
Info: Stage 1 initialization client has been added to sNVM.
```

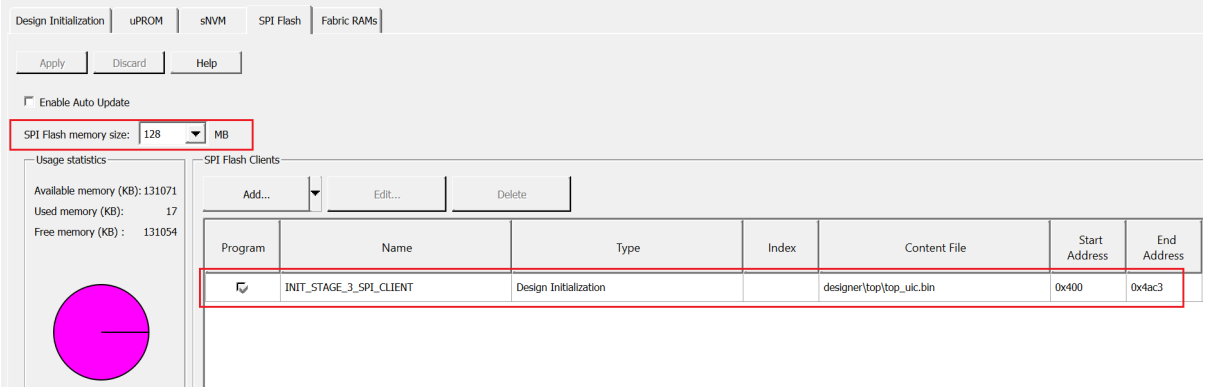
```
Info: Stage 2_3 initialization client has been added to sNVM
```

```
Info: Stage 3 initialization client has been added to SPI.
```

2. Click the **SPI Flash** tab to verify that the bin file has been added, as shown in the following figure.

Important: In order to streamline the SPI-Flash Programming support with FlashPro6, effective from Libero SoC v12.4, the vendor information is replaced with the density of the target memory.

Figure 2-39. SPI Flash Tab



Important: For more information about design initialization, see PolarFire Family Power-Up and Resets User Guide.

2.7.7 Generate Bitstream (Ask a Question)

To generate the programming bitstream:

Double-click **Generate Bitstream** on the **Design Flow** tab. When the bitstream is generated, a green tick mark appears next to Generate Bitstream.

2.7.8 Run PROGRAM Action (Ask a Question)

After generating the bitstream, the PolarFire Evaluation Board must be set up so the device is ready to be programmed. Also, the serial terminal emulation program (PuTTY) must be set up to view the output of the user application. This step involves the following:

Board Setup

Serial Terminal Emulation Program (PuTTY) Setup

Programming the PolarFire Device

2.7.8.1 Board Setup (Ask a Question)

To set up the board, perform the following steps:

1. Ensure that the jumper settings on the board are as listed in the following table.

Table 2-4. Jumper Settings

Jumper	Description
J18, J19, J20, J21, J22	Short pins 2 and 3 for programming the PolarFire FPGA through FTDI.
J28	Short pins 1 and 2 for programming through the on-board FlashPro5.
J26	Short pins 1 and 2 for programming through the FTDI SPI.
J27	Short pins 1 and 2 for programming through the FTDI SPI.
J23	Open pins 1 and 2 for programming SPI flash.
J4	Short pins 1 and 2 for manual power switching using SW3
J12	Short pins 3 and 4 for 2.5V.

Important: For more information about the Jumper locations on the board, see the silkscreen provided in UG0747: PolarFire FPGA Evaluation Kit User Guide.

2. Connect the power supply cable to the J9 connector on the board.
3. Connect the host PC to the J5 (USB) port on the PolarFire Evaluation Board using the USB cable.
4. Power on the board using the SW3 slide switch.

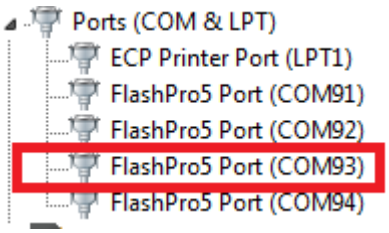
2.7.8.2 Serial Terminal Emulation Program (PuTTY) Setup (Ask a Question)

The user application (`MiV_uart_blinky.hex` file) prints the string "Hello World!" on the serial terminal through the UART interface.

To set up the serial terminal, perform the following steps :

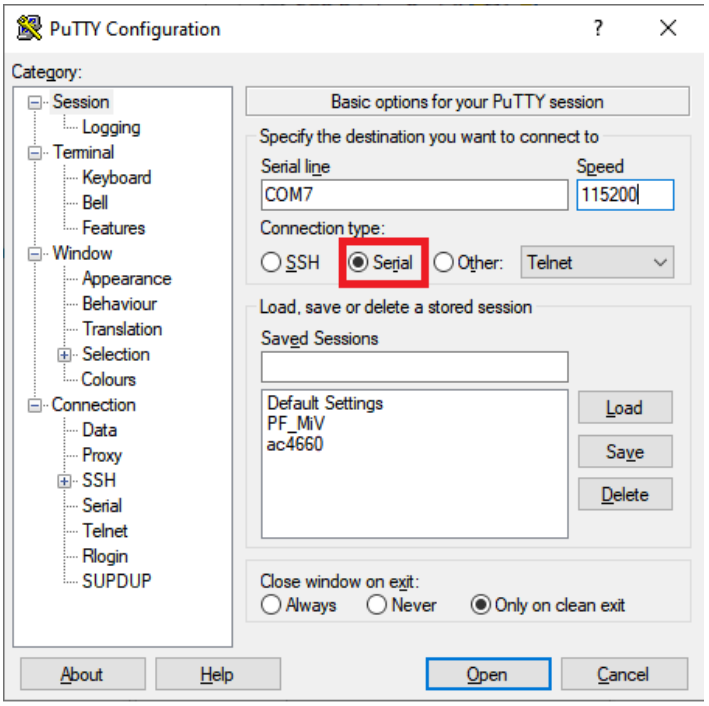
1. Start the PuTTY program.
2. Start Device Manager, note the second-highest COM port number, and use that in the PuTTY configuration. For example, in the list of ports shown in the following figure, COM93 is the port with the second highest number assigned to it.

Figure 2-40. COM Port Number



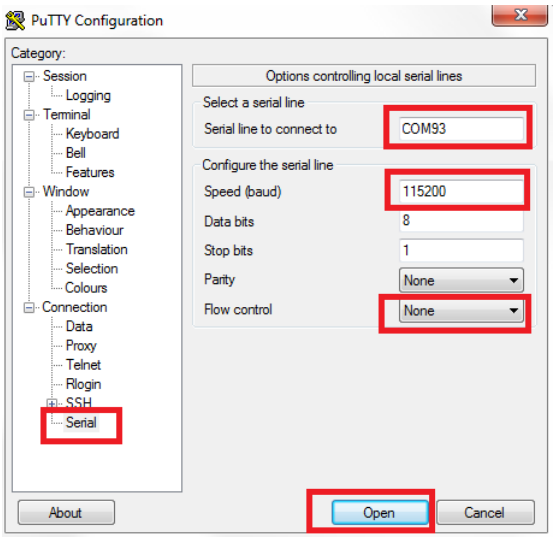
3. Select **Serial** as the **Connection type**, as shown in the following figure.

Figure 2-41. Connection Type Selection



4. Set the serial line to connect to the COM port number as noted in Step 3.
5. Set the **Speed (baud)** to **115200** and **Flow Control** to **None**, as shown in the following figure.

Figure 2-42. PuTTY Configuration



6. Click **Open**.

PuTTY opens successfully, and the serial terminal emulation program is set up.

2.7.8.3 Programming the PolarFire Device (Ask a Question)

To program the PolarFire device:

Double-click **Run PROGRAM Action** on the **Design Flow** tab. When the device is programmed, a green tick mark appears next to **Run PROGRAM action**.

2.7.9 Generate SPI Flash Image (Ask a Question)

To generate the SPI flash image, perform the following step:

Double-click **Generate SPI Flash Image** on the **Design Flow** tab. When the SPI file image is successfully generated, a green tick mark appears next to **Generate SPI Flash Image**.

2.7.10 Run PROGRAM_SPI_IMAGE Action (Ask a Question)

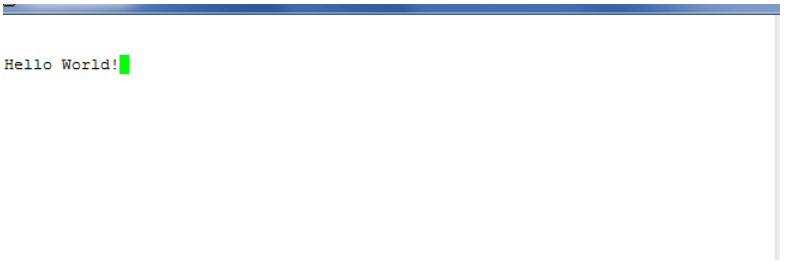
To program the SPI image:

1. Double-click **Run PROGRAM_SPI_IMAGE** on the **Design Flow** tab. In the dialog box that appears, click **Yes**.
2. When the SPI image is successfully programmed on to the device, a green tick mark appears next to **Run PROGRAM_SPI_IMAGE**.

After SPI flash programming is completed, the device needs to be reset to execute the application. The following sequence of operations occurs after device reset or power-cycling the board:

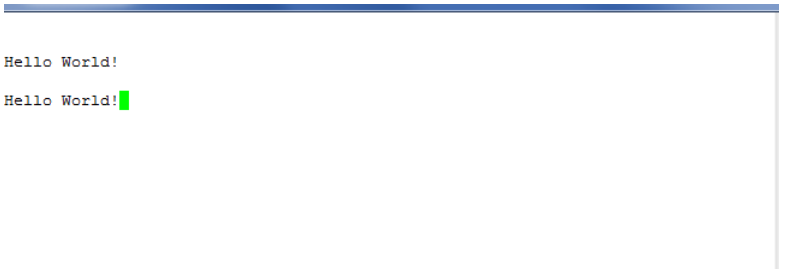
1. The PolarFire System Controller initializes the TCM with the user application code from the external SPI flash and releases the system reset.
2. The Mi-V processor exits reset after DDR3 controller is ready and executes the user application from the TCM. As a result, LEDs 4, 5, 6, and 7 blink, and the string **Hello World!** is printed on the serial terminal, as shown in the following figure.

Figure 2-43. Hello World String



3. When the board is power cycled, the device performs the same sequence of operations. As a result, LEDs 4, 5, 6, and 7 blink, and **Hello World!** is printed again on the serial terminal, as shown in the following figure.

Figure 2-44. Hello World String After the Board is Power Cycled



3. Building the User Application Using SoftConsole (Ask a Question)

This section describes how to build a RISC-V user application executable (`.hex`) file and debug it using SoftConsole.

Building the user application involves the following steps:

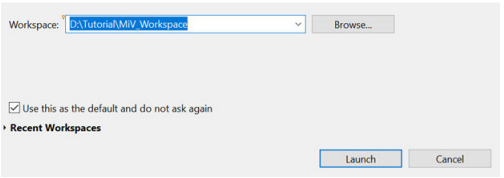
- Creating a Mi-V SoftConsole Project
- Downloading the Firmware Drivers
- Importing the Firmware Drivers
- Creating the main.c File
- Mapping Firmware Drivers and the Linker Script
- Mapping Memory and Peripheral Addresses
- Setting the UART Baud Rate
- Building the Mi-V Project

3.1 Creating a Mi-V SoftConsole Project (Ask a Question)

To create a Mi-V SoftConsole project, perform the following steps:

1. Create a SoftConsole workspace folder on the host PC for storing SoftConsole projects. For example, `D:\Tutorial\MiV_Workspace`.
2. Start SoftConsole.
3. In the **Workspace Launcher** dialog box, paste `D:\Tutorial\MiV_Workspace` as the workspace location, and click **Launch**, as shown in the following figure.

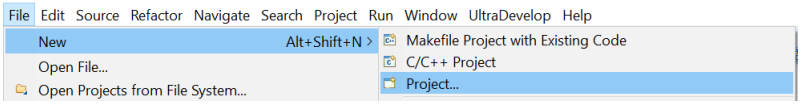
Figure 3-1. Workspace Launcher



When the workspace is successfully created, the SoftConsole main window opens.

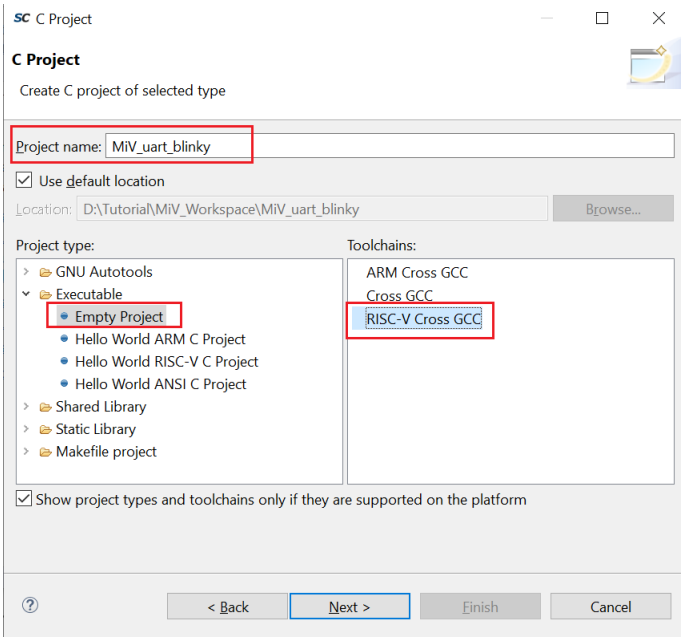
4. Select **File** > **New** > **Project**, as shown in the following figure.

Figure 3-2. New C Project Creation



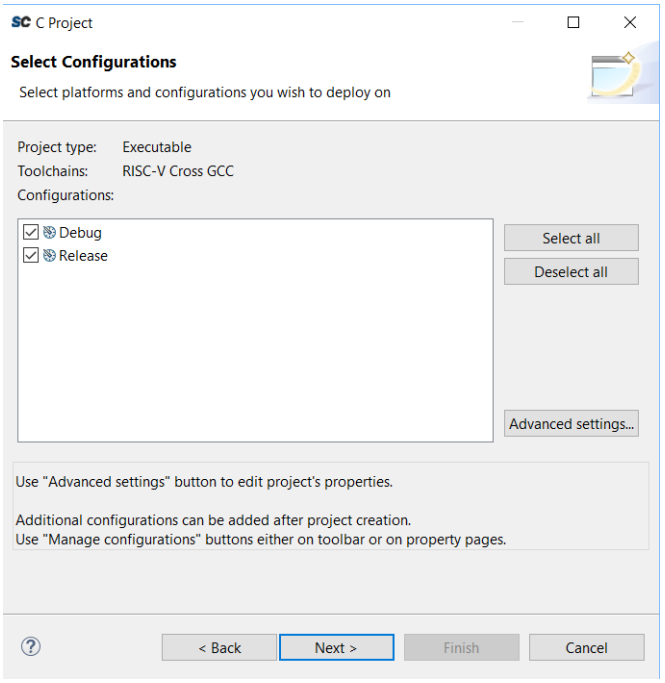
5. Expand **C/C++** and select **C Project** in the **New Project** dialog box.
6. Click **Next**.
7. In the **C Project** dialog box, perform the following steps:
   - Enter a name for the project in the **Project name** field. For example, MiV_uart_blinky.
   - In the **Project type** pane, expand **Executable**, and select **Empty Project** and the **Toolchains** as **RISC-V Cross GCC**, as shown in the following figure. Then, click **Next**.

Figure 3-3. C Project Dialog Box



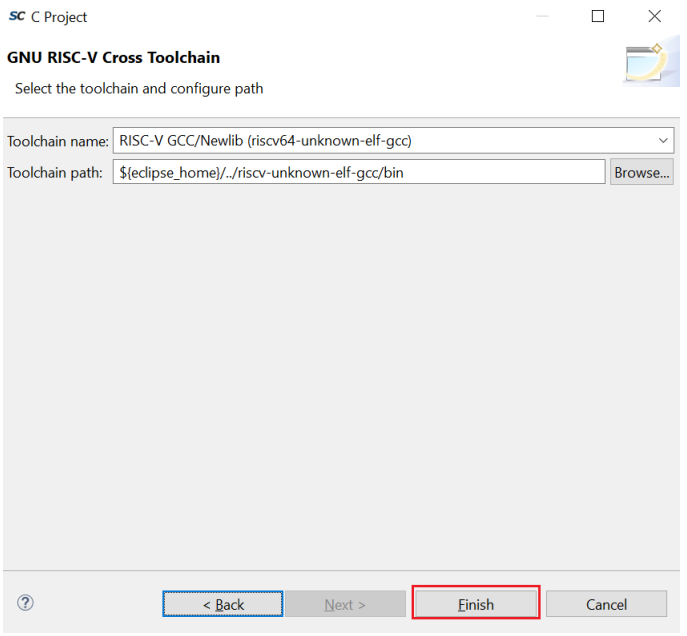
8. Select the platforms and configurations you want to deploy in the **Select Configurations** dialog box and click **Next**, as shown in the following figure.

Figure 3-4. Select Configurations Dialog Box



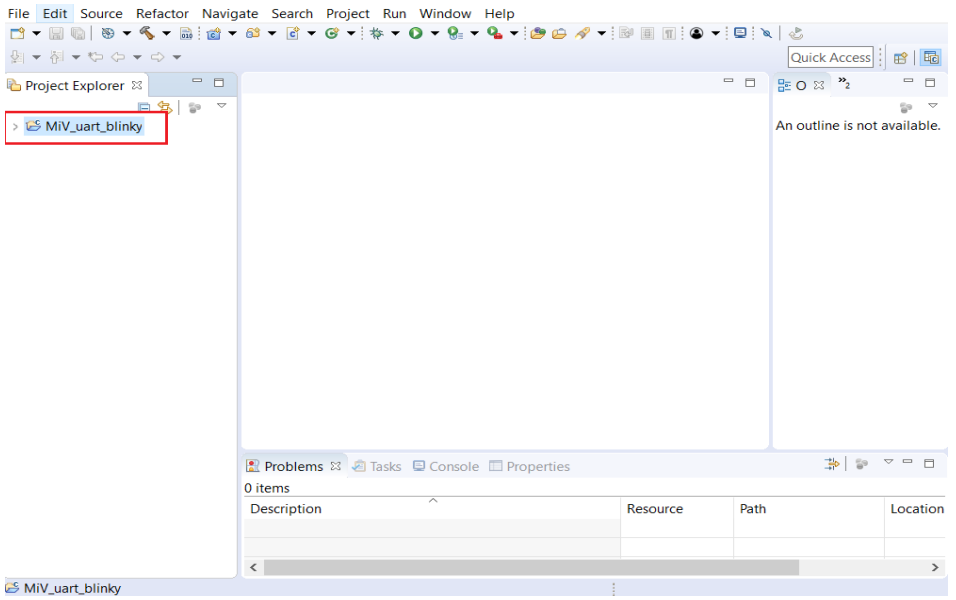
9. Ensure that the **Toolchain name** and **Toolchain path** in the **GNU RISC-V Cross Toolchain** dialog box are set as shown in the following figure.

Figure 3-5. GNU RISC-V Cross Toolchain



10. Click **Finish** in the **GNU RISC-V Cross Toolchain** wizard.
    An empty Mi-V project (MiV_uart_blinky) is created, as shown in the following figure.

Figure 3-6. Empty Mi-V Project



3.2 Downloading the Firmware Drivers (Ask a Question)

The empty Mi-V project requires the MIV_RV32 Hardware Abstraction Layer (HAL) files and the following peripheral drivers:

- CoreGPIO
- CoreUARTapb
- CoreSPI

Download the MIV_RV32 HAL files and drivers from Github using the link as follows: Mi-V soft processor platform.

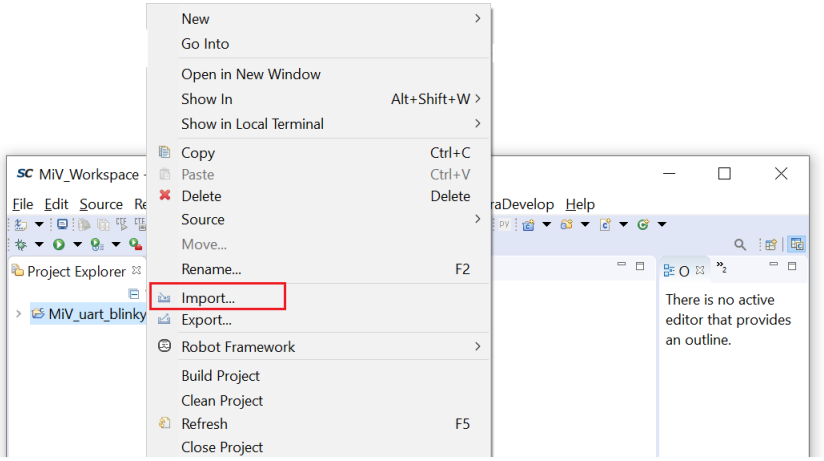
3.3 Importing the Firmware Drivers (Ask a Question)

After the driver files are downloaded, they must be imported into the empty project.

To import the drivers, perform the following steps:

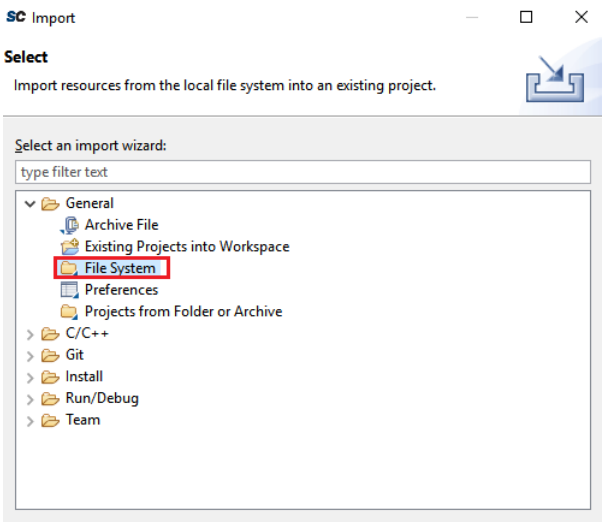
1. In SoftConsole, right-click the **MiV_uart_blinky** project, and select **Import**, as shown in the following figure.

Figure 3-7. Import Option



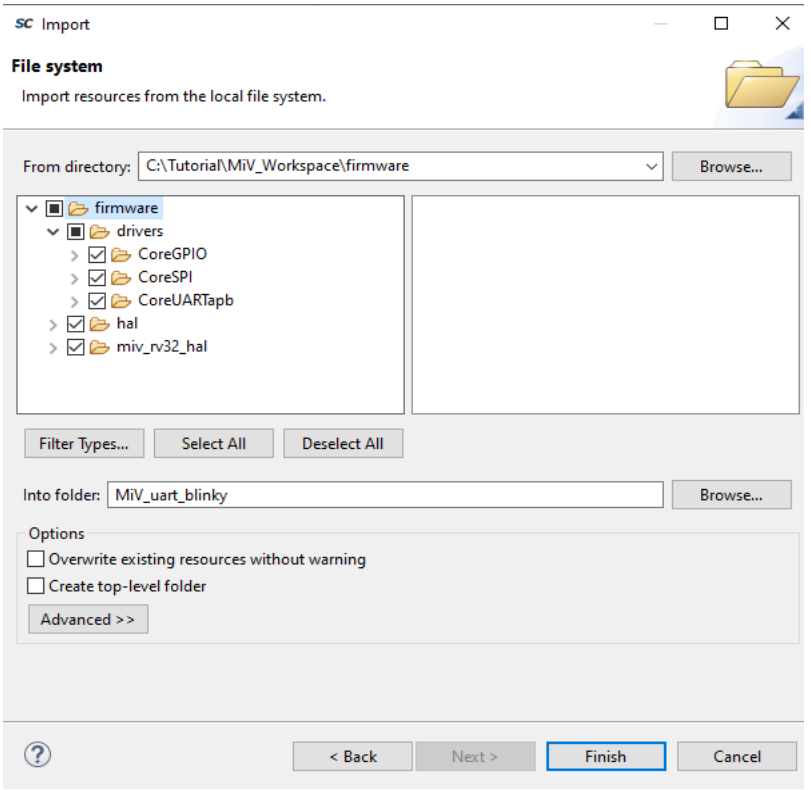
2. In the **Import** dialog box, expand the **General** folder, and double-click **File System**, as shown in the following figure.

Figure 3-8. Import Dialog Box



3. On the next page of the **Import** dialog box, do the following:
   - Click **Browse**, and locate the folder where you downloaded the drivers.
   - From the folder, select the **drivers**, **filelist**, **hal**, and **miv_rv32_hal** folders.
   - Click **Finish**.

Figure 3-9. Import Dialog Box - Page 2



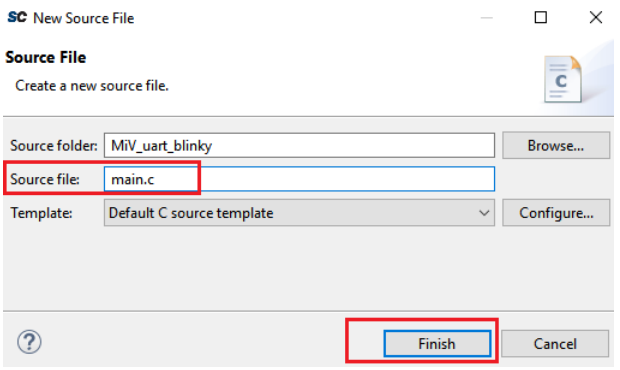
The miv_rv32_hal, hal, and driver files are imported into the **MiV_uart_blinky** project.

3.4 Creating the main.c File (Ask a Question)

To update the `main.c` file, perform the following steps:

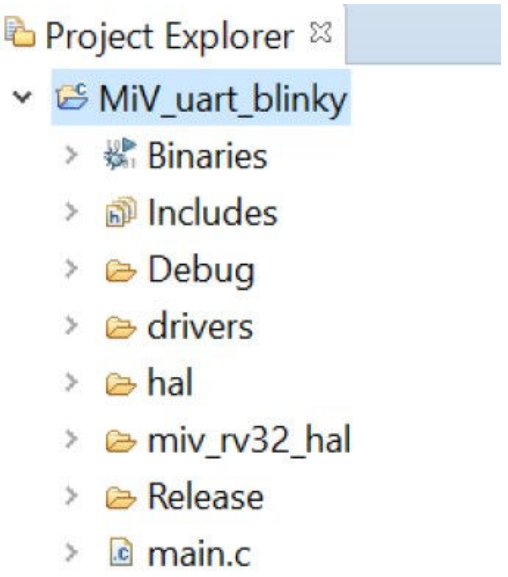
1. On the SoftConsole menu, click **File** > **New** > **Source File**.
2. In the **New Source File** dialog box, enter `main.c` in the **Source file** field, and then click **Finish**, as shown in the following figure.

Figure 3-10. main.c File Creation



The `main.c` file is created inside the project, as shown in the following figure.

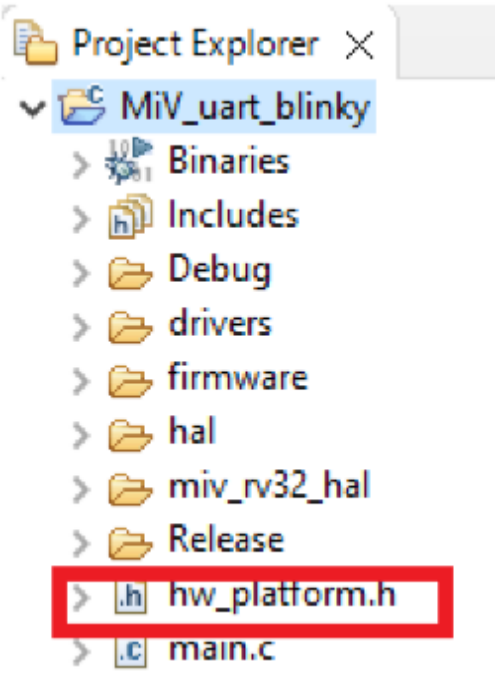
Figure 3-11. The main.c File



3. Copy all of the content of the `main.c` from `DesignFiles_directory\FW`, and paste it in the `main.c` file of the SoftConsole project.
4. Save the SoftConsole `main.c` file.
5. Similarly, create another file named `hw_platform.h`.
6. Copy all of the content of the `hw_platform.h` from the `DesignFiles_directory\FW`, and paste it in the newly created `hw_platform.h` file.

Important: The `hw_platform.h` file includes the system clock frequency, baud rate, and base addresses of peripherals. The `hw_platform.h` file appears, as shown in the following figure.

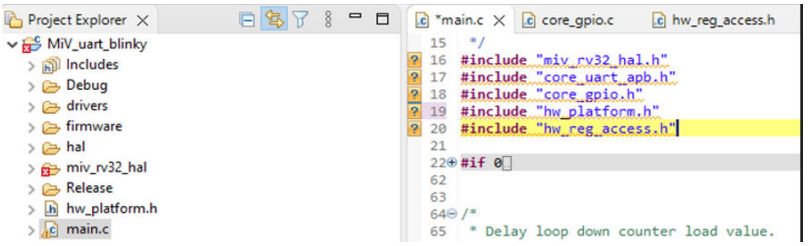
Figure 3-12. The `hw_platform.h` File



3.5 Mapping Firmware Drivers and the Linker Script (Ask a Question)

At this stage, the drivers and the `MIV_RV32 HAL` files are not mapped. Therefore, the corresponding header files in the `main.c` file are unresolved, as shown in the following figure.

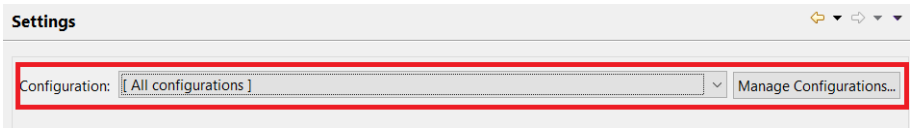
Figure 3-13. Unresolved Header Files



To map the drivers and HAL files, perform the following steps:

1. Right-click **MiV_uart_blinky** project, and select **Properties**.
2. Expand **C/C++ Build**, and select **Settings**.
3. Set the configuration to **All Configurations**, as shown in the following figure. This setting applies the upcoming tool settings to both release and debug modes.

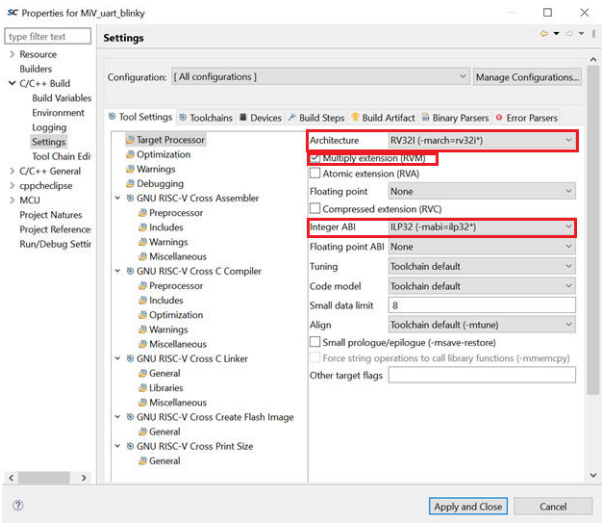
   Figure 3-14. C/C++ Build Settings



4. In the **Tool Settings** tab, expand **Target Processor**, and ensure to select the following settings.
   - Architecture: RV32I (-march=rv32i*)

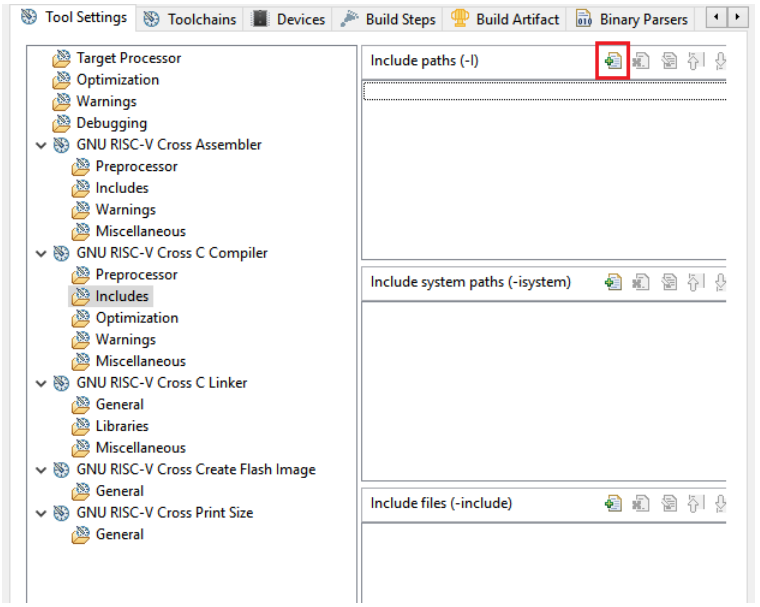
- Integer ABI: ILP32(-mabi=ilp32*)
- Multiply extension: Enabled

Figure 3-15. Target Processor Tool Settings



5. Expand **GNU RISC-V Cross C Compiler**, and select **Includes**.
6. Click the **Add** icon to add the driver and MIV_RV32 HAL directories, as shown in the following figure.

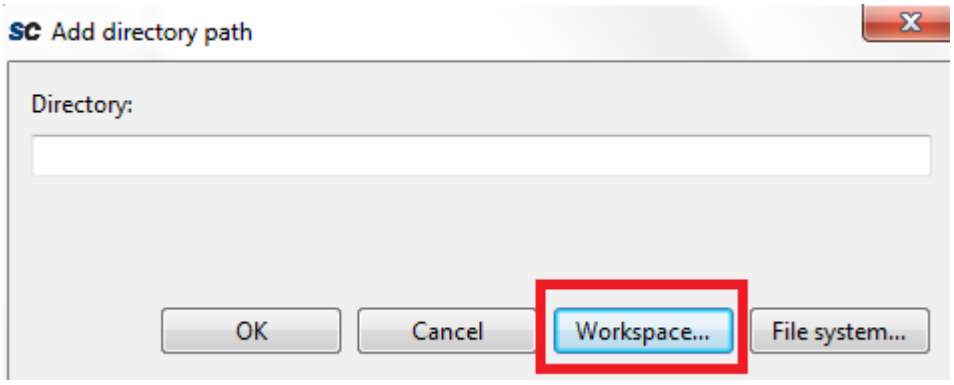
Figure 3-16. GNU RISC-V Cross C Compiler Tool Settings



Important: This application does not require including system paths and other files.

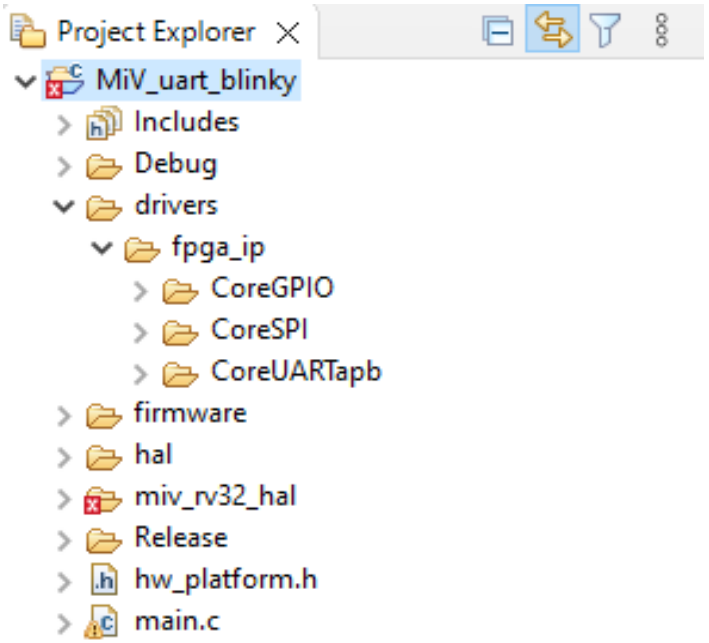
7. In the **Add directory path** dialog box, click **Workspace**, as shown in the following figure.

Figure 3-17. Add Directory Path Dialog Box



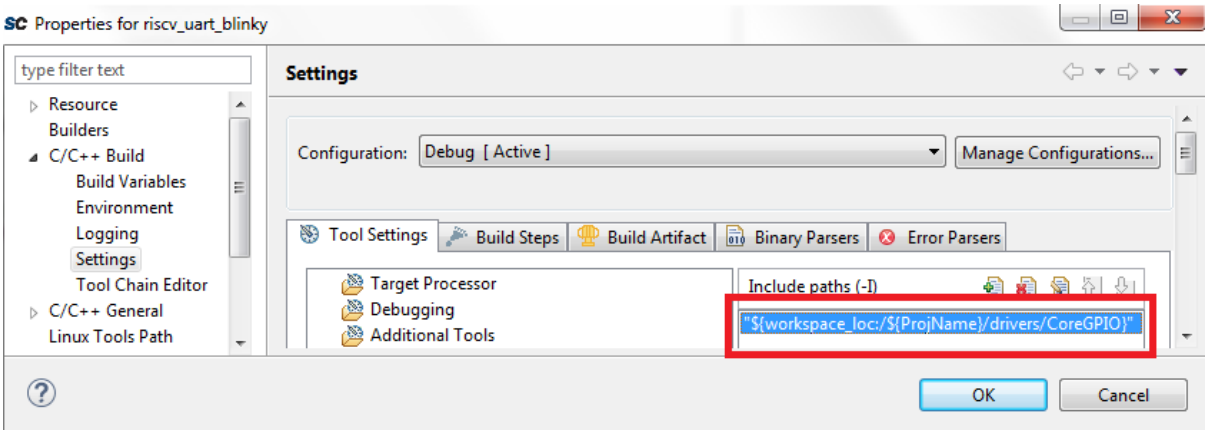
8. In the **Folder selection** dialog box, expand **MiV_uart_blinky project** > **drivers**, select the **CoreGPIO** folder, and click **OK**, as shown in the following figure.

Figure 3-18. CoreGPIO Folder Selection



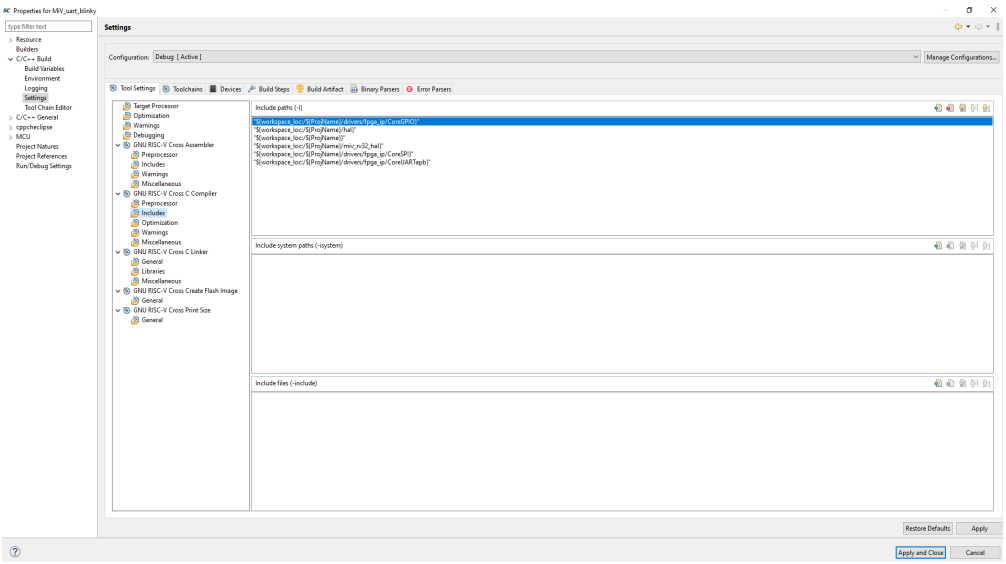
9. In the **Add directory path** dialog box, click **OK**.
The CoreGPIO folder path is added, as shown in the following figure.

Figure 3-19. Tool Settings Tab with CoreGPIO Path Added



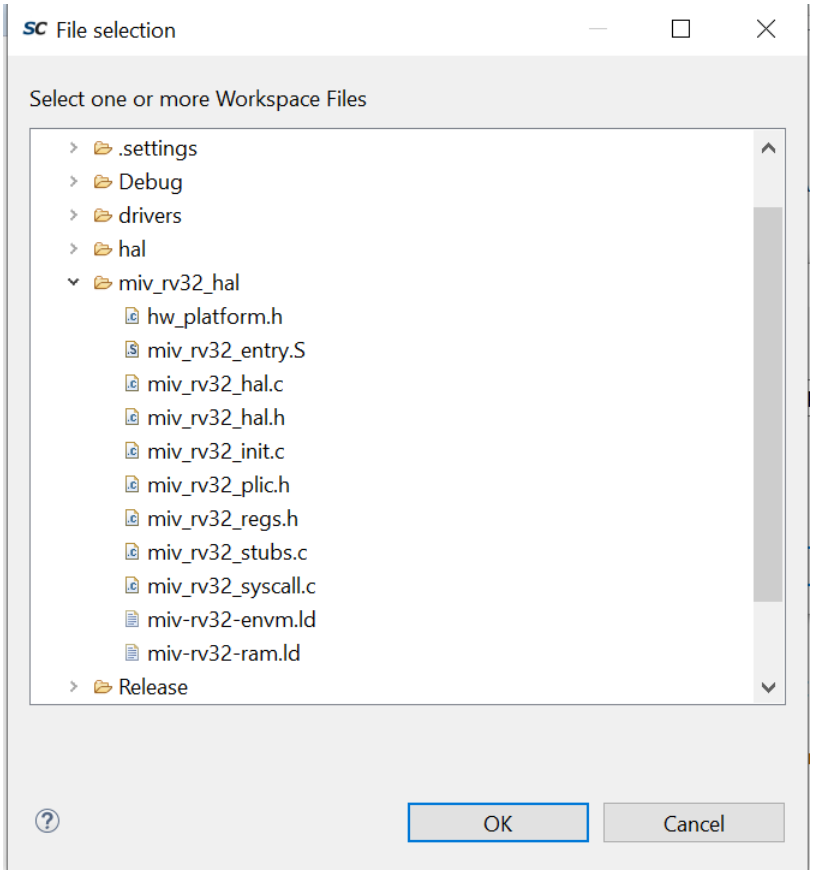
10. Repeat the preceding steps to add the **CoreUARTapb**, **CoreSPI**, **hal**, **MIV_RV32_HAL**, and **MiV_uart_blinky** (ProjName) folder paths.
    The drivers and MIV_RV32_HAL files are successfully mapped, as shown in the following figure.

Figure 3-20. Tool Settings Tab After Successful Mapping



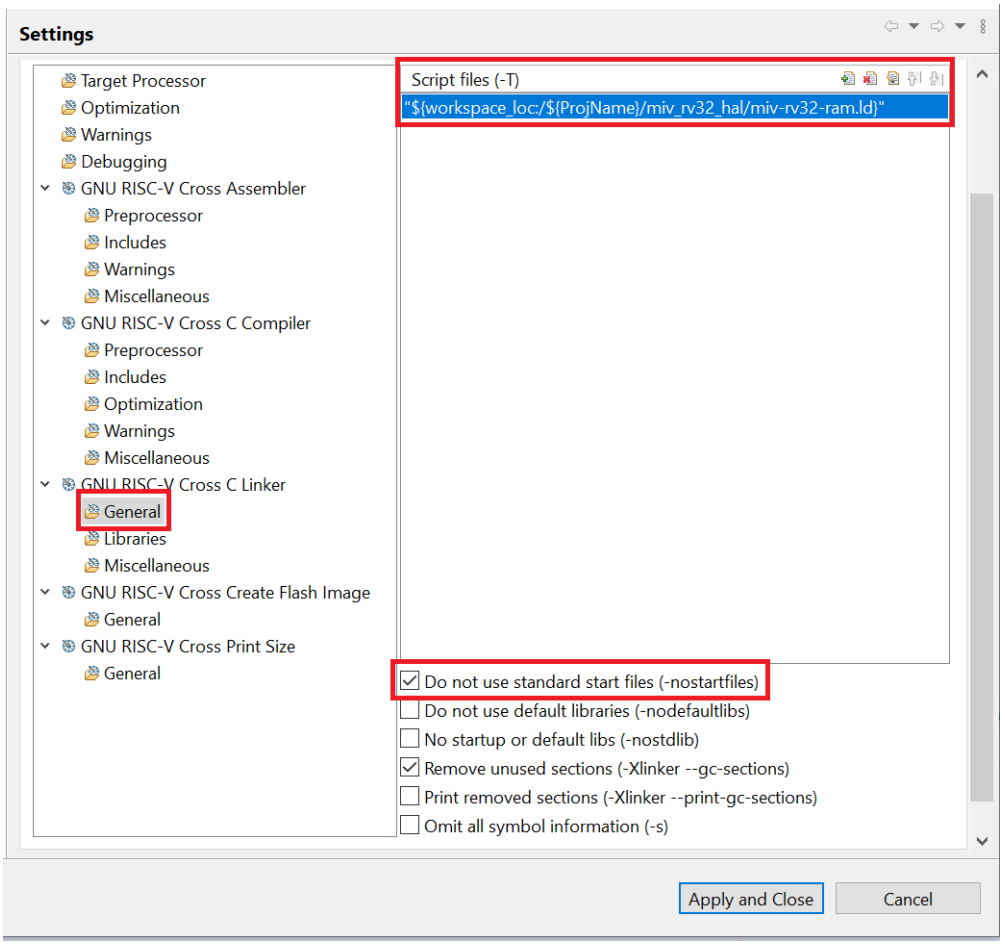
11. Select the **GNU RISC-V Cross C Linker** > **General** to map the linker script.
12. Click **Add** icon as shown in Figure 3-16, and in the **Add file path** dialog, click **Workspace** as shown in Figure 3-17.
13. In the **File selection** dialog box, expand **MiV_uart_blinky** and select the linker script, as shown in the following figure.

Figure 3-21. Selecting the Linker Script



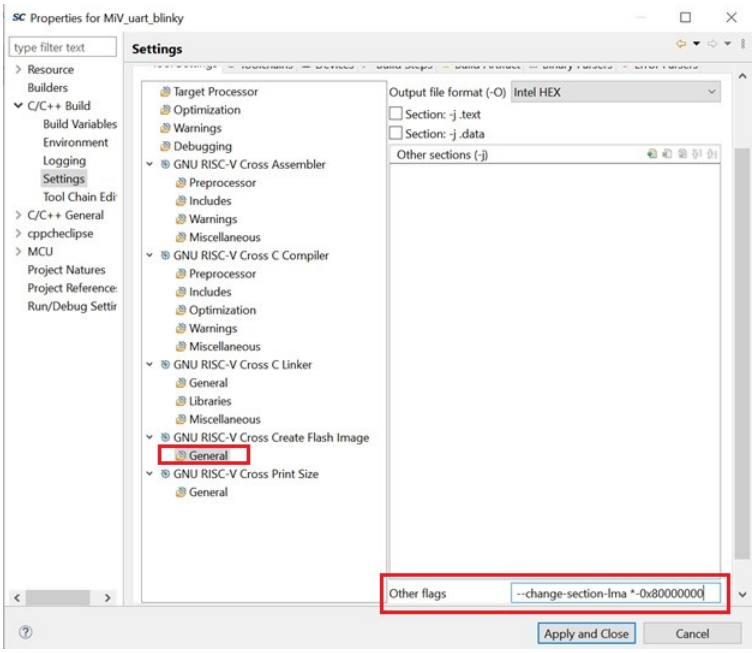
14. The linker script is mapped as shown in the following figure.

Figure 3-22. Linker Script Default Mapping



15. Select the **Do not use standard start files (-nostartfiles)** option as shown in preceding figure.
16. Select the **GNU RISC-V Cross Create Flash Image** > **General** and set **Other Flags** to "`--change-section-lma *-0x80000000`" as shown in the following figure. This excludes the extended linear record in the first line of the hex file.

Figure 3-23. RISC-V Flash Image Settings



17. Click **Apply** and when prompted to rebuild, choose **Yes**.
18. Then click **Apply** and **Close**.

The firmware drivers and linker script are successfully mapped. Notice that the header files are now resolved in the `main.c` file.

3.6 Mapping Memory and Peripheral Addresses (Ask a Question)

In the Libero design flow, the Mi-V processor execution memory address is mapped to 0x80000000, and its size is set to 64 KB. This information must be checked in the linker script before building the application.

To map the memory address, perform the following steps:

1. Open the linker script (`miv-rv32-ram.ld`) available in the MIV_RV32_HAL folder.
2. Ensure that the ram ORIGIN address is mapped to 0x80000000.
3. Ensure that the LENGTH of the ram is 64 KB.
4. Ensure that the HEAP_SIZE is 1 KB.
5. Save the file.

Important: The MTVEC_OFFSET macro places trap vectors appropriately. This macro is already defined in the `miv-rv32-ram.ld` file.

The following figure shows the linker script.

Figure 3-24. Linker Script

```
28 OUTPUT_ARCH( "riscv" )
29 ENTRY(_start)
30
31 MEMORY
32 {
33     ram (rwx) : ORIGIN = 0x80000000, LENGTH = 64k
34 }
35 STACK_SIZE            = 2k;          /* needs to be
36 HEAP_SIZE             = 1k;          /* needs to be
37
```

In the Libero design flow, the UART, GPIO, and SPI peripheral addresses are mapped to 0x61000000, 0x65000000, and 0x66000000 respectively. This information needs to be provided in the `hw_platform.h` file.

To map the peripheral address, perform the following steps:

1. Open the hardware platform header file (`hw_platform.h`).
2. Ensure that the SYS_CLK_FREQ macro is defined as 83333000UL.
3. Ensure that the COREUARTAPB0_BASE_ADDR macro is defined as 0x61000000.
4. Ensure that the COREGPIO_OUT_BASE_ADDR macro is defined as 0x65000000.
5. Ensure that the FLASH_CORE_SPI_BASE macro is defined as 0x66000000.
6. Save the file.

The following figure shows the`hw_platform.h` after these updates.

Figure 3-25. Updated hw_platform.h File

```
51 #endif
52
53 /*******************************************************************************//**
54  * Non-memory Peripheral base addresses
55  * Format of define is:
56  * <corename>_<instance>_BASE_ADDR
57  * The <instance> field is optional if there is only one instance of the core
58  * in the design
59  */
60 #define COREUARTAPB0_BASE_ADDR          0x61000000UL
61 #define COREGPIO_OUT_BASE_ADDR          0x65000000UL
62 #define FLASH_CORE_SPI_BASE             0x66000000UL
63
64 /*******************************************************************************//**
65  * Peripheral Interrupts are mapped to the corresponding Mi-V Soft processor
66  * interrupt from the Libero design.
67  *
68  * On the legacy RV32 cores, there can be up to 31 external interrupts (IRQ[30:0
69  * pins). The legacy RV32 Soft processor external interrupts are defined in the
70    riscv_plic.h
71  *
```

The memory and peripheral addresses are successfully mapped.

3.7 Setting the UART Baud Rate (Ask a Question)

The value of the `BAUD_VALUE_115200` macro in the `hw_platform.h` file must be defined according to the system clock frequency to achieve the UART baud rate of 115200. The baud value is calculated using the following formula.

BAUD_VALUE = (CLOCK / (16 * BAUD_RATE)) - 1

To define the system clock frequency:

Look for `#define SYS_CLK_FREQ` statement in the `hw_platform.h` file.

Define it as:

```
#define SYS_CLK_FREQ 83333000UL
```

The SYS_CLK_FREQ value must be same as that of the clock generated in the design.

The following figure shows the system clock frequency definition.

Figure 3-26. System Clock Frequency Definition

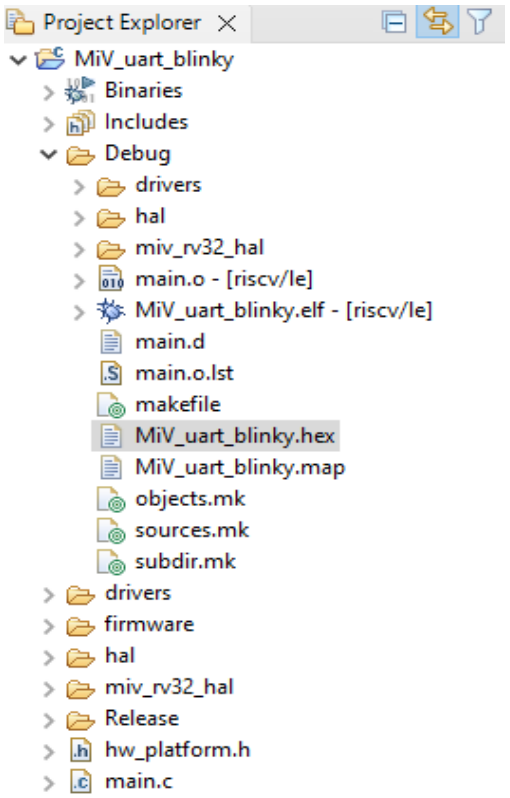
```
33
34 #ifndef HW_PLATFORM_H
35 #define HW_PLATFORM_H
36
37 /*******************************************************************************//**
38  * Soft-processor clock definition
39  * This is the only clock brought over from the Mi-V Libero design.
40  */
41 #ifndef SYS_CLK_FREQ
42 #define SYS_CLK_FREQ                    83333000UL
43 #endif
```

3.8 Building the Mi-V Project (Ask a Question)

To build the Mi-V project, right-click the **MiV_uart_blinky** project in SoftConsole, and select **Build Project**.

The project is built successfully, and the HEX file is generated in the Debug folder, as shown in the following figure.

Figure 3-27. HEX File

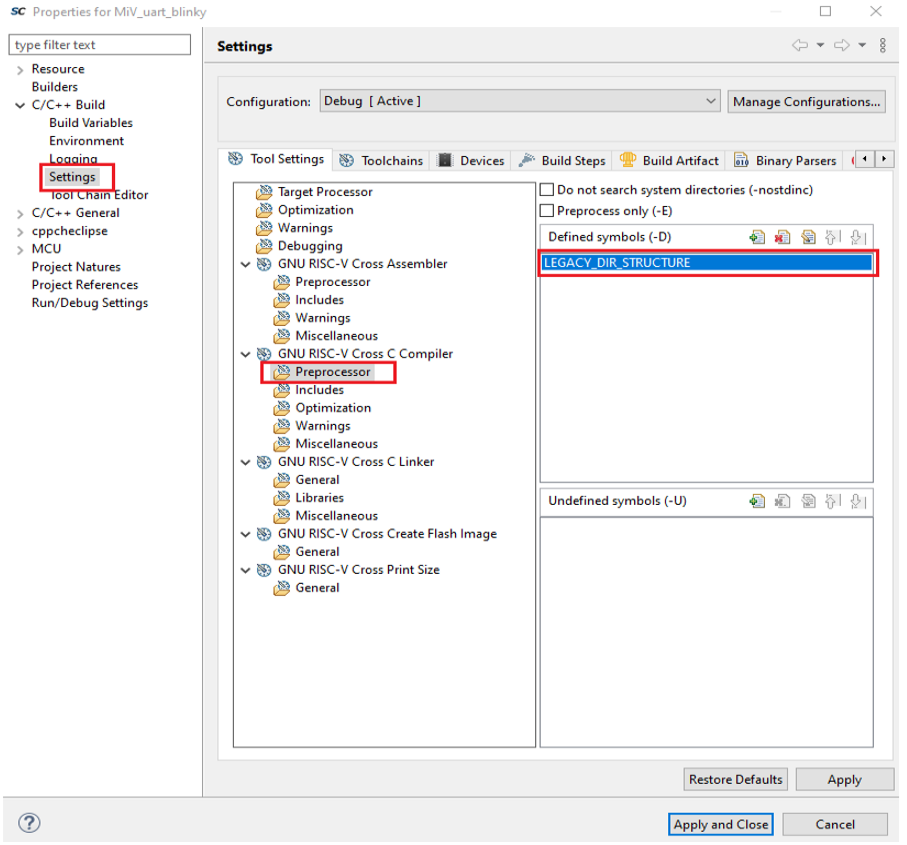


The HEX file is used for Design and Memory Initialization. For more information, see Configure Design Initialization Data and Memories.

Important: If the user is facing any issues with building the project, ensure that LEGACY_DIR_STRUCTURE is included as a defined symbol in preprocessor in the GNU RISC-V Cross C Compiler, perform the following steps:

1. Right-click **MiV_uart_blinky** project, and then go to **Properties**.
2. Go to **C/C++ Build**, click **Settings** and select **GNU RISC-V Cross Compiler** > **Preprocessor** > **Defined symbols (-D)**, and then add **LEGACY_DIR_STRUCTURE** as shown in the following figure.

Figure 3-28. Properties for MiV_uart_blinky



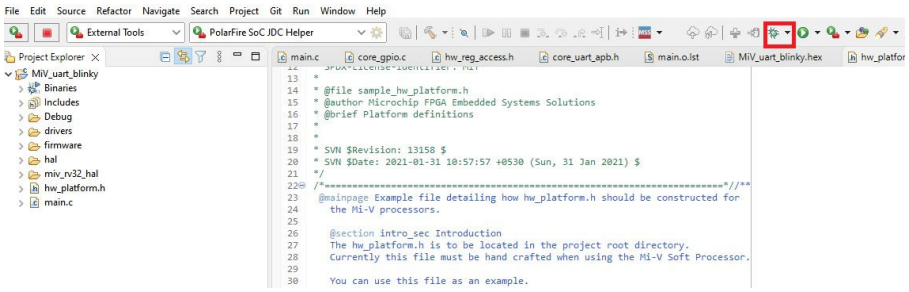
3.9 Debugging the User Application Using SoftConsole (Ask a Question)

Before debugging, the board and the serial terminal must be set up. For more information about the board and serial terminal setup, see Board Setup and Serial Terminal Emulation Program (PuTTY) Setup.

To debug the application, perform the following steps:

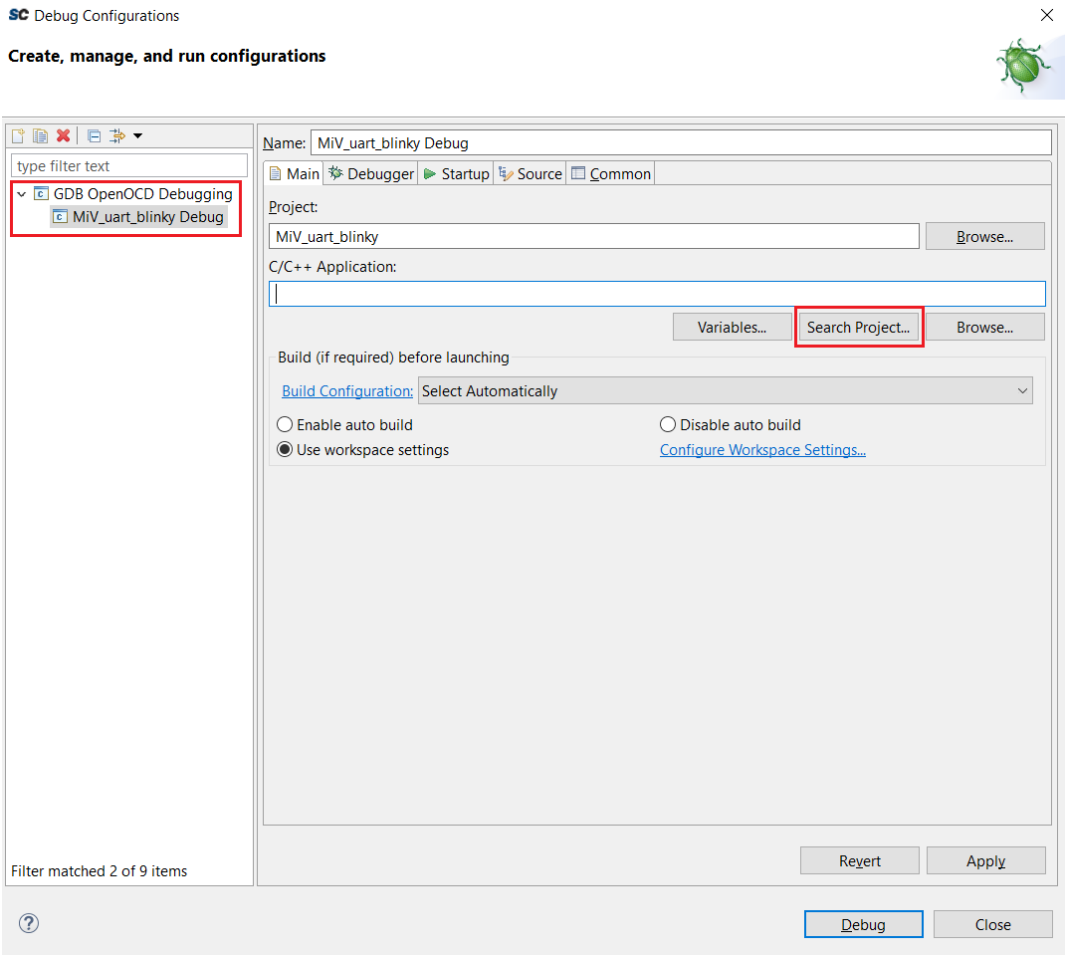
1. From the **Project Explorer**, select the **MiV_uart_blinky** project, and then click the **Debug** icon from the SoftConsole toolbar, as shown in the following figure.

Figure 3-29. Debug Icon



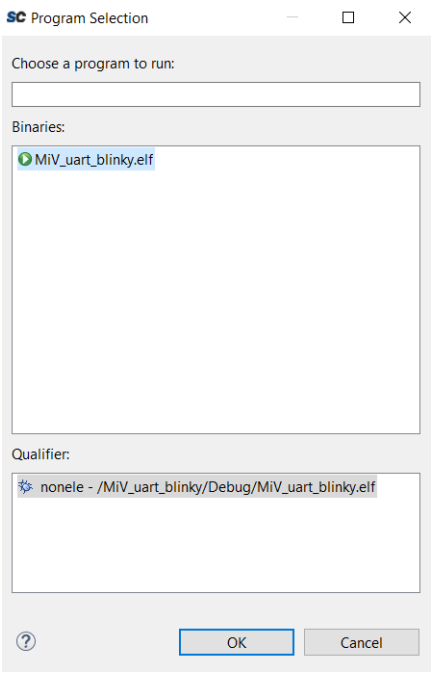
2. In the **Create, manage, and run configurations** window, double-click **GDB OpenOCD Debugging** to generate the debug configuration for the **MiV_uart_blinky** project.
3. Select the generated **MiV_uart_blinky Debug** configuration, and click **Search Project** (if by default not available), as shown in the following figure.

Figure 3-30. Create, manage, and run configurations Window – Main Tab



4. Select the **MiV_uart_blinky.elf** binary from the **Program Selection** window, and click **OK**, as shown in the following figure.

Figure 3-31. MiV_uart_blinky.elf Selection



5. Go to the **Debugger** tab, and replace the **Config options**, **Executable name**, and **Commands** as follows:
   - Config Options: `--file board/microsemi-riscv.cfg`
   - Executable name: `${cross_prefix}gdb${cross_suffix}`
   - Commands:

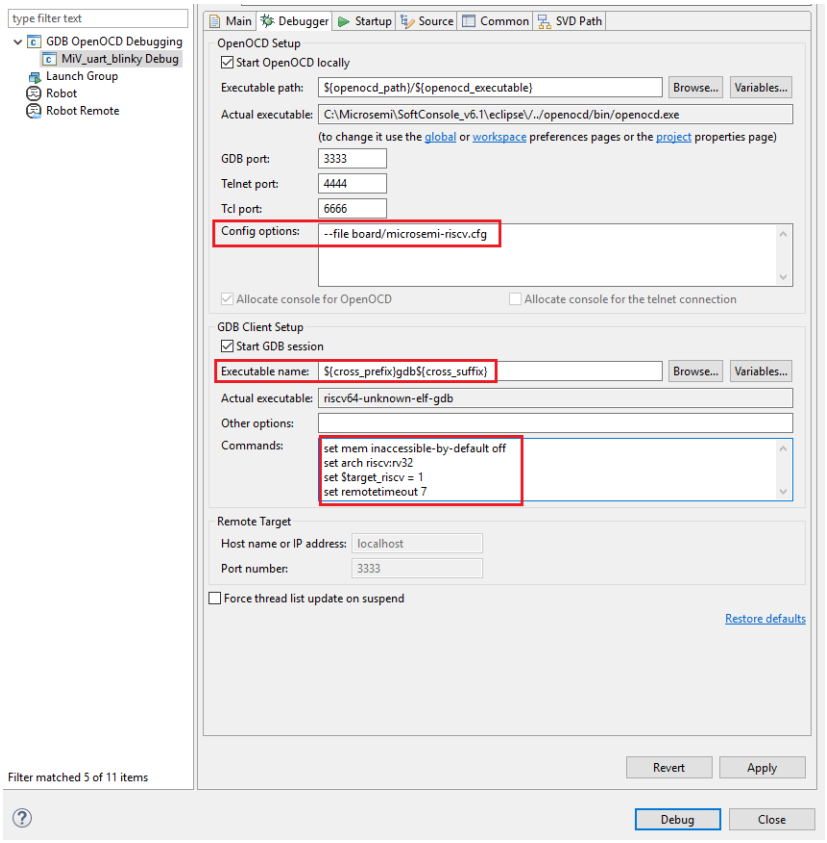
```
set mem inaccessible-by-default off
```

```
set arch riscv:rv32
```

```
set $target_riscv = 1
```

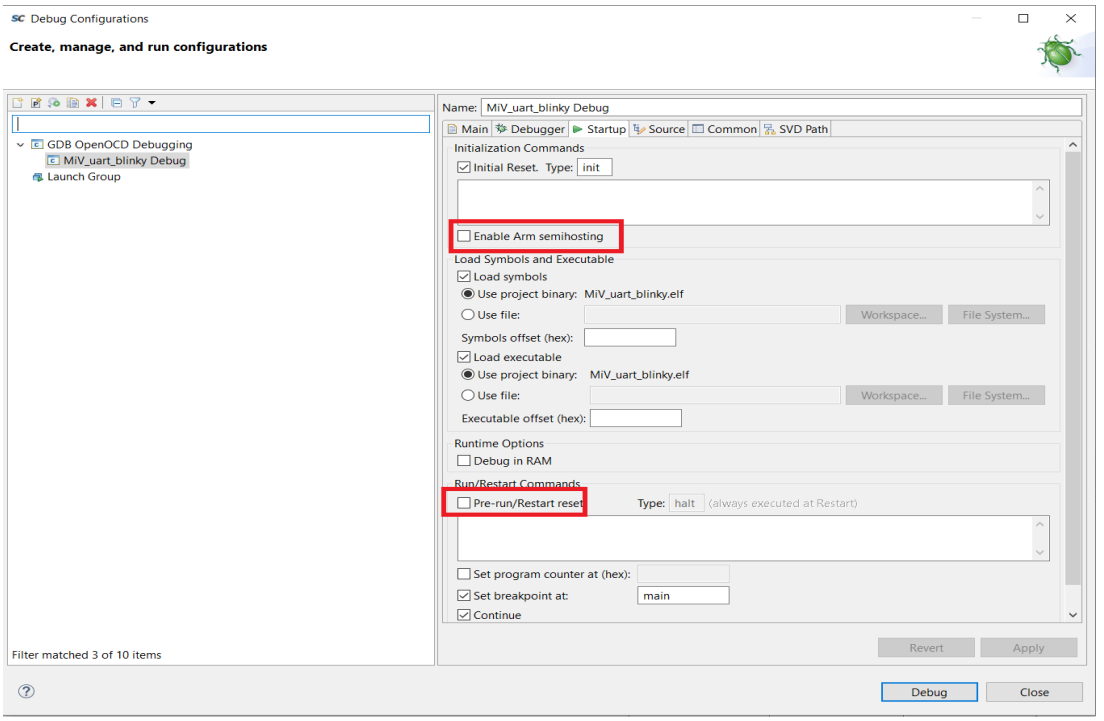
```
set remotetimeout 7
```

Figure 3-32. Create, manage, and run configurations Window – Debugger Tab



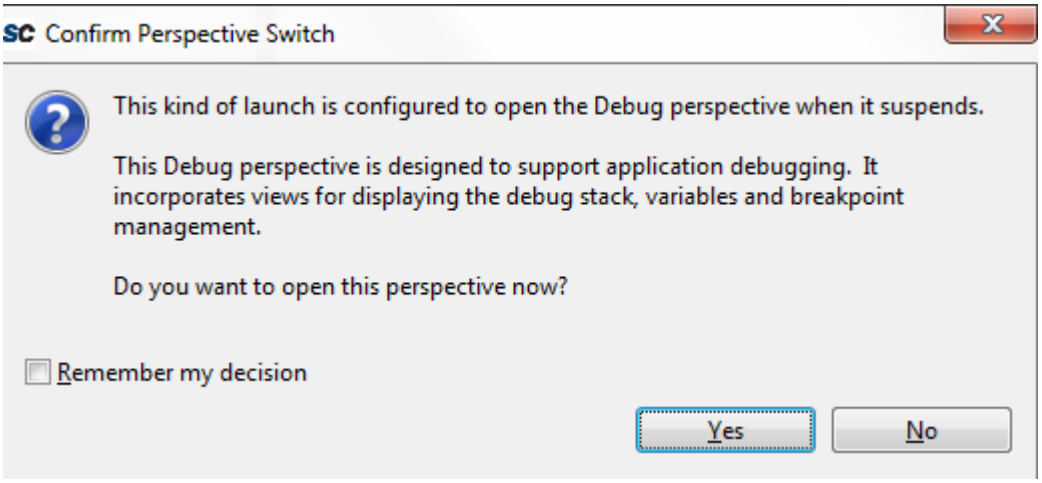
6. In **Debug Configurations** > **Startup** tab, clear the **Pre-run/Restart reset** check box to halt the program at the main () function, and clear the **Enable ARM semihosting** check box.

Figure 3-33. Debug Settings- Startup Tab



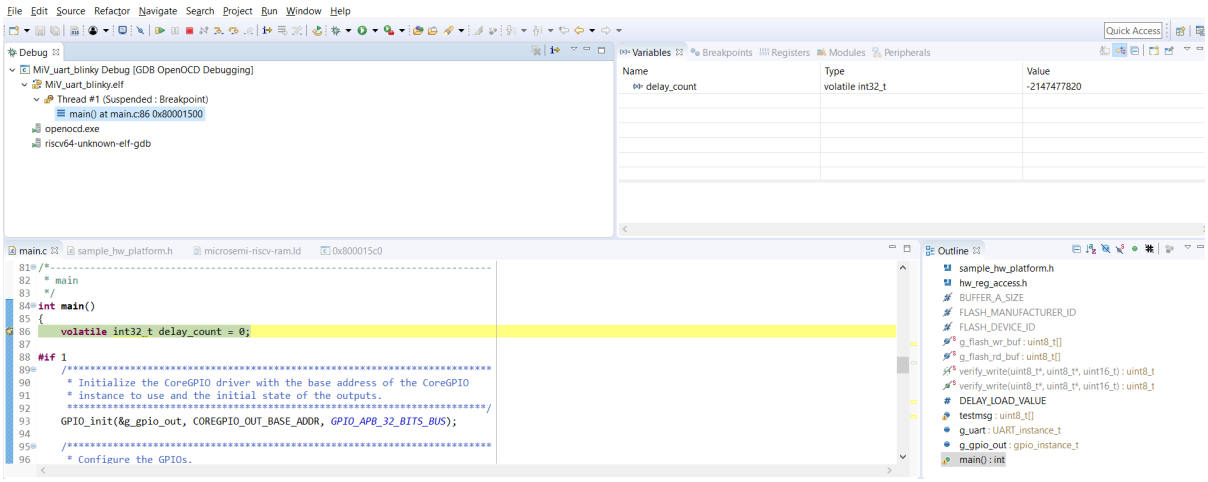
7. Click **Apply**, and then click **Debug**, as shown in the preceding figure.
   The **Confirm Perspective Switch** dialog opens, as shown in the following figure.

Figure 3-34. Confirm Perspective Switch Dialog Box



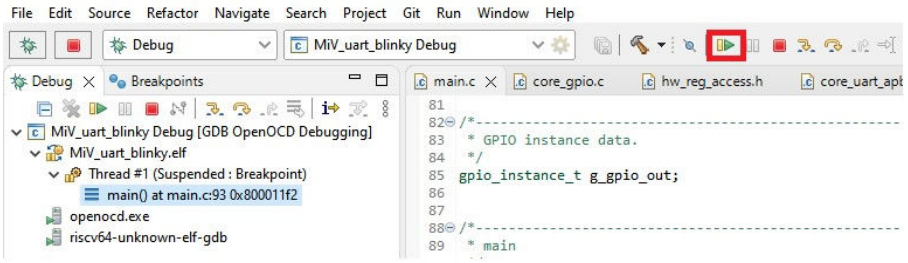
8. Click **Yes**.
   The debugger halts the execution at the first instruction in the `main.c` file, as shown in the following figure.

Figure 3-35. First Instruction in the main.c File



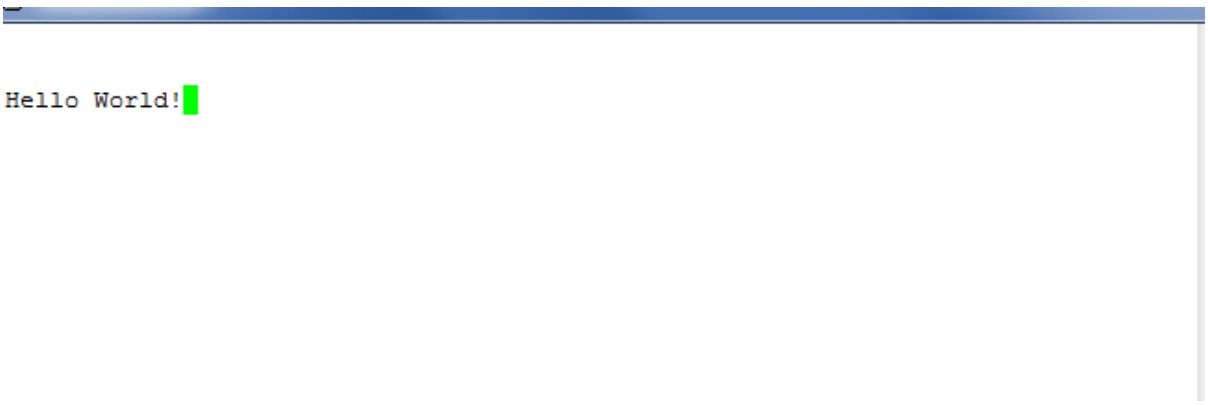
9. On the SoftConsole toolbar, click the **Resume** icon to resume the application execution, as shown in the following figure.

Figure 3-36. Resume Application Execution



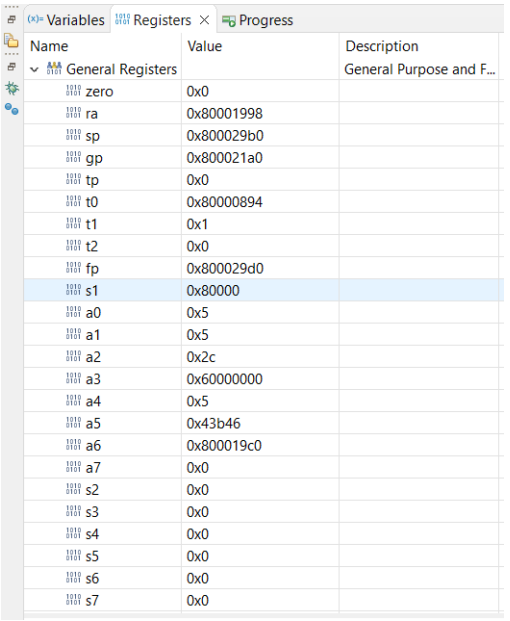
10. The string *Hello World!* is printed on the serial terminal, as shown in the following figure. Also, LEDs 4, 5, 6, and 7 on the PolarFire Evaluation Board blink.

Figure 3-37. Hello World in Debug Mode



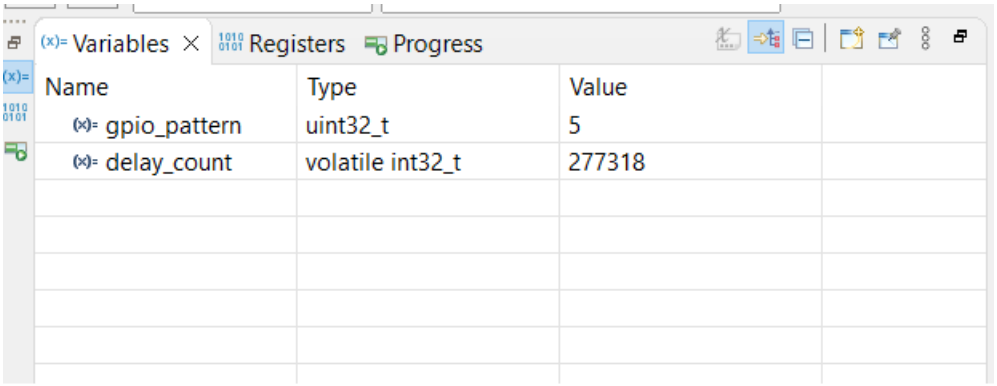
11. On the SoftConsole menu, click **Run** > **Suspend** to suspend the execution of the application.
12. Click the **Registers** tab to view the values of the Mi-V internal registers, as shown in the following figure.

Figure 3-38. Mi-V Register Values



13. Click the **Variables** tab to view the values of variables in the source code, as shown in the following figure.

Figure 3-39. Variable Values



14. From the SoftConsole toolbar, use the **Step Over** option to view the application execution line by line, or use the **Step Into** option to execute the instructions inside a function. Use the **Step Return** option to come out the function. You can also add breakpoints in the application source code.
15. On the SoftConsole toolbar, click **Terminate** to terminate the debugging of the application.
16. Close PuTTY and SoftConsole.

3.10 Debugging the User Application from DDR3 Memory (Ask a Question)

The SoftConsole debugger loads the application to the memory-mapped RAM based on the RAM start address specified in the `miv-rv32-ram.ld linker` file. The following figure shows the RAM Start Address parameters in the linker file.

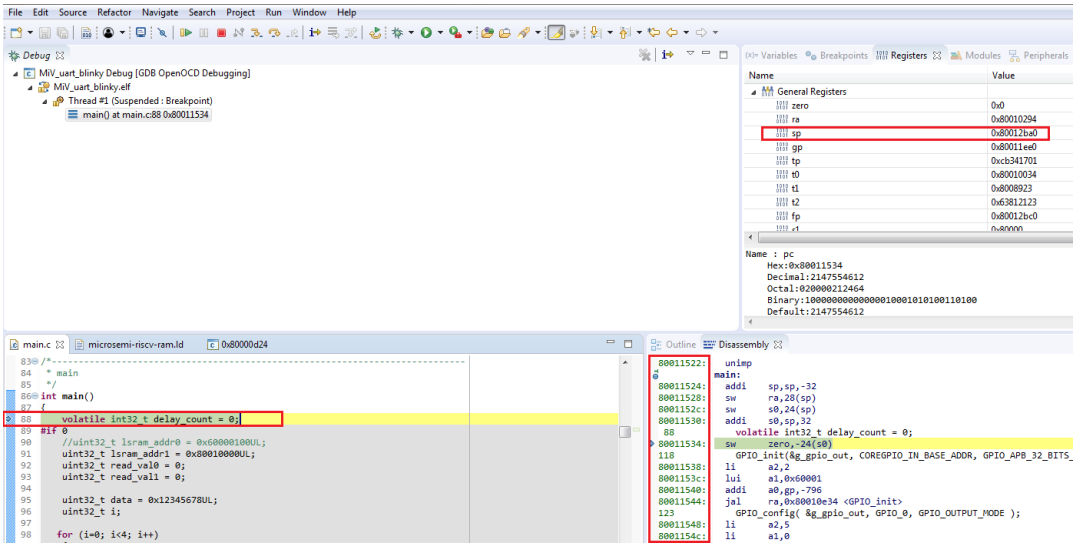
Figure 3-40. RAM Start Address Parameters

```
28 OUTPUT_ARCH( "riscv" )
29 ENTRY(_start)
30
31 MEMORY
32 {
33     ram (rwx) : ORIGIN = 0x80000000, LENGTH = 64k
34 }
35 STACK_SIZE          = 2k;          /* needs to be
36 HEAP_SIZE           = 1k;          /* needs to be
37
```

The SoftConsole reference project specifies the TCM start address, which is 0x80000000 (as shown in the preceding figure). To perform application debugging from DDR3 memory, modify this value to the DDR3 memory starting address, 0x80010000. After modifying the value, clean and build the project.

When the application is debugged from DDR3, the stack pointer and locations in the disassembly must point to DDR3 address, as shown in the following figure.

Figure 3-41. Debugging from DDR3



4. Appendix 1: Programming the Device using FlashPro Express (Ask a Question)

This chapter describes how to program the PolarFire device with the Job programming file using a FlashPro programmer. The default location of the `.job` file is: `mpf_an4997_df\Programming_files`

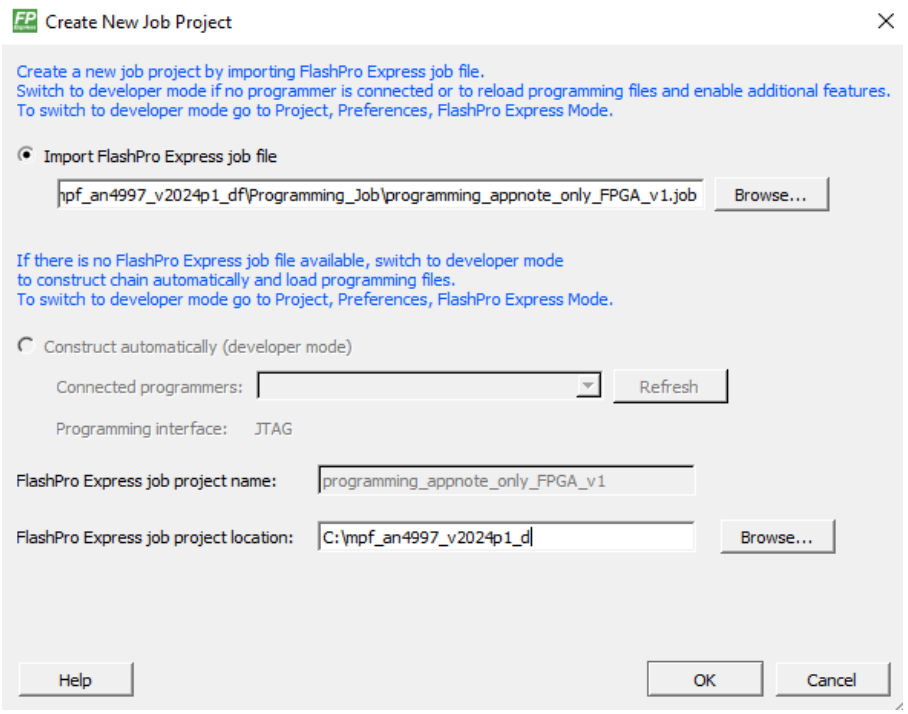
To program the PolarFire device using FP Express, perform the following steps:

1. Ensure that the jumper settings on the board are the same as listed in Table 2-4.

> **Important:** Switch off the power supply while making the jumper connections.

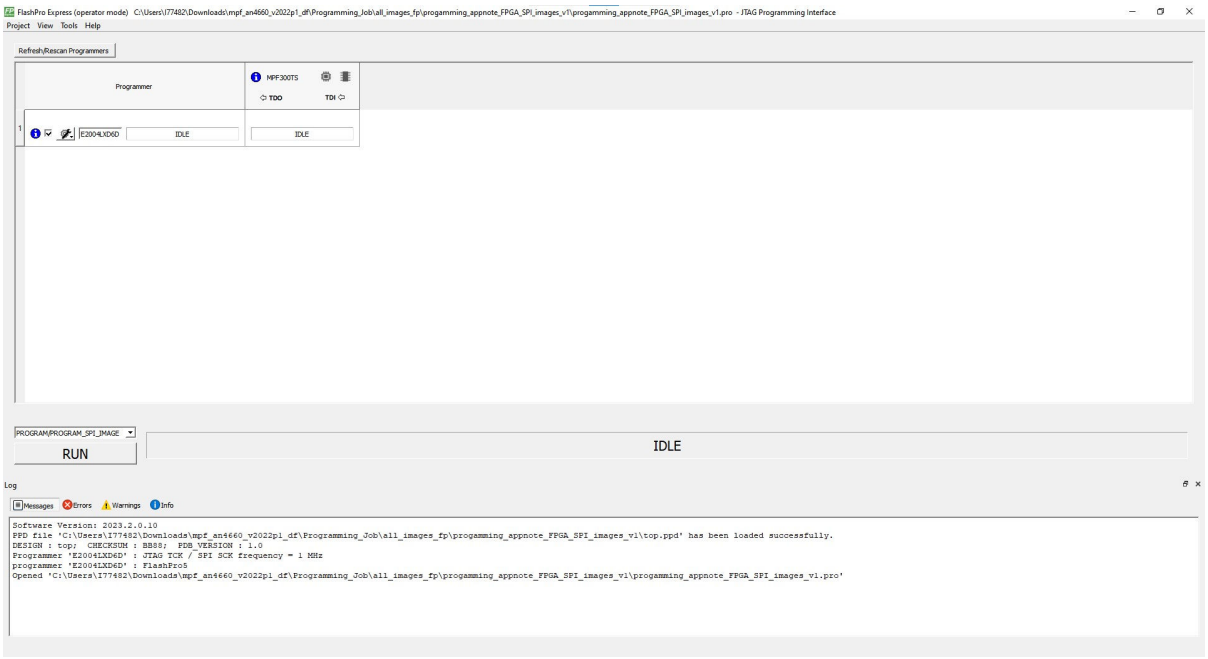
2. Connect the power supply cable to the J9 connector on the board.
3. Connect the USB cable from the Host PC to the J5 (FTDI port) on the board.
4. Power ON the board using the SW3 slide switch.
5. On the host PC, launch the FlashPro Express software.
6. To create a new job project, click **New** or select **New Job Project** from **Project** menu.
7. Enter the following in the **Create New Job Project** dialog box:
   - **Import FlashPro Express job file**: Click **Browse**, and navigate to the location where the `.job` file is located and select the file. The default location is: `<download_folder>\mpf_an4997_df\Programming_files`.
     - `mpf_an4997_df\Programming_files\top_RevD`
     - `mpf_an4997_df\Programming_files\top_RevF`
   - **FlashPro Express job project location**: Click **Browse** and navigate to the location where you want to save the project.

Figure 4-1. New Job Project from FlashPro Express Job



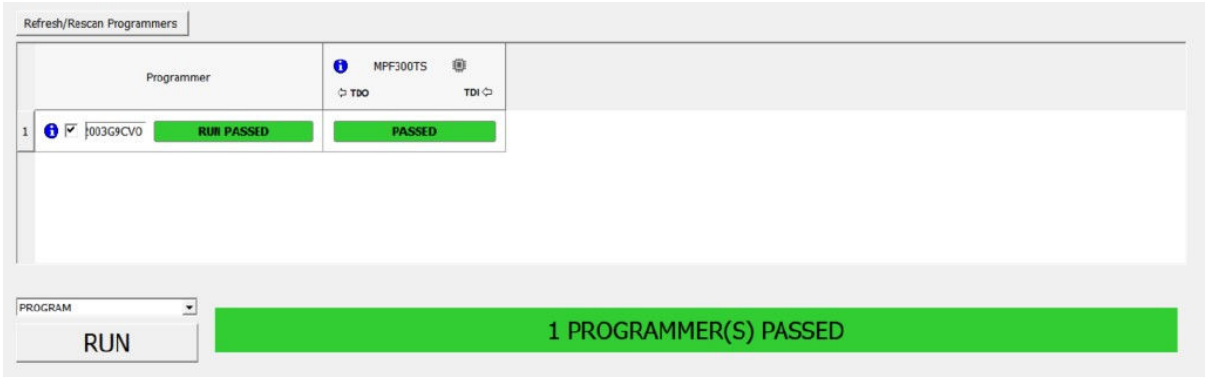
8. Click **OK**. The required programming file is selected and ready to be programmed in the device.
9. The FlashPro Express window appears as shown in the following figure. Confirm that a programmer number appears in the Programmer field. If it does not, confirm the board connections and click **Refresh/Rescan Programmers**.

Figure 4-2. Programming the Device



10. Click **RUN**. When the device is programmed successfully, a **RUN PASSED** status is displayed as shown in the following figure.

Figure 4-3. FlashPro Express—RUN PASSED



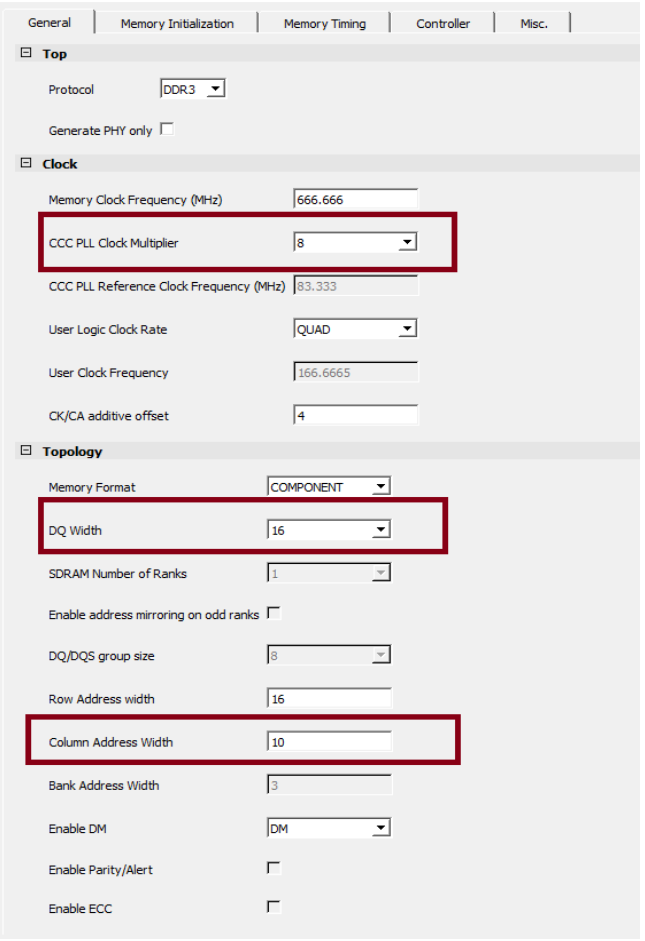
11. Close **FlashPro Express** or in the **Project** tab, click **Exit**.

5. Appendix 2 - DDR3 Configuration (Ask a Question)

If you are using Rev F kit, the following are the configurations for DDR3 controller with the initialization and timing parameters for **MT41K512M8DA-107: P** part present on the Rev F PolarFire Evaluation kit.

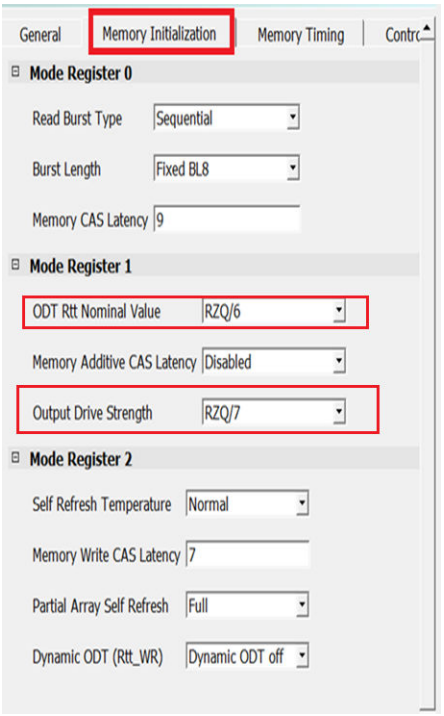
1. On **General** tab, set **CCC PLL Clock Multiplier** to **8**, **DQ Width** as **16**, and **Column Address Width** as **10**, as shown in following figure. The clock multiplier value of 8 sets the CCC PLL reference clock frequency to 83.333 MHz.
   A reference clock of this frequency is required for the PLL present inside the DDR3 subsystem. The PLL generates a 666.666 MHz DDR3 memory clock frequency and a 166.666 MHz DDR3 AXI clock frequency. The **DQ Width** is set to **16** to match the width of the DDR3 memory present on the board.

   Figure 5-1. General Tab



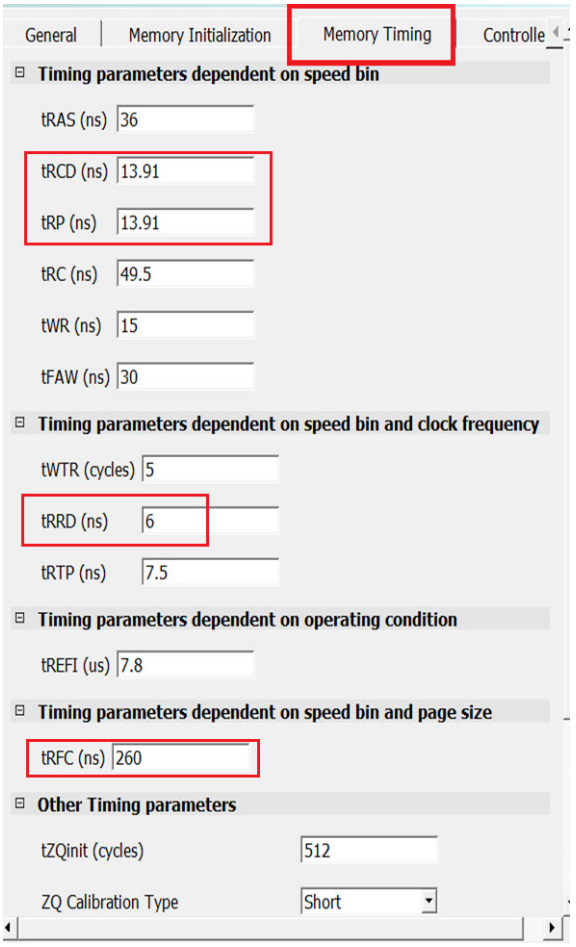
2. On the **Memory Initialization** tab, set the initialization configuration settings for the DDR3 memory as shown in the following figure.

Figure 5-2. Memory Initialization



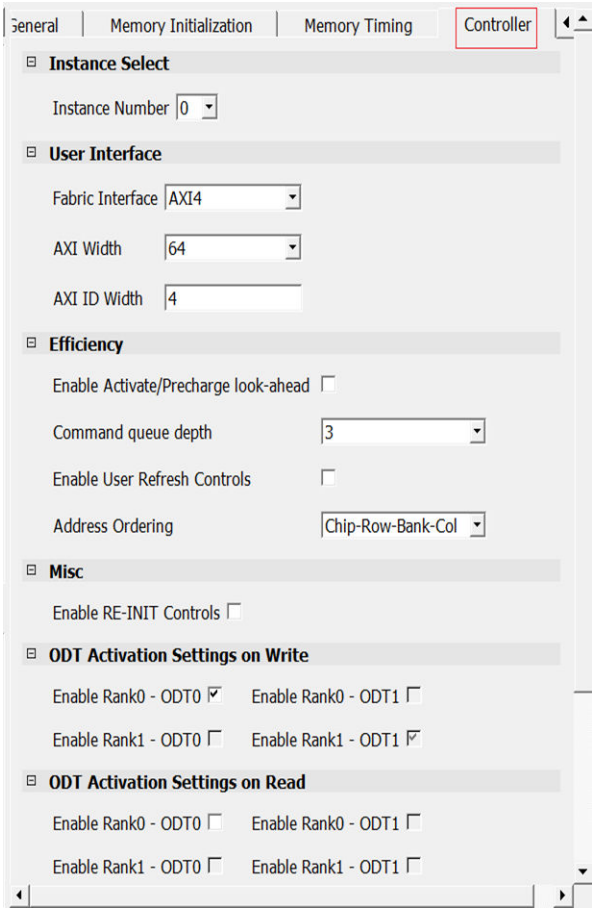
3. On the **Memory Timing** tab, set the timing configuration settings for the DDR3 memory as shown in the following figure.

Figure 5-3. Memory Timing



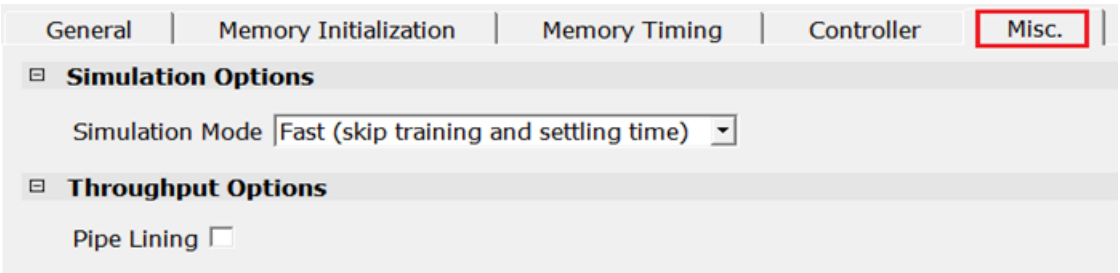
4. On the **Controller** tab, set the controller configuration settings for the DDR3 memory as shown in the following figure.

Figure 5-4. Controller



5. On the **Misc.** tab, set the miscellaneous configuration settings for the DDR3 memory as shown in the following figure.

Figure 5-5. Misc



6. Appendix 3 - References (Ask a Question)

This section lists documents that provide more information about RISC-V and other IP cores used to build the RISC-V subsystem.

- For more information about MIV_RV32, see *MIV_RV32 Handbook* from the Libero SoC Catalog.
- For more information about CoreJTAGDebug, see CoreJTAGDebug User Guide.
- See the following documents on PolarFire SoC Documentation web page:
  - For more information about PolarFire Initialization Monitor, see PolarFire Family Power-Up and Resets User Guide.
  - For more information about PolarFire Clock Conditioning Circuitry (CCC), see PolarFire Family Clocking Resources User Guide.
- For more information about Libero, ModelSim, and Synplify, see Libero SoC Documentation.
- For more information about SoftConsole, see SoftConsole
- For more information about loading a Job file using FlashPro Express, see the User Guide from **FlashPro Express** > **Help** > **User Guide**.

7. Revision History (Ask a Question)

The revision history describes the changes that were implemented in the document. The changes are listed by revision, starting with the current publication.

Table 7-1. Revision History

Revision	Date	Description
C	01/2025	The following is a summary of the changes made in revision C of this document. • Updated the document for Libero® v2024.2. • Updated the `.job` filepath and TCL script filepath throughout the document. • Updated Figure 2-10 and Figure 2-11 in the Instantiating DDR3 Memory Controller section. • Updated Figure 5-1 in the Appendix 2 - DDR3 Configuration section.
B	07/2024	The following is a summary of the changes made in revision B of this document. • Updated the document for Libero v2024.1 • Updated Figure 2-4 in Instantiating Mi-V Processor IP section • Updated Figure 2-9 in Instantiating AXI Interconnect Bus IP section • Updated "DDR3_0" to "PF_DDR3_C0" in Instantiating DDR3 Memory Controller section • Updated "DDR3_0_0" to "PF_DDR3_C0_0" and "CCC_0" to "PF_CCC_0" in Instantiating PolarFire Clock Conditioning Circuitry (CCC) section • Updated Figure 2-15 in Instantiating PolarFire Initialization Monitor section • Updated "reset_syn_0" to "corereset_pf_c0" and "reset_syn_1" to "corereset_pf_c1" in Instantiating CORERESET_PF section • Updated "COREJTAGDebug_0" to "corejtagdebug_c0" in Instantiating CoreJTAGDebug section • Added Instantiating MiV ESS Core section • In Connecting IP Instances in SmartDesign section: – Added Figure 2-22 – Updated "reset_syn_0" to "CORERESET_PF_C0_0", "reset_syn_1" to "CORERESET_PF_C1_0", "CCC_0_0" to "PF_CCC_C0_0", "COREAXI4INTERCONNECT_C1" to "coreaxi4interconnect_c0_0" – Updated Figure 2-23 – Added MIV_ESS: PCLK in Table 2-1 – Updated "DDR3_0_0:SYS_reset_N" to "PF_DDR3_C0_0: SYS_RESET_N" and "COREJTAGDebug_0_0" to "COREJTAGDEBUG_C0_0" – Updated "CoreJTAGDebug_0_0:TGT_TRSTB_0" to "CoreJTAGDebug_0_0:TGT_TRSTN_0" and "MIV_RV32_C0" to "MIV_RV32_C0_0" in Debug_Target Pin Connection table – Added connection description for MiV ESS Core • Updated "DDR3_0_0" to "PF_DDR3_C0_0" and added a note in Step 10 in Place and Route section • Updated Figure 2-41 in Serial Terminal Emulation Program (PuTTY) Setup section • Updated Downloading the Firmware Drivers section • Updated Figure 3-11 and Figure 3-12 in Creating the main.c File section • Updated Figure 3-24, peripheral addresses, and Figure 3-25 in Mapping Memory and Peripheral Addresses section • Added a note in the Building the Mi-V Project section • Updated Figure 3-29 in Debugging the User Application Using SoftConsole section • Updated Figure 3-40 in Debugging the User Application from DDR3 Memory section • Updated FlashPro Express to FPExpress, Figure 4-2, and Figure 4-3 in Appendix 1: Programming the Device using FlashPro Express

.....continued

Revision	Date	Description
A	06/2023	The following is a summary of the changes made in revision A of this document. • Document migrated from Microsemi template to Microchip template. • Document number was changed to DS-00004997A from TU0775. • Updated the document for Libero v2023.1.
9.0	—	The following is a summary of the changes made in this revision. • LSRAM was renamed to TCM throughout the document. • Added Figure 2-4. • Updated the reason for using the CoreAXI4Interconnect IP in Instantiating AXI Interconnect Bus IP. • Updated the start and end addresses of AXI4 Slave0 port in Instantiating AXI Interconnect Bus IP. • Updated Figure 2-8. • Updated the SYS_CLK_FREQ macro definition from 111111000UL to 83333000UL, see Mapping Memory and Peripheral Addresses.
8.0	—	The following is a summary of the changes made in this revision. • Updated for Libero SoC v2021.1. • Updated Table 1. • Added Appendix 2 - DDR3 Configuration.
7.0	—	The following is a summary of the changes made in this revision. • Updated Figure 1-1. • Replaced Figure 2-4. • Removed sections Instantiating On-chip SRAM, Instantiating the AXI3 to AHB-Lite Bridge, Instantiating the AHB-Lite Bus, and Instantiating the AHB-Lite to APB3 Bridge. • Updated section Connecting IP Instances in SmartDesign.
6.0	—	Updated for Libero SoC v12.5.
5.0	—	The following is a summary of the changes made in this revision. • Updated for Libero SoC v12.2. • Updated the design for AXI-based Mi-V Soft Processor for an enhanced performance with DDR memories. • Removed Libero SoC and SoftConsole version numbers.
4.0	—	The following is a summary of the changes made in this revision. Added Fabric RAMs Initialization. The document was updated for Libero SoC v12.0.
3.0	—	The following is a summary of the changes made in this revision. • Added Design Description. • The document was updated for Libero SoC PolarFire v2.1.
2.0	—	The following is a summary of the changes made in this revision. The document was updated for the Mi-V processor upgrade. The document was updated for Libero SoC PolarFire v2.0 and SoftConsole v5.2. For more information, see Building the User Application Using SoftConsole. Information about TCM initialization from external SPI flash was added. For more information, see Configure Design Initialization Data and Memories.
1.0	—	The first publication of this document.

Microchip FPGA Support

Microchip FPGA products group backs its products with various support services, including Customer Service, Customer Technical Support Center, a website, and worldwide sales offices. Customers are suggested to visit Microchip online resources prior to contacting support as it is very likely that their queries have been already answered.

Contact Technical Support Center through the website at www.microchip.com/support. Mention the FPGA Device Part number, select appropriate case category, and upload design files while creating a technical support case.

Contact Customer Service for non-technical product support, such as product pricing, product upgrades, update information, order status, and authorization.

- From North America, call **800.262.1060**
- From the rest of the world, call **650.318.4460**
- Fax, from anywhere in the world, **650.318.8044**

Microchip Information

Trademarks

The “Microchip” name and logo, the “M” logo, and other names, logos, and brands are registered and unregistered trademarks of Microchip Technology Incorporated or its affiliates and/or subsidiaries in the United States and/or other countries (“Microchip Trademarks”). Information regarding Microchip Trademarks can be found at https://www.microchip.com/en-us/about/legal-information/microchip-trademarks.

ISBN: 979-8-3371-0381-5

Legal Notice

This publication and the information herein may be used only with Microchip products, including to design, test, and integrate Microchip products with your application. Use of this information in any other manner violates these terms. Information regarding device applications is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. Contact your local Microchip sales office for additional support or, obtain additional support at www.microchip.com/en-us/support/design-help/client-support-services.

THIS INFORMATION IS PROVIDED BY MICROCHIP “AS IS”. MICROCHIP MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION INCLUDING BUT NOT LIMITED TO ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE, OR WARRANTIES RELATED TO ITS CONDITION, QUALITY, OR PERFORMANCE.

IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT, SPECIAL, PUNITIVE, INCIDENTAL, OR CONSEQUENTIAL LOSS, DAMAGE, COST, OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE INFORMATION OR ITS USE, HOWEVER CAUSED, EVEN IF MICROCHIP HAS BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE FULLEST EXTENT ALLOWED BY LAW, MICROCHIP’S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THE INFORMATION OR ITS USE WILL NOT EXCEED THE AMOUNT OF FEES, IF ANY, THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THE INFORMATION.

Use of Microchip devices in life support and/or safety applications is entirely at the buyer’s risk, and the buyer agrees to defend, indemnify and hold harmless Microchip from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any Microchip intellectual property rights unless otherwise stated.

Microchip Devices Code Protection Feature

Note the following details of the code protection feature on Microchip products:

- Microchip products meet the specifications contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is secure when used in the intended manner, within operating specifications, and under normal conditions.
- Microchip values and aggressively protects its intellectual property rights. Attempts to breach the code protection features of Microchip products are strictly prohibited and may violate the Digital Millennium Copyright Act.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of its code. Code protection does not mean that we are guaranteeing the product is “unbreakable”. Code protection is constantly evolving. Microchip is committed to continuously improving the code protection features of our products.