Quick Start Design Guide

PolarFire MIV_RV32IMC_ v2.1

March 2020

Contents

1 Revision History....1
1.1 Revision 1.0....1
2 Introduction....2
3 Create a Libero Project....3
3.1 Create the Core SmartDesign....5
3.2 Clock Conditioning Circuit Configuration....6
3.3 PolarFire RC Oscillator Configuration....7
3.4 CoreRESET_PF Configuration....7
3.5 PolarFire Initialization Monitor Configuration....8
3.6 Mi-V RV32IMC Configuration....9
3.7 CoreJTAGDebug Configuration....11
3.8 Memory and Peripheral Configurations—PolarFire SRAM....12
4 Peripheral Connections....13
4.1 CoreAPB3 Configuration....13
4.2 CoreUARTapb Configuration....13
4.3 CoreTimer 0 Configuration....13
4.4 CoreTimer 1 Configuration....14
4.5 CoreGPIO Input Configuration....14
4.6 CoreGPIO Output Configuration....14
4.7 Component Connections List....15
5 Design Constraints....19
6 Configure Design Initialization Data and Memories....21
6.1 Configure LSRAM Memory via LSRAM Configurator....21
6.2 Configure LSRAM Memory via Configure Design Initialization Data and Memories Configurator....21
7 Run Project from SoftConsole....24
7.1 Set System Clock Frequency and Peripheral Base Addresses....24
7.2 Build and Debug a Project....25
7.3 Communicate via UART....27

Tables

Table 1 • PF_OSC_C0_0.....15
Table 2 • PF_CCC_C0_0.....15
Table 3 • PolarFire Initialization Monitor.....15
Table 4 • MiV_RV32IMC.....15
Table 5 • CoreRESET_PF.....16
Table 6 • CoreAPB3_C0_0.....16
Table 7 • CoreUARTapb_C0_0.....17
Table 8 • CoreGPIO_IN_0.....17
Table 9 • CoreGPIO_OUT_0.....17
Table 10 • CoreJTAGDebug_C0_0.....17

Figures

Figure 1 • Create a Libero Project....3
Figure 2 • Project Name and Location....3
Figure 3 • Select Device and Part....4
Figure 4 • New Project Home Screen....4
Figure 5 • Create a New SmartDesign....4
Figure 6 • New SmartDesign Sheet....5
Figure 7 • Design Segment....5
Figure 8 • Create a Component for SmartDesign....6
Figure 9 • Clock Conditioning Circuitry (CCC) Configuration Menu....6
Figure 10 • Uncheck Expose PowerDown Port....6
Figure 11 • Change Requested Frequency to 50 MHz....7
Figure 12 • Enable 160 MHz Oscillator....7
Figure 13 • Instantiating CoreRESET_PF 8
Figure 14 • CoreRESET_PF Configurator Default Settings....8
Figure 15 • PolarFire Initialization Monitor Configurator Required Settings....9
Figure 16 • Instantiate MIV_RV32IMC Core....10
Figure 17 • MiV RV32IMC Configurator Setup....10
Figure 18 • Update the MiV_RV32IMC Memory Map....11
Figure 19 • Instantiate CoreJTAGDebug....11
Figure 20 • CoreJTAGDebug Configurator Default Settings....12
Figure 21 • PF_SRAM_AHBL_AXI Configuration....12
Figure 22 • CoreAPB3 Required Configuration....13
Figure 23 • CoreGPIO_IN Configuration....14
Figure 24 • CoreGPIO_OUT Configuration....14
Figure 25 • Final Design....18
Figure 26 • Design Flow Progression....22
Figure 27 • Design Initialization Data and Memories Configurator....22
Figure 28 • Fabric RAM Initialization Configurator....23
Figure 29 • Design Initialization Data and Memories Configurator....23
Figure 30 • Set System Clock Frequency and Peripheral Base Addresses....24
Figure 31 • Modify Memory Map Dialog....25
Figure 32 • RV32IMC Configurator Memory Map....25
Figure 33 • Build and Debug a Project....26
Figure 34 • Select a Debug Configuration....26
Figure 35 • Run a Program....27
Figure 36 • Open Device Manager....27
Figure 37 • Find COM Ports in Device Manager....28
Figure 38 • Location of Baud Rate Value....28
Figure 39 • Serial Output from COM9 Terminal....29
Figure 40 • Output from SoftConsole Terminal....29
Figure 41 • UART Clock Frequency Issues....30

1 Revision History

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

1.1 Revision 1.0

Revision 1.0 was published in March 2020. It is the first publication of this document.

2 Introduction

This guide provides quick-start steps to set up the Mi-V RV32IMC IP core in Libero, generate a bitstream, and download the FPGA data to a device. Debug software and use a serial terminal to capture UART output in SoftConsole. This guide is targeted towards the Future Avalanche Board. Note that the MIV_RV32IMC IP core is written in System Verilog.

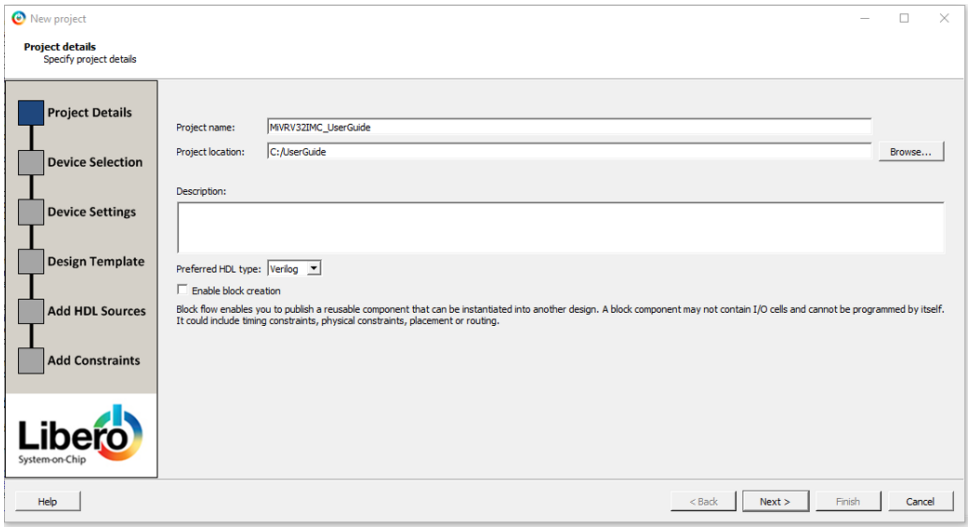
3 Create a Libero Project

Open Libero and select **Project** > **New Project** from the top menu.

Figure 1 • Create a Libero Project

From Project Details, enter **Project name** and **Project location** then click **Next**.

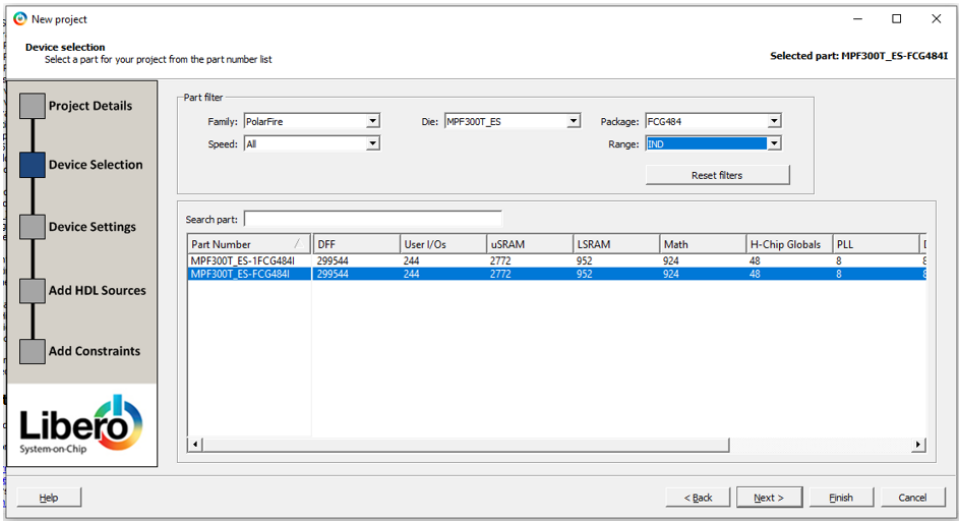
Figure 2 • Project Name and Location



From Device Selection, for example select **MPF300T_ES-FCG484I**. The following Part Filter search terms can be used to find the correct device.

- Family: PolarFire
- Die: MPF300T_ES
- Package: FCG484
- Range: IND

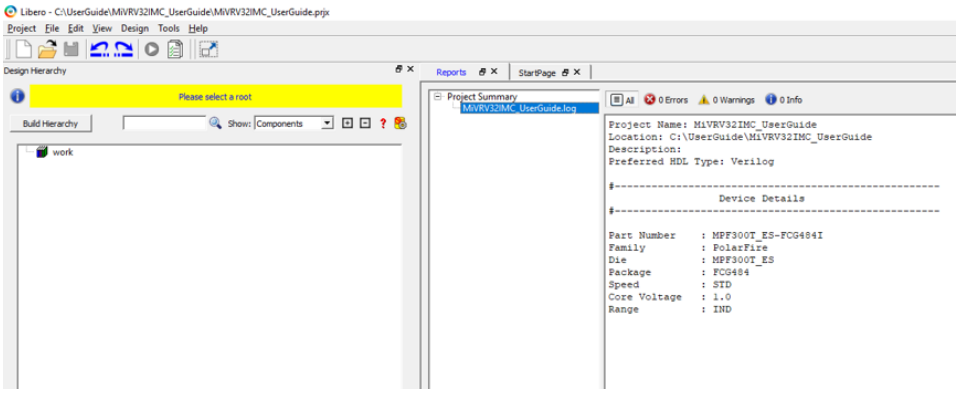
Figure 3 • Select Device and Part



Device Settings do not need to be changed, as Add HDL Sources will not be used. Add Constraints later. Click **Finish**.

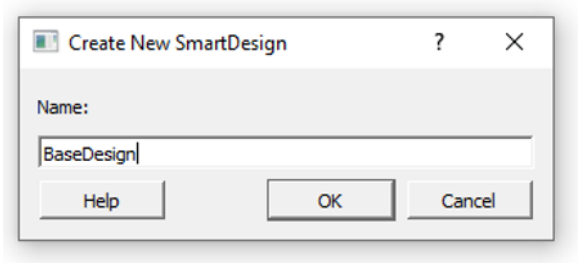
The following New Project Home Screen will appear.

Figure 4 • New Project Home Screen



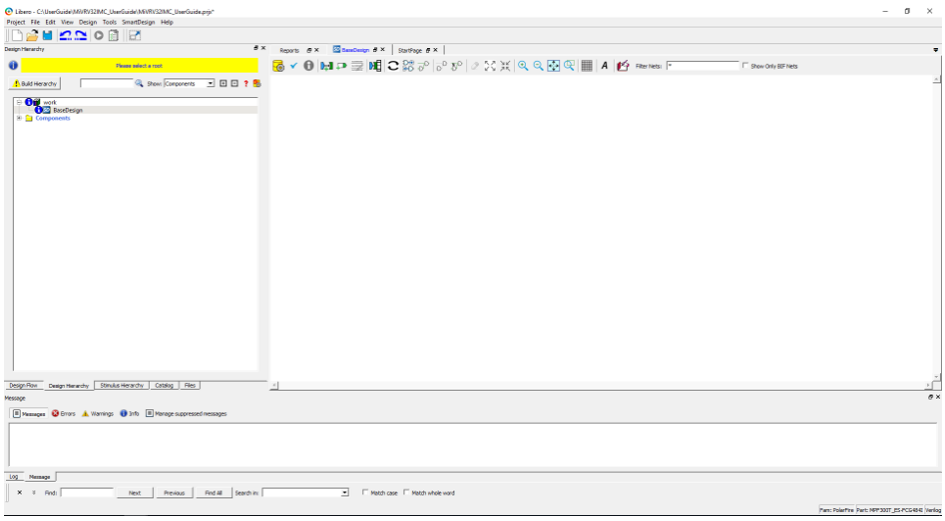
Create a new SmartDesign by selecting **File** > **New** > **SmartDesign** and name it BaseDesign.

Figure 5 • Create a New SmartDesign



Once the SmartDesign has been created, a blank canvas will appear where the example design will be created.

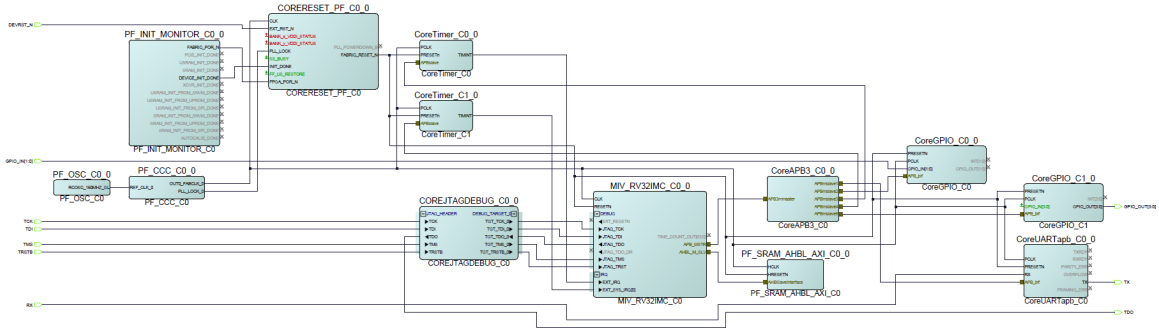
Figure 6 • New SmartDesign Sheet



3.1 Create the Core SmartDesign

The following design segment can be produced using this document.

Figure 7 • Design Segment



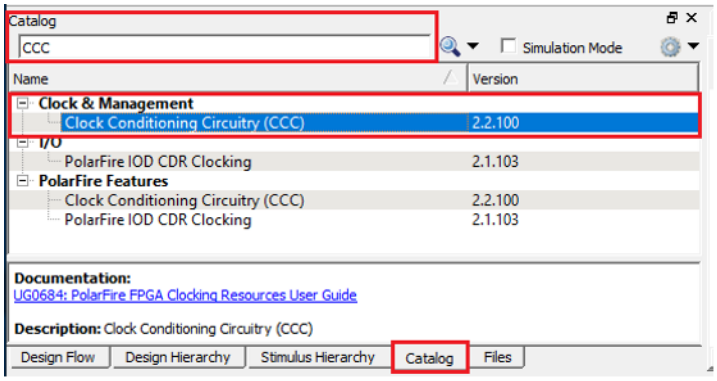
Components needed:

- 1 × PolarFire RC Oscillators (OSC) (v1.0.102)
- 1 × Clock Conditioning Circuit (CCC) (v2.2.100)
- 1 × CORERESET_PF (v2.2.107)
- 1 × PF_INIT_MONITOR (v2.0.104)
- 1 × CoreJTAGDebug (v3.1.100)
- 1 × Mi-V RV32IMC (v2.1.100)
- 1 × PF_SRAM_AHBL_AXI (v1.2.101)
- 1 × CoreAPB3 (v4.1.100)
- 1 × CoreUARTapb (v5.6.102)
- 2 × CoreGPIO (GPIO_IN and GPIO_OUT) (v3.2.102)
- 2 × CoreTimer (v2.0.103)

3.2 Clock Conditioning Circuit Configuration

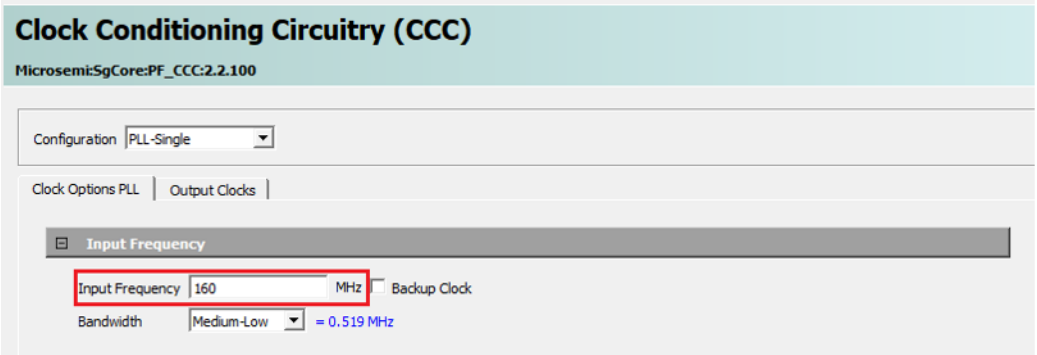
To add the Clock Conditioning Circuitry to the SmartDesign, select the **Catalog** tab, search for **CCC**, and double click on it. A dialog box will appear with the default name PF_CCC_0. Select **OK**.

Figure 8 • Create a Component for SmartDesign



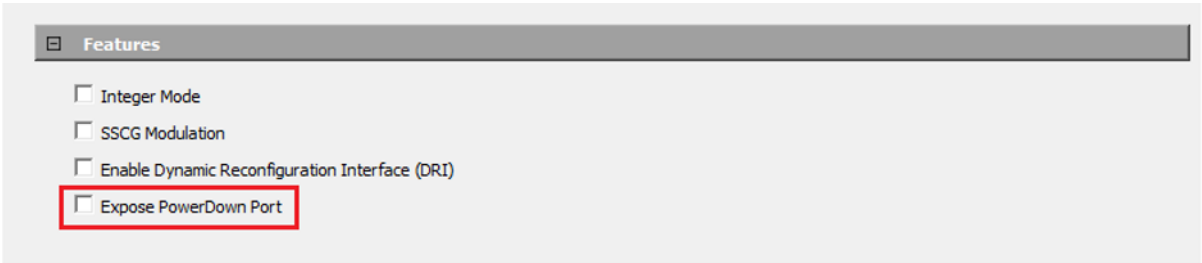
The following Clock Conditioning Circuitry (CCC) configuration menu will open. Change the clock **Input Frequency** to **160 MHz**.

Figure 9 • Clock Conditioning Circuitry (CCC) Configuration Menu



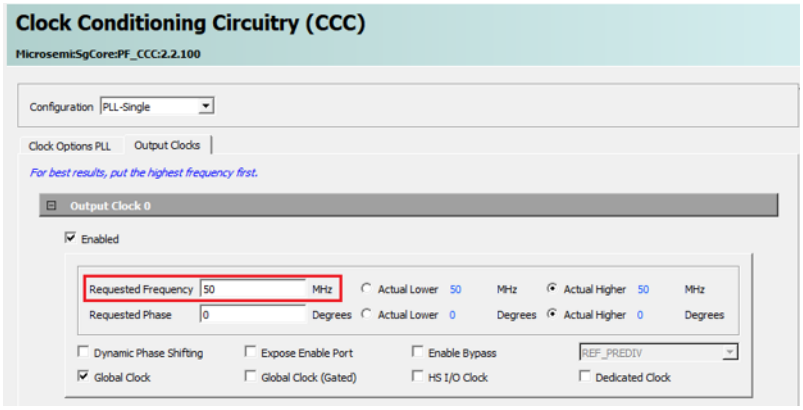
Under Features, uncheck **Expose PowerDown Port**.

Figure 10 • Uncheck Expose PowerDown Port



Under Output Clock 0, change **Requested Frequency** to **50 MHz**.

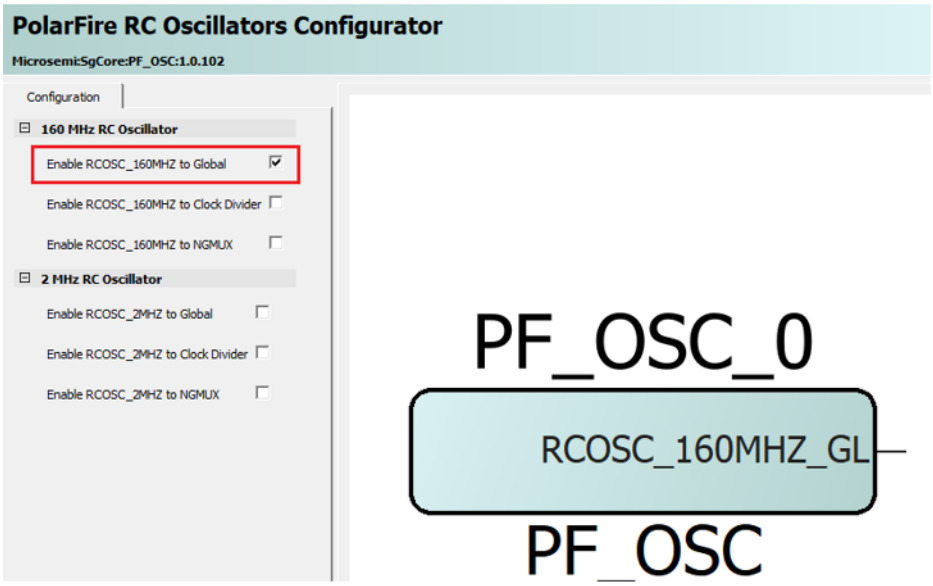
Figure 11 • Change Requested Frequency to 50 MHz



3.3 PolarFire RC Oscillator Configuration

To add the PolarFire RC Oscillator to the SmartDesign, select the **Catalog** and search for **OSC**. Double click on the **PolarFire RC Oscillator** to instantiate it in the SmartDesign. The default configuration should be used.

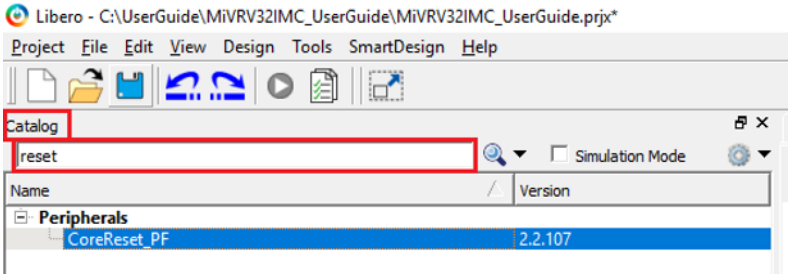
Figure 12 • Enable 160 MHz Oscillator



3.4 CoreRESET_PF Configuration

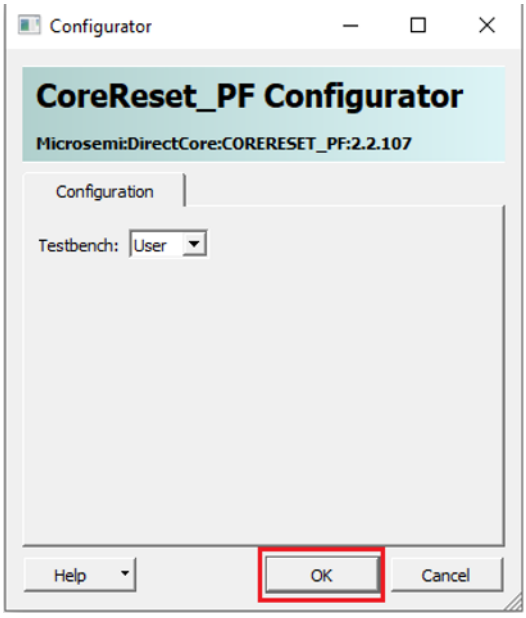
To add the CORERESET_PF to the SmartDesign, select the **Catalog** and search for **reset**. Double click on **CoreReset_PF** to instantiate it into the SmartDesign.

Figure 13 • Instantiating CoreRESET_PF



The default configuration for CoreRESET_PF can be used.

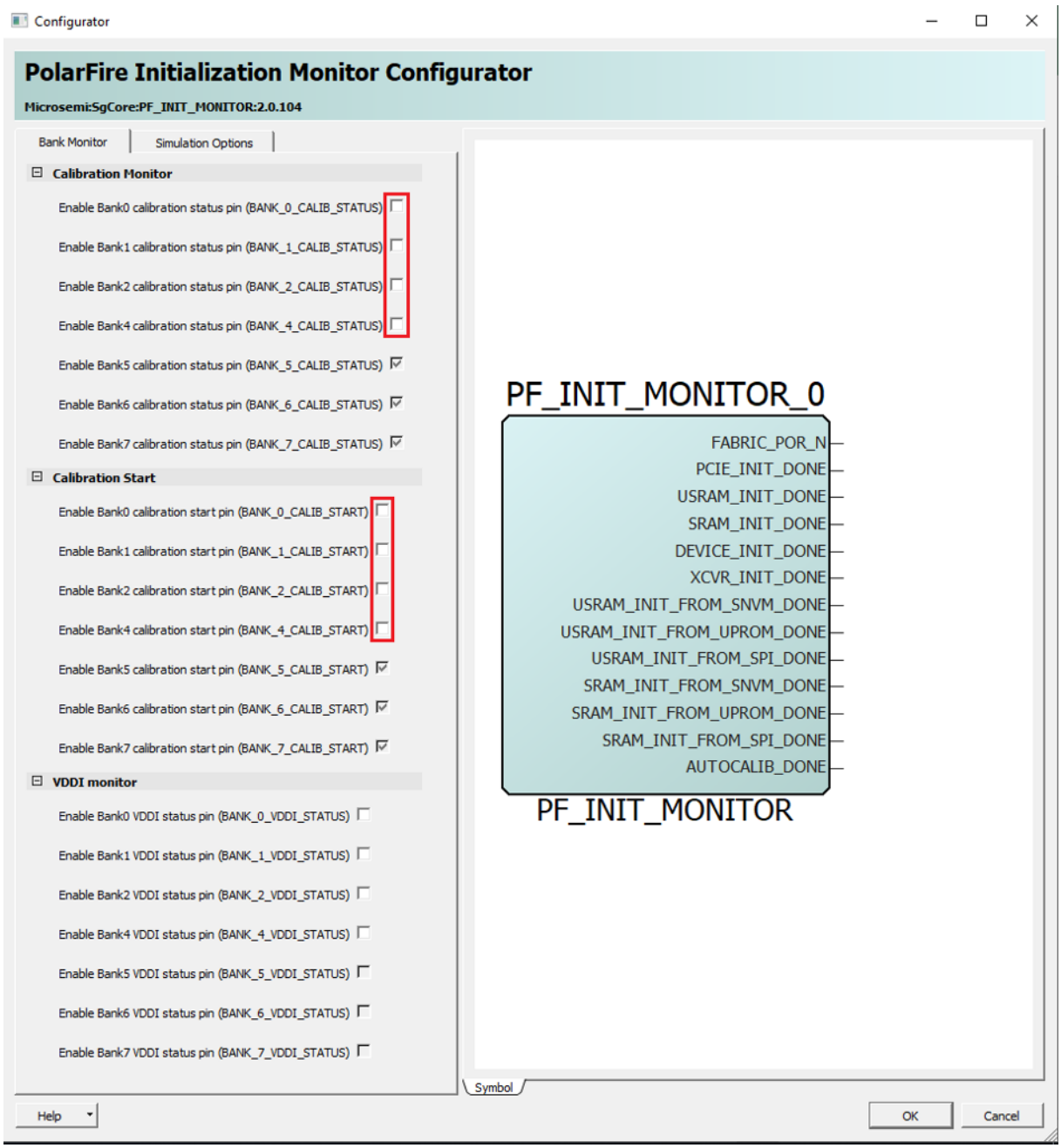
Figure 14 • CoreRESET_PF Configurator Default Settings



3.5 PolarFire Initialization Monitor Configuration

Search for monitor in the catalog and double click **PolarFire Initialisation Monitor** to instantiate in the SmartDesign. Uncheck the **Calibration Monitor** and **Calibration Start** boxes as shown in the following figure.

Figure 15 • PolarFire Initialization Monitor Configurator Required Settings

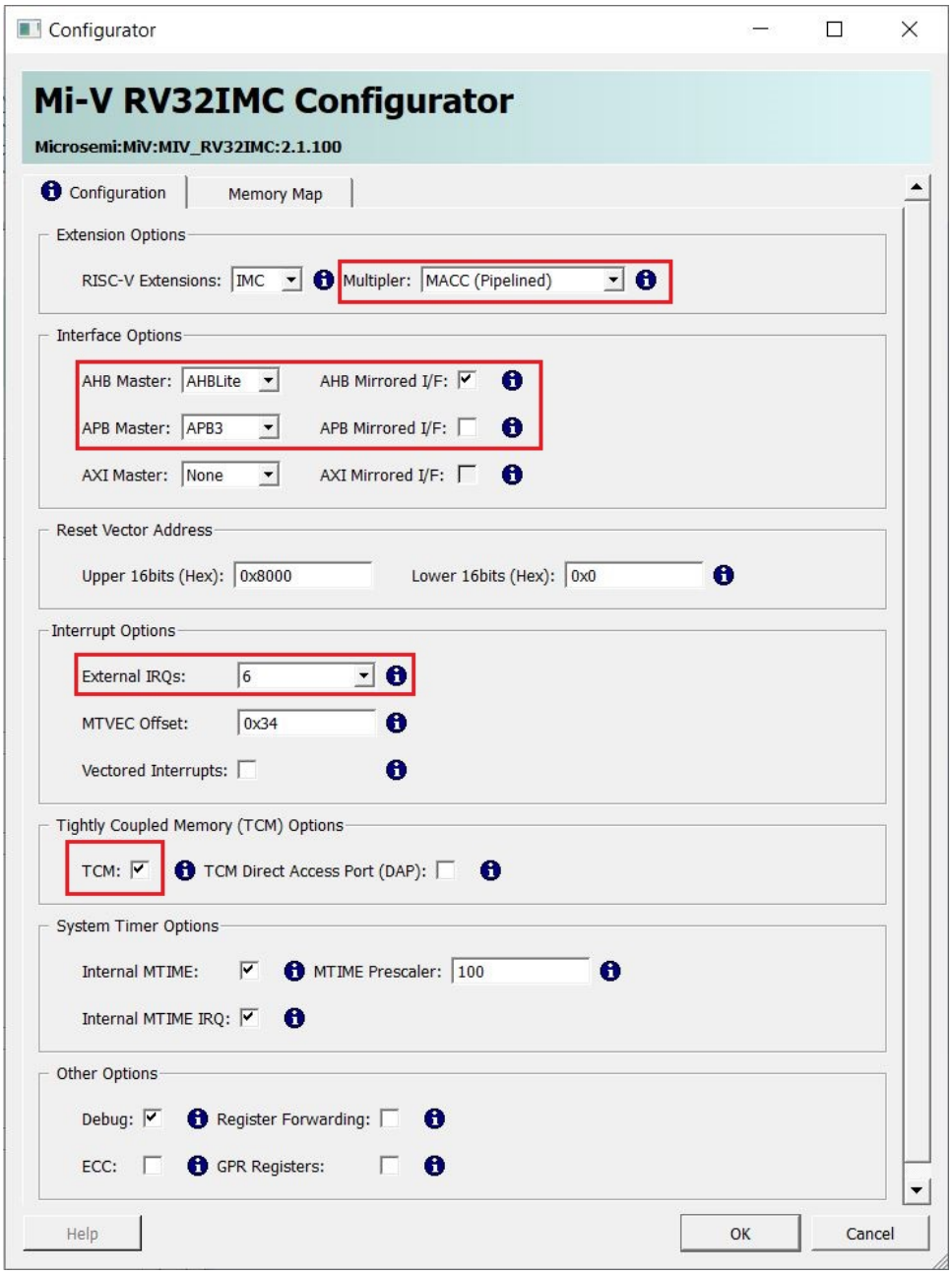


3.6 Mi-V RV32IMC Configuration

To add the Mi-V RV32IMC to the SmartDesign, search for RV32IMC in the catalog and double click to instantiate. The default configuration is shown in the following figure.

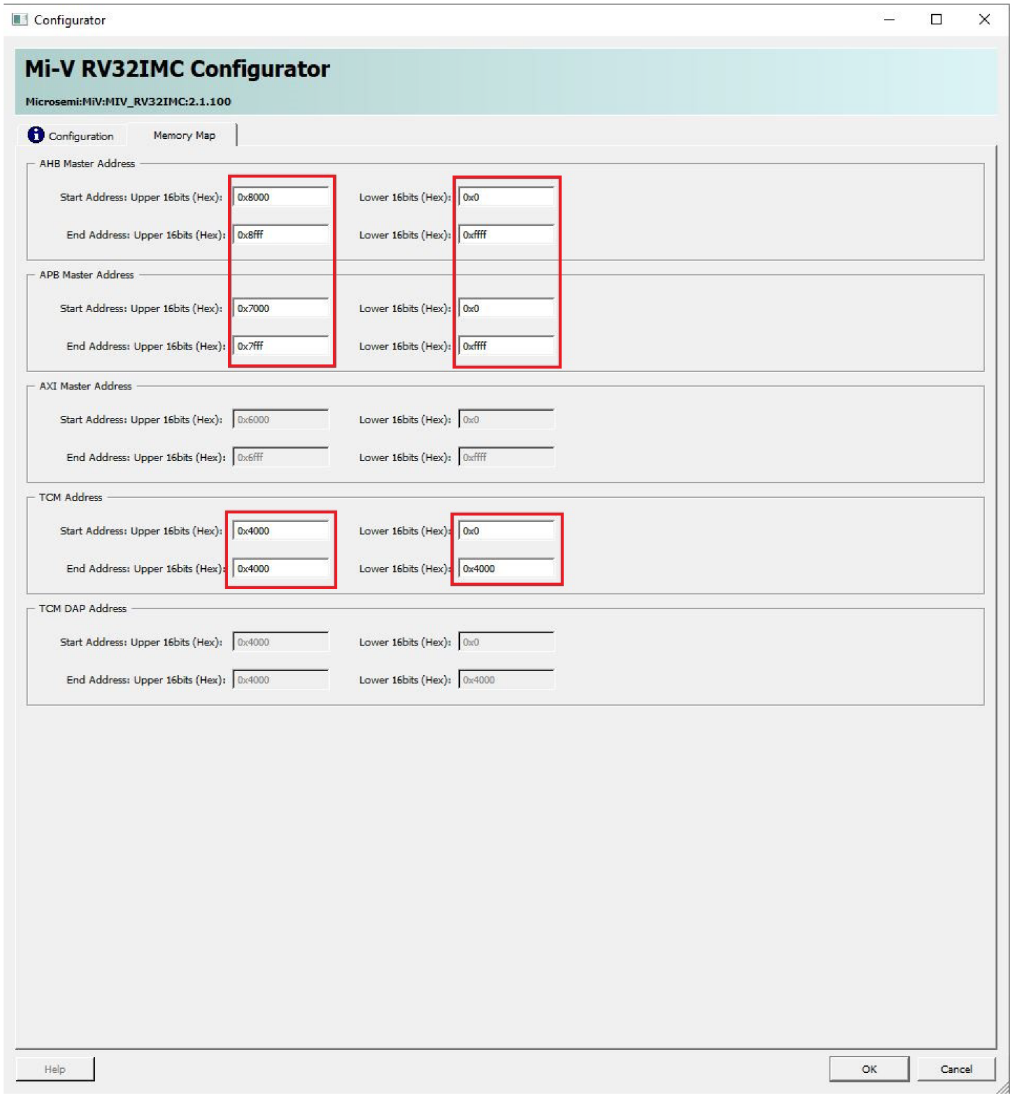
Figure 16 • Instantiate MIV_RV32IMC Core

Figure 17 • MiV RV32IMC Configurator Setup



The memory map is set to the following address range.

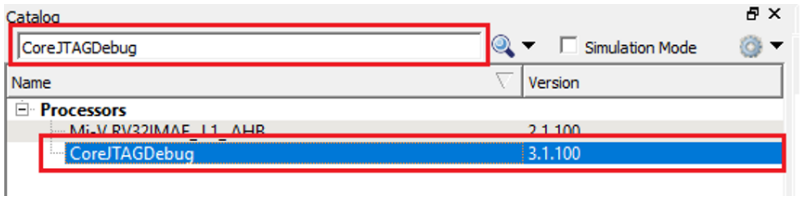
Figure 18 • Update the MiV_RV32IMC Memory Map



3.7 CoreJTAGDebug Configuration

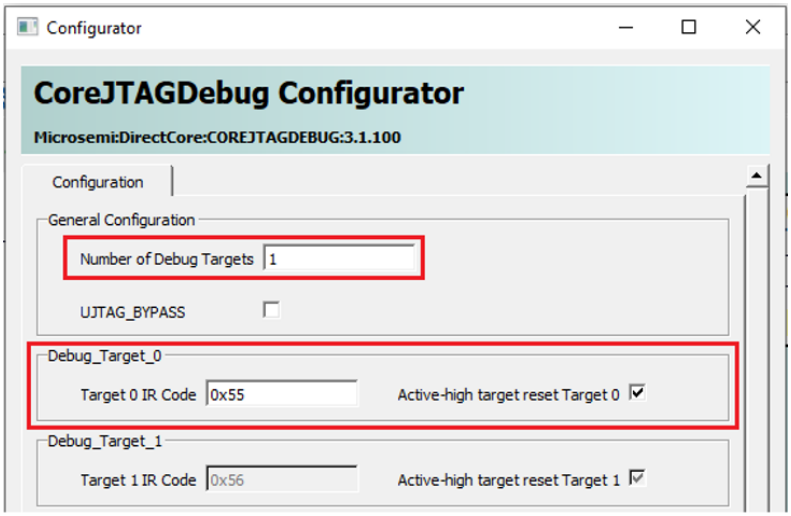
To add the CoreJTAGDebug block to the SmartDesign, search CoreJTAGDebug in the catalog and double click to instantiate.

Figure 19 • Instantiate CoreJTAGDebug



The default configuration for CoreJTAGDebug can be used.

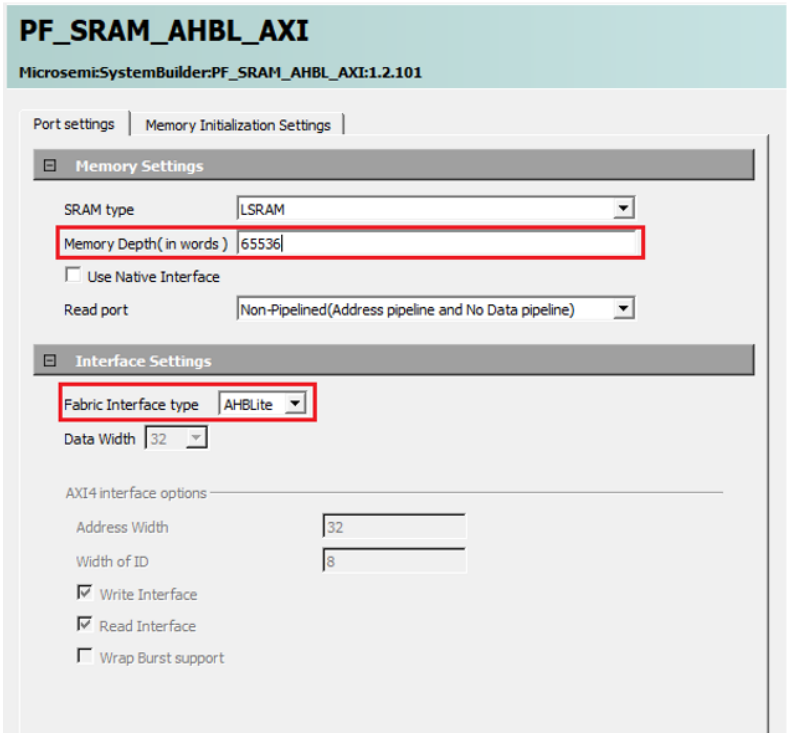
Figure 20 • CoreJTAGDebug Configurator Default Settings



3.8 Memory and Peripheral Configurations—PolarFire SRAM

To add the PolarFire SRAM to the SmartDesign, search for sram and select PolarFire SRAM (AHBLite and AXI). Under Memory Settings, configure **Memory Depth (in words)** to **65536**. Under Interface Settings, configure **Fabric Interface type** to **AHBLite**. The MiV RV32IMC core is connected directly to the PF_SRAM_AHBL_AXI block.

Figure 21 • PF_SRAM_AHBL_AXI Configuration



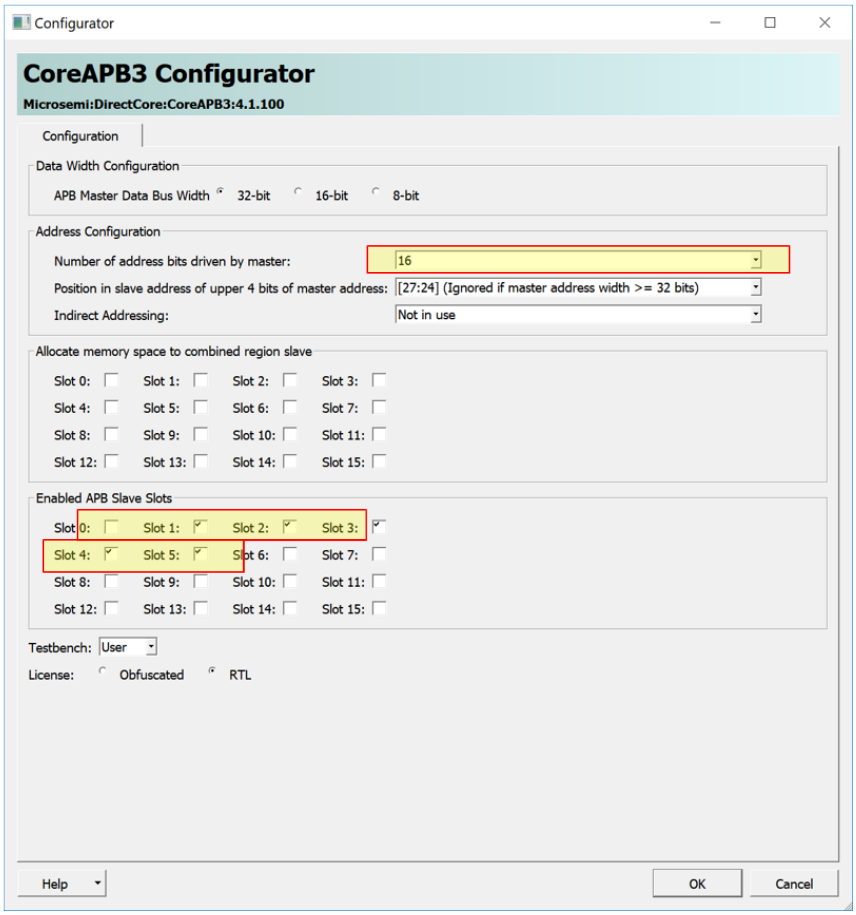
4 Peripheral Connections

The following section describes the peripheral configurations and connections for the design.

4.1 CoreAPB3 Configuration

To add the CoreAPB3 to the SmartDesign, search for coreapb3 and double-click to instantiate it in the SmartDesign. Configure the core to set the **Number of address bits driven by master** to **16** and under **Enable APB Slots** check **Slots 1, 2, 3, 4, and 5**. CoreAPB3 can be connected directly to the core.

Figure 22 • CoreAPB3 Required Configuration



4.2 CoreUARTapb Configuration

To add CoreUARTapb to the SmartDesign, search for CoreUARTapb in the catalog and double click to instantiate. The default name and configuration for CoreUARTapb can be used.

4.3 CoreTimer 0 Configuration

To add CoreTimer_0 to the SmartDesign search for CoreTimer in the catalog and double click to instantiate. The default name and configuration for CoreTimer can be used.

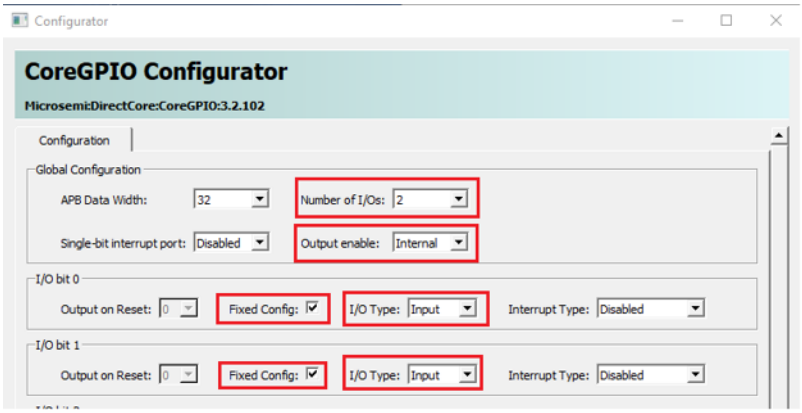
4.4 CoreTimer 1 Configuration

To add CoreTimer_1 to the SmartDesign search for CoreTimer in the catalog and double click to instantiate. The default name and configuration for CoreTimer can be used.

4.5 CoreGPIO Input Configuration

To add CoreGPIO to the SmartDesign search for CoreGPIO in the catalog and double click to instantiate. Enter the component name as CoreGPIO_IN. This CoreGPIO is configured to have two fixed configured inputs as shown in the following figure.

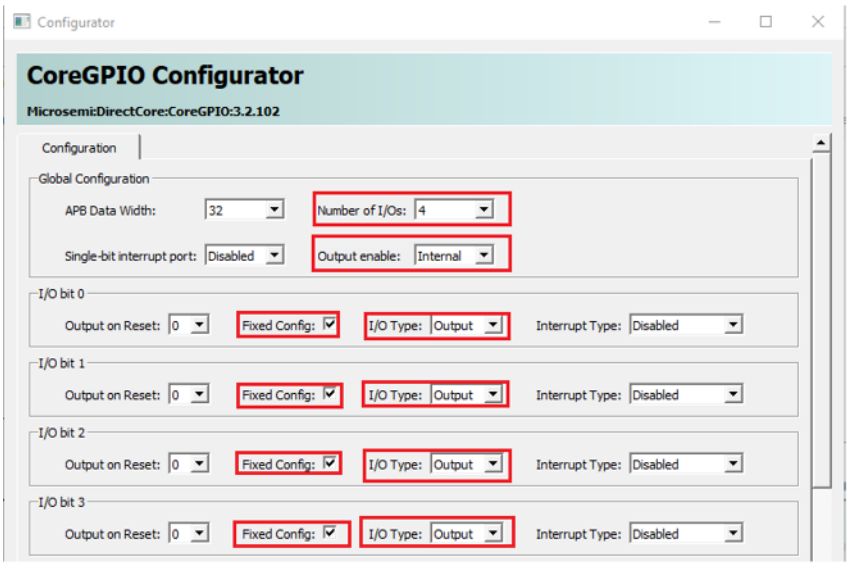
Figure 23 • CoreGPIO_IN Configuration



4.6 CoreGPIO Output Configuration

To add CoreGPIO to the SmartDesign search for CoreGPIO in the catalog and double click to instantiate. Enter the component name as CoreGPIO_OUT. This CoreGPIO is configured to have four fixed configured outputs as shown in the following figure.

Figure 24 • CoreGPIO_OUT Configuration



4.7 Component Connections List

The following tables list each component and each component it is connected to.

Table 1 • PF_OSC_C0_0

Connect From	Connect To
PF_OSC_C0_0:RCOSC_160MHZ_GL	PF_CCC_C0_0:REF_CLK_0

Table 2 • PF_CCC_C0_0

Connect From	Connect To
PF_OSC_C0_0:RCOSC_160MHZ_GL	CORERESET_PF_C0_0:CLK
	MIV_RV32IMC_C0_0:clk
	PF_SRAM_AHBL_AXI_C0_0:HCLK
	COREUARTAPB_C0_0:PCLK
	CORETIMER_C0_0:PCLK
	CORETIMER_C1_0:PCLK
	COREGPIO_IN_0:PCLK
PF_CCC_C0_0:PLL_LOCK_0	CORERESET_PF_C0_0:PLL_LOCK

Table 3 • PolarFire Initialization Monitor

Connect From	Connect To
PF_INIT_MONITOR_C0_0:DEVICE_INIT_DONE	CORERESET_PF_C0_0:INIT_DONE

Table 4 • MiV_RV32IMC

Connect From	Connect To
MIV_RV32IMC_C0_0:JTAG_TCK	COREJTAGDEBUG_C0_0:TGT_TCK_0
MIV_RV32IMC_C0_0:JTAG_TDI	COREJTAGDEBUG_C0_0:TGT_TDI_0
MIV_RV32IMC_C0_0:JTAG_TDO	COREJTAGDEBUG_C0_0:TGT_TDO_0
MIV_RV32IMC_C0_0:JTAG_TMS	COREJTAGDEBUG_C0_0:TGT_TMS_0
MIV_RV32IMC_C0_0:JTAG_TRSTN	COREJTAGDEBUG_C0_0:TGT_TRSTN_0
MIV_RV32IMC_C0_0:EXT_RESETN	Mark unused
MIV_RV32IMC_C0_0:JTAG_TDO_DR	Mark unused
MIV_RV32IMC_C0_0:EXT_IRQ	CORETIMER_C1_0:TIMINT
MIV_RV32IMC_C0_0:EXT_SYS_IRQ[5:1]	Tie low

Connect From	Connect To
MIV_RV32IMC_C0_0:EXT_SYS_IRQ[0]	CORETIMER_C0_0:TIMINT
MIV_RV32IMC_C0_0:TIME_COUNT_OUT[63:0]	Mark unused
MIV_RV32IMC_C0_0:AHBL_M_SLV	PF_SRAM_AHBL_AXI_C0_0:AHBSLAVEINTERFACE
MIV_RV32IMC_C0_0:APB_MSTR	COREAPB3_C0_0:APB3MASTER

Table 5 • CoreRESET_PF

Connect From	Connect To
CORERESET_PF_C0_0:EXT_RSTN	Promote to top level and rename to DEVRST_N. To rename the input port EXT_RSTN, right-click the port and select Modify/R-ename, a text box will open. Enter DEVRST_N and select OK.
CORERESET_PF_C0_0:BANK_x_VDDI_STATUS	Tie high
CORERESET_PF_C0_0:BANK_y_VDDI_STATUS	Tie high
CORERESET_PF_C0_0:SS_BUSY	Tie low
CORERESET_PF_C0_0:FF_US_RESTORE	Tie low
CORERESET_PF_C0_0:FPGA_POR_N	Tie low
CORERESET_PF_C0_0:PLL_POWERDOWN_B	Mark unused
CORERESET_PF_C0_0:FABRIC_RESET_N	MIV_RV32IMC_C0_0:RESETN
	PF_SRAM_AHBL_AXI_C0_0:HRESETN
	COREUARTAPB_C0_0:PRESETN
	CORETIMER_C0_0:PRESETN
	CORETIMER_C1_0:PRESETN
	COREGPIO_IN_0:PRESETN
	COREGPIO_OUT_0:PRESETN

Table 6 • CoreAPB3_C0_0

Connect From	Connect To
COREAPB3_C0_0:APBMSLAVE1	COREUARTAPB_C0_0:APB_BIF
COREAPB3_C0_0:APBMSLAVE2	GOREGPIO_IN_0:APB_BIF
COREAPB3_C0_0:APBMSLAVE3	CORETIMER_C0_0:APBSLAVE
COREAPB3_C0_0:APBMSLAVE4	CORETIMER_C1_0:APBSLAVE
COREAPB3_C0_0:APBMSLAVE5	COREGPIO_OUT_0:APB_BIF

Table 7 • CoreUARTapb_C0_0

Connect From	Connect To
COREUARTAPB_C0_0:RX	Promote to top level
COREUARTAPB_C0_0:TX	Promote to top level
COREUARTAPB_C0_0:TXRDY	Mark unused
COREUARTAPB_C0_0:RXRDY	Mark unused
COREUARTAPB_C0_0:PARITY_ERR	Mark unused
COREUARTAPB_C0_0:OVERFLOW	Mark unused
COREUARTAPB_C0_0:FRAMING_ERR	Mark unused

Table 8 • CoreGPIO_IN_0

Connect From	Connect To
COREGPIO_IN_0:GPIO_IN[1:0]	Promote to top level
COREGPIO_IN_0:INT[1:0]	Mark unused
COREGPIO_IN_0:GPIO_OUT[1:0]	Mark unused

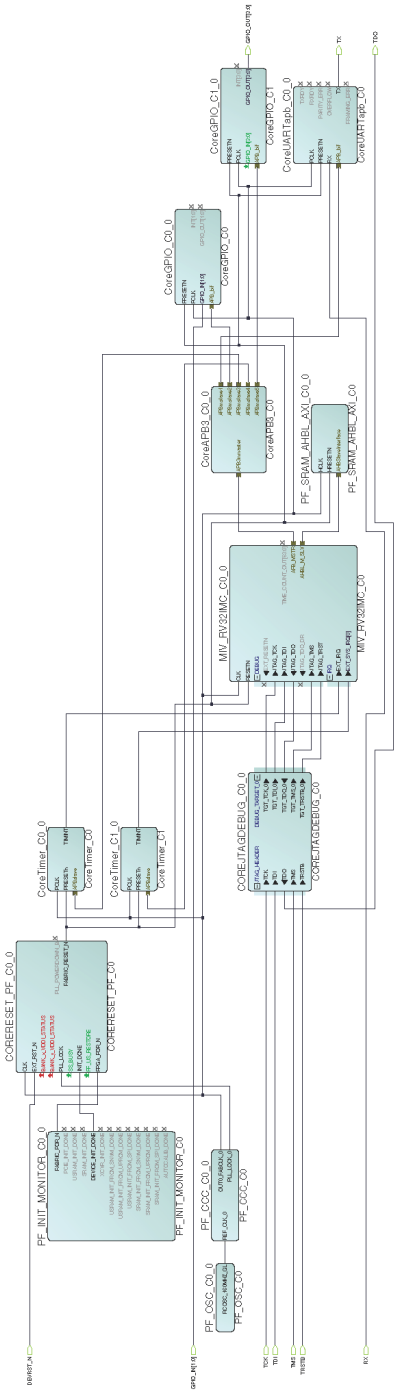
Table 9 • CoreGPIO_OUT_0

Connect From	Connect To
COREGPIO_OUT_0:GPIO_IN[3:0]	Tie low
COREGPIO_OUT_0:INT[3:0]	Mark unused
COREGPIO_OUT_0:GPIO_OUT[3:0]	Promote to top level

Table 10 • CoreJTAGDebug_C0_0

Connect From	Connect To
COREJTAGDEBUG_C0_0:TCK	Promote to top level
COREJTAGDEBUG_C0_0:TDI	Promote to top level
COREJTAGDEBUG_C0_0:TDO	Promote to top level
COREJTAGDEBUG_C0_0:TMS	Promote to top level
COREJTAGDEBUG_C0_0:TRSTB	Promote to top level

Figure 25 • Final Design



5 Design Constraints

Add the I/O constraints to the design flow tab and select Manage Constraints. Under the I/O Attributes tabs, add the following constraints by selecting New and naming the file io_constraints. Then, copy the constraints below into this file. Save the constraints file and select the check box under Place and Route. Save the I/O constraints.

```
# User Locked I/O Bank Settings

set_iobank -bank_name Bank5  \
    -vcci 3.30               \
    -fixed true              \
    -update_iostd true

# User Locked I/O settings

set_io -port_name {GPIO_IN[0]}  \
    -pin_name E13                \
    -fixed true                  \
    -DIRECTION INPUT

set_io -port_name {GPIO_IN[1]}  \
    -pin_name E14                 \
    -fixed true                  \
    -DIRECTION INPUT

set_io -port_name {GPIO_OUT[0]}  \
    -pin_name D7                 \
    -fixed true                  \
    -DIRECTION OUTPUT

set_io -port_name {GPIO_OUT[1]}  \
    -pin_name D8                 \
    -fixed true                  \
    -DIRECTION OUTPUT

set_io -port_name {GPIO_OUT[2]}  \
    -pin_name D9                 \
    -fixed true                  \
    -DIRECTION OUTPUT

set_io -port_name {GPIO_OUT[3]}  \
    -pin_name D6                 \
    -fixed true                  \
    -DIRECTION OUTPUT

set_io -port_name RX  \
    -pin_name F16     \
    -fixed true       \
    -DIRECTION INPUT

set_io -port_name TX   \
    -pin_name F17      \
    -fixed true        \
    -DIRECTION OUTPUT

set_io -port_name DEVRST_N  \
    -pin_name F5          \
    -fixed true           \
    -DIRECTION INPUT
```

Add the Timing constraints to the design flow tab and select Manage Constraints. Under the Timing tab, add the following constraints by selecting New and naming the file io_jtag_constraints. Then, copy the constraints below into this file.

```
#Constraining the JTAG clock to 6 MHz
create_clock -name {TCK} -period 166.67 -waveform {0 83.33} [ get_ports { TCK } ]
set_clock_groups -name {async1} -asynchronous -group [ get_clocks {
CCC_0_inst_0/CCC_0_0/pll_inst_0/OUT0 } ] -group [ get_clocks { TCK } ]
```

Save the constraints file, and associate it with Synthesis, Place and Route and Timing Verification by selecting each of the check boxes. It is also important to add in the derived constraints. This is done by selecting Derive Constraints in the Timing tab and excepting the default settings. Save the Timing Constraints.

6 Configure Design Initialization Data and Memories

The PolarFire LSRAM can be configured to boot a program from power-on reset. There are two ways to configure this:

- Configure the LSRAM memory via the LSRAM configurator.
- Configure the LSRAM memory via the Configure Design Initialisation Data and Memories configurator.

6.1 Configure LSRAM Memory via LSRAM Configurator

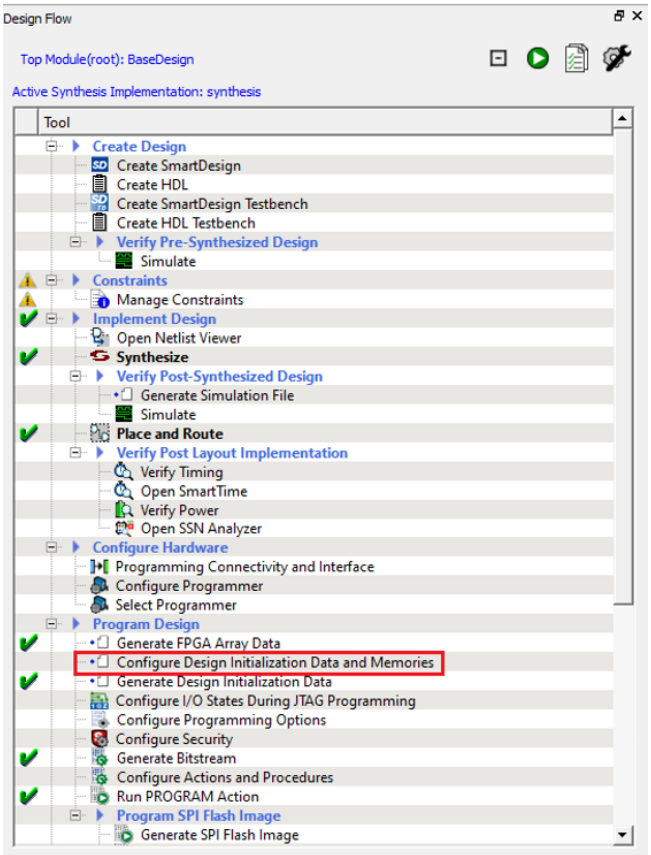
Open the SmartDesign, BaseDesign if closed. Locate the PF_SRAM_AHBL_AXI_C0_0.

1. Right-click on the core and select **Configure**.
2. Select the Memory Initialisation Settings tab.
3. Select the **Initialise RAM at Power up** check box.
4. Select the hex file created by SoftConsole when building a project (ensure the first line of this hex project is removed).
5. In the Design Flow tab, select **Generate bitstream**. Once this has completed, the programming file is ready to download to the FPGA board.
6. Connect the power cable to your FPGA board.
7. Connect the USB cable to the Embedded FlashPro5 micro USB port and to the USB port of your computer.
8. In the Design Flow tab, select **Run Program Action** to download the programming file to the FPGA.

6.2 Configure LSRAM Memory via Configure Design Initialization Data and Memories Configurator

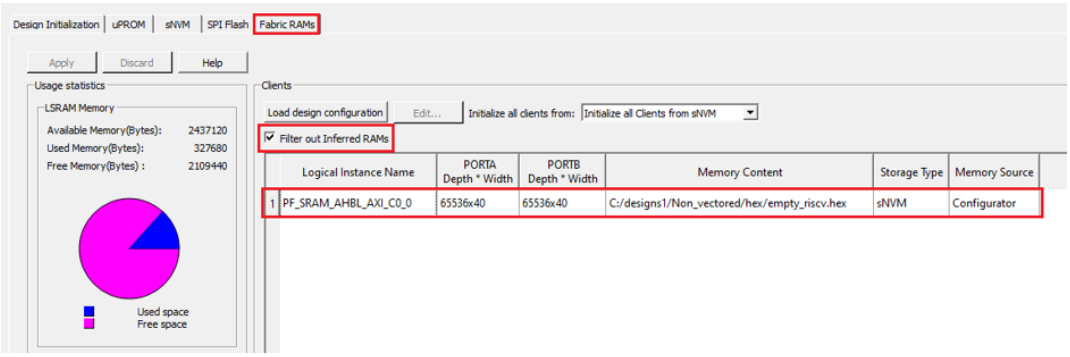
If you have run through the bitstream generation and you'd like to add or change the firmware in the LSRAM, this can be done via the Configure Design Initialization Data and Memories configurator.

Figure 26 • Design Flow Progression



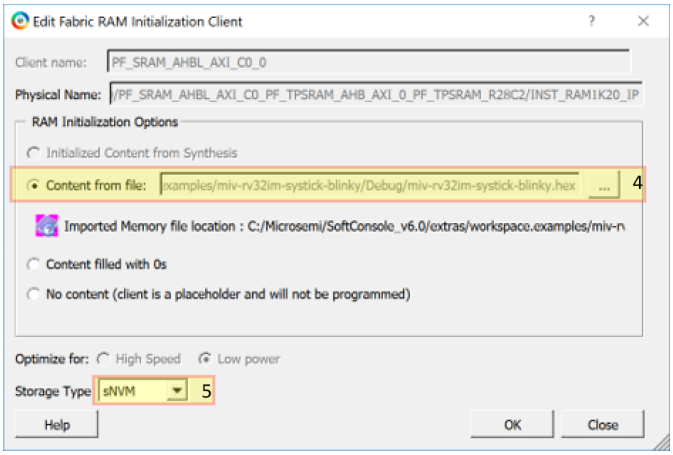
1. Run the flow so that the "Generate FPGA Array Data" step in the design flow is completed.
2. Select the Fabric RAMs tab.
3. Select Filter out Inferred RAM.
4. Double-click on the RAM to be initialized to open its configurator.

Figure 27 • Design Initialization Data and Memories Configurator



5. Under **Ram Initialization Options**, select **Content from file** and select the hex file created by SoftConsole when building a project (ensure the first line of this hex file is removed).
6. Select **Storage Type** as **sNVM**.

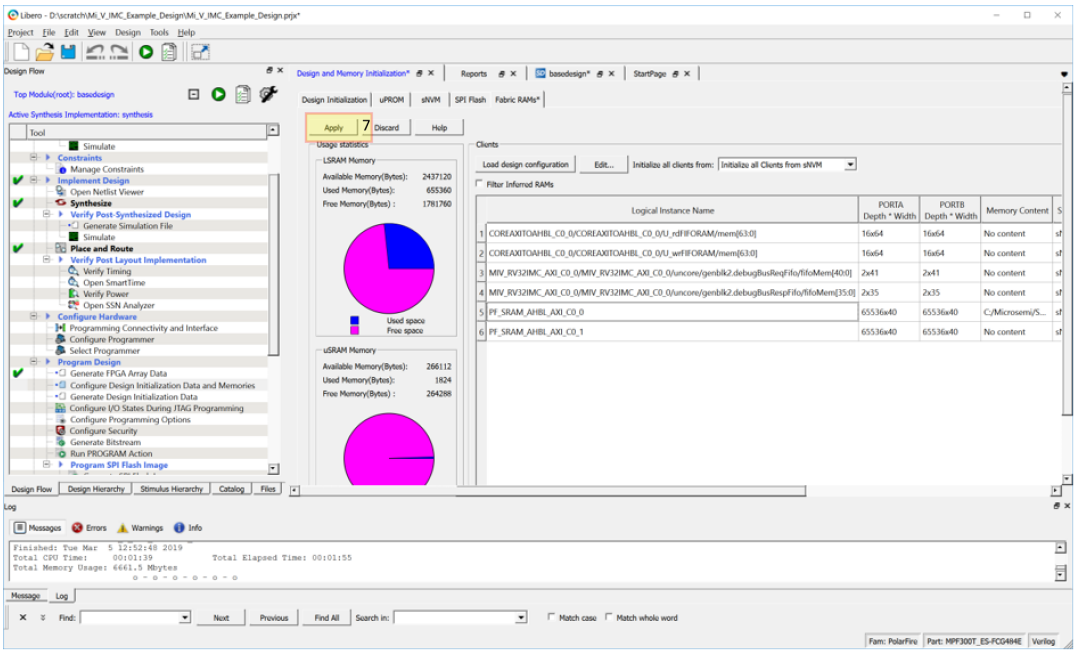
Figure 28 • Fabric RAM Initialization Configurator



7. Click **OK** and return to Libero.

8. Click **Apply** in the Fabric RAMs tab.

Figure 29 • Design Initialization Data and Memories Configurator



9. In the Design Flow tab, select **Generate bitstream**. Once this has completed, the programming file is ready to download to the FPGA board.

10. Connect the power cable to your FPGA board.

11. Connect the USB cable to the Embedded FlashPro5 USB port and to the USB port of your computer.

12. In the Design Flow tab, select **Run Program Action** to download the programming file to the FPGA.

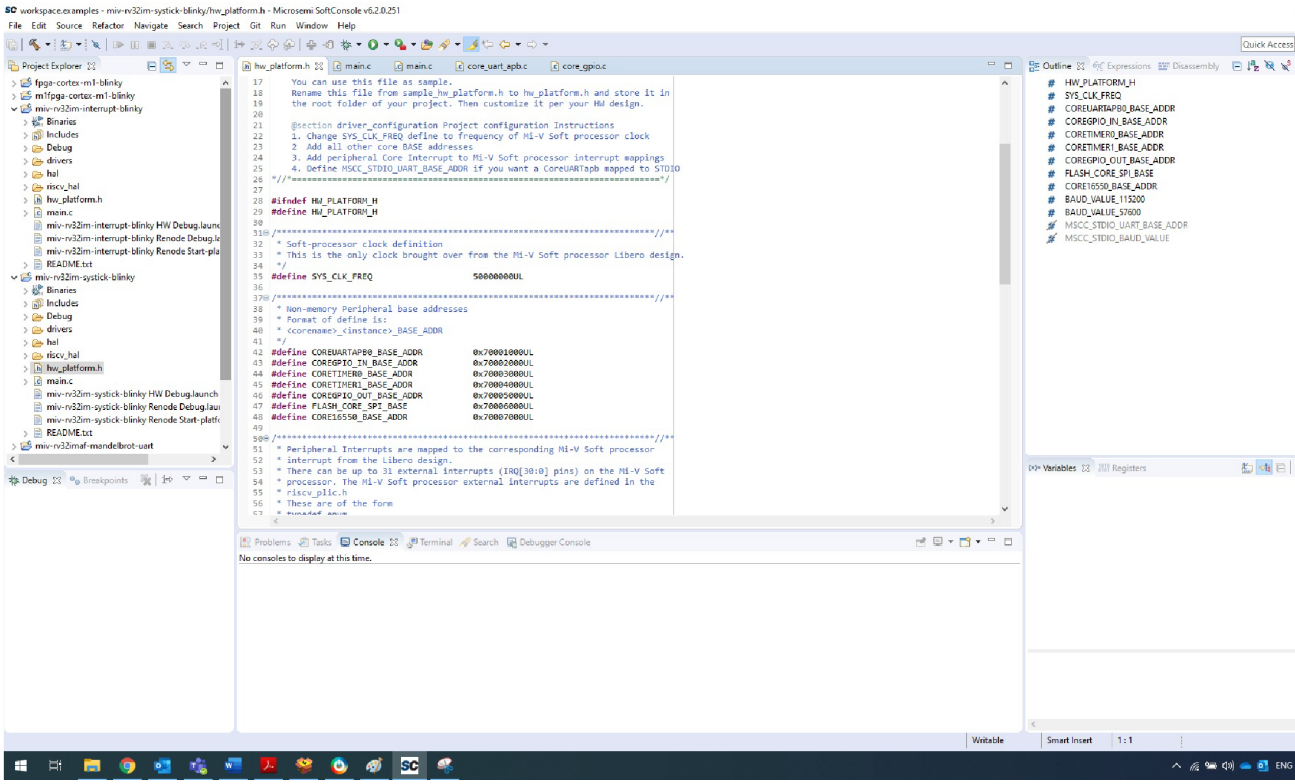
7 Run Project from SoftConsole

The following steps explain how to build and run a project in SoftConsole for the MIV_RV32IMC core. Open SoftConsole and the example project X.

7.1 Set System Clock Frequency and Peripheral Base Addresses

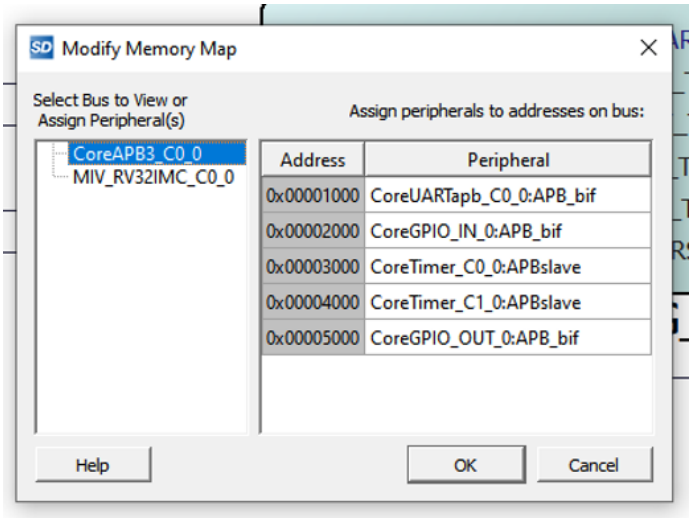
If UART is being used, the system clock frequency needs to be provided to the software and is done in the “hw_platform.h” file by changing the #define SYS_CLK_FREQ to the clock frequency (note this value must be in hertz).

Figure 30 • Set System Clock Frequency and Peripheral Base Addresses



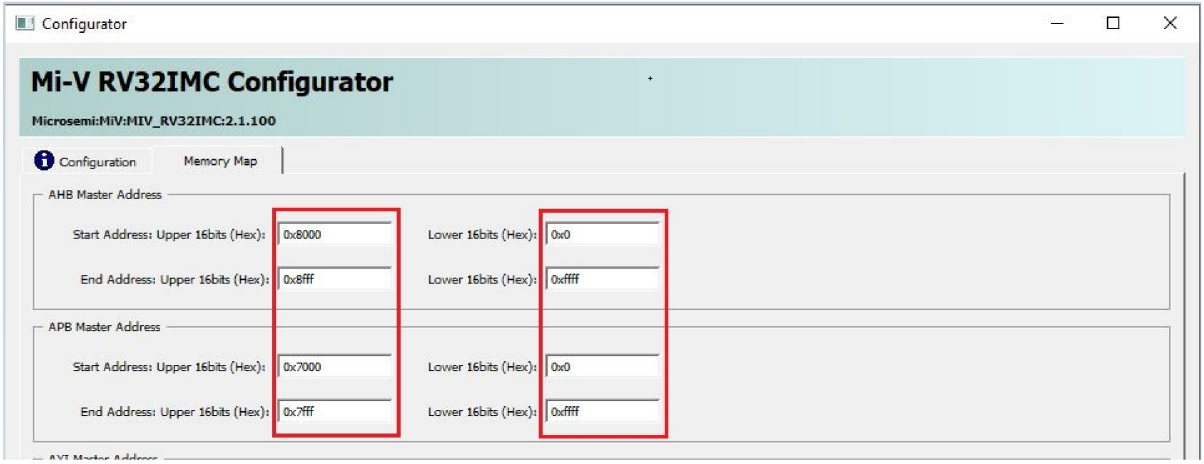
The “hw_platform.h” file is also used to set the base address for peripherals, the base address of a peripheral can be found in the project memory map generated by Libero.

Figure 31 • Modify Memory Map Dialog



The peripheral address in the hw_platform.h file must match the address in Libero for the peripheral to function correctly. These peripherals are addressed for 0x0 because the core redirects these addresses accordingly. Configuration settings for peripherals are displayed in the following figure.

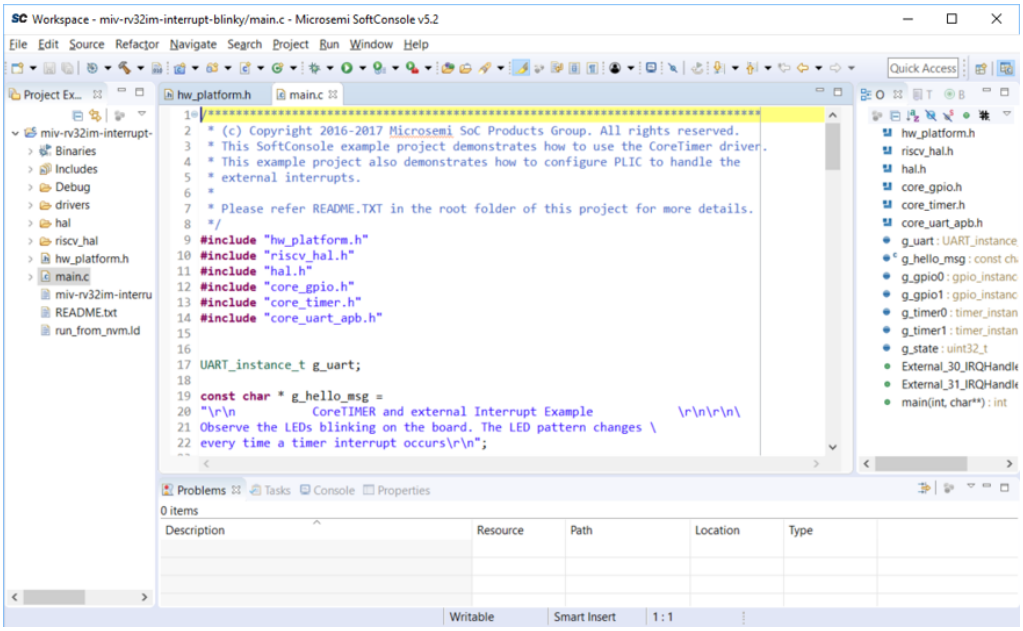
Figure 32 • RV32IMC Configurator Memory Map



7.2 Build and Debug a Project

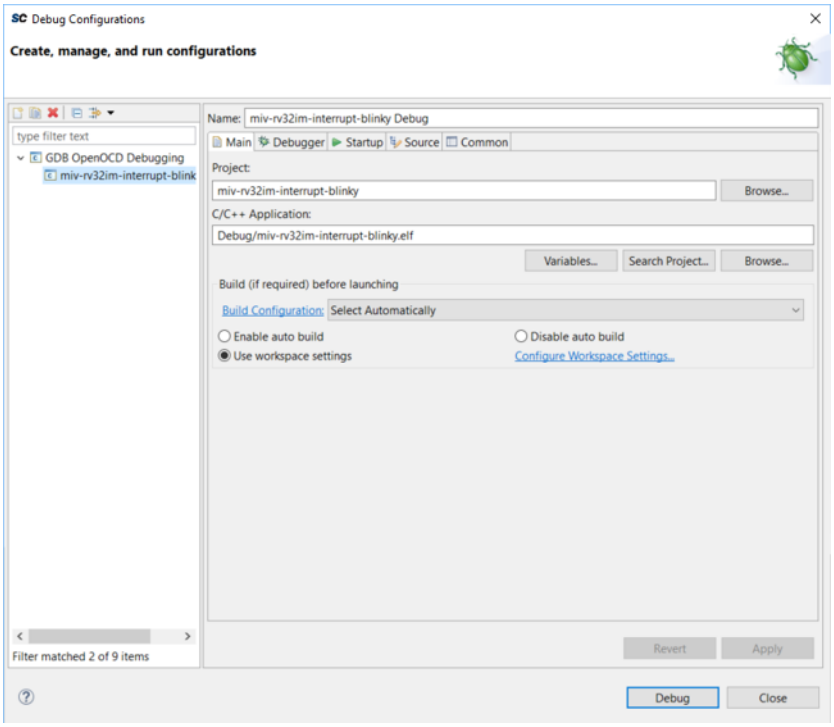
When you're ready to build your project, click the **Build** icon. In this case, as shown in the following figure, the project is being built for debug.

Figure 33 • Build and Debug a Project



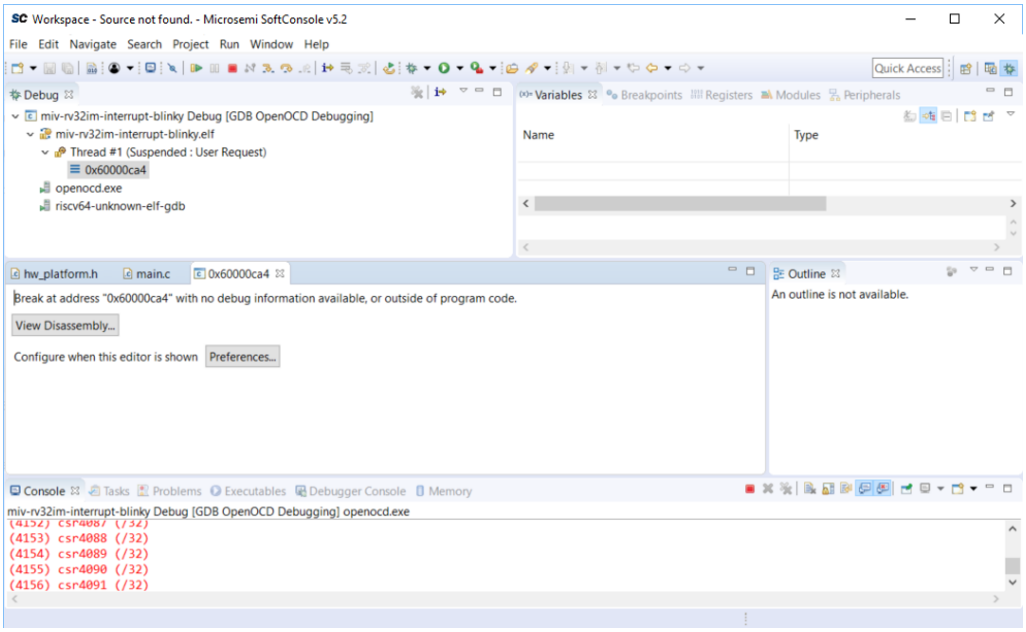
Once you have an error-free build, you can test it on your device. Make sure your device is connected and powered on, then click the **Debug** icon. If this is your first time debugging this project, you will have to select a debug configuration.

Figure 34 • Select a Debug Configuration



Once you have selected a debug configuration, click **Debug**. This will download the program to your device and take you to the debug perspective. The program will launch and halt on main. It can be run by clicking **Resume** or pressing **F8**.

Figure 35 • Run a Program



To add breakpoints and step through the code, ensure you are in the main.c tab.

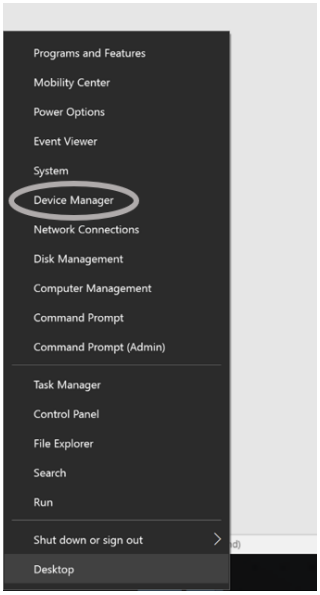
7.3 Communicate via UART

The following are needed to communicate via UART:

- The COM port your is using and its baud rate
- A serial communication client (the built-in terminal in SoftConsole is used in this example)

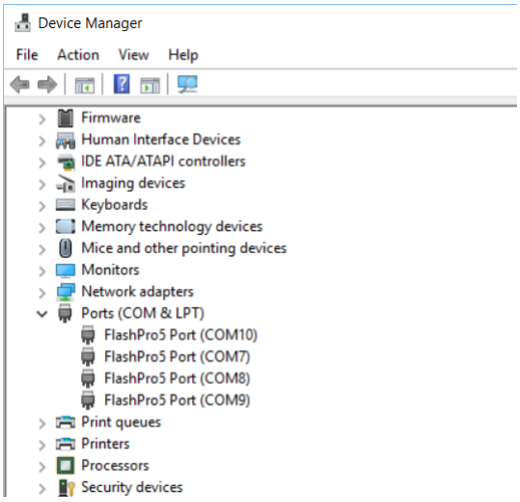
Device Manager can be used to find the COM port of your device. To open it, right-click on **Start** and select **Device Manager**.

Figure 36 • Open Device Manager



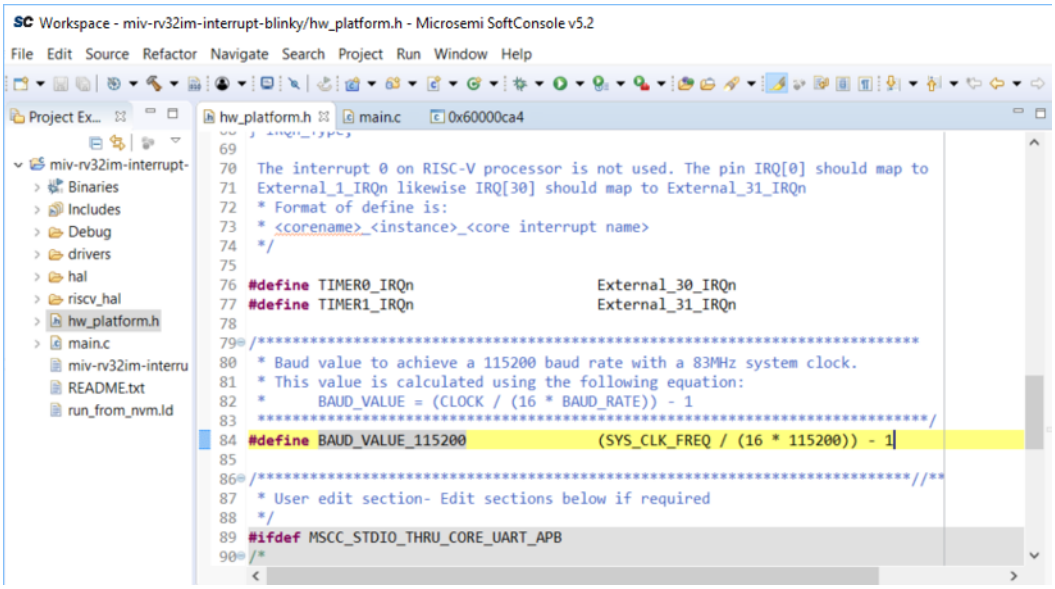
The COM port for your device will be one of those listed under **Ports (COM & LPT)**. It may not be clear which one is your device and may require trial and error to find the correct port.

Figure 37 • Find COM Ports in Device Manager



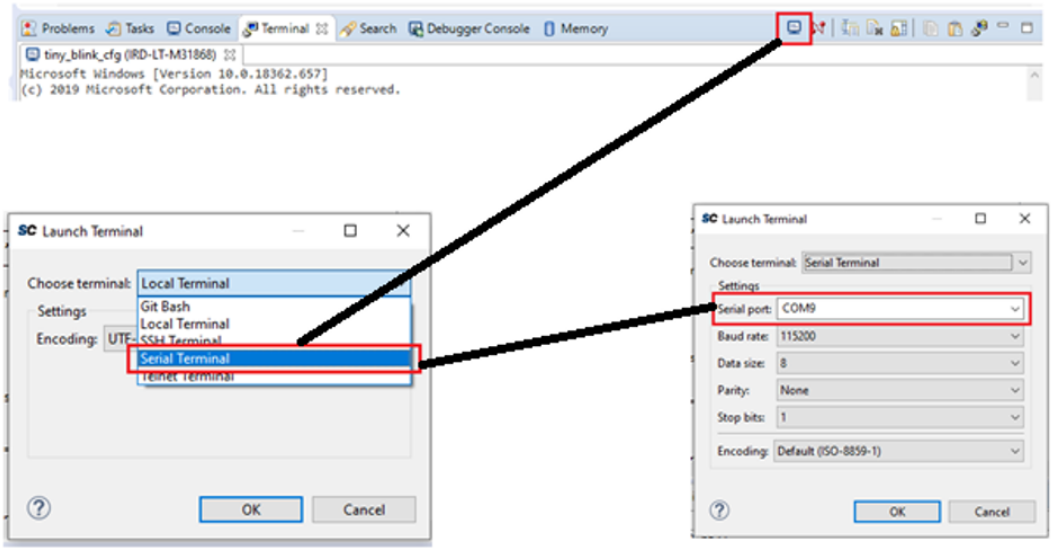
The baud rate for the UART connection can be found in the hw_platform.h file within the SoftConsole project

Figure 38 • Location of Baud Rate Value



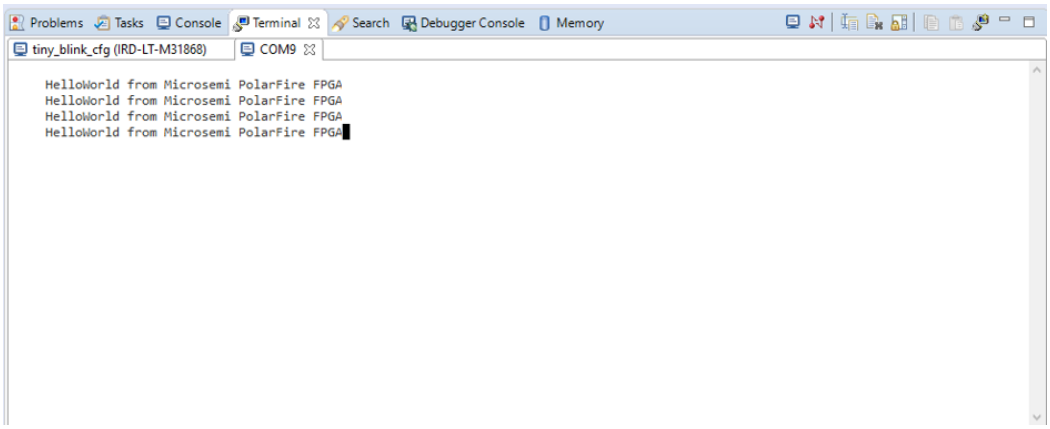
Open the serial communication client within SoftConsole by selecting **Window** > **Show View** > **Terminal**. Then, input the values for your COM port and baud rate.

Figure 39 • Serial Output from COM9 Terminal



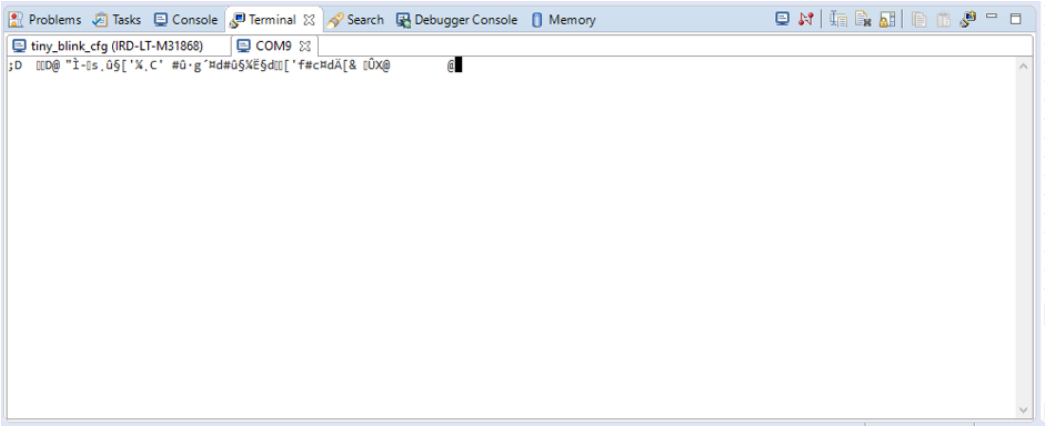
Run your program and any UART output will be displayed in the communication client's terminal.

Figure 40 • Output from SoftConsole Terminal



If the UART output isn't as expected (that is, the random/unknown characters are as shown in the following figure) it is more than likely an issue with the clock frequency in the hw_platform.h file and can be fixed by following the steps in this section.

Figure 41 • UART Clock Frequency Issues



Microsemi
2355 W. Chandler Blvd.
Chandler, AZ 85224 USA

Within the USA: +1 (480) 792-7200
Fax: +1 (480) 792-7277

Microsemi Corporation, a subsidiary of Microchip Technology Inc. (Nasdaq: MCHP), and its corporate affiliates are leading providers of smart, connected and secure embedded control solutions. Their easy-to-use development tools and comprehensive product portfolio enable customers to create optimal designs which reduce risk while lowering total system cost and time to market. These solutions serve more than 120,000 customers across the industrial, automotive, consumer, aerospace and defense, communications and computing markets. Headquartered in Chandler, Arizona, the company offers outstanding technical support along with dependable delivery and quality. Learn more at **www.microsemi.com**.

50200909