Design Guide

Mi-V PolarFire Quick Start Design Guide

Final

October 2018

Microsemi

a Microchip company

Contents

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 October 2018. It was the first publication of this document.

2 Introduction

This guide will provide quick start steps in setting up the Mi-V AXI cores for use in PolarFire FPGA designs using Libero SoC PolarFire v2.1, generating a bitstream and downloading the FPGA data to a device and debugging, and using serial communication in SoftConsole v5.2. The design produced by following the steps outlined in this document will have the same function as those found on the Microsemi Github page and can be downloaded from there.

Any components needed for designs can be found in the Libero Catalog or on the Microsemi Github page. This guide will be covering multiple devices, and there will be some device specific components /configurations that will need to be done. To try and keep the setup as easy as possible, these device specific components and configurations will be dealt with separately at the start of the guide, and the remainder of the components and configurations will be generic across device families.

3 Creating the Libero Project

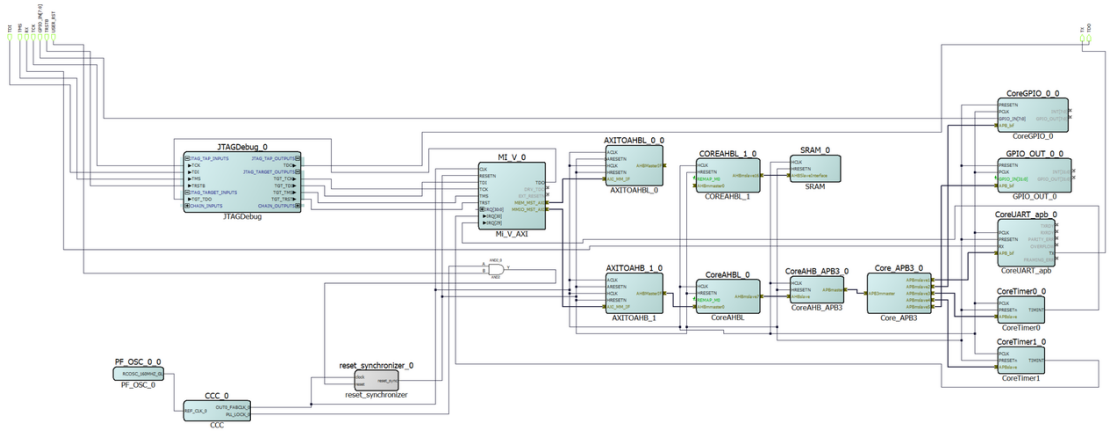
This section describes how to create the Libero project.

1. Open Libero and select New.
2. Enter a project name and project folder location and then click next (there will be several subdirectories, the path should be kept to a minimum to ensure path length remains within limits).
3. In the Device Selection page, select your device family. Use the search feature to locate your device and click finish.
4. If prompted whether you would like to use the enhanced constraints flow, select Use Enhanced Constraints Flow.
5. From the Design Flow, select Create SmartDesign and name your design.

3.1 Mi-V Placing and Configuring Components

The following graphic is the design that can be produced for the Mi-V in its AX13 configuration, using all of the component configurations outlined in this graphic.

Figure 1 • Finished Core Design Using the AXI3 Interface



The following table shows the components required to produce this design.

Table 1 • Table of Required Components

Component	Quantity	Version
CoreJTAGDebug	1	v2.0.100
MI_V_AXI	1	v2.0.100
CoreAXITOAHBL	2	v3.3.101
CoreAHBLite	2	v5.3.200
SRAM (also known as “PolarFire SRAM (AHBLite and AXI)”)	1	v1.1.101
PF_OSC (also known as “PolarFire RC Oscillators”)	1	v1.0.102
CCC	1	v1.0.112
AND2	1	v1.0
Reset synchronizer	1	Code included

Set up the components as follows:

1. COREJTAGDEBUG: no configuration required
2. Mi-V: choose the AXI3 configuration
3. And2: no configuration required
4. Reset synchronizer: code included

3.1.1 PF_CCC configuration

This section shows the PF_CCC configuration.

Figure 2 • CCC Default Settings:Clock Options PLL

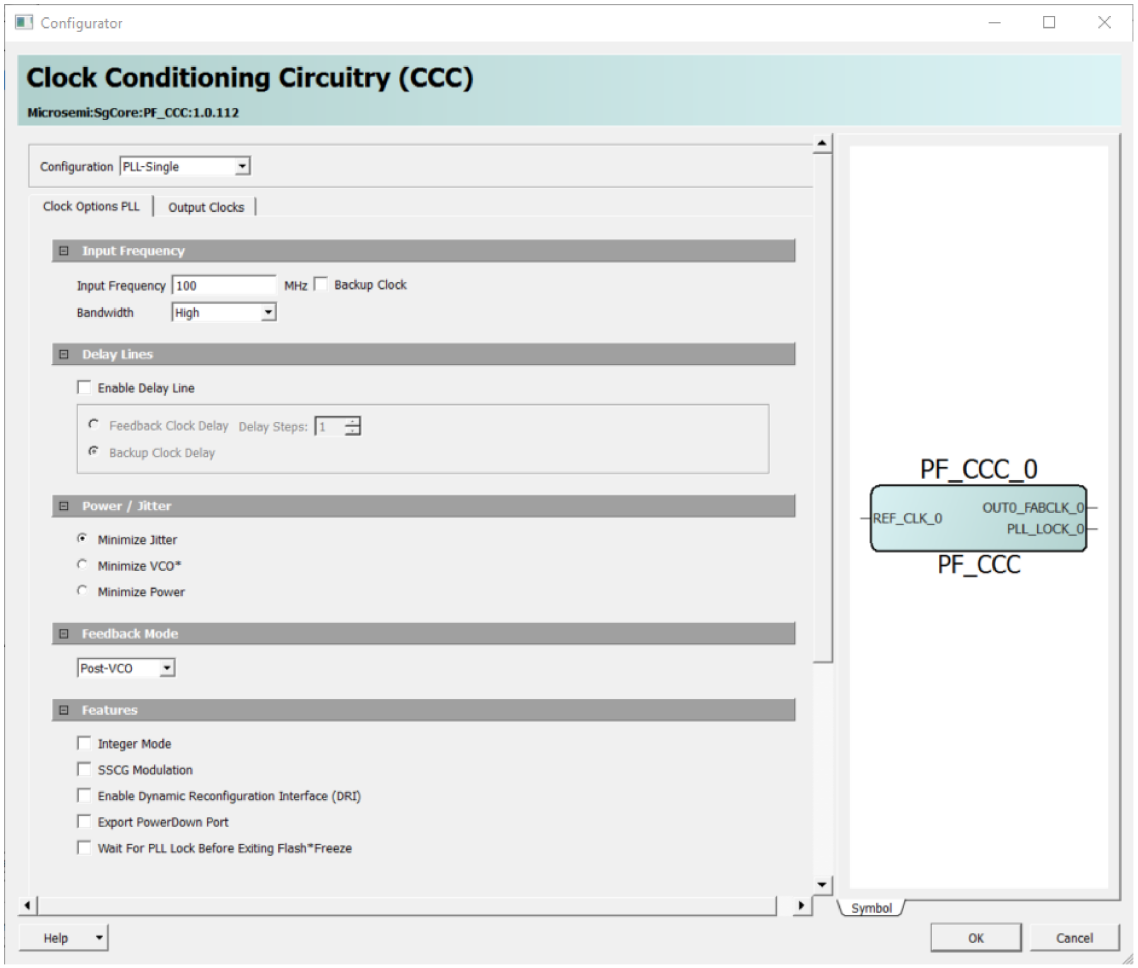


Figure 3 • CCC Required Settings-Clock Options PLL

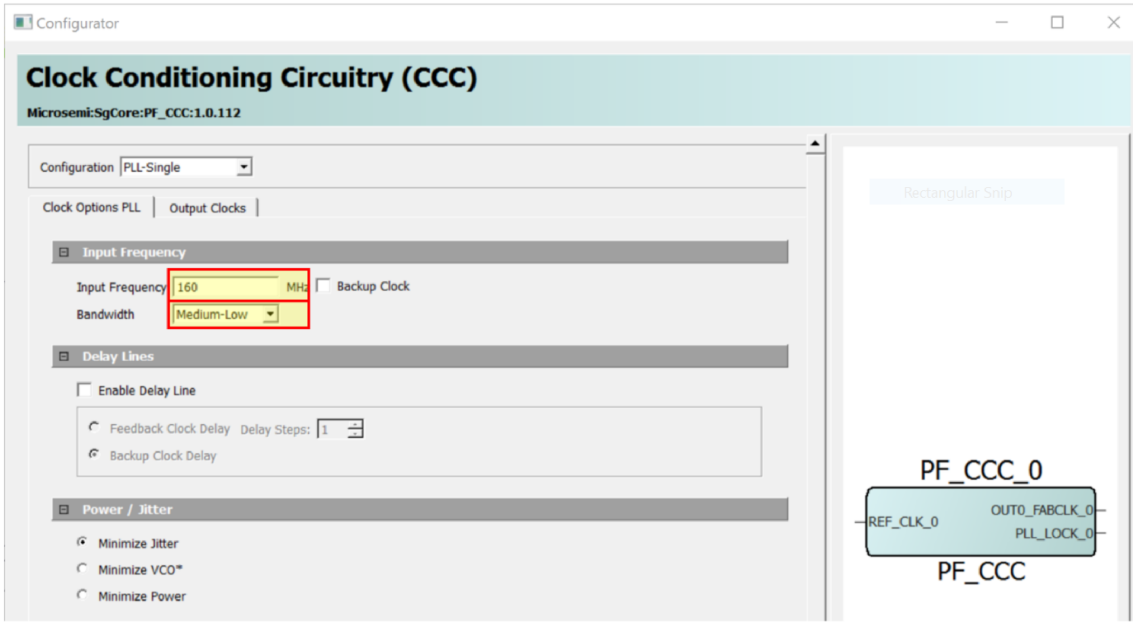


Figure 4 • CCC Default Settings-Output Clocks

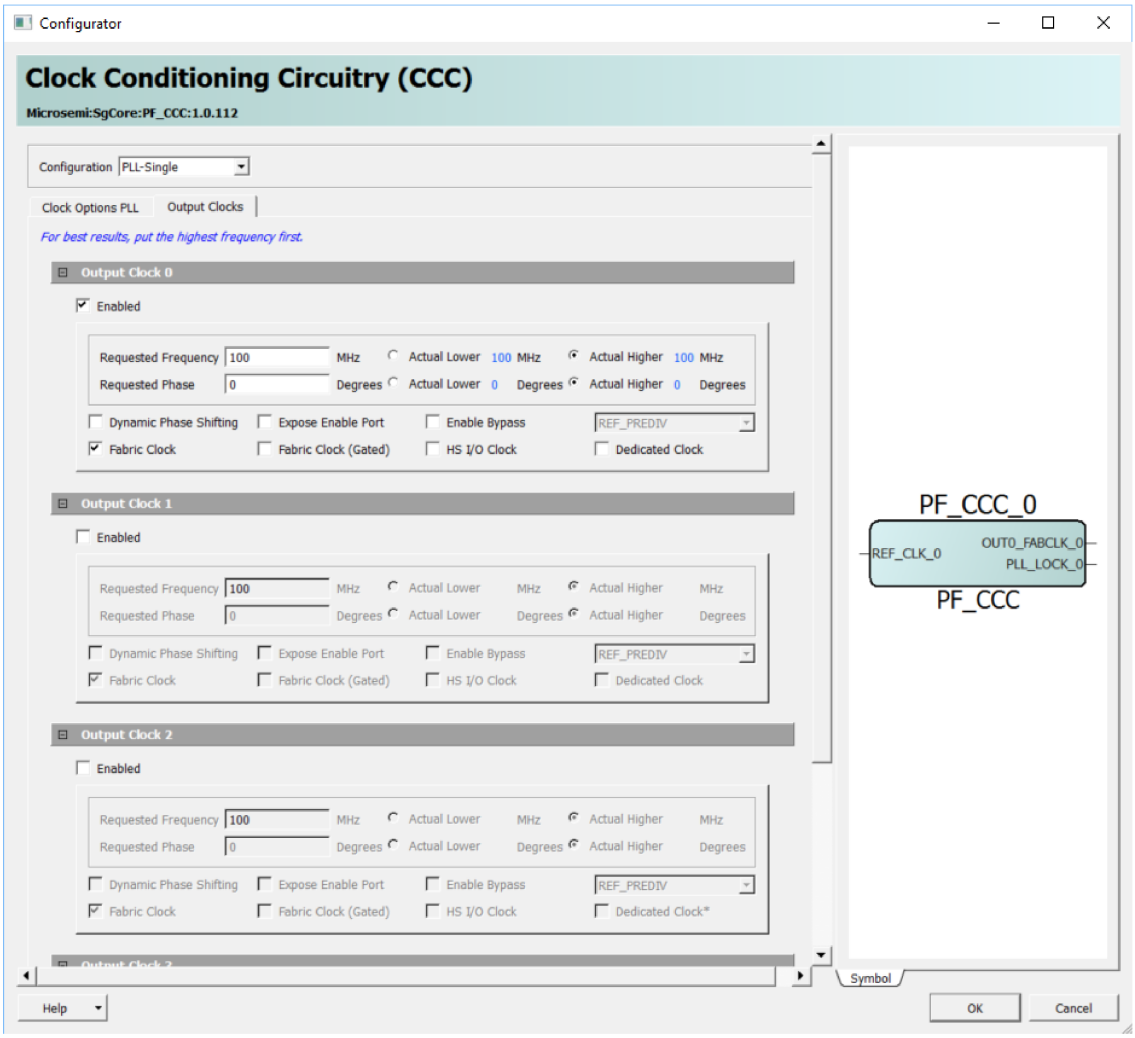
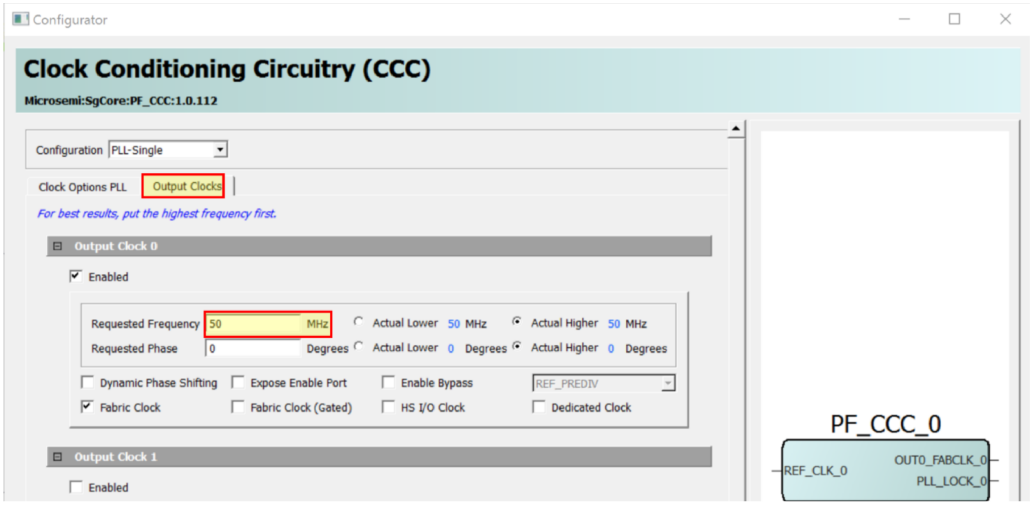


Figure 5 • CCC Required Settings-Output Clocks



3.1.2 Reset_synchronizer Configuration

The function of this synchronizer is to make the reset assertion and deassertion synchronous to the clock while guaranteeing that the reset will be asserted for one or more CLK cycles. This is done within the Mi-V CPU to ensure that it is registered by all sequential elements.

The reset synchronizer is not currently included as a core in the catalog, and because of this, the verilog code must be added manually. To do this:

1. Click File -> New -> HDL.
2. The HDL type is Verilog. Name the module and click OK.
3. The source editor will open with the default template. Replace all of the text in the source editor with the following:

```
module reset_synchronizer (

input clock,

input reset,

output reset_sync

);

reg [1:0] sync_deasert_reg;

reg [1:0] sync_asert_reg;

always @ (posedge clock or negedge reset)

begin

if (!reset)

begin

sync_deasert_reg[1:0] <= 2'b00;

end

else

begin

sync_deasert_reg[1:0] <= {sync_deasert_reg[0], 1'b1};

end

end

always @ (posedge clock)

begin

sync_asert_reg[1:0] <= {sync_asert_reg[0], sync_deasert_reg[1]};

end
```

```
assign reset_sync = sync_asert_reg[1];

endmodule
```

Figure 6 • Creating HDL File

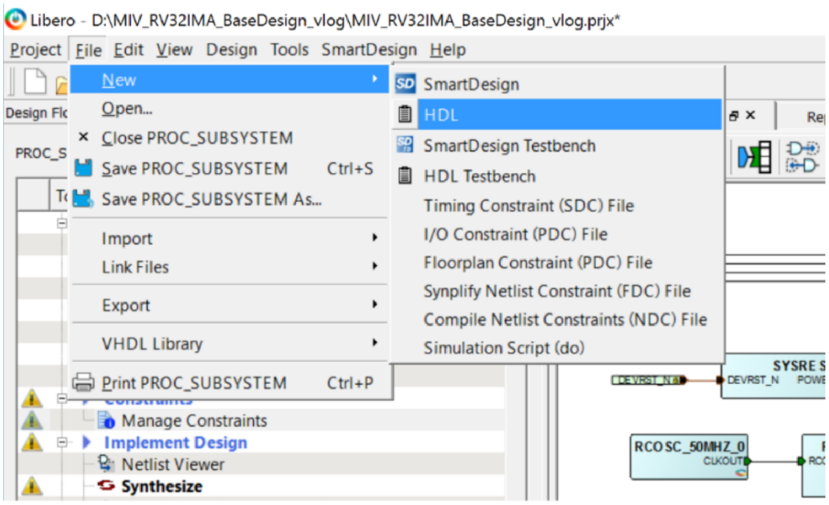
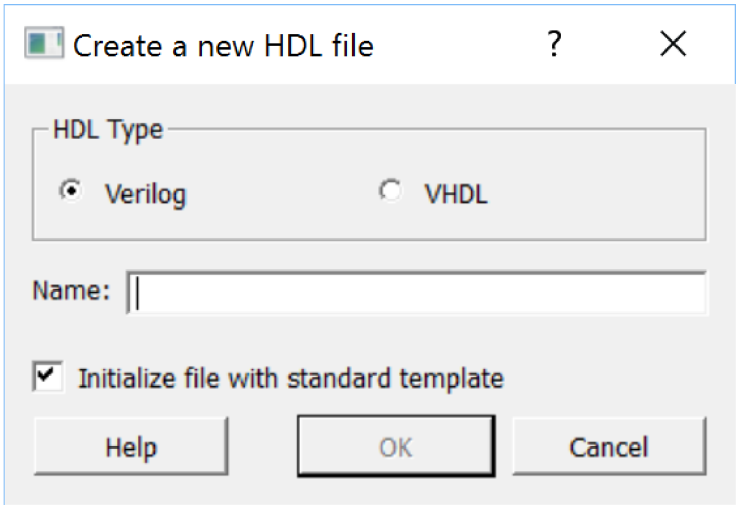


Figure 7 • Creating HDL File 2



4. Save the file and return to your smart design.

5. Open the Design Hierarchy tab and select the "Build Hierarchy."

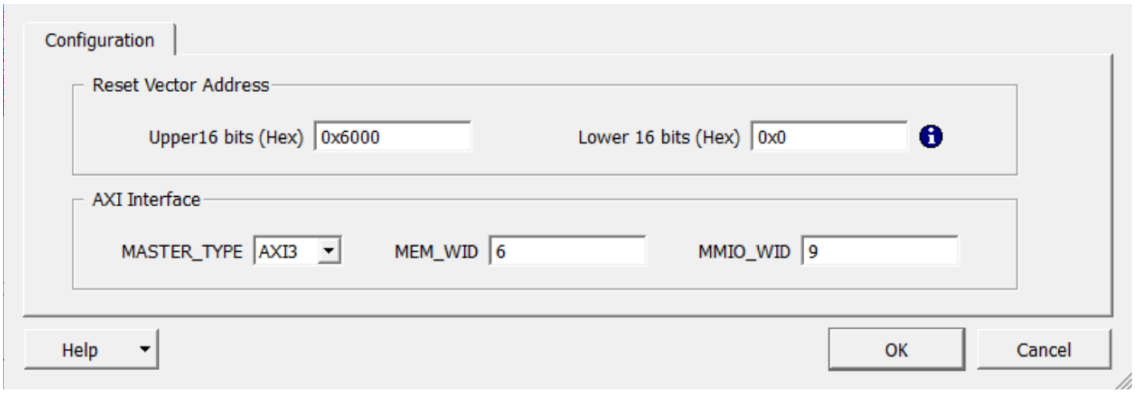
6. The reset_synchronizer.v file that was created should now be listed here.

7. Drag the file to the SmartDesign to add the reset synchronizer.

3.1.3 Mi-V CPU Configuration

The Mi-V CPU can be used in either an AXI4 or AXI3 master configuration. Drag the core to the canvas and double-click on it to open the configurator. The default configuration is AXI3 and this can be left unchanged.

Figure 8 • Mi-V Default AXI3 Configuration



3.1.4 COREAXITOAHBL x2 Configuration

This sections shows the CoreAXITOAHBL configuration.

Figure 9 • CoreAXITOAHBL Default Configuration

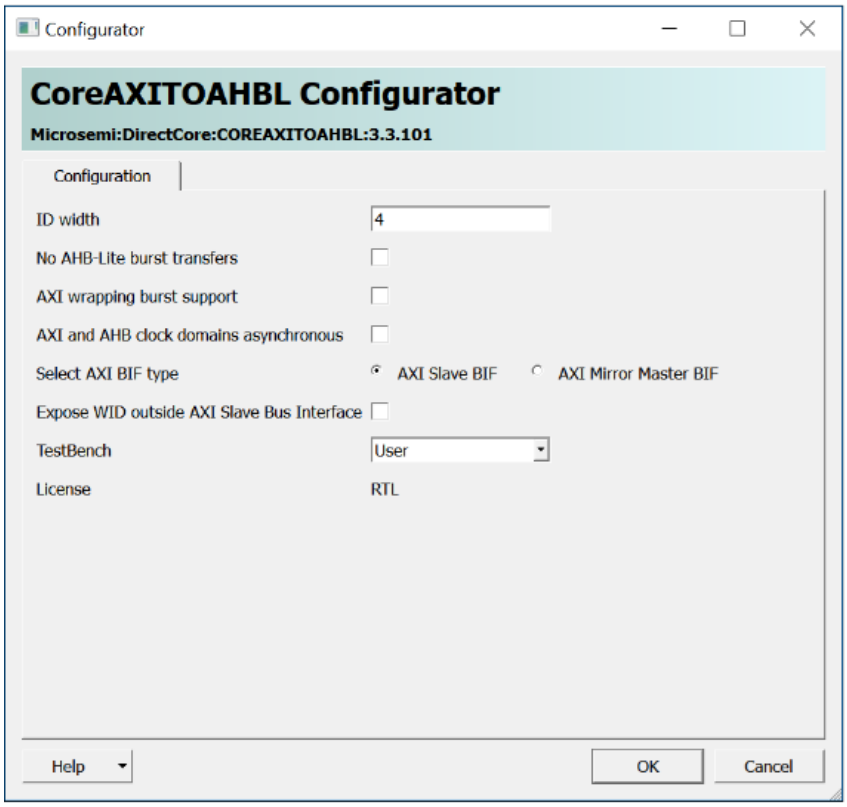
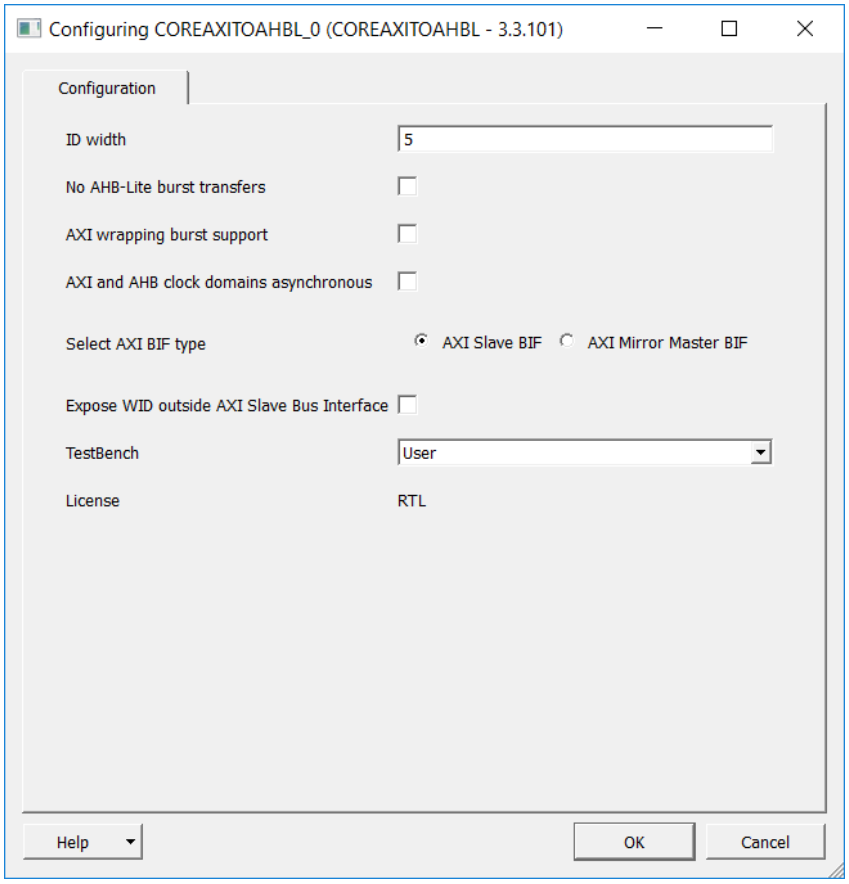


Figure 10 • CoreAXITOAHBL Required Configuration



3.1.5 CoreAHBLite_0 Configuration

The following shows the CoreAHBLite_0 default configuration.

Figure 11 • CoreAHBLite_0 Default Configuration

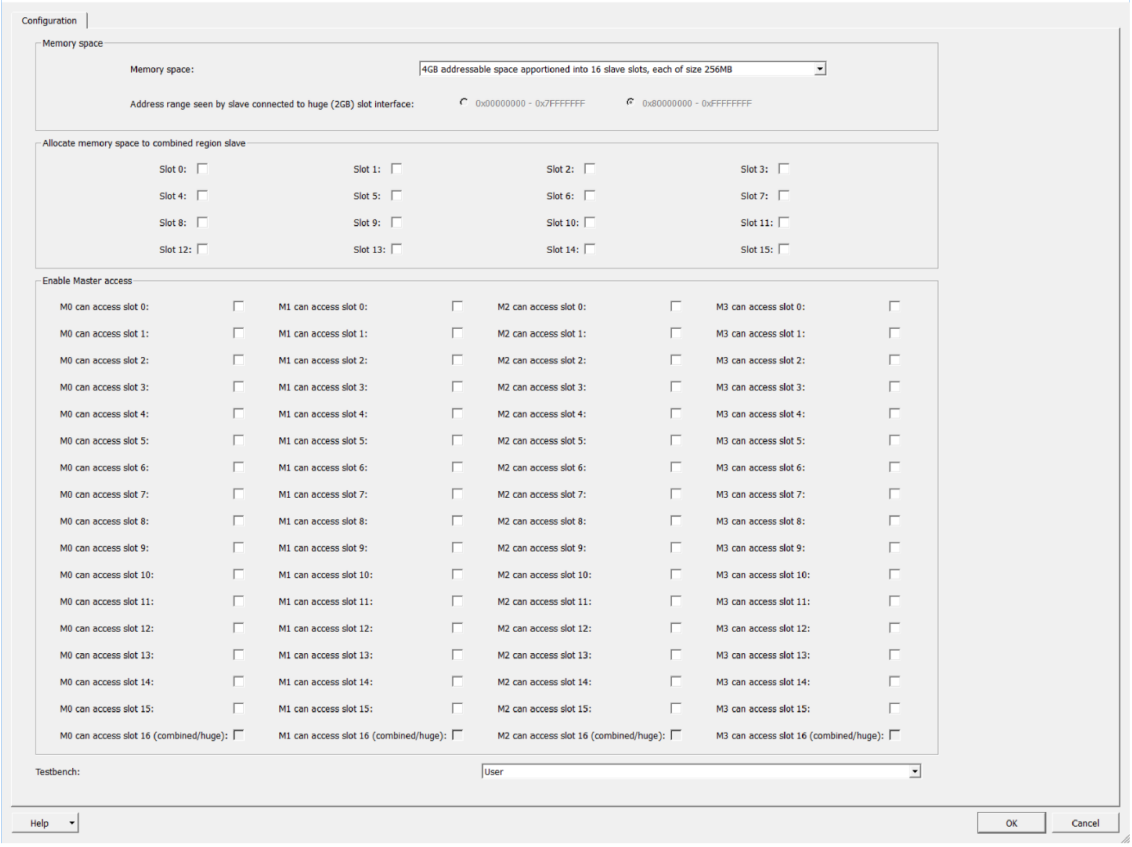
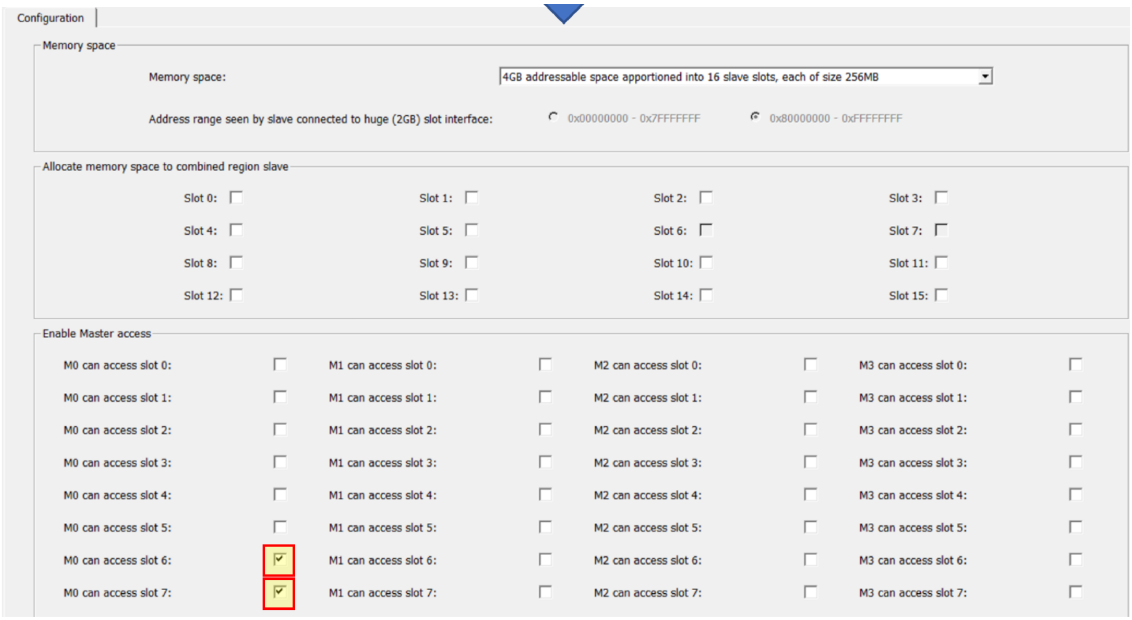


Figure 12 • CoreAHBLite_0 Default Configuration



3.1.6 CoreAHBLite_1 Configuration

The following shows the CoreAHBLite_1 default configuration.

Figure 13 • CoreAHBLite_1 Default Configuration

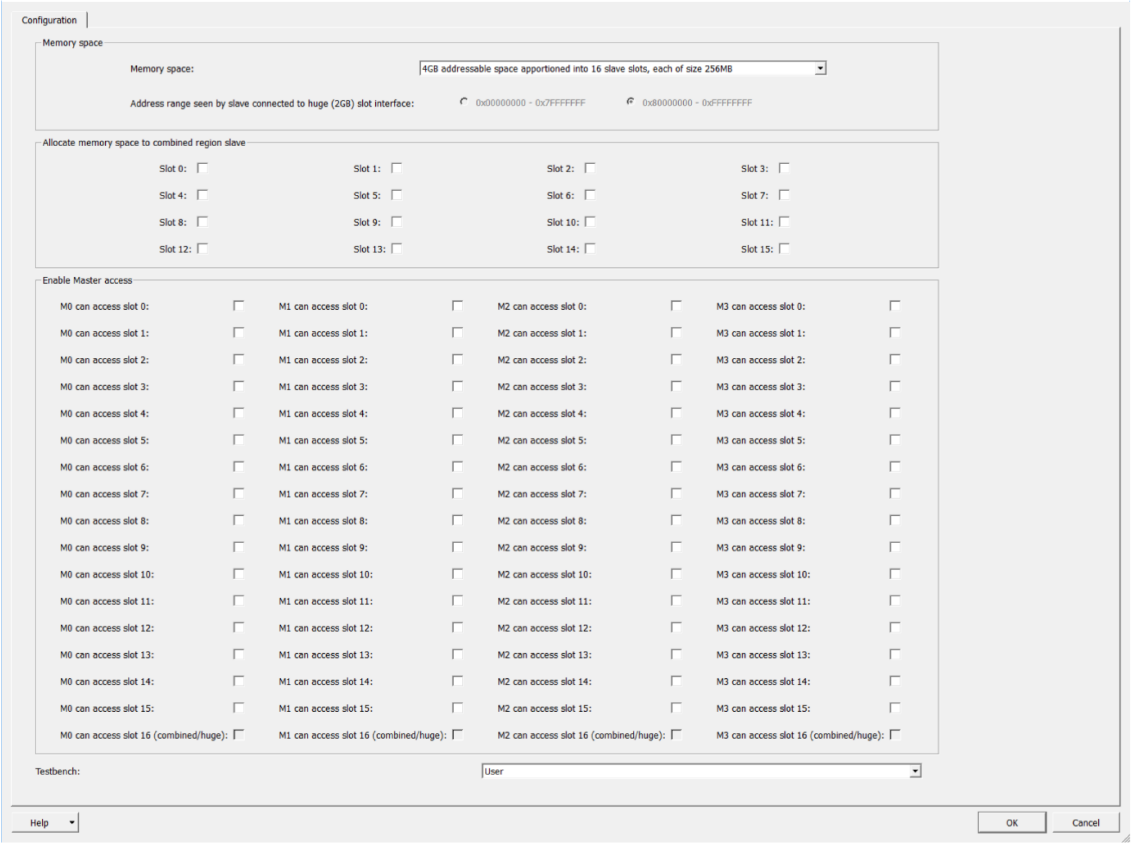
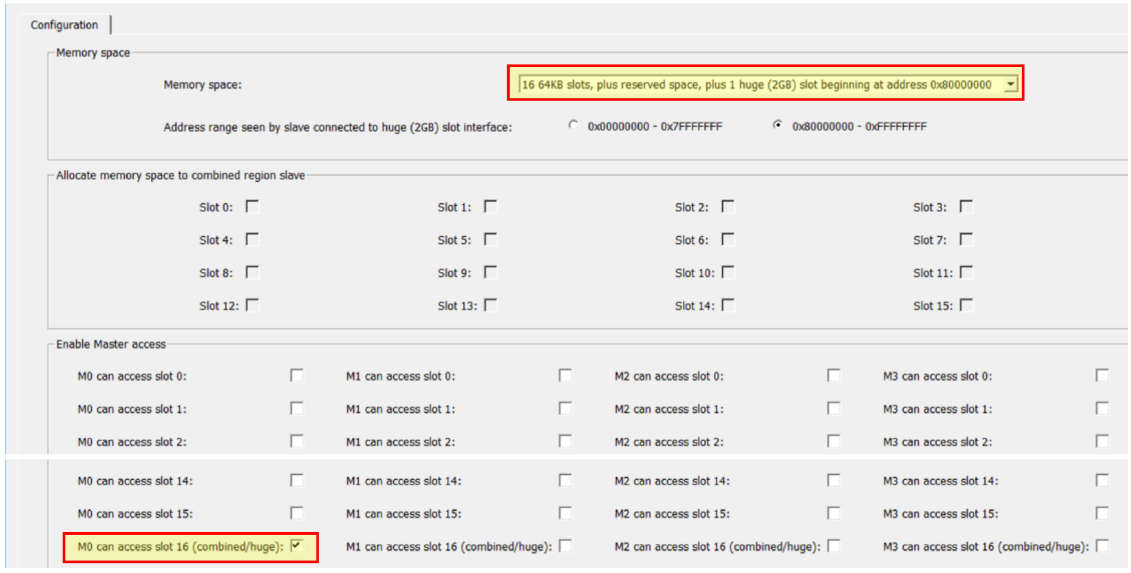


Figure 14 • CoreAHBLite_1 Required Configuration



3.1.7 SRAM Configuration

The following illustrations show the SRAM configuration.

Figure 15 • SRAM Default Configuration

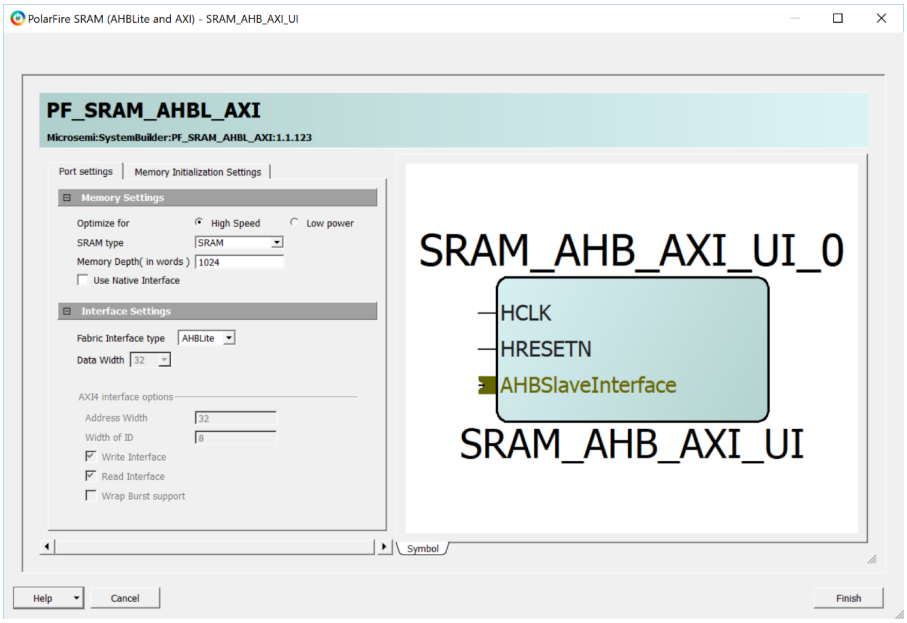
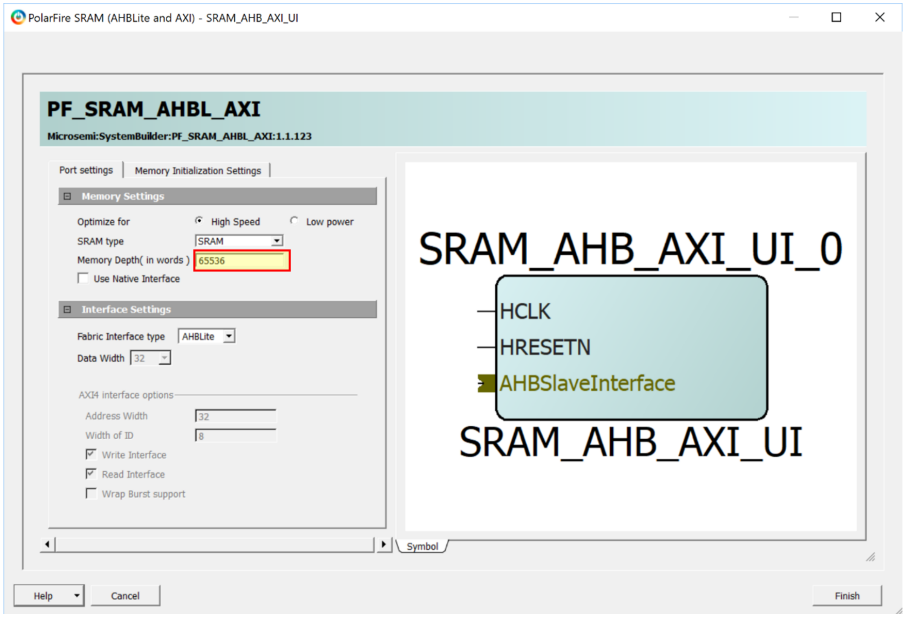


Figure 16 • SRAM Required Configuration

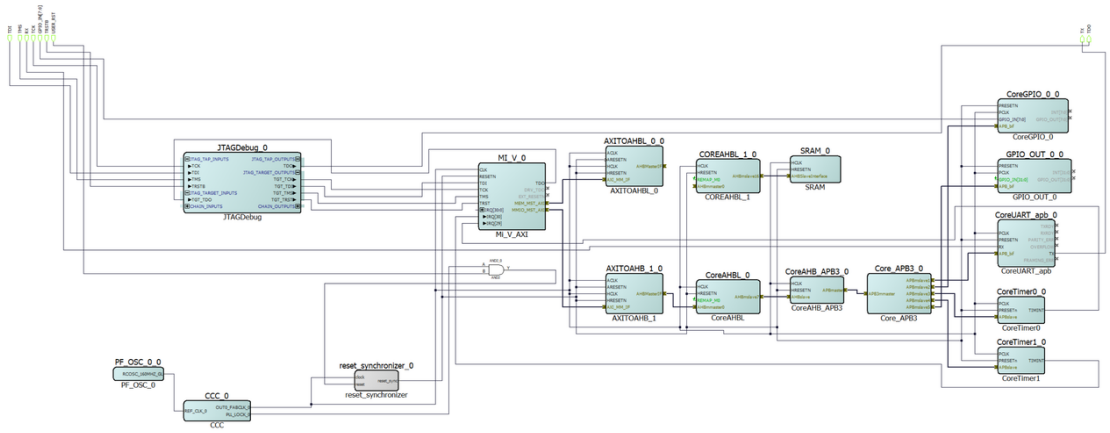


3.2 Mi-V Connecting Components

Once you have placed and configured your components,connect them as shown in the image below.

The final design layout with all ports connected is shown in the following illustration.

Figure 17 • Core Design Layout Design with Connections



The following table shows the port connections.

Table 2 • Port Connections

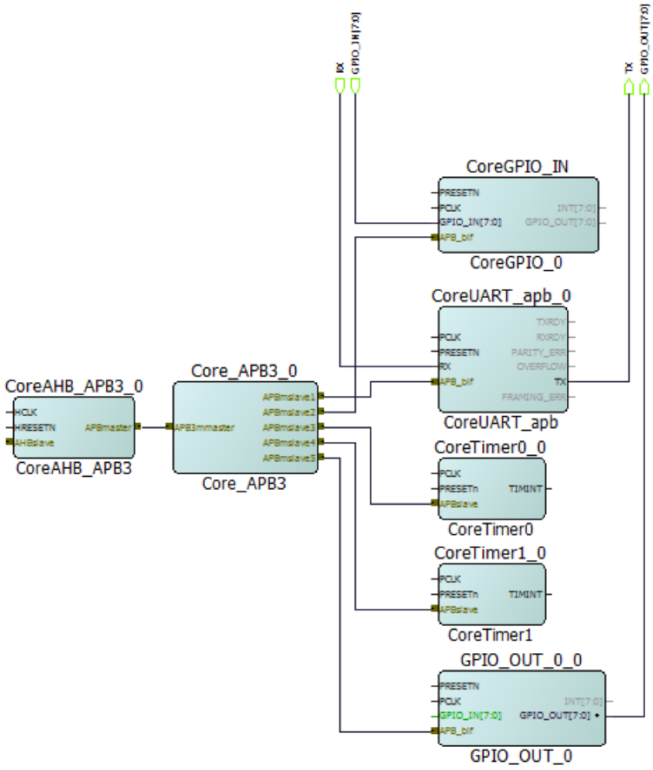
Device/Port	Signal/Port Name	End Device/Port	End Pad/Port Name
Input port	TDI	COREJTAGDEBUG_0	TDI
Input port	TCK	COREJTAGDEBUG_0	TCK
Input port	TMS	COREJTAGDEBUG_0	TMS
Input port	TRSTB	COREJTAGDEBUG_0	TRSTB
Input port	USER_RST	AND2_0	B
PF_OSC_0_0	RCOSC_160MHZ_GL	CCC_0	REF_CLK
CCC_0	OUT0_FABCLK_0	*Global clock pin*	
	PLL_LOCK	AND2_0	A
AND2_0	Y	Reset_synchronizer_0	Reset
Reset_synchronizer_0	Reset_sync	* Global reset pin *	
COREJTAGDEBUG	TGT_TCK	Mi-V_0	TCK
	TGT_TRST	Mi-V_0	TRST
	TGT_TMS	Mi-V_0	TMS
	TGT_TDI	Mi-V_0	TDI
	TDO	Output port	TDO
	UDRCAP_OUT	*Unused*	
	UDRSH_OUT	*Unused*	
	UIREG_OUT[7:0]	*Unused*	
	URSTB_OUT	*Unused*	
	UDRCK_OUT	*Unused*	
	UTDI_OUT	*Unused*	

Device/Port	Signal/Port Name	End Device/Port	End Pad/Port Name
Mi_V_0	DEBUG_DMACTIVE	Output port	DM_ACTIVE
	TDO	COREJTAGDEBUG	TGT_TDO
	MEM_MST_AXI*	COREAXITOAHBL_1_0	AXISlaveIF
	AXI4_MST_MMIO*	COREAXITOAHBL_0_0	AXISlaveIF
	DRV_TDO	*Unused*	
	MEM_AXI_WID[4:0]	COREAXITOAHBL_0_0	WID[4:0]
	MMIO_AXI_WID[4:0]	COREAXITOAHBL_1_0	WID[4:0]
COREAXITOAHBL_0_0*	AHBMasterIF	CoreAHBL_0	AHBmmaster0
CoreAHBL_1	AHBmslave8	SRAM_0	AHBSlaveInterface

3.3 Mi-V Peripheral Setup

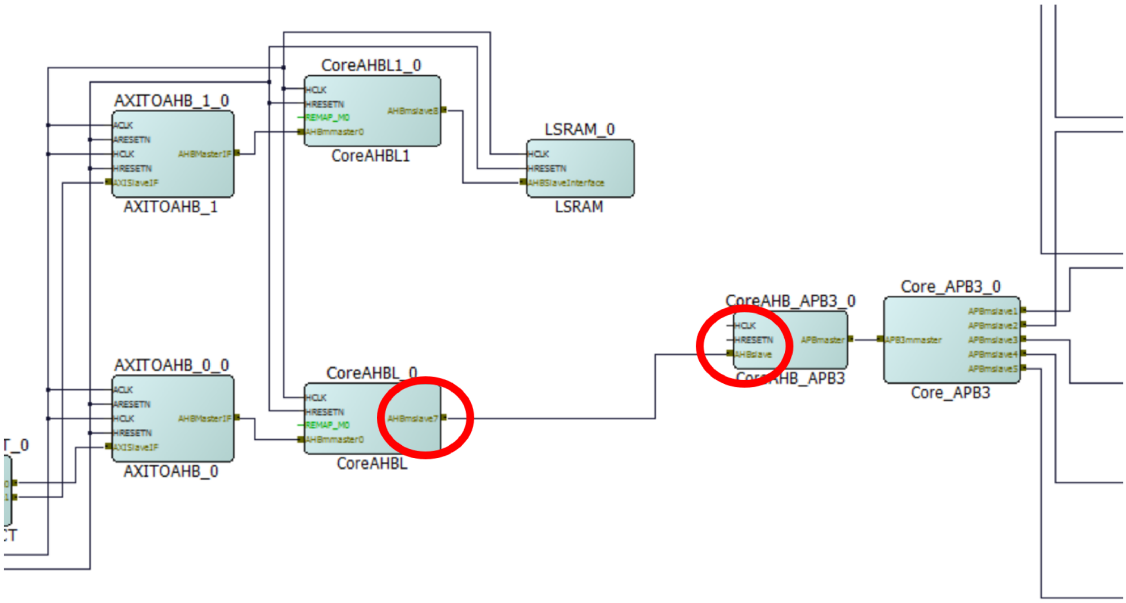
The previous section demonstrated the steps involved in setting up a Mi-V core. To set up peripherals for use with the core, follow these instructions. This setup will be done in the same project as the core and connected to port AHBmslave7 of CoreAHBL_0. The section of the design that will be created is shown in the following illustration.

Figure 18 • Peripherals Design



It will connect to the previously created design as shown in the following illustration.

Figure 19 • Peripherals Connection to Core Design



To begin, drag components to the SmartDesign canvas.

Table 3 • SmartDesign Canvas

Component	Quantity	Version
COREAHBTOAPB3	1	v3.1.100
CoreABP3	1	v4.1.100
CoreUARTapb	1	v5.5.102
CoreGPIO	1	v3.2.102
CoreTimer	1	v2.0.103

Set up the components as detailed below:

1. COREAHBTOAPB3: no setup required
2. CoreUART_apb: no setup required
3. CoreTimer: no setup required

3.3.1 Core APB3 Configuration

The following section details the core APB3 configuration.

Figure 20 • Core APB3- Default Configuration

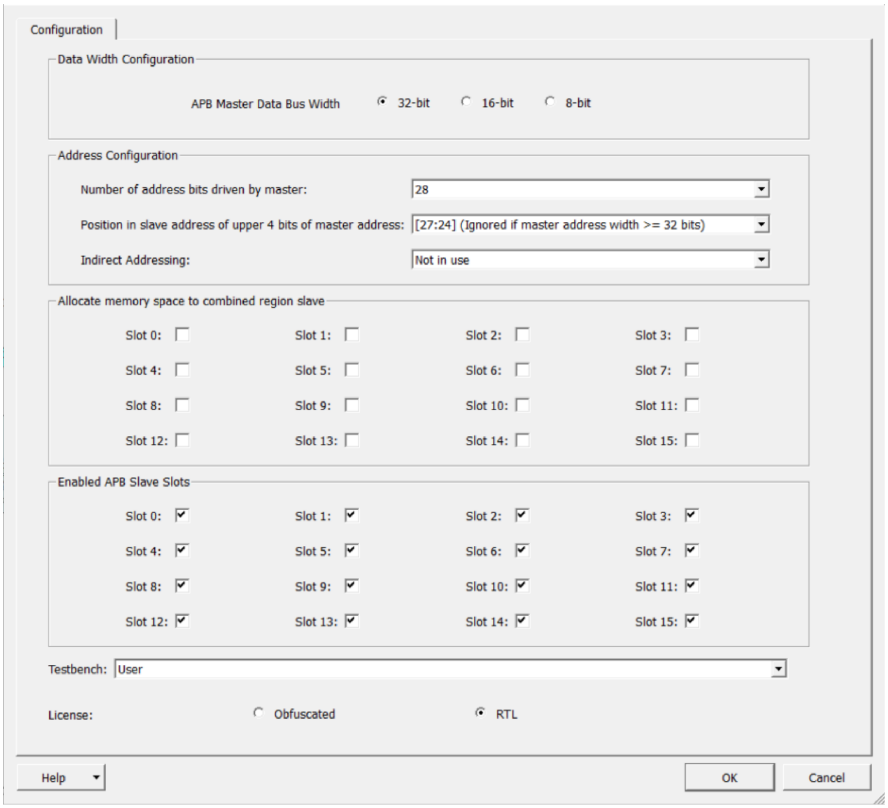
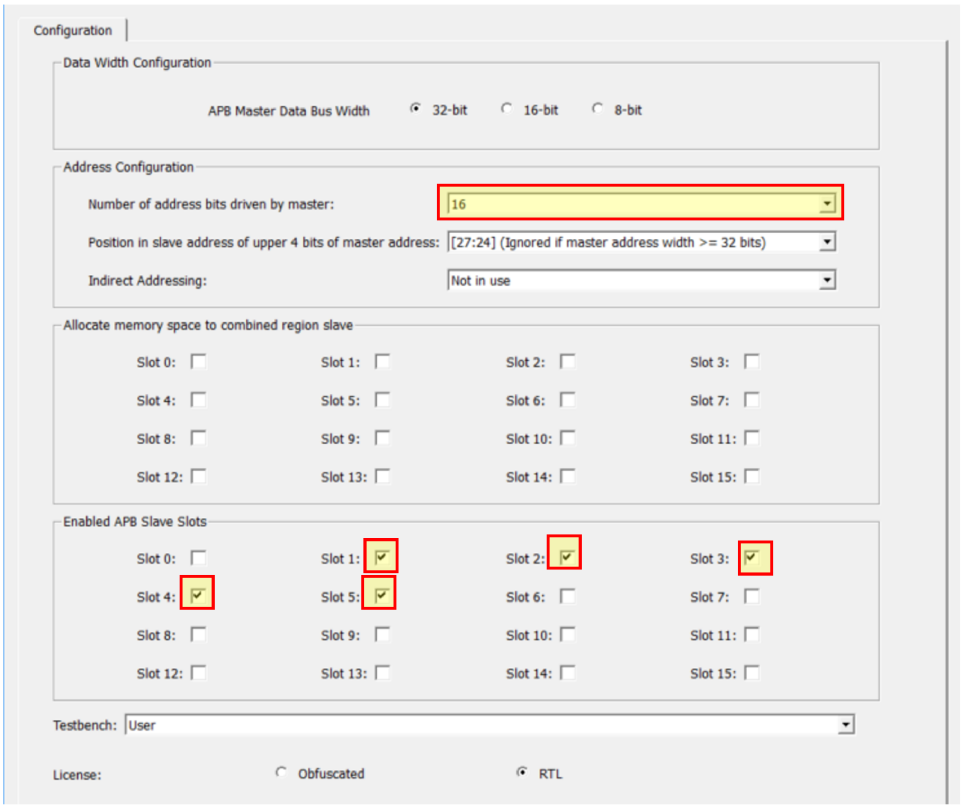


Figure 21 • CoreAPB3-Required Configuration



3.3.2 CoreGPIO_IN Configuration

The following section details the CoreGPIO_IN configuration.

Figure 22 • CoreGPIO_IN- Default Configuration

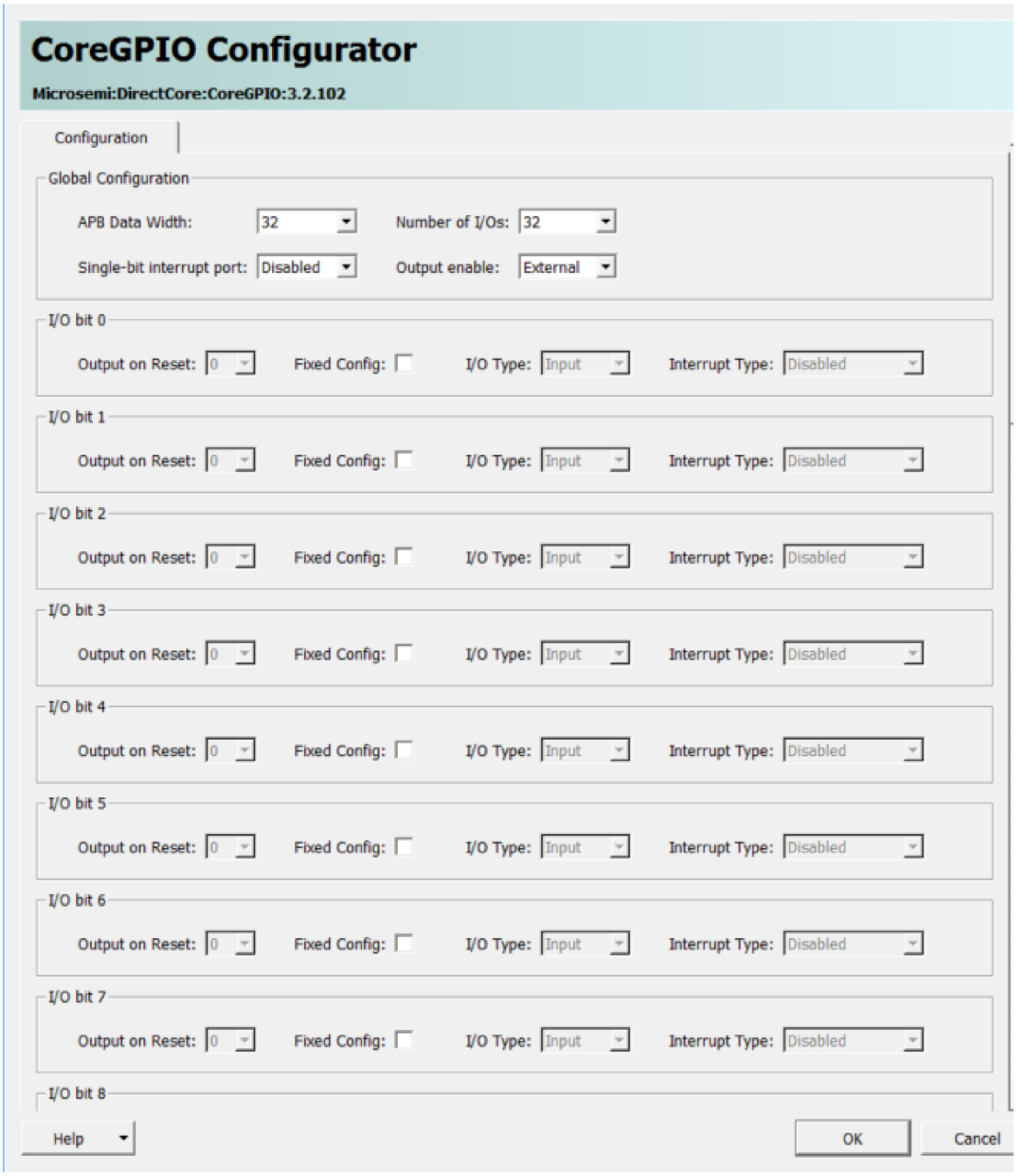


Figure 23 • CoreGPIO_IN- Required Configuration

3.3.3 CoreGPIO_OUT Configuration

The following section details the CoreGPIO_OUT configuration.

Figure 24 • CoreGPIO_OUT- Default Configuration

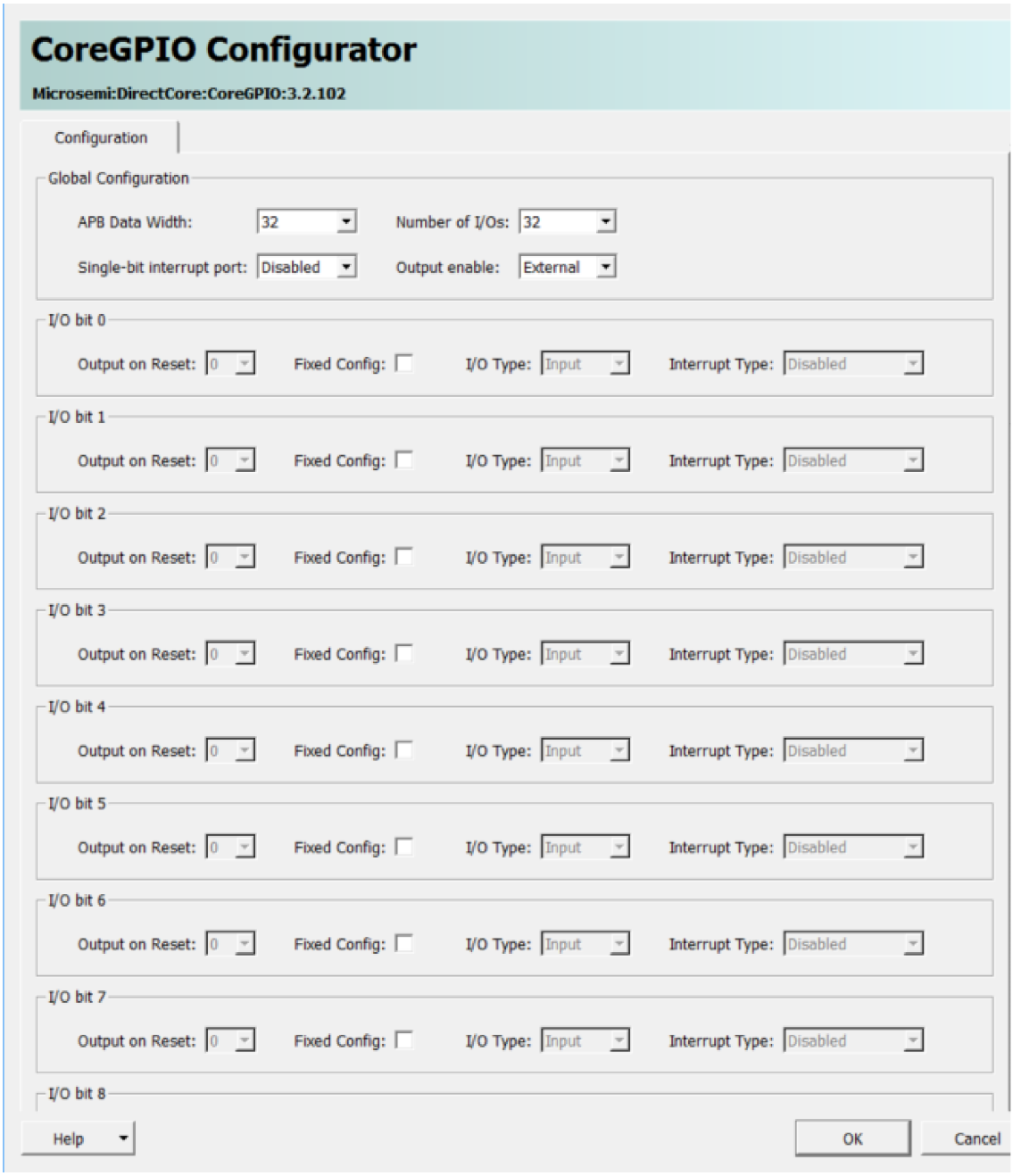
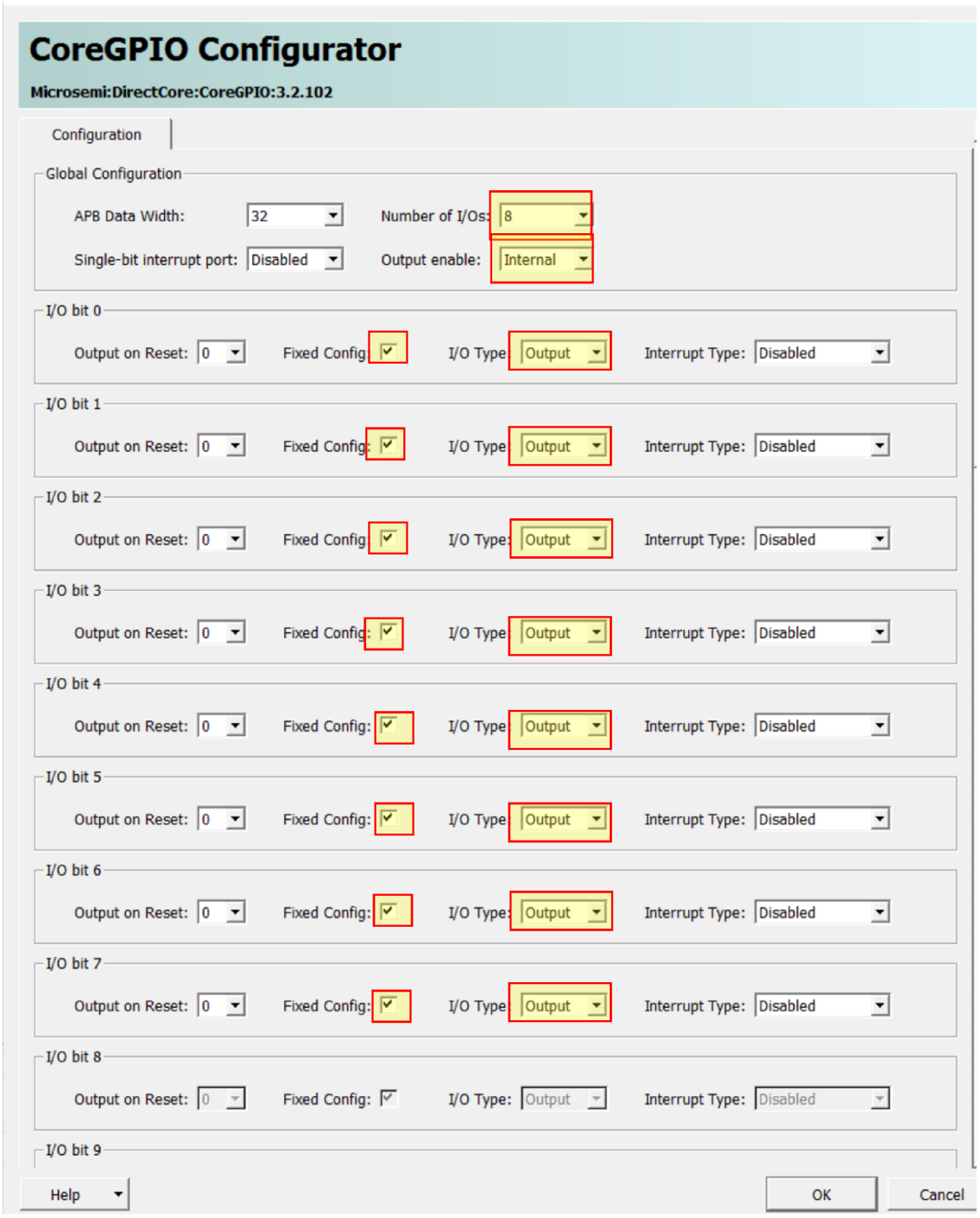


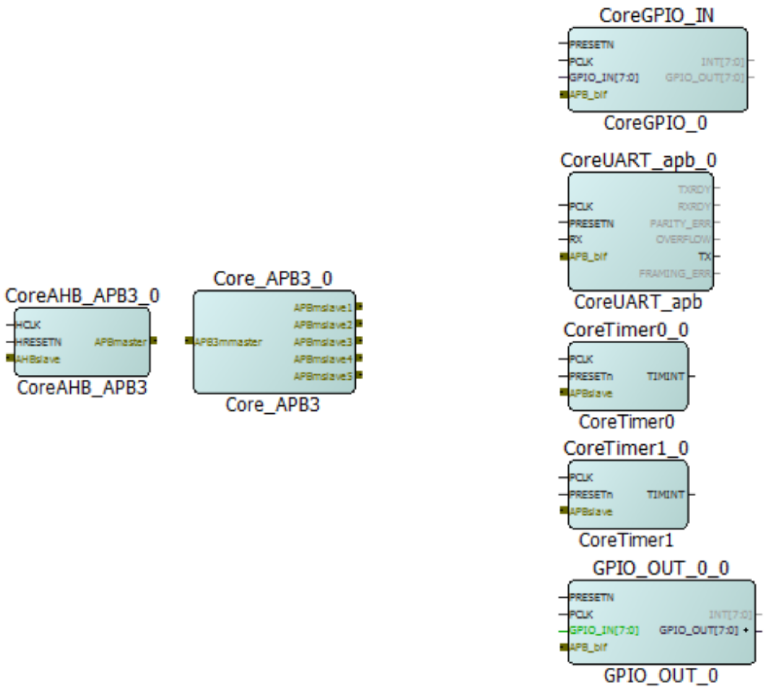
Figure 25 • CoreGPIO_OUT- Required Configuration



3.4 Mi-V Connecting Peripherals

Once you have placed and configured the components, you should have a layout similar to the following illustration.

Figure 26 • Peripheral Design, No Ports Connected



They should be connected as shown in the following illustration.

Figure 27 • Peripheral Design Port Connections

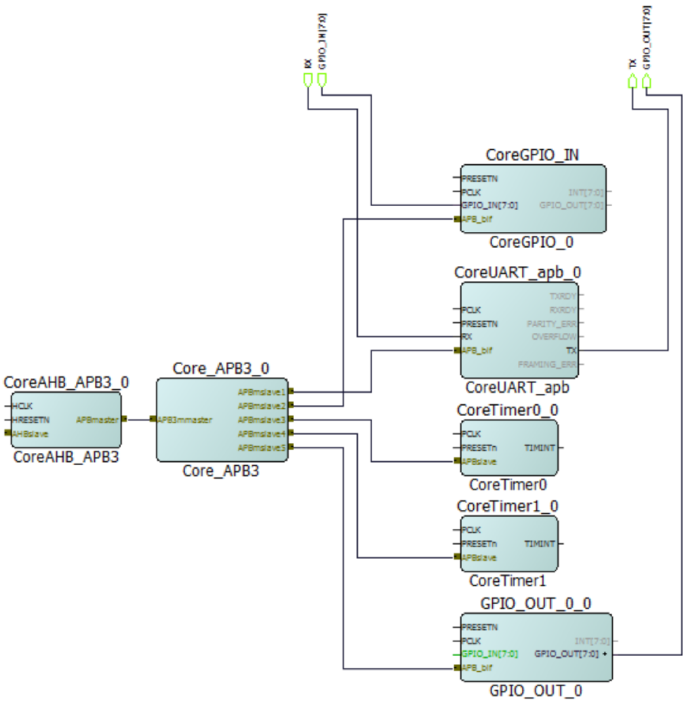


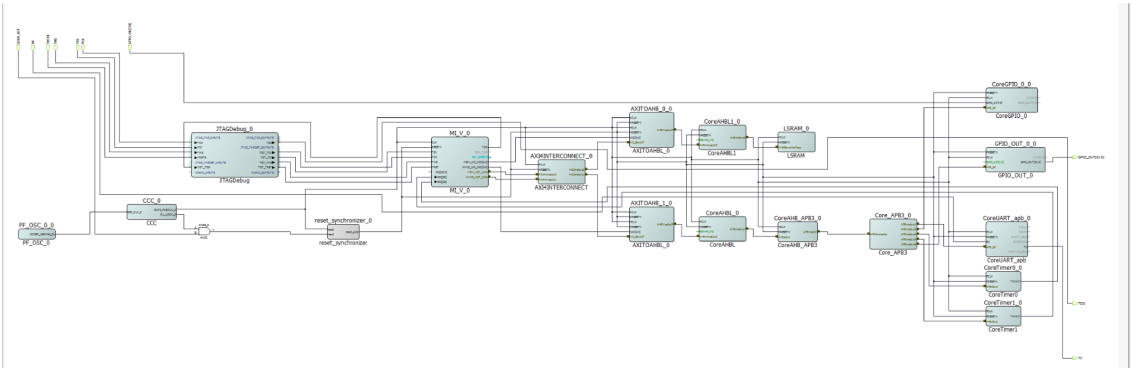
Table 4 • Table of Peripheral Design Connections

Device / port	Signal / port name	End device / port	End pad / port name
Input port	GPIO_IN[7:0]	CoreGPIO_IN	GPIO_IN[7:0]
Input port	RX	CoreUART_apb_0	RX
CoreUART_apb_0	TX	Output port	TX
GPIO_OUT_0_0	GPIO_OUT[7:0]	Output port	GPIO_OUT[7:0]
CoreAHB_APB3_0	APBmaster	Core_APB3_0	APB3mmaster
Core_APB3_0	APBmslave1	CoreUART_apb_0	APB_bif
	APBmslave2	CoreGPIO_IN	APB_bif
	APBmslave3	CoreTimer0_0	APBslave
	APBmslave4	CoreTimer1_0	APBslave
	APBmslave5	GPIO_OUT_0_0	APB_bif
CoreTimer0_0	TIMINT	MI_V_0	IRQ29
CoreTimer1_0	TIMINT	MI_V_0	IRQ30

3.5 Connecting the Core Design to the Peripherals Design

Connecting the two designs is simple. All that needs to be done is to connect the Core_AHBL_0's "AHBmslave7" port to CoreAHB_APB3_0's "AHBslave" port, and then connect the clock and reset pins. The pin that the peripherals are connected to can be changed by editing the base address in the hw_platform.h file included with the RISC-V HAL in SoftConsole. If new or different peripherals are connected, that address can also be changed in this file.

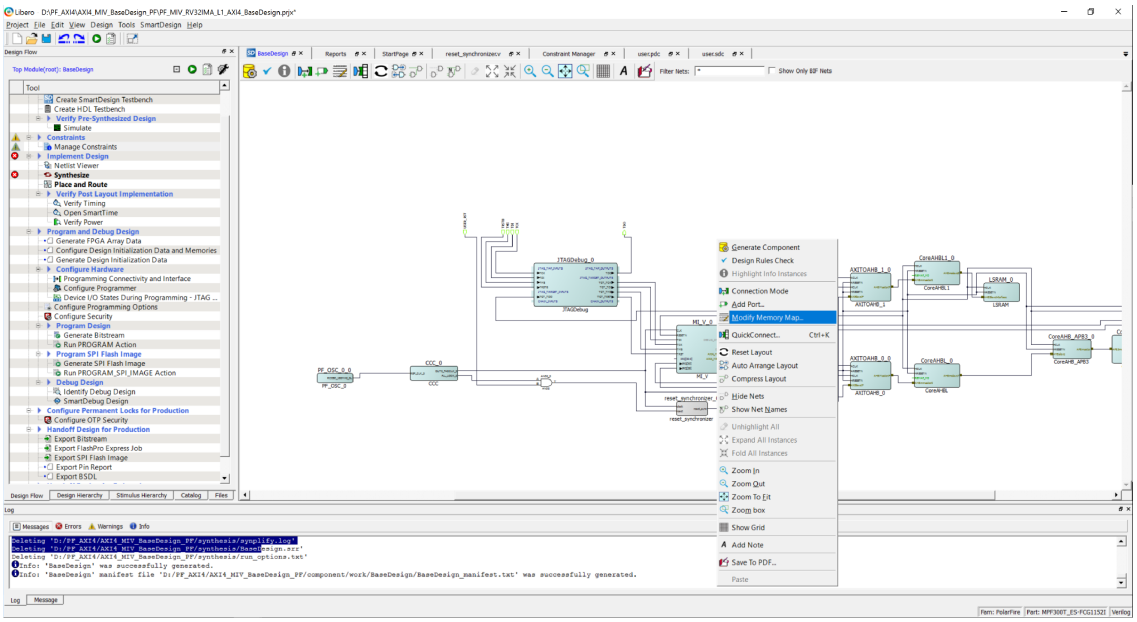
Figure 28 • Connected Core and Peripheral Designs



4 Generating a Memory Map

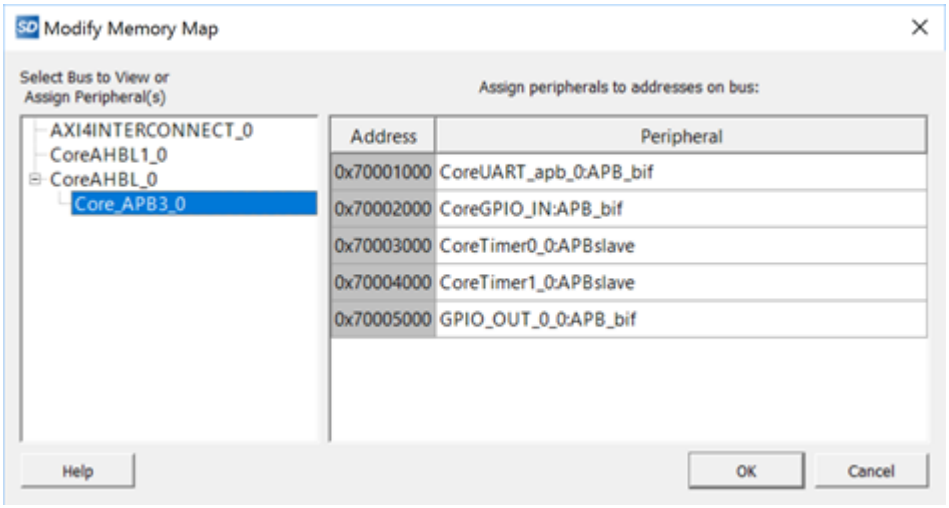
A memory map will show the slot numbers on different interfaces. These slot numbers correspond to the memory address on the bus. To see the memory map, right click on the design canvas and select Modify Memory Map.

Figure 29 • Accessing Memory Map from Design Canvas



This will display the memory map for the entire design and allow you to see the memory addresses for each device connected to the bus.

Figure 30 • Contents of Memory Map



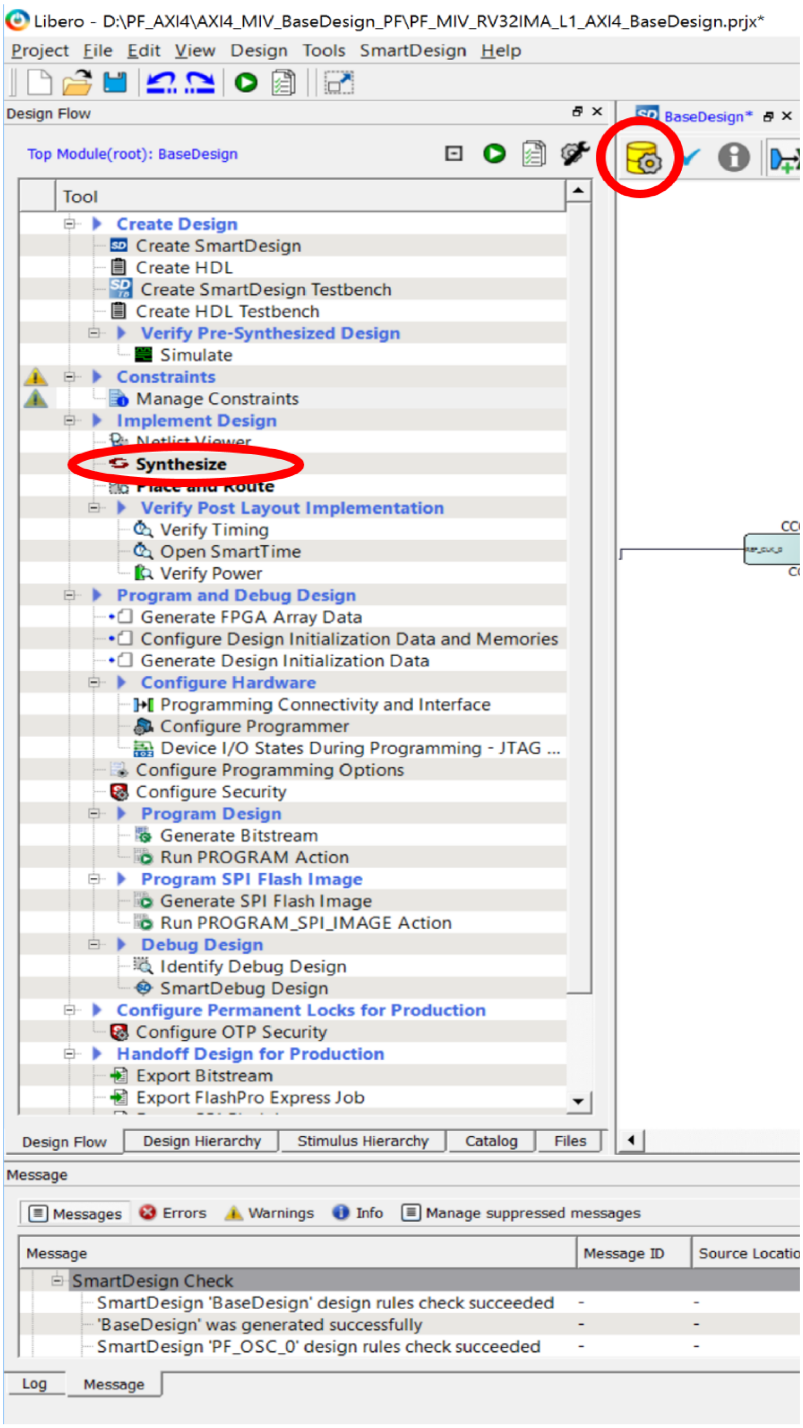
5 Generating a Bitstream and Downloading the FPGA Data to a Device

There are several stages to complete before downloading the design to the FPGA.

The design first needs to be generated. Select the Generate button to do this.

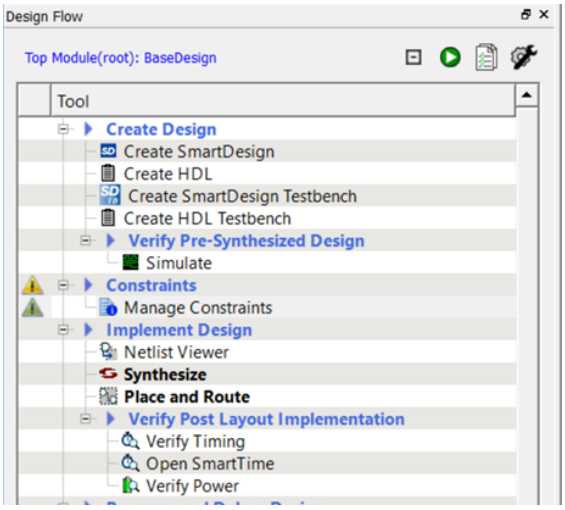
Once the design has finished generating, it needs to be synthesized. To do this, double click Synthesize.

Figure 31 • Generate and Synthesize



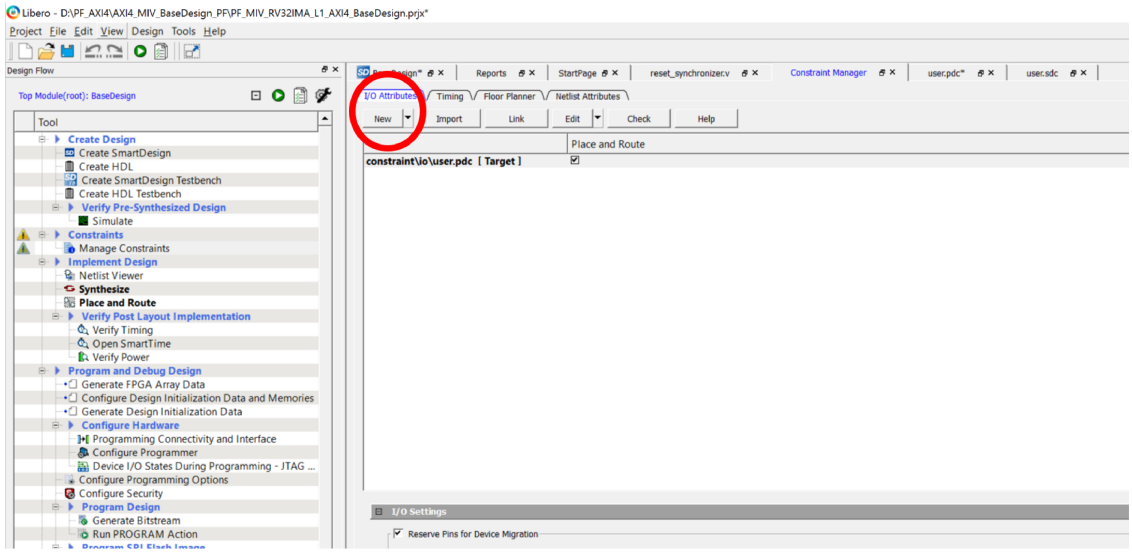
Once synthesis is complete, you will need to provide I/O and timing constraints using the constraints manager.

Figure 32 • Opening the Constraints Manager



Double click Manage Constraints to open the constraints manager.

Figure 33 • Constraints Manager



Name your constraint file and give it the extension ".pdc."

This file holds the input and output attributes for the top level ports in the design. Below is a constraint file for the design created in the document.

Note: These constraints are accurate for MPF_300TS_ES_FCG1152 devices. If you are using a different device, please change the constraints accordingly.

```
set_io -port_name {GPIO_IN[0]} \

-pin_name H23 \

-fixed true \
```

```
-DIRECTION INPUT
set_io -port_name {GPIO_IN[1]} \
-pin_name D21 \
-fixed true \
-DIRECTION INPUT
set_io -port_name {GPIO_IN[2]} \
-pin_name H24 \
-fixed true \
-DIRECTION INPUT

set_io -port_name {GPIO_IN[3]} \
-pin_name C22 \
-fixed true \
-DIRECTION INPUT

set_io -port_name {GPIO_IN[4]} \
-pin_name B21 \
-fixed true \
-DIRECTION INPUT

set_io -port_name {GPIO_IN[5]} \
-pin_name G20 \
-fixed true \
-DIRECTION INPUT

set_io -port_name {GPIO_IN[6]} \
-pin_name F24 \
-fixed true \
-DIRECTION INPUT

set_io -port_name {GPIO_IN[7]} \
-pin_name F25 \
-fixed true \
```

```
-DIRECTION INPUT

set_io -port_name {GPIO_OUT[0]} \
-pin_name D25 \
-fixed true \
-DIRECTION OUTPUT

set_io -port_name {GPIO_OUT[1]} \
-pin_name C26 \
-fixed true \
-DIRECTION OUTPUT

set_io -port_name {GPIO_OUT[2]} \
-pin_name B26 \
-fixed true \
-DIRECTION OUTPUT

set_io -port_name {GPIO_OUT[3]} \
-pin_name F22 \
-fixed true \
-DIRECTION OUTPUT

set_io -port_name {GPIO_OUT[4]} \
-pin_name H21 \
-fixed true \
-DIRECTION OUTPUT

set_io -port_name RX \
-pin_name H18 \
-fixed true \
-DIRECTION INPUT

set_io -port_name TX \
-pin_name G17 \
```

```
-fixed true \
-DIRECTION OUTPUT

set_io -port_name USER_RST \
-pin_name K22 \
-fixed true \
-DIRECTION INPUT

set_io -port_name TCK \
-pin_name J10 \
-DIRECTION INPUT

set_io -port_name TDI \
-pin_name K11 \
-DIRECTION INPUT

set_io -port_name TDO \
-pin_name K9 \
-DIRECTION OUTPUT

set_io -port_name TMS \
-pin_name J9 \
-DIRECTION INPUT

set_io -port_name TRSTB \
-pin_name N14 \
-DIRECTION INPUT
```

Copy this text into the editor that appears when a new constraint file is created. Save and close the file. Return to the constraints manager and ensure the I/O constraints are selected for Place and Route. Next, open the Timing tab in the constraints manager. Create a new timing constraints file, naming it and giving the extension ".sdc".

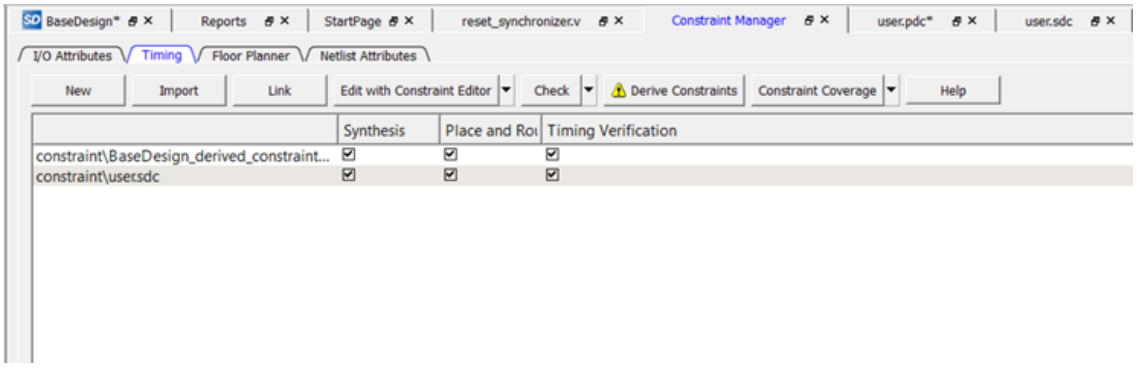
Below is the text for an SDC file for this project:

```
create_clock -name { TCK } \
-period 166.67 \
-waveform { 0 83.33 } \
```

```
[ get_ports { TCK } ]

set_false_path -from [ get_clocks { TCK } ] \

-to [ get_clocks { CCC_0/CCC_0/pll_inst_0/OUT0 } ]

set_false_path -from [ get_clocks { CCC_0/CCC_0/pll_inst_0/OUT0 } ] \

-to [ get_clocks { TCK } ]
```

Copy this text into the editor and save the file. Click Derive Constraints and allow it to run. Ensure the timing constraints are selected for Synthesis, Place and Route, and Timing Verification.

Figure 34 • Selecting Constraints for Use with Tools



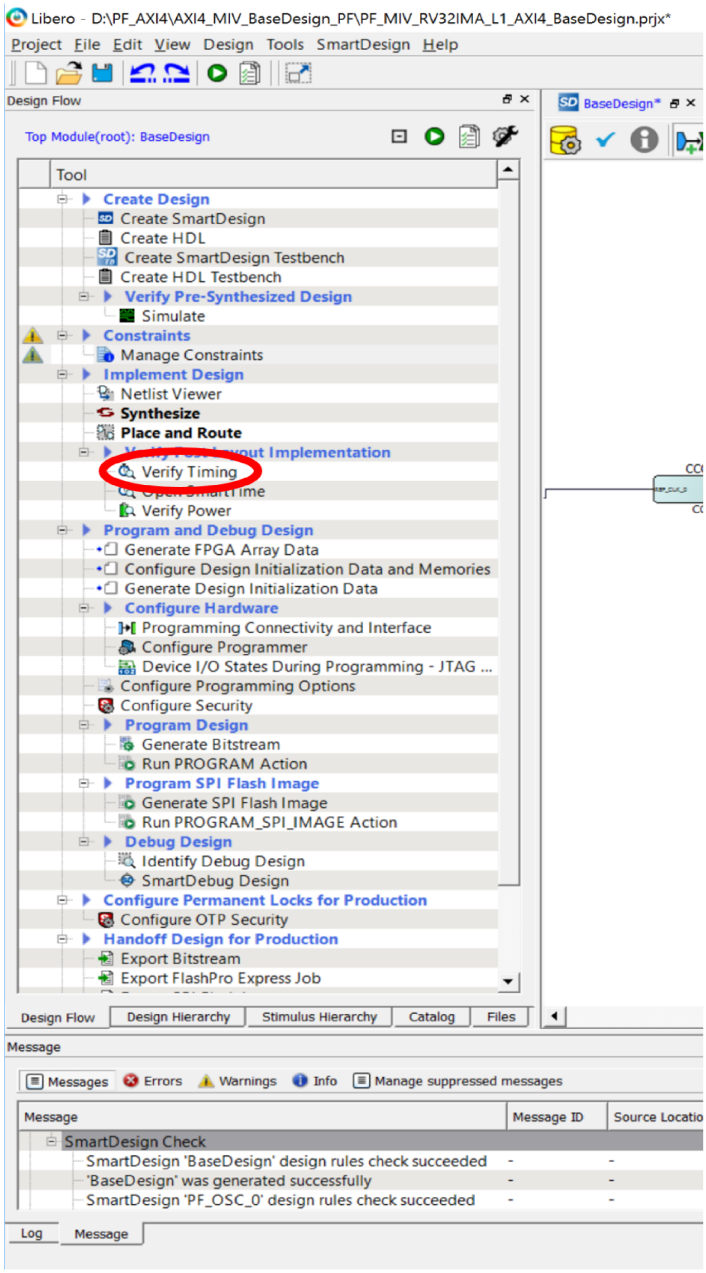
Finally, open the Floor Planner tab and create another user.pdc file. Below is the text for this file:

set_location -inst_name CCC_0/CCC_0/pll_inst_0 -location PLL0_SE

Copy this text into the editor and save the file. Make sure the floor planning is selected for place and route.

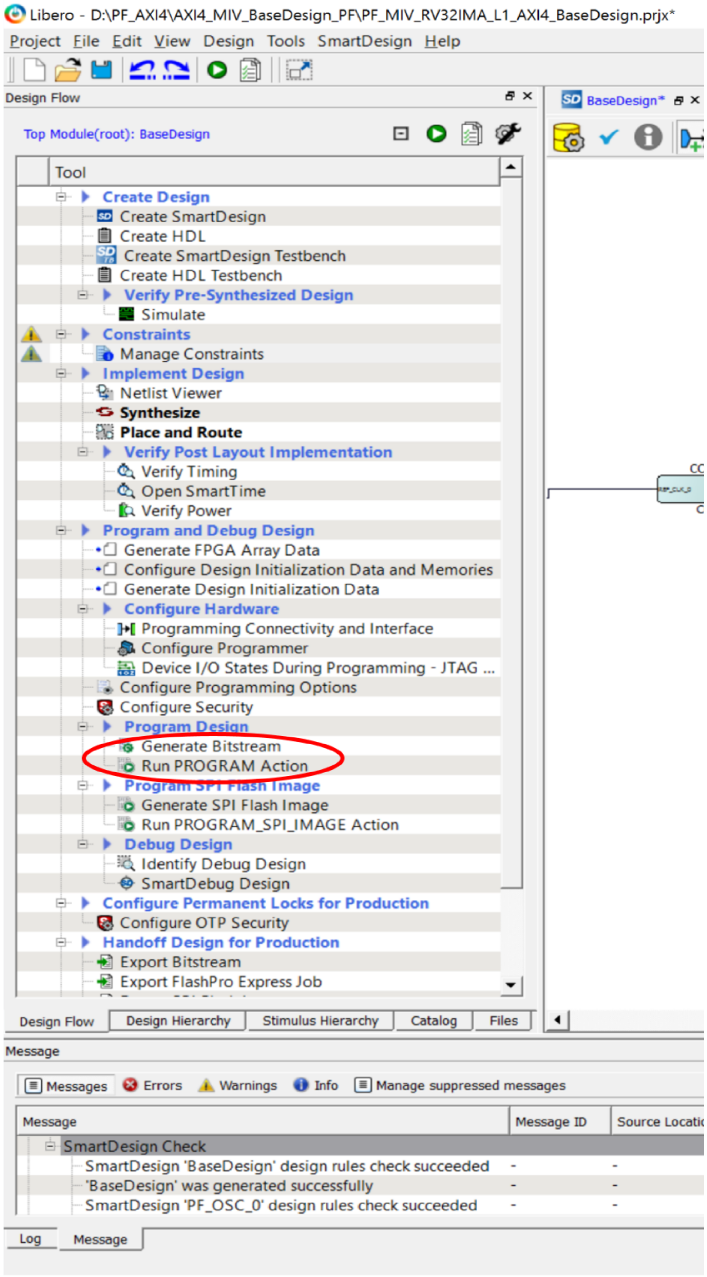
Once you have added the constraints, return to the design flow and scroll down to Verify Timing and double click. If you try to use a design that does not meet timing, it may not work correctly or function at all. If timing verification fails, you can examine the violating paths using the SmartTime tool.

Figure 35 • Generating a Bitstream and Downloading to the FPGA



Once you have verified that your design meets timing requirements, click Generate Bitstream. Once the bitstream has been generated, connect your device and make sure it is powered on. Then, select Run PROGRAM action. Once these actions have been completed, you can close Libero.

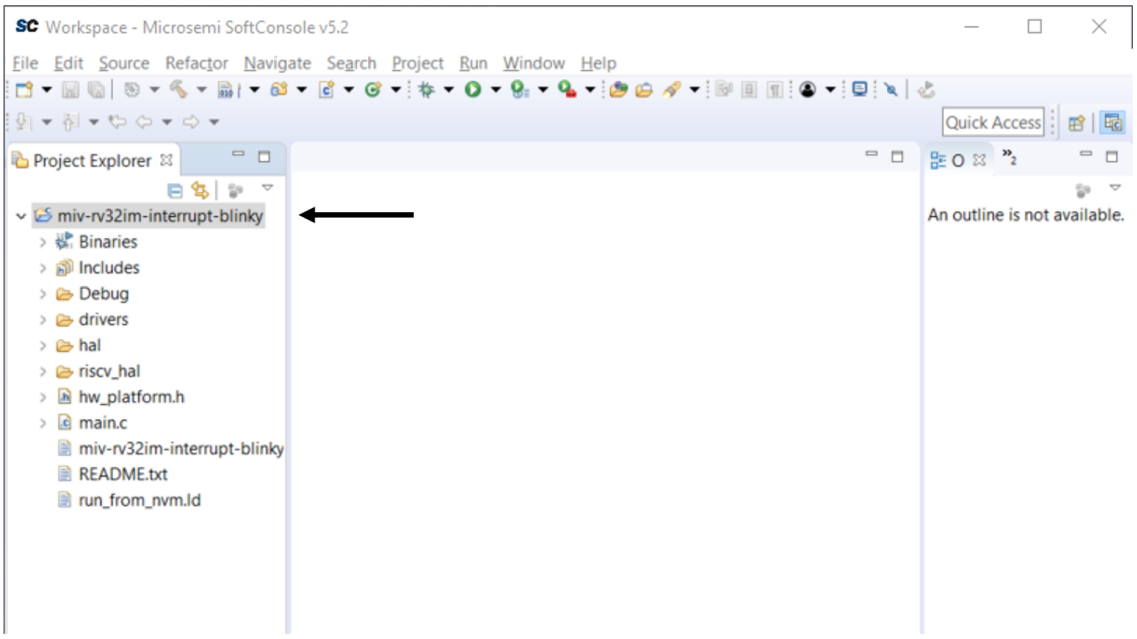
Figure 36 • Run Program



6 Running a Project from SoftConsole

To run RISC-V projects, you must have SoftConsole version 5.2 or later installed. Sample projects come bundled with SoftConsole in the workspace, and these can also be downloaded from the Microsemi Github page. To begin, open SoftConsole and double click on one of the miv projects to open it. In this case, miv-rv32im-interruptblinky will be demonstrated.

Figure 37 • Opening a Project in SoftConsole

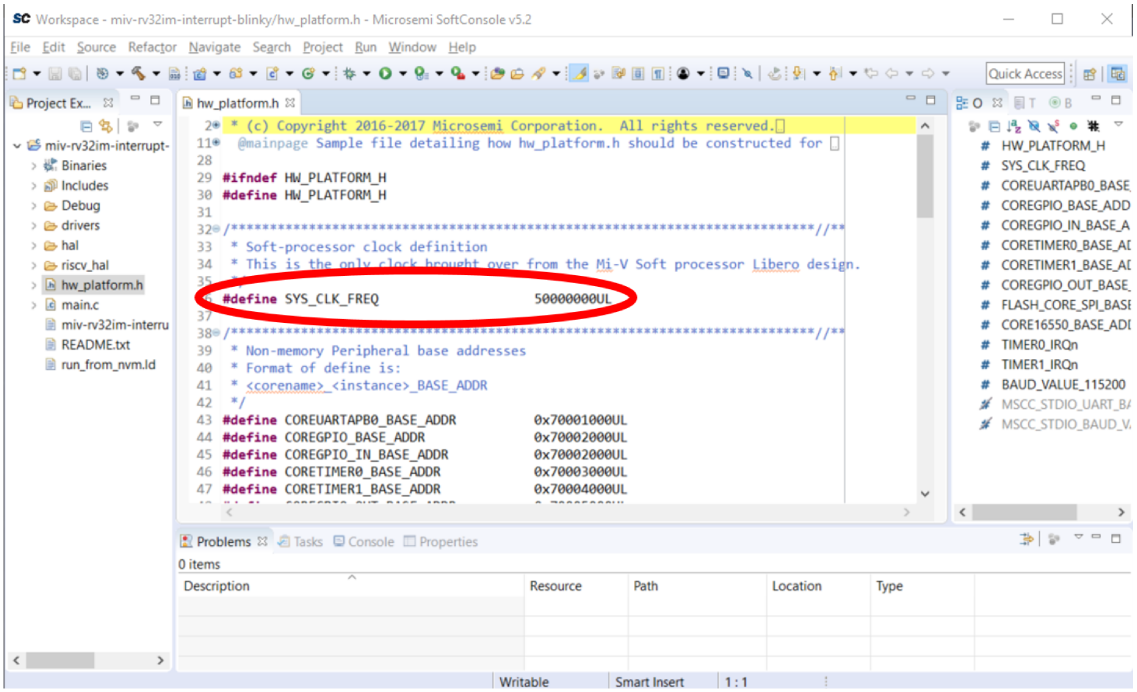


The driver and HAL files are already included in the project for you. If newer versions of these drivers are available, or if a core version is updated, it is advised to download the latest versions through the firmware catalog that is installed with Libero.

6.1 Mi-V Setting the System Clock Frequency and Peripheral Base Addresses

In PF_CCC configuration (see page 5), the clock conditioning circuit (CCC) was configured. The clock frequency is the system clock frequency. This also needs to be set in the core, which is done so in the "hw.platform.h" file by changing the #define SYS_CLK_FREQ in the clock frequency. This value must be in hertz.

Figure 38 • Setting Clock Frequency and Peripheral Base Addresses

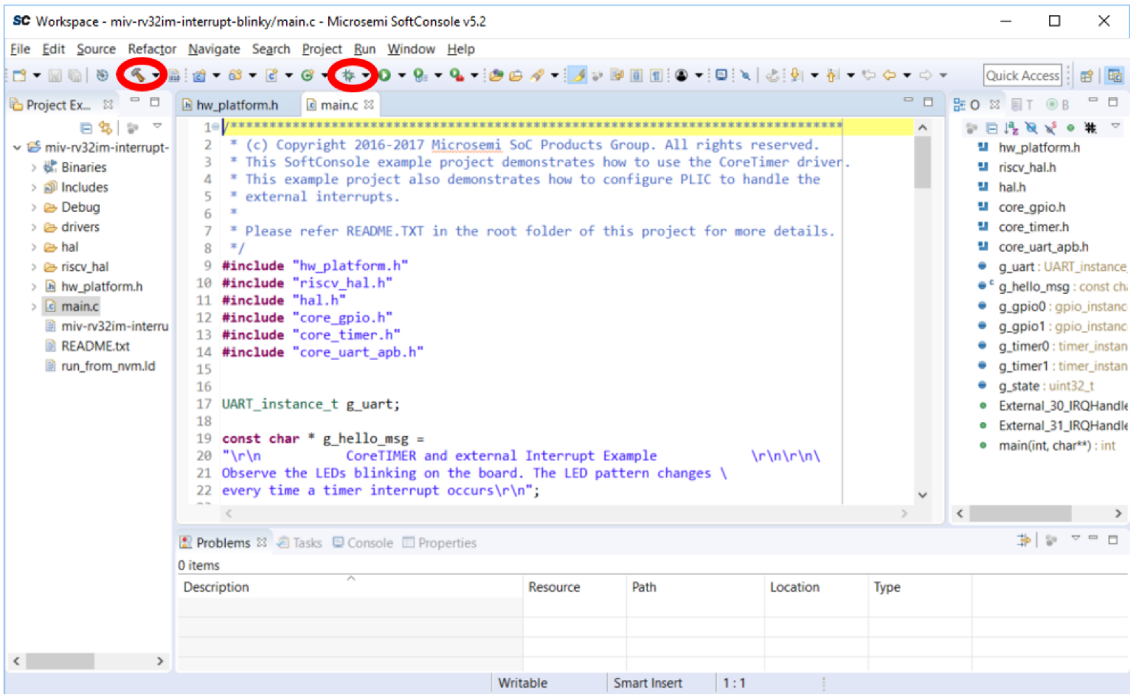


The “hw_platform.h” file is also used to set the base address for peripherals. In the Peripheral Design No Ports Connected (see page) illustration, the memory map was generated. The values from the memory should be in this file as the base addresses for the peripherals. The peripheral address must be correct for peripherals to function.

6.2 Building and Debugging a Project

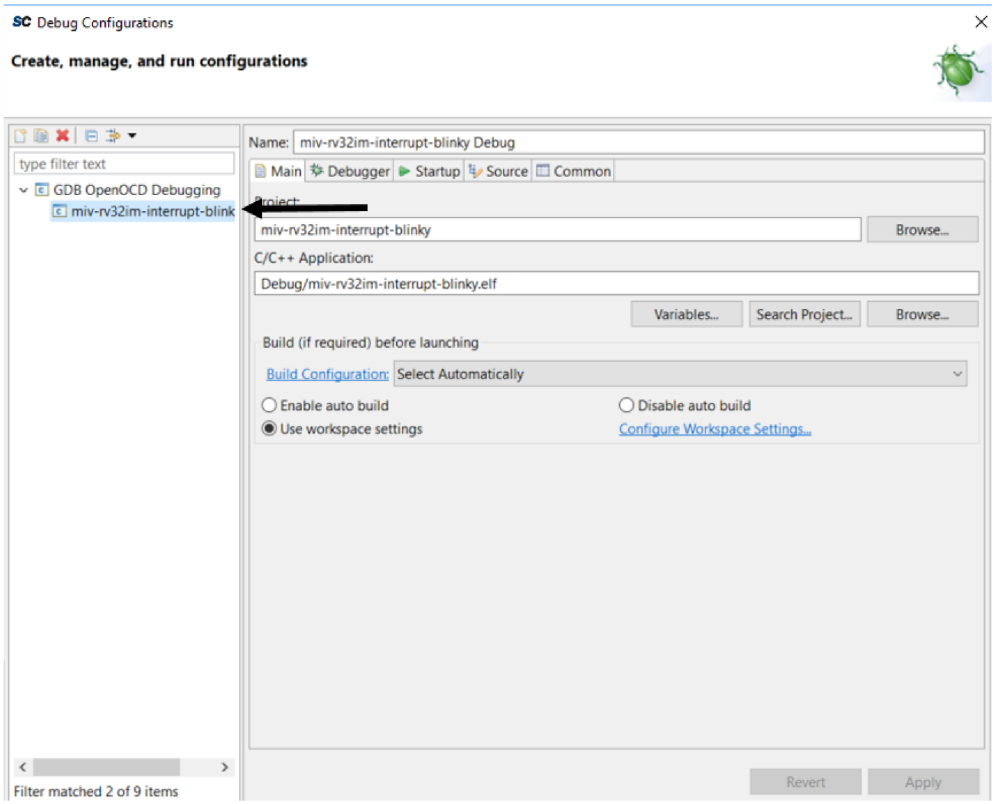
When you are ready to build your own project, you can click on the Build icon. In this case, the project is being built for debug.

Figure 39 • Building and Debugging 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 and then click the Debug icon. If this is your first time debugging this project, you will have to select a debug configuration.

Figure 40 • Running a Program



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 open but not run. It can be run by clicking the Resume button or pressing F8.

To add breakpoints and step through the code ensure you are in the "main.c" tab to view your code.

6.3 Mi-V Communicating through UART

This section illustrates communication through UART.

To communicate via UART, you need the device baud rate and the COM port that it is using. The device manager can be used to find the COM port of your device. To open it, right-click Start and select Device Manager. The COM port for your device will be listed under Ports (COM and LPT), It can be unclear which one is your device, so there may be trial and error to find the correct port.

Figure 41 • Opening Device Manager

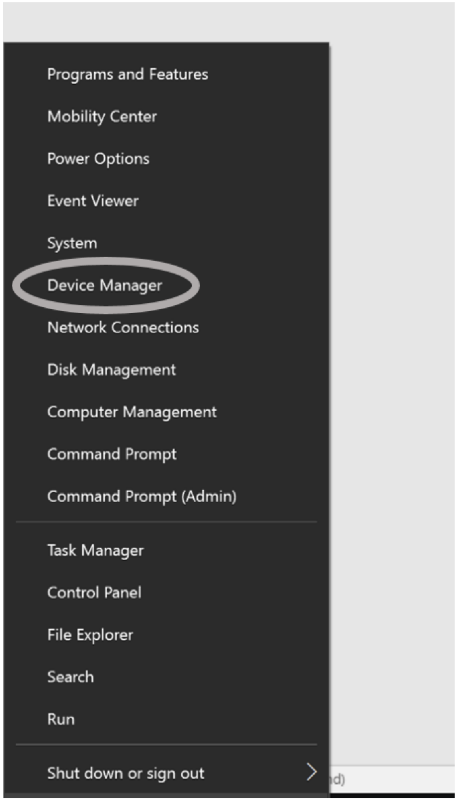


Figure 42 • Finding COM Ports in Device Manager

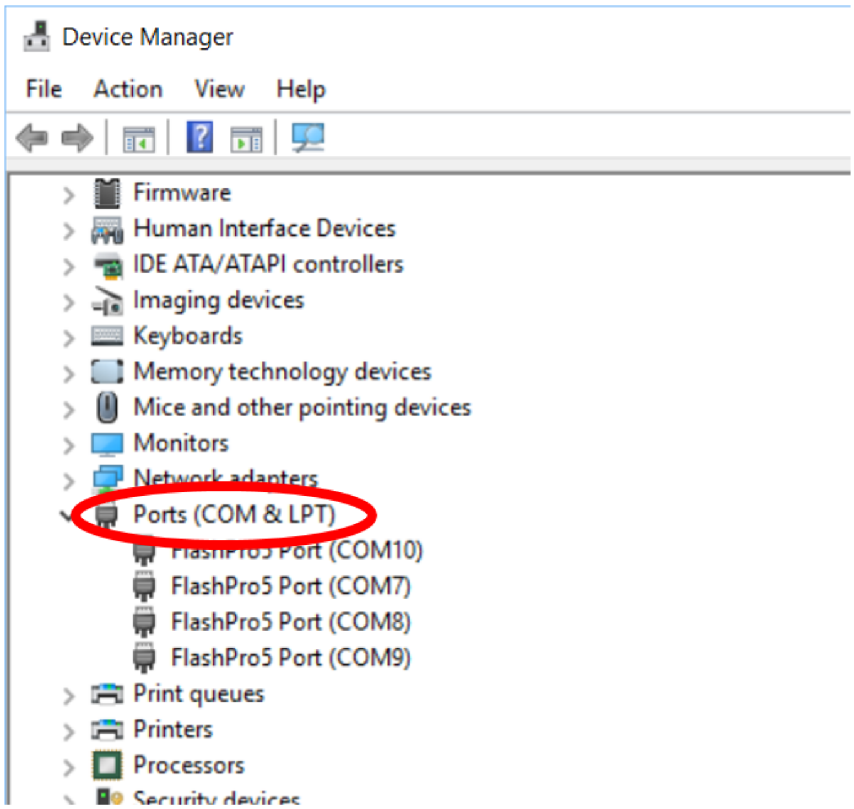


Figure 43 • Location of BAUD Rate Value

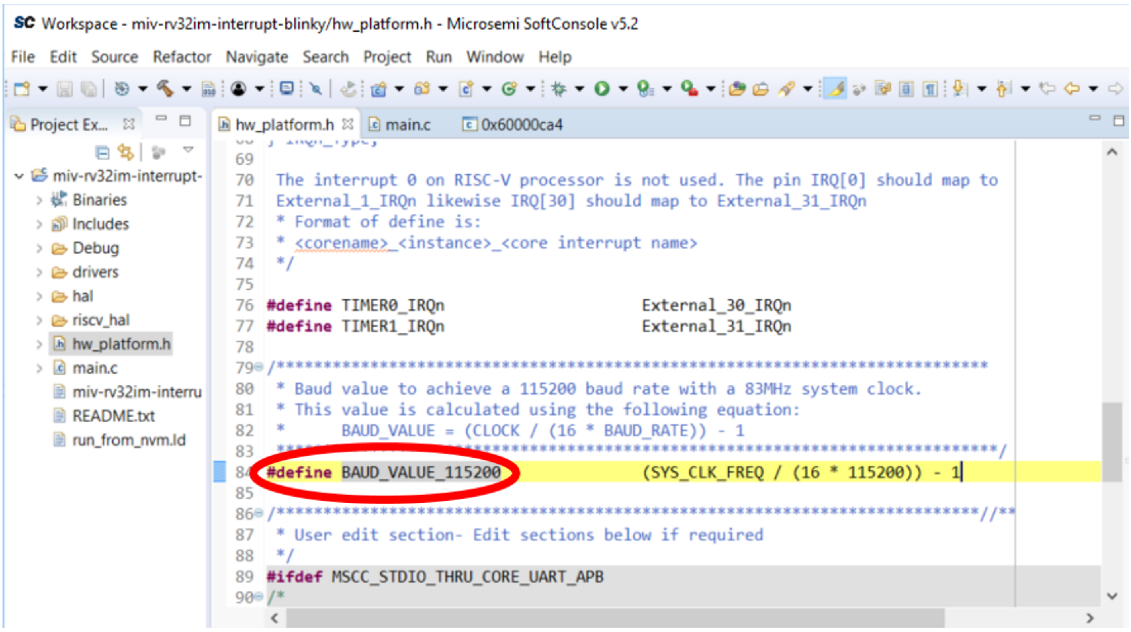


Figure 44 • Opening the Terminal Configurator

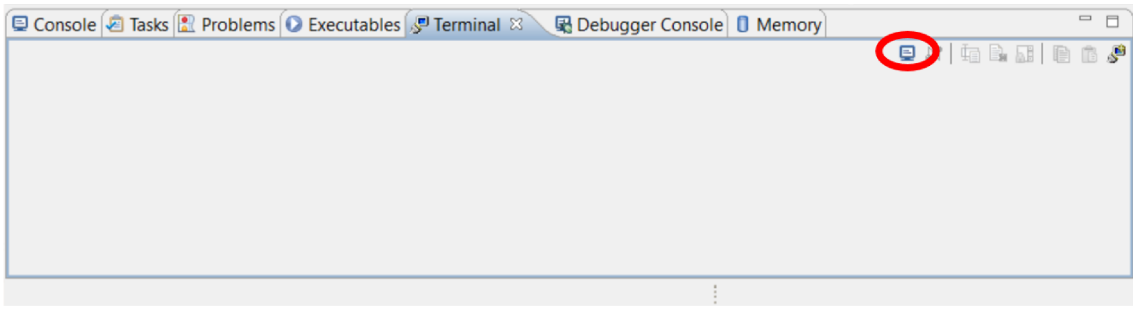


Figure 45 • Configuring the Terminal

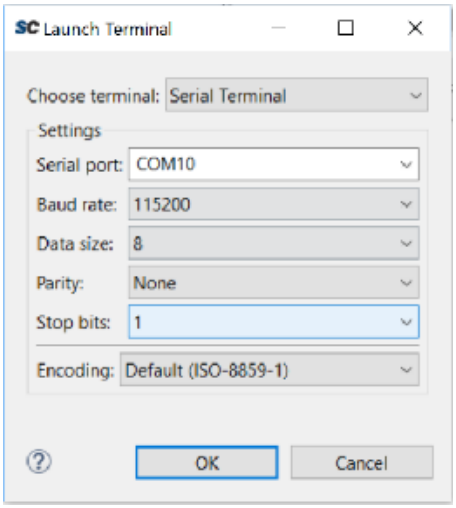


Figure 46 • Serial Output from COM Terminal

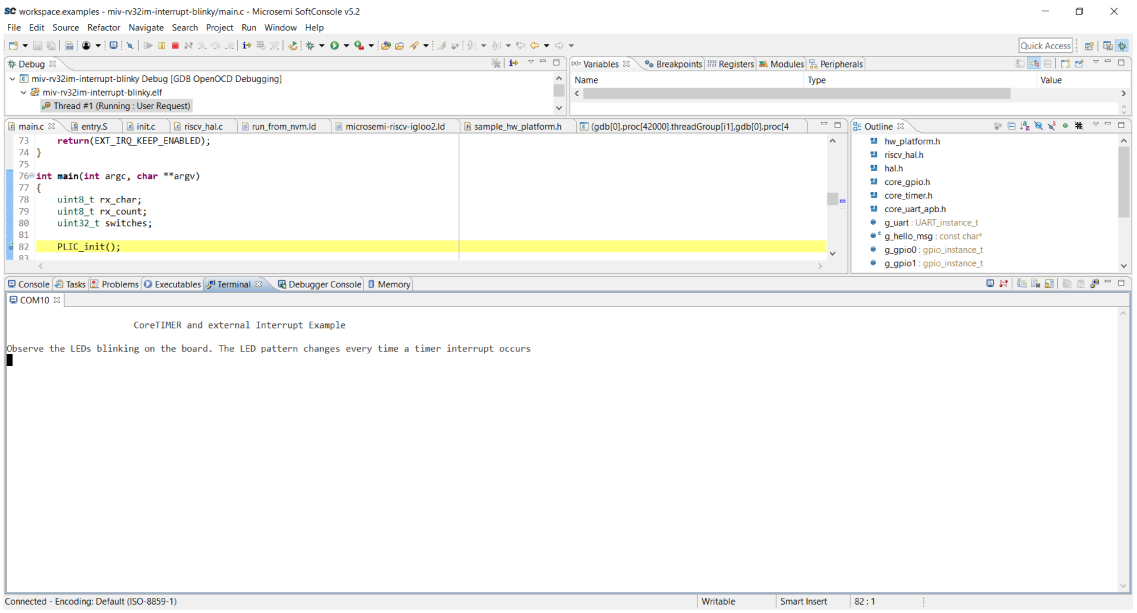
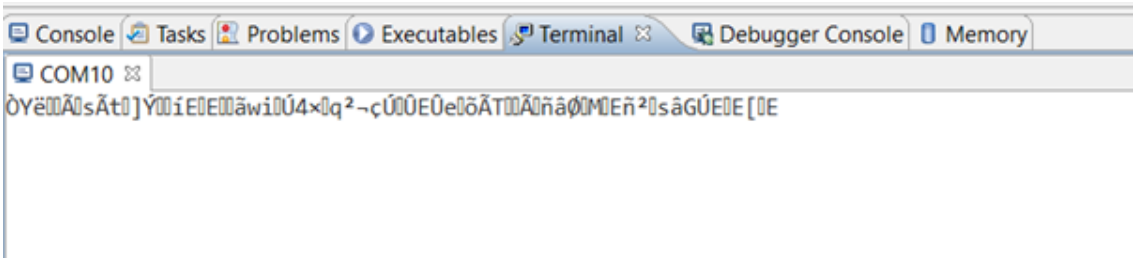


Figure 47 • UART Clock Frequency Issues



Microsemi Headquarters
One Enterprise, Aliso Viejo,
CA 92656 USA
Within the USA: +1 (800) 713-4113
Outside the USA: +1 (949) 380-6100
Sales: +1 (949) 380-6136
Fax: +1 (949) 215-4996
Email: sales.support@microsemi.com
www.microsemi.com

Microsemi, a wholly owned subsidiary of Microchip Technology Inc. (Nasdaq: MCHP), offers a comprehensive portfolio of semiconductor and system solutions for aerospace & defense, communications, data center and industrial markets. Products include high-performance and radiation-hardened analog mixed-signal integrated circuits, FPGAs, SoCs and ASICs; power management products; timing and synchronization devices and precise time solutions, setting the world's standard for time; voice processing devices; RF solutions; discrete components; enterprise storage and communication solutions; security technologies and scalable anti-tamper products; Ethernet solutions; Power-over-Ethernet ICs and midspans; as well as custom design capabilities and services. Microsemi is headquartered in Aliso Viejo, California, and has approximately 4,800 employees globally. Learn more at www microsemi.com.

MSCC-0101-QS-01003-1.00-1018