Design Guide

Mi-V SmartFusion2, IGLOO2, and RTG4 Quick Start Guide

Final

October 2018

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 the steps in setting up the Mi-V AXI cores for use in SmartFusion2, IGLOO2, and RTG4 designs using Libero SoC V11.8. This will include generating a bitstream and downloading the FPGA data to a device, as well as debugging through serial communication in SoftConsole V5.2.

By following the steps outlined in this document, the produced design will have the same functions as those found on the Microsemi Github page. Any components needed for designs can be found in the Libero catalog or on the Microsemi Github page.

Because this guide covers multiple devices, there are some device-specific components and configurations that need to be made. To keep setup as easy as possible, these device-specific components and configurations will be dealt with at the start of the guide. The remaining components and configurations will be generic across device families. The use of the core in its AXI4 mode is shown in this guide. This is for illustrative purposes, and it is recommended to use the core in its AXI3 configuration.

3 Creating the Libero Project

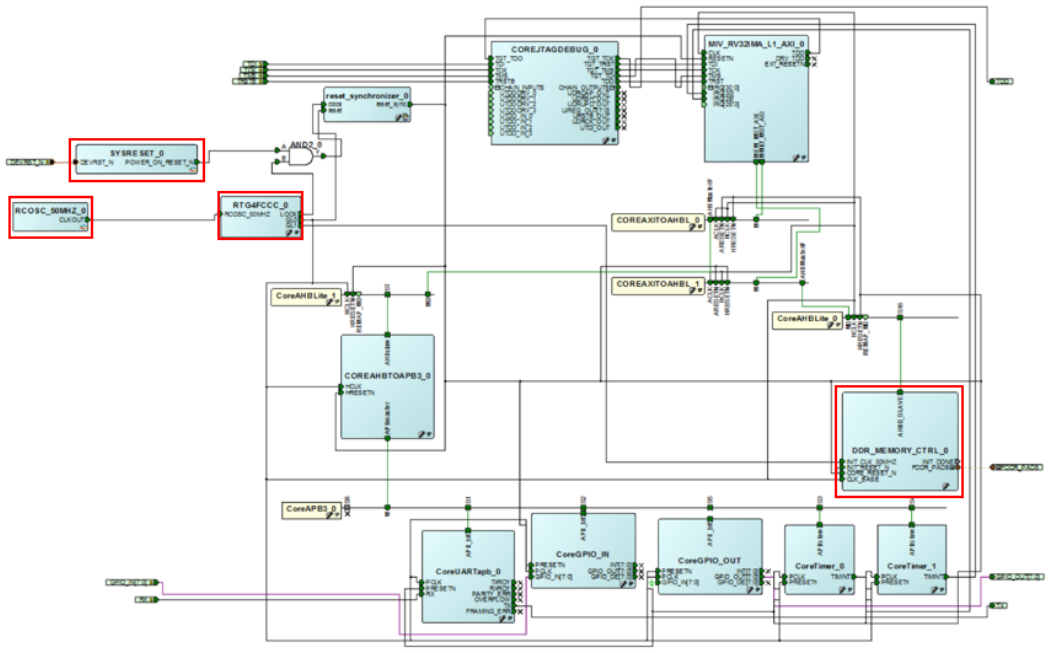
This section describes how to create the Libero project.

1. Open Libero and select New.
2. Enter a project name and project folder location and then click Next (there will be several subdirectories, so the path should be kept to a minimum to ensure path lengths remain 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 or not 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 Placing and Configuring Components

The following illustration shows the design segment that can be produced for the Mi-V CPU in its AXI3 configuration, using all of the component configurations except for the AXI4 Interconnect and the AXI to AHB cores. Components highlighted in red are device-specific. For set-up instructions, see Section 2.2 (see page 5) or Section 2.3 (see page 9), depending on your device family.

Figure 1 • Finished Core Design Using the AXI3 Interface for an RTG4



3.2 SmartFusion2 and IGLOO2 Specific Configurations

Because multiple device families are covered, some device-specific components must be configured. This section will cover those necessary configurations for a SmartFusion2 or IGLOO2 project. If you are using an RTG4, you do not need these components.

The following table shows the CoreAXI4Interconnect.

Table 1 • CoreAXI4Interconnect

Component	Quantity	Version	Device
Chip oscillators (OSC)	1	V2.0.101	SF2, IG2
CoreAHBLSRAM	1	V2.2.104	SF2, IG2
CCC	1	V2.0.201	SF2, IG2

3.2.1 OSC Configuration SmartFusion2 and IGLOO2

This section shows the OSC configuration of SmartFusion2 and IGLOO2.

Figure 2 • SF2 & IG2 OSC Default Configuration

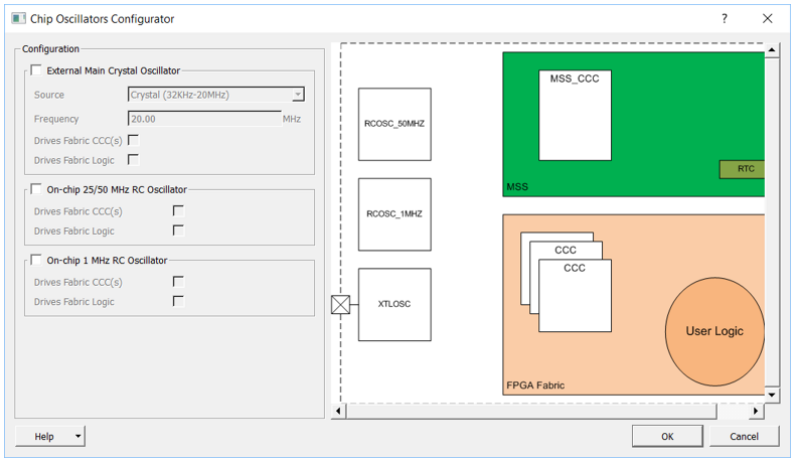
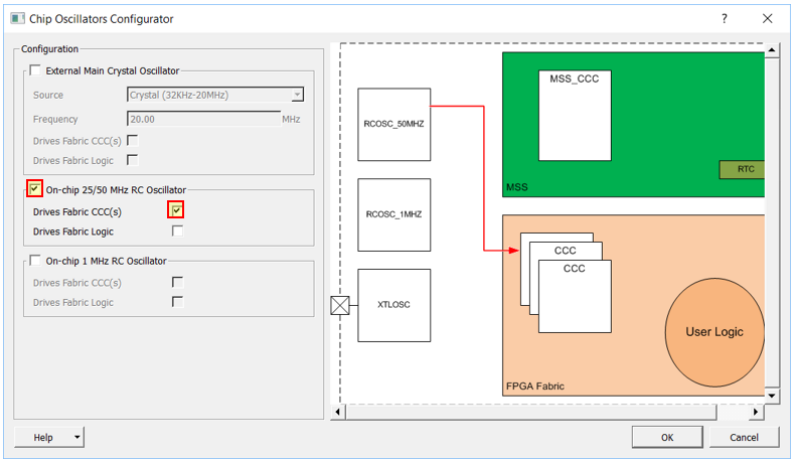


Figure 3 • SF2 & IG2 OSC Required Configuration



3.2.2 CCC Configuration SmartFusion2 and IGLOO2

This section shows the CCC configuration of SmartFusion2 and IGLOO2.

Figure 4 • SF2 & IG2 CCC Default Configuration

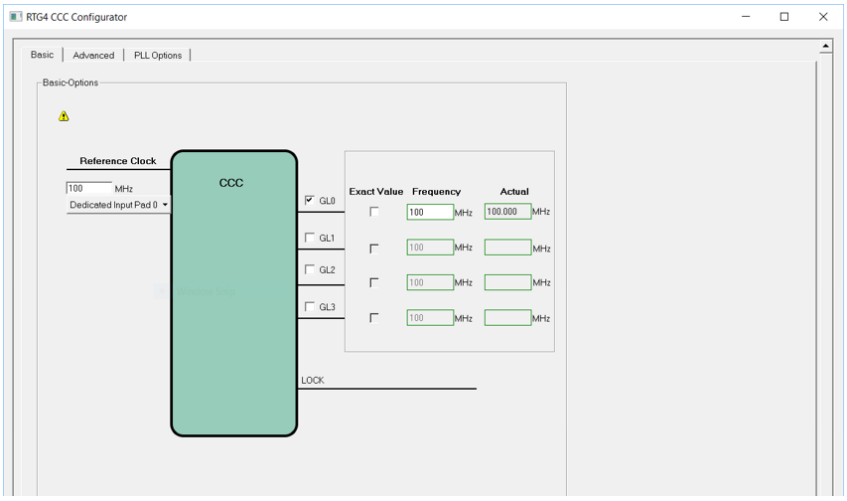
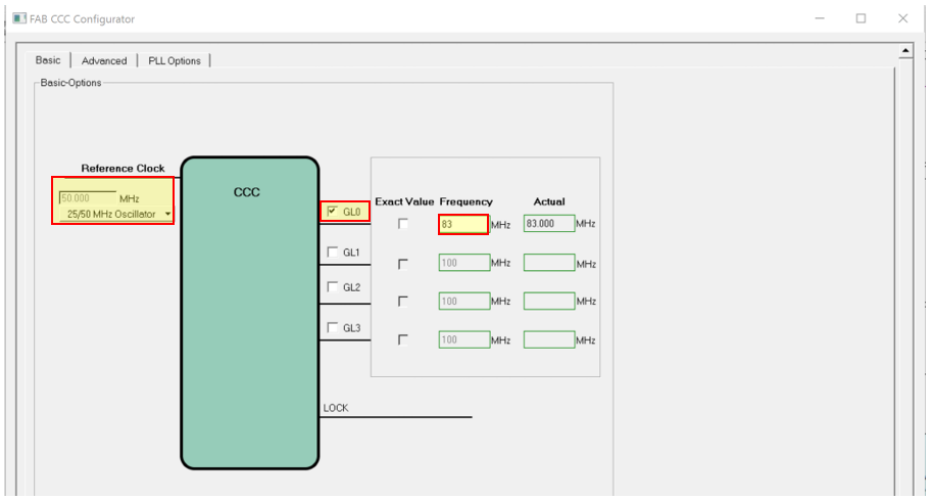


Figure 5 • SF2 & IG2 CCC Required Configuration



3.2.3 LSRAM Configuration SmartFusion2 and IGLOO2

This section shows the LSRAM configuration of SmartFusion2 and IGLOO2.

Figure 6 • SF2 & IG2 LSRAM Default Configuration

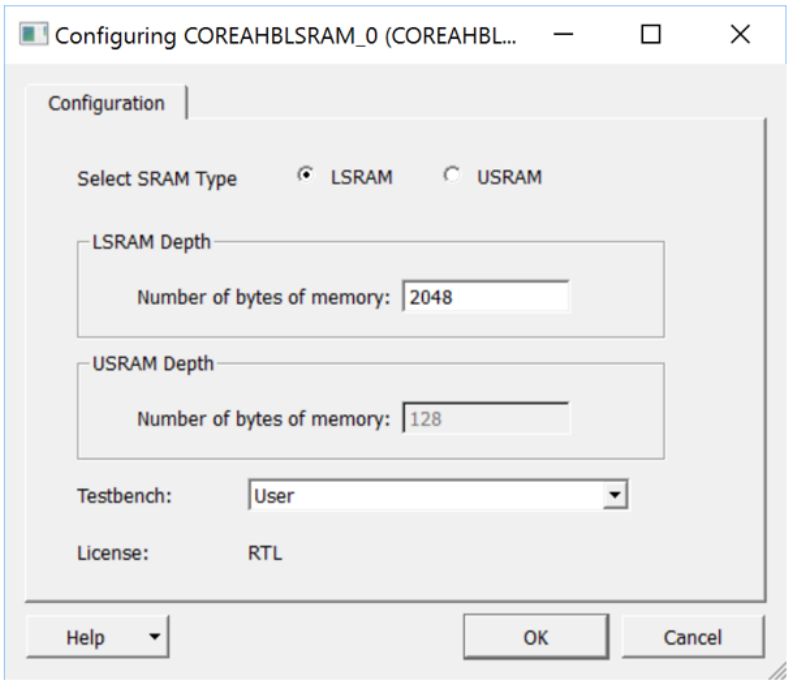
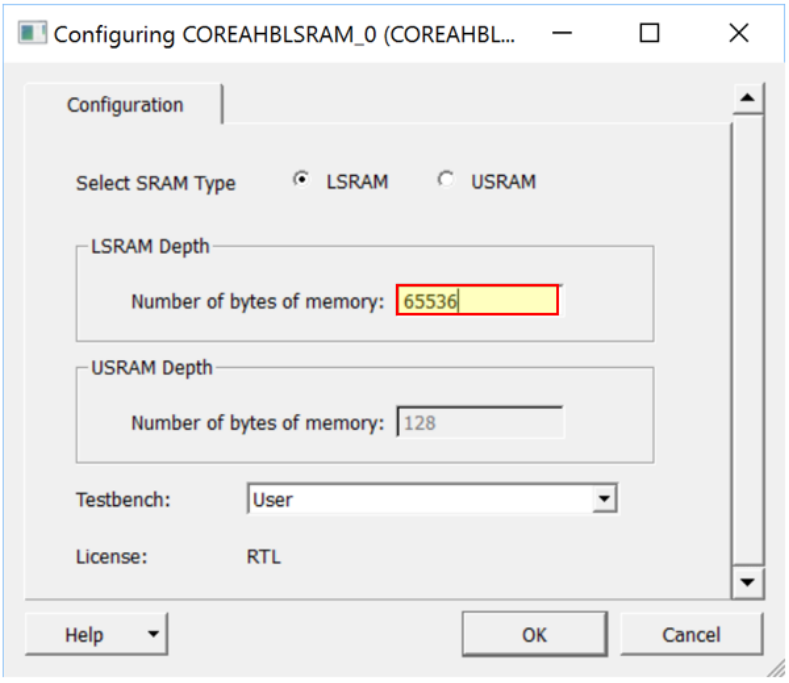


Figure 7 • SF2 & IG2 LSRAM Required Configuration



3.3 RTG4 Specific Configurations

Because multiple device families are covered in this document, there are several device-specific components that must be configured. This section will cover those necessary configurations for an RTG4 project. If you are using a SmartFusion2 or IGLOO2, you do not need these components.

Also note that the AXI4Interconnect is incompatible with the RTG4, meaning that the Mi-V CPU can only be used in the AXI3 configuration.

The following table lists the necessary additional components.

Component	Quantity	Version	Device
RTG4 clock conditioning circuit	1	V1.1.217	RTG4
RTG4 DDR memory controller with initialization	1	V1.0.102	RTG4

3.3.1 RTG4_CCC Configuration

This section shows the CCC configuration for the RTG4.

Figure 8 • RTG4_CCC Configuration

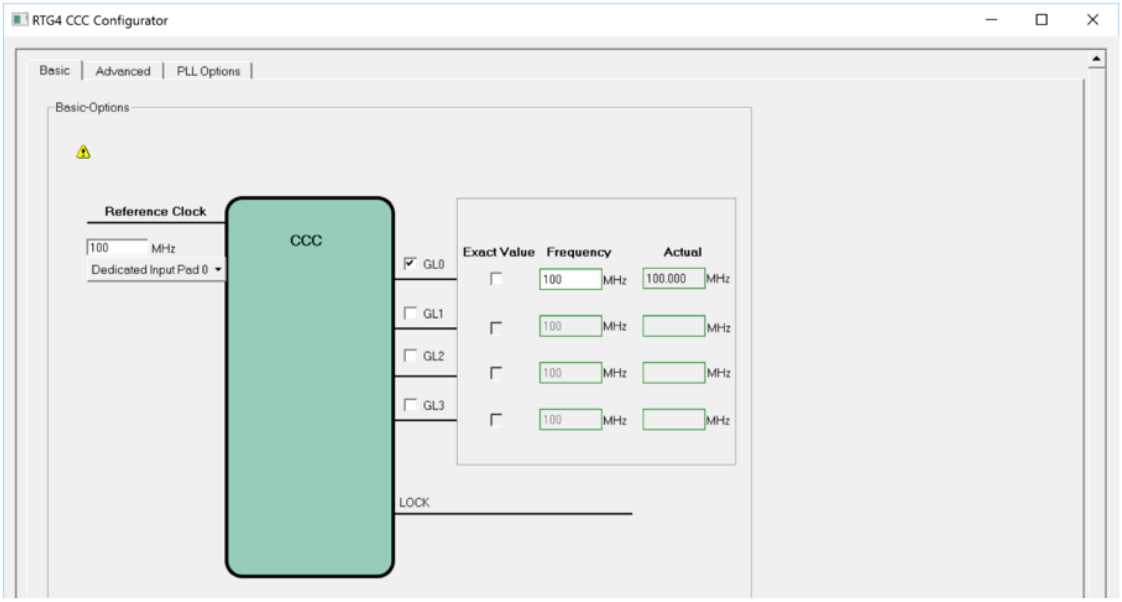
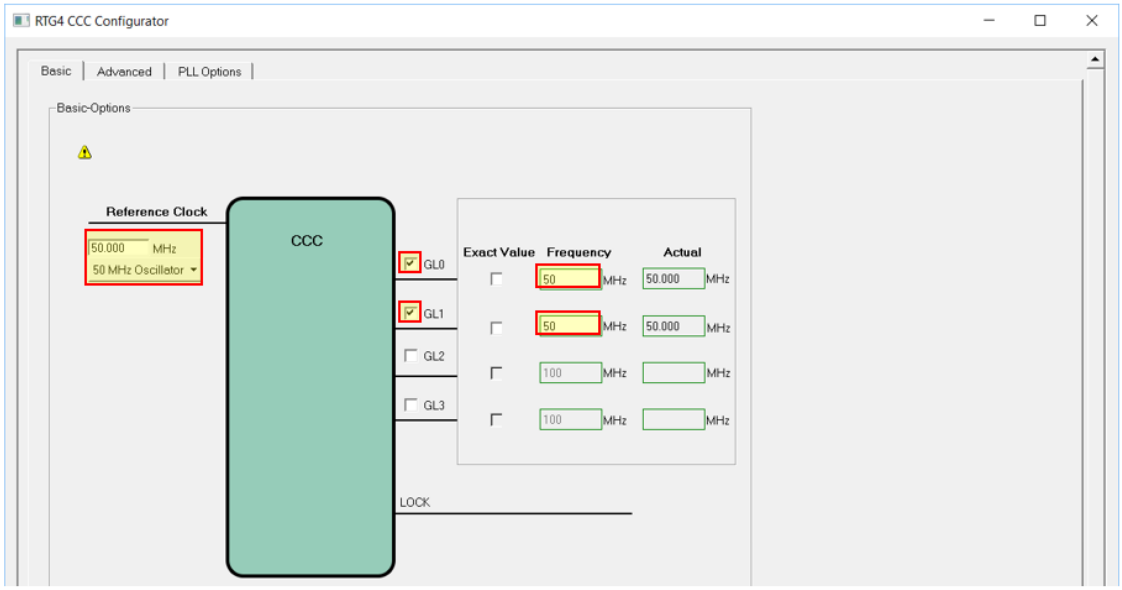


Figure 9 • RTG4 CCC Required Configuration



3.3.2 DDR Configuration RTG4

This section shows the DDR configuration for the RTG4.

Figure 10 • RTG4 DDR Default Configuration - General

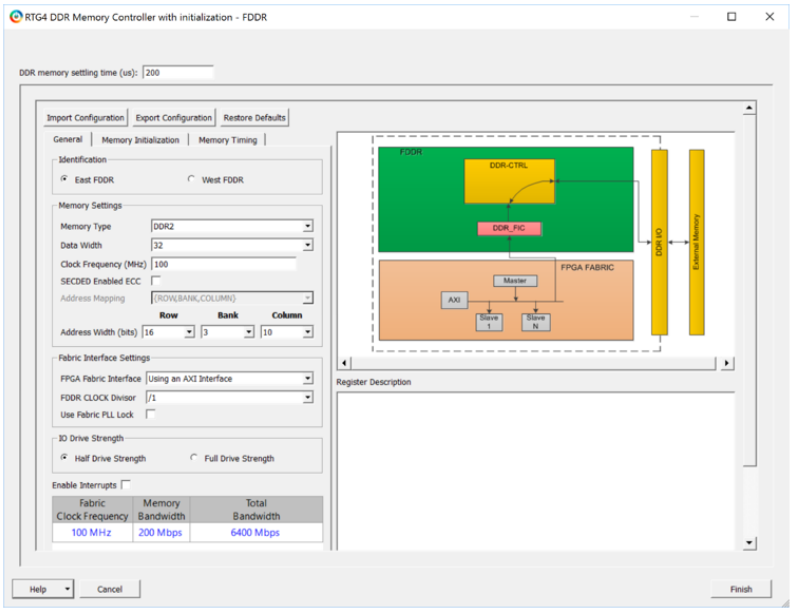


Figure 11 • RTG4 DDR Required Configuration - General

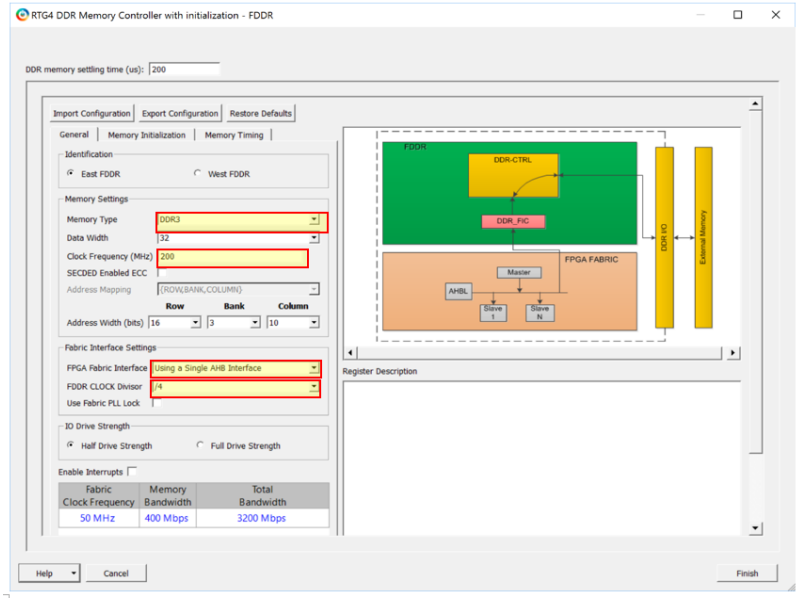


Figure 12 • RTG4 DDR Default Configuration - Memory Initialization

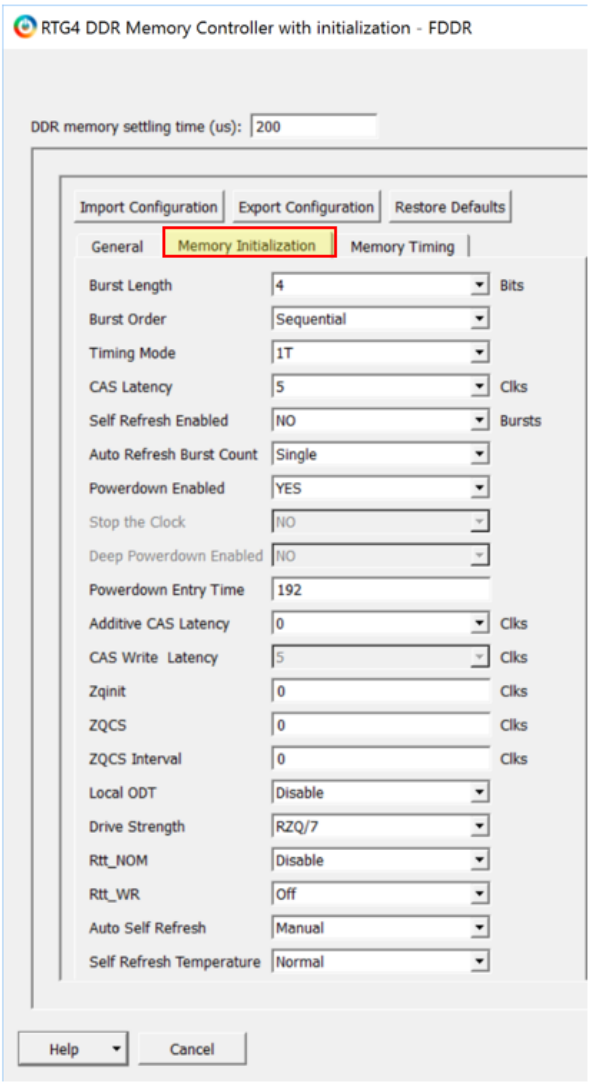


Figure 13 • RTG4 DDR Required Configurati Memory Initialization

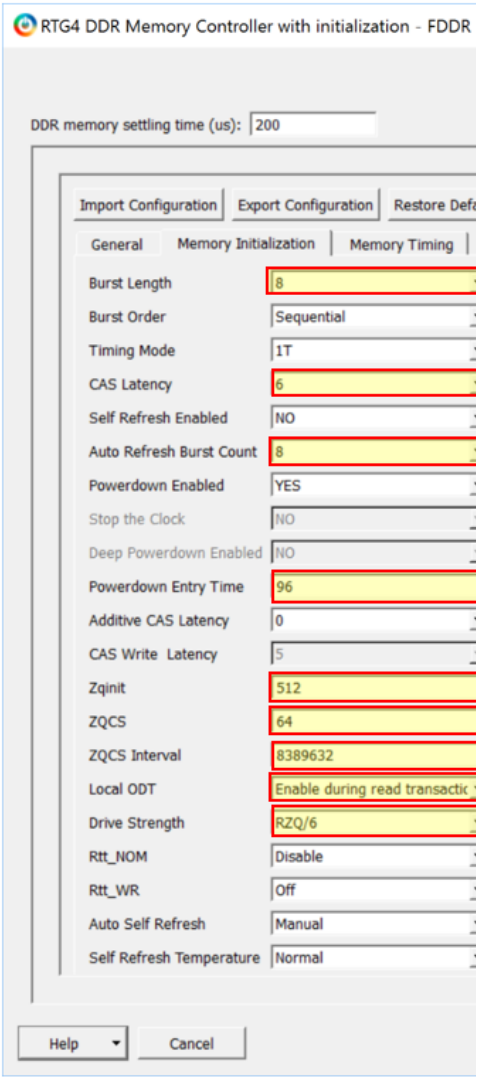


Figure 14 • RTG4 DDR Default Configuration - Memory Timing

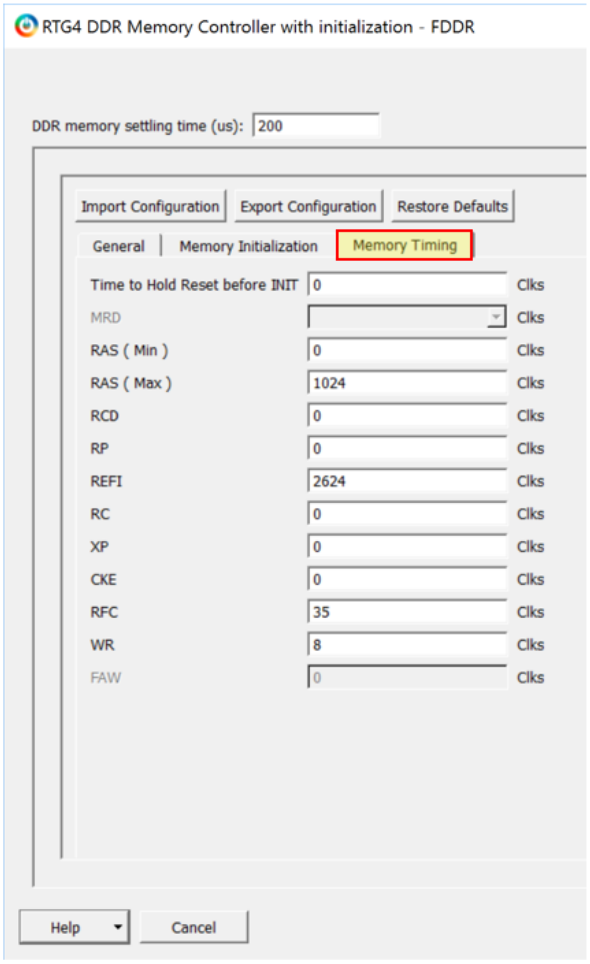
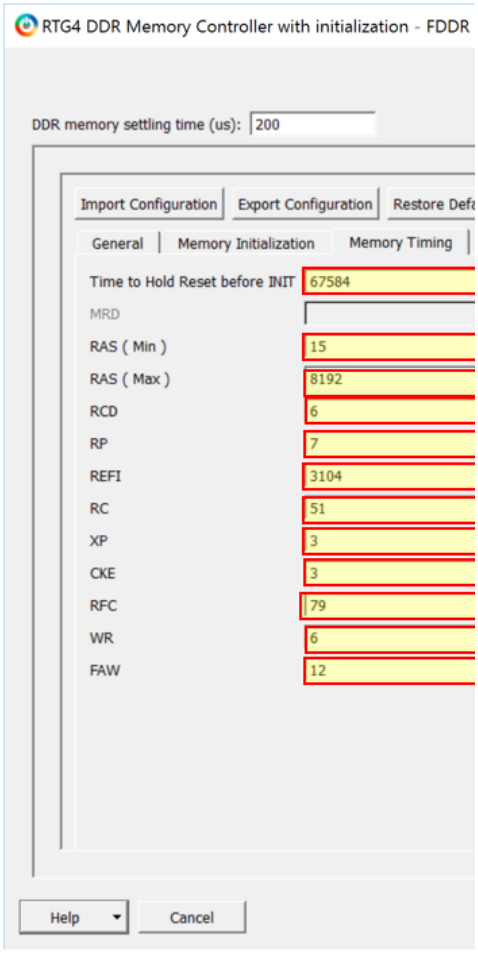


Figure 15 • RTG4 DDR Required Configurati Memory Timing



3.4 Non-Device Specific Components

The remaining components covered in this section can be used with all device families.

Component	Quantity	Version	Device
CoreJTAGDebug	1	V2.0.100	All
MIV_RV32IMA_L1_AXI	1	V2.0.100	All
SYSRESET	1	V1.0	All
CoreAXITOAHBL	2	V3.3.101	All
CoreAHBLite	2	V5.3.200	All
And2	1	V1.0	All
Reset synchronizer (code follows)	1	N/A	All

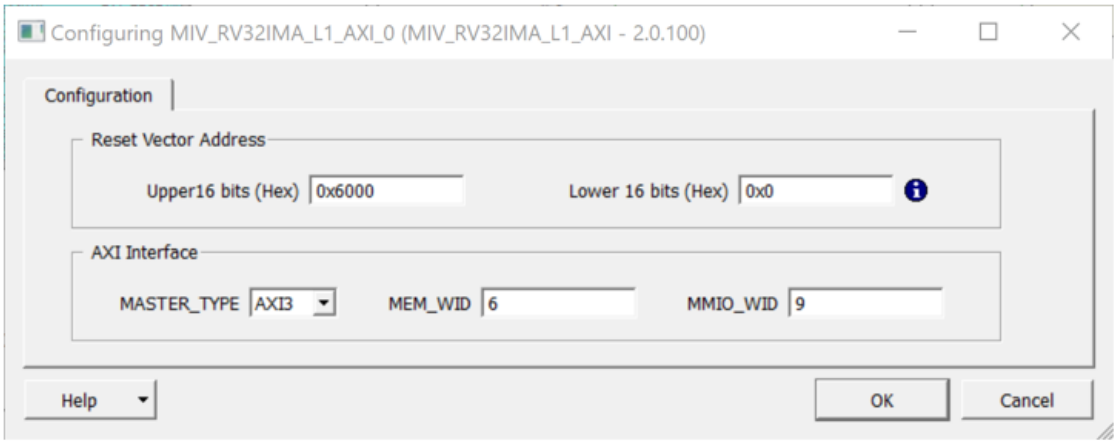
Set up the components as follows:

1. COREJTAGDEBUG: no configuration required
2. And2: no configuration required
3. Reset synchronizer: code included

3.4.1 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 to open the configurator. The default configuration is AXI3.

Figure 16 • Mi-V Default AXI3 Configuration



3.4.2 Reset_Synchronizer Configuration

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

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

1. Click File -> New -> HDL.

Figure 17 • Creating HDL File

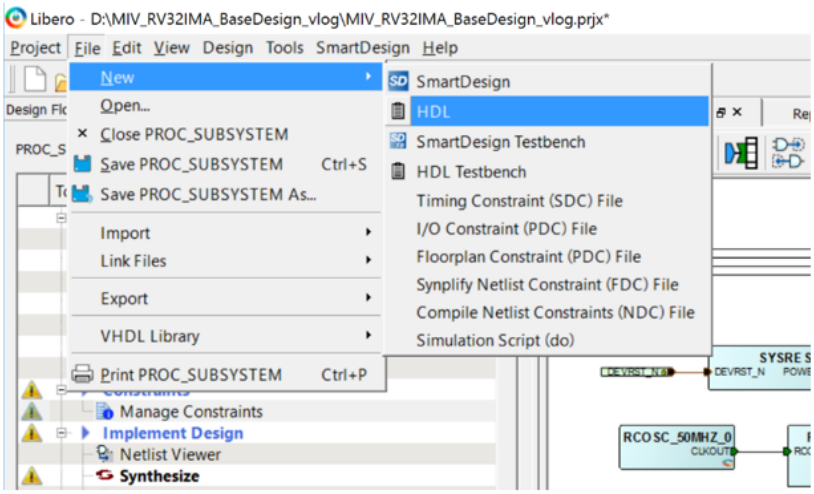
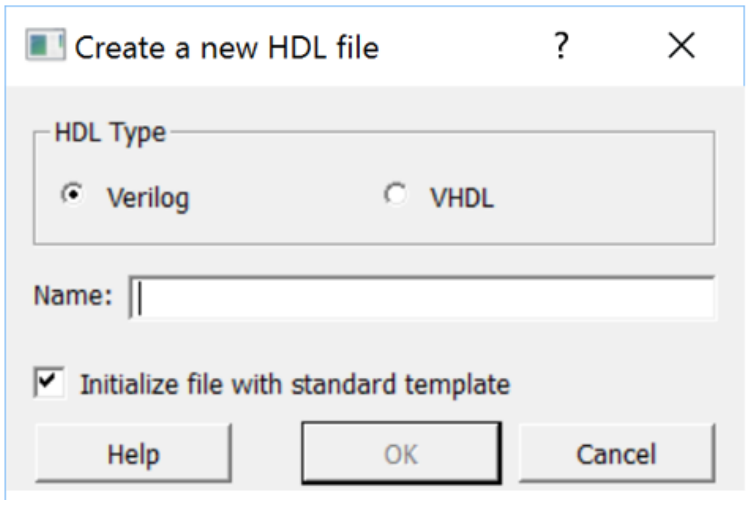


Figure 18 • Creating HDL File



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
```

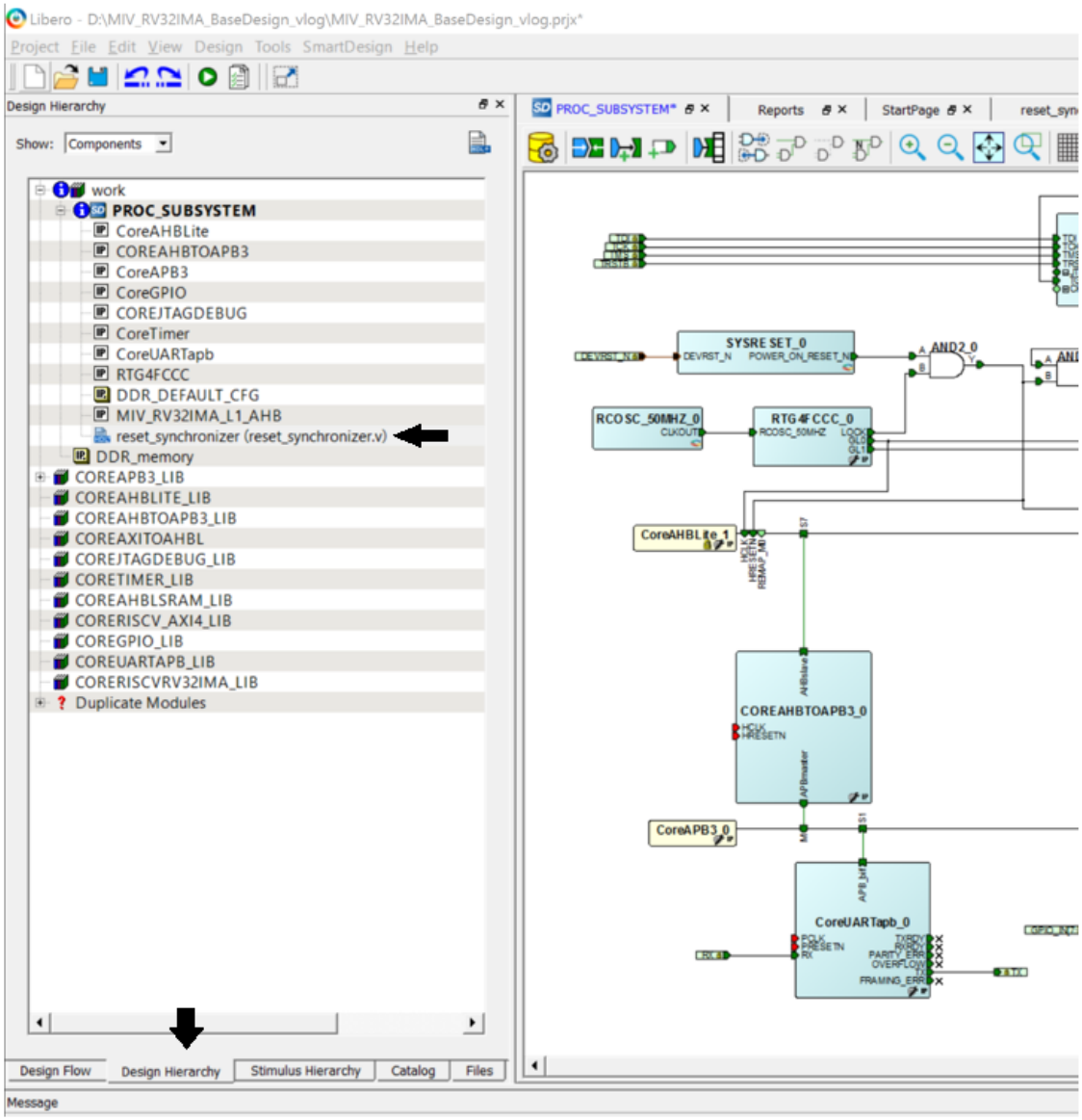
4. Save the file, and return to your smart design.

5. Open the Design Hierarchy tab.

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

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

Figure 19 • Adding HDL File to SmartDesign



3.4.3 COREAXITOAHBL x2 Configuration

The following section shows the COREAXITOAHBL x2 configuration.

Figure 20 • CoreAXIToAHBL - Default Configuration/AXI3 Configuration

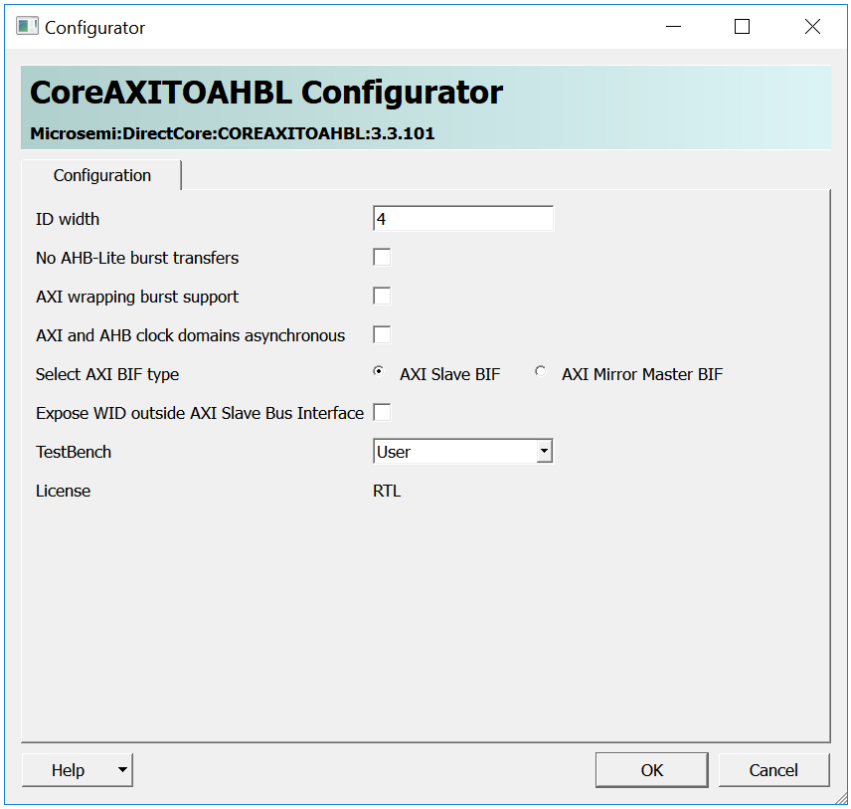
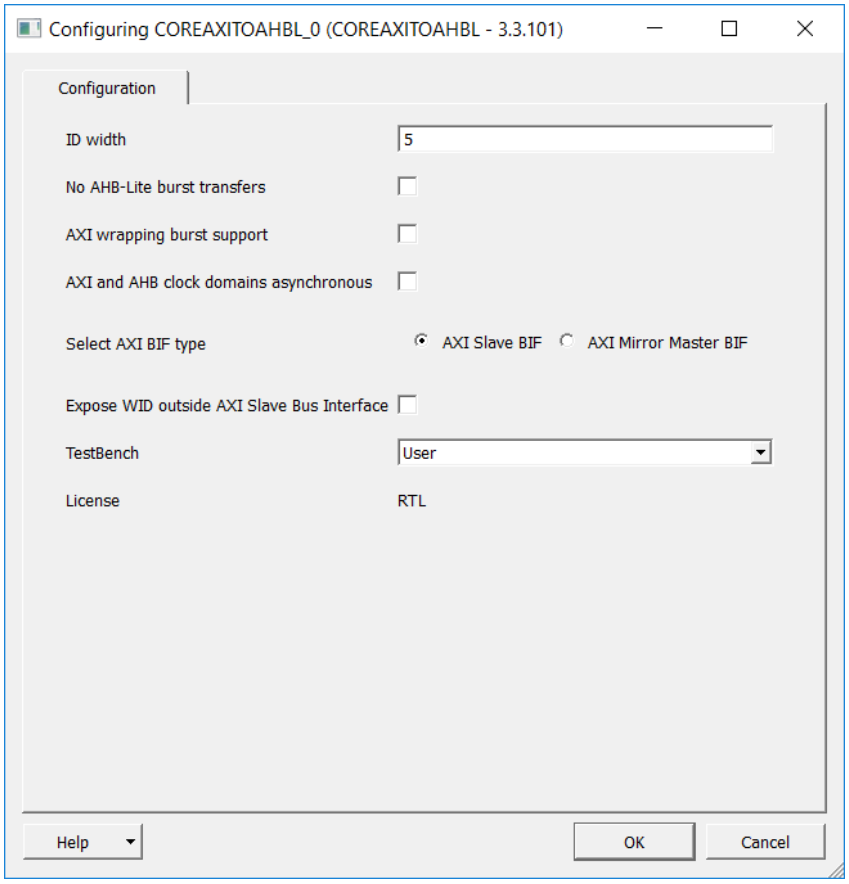


Figure 21 • CoreAXIToAHBL - Required Configuration for AXI4



3.4.4 CoreAHBLite_0 Configuration

The following shows the CoreAHBLite_0 configuration.

Figure 22 • CoreAHBLite_0 - Default Configuration

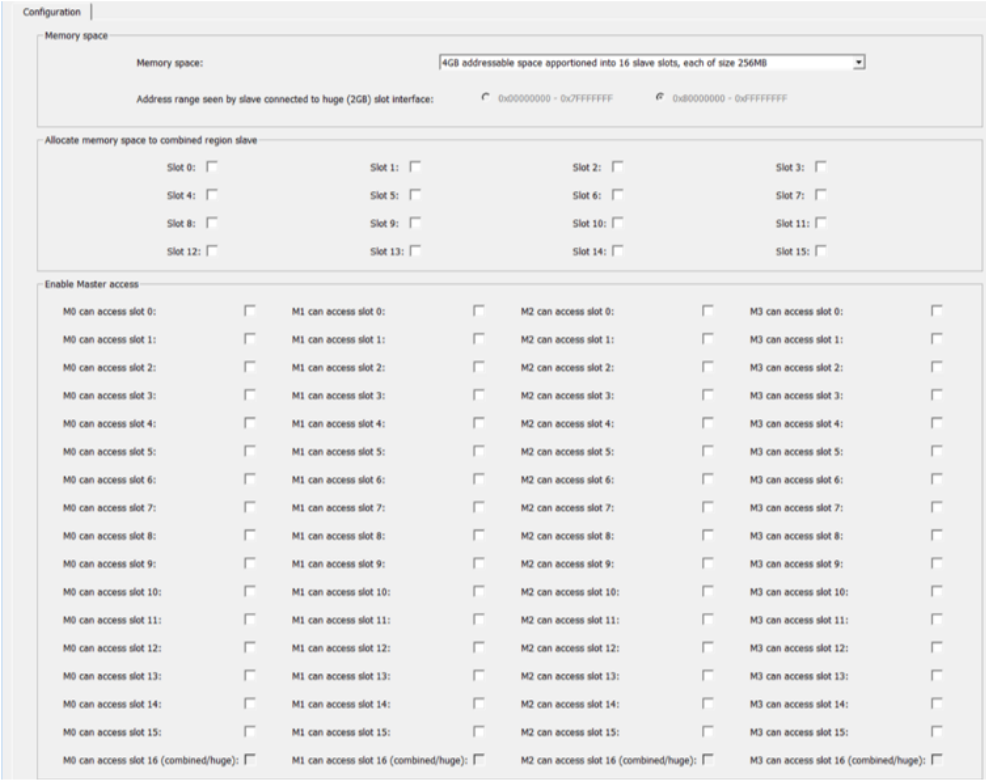
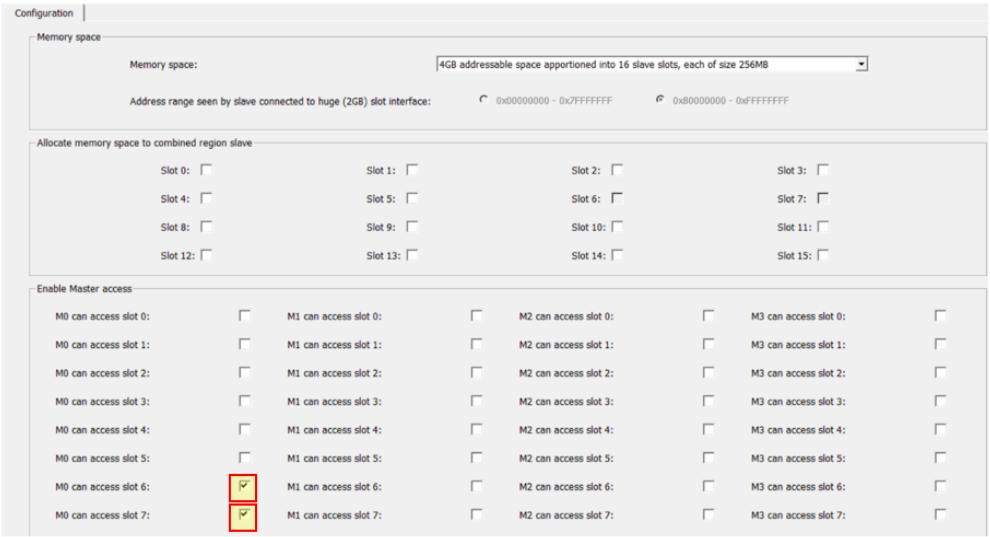


Figure 23 • CoreAHBLite_0 - Required Configuration



3.4.5 CoreAHBLite_1 Configuration

The following shows the CoreAHBLite_1 configuration.

Figure 24 • CoreAHBLite_1 - Default Configuration

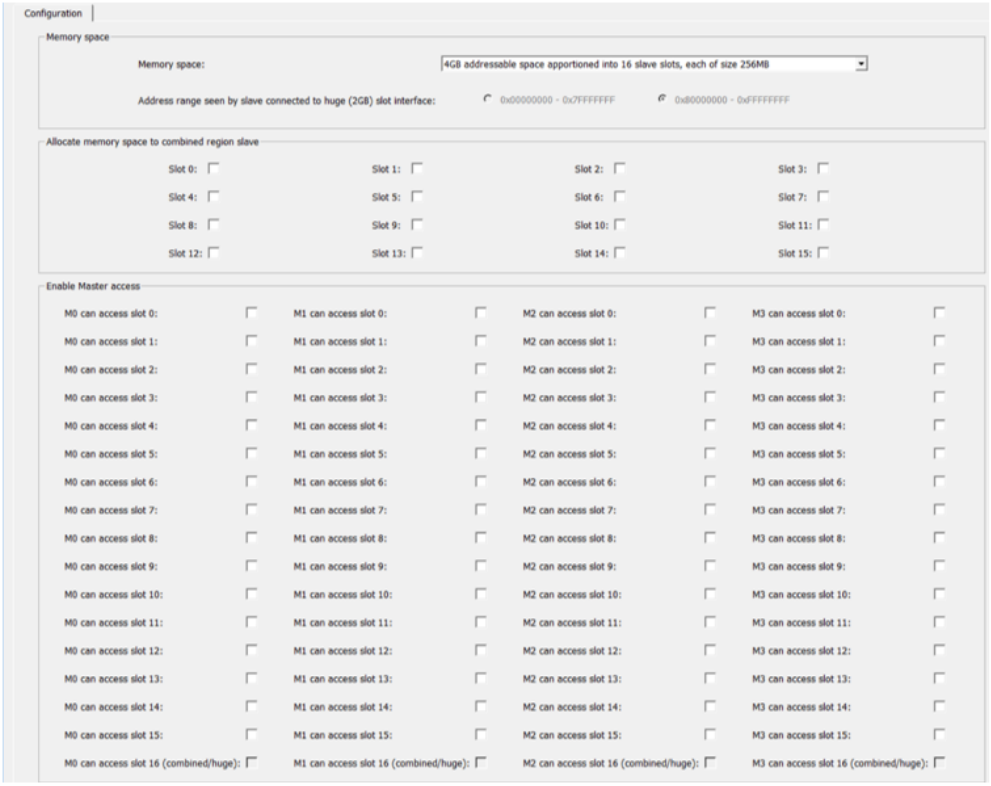
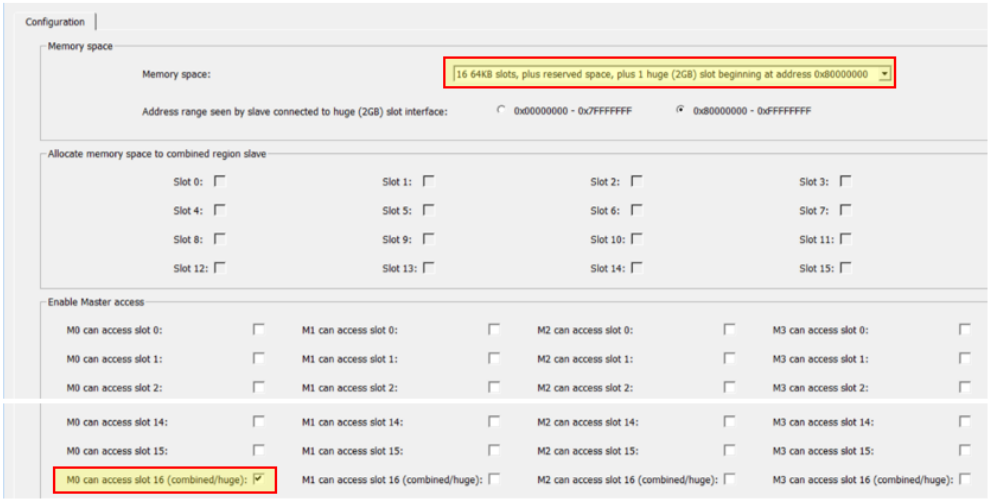


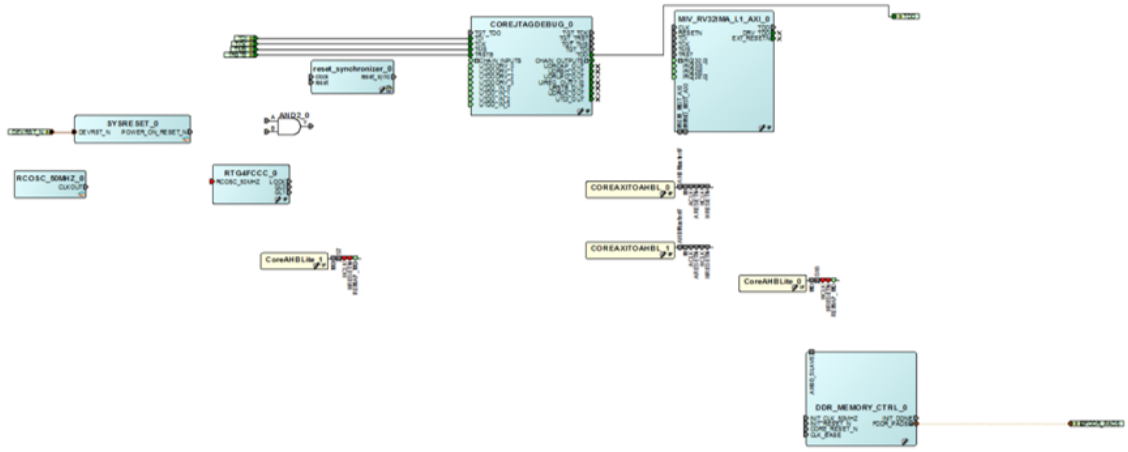
Figure 25 • CoreAHBLite_1 - Required Configuration



3.5 Connecting Components

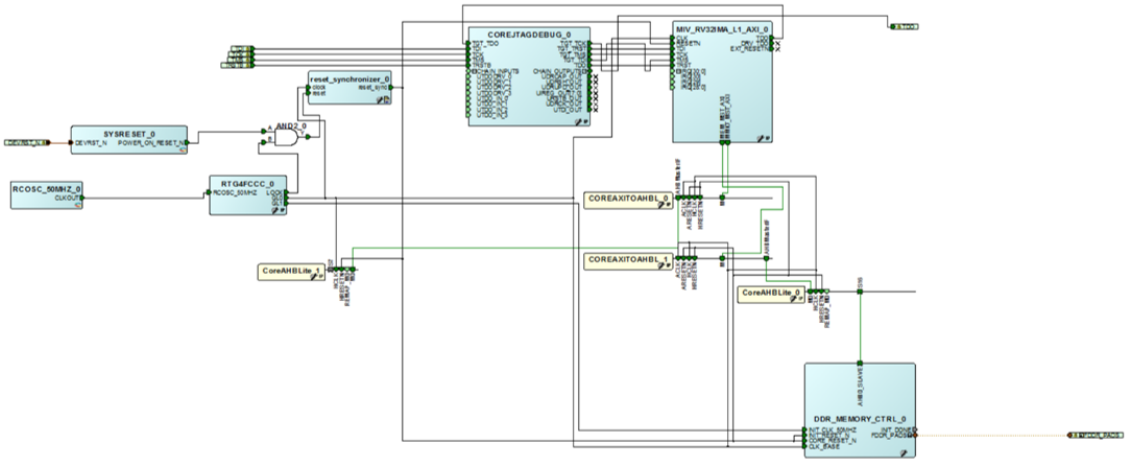
Once you have placed and configured all the components, your design layout should be similar to the following illustration.

Figure 26 • Core Design Layout With No Connections



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

Figure 27 • Core Design Layout With Connections



The next page contains a table of all port connections for the design in the following form:

Device + port name → Device + port name

The final step is to connect internal wires and external ports.

The following table shows the port connections.

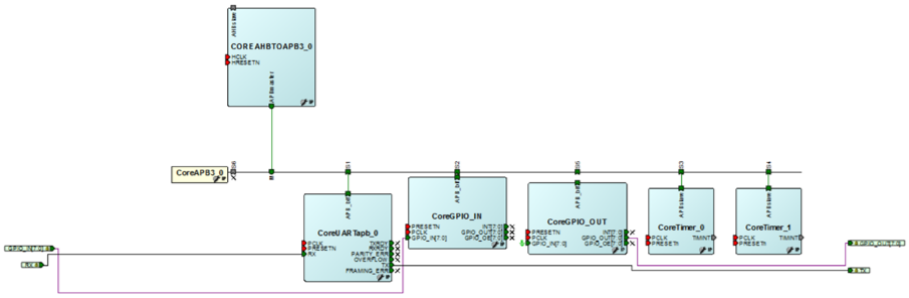
Table 2 • Table of Connections for Core Design

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
SYSRESET	POWER_ON_RESET_N	AND2	B
OSC_0	RCOSC_25_50MHZ_CCC_OUT	FCCC	RCOSC_25_50MHZ_CCC_IN
(RTG4 ONLY)	GL1	DDR_MEMORY_CTRL_0	CLK_BASE
	LOCK	AND2_0	A
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*	
Mi_V_0	DEBUG_DMACTIVE	Output port	DM_ACTIVE
	TDO	COREJTAGDEBUG	TGT_TDO
	MEM_MST_AXI*	COREAXITOAHBL_1_0	AXISlaveIF
	MMIO_MST_AXI*	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_1_0*	AHBMasterIF	CoreAHBL_1	AHBmmaster0
CoreAHBL_1	AHBmslave8	LSRAM_0 / DDR_MEMORY_CTRL_0	AHBSlaveInterface

3.6 Peripheral Set Up

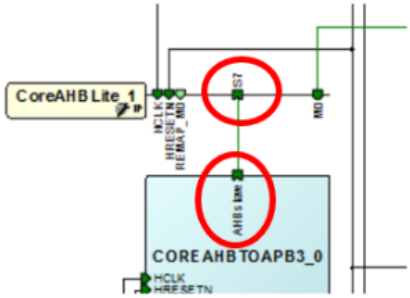
To set up peripherals for use with core, follow these instructions. This setup will be done in the same project as the core and connected to port AHBmslave7 of CoreAHBL_0. The section of the design that will be created is shown in the following illustration.

Figure 28 • Peripherals Design



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

Figure 29 • Connecting Peripheral and Core Designs



To begin, drag components to the 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

3.6.1 CoreAPB3 Configuration

The following section details the core APB3 configuration.

Figure 30 • CoreAPB3 - Default Configuration

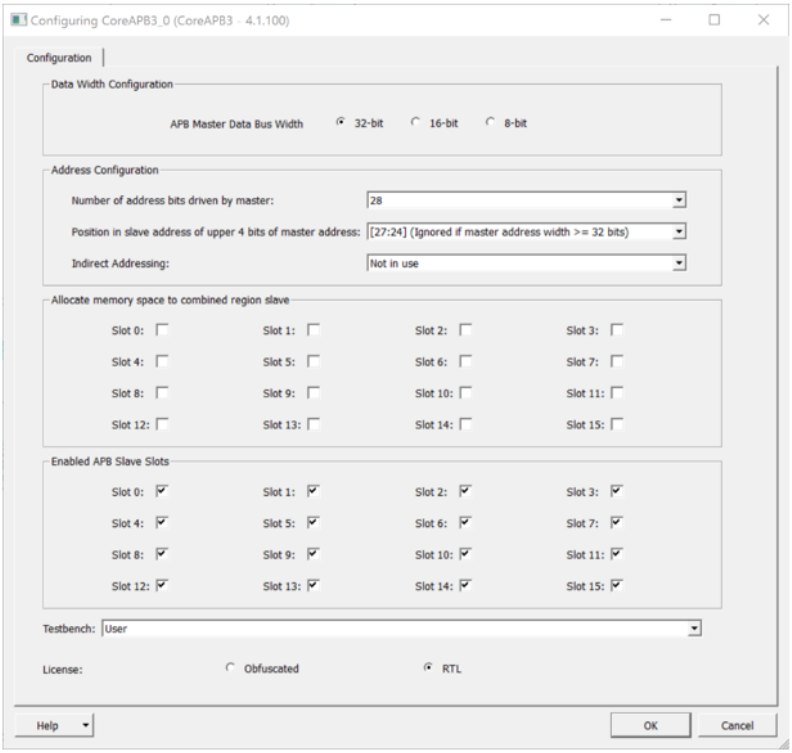
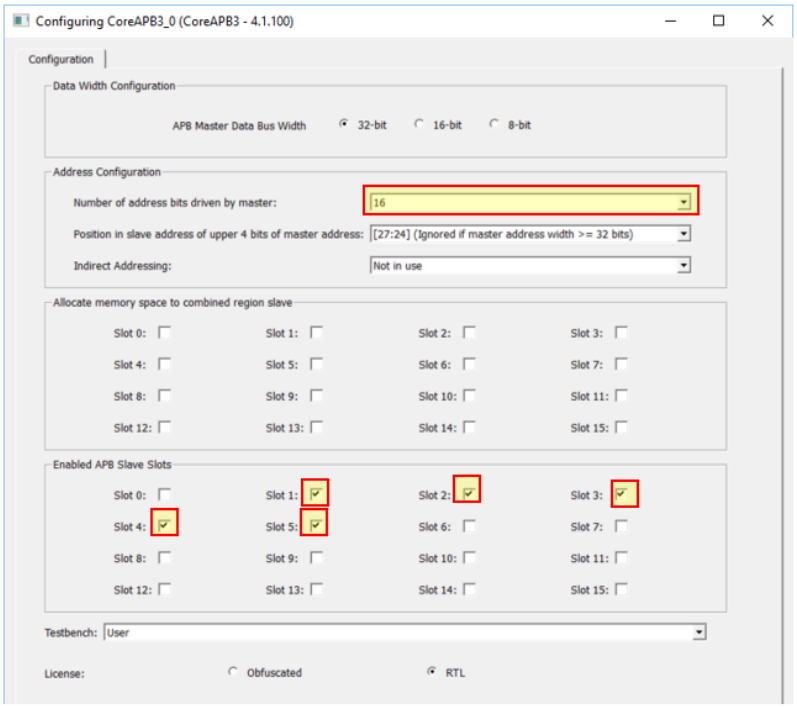


Figure 31 • CoreAPB3 - Required Configuration



3.6.2 CoreGPIO_IN Configuration

The following section details the CoreGPIO_IN configuration.

Figure 32 • CoreGPIO_IN - Default Configuration

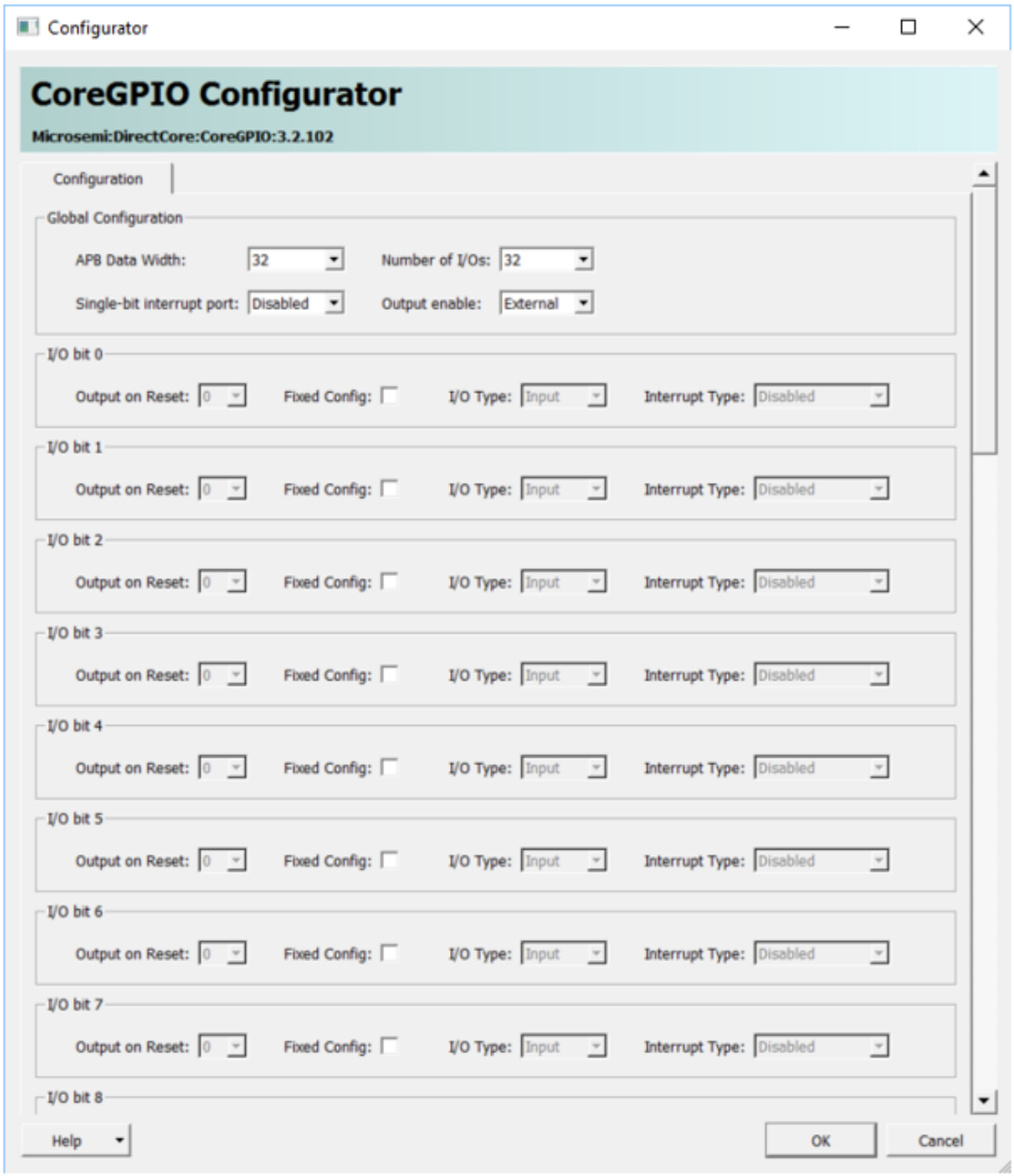
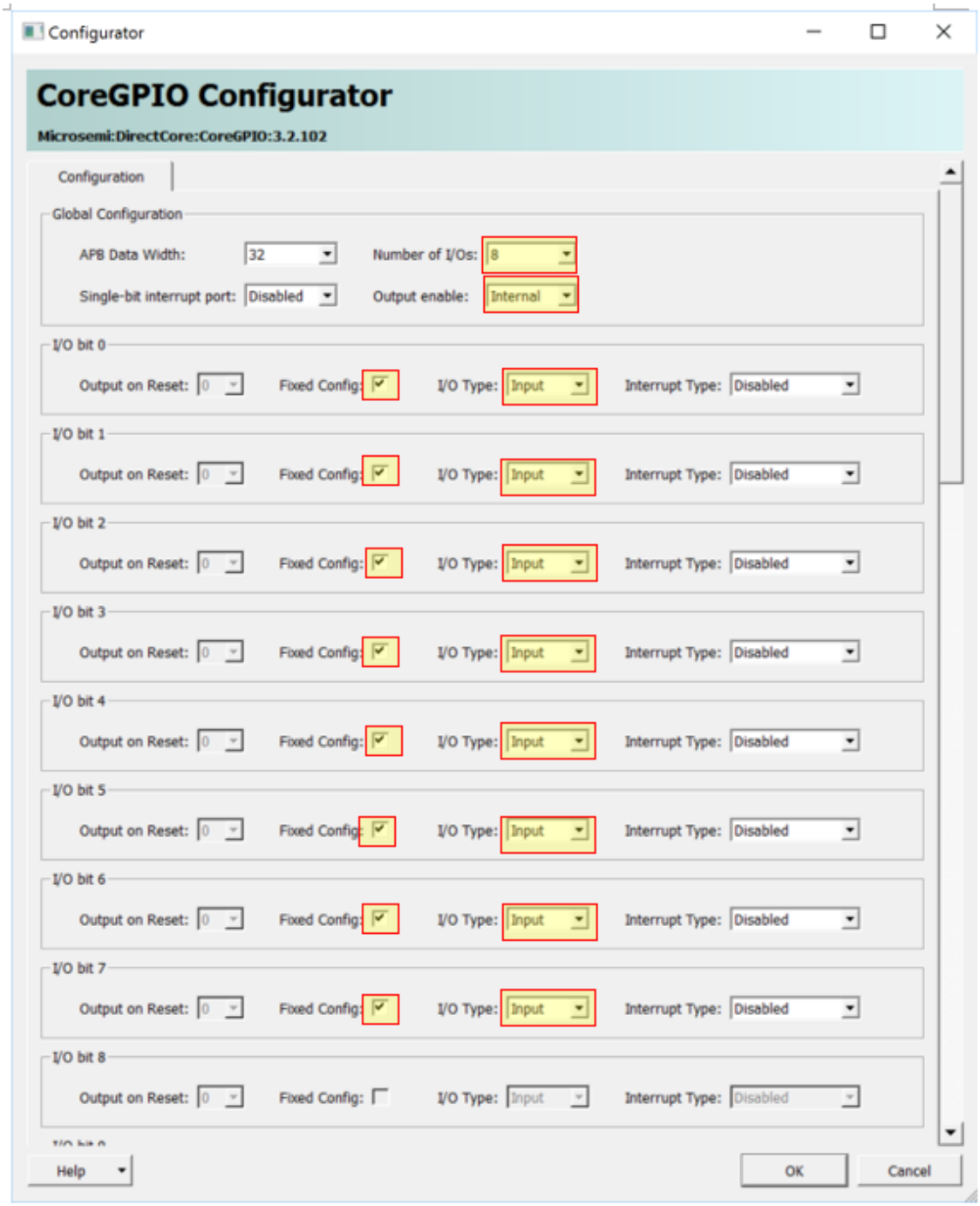


Figure 33 • CoreGPIO_IN - Required Configuration



Note: For the IGLOO2 devices, only two GPIO_in connections are needed. Set the Number of I/Os field to 2 instead of 8.

3.6.3 CoreGPIO_OUT Configuration

The following section details the CoreGPIO_OUT configuration.

Figure 34 • CoreGPIO_OUT - Default Configuration

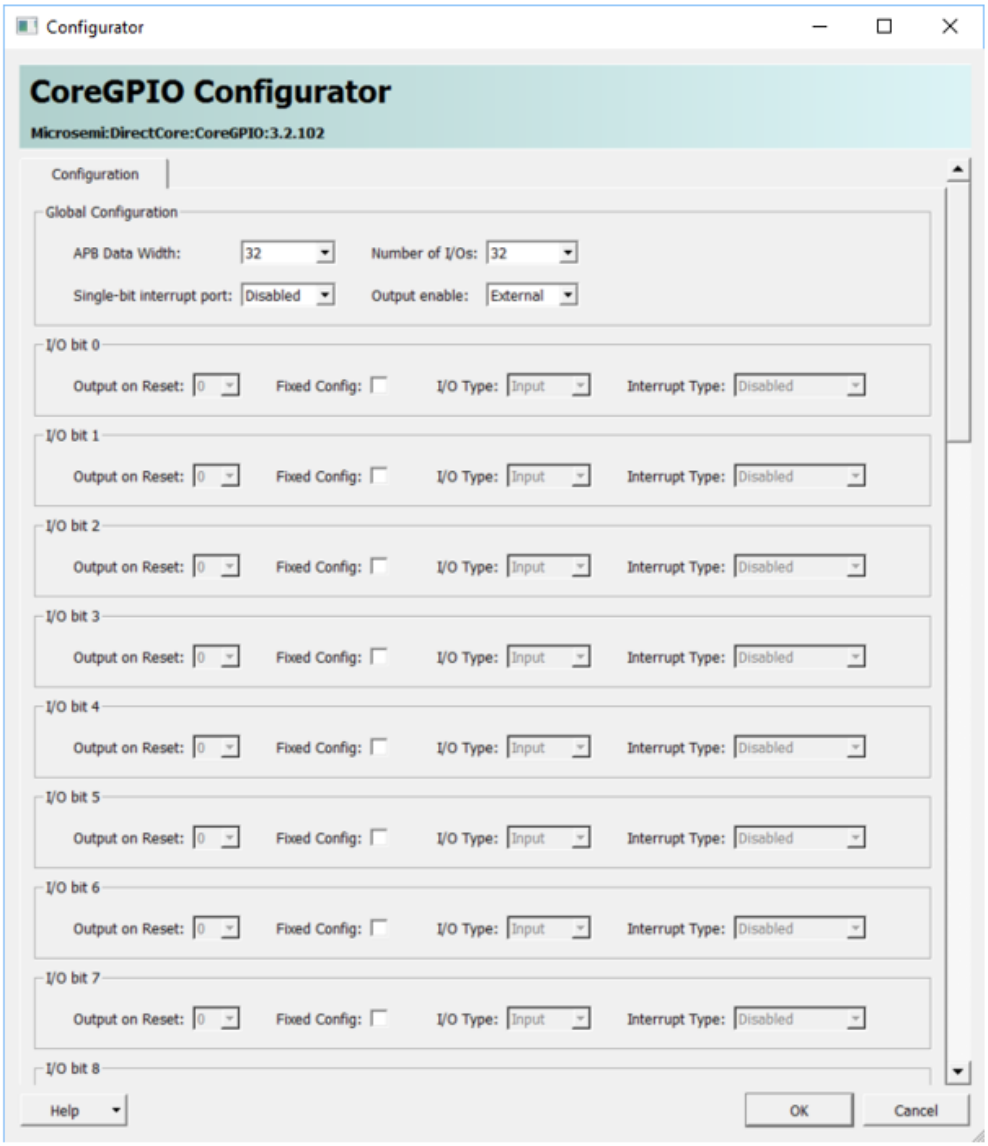
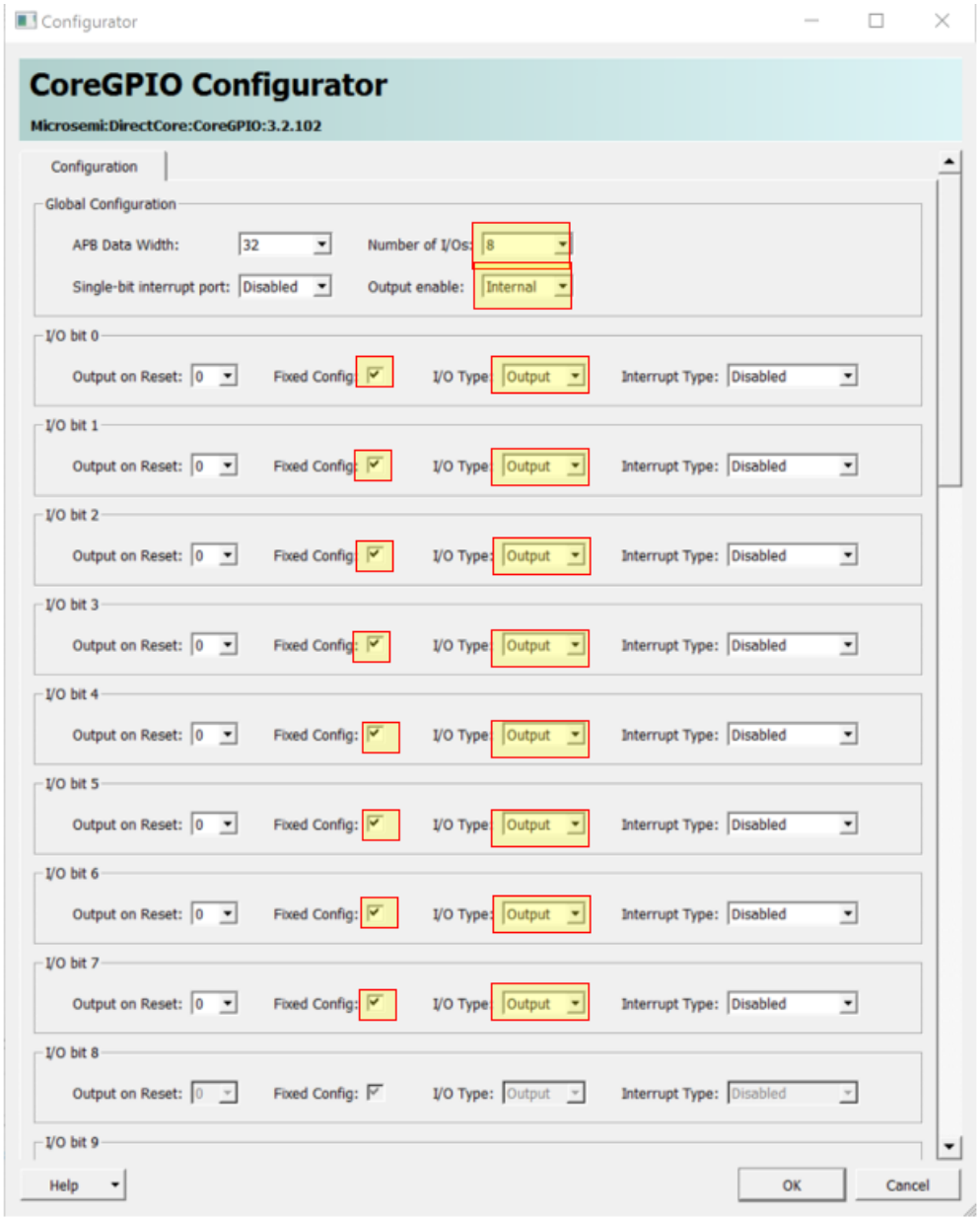


Figure 35 • CoreGPIO_OUT - Required Configuration

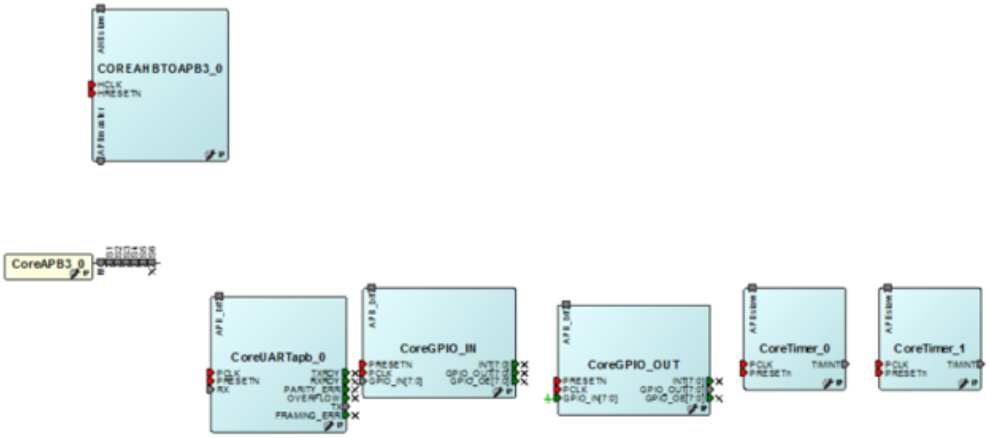


Note: For the IGLOO2 devices only, four GPIO_out connections are needed. Set the Number of I/Os field to 4 instead of 8.

3.7 Connecting Peripherals

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

Figure 36 • Peripheral Design No Ports Connected



They should be connected as shown in the following illustration.

Figure 37 • Peripheral Design Port Connections

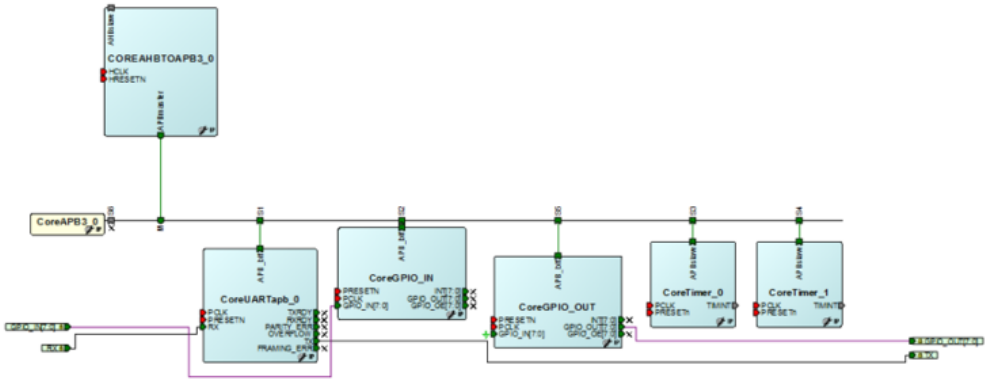


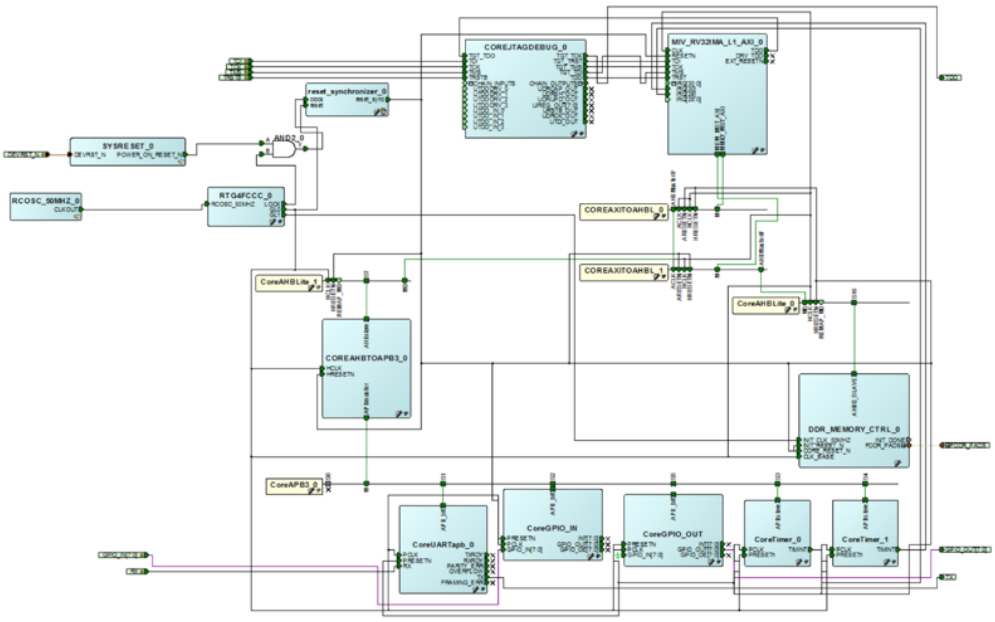
Table 3 • Table of Peripheral Design Connections

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

3.8 Connecting the Core Design to the Peripherals Design

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

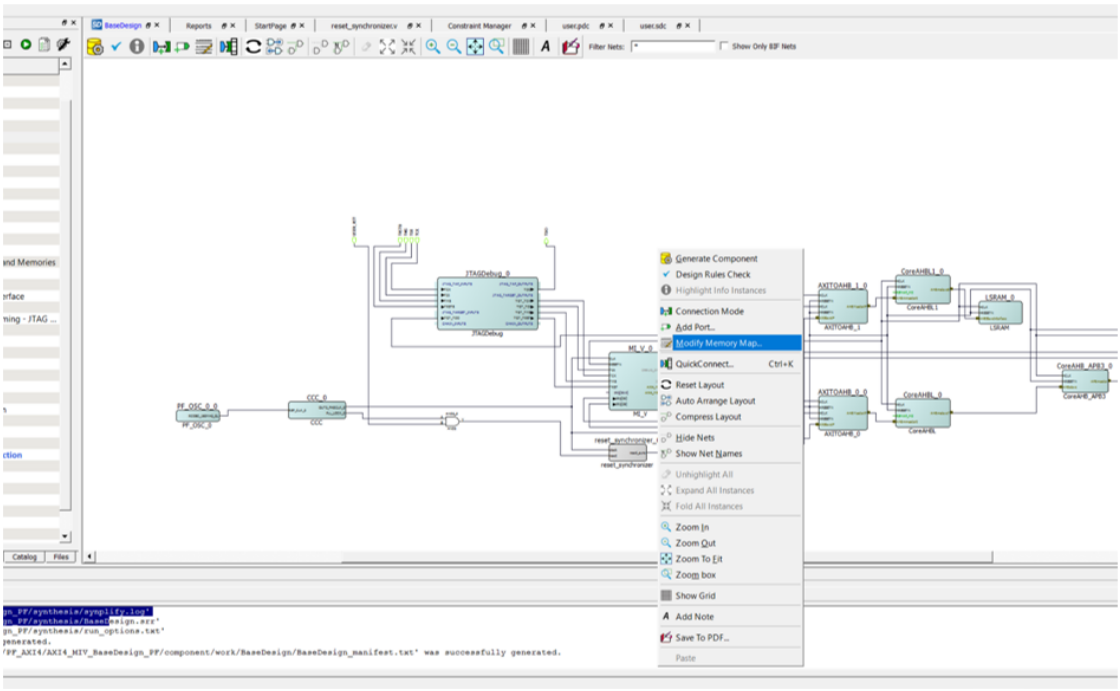
Figure 38 • Connected Core and Peripheral Designs



4 Generating a Memory Map

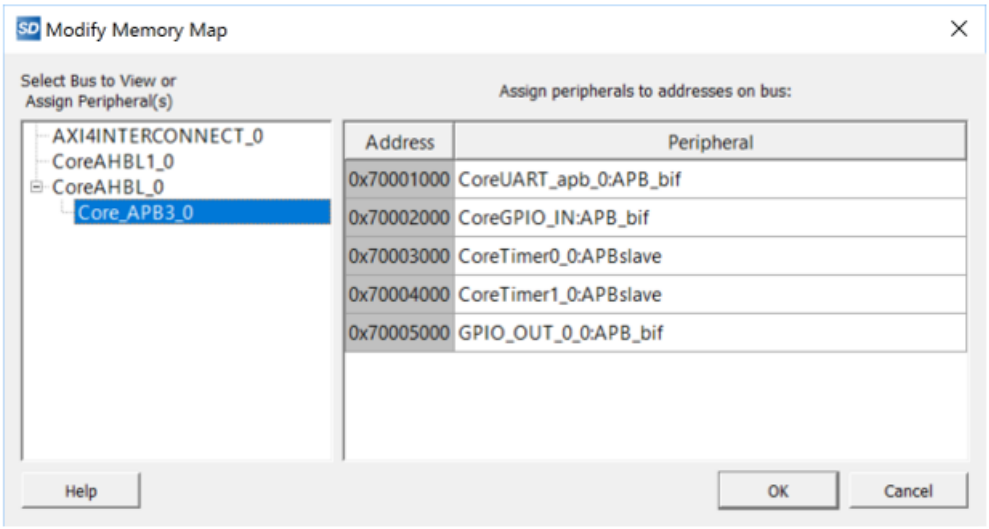
A memory map will show the slot numbers on different interfaces. These slot numbers correspond to the memory address on the bus. To see the memory map, right click on the design canvas and select Modify Memory Map.

Figure 39 • 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 40 • Contents of Memory Map



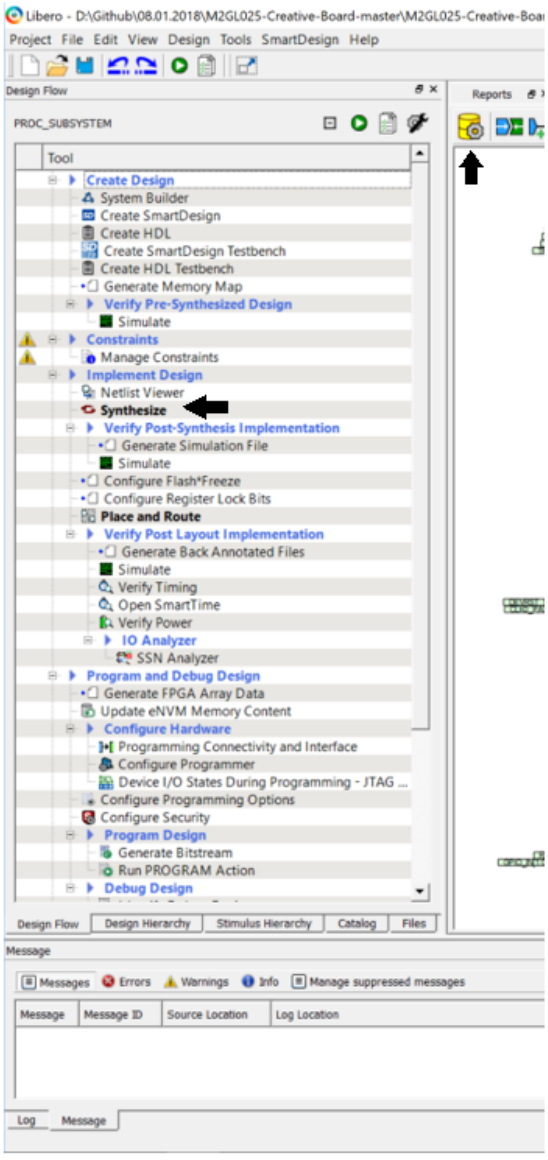
5 Generating a Bitstream and Downloading the FPGA Data to a Device

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.

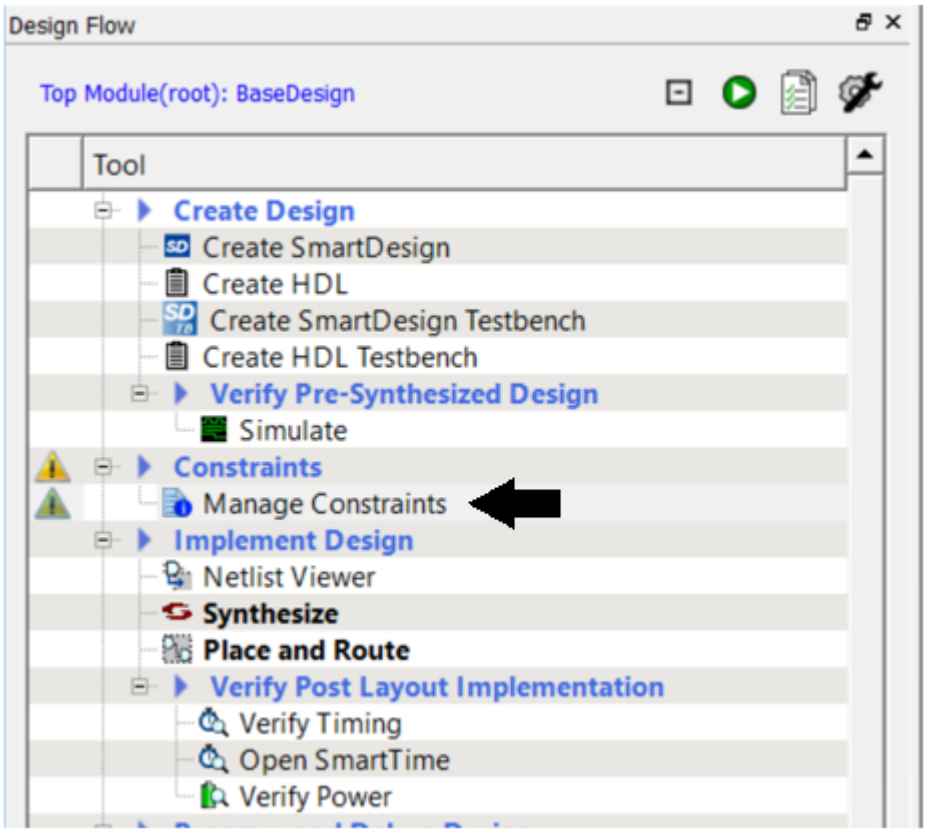
Figure 41 • Generate and Synthesize



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

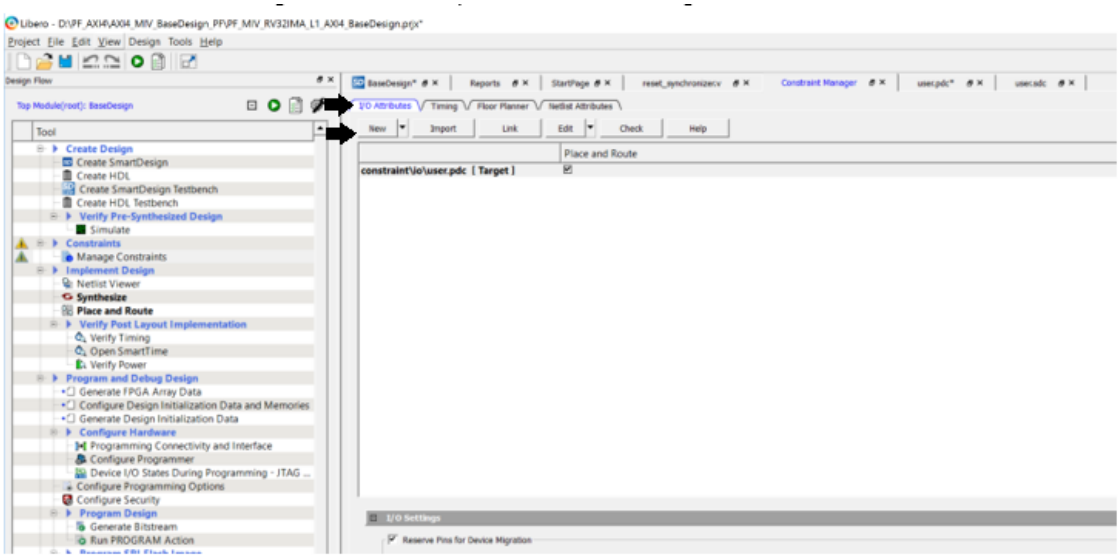
Double click Manage Constraints to open the constraint manager.

Figure 42 • Opening the Constraints Manager



Open the I/O Attributes tab. Select New, followed by Create New I/O Constraint.

Figure 43 • 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 SmartFusion2 M2S090-FG484 devices – constraints for some IGLOO2 and RTG4 projects are in Section 6 (see page 51) of this document.

```
set_io {GPIO_IN[0]} \
-pinname L20 \
-fixed yes \
-DIRECTION INPUT

set_io {GPIO_IN[1]} \
-pinname K16 \
-fixed yes \
-DIRECTION INPUT

set_io {GPIO_IN[2]} \
-pinname K18 \
-fixed yes \
-DIRECTION INPUT

set_io {GPIO_IN[3]} \
-pinname J18 \
-fixed yes \
-DIRECTION INPUT

set_io {GPIO_IN[4]} \
-pinname L19 \
-fixed yes \
-DIRECTION INPUT

set_io {GPIO_IN[5]} \
-pinname L18 \
-fixed yes \
-DIRECTION INPUT

set_io {GPIO_IN[6]} \
-pinname K21 \
-fixed yes \
-DIRECTION INPUT

set_io {GPIO_IN[7]} \
-pinname K20 \
-fixed yes \
-DIRECTION INPUT

set_io {GPIO_OUT[0]} \
-pinname E1 \
-fixed yes \
-DIRECTION OUTPUT

set_io {GPIO_OUT[1]} \
-pinname F4 \
-fixed yes \
-DIRECTION OUTPUT
```

```
set_io {GPIO_OUT[2]} \
-pinname F3 \
-fixed yes \
-DIRECTION OUTPUT

set_io {GPIO_OUT[3]} \
-pinname G7 \
-fixed yes \
-DIRECTION OUTPUT

set_io {GPIO_OUT[4]} \
-pinname H7 \
-fixed yes \
-DIRECTION OUTPUT

set_io {GPIO_OUT[5]} \
-pinname J6 \
-fixed yes \
-DIRECTION OUTPUT

set_io {GPIO_OUT[6]} \
-pinname H6 \
-fixed yes \
-DIRECTION OUTPUT

set_io {GPIO_OUT[7]} \
-pinname H5 \
-fixed yes \
-DIRECTION OUTPUT

set_io RX \
-pinname G18 \
-fixed yes \
-DIRECTION INPUT

set_io TX \
-pinname H19 \
-fixed yes \
-DIRECTION OUTPUT
```

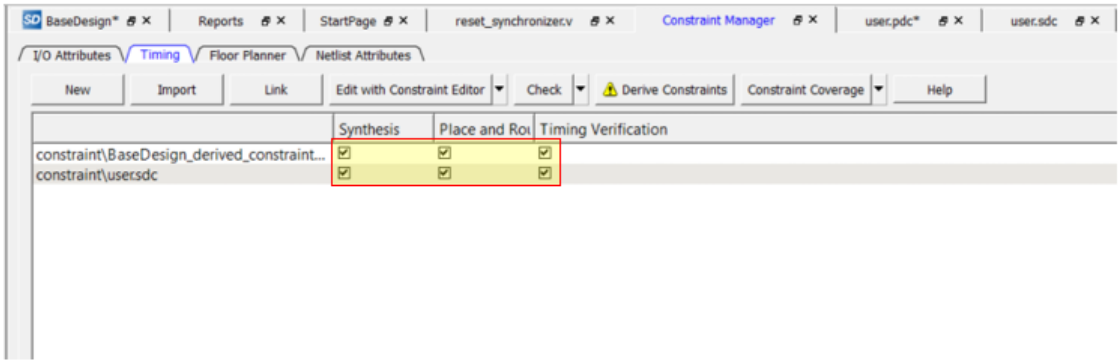
Copy this text into the editor that appears when a new constraint file is created. Save and close the file. Return to the constraints manager, and ensure that the I/O constraints are selected for "Place and Route". Next open the Timing tab in the constraints manager. Create a new timing constraints file,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

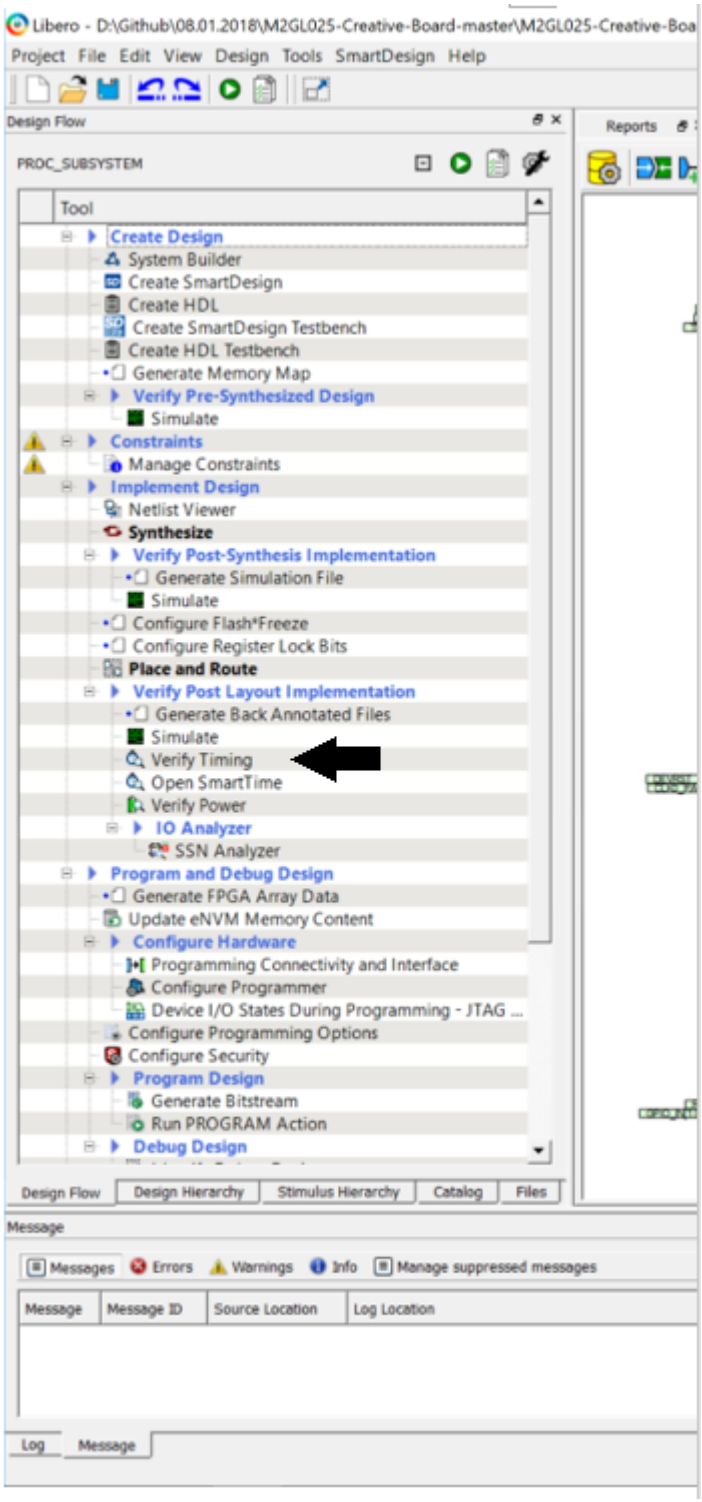
Copy this text into the editor and save the file. Click Derive Constraints and allow it to run. Ensure the timing constraints are selected for Synthesis, Place and Route, and Timing Verification.

Figure 44 • Selecting Constraints for Use with Tools

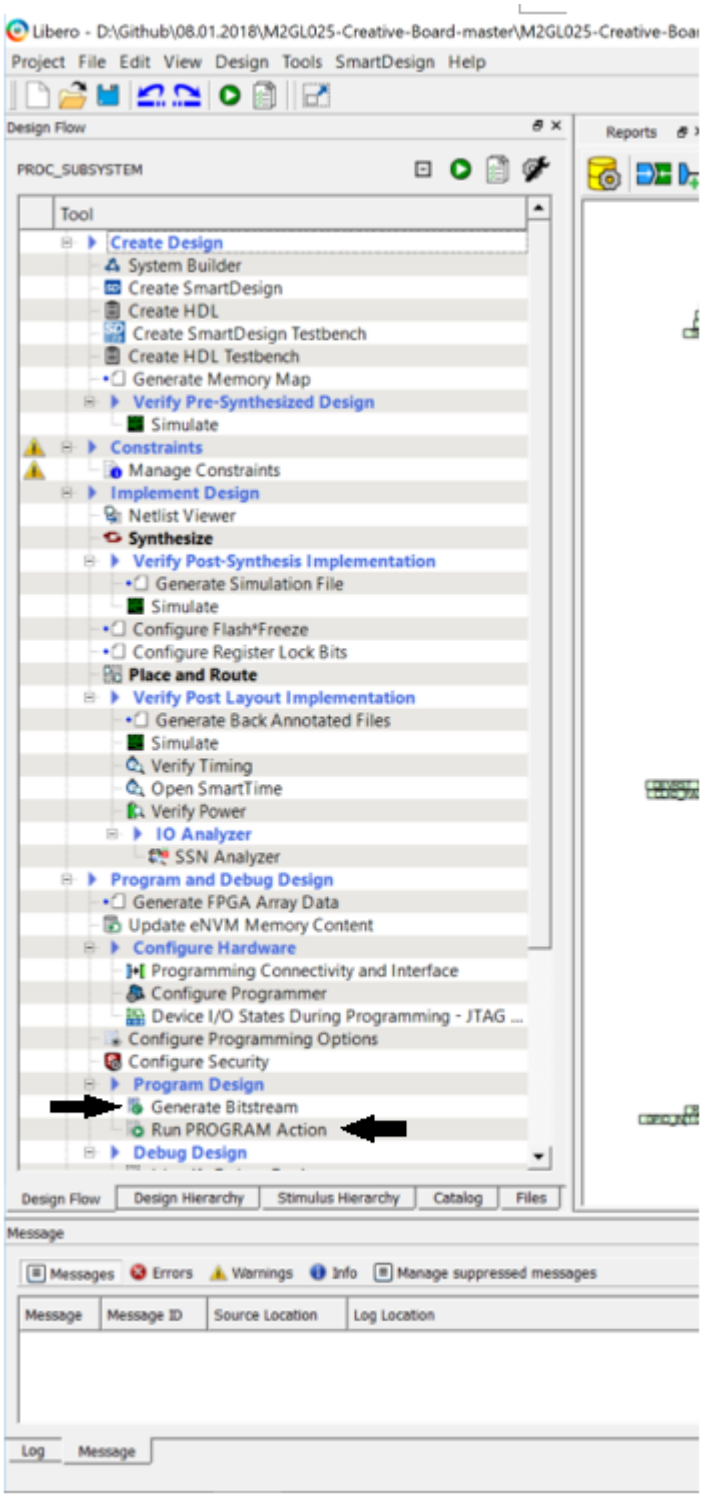


Once you have added the constraints, return to the design flow and click Verify Timing. If you try to use a design that does not meet timing, it may not work correctly or function at all. If timing verification fails, you can examine the violating paths using the SmartTime tool.

Figure 45 • Generating a Bitstream and Downloading to the FPGA



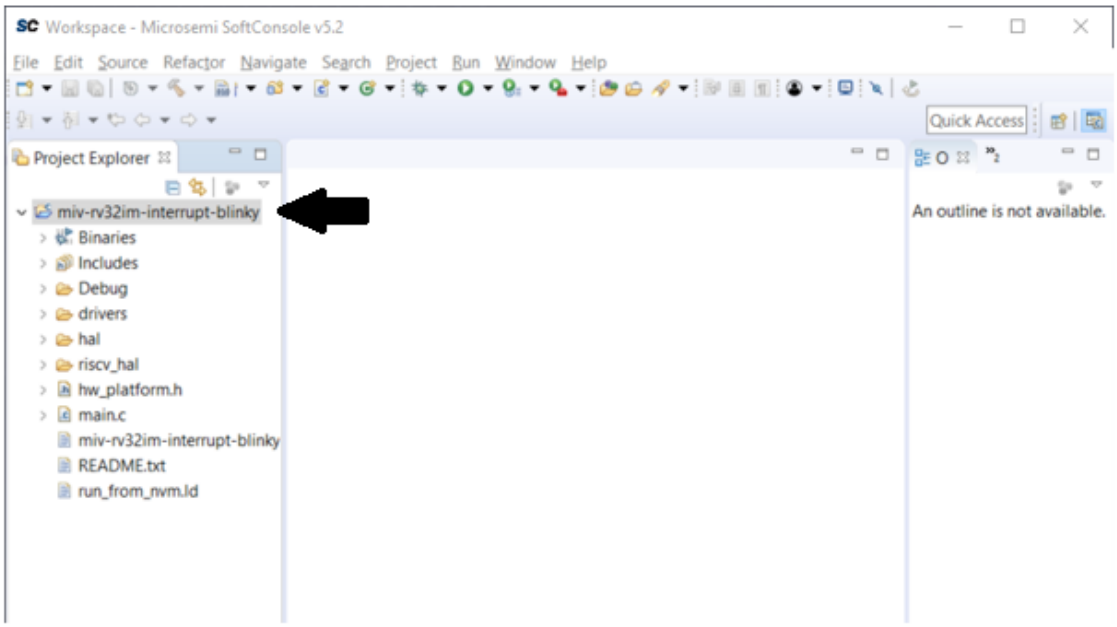
Once you have verified that your design meets timing requirements, click Generate Bitstream to generate the bitstream. Once the bitstream has been generated, connect your device and make sure it is powered on. Then, select Run PROGRAM Action. Once these actions have been completed, you can close Libero.



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. These can also be downloaded from the Microsemi Github page. To begin, open SoftConsole and open one of the miv projects. In this case, “miv-rv32im-interrupt-blinky” will be demonstrated.

Figure 46 • Opening a Project in SoftConsole

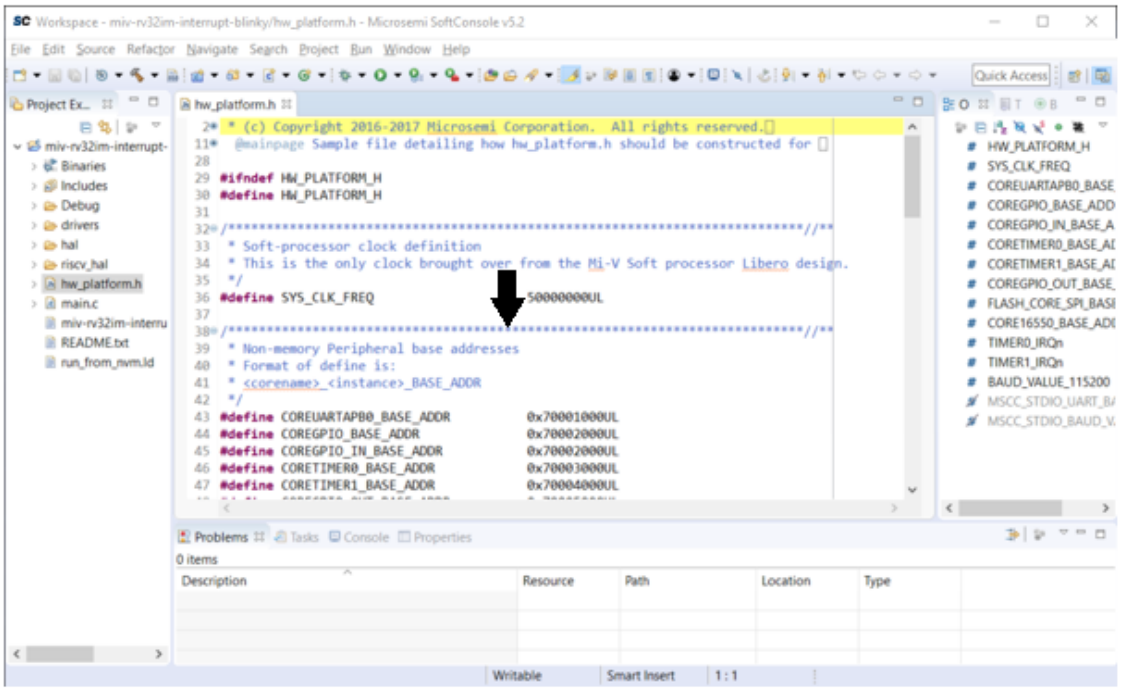


The driver files and HAL files are already included in the project for you. If newer versions of these drivers are available—or if a core version is updated—it is advised to download the latest versions through the firmware catalog, which is installed with Libero.

6.1 Setting the System Clock Frequency and Peripheral Base Addresses

In Section 2 (see page 7), the clock conditioning circuit (CCC) was configured. The clock frequency is the system clock frequency. This also needs to be set in the core. This is done in the "hw_platform.h" file by changing the #define SYS_CLK_FREQ to the clock frequency. This value must be in hertz.

Figure 47 • Setting Clock Frequency and Peripheral Base Addresses

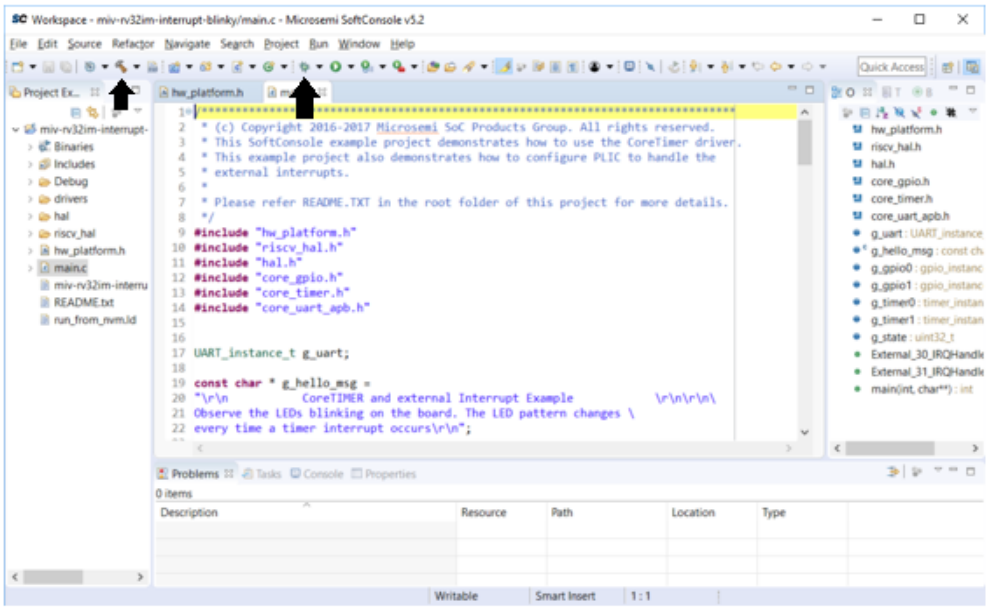


The “hw_platform.h” file is also used to set the base address for peripherals. Figure 35 (see page 23) illustrates the generated memory map. The values from the memory map should be in this file as the base addresses for the peripherals. The peripheral address must be correct for peripherals to function.

6.2 Building and Debugging a Project

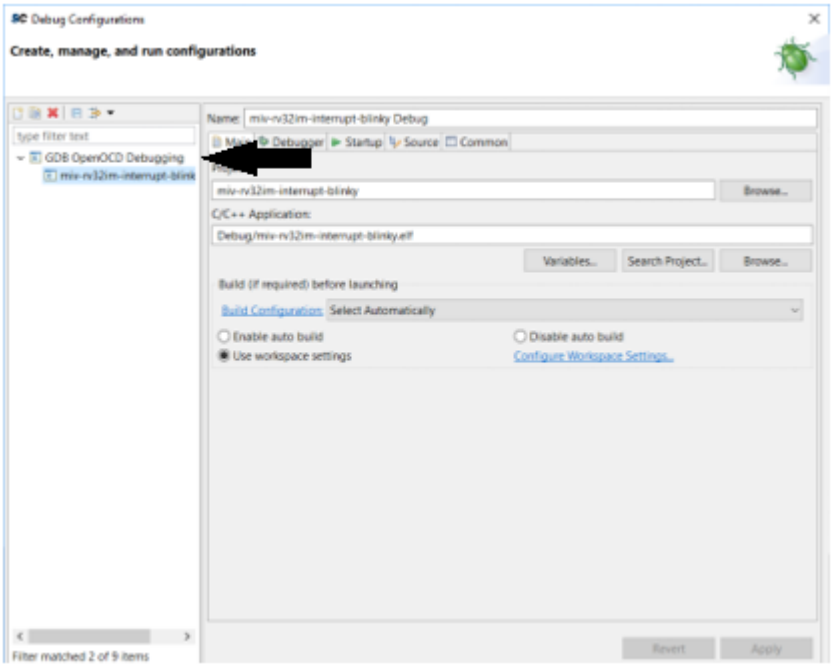
When you are ready to build your project, you can click on the Build icon. In the following illustration, the project is being built for debug.

Figure 48 • Building and Debugging a Project



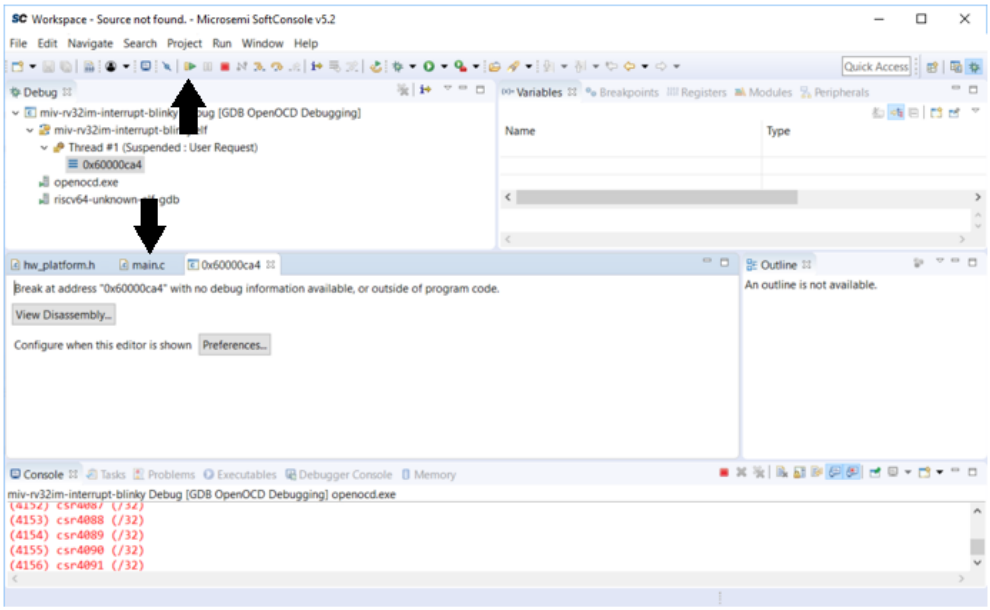
Once you have an error free build, you can test it on your device. Make sure your device is connected and powered on, and then click the Debug icon. If this is your first time debugging this project, you will have to select a debug configuration.

Figure 49 • Selecting 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 open but not run. It can be run by clicking the Resume button or pressing F8.

Figure 50 • Running a Program



To add breakpoints and step through the code, make sure that you are in the “main.c” tab to view your code.

6.3 Communicating 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 & LPT). It can be unclear which one is your device, so there may be trial and error to find the correct port.

Figure 51 • Opening Device Manager

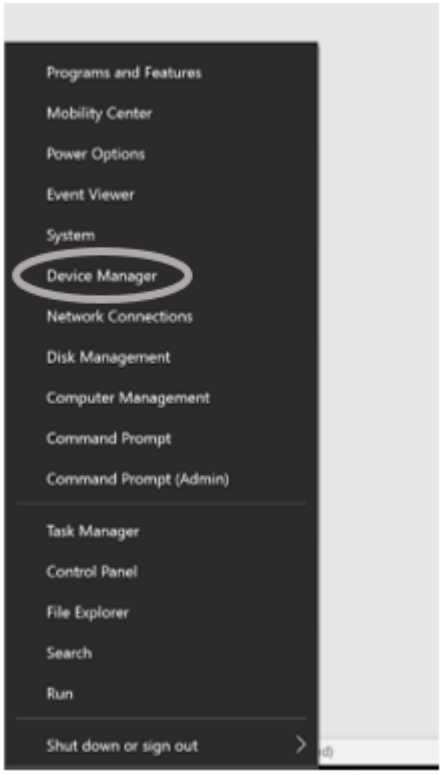
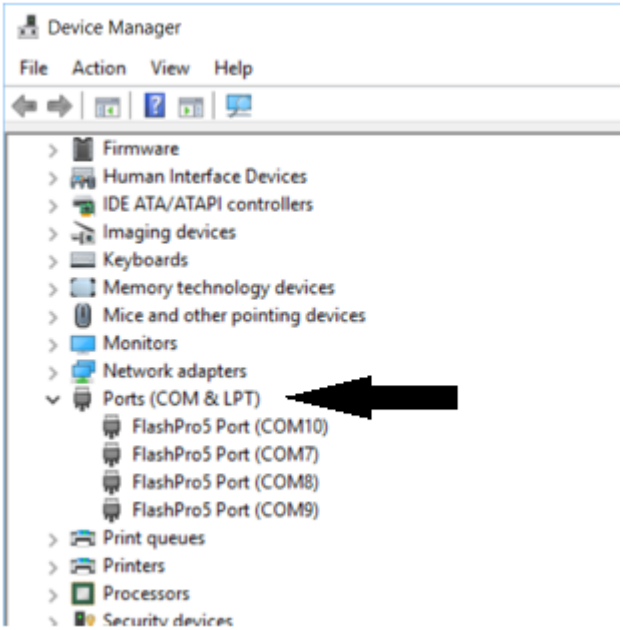
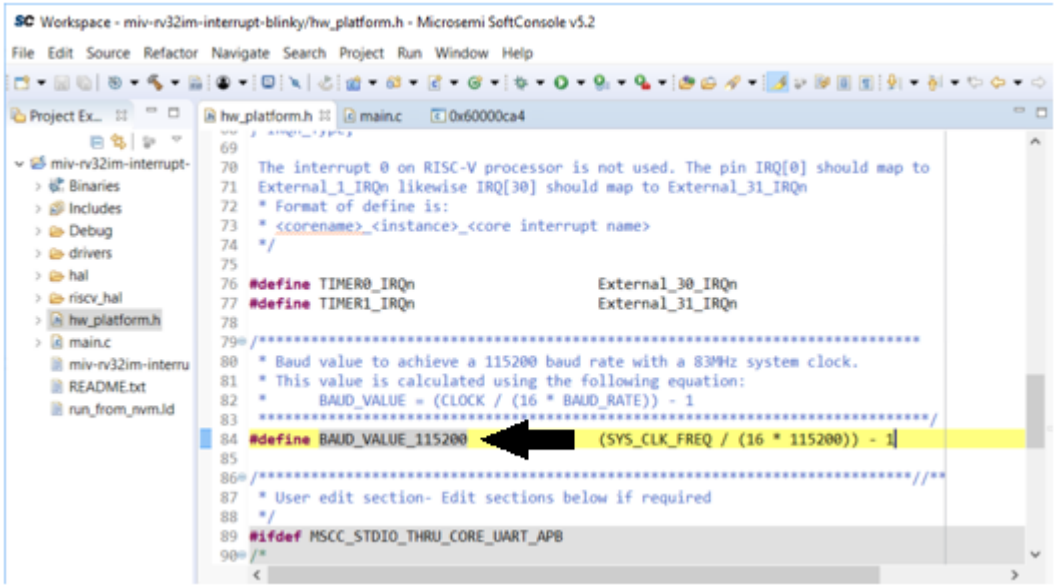


Figure 52 • Finding COM Ports in Device Manager



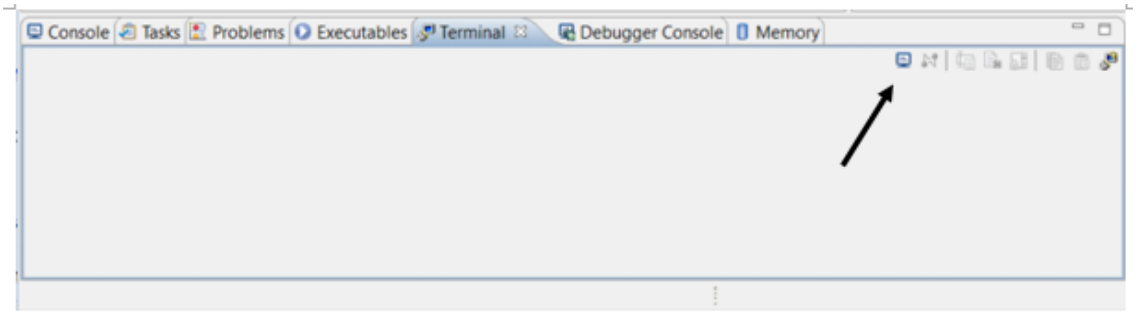
The baud rate for the UART connection can be found in the “hw_platform.h” file within the SoftConsole project.

Figure 53 • Location of BAUD Rate Value



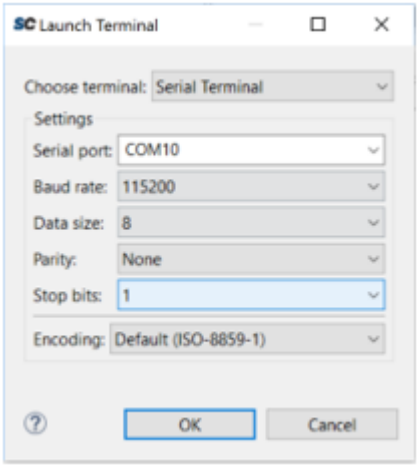
Open the terminal in SoftConsole by clicking Window -> Show View -> Terminal.

Figure 54 • Opening the Terminal Configurator



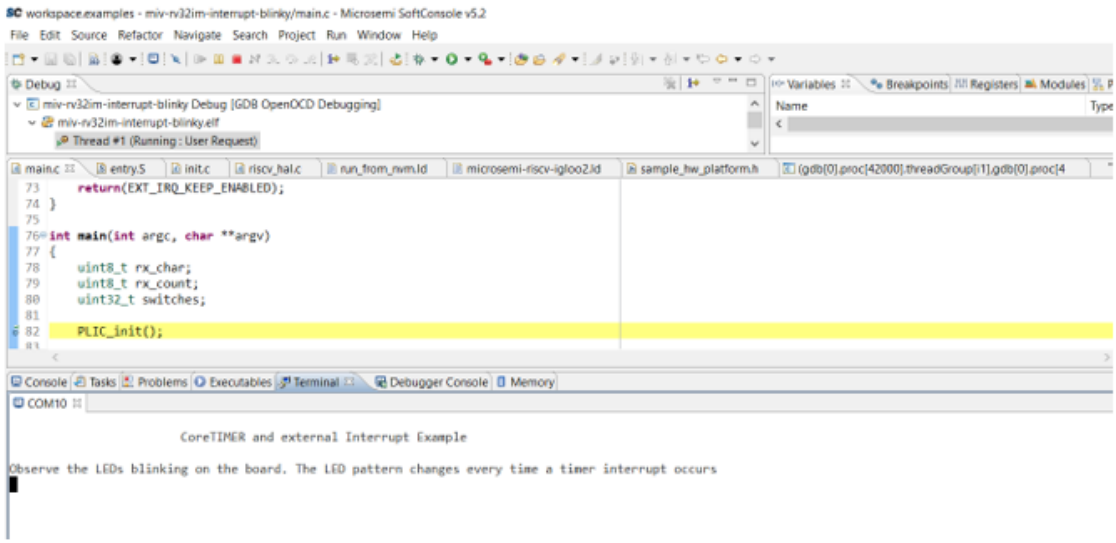
Once the terminal tab opens, click this button to configure a new connection. The terminal type will be serial. The port is the COM port from device manager. The baud rate can be found in the "hw_platform.h" file. The data size and parity are set during configuration.

Figure 55 • Configuring the Terminal



Once these settings have been entered, click OK to open the connection.

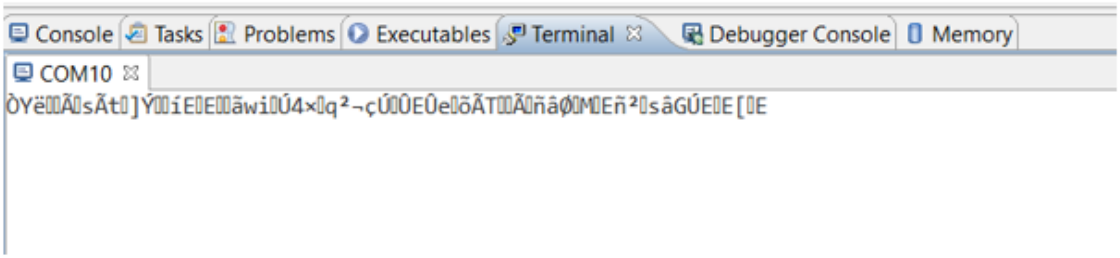
Figure 56 • Serial Output from COM Terminal



Run your program, and all UART outputs will be displayed in the terminal.

If the UART output is not as expected (for example, random/unknown characters as shown below), it is likely an issue with the clock frequency in the "hw_platform.h" file. This can be fixed by following the steps in Section 6.1 (see page 44).

Figure 57 • UART Clock Frequency Issues



7 Constraints for IGLOO2 Projects

Below are the constraints for the M2GL025TS-FCG484:

```
set_io {GPIO_IN[0]} \
-pinname H12 \
-fixed yes \
-DIRECTION INPUT

set_io {GPIO_IN[1]} \
-pinname H13 \
-fixed yes \
-DIRECTION INPUT

set_io {GPIO_OUT[0]} \
-pinname K16 \
-fixed yes \
-DIRECTION OUTPUT

set_io {GPIO_OUT[1]} \
-pinname M16 \
-fixed yes \
-DIRECTION OUTPUT

set_io {GPIO_OUT[2]} \
-pinname J16 \
-fixed yes \
-DIRECTION OUTPUT

set_io {GPIO_OUT[3]} \
-pinname N16 \
-fixed yes \
-DIRECTION OUTPUT

set_io RX \
-pinname H3 \
-fixed yes \
-DIRECTION INPUT

set_io TX \
-pinname G3 \
-fixed yes \
-DIRECTION OUTPUT
```

7.1 Constraints for RTG4 Projects

Below are the constraints for the RT4G150-1CG1657M:

```
set_io {GPIO_IN[0]} \
-pinname V33 \
-fixed yes \
-DIRECTION INPUT

set_io {GPIO_IN[1]} \
-pinname V34 \
-fixed yes \
-DIRECTION INPUT

set_io {GPIO_IN[2]} \
-pinname U34 \
-fixed yes \
-DIRECTION INPUT

set_io {GPIO_IN[3]} \
-pinname W36 \
-fixed yes \
-DIRECTION INPUT

set_io {GPIO_IN[4]} \
-pinname AA30 \
-fixed yes \
-DIRECTION INPUT

set_io {GPIO_IN[5]} \
-pinname AB31 \
-fixed yes \
-DIRECTION INPUT

set_io {GPIO_IN[6]} \
-pinname AB30 \
-fixed yes \
-DIRECTION INPUT

set_io {GPIO_IN[7]} \
-pinname AB32 \
-fixed yes \
-DIRECTION INPUT

set_io {GPIO_OUT[0]} \
-pinname W35 \
-fixed yes \
-DIRECTION OUTPUT

set_io {GPIO_OUT[1]} \
-pinname W34 \
-fixed yes \
-DIRECTION OUTPUT

set_io {GPIO_OUT[2]} \
-pinname V30 \
```

```
-fixed yes \
-DIRECTION OUTPUT

set_io {GPIO_OUT[3]} \
-pinname W33 \
-fixed yes \
-DIRECTION OUTPUT

set_io {GPIO_OUT[4]} \
-pinname T33 \
-fixed yes \
-DIRECTION OUTPUT

set_io {GPIO_OUT[5]} \
-pinname U35 \
-fixed yes \
-DIRECTION OUTPUT

set_io {GPIO_OUT[6]} \
-pinname R36 \
-fixed yes \
-DIRECTION OUTPUT

set_io {GPIO_OUT[7]} \
-pinname T34 \
-fixed yes \
-DIRECTION OUTPUT

set_io RX \
-pinname E27 \
-fixed yes \
-DIRECTION INPUT

set_io TX \
-pinname E28 \
-fixed yes \
-DIRECTION OUTPUT
```

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-01004-1.00-1018