TU0775 Tutorial PolarFire FPGA: Building a Mi-V Processor Subsystem





a 🔨 MICROCHIP company

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

©2019 Microsemi, a wholly owned subsidiary of Microchip Technology Inc. All rights reserved. Microsemi and the Microsemi logo are registered trademarks of Microsemi Corporation. All other trademarks and service marks are the property of their respective owners. Microsemi makes no warranty, representation, or guarantee regarding the information contained herein or the suitability of its products and services for any particular purpose, nor does Microsemi assume any liability whatsoever arising out of the application or use of any product or circuit. The products sold hereunder and any other products sold by Microsemi have been subject to limited testing and should not be used in conjunction with mission-critical equipment or applications. Any performance specifications are believed to be reliable but are not verified, and Buyer must conduct and complete all performance and other testing of the products, alone and together with, or installed in, any end-products. Buyer shall not rely on any data and performance specifications or parameters provided by Microsemi. It is the Buyer's responsibility to independently determine suitability of any products and to test and verify the same. The information provided by Microsemi hereunder is provided "as is, where is" and with all faults, and the entire risk associated with such information is entirely with the Buyer. Microsemi does not grant, explicitly or implicitly, to any party any patent rights, licenses, or any other IP rights, whether with regard to such information itself or anything described by such information. Information provided in this document is proprietary to Microsemi, and Microsemi reserves the right to make any changes to the information in this document or to any products and services at any time without notice.

About Microsemi

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. Learn more at www.microsemi.com.



Contents

1	Revisi 1.1 1.2 1.3 1.4 1.5	on History	1 1 1 1
2	Buildir	ng a Mi-V Processor Subsystem	2
_	21	Requirements	2
	2.2	Prereguisites	2
	2.3	Design Description	3
		2.3.1 Fabric RAMs Initialization	3
	2.4	Creating a Mi-V Processor Subsystem	4
		2.4.1 Creating a Libero Project	4
		2.4.2 Creating a New SmartDesign Component	5
		2.4.3 Instantiating IP Coles in SmartDesign	9
		2.4.5 Generating SmartDesign Component	4
		2.4.6 Managing Timing Constraints	4
		2.4.7 Running the Libero Design Flow	5
3	Buildir	ng the User Application Using SoftConsole	5
	3.1	Creating a Mi-V SoftConsole Project	5
	3.2	Downloading the Firmware Drivers	7
	3.3	Importing the Firmware Drivers	9
	3.4	Creating the main.c File	1
	3.5	Mapping Firmware Drivers and the Linker Script	2
	3.6	Mapping Memory and Peripheral Addresses	8
	3.7	Setting the UART Baud Rate	0
	3.8	Building the Mi-V Project	0
	3.9	Debugging the User Application Using SoftConsole	1
	3.10	Debugging the User Application from DDR3 Memory	7
4	Apper	ndix	3
	4.1	References	8



Figures

Figure 1	Download New Cores Option	3
Figure 2	Block Diagram	3
Figure 3	New Project Details	4
Figure 4	Device Selection	5
Figure 5	Create New SmartDesign	5
Figure 6	Mi-V Configuration	6
Figure 7	CoreAXI/Interconnect Configurator – Bus Configuration Section	6
Figure 8	CoreAXI/Interconnect - Master() and Master1 Configurations	7
Figure 9	CoreAXI/Interconnect Configurator – Slaven Configuration	7
Figure 10	CoreAXI/Interconnect Configurator – Slavet Configuration	2 Q
Figure 11	CoreAXI/Interconnect Configurator – Slave? Configuration	Q Q
Figure 12	Crossbar Configuration and Enabling Master Write Access Sattings	0
Figure 12	Enable Master Paed Assess Settings	9
Figure 13		9
	PF_SRAM_ARDE_AAI CONIIguialoi	
Figure 15		0
Figure 16	DDR3 General Configuration	
Figure 17	DDR3 Controller Configuration	2
Figure 18		3
Figure 19	CoreAPB3 Configuration	4
Figure 20	CoreGPIO Configuration	5
Figure 21	CoreSPI Configuration	6
Figure 22	CCC Configurator Clock Options PLL Tab1	7
Figure 23	CCC Configurator Output Clocks Tab 1	8
Figure 24	INIT_MONITOR Configuration 1	8
Figure 25	PROC_SUBSYSTEM with All Components Instantiated1	9
Figure 26	Connection Method	9
Figure 27	Mi-V Subsystem Connected	20
Figure 28	Edit Slices Window	23
Figure 29	AHBLite_0 Peripherals Address Map	23
Figure 30	APB3_0 Peripherals Address Map	24
Figure 31	Generate Component Icon	24
Figure 32	Derive Constraints Button	24
Figure 33	Derived Constraints	25
Figure 34	I/O Attributes	26
Figure 35	Edit with I/O Editor Option	26
Figure 36	Port View	26
Figure 37	I/O Editor Design View – DDR3 Selection	26
Figure 38	Memory View [active] Tab with DDR3 Subsystem Placement	7
Figure 39	DDR3 0 Placed	 7
Figure 40	Fabric RAMs Tab	20
Figure 41	Edit Eabric RAM Initialization Client Dialog Box	20.
Figure 42	Import Memory File Dialog Box	20
Figure 13	Eabric PAMs Tab - Apply Button	20
Figure 40	Posicial Anticipation Data	.9 20
Figure 44	SDI Flack Tak	21
Figure 45	COM Dort Number 2	וי רי
Figure 40	Connection Type Selection)Z
Figure 47	Dufficulturing the Selection)∠ 22
Figure 40	Full F Conniguration 3 Halle World String 3). ∧ (
Figure 49	Hello World String After the Board is Dever Cycled)4 ∧ 4
Figure 50	Merkensee Leureber)4)/
Figure 51	Workspace Launcher	5
Figure 52		15
Figure 53		56
⊢ıgure 54	Select Configurations Dialog Box	6



Figure 55	Empty Mi-V Project	37
Figure 56	Einpty Mi-V Troject	38
Figure 57		38
Figure 58	Corel IARTanh Files Report	38
Figure 50	CoreGPIO Files Report	30 30
Figure 60	CoreSPI Driver Files Report	30
Figure 61		30
Figure 62	Import Dialog Box	10
Figure 63	Import Dialog Box	40 40
Figure 64	main c File Creation	40 // 1
Figure 65		71 //1
Figure 66		+ i イン
Figure 67		+2 12
Figure 68	Target Processor Tool Settings	+2 /2
Figure 60	CNILL PISC V Cross C Compiler Tool Settings	40
Figure 70	Add Directory Bath Dialog Box	44 11
Figure 70		44 15
Figure 72	Tool Softings Tob with CoroCPIO Both Added	45 45
Figure 72	Tool Settings Tab After Successful Mapping	40
Figure 73	Selecting the Linker Second Mapping	40
Figure 74		40
Figure 75		+1 10
Figure 70	Linker Serint	40
Figure 77	Linker Script	49
Figure 70	System Clock Frequency Definition	49 50
Figure 79		50
Figure 80		50
Figure 01	Create manage and run configurations Window. Main Tab	51
Figure 62	Mil/ uprt blipky of Soloction	31 52
Figure 03	Miv_uait_bilitiky.ell Selection	52 52
Figure 04	Create, manage, and full configurations window – Debugger Tab	55
Figure 05	Confirm Derengetive Switch Dieleg Pey	54
Figure 60		54
Figure 87		55
Figure 88	Resume Application Execution	55
Figure 69		55
Figure 90		50
Figure 91	Vallable Values	30 57
Figure 92	RAINI SIAILAUULESS PALAIHELEIS	57 57
Figure 93	ראטעשעט אווטוו פאדע איז	57



Tables

Table 1	Tutorial Requirements	. 2
Table 2	CCC 0 0 Pin Connections	20
Table 3	DEBUG_TARGET Pin Connections	21
Table 4	AXI4_Interconnect_0 Pin Connections	22
Table 5	APB3_0 Pin Connections	22
Table 6	Jumper Settings	31



1 Revision History

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

1.1 Revision 5.0

The following is a summary of the changes made in this revision.

- Updated for Libero SoC v12.2.
- Removed Libero SoC and SoftConsole version numbers.

1.2 Revision 4.0

The following is a summary of the changes made in this revision.

- Added Fabric RAMs Initialization, page 3.
- The document was updated for Libero SoC v12.0.

1.3 Revision 3.0

The following is a summary of the changes made in this revision.

- Added Design Description, page 3.
- The document was updated for Libero SoC PolarFire v2.1.

1.4 Revision 2.0

The following is a summary of the changes made in this revision.

- The document was updated for the Mi-V processor upgrade.
- The document was updated for Libero SoC PolarFire v2.0 and SoftConsole v5.2. For more information, see Building the User Application Using SoftConsole, page 35.
- Information about LSRAM initialization from external SPI flash was added. For more information, see Configure Design Initialization Data and Memories, page 27.

1.5 Revision 1.0

The first publication of this document.



2 Building a Mi-V Processor Subsystem

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

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

2.1 Requirements

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

Requirement	Version
Hardware	
Host PC	Windows 7, 8.1, or 10
POLARFIRE-EVAL-KIT (MPF300TS-FCG1152I) – PolarFire Evaluation Board – 12 V/5 A AC power adapter and cord – USB 2.0 A to mini-B cable	Rev D or later
Software	
Libero SoC Design Suite	See the readme.txt file provided in the design files for all software versions needed to create this reference design.
Firmware Catalog ¹	
SoftConsole	See the readme.txt file provided in the design files for all software versions needed to create this reference design.
PuTTY (serial terminal emulation program)	

Table 1 •Tutorial Requirements

1. Firmware catalog is included in the installation package of Libero SoC.

2.2 **Prerequisites**

 Download the design files from: http://soc.microsemi.com/download/rsc/?f=mpf_tu0775_df

The design files folder contains the following folders:

- **Programming_Job**: Contains the programming file (.job) for reference
- Solution: Contains the final Libero and SoftConsole projects for reference
- Source: Contains the source files required to complete this tutorial
- Download and install Libero SoC from: https://www.microsemi.com/product-directory/design-resources/1750-libero-soc#downloads
- Download and install SoftConsole from: https://www.microsemi.com/products/fpga-soc/design-resources/design-software/softconsole#downloads



4. From the Libero Catalog, download the latest versions of the IP cores from the warning pop-up as shown in the following figure.

Figure 1 • Download New Cores Option



2.3 Design Description

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

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

Figure 2 • Block Diagram



2.3.1 Fabric RAMs Initialization

Each logical RAM instance in the design can be initialized from a different source– sNVM, µPROM, or SPI-Flash. The initialization client storage location is configurable. Generate the initialization data to add the initialization clients to the chosen non-volatile memories and program the device. Program SPI-Flash, if chosen as storage location for initialization data. For more information, see Configure Design Initialization Data and Memories, page 27.

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



2.4 Creating a Mi-V Processor Subsystem

Creating a Mi-V processor subsystem involves:

- Creating a Libero Project, page 4
- Creating a New SmartDesign Component, page 5
- Instantiating IP Cores in SmartDesign, page 5
- Connecting IP Instances in SmartDesign, page 19
- Generating SmartDesign Component, page 24
- Managing Timing Constraints, page 24
- Running the Libero Design Flow, page 25

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

2.4.1 Creating a Libero Project

Follow these steps to create a Libero project:

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

Figure 3 • New Project Details

🕑 New project							
Project details Specify project details							
Project Details	Project name:	PF_Mi_V_Tut					
Device Selection	Project location:	F:/Libero_Projects					Browse
Device Settings	Description:						
Design Template	Preferred HDL type	e: Verilog -					
Add HDL Sources	E Enable block of	eauon					
Add Constraints							
Help				< <u>B</u> ack	<u>N</u> ext >	Einish	Cancel



- 3. To choose the PolarFire device present on the PolarFire Evaluation Board, select the following settings in the **Device Selection** window, and click **Next**.
 - Family: PolarFire
 - Die: MPF300TS
 - Package: FCG1152
 - Speed: -1
 - Range: IND
 - Part Number: MPF300TS-1FCG1152I

Figure 4 • Device Selection

Family: PolarFire		- Die: N	MPF300TS	-	Package:	FCG1152	•		
Speed: -1		•			Range:	IND	•		
						Deee	- Chara - 1		
						Rese	t niters		
						Rese	t filters		
arch part:						Kese			
arch part:	DFF	User I/Os	uSRAM	LSRA	M	Math	H-Chip Globals	PLL	DLL

- 4. In the Device Settings window, click Next to retain the default core voltage and I/O settings.
- 5. In the Add HDL Sources window, click Next to retain the default settings.
- 6. In the Add constraints window, click Import file to import the I/O constraint file.
- 7. In the Import files window, locate the io_constraints.pdc file in the DesignFiles_directory\Source\io folder, and double-click it.
- 8. Click Finish.

The Log pane displays a message indicating that the PF_Mi_V_Tut project was created.

2.4.2 Creating a New SmartDesign Component

To create a new SmartDesign component:

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

Figure 5 • Create New SmartDesign

Create New SmartD	Design 8 X
Name:	
PROC_SUBSYSTEM	
Help	OK Cancel

3. Click OK.

The PROC_SUBSYSTEM SmartDesign component is created.

2.4.3 Instantiating IP Cores in SmartDesign

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



2.4.3.1 Instantiating Mi-V Processor IP

- 1. From the Catalog, drag the Mi-V RV32IMA_L1_AXI to SmartDesign. In this tutorial AXI version of the Mi-V core is used for optimum performance.
- 2. In the Create Component dialog box, enter MiV_AXI as the component name, and click OK.
- 3. In the Configurator, set the following configuration:
 - Set Reset Vector Address -> Upper 16 bits (Hex) to 0x8000 and retain the default setting for Lower 16 Bits (Hex). This is the address the processor will start executing from after a reset. The processor's main memory must be accessible to Mi-V AXI memory interface whose memory-mapped address ranges from 0x80000000 to 0x8FFFFFFF. The Mi-V memory interface supports cached transactions, whereas Mi-V MMIO interface does not support them.
 Set the MASTER TYPE to AXI
 - Set the **MASTER_TYPE** to **AXI4**.
- 4. Retain the default settings for **MEM_WID**, and **MMIO_WID** options.

Figure 6 • Mi-V Configuration

Configuration	
Reset Vector Address	
Upper 16 bits (Hex) 0x8000 Lower 16 bits (Hex) 0x0	
AXI Interface	7
MASTER_TYPE AX14 V MEM_WID 6 MMIO_WID 9	
	_

2.4.3.2 Instantiating AXI Interconnect Bus IP

The AXI interconnect bus must be configured to connect the Mi-V core with memory and peripherals.

- 1. From the Catalog, drag the CoreAXI4Interconnect IP core to SmartDesign.
- 2. In the **Create Component** dialog box, enter **AXI4_Interconnect** as the component name, and click **OK**.

The Configurator opens.

3. In the **Bus Configuration** section, configure the AXI4_Interconnect IP to have three slaves with an ID width of 4, as shown in the following figure. Leave the rest as defaults.

Figure 7 • CoreAXI4Interconnect Configurator – Bus Configuration Section

0	Configuration Master Configu	ration	Slave Configuration		Crossbar Configuration		
Ξ	Bus Configuration						0
_	Number of Masters: 2	•	Number of Slaves:	3	•		
	ID Width: 4	· • 6	Address Width:	32			
	User Width: 1	_					
Ξ	OPTIMIZATION Configuration						
	Optimization: C Performance	C Area	• User				
Ξ	OPTIMIZATION Configuration						0
	Number of Threads:	1	•	[Max Outstanding Transactions:	2	
	Slave FIFO Address Depth:	4		1	Slave FIFO Data Depth:	4	
	DWC Address FIFO Depth Ceiling	10			Read Arbitration Enable:	v	
	Crossbar Mode:	C SASD	SAMD	0			

- 4. In the Master Configuration section, retain the following Master0 and Master1 default settings:
 - M0 Type: AXI4
 - M0 Data Width: 64 bits
 - M0 DWC Data FIFO Depth: 16
 - M0 Register Slice: Selected



- M1 Type: AXI4
- M1 Data Width: 64 bits
- M1 DWC Data FIFO Depth: 16
- M1 Register Slice: Selected

The Master0 port must be connected to the Mi-V AXI MEM interface and Master1 port must be connected to the Mi-V AXI MMIO interface. The following figure shows the Master0 and Master1 configurations.

Figure 8 • CoreAXI4Interconnect - Master0 and Master1 Configurations

Configuration	Master Configuration	Slave Configuration	Crossbar Configuration
Master0 Cor	nfiguration		
M0 Type:	AXI4 💌	M0 Data Width:	64 💌
M0 DWC Data	FIFO Depth: 16	M0 Register Slice:	$\overline{\mathbf{v}}$
M0 Clock Dom	ain Crossing: 🗔	M0 Read Interleaving:	
Master1 Cor	figuration		
M1 Type:	AXI4 💌	M1 Data Width:	64 💌
M1 DWC Data	FIFO Depth: 16	M1 Register Slice:	v
M1 Clock Dom	ain Crossing:	M1 Read Interleaving:	

- 5. In the Slave Configuration section, configure the Slave0 port as follows:
 - S0 SLAVE Start Address (Lower 32 bits): 0x8000000
 - S0 SLAVE End Address (Lower 32 bits): 0x8000FFFF
 - S0 Clock Domain Crossing: Disabled
 - Leave the rest as defaults

The S0 slot is used to connect 64KB of internal LSRAMs through AXI4 interface. The address range for the S0 slot is set to 0x80000000 - 0x8000FFFF which is accessible by Mi-V processor through MEM interface. The Mi-V MEM interface address ranges from 0x80000000-0x8FFFFFFF.

Figure 9 • CoreAXI4Interconnect Configurator – Slave0 Configuration

Ξ	Slave0 Configuration			
	S0 Type:	AXI4	S0 Data Width:	64 💌
	S0 DWC Data FIFO Depth:	16 💌	S0 Register Slice:	V
	S0 SLAVE Start Address (Upper 32 Bits):	0x0	S0 SLAVE Start Address (Lower 32 Bits):	0x8000000
	S0 SLAVE End Address (Upper 32 Bits):	0x0	S0 SLAVE End Address (Lower 32 Bits):	0x8000ffff
	S0 Clock Domain Crossing:		S0 Read Interleaving:	

- 6. In the Slave Configuration section, configure Slave1 port as follows:
 - S1 SLAVE Start Address (Lower 32 bits): 0x80010000
 - S1 SLAVE End Address (Lower 32 bits): 0x80FFFFFF
 - S1 Clock Domain Crossing: Enabled
 - · Leave the rest as defaults

The S1 slot is used to connect an external DDR3 memory through a DDR3 controller IP. The address range for the S1 slot is set to 0x80010000 - 0x80FFFFFF which is accessible by Mi-V processor through MEM interface. The Mi-V processor MEM interface operates at 111.111 MHz where as the DDR3 controller AXI4 interface operates at 166.666 MHz. Hence the **S1 Clock Domain Crossing** option is enabled to handle the data transfers between two clock domains.



Figure 10 • CoreAXI4Interconnect Configurator – Slave1 Configuration

Slave1 Configuration			
S1Type:	AXI4	S1 Data Width:	64
S1 DWC Data FIFO Depth:	16 💌	S1 Register Slice:	$\overline{\mathbf{v}}$
S1 SLAVE Start Address (Upper 32 Bits):	0x0	S1 SLAVE Start Address (Lower 32 Bits):	0x80010000
S1 SLAVE End Address (Upper 32 Bits):	0x0	S1 SLAVE End Address (Lower 32 Bits):	0x80ffffff
S1 Clock Domain Crossing:		S1 Read Interleaving:	

- 7. In the **Slave Configuration** section, configure Slave2 port as follows:
 - S2 Type: AXI3
 - S2 SLAVE Start Address (Lower 32 bits): 0x60000000
 - S2 SLAVE End Address (Lower 32 bits): 0x60FFFFFF
 - S2 Clock Domain Crossing: Disabled
 - Leave the rest as defaults

The S2 slot is used to connect the peripherals to the Mi-V processor. The address range for the S2 slot is set to 0x60000000 - 0x60FFFFF, which is accessible by Mi-V processor through the MMIO interface. The Mi-V MMIO interface address ranges from 0x6000000-0x6FFFFFF.

Figure 11 • CoreAXI4Interconnect Configurator – Slave2 Configuration

Slave2 Configuration

-	Slavez comgaration					
	S2 Type:	AXI3	•	S2 Data Width:	64	•
	S2 DWC Data FIFO Depth:	16	•	S2 Register Slice:	v	
	S2 SLAVE Start Address (Upper 32 Bits):	0x0		S2 SLAVE Start Address (Lower 32 Bits):	0x6000000	
	S2 SLAVE End Address (Upper 32 Bits):	0x0	[S2 SLAVE End Address (Lower 32 Bits):	0x60ffffff	
	S2 Clock Domain Crossing:			S2 Read Interleaving:	Γ	

- 8. In the Crossbar Configuration section, ensure that the following options are set:
- Under Enable Master Write Access, enable M0 access S0, M0 access S1, and M0 access S2.
- Under Enable Master Write Access, enable M1 access S0, M1 access S1, and M1 access S2.
- Under Enable Master Read Access, enable M0 access S0, M0 access S1, and M0 access S2.
- Under Enable Master Read Access, enable M1 access S0, M1 access S1, and M1 access S2.
- Leave the rest as defaults.



Figure 12 • Crossbar Configuration and Enabling Master Write Access Settings

Configuration	Mas	ter Configuration		Slave Configuration		Crossbar Configurat	tion		
Data Width Configu	ration -								
Crossbar Data	Width:	64 💌							
Enable Maste	r Write	Access							
M0 access S0:	V	M0 access S1:	▼	M0 access S2:	◄	M0 access S3:	ম]	
M0 access S4:	$\overline{\mathbf{v}}$	M0 access S5:	V	M0 access S6:	M	M0 access S7: J	V		
M0 access S8:	2	M0 access S9:	V	M0 access S10:	M	M0 access S11:	V		
M0 access S12	: 🔽	M0 access S13:	V	M0 access S14:	M	M0 access S15: J	2		
M0 access S16	: 🔽	M0 access S17:	₹	M0 access S18:	M	M0 access S19: J	V		
M0 access S20	: 🖂	M0 access S21:	V	M0 access S22:	M	M0 access S23: J	V		
M0 access S24	: IZ	M0 access S25:	₽	M0 access S26:	M	M0 access S27: J	V		
M0 access S28	: IZ	M0 access S29:	M	M0 access S30:	M	M0 access S31: J	2		
M1 access S0:	•	M1 access S1:	◄	M1 access S2:	◄	M1 access S3:	N		

Figure 13 • Enable Master Read Access Settings

Ξ	Enable Master	Read Ao	cess					
	M0 access S0:	v	M0 access S1:	~	M0 access S2:		M0 access S3:	☑
	M0 access S4:	ম	M0 access S5:	N	M0 access S6:	M	M0 access S7:	☑
	M0 access S8:	$\overline{\lor}$	M0 access S9:	2	M0 access S10:	2	M0 access S11:	₹
	M0 access S12:	$\overline{\lor}$	M0 access S13:	2	M0 access S14:	2	M0 access S15:	₹
	M0 access S16:	$\overline{\mathbf{v}}$	M0 access S17:	2	M0 access S18:	2	M0 access S19:	₹
	M0 access S20:	V	M0 access S21:	ন	M0 access S22:		M0 access S23:	☑
	M0 access S24:	~	M0 access S25:	ন	M0 access S26:		M0 access S27:	⊵
	M0 access S28:	~	M0 access S29:	ন	M0 access S30:	V	M0 access S31:	☑
	M1 access S0:	v	M1 access S1:	v	M1 access S2:	V	M1 access S3:	⊽

2.4.3.3 Instantiating On-chip SRAM

- 1. From the Catalog, drag the PolarFire SRAM (AHBLite and AXI) IP core to SmartDesign.
- 2. In the **Create Component** dialog box, enter **LSRAM** as the component name, and click **OK**.
- 3. In the **Port Settings** tab of the PF_SRAM_AHBL_AXI Configurator, select the following settings:
 - Memory Depth (in words): 8192 (this creates 64 KB (8192 × 8 bytes) of memory)
 - Fabric Interface type: AXI
 - Data Width: 64
 - Width of ID: 5
 - Wrap Bust support: Enabled
 - · Leave the rest as defaults



Figure 14 • PF_SRAM_AHBL_AXI Configurator

Port settings Memory Initia	lization Settings
Memory Settings	
SRAM type	LSRAM
Memory Depth(in words)	8192
Use Native Interface	
Read port	Non-Pipelined(Address pipeline and No Data pipeline) 💌
Interface Settings Fabric Interface type A Data Width 64	XI I
AXI4 interface options — Address Width Width of ID Write Interface Read Interface	32

4. Click OK.

2.4.3.4 Instantiating DDR3 Memory Controller

This tutorial demonstrates how to build and debug an application from DDR3 memory. Executing an application from DDR3 memory in the release mode requires a bootloader. The bootloader use case is not in the scope of this tutorial.

- 1. From the Catalog, drag the **PolarFire DDR3** IP core to SmartDesign.
- 2. In the Create Component dialog box, enter DDR3_0 as the component name, and click OK.
- 3. In the left pane of the Configurator, expand Microsemi PolarFire Evaluation Kits > PolarFire Evaluation Kit > MPF300T.
- Left-click MT41K1G8SN-125, and click Apply, as shown in the following figure. This configures the DDR3 controller with the initialization and timing parameters of the DDR3 memory (MT41K1G8SN-125) present on the PolarFire Evaluation Kit.

Figure 15 • Apply Option for MPF300T

PF_DDR3_UI_default_configuration ∄- JEDEC
I JEDEC
🖻 Microsemi PolarFire Evaluation Kits
PolarFire Evaluation Kit
MPF300T
MT41K1G8SN-125
Holdfride Conduction Rit Horson MT41K1G8SN-125

5. On the **General** tab, set the **CCC PLL Clock Multiplier** to **6**, and the **DQ Width** to **16**, as shown in Figure 16, page 11.

The clock multiplier value of **6** sets the CCC PLL reference clock frequency to 111.111 MHz. A reference clock of this frequency is required for the PLL present inside the DDR3 subsystem. The PLL generates a 666.666 MHz DDR3 memory clock frequency and a 166.666 MHz DDR3 AXI clock frequency.



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

Figure 16 • DDR3 General Configuration

gui autori	
General Memory Initialization	Memory Timing Controller Misc.
🗆 Тор	
Protocol DDR3	
Generate PHY only	
Clock	
Memory Clock Frequency (MHz)	666.666
CCC PLL Clock Multiplier	6
CCC PLL Reference Clock Frequency	(MHz) 111.111
User Logic Clock Rate	QUAD
User Clock Frequency	166.6665
∃ Topology	
Memory Format	COMPONENT
DQ Width	16 💌
SDRAM Number of Ranks	1
Enable address mirroring on odd rank	is T
DQ/DQS group size	8
Row Address width	16
Column Address Width	11
Bank Address Width	3
Enable DM	DM
Enable Parity/Alert	Г
Enable ECC	Г



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

Figure 17 • DDR3 Controller Configuration

General Memory Initializ	ation Memory Timing	Controller	Misc.
Instance Select			
Instance Number 0]		
User Interface			
Fabric Interface AXI4	•		
AXI Width 64	•		
AXI ID Width 4			
Efficiency			
Enable Activate/Precharge lo	ook-ahead 🗖		
Command queue depth	3	•	
Enable User Refresh Controls	s 🗖		
Address Ordering	Chip-Row-Bank-Col	•	
Misc			
Enable RE-INIT Controls			
ODT Activation Settings of the setting of the se	on Write		
Enable Rank0 - ODT0 🔽	Enable Rank0 - ODT1 厂		
Enable Rank1 - ODT0 厂	Enable Rank1 - ODT1 🔽		
ODT Activation Settings of Control of Con	on Read		
Enable Rank0 - ODT0 🕅	Enable Rank0 - ODT1 厂		
Enable Rank1 - ODT0 厂	Enable Rank1 - ODT1 厂		

7. Retain the default settings for others tabs and click **OK**.



2.4.3.5 Instantiating the AXI3 to AHB-Lite Bridge

The CoreAXItoAHBL IP connects an AXI bus to an AHB-Lite bus, enabling an AXI master to communicate with an AHBL slave/subsystem. To instantiate the CoreAXItoAHBL IP:

- 1. From the Catalog, drag the **CoreAXITOAHBL** IP core to SmartDesign.
- 2. In the **Create Component** dialog box, enter **CORAXITOAHBL_0** as the component name, and click **OK**.
- 3. In the CoreAXITOAHBL Configurator, retain the default configuration, and click OK.

2.4.3.6 Instantiating the AHB-Lite Bus

- 1. From the Catalog, drag the CoreAHBLite IP core to SmartDesign.
- 2. In the **Create Component** dialog box, enter COREAHBLITE_0 as the component name, and click **OK**.
- 3. In the CoreAHBLite Configurator, do the following:
 - From the Memory space list, select 4GB addressable space apportioned into 16 slave slots, each of size 256MB.
 - Under Enable Master Access, select M0 can access slot 6.

This configuration sets the slave address map to 0x6000000-0x6FFFFFF. The peripherals must be connected on this slave interface for the Mi-V processor to access them.

Figure 18 • CoreAHBLite Configuration

Configuration				
Memory space				- 11
Memory space:		4GB addressable space apport	ioned into 16 slave slots, each of size 256MB	
Address range seen by slave connect	ted to huge (2GB) slot interface	e: 🔘 0x00000000 - 0x7FFFFFF	0x8000000 - 0xFFFFFFF	
Allocate memory space to combined region	n slave			
Slot 0: Slot 1:	Slot 2: Slot 3:			E
Slot 4: Slot 5:	Slot 6: Slot 7:			
Slot 8: Slot 9:	Slot 10: 🔲 Slot 11:			
Slot 12: Slot 13:	Slot 14: 🔲 Slot 15:			
Enable Master access				
M0 can access slot 0:	M1 can access	slot 0:	M2 can access slot 0:	
M0 can access slot 1:	M1 can access	slot 1:	M2 can access slot 1:]
M0 can access slot 2:	M1 can access	slot 2:	M2 can access slot 2:]
M0 can access slot 3:	M1 can access	slot 3:	M2 can access slot 3:]
M0 can access slot 4:	M1 can access	slot 4:	M2 can access slot 4:	
M0 can access slot 5:	M1 can access	slot 5:	M2 can access slot 5:	
M0 can access slot 6:	M1 can access	slot 6:	M2 can access slot 6:	
•			_	
Help 🔻			OK Can	cel

4. Click OK.

2.4.3.7 Instantiating the AHB-Lite to APB3 Bridge

The AHB-Lite to APB3 Bridge connects APB peripherals such as UART, SPI and GPIO to AHB-Lite masters. To instantiate the AHB-Lite to APB3 Bridge:

- 1. From the Catalog, drag the CoreAHBtoAPB3 IP core to SmartDesign.
- 2. In the **Create Component** dialog box, enter COREAHBTOAPB3_0 as the component name, and click **OK**.



3. In the CoreAHBtoAPB3 Configurator, retain the default configuration, and click **OK**.

2.4.3.8 Instantiating APB3 Bus

- 1. From the Catalog, drag the CoreAPB3 IP core to SmartDesign.
- 2. In the Create Component dialog box, enter APB3 as the component name, and click OK.
- 3. In the CoreAPB3 Configurator, select the following data width and address configuration settings, as shown in the following figure:
 - APB Master Data Bus Width: 32-bit
 - Number of address bits driven by master: 16
 - Position in slave address of upper 4 bits of master address: [27:24] (Ignored if master address width >= 32 bits)
 - Enabled ABP Slave Slots: Slot 0, Slot 1, and Slot 2.

This configuration sets the slave address map as follows:

- Slot0: 0x0000 0x0FFF
- Slot1: 0x1000 0x1FFF
- Slot2: 0x2000 0x2FFF

Figure 19 • CoreAPB3 Configuration

Number of addre	ess bits driven by m	aster:	16	•
Position in slave	address of upper 4	bits of master addr	ess: $[27:24]$ (Ignored if master address width >= 32 bit	rs) 🔻
Indirect Address	sing:		Not in use	•
cate memory spa	ce to combined regi	on slave		
Slot 0:	Slot 1:	Slot 2:	Slot 3:	
Slot 4:	Slot 5:	Slot 6:	Slot 7:	
Slot 8:	Slot 9:	Slot 10: 🔲	Slot 11:	
Slot 12: 📃	Slot 13:	Slot 14:	Slot 15:	
abled APB Slave S	lots			
Slot 0: 🔽	Slot 1: 🔽	Slot 2: 🔽	Slot 3:	
Slot 4:	Slot 5:	Slot 6:	Slot 7:	
Slot 8:	Slot 9:	Slot 10: 🔲	Slot 11:	
Slot 12: 📃	Slot 13: 🔲	Slot 14: 📃	Slot 15:	
and Lines -	1			

4. Click OK.



2.4.3.9 Instantiating UART Controller

- 1. From the Catalog, drag the CoreUARTapb IP core to SmartDesign.
- 2. In the Create Component dialog box, enter UART_apb as the component name, and click OK.
- 3. In the CoreUARTapb Configurator, retain the default configuration, and click OK.

2.4.3.10 Instantiating the GPIO Controller

- 1. From the Catalog, drag the **CoreGPIO** IP core to SmartDesign.
- 2. In the **Create Component** dialog box, enter Core**GPIO_0** as the component name, and click **OK**.
- 3. In the CoreGPIO Configurator, select the following **Global Configuration** settings, as shown in the following figure:
 - APB Data Width: 32
 - Number of I/Os: 4
 - Single-bit interrupt port: Disabled
 - Output enable: Internal
- 4. Under I/O bit 0, I/O bit 1, I/O bit 2, and I/O bit 3, do the following, as shown in the following figure:
 Select Fixed Config.
 - Set the I/O type as **Output**.
 - Select the interrupt type as **Disabled**.
 - Four GPIO outputs are configured.

Figure 20 • CoreGPIO Configuration

Configuration		
-Global Configuration		
APB Data Width: 32	Number of I/Os: 4	
Single-bit interrupt port: Disabled	Output enable: Internal	
I/O bit 0		
Output on Reset: 0 V Fixed Conf	g: 🔽 I/O Type: Output 💌	Interrupt Type: Disabled
I/O bit 1		
Output on Reset: 0 💌 Fixed Conf	g: 🔽 I/O Type: Output 💌	Interrupt Type: Disabled
-I/O bit 2		
Output on Reset: 0 💌 Fixed Conf	g: 🔽 I/O Type: Output 💌	Interrupt Type: Disabled
I/O bit 3		
Output on Reset: 0 💌 Fixed Conf	g: 🔽 I/O Type: Output 💌	Interrupt Type: Disabled

5. Click **OK** to close the CoreGPIO Configurator.



2.4.3.11 Instantiating CoreSPI

The PolarFire Evaluation board contains two SPI Flash memories. One SPI Flash is connected to the System Controller SPI interface (SC_SPI) for design initialization. The CoreSPI IP is used to interface with the other SPI Flash, which is connected to the fabric I/Os. To instantiate CoreSPI:

- 1. From the Catalog, drag the **CoreSPI** IP core to SmartDesign.
- 2. In the Create Component dialog box, enter SPI_Controller as the component name, and click OK.
- 3. In the CoreSPI Configurator, do the following:
 - Set the APB Data Width to 32
 - In the SPI Configuration section, set the mode to Motorola, frame size to 8, FIFO depth to 32, and clock rate to 16.
 - In the Motorola Configuration section, set the mode to Mode 0, and select the Keep SSEL active check box.

Figure 21 • CoreSPI Configuration

Configuration
APB Data Width: C 8 C 16 🕫 32
-SPI Configuration
Mode: 🕑 Motorola Mode C TI Mode C NSC Mode
Frame Size (4-32): 8
FIFO Depth (1-32): 32
Clock Rate (0-255): 16
Motorola Configuration
Mode: Mode 0 C Mode 1 C Mode 2 C Mode 3
Keep SSEL active 🔽
-TI/NSC Configuration
Transfer Mode: C Normal C Custom
Free running dock
Jumbo frames
NSC Specific Configuration Standard
Testbench: User
License: RTL

4. Click OK.



2.4.3.12 Instantiating PolarFire Clock Conditioning Circuitry (CCC)

The PolarFire Clock Conditioning Circuitry (CCC) block generates a 111.111 MHz clock to the processor subsystem, which is used as a reference clock to the DDR3_0_0 PLL. To instantiate the CCC block:

- 1. From the Catalog, drag the Clock Conditioning Circuitry (CCC) core to SmartDesign.
- 2. In the Create Component dialog box, enter CCC_0 as the component name, and click OK.
- 3. In the Configurator, set the configuration to PLL-Single.
- 4. In the Clock Options PLL tab, do the following:
 - Set the input frequency to 50 MHz.
 - Under Power/Jitter, select Maximize VCO for Lowest Jitter.
 - Set the feedback mode to **Post-VCO**.
 - Set the Bandwidth to **High**.

Figure 22 • CCC Configurator Clock Options PLL Tab

Configuration PLL-Single
Clock Options PLL Output Clocks
Input Frequency
Input Frequency 50 MHz Backup Clock
Bandwidth High = 0.099 MHz
Delay Line
Enable Delay Line
Reference Clock Delay
Delay Steps: 1 🔅
Power / Jitter
Maximize VCO for Lowest Jitter VCO = 4888.88 MHz
C Minimize VCO for Lowest Power
Feedback Mode
Post-VCO 🔽
E Features
SSCG Modulation
Enable Dynamic Reconfiguration Interface (DRI)
I Expose PowerDown Port



- 5. In the Output Clocks tab, under the Output Clock 0 section, do the following:
 - Select the **Enabled** check box to enable PLL output 0.
 - Set the requested frequency to 111.111 MHz.
 - Select the **Global Clock** check box.

Figure 23 • CCC Configurator Output Clocks Tab

Configuration PLL-Single	
Clock Options PLL Output Clocks	
For best results, put the highest frequency first.	
Output Clock 0	
I ← Enabled	
Requested Frequency 111.111 MHz C Actual Lower	111.111 MHz (Actual Higher 111.111 MHz
Requested Phase 0 Degrees C Actual Lower	0 Degrees
Dynamic Phase Shifting Expose Enable Port	Enable Bypass REF_PREDIV
Global Clock Gated)	I HS I/O Clock Dedicated Clock

6. Click **OK** and acknowledge the pop-up.

2.4.3.13 Instantiating PolarFire Initialization Monitor

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

- 1. From the Catalog, drag the PolarFire Initialization Monitor core to SmartDesign.
- 2. In the Create Component dialog box, enter INIT_Monitor as the component name, and click OK.
- In the INIT_MONITOR Configurator > Bank Monitor tab, clear all the check boxes under Calibration Monitor except for BANK1_CALIB_STATUS, and click OK.







2.4.3.14 Instantiating CORERESET_PF

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

- 1. From the Catalog, drag the CORERESET_PF IP.
- 2. In the Component Name dialog box, enter reset_syn_0 as the name of this component, and click **OK**.
- 3. Retain the default configuration for this IP and click OK.
- 4. Similarly, instantiate another instance with reset_syn_1 as its name.

2.4.3.15 Instantiating CoreJTAGDebug

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

- 1. From the Catalog, drag the **CoreJTAGDebug** IP core to SmartDesign.
- 2. In the Create Component window, enter **COREJTAGDebug_0** as the component name, and click **OK**.
- 3. In the Configurator, retain the default configuration, and click OK.

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

Figure 25 • PROC_SUBSYSTEM with All Components Instantiated



2.4.4 Connecting IP Instances in SmartDesign

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

 Using the Connection Mode icon: You can initiate the connection mode in SmartDesign by clicking the Connection Mode icon in the SmartDesign toolbar, as shown in the following figure. The cursor changes from a normal arrow to the shape of the connection mode icon. To make a connection in this mode, click the first pin and drag it to the second pin that you want to connect.

Figure 26 • Connection Method



- Using the Connect option in the Context menu: You can also connect pins by selecting the pins, and then selecting Connect from the context menu. To connect multiple pins, hold down the Ctrl key while selecting the pins. Right-click the input source signal, and select Connect. To disconnect signals, right-click the input source signal, and select Disconnect.
- Right-clicking on a pin provides a list of options like Mark Unused, Edit Slice, Tie Low, Promote to Top-Level, and Tie High. Use these options for individual pins settings.



Figure 27, page 20 shows the Mi-V subsystem in SmartDesign with all IP blocks connected and top-level I/Os.

Figure 27 • Mi-V Subsystem Connected



Note: Grayed out pins are marked unused, green pins are tied Low, and red pins are tied High. Ensure that **unused**, **tied-low**, and **tied-high** pins are strictly set as per Figure 27, page 20.

Follow these steps to connect the IP blocks as per Figure 27, page 20:

- 1. Set the pins as follows on INIT_MONITOR_0:
 - Select FABRIC_POR_N, PCIE_INIT_DONE, USRAM_INIT_DONE, SRAM_INIT_DONE, XCVR_INIT_DONE, USRAM_INIT_FROM_SNVM_DONE, USRAM_INIT_FROM_UPROM_DONE, USRAM_INIT_FROM_SPI_DONE, SRAM_INIT_FROM_SNVM_DONE, SRAM_INIT_FROM_UPROM_DONE, SRAM_INIT_FROM_SPI_DONE, and AUTOCALIB_DONE pins.
 - Right-click the pins, and select Mark Unused.
 - Connect the DEVICE_INIT_DONE pin to reset_syn_0_0:INIT_DONE and BANK_1_CALIB_STATUS pin to reset_syn_1_0:INIT_DONE.
- 2. Set the pins as follows on CCC_0_0:
 - Right-click the REF_CLK_0 pin, and select Promote to Top Level.
 - Connect the other pins as specified in the following table:

Table 2 • CCC_0_0 Pin Connections

Connect From	Connect To
PLL_LOCK_0	reset_syn_0_0:PLL_LOCK and reset_syn_1_0:PLL_LOCK



Table 2 •	CCC_0_0 Pin Connections
-----------	-------------------------

Connect From	Connect To		
OUT0_FABCLK_0	reset_syn_0_0:CLK and reset_syn_1_0:CLK		
	MiV_AXI_0:CLK		
	LSRAM_0:ACLK		
	CORAXITOAHBL_0_0:ACLK CORAXITOAHBL_0_0:HCLK		
	DDR3_0_0:PLL_REF_CLK		
	SPI_Controller_0:PCLK		
	COREAHBLITE_0_0:HCLK		
	COREAHBTOAPB3_0_0:HCLK		
	UART_apb_0:PCLK		
	COREGPIO_0:PCLK AXI4_Interconnect_0:ACLK		

- 3. Set the pins of reset_syn_0_0 as follows:
 - Connect EXT_RST_N pin to DDR3_0_0:CTRLR_READY.
 - Right-click SS_BUSY and FF_US_RESTORE pins and tie them low select Tie Low.
 - Connect the reset_syn_0_0:FABRIC_RESET_N to the following pins:
 - MiV_AXI_0:RESETN

4.

6.

- AXI4_Interconnect_0:ARESETN
- LSRAM_0:ARESETN
- CORAXITOAHBL_0_0:ARESETN
- CORAXITOAHBL_0_0:HRESETN
- COREAHBLITE_0_0:HRESETN
- COREAHBTOAPB3_0_0:HRESETN
- UART_apb_0:PRESETN
- COREGPIO_0:PRESETN
- DDR3_0_0:SYS_RESET_N
- SPI Controller 0:PRESETN
- **Note:** As DDR3_0_0:CTRL_READY pin is connected to reset_syn_0_0:EXT_RST_N, the Mi-V processor is held in reset until the DDR3 controller is ready. The rest of the system is out of reset as soon as device initialization is done.
 - 5. Set the pins of reset_syn_1_0 as follows:
 - Right-click SS_BUSY and FF_US_RESTORE pins and tie them low using the Tie Low option.
 - Select the EXT_RST_N pin and promote it to top level and rename it to resetn.
 - Connect the FABRIC_RESET_N pin to DDR3_0_0:SYS_RESET_N.
 - Set the pins as follows on COREJTAGDebug_0_0:
 - Expand JTAG HEADER.
 - Right-click the TDI, TCK, TMS, and TRSTB pins, and select Promote to Top Level.
 - Expand JTAG HEADER.
 - Right-click the TDO pin, and select **Promote to Top Level**.
 - Connect the other pins as specified in the following table

Table 3 • DEBUG_TARGET Pin Connections

Connect From	Connect to
CoreJTAGDebug_0_0:TGT_TCK	MiV_AXI_0:TCK
CoreJTAGDebug_0_0:TGT_TSRT	MiV_AXI_0:TRST
CoreJTAGDebug_0_0:TGT_TMS	MiV_AXI_0:TMS



Table 3 •	DEBUG_	TARGET	Pin Connections	(continued)
-----------	--------	--------	------------------------	-------------

Connect From	Connect to	_
CoreJTAGDebug_0_0:TGT_TDI	MiV_AXI_0:TDI	_
CoreJTAGDebug_0_0:TGT_TDO	MiV_AXI_0:TDO	_

7. Set the pins as follows on MiV AXI 0:

- Right-click the IRQ[30:0] pin, and select **Tie Low**.
- Right-click the DRV_TDO pin, and select Mark Unused.
- Connect MEM_MST_AXI4 to AXI4_Interconnect_0:AXI4mmaster0.
- Connect MMIO MST AXI4 to AXI4 Interconnect 0:AXI4mmaster1.
- 8. Connect the AXI4 Interconnect 0 pins as specified in the following table.

Table 4 • AXI4_Interconnect_0 Pin Connections

Connect To
DDR3_0_0:SYS_CLK
LSRAM_0:AXI4_Slave
DDR3_0_0:AXI4slave0
CORAXITOAHBL_0_0:AXISlaveIF

- 9. Connect CORAXITOAHBL_0_0:AHBMasterIF to COREAHBLITE_0_0:AHBmmaster0.
- 10. Set the pins as follows on COREAHBLITE_0_0:
 - Right-click the REMAP_M0 pin, and select **Tie Low**.
 - Connect AHBmslave6 to COREAHBTOAPB3_0_0:AHBslave.
- 11. Connect COREAHBTOAPB3_0_0:APBmaster to APB3_0:APB3mmaster.
- 12. Connect the APB3_0 pins as specified in the following table.

Table 5 • APB3_0 Pin	Connections
----------------------	-------------

Connect From	Connect To
APB3_0:APBmslave0	UARTapb_0:APB_bif
APB3_0:APBmslave1	COREGPIO_0:APB_bif
APB3_0:APBmslave2	SPI_Controller_0:APB_bif

- 13. Set the pins as follows on DDR3_0_0:
 - Right-click the PLL_LOCK output pin, and select **Mark Unused**.
 - Right-click the CTRLR_READY pin, and select **Promote to Top Level** for debug purpose. The CTRLR_READY signal is used to monitor the status of the DDR controller.
- 14. Set the pins as follows on SPI_Controller_0:
 - Right-click the SPISSI pin, and select Tie High.
 - Right-click the SPICLKI pin, and select Tie Low.
 - Right-click the SPIINT, SPIRXAVAIL, SPITXRFM, SPIOEN, and SPIMODE pins, and select Mark Unused.
 - Right-click the SPISDI, SPISCLKO and SPISDO pins, and select Promote to Top Level.
- 15. Right-click the SPISS[7:0] pin, select Edit Slices, and edit the slices shown in the following figure.
- **Note:** In this tutorial, a single SPI Flash is used. Hence, while settings the pins of the SPI_Controller_0 block, we need only 0th bit of the SPISS. Bits 1:7 need to be sliced and marked as unused.



Figure 28 • Edit Slices Window

Edit Slices - SPISS[7:0]			?	×	
Create 8 slices of width 1 Add Slices					
SPISS[7:0] D		Left	Right		
	1	0	0		
	2	7	1		
Help		0	к	Cancel	

- Right-click the SPISS[7:1] pin, and select Mark Unused.
- Right-click the SPISS[0] pin, and select Promote to Top Level.
- 16. Set the pins as follows on UARTapb_0:
 - Right-click the RX and TX pins, and select **Promote to Top Level**.
 - Right-click the TXRDY, RXRDY, PARITY_ERR, OVERFLOW, FRAMING_ERR pins, and select **Mark Unused**.
- 17. Set the pins as follows on GPIO_0:
 - Right-click the GPIO_IN[3:0] pin, and select **Tie Low**.
 - Right-click the INT[3:0] pin, and select Mark Unused.
 - Right-click the GPIO_OUT[3:0] pin, and select **Promote to Top Level**.
- 18. Right-click the PROC_SUBSYSTEM SmartDesign canvas, and select Auto Arrange Layout.
- 19. Click File > Save PROC_SUBSYSTEM.

The IP blocks are successfully connected. Figure 27, page 20 shows all the IP blocks of the Misubsystem connected.

After the Mi-V processor subsystem is successfully designed in SmartDesign, you can view the system address map by right-clicking the SmartDesign canvas and selecting **Modify Memory Map**. The following figure shows the address map for the COREAHBLITE_0 peripherals.

Figure 29 • AHBLite_0 Peripherals Address Map

题 Modify Memory Map		>	×
Select Bus to View or Assign Peripheral(s)	As	sign peripherals to addresses on bus:	
AXI4_Interconnect_0	Address	Peripheral	
	0x6000000	COREAHBTOAPB3_0_0:AHBslave	
APB3_0			
Help		OK Cancel	

The following figure shows the address map for the APB3_0 peripherals.



Figure 30 • APB3_0 Peripherals Address Map

5 Modify Memory Map		×
Select Bus to View or Assign Peripheral(s)	As	sign peripherals to addresses on bus:
AXI4_Interconnect_0	Address	Peripheral
COREAHBLITE_0_0	0x60000000	UART_apb_0:APB_bif
APB3_0	0x60001000	COREGPIO_0_0:APB_bif
	0x60002000	SPI_Controller_0:APB_bif
Help		OK Cancel

Note: The AXI memory map is not supported. For more information about the AXI memory map, see Instantiating AXI Interconnect Bus IP, page 6.

2.4.5 Generating SmartDesign Component

To generate the SmartDesign component:

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

Figure 31 • Generate Component Icon

г			2		1	
L		÷	2	Ξ	٩	6
г	-	1	с			
		s		-	2	ş

When the Mi-V component is generated, the **Message** window displays the message, "The PROC_SUBSYSTEM was generated successfully."

2.4.6 Managing Timing Constraints

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

2.4.6.1 Deriving Constraints

To derive constraints:

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

Figure 32 • Derive Constraints Button

ſ	I/O Attributes V Timing V Floor Planner V Netlist Attributes								
	New Import	Link Edit with Co	onstraint Editor 💌	Check 👻	<u> Derive</u> Constraints				
			Synthesis	Place and Rou	te Timing Verification				

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

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



Figure 33 • Derived Constraints

5	PROC_SUBSYSTEM A × Reports A × C	Constraint Manager	₽×							
L	I/O Attributes / Timing / Floor Planner / Netlist Attributes /									
	New Import Link Edit with	Constraint Editor	Check 🔻 De	rive Constraints) Constraint Coverage 💌 Help						
		Synthesis	Place and Route	Timing Verification						
	constraint\PROC_SUBSYSTEM_derived_constraints.sdc	\checkmark	\checkmark							

3. Save the project.

2.4.6.2 Importing Other Constraint Files

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

To import and map the constraint files:

- 1. On the **Timing** tab, click **Import**.
- 2. Navigate to the DesignFiles_directory\Source folder, and select the timing user constraints.sdc file.
- Select the Synthesis, Place and Route, and Timing Verification check boxes next to the timing_user_constraints.sdc file. This constraint file defines that the CCC_0_0 output clock and DDR3_0_0 AXI clock are asynchronous clocks.
- 4. Save the project.

2.4.7 Running the Libero Design Flow

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

- Synthesis, page 25
- Place and Route, page 25
- Verify Timing, page 27
- Generate FPGA Array Data, page 27
- Configure Design Initialization Data and Memories, page 27
- Generate Design Initialization Data, page 30
- Generate Bitstream, page 31
- Run PROGRAM Action, page 31
- Generate SPI Flash Image, page 33
- Run PROGRAM_SPI_IMAGE Action, page 34

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

2.4.7.1 Synthesis

To synthesize the design:

- Double-click Synthesis on the Design Flow tab. When the synthesis is complete, a green tick mark appears next to Synthesize.
- 2. Right-click **Synthesize** and select **View Report** to view the synthesis report in the **Reports** tab.

2.4.7.2 Place and Route

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

- Selecting the already imported io constraints.pdc file
- Placing the DDR3_0_0 block using the I/O Editor
- Ensuring all the I/Os are locked

To complete these steps and to place and route the design:

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



Figure 34 • I/O Attributes

I/	O Attributes Timing Floor P	Timina Floor Planner Netist Attributes Import Link Edit View Check Help						
	New Timport	Link	Edit	-	View	Check	Help	
				Plac	e and Route			
	constraint\io\io_constraints.	odc		~				

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

Figure 35 • Edit with I/O Editor Option

I/O Attributes Timing Floor Planner Netlist Attributes							
New 🔻 Import Link	Edit Check	Help					
	ing V Floor Planner V Netlist Attributes import Link Edit Check Help Edit with I/O Editor						

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

Figure 36 • Port View

Port View [active]		ort View [active] 🛛 🗗	Pin View 🗗	Memory	View 🗗	IOD	View 🗗	XC	VR. View 🗗		Package View	8
Port Name		1	Direction	י 💌	I/O Standar	d 💌	Pin Number	•	Locked 🔻			
	1	CTRLR_READY			Output		LVCMOS18	3	C27			

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

Figure 37 • I/O Editor Design View – DDR3 Selection





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

Figure 38 • Memory View [active] Tab with DDR3 Subsystem Placement



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

Figure 39 • DDR3_0 Placed

Port	View 🗗 🛛 Pin View 🗗	XCVR View 🗗 Memory View [active]	8
Mem	ory Type: DDR3 💌		
	Port Function 1	Port Name 💌	Pin N
1	NORTH_NE	DDR3_0_0(width=16, rate=1333.33)	
146	NORTH_NW	Unassigned	
291	SOUTH_SE	Assigned	
364	SOUTH_SW	Unassigned	
458	▶ WEST_NW	Unassigned	
599	WEST_SW	Unassigned	

- 7. From I/O Editor Port View tab, check if there are any unlocked I/Os, and lock them as mapped in the io_constraints.pdc file available in the Design_Files_Directory\Source\io folder.
- 8. Click Save.
- 9. Close the I/O Editor.

A user.pdc file is created for DDR3_0_0 block in the **Constraint Manager** > I/O Attributes and **Floor Planner** tabs.

- Note: DDR3_0_0 can also be placed using the fp_constraints.pdc. Import the fp_constraints.pdc from Constraint Manager > Floor Planner tab and select the place and route option after synthesis. This constraint file is available in the Design Files Directory\Source\fp folder.
 - Double-click Place and Route from the Design Flow tab.
 When place and route is successful, a green tick mark appears next to Place and Route.

2.4.7.3 Verify Timing

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

2.4.7.4 Generate FPGA Array Data

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

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

2.4.7.5 Configure Design Initialization Data and Memories

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



This process requires the user application executable file (HEX file) as input to initialize the LSRAM blocks after device power-up. The hex file is provided with the design files. For more information about building the user application, see Building the User Application Using SoftConsole, page 35.

Note: To make the HEX file generated by SoftConsole compatible with the process of configuring design initialization data and memories in Libero, delete the extended linear record present in the first line of the HEX file. The HEX file available in the DesignFiles Directory\Source folder is already modified to be compatible.

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

- 1. Double-click Configure Design Initialization Data and Memories on the Design Flow tab.
- 2. On the Fabric RAMs tab, select PROC_SUBSYSTEM/LSRAM_0 from the list of logical instances, and click Edit, as shown in the following figure. The PROC SUBSYSTEM/LSRAM 0 instance is the Mi-V processor's main memory. The System Controller initializes this instance with the imported client at power-up.

Figure 40 • Fabric RAMs Tab

C	lient									
	Loai	oed design configuration Edt Initialize all clents from: Initialize all Clents from aVVM 💌								
		Logical Instance Name	PORTA Depth * Width	PORTB Depth * Width	Memory Content					
	22	DDR3_0_0/MSC_i_0/MSC_i_1/MSC_i_213/MSC_i_217/MSC_i_218/MSC_i_221/mem[63:0]	32x64	32x64	No content					
ſ	23	DDR3_0_0/MSC_i_0/MSC_i_1/MSC_i_8/MSC_i_104/MSC_i_135/mem[128:0]	1024x129	1024x129	No content					
ſ	24	DDR3_0_0/MSC_i_0/MSC_i_1/MSC_i_8/MSC_i_13/MSC_i_16/mem[76:0]	256x73	256x73	No content					
ſ	25	DDR3_0_0/MSC_i_0/MSC_i_1/MSC_i_8/MSC_i_17/MSC_i_20/mem(5:0)	16x2	16x2	No content					
ſ	26	DDR3_0_0/MSC_i_0/MSC_i_1/MSC_i_8/MSC_i_21/MSC_i_50/mem[144:0]	512x145	512x145	No content					
ſ	27	DDR3_0_0/MSC_i_0/MSC_i_1/MSC_i_8/MSC_i_51/MSC_i_71/mem[94:0]	16x95	16x95	No content					
ſ	28	DDR3_0_0/MSC_i_0/MSC_i_1/MSC_i_8/MSC_i_72/MSC_i_103/mem[45:0]	1024x46	1024x46	No content					
	29	DDR3_0_0/MSC_i_0/MSC_i_1/MSC_i_4/MSC_i_8/MSC_i_9/MSC_i_12/mem[69:0]	512x70	512x70	No content					
	30	LSRAM_0	8192x80	8192x80	No content					
T	31	MiV_AXI_0/MiV_AXI_0/ChiselTop0/tile/core/_T_1151[31:0]	31x32	31x32	No content					
	32	MiV_AXI_0/MiV_AXI_0/ChiseITop0/tile/core/_T_1151_1[31:0]	31x32	31x32	No content					
	33	MiV_AXI_0/MiV_AXI_0/ChiselTop0/tile/dcache/MIV_RV32IMA_L1_AXI_TAG_ARRAV/MIV_RV32IMA_L1_AXI_TAG_ARRAY_EXT/ram[20:0]	128x21	128x21	No content					
	34	MiV_AXI_0//MiV_AXI_0/ChiselTop0/tile/dcache/data/MIV_RV32IMA_L1_AXI_DATA_ARRAYS_0/MIV_RV32IMA_L1_AXI_DATA_ARRAYS_0_EXT/ram[7:0]	2048x8	2048x8	No content					
	35	MiV_AXI_0//MiV_AXI_0/ChiselTop0/tile/dcache/data/MIV_RV32IMA_L1_AXI_DATA_ARRAYS_0/MIV_RV32IMA_L1_AXI_DATA_ARRAYS_0_EXT/ram_1[7:0]	2048x8	2048x8	No content					
	36	MiV_AXI_0//MiV_AXI_0/ChiselTop0/tile/dcache/data/MIV_RV32IMA_L1_AXI_DATA_ARRAYS_0/MIV_RV32IMA_L1_AXI_DATA_ARRAYS_0_EXT/ram_2[7:0]	2048x8	2048x8	No content					
1										

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

Figure 41 • Edit Fabric RAM Initialization Client Dialog Box

🙆 Edit Fabric	RAM Initialization Client	?	×
Client name;	LSRAM_0		,
Physical Name:	HB_AXI_0/LSRAM_PF_TPSRAM_AHB_AXI_0_PF_TPSRAM_R9C0/INST_RA	M1K20_1	(P
RAM Initializa	tion Options		
C Initialized	Content from Synthesis		
Content fr	rom file:		
C Content fi	lled with 0s		
C No conten	t (dient is a placeholder and will not be programmed)		
Optimize for: 🤇	High Speed 🕫 Low power		
Storage Type	sNVM 🔽		
	SNVM		
Help	OK	Close	

4. In the Import Memory File dialog box, locate the MiV uart blinky.hex file from DesignFiles_directory\Source folder. Select the "Use relative path from project directory" option.



Figure 42 • Import Memory File Dialog Box

🕑 Import Memory File	2						?	×
Look in: D:\mp	f_tu0775_liberosoc_df\Source				- 0 0	0	Ø	:: 🔳
S My Computer	Name	Δ	Size	Туре	Date Modified			
130827	fp io			Filder Filder	9/23/2:15 PM 9/23/2:15 PM			
	MiV_uart_blinky.hex		9 KB	hex File	9/20/2:26 PM	J		
File name: MiV_uart_	blinky.hex						C	ipen
Files of type: Intel-Hex((*.hex *.ihx)					•	C	ancel
🔍 Use absolute path (fil	e will not be copied if you move th	ne design)						
Use relative path from	n project directory							
Copy memory file to p	project directory					_		

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

Figure 43 • Fabric RAMs Tab - Apply Button

Design Initialization uPROM sNVM SPI Flash Fabric RAMs*

Apply Discard Help	- Cliente				
LSRAM Memory Available Memory(Bytes): 2437120 Lised Memory(Bytes): 184320	Load design configuration Edit In	nitialize all clients from: User Selec	tion	•	
Free Memory(Bytes) : 2252800	Logical Instance Name	PORTA PORTB Depth * Width Depth * Widtl	Memory Content	Storage Type	Memory Source



- In the Design Initialization tab, under Third stage (uPROM/sNVM/SPI-Flash), select the SPI-Flash

 No-binding Plaintext option is selected and ensure that the SPI Clock divider value is set to 6, as shown in the following figure. This means that the imported user application will be written to SPI-Flash without encryption and authentication.
- **Note:** The SPI Clock divider value specifies the required SPI SCK frequency to read the initialization data from SPI Flash. The SPI Clock divider value must be selected based on the external SPI Flash operating frequency range.
 - 8. Click Apply.

Figure 44 • Design Initialization Data

esign Initialization* UPROM SNVM SPI Flash Fabric RAMs
Apply Discard Hep In design initialization, user design blocks such as LSRAM, µSRAM, transceivers, and PCIe can be initialized as an option using data stored in the non-volatile storage memory. The initialization data can be stored in µRCM, sNVM, or an external SPI Flash.
Follow the below steps to program the initialization data: 1. Set up your fabric RAMs initialization data; if any, using the 'Fabric RAMs' tab 2. Define the storage location of the initialization data 3. Generate the initialization dents 4. Generate or export the bitstream 5. Program the device Device initialization excellenting
First stage (sNVM) In the first stage, the initialization sequence de-asserts FABRIC_POR_N.
In the second stage, the initialization sequence initializes the PCIe and XCVR blocks present in the design. Start address for second stage initialization client: 0x 00000000
Third stage (sNVM/uPROM/SPI-Flash)
In the third stage, the initialization sequence initializes the Fabric RAMs present in the design.
To save the initialization instructions in sNVM/uPROM/SPI-Flash, please use 'Fabric RAMs' tab to make your selection for each RAM client.
∀ Start address for sNVM dients: 0x 00000000 sNVM start page: 0
Start address for UPROM dients: 0x 00000000
✓ Start address for SPI-Flash dients: 0x 00000400
SPI-Flash Binding: SPI-Flash - No-binding Plaintext
✓ Broadcast instructions to initialize RAM's to zero's

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

2.4.7.6 Generate Design Initialization Data

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

Info: 'Generate design initialization data' has completed successfully. Info: Stage 1 initialization client has been added to sNVM. Info: Stage 3 initialization client has been added to SPI.

2. Click the **SPI Flash** tab to verify that the bin file has been added, as shown in the following figure. The PolarFire Evaluation Board uses a Micron SPI flash. Therefore, ensure that Micron is selected in the **Manufacturer** list.



Figure 45 • SPI Flash Tab

Design Initialization uPROM sNV	M SPI Flash	Fabric RAMs					
Apply Discard	Help						
Enable Auto Update							
Manufacturer: MICRON 🔻	Part No: MT	25QL01GBBB8ESF-0SIT					
Usage statistics	SPI Flash Cli	ents					
Available memory (KB): 131071 Add V Edit Delete							
Free memory (KB) : 131058	Program	Name	Туре	Index	Content File	Start Address	End Address
	$\overline{\vee}$	INIT_STAGE_3_SPI_CLIENT	Design Initialization		designer\PROC_SUBSYSTEM\PRO	0x400	0x344f

Note: For more information about design initialization, see *UG0725: PolarFire FPGA Device Power-Up and Resets User Guide.*

2.4.7.7 Generate Bitstream

To generate the programming bitstream:

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

2.4.7.8 Run PROGRAM Action

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

2.4.7.8.1 Board Setup

To set up the board:

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

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

Table 6 •Jumper Settings

- **Note:** For more information about the Jumper locations on the board, see the silkscreen provided in *UG0747: PolarFire FPGA Evaluation Kit User Guide.*
 - 2. Connect the power supply cable to the **J9** connector on the board.
 - 3. Connect the host PC to the **J5** (USB) port on the PolarFire Evaluation Board using the USB cable.
 - 4. Power on the board using the SW3 slide switch.



2.4.7.8.2 Serial Terminal Emulation Program (PuTTY) Setup

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

Follow these steps to set up the serial terminal:

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

Figure 46 • COM Port Number



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

Figure 47 • Connection Type Selection

Session	Basic options for your Pu	TY session
⊡ Logging ⊡ Terminal ⊡ Keyboard	Specify the destination you want to Serial line	connect to Speed
- Window	Connection type:	SSH Serial
Appearance Behaviour Translation Selection	Load, save or delete a stored session Saved Sessions	n
Colours Connection Data Proxy Telnet Rlogin	Default Settings 57600 com8_115200	Load Save Delete
erial	Close window on exit: ◎ Always ◎ Never ◎ Onl	y on clean exit



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

Figure 48 • PuTTY Configuration

tegory:					
Session	Options controlling local serial lines				
····· Logging ⊐·· Terminal ···· Keyboard	Select a serial line Serial line to connect to	COM93			
Bell	Configure the serial line				
⊒ Window	Speed (baud)	115200			
Appearance Behaviour	Data bits	8			
···· Translation	Stop bits	1			
Selection Colours	Parity	None 🔻			
Connection	Flow control	None 🔻			
Proxy Telnet Rlogin SSH					

6. Click Open.

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

2.4.7.8.3 Running the PROGRAM Action

To run the PROGRAM action:

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

2.4.7.9 Generate SPI Flash Image

To generate the SPI flash image:

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



2.4.7.10 Run PROGRAM_SPI_IMAGE Action

To program the SPI image:

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

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

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

Figure 49 • Hello World String

Н	ello World!		

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

Figure 50 • Hello World String After the Board is Power Cycled

Hello World! Hello World!



3 Building the User Application Using SoftConsole

This section describes how to build a RISC-V user application executable (.hex) file and debug it using SoftConsole v6.1 or higher.

Building the user application involves the following steps:

- Creating a Mi-V SoftConsole Project, page 35
- Downloading the Firmware Drivers, page 37
- Importing the Firmware Drivers, page 39
- Creating the main.c File, page 41
- Mapping Firmware Drivers and the Linker Script, page 42
- Mapping Memory and Peripheral Addresses, page 48
- Setting the UART Baud Rate, page 50
- Building the Mi-V Project, page 50

3.1 Creating a Mi-V SoftConsole Project

To create a Mi-V SoftConsole project:

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

Figure 51 • Workspace Launcher

Workspace: D:/Tutorial/MiV_Workspace	~	Browse	
 Use this as the default and do not ask again Recent Workspaces 			
	La	unch	Cancel

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

- 4. Select File > New > Project, as shown in the following figure.
- Figure 52 New C Project Creation

File	Edit	Source	Refactor	Navigate	Search	Project	Ru	n Window	Help
	New				Alt+Sh	ift+N >	C.	Makefile Pro	ject with Existing Code
	Open File		C	C/C++ Proje	ect				
	Open	Projects f	rom File Sy	stem			Ľ	Project	

5. Expand C/C++ and select C Project in the New Project dialog box, and click Next.



- 6. In the C Project dialog box, do the following:
 - Enter a name for the project in the Project name field. For example, MiV_uart_blinky.
 - In the **Project type** pane, expand **Executable**, and select **Empty Project** and **RSIC-V Cross GCC**, as shown in the following figure. Then, click **Next**.

Figure 53 • C Project Dialog Box

sc			
C Project Create C project of selected type			Ź
Project name: MiV_uart_blinky			
Use <u>d</u> efault location			
Location: D:\Tutorial\MiV_Workspace\N	/liV_uart_blinky		B <u>r</u> owse
Project type:	Toolchains:		
 Executable Empty Project Hello World ARM C Project Hello World RISC-V C Project Shared Library Static Library Makefile project 	ARM Cross GCC RISC-V Cross GCC	platfo	prm
? < <u>B</u> ack	<u>N</u> ext > <u>F</u> inish		Cancel

- 7. After selecting the platforms and configurations you want to deploy, click **Next**.
- Figure 54 Select Configurations Dialog Box

SC C Project		- 0	×
Select Config Select platform		Ŷ	
Project type: Toolchains: Configurations:	Executable RISC-V Cross GCC		
♥ ⓑ Debug ♥ ⓑ Release		Select all	II
Use "Advanced	settings" button to edit project's properties.	Advanced sett	ings
Additional conf Use "Manage co	igurations can be added after project creation. onfigurations" buttons either on toolbar or on property page	25.	
?	< Back Next > Finish	Cance	el

8. Click Finish in the GNU RISC-V Cross Toolchain wizard.



An empty Mi-V project (MiV_uart_blinky) is created, as shown in the following figure.

Figure 55 • Empty MI-V Proje	X		
File Edit Source Refactor Navi	gate Search Project Run Window Help		
📑 👻 🔚 🐚 🛞 👻 🇞 👻 🛗 🔯	• 🚳 🕶 🖻 🕶 🎯 🕶 🐎 🕶 💽 🕶 🚱 🕶 🔒 🕶	😕 🗀 🛷 🛨 📴 💷 🖉 🖛 😫	2 🔌 🕹
▲ ▲ ▲ ◆ ◆ ★ ◆ ▲ ●			Quick Access 🔡 🖻
陷 Project Explorer 🛛 🗖 🗖			□ 🗄 O 🛛 🔭 🗆 🗆
> 🖆 MiV_uart_blinky 🔍 🖗 ♥			P ~ An outline is not available.
	🖹 Problems 🛱 🚈 Tasks 💷 Console 🗔 Prop	perties	⊉ 😜 ∨ 🗆 🗖
	0 items		
	Description	Resource	Path Location
	1		×
MiV upt blipky		1	

3.2 Downloading the Firmware Drivers

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

- CoreGPIO
- CoreUARTapb
- CoreSPI Driver

Download the RISC-V HAL files and drivers using the Firmware Catalog application.



To download the drivers:

- 1. Create a folder named **firmware** in the Mi-V project workspace.
- 2. Open Firmware Catalog. The following figure shows the Firmware Catalog window.

Figure 56 • Firmware Catalog Window

View (<u>53/167</u>):			Search by all fields (53/53):
🛛 🌠 All 🛛 😨 Vault 🔍 Web reposi	tories		
display only the latest version of a core		1	
Name 🛆	Version	Size (MB)	Status
CoreMMC Driver	3.0.101	1.54	
CoreMMC Driver	2.0.100	0.62	
CorePWM Driver	2.4.100	1.79	
CoreSDLC Driver	2.1.100	1.27	
CoreSPI Driver	3.3.100	1.94	
CoreSysServices_PF Driver	2.0.102	0.87	
CoreTSE Driver	2.5.100	5.55	
CoreTimer Driver	2.4.100	1.22	
CoreUARTapb Driver	3.3.101	1.22	
CoreWatchdog Driver	2.2.100	0.50	
Cortex-M1 CMSIS Hardware Abstraction	2.0.105	9.81	
Hardware Abstraction Layer (HAL)	2.3.102	0.22	
PolarFire PCIe Driver	2.1.100	0.82	
PolarFire Transceiver Driver	2.0.106	0.54	
PolarFire User Crypto Driver	2.2.102	3.05	
RISC-V Hardware Abstraction Layer (HAL)	2.2.103	1.23	
SmartFusion CMSIS-PAL	2.4.102	0.44	🖼 Generate
SmartFusion MSS ACE Driver	2.3.105	2.22	M Dama and farma analt
SmartFusion MSS Ethernet MAC Driver	3.1.102	4.13	Kemove from vault
SmartFusion MSS GPIO Driver	2.1.100	1.00	
SmartFusion MSS I2C Driver	3.1.101	1.07	Show details
SmartFusion MSS IAP Driver	2.3.100	1.67	Open documentation
SmartFusion MSS Peripheral DMA Driver	2.0.102	0.70	Generate sample project

- 3. If new cores are available, click Download them now!
- 4. Right-click RISC-V Hardware Abstraction Layer (HAL), and select Generate.
- 5. In the Generate Options window, enter D:\Tutorial\MiV_Workspace\firmware as the project folder, and click OK.

When the files are generated, the Reports window lists the files generated, as shown in the following figure.

Figure 57 • RISCV HAL Files Report

Files generated in 'D:\Tutorial\WIV_Workspace\firmware': hal\pu_types.h hal\hal_assert.h hal\hal_ral_arg.c hal\hww_macros.h hal\hw_reg_access.h hal\hw_reg_access.h hal\hw_reg_access.s riscv_hal\encoding.h riscv_hal\enco

- 6. Right-click CoreUARTapb Driver, and select Generate.
- In the Generate Options window, enter D:\Tutorial\MiV_Workspace\firmware as the project folder, and click OK.

When the files are generated, the **Reports** window lists the files, as shown in the following figure.

Figure 58 • CoreUARTapb Files Report

Files generated in 'D: \Tutorial \MiV_Workspace \firmware':

drivers\CoreUARTapb\coreuartapb_regs.h drivers\CoreUARTapb\core_uart_apb.c drivers\CoreUARTapb\core_uart_apb.h

- 8. Right-click CoreGPIO Driver, and select Generate.
- 9. In the Generate Options dialog box, enter D:\Tutorial\MiV_Workspace\firmware as the project folder, and click OK.



When the files are generated, the **Reports** window lists the files, as shown in the following figure.

Figure 59 • CoreGPIO Files Report

```
Files generated in 'D:\Tutorial\MiV_Workspace\firmware':
drivers\CoreGPIO\coregpio_regs.h
drivers\CoreGPIO\core_gpio.c
drivers\CoreGPIO\core_gpio.h
```

- 10. Right-click CoreSPI Driver, and select Generate.
- 11. In the Generate Options window, enter D:\tutorial\MiV_Workspace\firmware as the project folder, and click OK.

When the files are generated, the Reports window lists the files, as shown in the following figure.

Figure 60 • CoreSPI Driver Files Report

Files generated in 'D:\Tutorial\MiV_Workspace\firmware':

drivers\CoreSPI\corespi_regs.h drivers\CoreSPI\core_spi.c drivers\CoreSPI\core_spi.h

The RISC-V HAL and firmware drivers are generated.

3.3 Importing the Firmware Drivers

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

To import the drivers:

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

Figure 61 • Import Option





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

Figure 62 • Import Dialog Box

SC Import		×
Select Import resources from the local file system into an existing project.	Ľ	1
Select an import wizard:		
type filter text		
 ✓ ➢ General ▲ Archive File ➢ Existing Projects into Workspace ➢ File System ➢ Projects from Folder or Archive ➢ Projects from Folder or Archive ➢ C/C++ ➢ Git ➢ Install ➢ Run/Debug ➢ Team 		

- 3. On the next page of the **Import** dialog box, do the following (see Figure 63, page 40):
 - Click Browse, and locate the D:\Tutorial\MiV Workspace\firmware folder.
 - Select the firmware folder, and click OK.
 - Expand the **firmware** folder, and select the **drivers**, **hal**, and riscv_hal folders.
 - Click Finish.

Figure 63 • Import Dialog Box - Page 2

SC Import			
File system Import resources from the local file system.			
From directory: D:\Tutorial\MiV_Workspace\firm	nware	~	B <u>r</u> owse
 ✓ ⊘ ⇒ firmware ✓ ⊘ ⇒ drivers > ⊘ ⇒ CoreGPIO > ⊘ ⇒ CoreVARTapb ✓ ⇒ filelist > Ø ⇒ hal > Ø ⇒ riscv_hal 			
Filter Types Select All Deselect All Into folder: MiV_uart_blinky	JI		Bro <u>w</u> se
Options Overwrite existing resources without warning Create top-level folder Advanced >>			

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



3.4 Creating the main.c File

To update the main.c file:

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

Figure 64 • main.c File Creation

SC New Sourc	e File						×
Source File Create a new s	source file.					C	
Source folder:	MiV_uart_blin	у.	 			Browse	
Source file:	main.c						
Template:	Default C sour	ce template			\sim	Configu	e
?				Finish		Cance	I

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

Figure 65 • The main.c file

<u>File Edit Source</u>	Refac <u>t</u> or	<u>N</u> avigate	Se <u>a</u> rch	<u>P</u> roject	<u>R</u> un	<u>W</u> indow	<u>H</u> elp	
i 📬 🗕 🔚 👘 i 🛞 i	• 🔨 •	🗟 🔂 🔻	<u>6</u>	c - G	- ∦	s 🕶 🚺 🕶	° 💁 🔹 🤔	s?
Project Explorer 🔅	3	- 6	1	*main.c 🛛	3			
		E 🚯 🕚	7 1	ī				
✓ [™] MiV_uart_blinky [™]	/							
> 🛐 Includes								
> 👝 drivers								
> 👝 hal								
> 👝 riscy hal								
> 💽 main.c								

- 3. Copy all of the content of the DesignFiles_directory\Source\main.c file, and paste it in the main.c file of the SoftConsole project.
- 4. Save the SoftConsole main.c file.



3.5 Mapping Firmware Drivers and the Linker Script

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

Figure 66 • Unresolved Header Files

Project Explorer 🛛		
 ✓ [™] MiV_uart_blinky > MiV_uart_blinky > → Includes > → drivers > → filelist > → hal 	□ 🔄 🔻	<pre>1⊖/************************************</pre>
> 🔁 riscv_hal > 🔁 main.c		<pre>9 #include "riscy_hal.h" 9 10 #include "core_uart_apb.h" 9 11 #include "core_gpio.h" 9 12 #include "sample_hw_platform.h" 9 13 #include "hw_reg_access.h" 14</pre>

To map the drivers and HAL files:

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

Figure 67 • C/C++ Build Settings

S	ettings		
ſ			
l	Configuration:	[All configurations]	Manage Configurations



- 4. In the **Tool Settings** tab, expand **Target Processor**, and select the following settings:
 - Architecture: RV32I(-march=rv32i*)
 - Integer ABI: ILP32(-mabi=il32*)
 - Multiply extension: Enabled

Figure 68 • Target Processor Tool Settings

 Optimization Warnings Debugging GNU RISC-V Cross Assembler Preprocessor Includes 	Multiply extensi Atomic extensic Floating point Compressed ext	on (RVM) on (RVA) None
 Warnings Miscellaneous GNU RISC-V Cross C Compiler Preprocessor Includes Optimization Warnings Miscellaneous GNU RISC-V Cross C Linker General Libraries Miscellaneous GNU RISC-V Cross Create Flash Image General General GNU RISC-V Cross Print Size General 	Floating point ABI Tuning Code model Small data limit Small prologue, Force string ope Allow use of PLT Floating-point c Integer divide in Other target flags	None Image: Second Se

5. Expand GNU RISC-V Cross C Compiler, and select Includes.



6. Click **Add** to add the driver and Mi-V HAL directories, as shown in the following figure.

Figure 69 • GNU RISC-V Cross C Compiler Tool Settings

🛞 Tool Settings 🛞 Toolchains 🔳 Devices 🎤	🖱 Build Steps 🛛 🙅 Build Artifact 🛛	🗟 Binary Parsers 🚺 🔸
 Target Processor Optimization Warnings Debugging Solut RISC-V Cross Assembler Preprocessor Includes Warnings Miscellaneous 	Include paths (-I)	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 SINU RISC-V Cross C Compiler Preprocessor Optimization Warnings Wiscellaneous S GNU RISC-V Cross C Linker General Libraries Miscellaneous Solution 	Include system paths (-isystem)	원 🌒 🗟 👸 🖞
실 General <	Include files (-include)	월 🌒 🗟 중니 &

- Note: This application does not require including system paths and other files.
 - 7. In the Add directory path dialog box, click Workspace, as shown in the following figure.

Figure 70 • Add Directory Path Dialog Box

SC Add dired	ctory path			×
Directory:				
	OK	Cancel	Workspace	File system

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



Figure 71 • CoreGPIO Folder Selection

SC Folder selection			×
Select one or more Workspace Folder	rs		
 MiV_uart_blinky Settings Debug drivers CoreGPIO CoreSPI CoreUARTapb hal Release riscv_hal RemoteSystemsTempFiles 			
?	ОК	Cancel	

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

Figure 72 • Tool Settings Tab with CoreGPIO Path Added

ype filter text	Settings			() ▼ () ▼
Resource Audit ders				
 ∠/C++ Build Build Variables Environment 	Configuration:	Debug [Active]		✓ Manage Configurations
Logging Settings	Tool Setting	as 🎤 Build Steps 🚇 Buil	d Artifact 🗟 Binary Parsers 🔕 Err	or Parsers
Tool Chain Editor	🖄 Target	t Processor	Include paths (-I)	🗐 🗿 🖓 취 문)
C/C++ General Linux Tools Path	Debug 🖄 Additi	gging onal Tools	"\${workspace_loc:/\${Pro	Name}/drivers/CoreGPIO}"

10. Repeat the preceding steps to add the CoreUARTapb, CoreSPI, hal, and riscv_hal folder paths. The drivers and riscv-HAL files are successfully mapped, as shown in the following figure.





Include paths (-1) Include paths (-1) Includes Includes Includes Includes Includes Includes Includes Includes Includes Includes Includes Include files (-include) Include files (-include) Include files (-include)	🛞 Tool Settings 🛞 Toolchains 🎤 Build Steps	P Build Artifact 🗟 Binary Parsers 🔕 Error Parsers	
 Miscellaneous S GNU RISC-V Cross Create Flash Image General GNU RISC-V Cross Print Size General 	 Target Processor Optimization Warnings Debugging S GNU RISC-V Cross Assembler Preprocessor Includes Warnings Miscellaneous S GNU RISC-V Cross C Compiler Preprocessor Includes Optimization Warnings Wiscellaneous S GNU RISC-V Cross C Linker General Libraries 	Include paths (-I) "\$(workspace_loc;/\$(ProjName)/drivers/CoreGPIO)" "\$(workspace_loc;/\$(ProjName)/drivers/CoreUARTapb)" "\$(workspace_loc;/\$(ProjName)/hal)" "\$(workspace_loc;/\$(ProjName)/riscv_hal)" Include files (-include)	
Restore Defaults Apply	 Biscellaneous Solut RISC-V Cross Create Flash Image General Solut RISC-V Cross Print Size General 	Restore Defaul	ts Apply

- 11. Select the **GNU RISC-V Cross C Linker > General** to map the linker script.
- 12. Click **Add** as shown in Figure 69, page 44, and in the Add file path dialog, click **Workspace** as shown in Figure 70, page 44.
- 13. In the File Selection dialog box, expand MiV_uart_blinky and select the linker script as shown in the following figure.

Figure 74 • Selecting the Linker Script



14. The linker script is mapped as shown in Figure 75, page 47.



Figure 75 • Linker Script Default Mapping

Settings		↓ ↓ ★ ★
Configuration: Debug [Active]		V Manage Configurations
🛞 Tool Settings 🛞 Toolchains 🎤 Build Step	s 🍨 Build Artifact 🗟 Binary Parsers 🧕 Error Parser	'S
 Target Processor Optimization Warnings Debugging S GNU RISC-V Cross Assembler Preprocessor Includes Warnings Miscellaneous S GNU RISC-V Cross C Compiler Miscellaneous Sinscellaneous S GNU RISC-V Cross C Linker Conso C Linker Conso C Linker Conso C Linker GNU RISC-V Cross Create Flash Image General S GNU RISC-V Cross Print Size General 	Script files (-T) Script files (-T) Do not use standard start files (-nostartfiles) Do not use default libraries (-nostartfiles) No startup or default libs (-nostdlib) No startup or default libs (-nostdlib)	● ● 당 상(i-riscv-ram.ld)"
	Print removed sections (-Xlinkerprint-gc-sectio	ns)
		Restore Defaults Apply
	[Apply and Close Cancel

- 15. Select the Do not use standard start files (-nostartfiles) option as shown in Figure 75, page 47.
- 16. Select the GNU RISC-V Cross Create Flash Image > General and set Other Flags to "--change-section-lma *-0x80000000" as shown in Figure 76, page 48. This excludes the extended linear record in the first line of the hex file.



Figure 76 • RISC-V Flash Image Settings

onfiguration: Debug [Active]		Ŷ	Manage Configuration
🖲 Tool Settings 🛞 Toolchains 📕 Devices	; 🎤 Build Steps 😨 B	uild Artifact 📓 Binary Parsers 😣 Error Parser	5
 Barget Processor Optimization Warnings Debugging 	Output file format (-O)	Intel HEX	,
Soft RISC-V Cross Assembler Preprocessor Miscellaneous Warnings Warnings Warnings Warnings Preprocessor Includes Preprocessor Miscellaneous Warnings Wiscellaneous Soft RISC-V Cross C Linker Wiscellaneous Soft RISC-V Cross Create Flash Image Softereal Soft RISC-V Cross Create Listing Soft RISC-V Cross Create Listing Soft RISC-V Cross Print Size Soft RISC-V Cross Print P	Other sections (-j)		2 2 3 3 5 1 4
	Other flags	change-section-Ima *-0x80000000	
		Restor	e Defaults Apply

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

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

3.6 Mapping Memory and Peripheral Addresses

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

To map the memory address:

- 1. Open the linker script (microsemi-riscv-ram.ld) available in the riscv_hal folder.
- 2. Ensure that the ram ORIGIN address is mapped to 0x80000000.
- 3. Ensure that the LENGTH of the ram is 64 KB.
- 4. Ensure that the RAM_START_ADDRESS is mapped to 0x80000000.
- 5. Ensure that the RAM SIZE is 64 KB.
- 6. Ensure that the STACK SIZE is 2 KB.
- 7. Ensure that the HEAP_SIZE is 2 KB.
- 8. Save the file.



The following figure shows the linker script.

Figure 77 • Linker Script

```
2 * (c) Copyright 2016-2018 Microsemi SoC Products Group. All rights reserved.
3 *
4 * file name : microsemi-riscv-ram.ld
^5 * Mi-V soft processor linker script for creating a SoftConsole downloadable
6 * debug image executing in SRAM.
7 *
8 * This linker script assumes that the SRAM is connected at on the Mi-V soft
9 * processor memory space. The start address and size of the memory space must
10 * be correct as per the Libero design.
11 *
12 * SVN $Revision: 9661 $
13 * SVN $Date: 2018-01-15 16:13:33 +0530 (Mon, 15 Jan 2018) $
14 */
15
16 OUTPUT ARCH( "riscv" )
17 ENTRY(_start)
18
19
20 MEMORY
21 {
22
      ram (rwx) : ORIGIN = 0x80000000, LENGTH = 64k
23 }
24
25 RAM_START_ADDRESS
                    = 0 \times 80000000;
                                        /* Must be the same value MEMORY region ram ORIGIN above. */
                     = 64k;
26 RAM SIZE
                                       /* Must be the same value MEMORY region ram LENGTH above. */
27 STACK_SIZE
                     = 2k;
                                        /* needs to be calculated for your application */
28 HEAP_SIZE
                    = 2k;
                                        /* needs to be calculated for your application */
29
```

In the Libero design flow, the UART, GPIO, and SPI peripherals addresses are mapped to 0x60000000, 0x60001000, and 0x60002000 respectively. This information needs to be provided in the sample hw platform.h file provided in the riscv_hal folder.

To map the peripheral address:

- 1. Open the hardware platform header file (sample_hw_platform.h) available in the riscv_hal folder.
- 2. Locate the COREUARTAPB0_BASE_ADDR macro, and define it as 0x6000000UL.
- 3. Locate the COREGPIO_OUT_BASE_ADDR macro, and define it as 0x60001000UL.
- 4. Locate the FLASH_CORE_SPI_BASE macro, and define it as 0x60002000UL.
- 5. Save the file.

The following figure shows the sample hw platform.h after these updates.

Figure 78 • Updated sample_hw_platform.h File

200	/*****	*****	******
39 40	<pre>* Non-memory Peripheral base addres * Format of define is:</pre>	ses	
41	* <corename> <instance> BASE ADDR</instance></corename>		
42	*/		
43	#define COREUARTAPB0 BASE ADDR	0x6000000UL	
44	<pre>#define COREGPIO_IN_BASE_ADDR</pre>	0x70002000UL	
45	#define CORETIMER0_BASE_ADDR	0x70003000UL	
46	#define CORETIMER1 BASE ADDR	0x70004000UL	
47	<pre>#define COREGPIO_OUT_BASE_ADDR</pre>	0x60001000UL	
48	#define FLASH CORE SPI BASE	0x60002000UL	
49	<pre>#define CORE16550_BASE_ADDR</pre>	0x70007000UL	

The memory and peripheral addresses are successfully mapped.



3.7 Setting the UART Baud Rate

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

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

To define the system clock frequency:

- 1. Look for #define SYS_CLK_FREQ statement in the sample_hw_platform.h file.
- 2. Define it as:

#define SYS_CLK_FREQ 111111000UL

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

The following figure shows the system clock frequency definition.

Figure 79 • System Clock Frequency Definition



3.8 Building the Mi-V Project

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

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

Figure 80 • Hex File



The HEX file can be used for Design and Memory Initialization. For more information, see Configure Design Initialization Data and Memories, page 27.



3.9 Debugging the User Application Using SoftConsole

Before debugging, the board and the serial terminal must be set up. For more information about the board and serial terminal setup, see Board Setup, page 31 and Serial Terminal Emulation Program (PuTTY) Setup, page 32.

To debug the application:

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

Figure 81 • Debug Icon



- 2. In the **Create, manage and run configurations** window, double-click **GDB OpenOCD Debugging** to generate the debug configuration for the **MiV_uart_blinky** project.
- 3. Select the generated **MiV_uart_blinky Debug** configuration, and click **Search Project**, as shown in the following figure.

Figure 82 • Create, manage, and run configurations Window – Main Tab

C Debug Configurations				>
reate, manage, and run conf	igurations			Ś
1 🗎 🗶 📄 🏇 🕶	Name: MiV uart blinky Debug			
ype filter text	Main 🏂 Debugger 🕨 Startup 💱 Source	e 🗖 Common		
GDB OpenOCD Debugging	Project:			
MiV_uart_blinky Debug	MiV uart blinky			Browse
	C/C++ Application:			
		Variables	Search Project	Browse
	Build (if required) before launching			
	Build Configuration: Select Automatically	/		~
	O Enable auto build	O Disable auto buil	d	
	Use workspace settings	Configure Workspace	e Settings	
er matched 2 of 9 items			Re <u>v</u> ert	Apply
Ð			<u>D</u> ebug	Close



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

Figure 83 • MiV_uart_blinky.elf Selection

sc Program Selection				×
Choose a program to	run:			
Binaries:				
MIV_uart_blinky.elf				
Qualifier:				
券 nonele - /MiV_uai	t_blinky/Debug/N	fiV_uart_b	olinky.elf	
?	ОК		Cancel	

- 5. Go to the **Debugger** tab, and replace the Config Options, Executable, and Commands as follows:
 - Config Options: --file board/microsemi-riscv.cfg
 - **Executable**: \${cross_prefix}gdb\${cross_suffix}
 - Commands: set mem inaccessible-by-default off, set \$target_riscv = 1, set remotetimeout 7, and set arch riscv:rv32



type filter text	📄 Main 🕸 Debugg	ger 🝺 Startup 🦆 Source 🔲 Common 🔒 SVD Path			
🗸 💽 GDB OpenOCD Debugging	OpenOCD Setup				
C MiV_uart_blinky Debug	Start OpenOCD	locally			
Robot	Executable path:	\${openocd_path}/\${openocd_executable}		Browse	Variables
🖲 Robot Remote	Actual executable:	C:\Microsemi\SoftConsole_v6.1\eclipse\//openocd/bir	n/openocd	.exe	
		(to change it use the <u>global</u> or <u>workspace</u> preferences page	ges or the p	project prope	rties page)
	GDB port:	3333			
	Telnet port:	4444			
	Tcl port:	6666			
	Config options:	file board/microsemi-riscy.cfg			1
		······································			
	Allocate consol	e for OpenOCD Allocate console	for the tel	net connecti	on
	GDB Client Setup				
	Start GDB sessio	n			
	Executable name:	\${cross prefix}adb\${cross suffix}		Browse	Variables.
	Actual executable	rice of unknown of adh			
	Other entires	Insever-unknown-en-gub			
	Commands				
	Commands.	set mem inaccessible-by-default off set arch riscv:rv32			· · · · · · · · · · · · · · · · · · ·
		set Starget_riscv = 1			
		set remotetimeout /			
	Remote Target				
	Host name or IP ac	ldress: localhost			
	Port number:	3333			
	Force thread list u	pdate on suspend			
					Restore defa
lter matched 5 of 11 items			R	evert	Apply
2					
?)				Debug	Close

Figure 84 • Create, manage, and run configurations Window – Debugger Tab



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

Figure 85 • Debug Settings- Startup Tab

Name: MiV_uart_blinky Debug		
📄 Main 🕸 Debugger 🕟 Startup 🦃 Source 🔲 Common 🗛 SVD Path		
Initialization Commands		^
Initial Reset. Type: init		
	~	
Enable ARM semihosting		
Load Symbols and Executable		
Use project binary: MiV_uart_blinky.elf		
O Use file:	Workspace File System	Ε
Symbols offset (hex):		
☑ Load executable		
Output See Description of the second seco		
🔿 Use file:	Workspace File System	
Executable offset (hex):		
Runtime Options Debug in RAM		
Run/Restart Commands		
Pre-run/Restart reset Type: halt (always executed at Restart))	
	*	
	-	-

 Click Apply, and then click Debug, as shown in the preceding figure. The Confirm Perspective Switch dialog opens, as shown in Figure 86, page 54.

Figure 86 • Confirm Perspective Switch Dialog Box

SC Conf	firm Perspective Switch
\bigcirc	This kind of launch is configured to open the Debug perspective when it suspends.
•	This Debug perspective is designed to support application debugging. It incorporates views for displaying the debug stack, variables and breakpoint management.
	Do you want to open this perspective now?
🔲 <u>R</u> en	nember my decision
	Yes <u>N</u> o

8. Click Yes.

The debugger halts the execution at the first instruction in the main.c file, as shown in the following figure.



Figure 87 • First Instruction in the main.c File

	- §l - t≎ ¢ - ⇒	-			Quick Access
₩ Debug 🛛	🍇 it 👻 🗖 🗖	🕬• Variables 🖾 💁 Breakpoints	🛛 🕅 Registers 🛋 Modules 🕋 Peripherals		20 🕫 🖻 📑 😁 🔍
C IMU, uart, blinky Debug (GBB OpenOCD Debugging) C IMU, uart, blinky Defug (GBB OpenOCD Debugging) C IMU, uart, blinky Debugging) <pc blinky="" debugging<="" imu,="" th="" uart,=""><th></th><th>Name ↔ delay_count</th><th>Type volatile int32_t</th><th>Value -2147477820</th><th></th></pc>		Name ↔ delay_count	Type volatile int32_t	Value -2147477820	
에 main.c 않 @ sample_hw_platform.h @ microsemi-riscv-ram.ld ⓒ 0x800015c0		<		e Outline 🛛 🕒 🖡	ર્ષે≮ ● ₩ ୭ ⊽
<pre>SIM of the second second</pre>				sample_hw_platform.h work_reg_access.h work_reg_access.h untreg_access.h work_reg_access.h work_reg_access.	nt16_t) : uint8_t nt16_t) : uint8_t

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

Figure 88 • Resume Application Execution



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

Figure 89 • Hello World in Debug Mode

Heilo world:		

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



Figure 90 • Mi-V Register Values

(x)= Variables 💁 Breakpoints 📶 Registers	🛿 🚼 Peripherals 🛋 Modules	
Name	Value	Description
General Registers		General Purpose and FPU Register Group
1010 0101 zero	0x0	
0101 ra	0x80001614	
1010 sp	0x80003ce0	
1111 gp	0x80001f80	
3131 tp	0x90204080	
1000 tO	0x8000003c	
1010 tl	0x1922400	
1010 t2	0x2581204a	
1000 fp	0x80003d00	
1010 s1	0x2800000	
1010 aO	0x600010a0	
1000 al	0xa	
10101 a2	0x40	
1010 a3	0x6000000	
1010 a4	0xa	
1010 a5	0x64e86	
1010 a6	0x11100801	
1010 a7	0x8395000	
1010 s2	0x100c0800	
1111 s3	0x25008304	
1010 s4	0x80040	
1010 55	0x3092a114	

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

Figure 91 • Variable Values

(X)= Variables 🔀 💁 Breakpoints 👭 Registers 📴 Peripherals 🛋 Modules			
Name	Туре	Value	
(x)= gpio_pattern	uint32_t	10	
(x)= delay_count	volatile int32_t	413319	

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



3.10 Debugging the User Application from DDR3 Memory

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

Figure 92 • RAM Start Address Parameters



The SoftConsole reference project specifies the LSRAM start address, which is 0x80000000 (highlighted in Figure 92, page 57). To perform application debugging from DDR3 memory, modify this value to the DDR3 memory starting address, 0x80010000. After modifying the value, clean and build the project.

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

Figure 93 • Debugging from DDR3





4 Appendix

4.1 References

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

- For more information about Mi-V, see MI-V RV32IMA_L1_AXI_HB.pdf from the Libero SoC Catalog.
- For more information about CoreJTAGDebug, see CoreJTAGDebug_HB.pdf.
- For more information about CoreAHBtoAPB3, see CoreAHBtoAPB3_HB.pdf.
- For more information about CoreAXITOAHBL, see CoreAXItoAHBL_HB.pdf.
- For more information about CoreGPIO, see CoreGPIO_HB.pdf.
- For more information about CoreUARTapb, see CoreUARTapb_HB.pdf.
- For more information about CoreAHBLite, see CoreAHBLite_HB.pdf.
- For more information about CoreAPB3, see CoreAPB3_HB.pdf.
- For more information about PolarFire Initialization Monitor, see UG0725: PolarFire FPGA Device Power-Up and Resets User Guide.
- For more information about PolarFire Clock Conditioning Circuitry (CCC), see UG0684: PolarFire FPGA Clocking Resources User Guide.
- For more information about PolarFire SRAM, see UG0680: PolarFire FPGA Fabric User Guide.
- For more information about Libero, ModelSim, and Synplify, see the *Libero SoC PolarFire webpage*.
- For more information about SoftConsole, see the SoftConsole webpage.
- For more information about loading a Job file using FlashPro Express, see the User Guide from FlashPro Express - > Help -> User Guide.