DG0824
Demo Guide
PolarFire FPGA System Services Using Splash Kit

Microsemi
Power Matters.™

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

© 2018 Microsemi Corporation. All rights reserved. Microsemi and the Microsemi logo are 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 Corporation (Nasdaq: MSCC) offers a comprehensive portfolio of semiconductor and system solutions for aerospace & defense, communications, data center and industrial markets. Products include high-performance and radiation-hardened analog mixed-signal integrated circuits, FPGAs, SoCs and ASICs; power management products; timing and synchronization devices and precise time solutions, setting the world's standard for time; voice processing devices; RF solutions; discrete components; enterprise storage and communication solutions, security technologies and scalable anti-tamper products; Ethernet solutions; Power-over-Ethernet ICs and midspans; as well as custom design capabilities and services. Microsemi is headquartered in Aliso Viejo, California, and has approximately 4,800 employees globally. Learn more at www.microsemi.com.

50200824.2.0 7/18

Contents

1 Revision History . . . 1
1.1 Revision 2.0 . . . 1
1.2 Revision 1.0 . . . 1
2 PolarFire FPGA System Services . . . 2
2.1 CoreSysServices_PF IP Overview . . . 2
2.2 Design Requirements . . . 4
2.3 Prerequisites . . . 5
2.4 Demo Design . . . 5
2.4.1 Design Implementation . . . 7
2.4.2 IP Configuration . . . 8
2.5 Clocking Structure . . . 19
3 Libero Design Flow . . . 20
3.1 Synthesize . . . 21
3.2 Place and Route . . . 21
3.2.1 Resource Utilization . . . 21
3.3 Verify Timing . . . 21
3.4 Generate FPGA Array Data . . . 21
3.5 Configure Design Initialization Data and Memories . . . 22
3.6 Configure Programming Options . . . 25
3.7 Generate Bitstream . . . 26
3.8 Run PROGRAM Action . . . 26
4 Programming the Device Using FlashPro . . . 27
5 Setting up the Serial Terminal Program - PuTTY . . . 28
6 Running the Demo . . . 30
7 Appendix: Device Certificate Information . . . 37
8 Appendix: Query Security . . . 39
9 Appendix: Debug Information . . . 40
10 Appendix: Digest Information . . . 41
11 Appendix: References . . . 42

Figures

Figure 1 Core System Services IP Interfacing with Fabric User Logic . . . 3
Figure 2 Firmware catalog . . . 4
Figure 3 System Services Design Block Diagram . . . 6
Figure 4 Top Level Libero Design . . . 7
Figure 5 PF_INIT_MONITOR Configuration . . . 8
Figure 6 PF_CCC_0 Input Clock Configuration . . . 9
Figure 7 PF_CCC_0 Output Clock Configuration . . . 9
Figure 8 Mi-V Configuration . . . 10
Figure 9 CoreUARTapb Configuration . . . 11
Figure 10 CoreJTAGDebug Configuration . . . 11
Figure 11 PF_SRAM_AHBL_AXI Configuration . . . 12
Figure 12 CoreGPIO_0 Configuration . . . 13
Figure 13 CoreSysServices_PF Configuration . . . 14
Figure 14 Memory Map . . . 15
Figure 15 CoreAHBLite_0 Configuration . . . 16
Figure 16 CoreAHBLite_1 Configuration . . . 17
Figure 17 COREAHBTOAPB3 Configuration . . . 17
Figure 18 CoreAPB3_0 Configuration . . . 18
Figure 19 Clocking Structure . . . 19
Figure 20 Libero Design Flow Options . . . 20
Figure 21 Design and Memory Initialization . . . 22
Figure 22 Fabric RAMs Tab . . . 23
Figure 23 Edit Fabric RAM Initialization Client . . . 23
Figure 24 Apply Fabric RAM Content . . . 24
Figure 25 Add PlainText NonAuthenticated Option . . . 24
Figure 26 Edit PlainText NonAuthenticated Client . . . 25
Figure 27 Configure Programming Options . . . 25
Figure 28 Board Setup . . . 26
Figure 29 Run Passed . . . 27
Figure 30 Finding the COM Port . . . 28
Figure 31 Select Serial as the Connection Type . . . 28
Figure 32 PuTTY Configuration . . . 29
Figure 33 System Services Options . . . 30
Figure 34 Device Serial Number . . . 30
Figure 35 Device User-code . . . 30
Figure 36 Device Design Information . . . 31
Figure 37 Device Certificate . . . 32
Figure 38 Digest . . . 33
Figure 39 Security Locks Information . . . 33
Figure 40 Debug Information . . . 33
Figure 41 Digital Signature . . . 34
Figure 42 Secure NVM Services . . . 35
Figure 43 PUF Emulation Service . . . 35
Figure 44 Generated Nonce . . . 35
Figure 45 Flash Freeze . . . 36
Figure 46 I/O State in Flash*Freeze . . . 36
Figure 47 Copy Device Certificate . . . 37
Figure 48 Certificate Decoding Using Java Script . . . 38
Figure 49 Decoded Certificate . . . 38

Tables

Table 1 System Services in the Demo . . . 2
Table 2 System Services Descriptor . . . 3
Table 3 Design Requirements . . . 4
Table 4 I/O Signals . . . 7
Table 5 Resource Utilization . . . 21
Table 6 Jumper Settings for PolarFire Device Programming . . . 26
Table 7 Device Certificate Fields (1024 bytes) . . . 37
Table 8 Security Locks Fields . . . 39
Table 9 Debug Info Fields . . . 40
Table 10 Digest Information Bit Fields . . . 41

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 2.0

Updated the document for Libero SoC PolarFire v2.2.

1.2 Revision 1.0

The first publication of this document.

2 PolarFire FPGA System Services

System services are the system controller actions initiated from the FPGA design using the CoreSysServices_PF IP core. The system controller hard block in PolarFire® FPGAs provides various system services. The CoreSysServices_PF IP core issues service requests to the system controller and fetches the relevant data.

This document describes how to run the system services listed in the following table using the demo design.

Table 1 • **System Services in the Demo**

Service Category	Services
Device and Data Services	Read Device Serial Number Read Device User-code Read Device Design-info
Design and Data Security Services	Read Device Certificate Read Digest Query security Read Debug Information Digital signature Secure NVM services PUF Emulation Nonce service
Fabric Services	Flash*Freeze service

Note: In the demo, the Flash*Freeze service is demonstrated with timed entry and timed exit.

The demo design includes the Mi-V soft processor, which initiates the system service requests and enables the CoreSysService_PF IP core to access the system controller. For more information about the system services design implementation, and the necessary blocks and IP cores instantiated in Libero SoC PolarFire, see Demo Design, page 5.

The demo design can be programmed using any of the following options:

- **Using the pre-generated `.stp` file:** To program the device using the `.stp` file provided along with the demo design, see Programming the Device Using FlashPro, page 27.
- **Using Libero SoC PolarFire:** To program the device using Libero SoC PolarFire, see Libero Design Flow, page 20.

The demo design can be used as a reference to build a fabric design with the system services feature.

2.1 CoreSysServices_PF IP Overview

The system controller actions are initiated by the fabric logic through the system service interface (SSI) of the system controller. The fabric logic requires the CoreSysServices_PF IP for initiating the system services. A service request interrupt to the system controller is triggered when the fabric user logic writes a 16-bit system service descriptor to the SSI. The lower seven bits of the descriptor specify the service to be performed. The upper nine bits specify the address offset (0–511) in the 2 KB mailbox RAM. The mailbox address specifies the service-specific data structure used for any additional inputs or outputs for the service. The fabric logic must write additional parameters to the mailbox before requesting a system service.

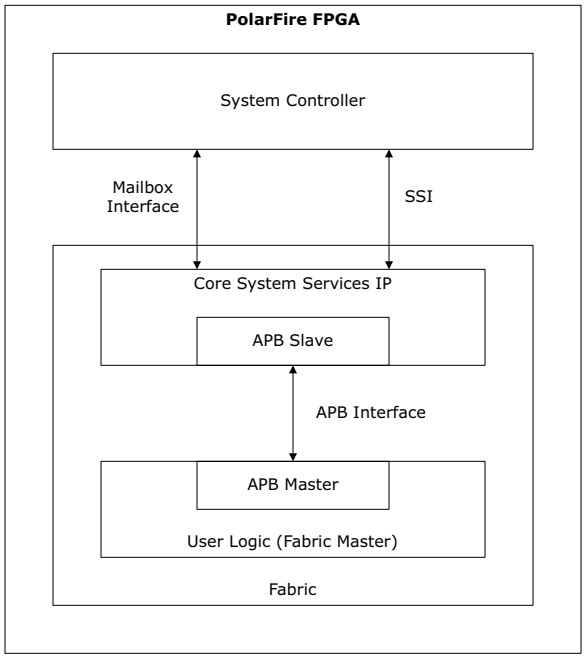
The following table lists the system service descriptor bits.

Table 2 • **System Services Descriptor**

Descriptor Bit	Value
15:7	MBOXADDR
6:0	SERVICEID

SSI consists of an asynchronous command-response interface that transfers a system service command from the fabric master to the system controller and the status from the system controller to the fabric master. The following figure shows how the CoreSysServices_PF IP interfaces with the fabric logic.

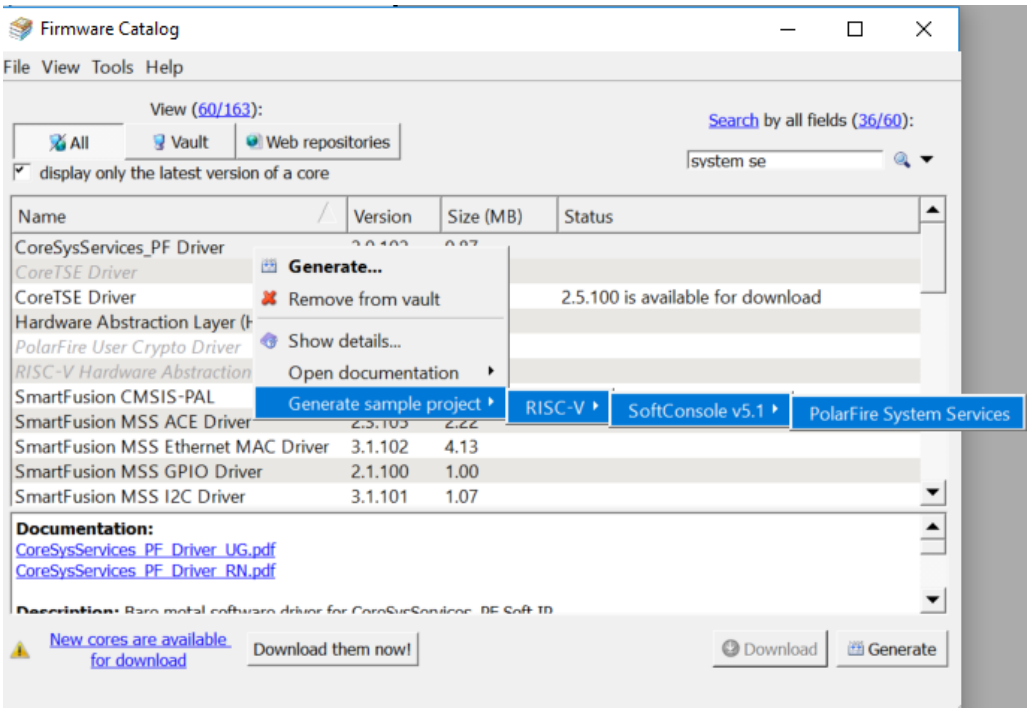
Figure 1 • **Core System Services IP Interfacing with Fabric User Logic**



The system services driver and the sample SoftConsole project are generated from Firmware Catalog as shown Figure 2, page 4.

In this demo, the sample SoftConsole project is migrated to SoftConsole v5.2 and the application file `main.c` is modified to provide the user options.

Figure 2 • **Firmware catalog**



2.2 Design Requirements

The following table lists the resources required to run the demo.

Table 3 • **Design Requirements**

Requirement	Version
Operating System	Windows 7, 8.1, or 10
Hardware	
PolarFire Splash Kit (MPF300TS-1FCG484EES) • PolarFire splash board • 12 V, 5 A AC power adapter and cord • USB 2.0 A to mini-B cable for universal asynchronous receiver-transmitter (UART) and programming	Rev 2 or later
Host PC	
Software	
FlashPro	12.200.30.10
Libero SoC PolarFire Design Suite	2.2
Serial Terminal Emulation Program	PuTTY or HyperTerminal *www.putty.org*
IP	
PF_INIT_MONITOR	2.0.103

Table 3 • **Design Requirements**

PF_CCC	1.0.113
CoreJTAGDEBUG	2.0.100
CORESET_PF	2.1.100
Mi-V soft processor (MIV_RV32IMA_L1_AHB)	2.0.100
COREAHBLite	5.3.101
COREAHBTOAPB3	3.1.100
CoreAPB3	4.1.100
CoreUARTapb	5.6.102
CoreGPIO	3.2.102
CoreSysServices	2.3.116
PF_SRAM_AHBL_AXI	1.1.125
Core_SPI	5.1.104

Note: Any serial terminal emulation program can be used. PuTTY is used in this demo.

2.3 Prerequisites

Before you start:

Download the demo design files from the following location:

http://soc.microsemi.com/download/rsc/?f=mpf_dg0824_liberosocpolarfirev2p2_df

1. Download and install Libero SoC PolarFire v2.2 on the host PC from the following location. https://www.microsemi.com/products/fpga-soc/design-resources/design-software/libero-soc-polarfire#downloads

   The latest versions of ModelSim and Synplify Pro are included in the Libero SoC PolarFire installation package.

2.4 Demo Design

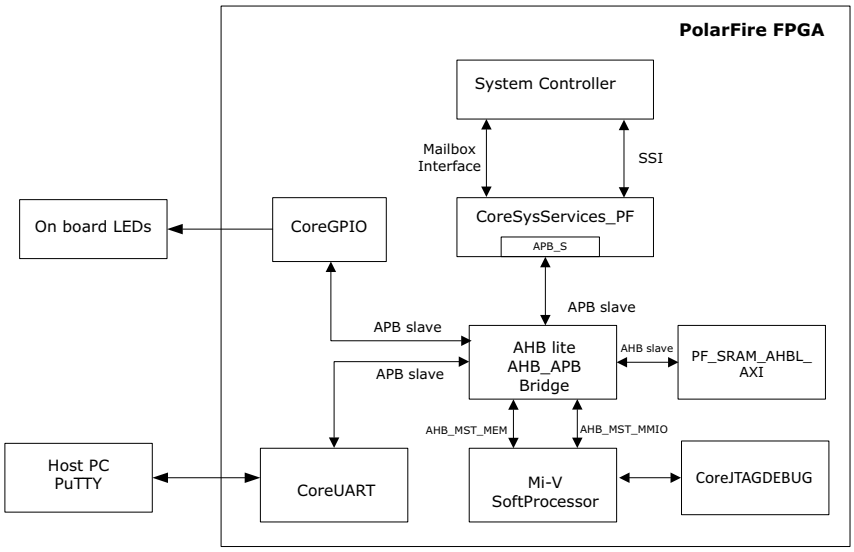
The following steps describe the data flow in the demo design:

In the demo design:

1. The host PC sends the system service requests to CoreUARTapb block through the UART Interface.
2. The Mi-V soft processor initializes the system controller using the CoreSysServices_PF IP and sends the requested system service command to the system controller.
3. The system controller executes the system service command and sends the relevant response to the CoreSysServices_PF IP over the mailbox interface.
4. The Mi-V processor receives the service response and forwards the data to the UART interface.

The following figure shows the block diagram of the system services design.

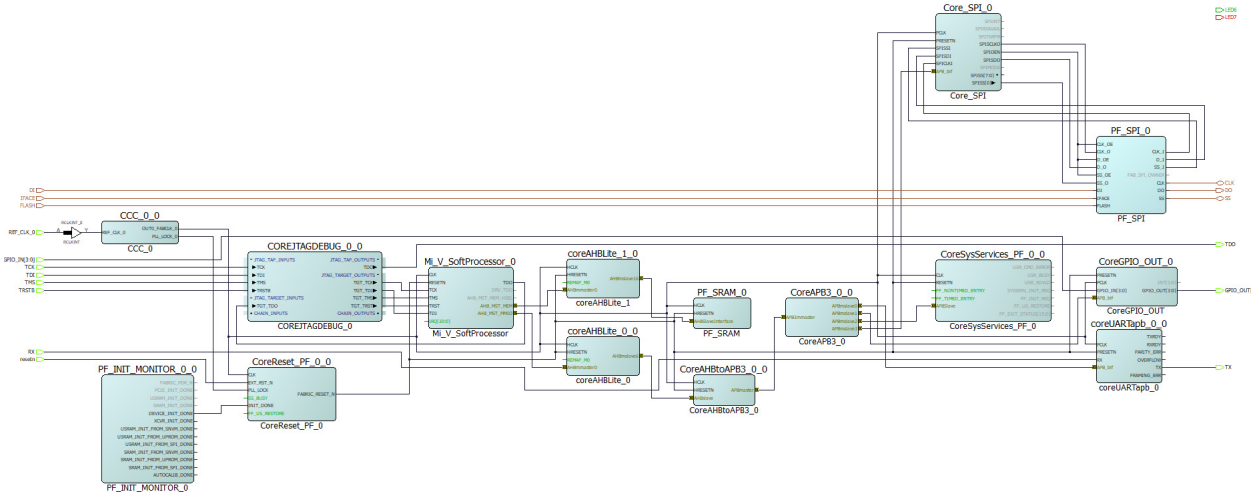
Figure 3 • **System Services Design Block Diagram**



2.4.1 Design Implementation

The following figure shows the top-level Libero design of the PolarFire system services design.

Figure 4 • **Top Level Libero Design**



The following table lists the important I/O signals of the design.

Table 4 • **I/O Signals**

Signal	Description
REF_CLK_0	Input 50 MHz clock from the onboard 50 MHz oscillator
resetn	Onboard reset push-button for the PolarFire device
RX	Input signals received from the serial UART terminal
TX	Output signals transmitted to the serial UART terminal
GPIO_OUT[3:0]	Onboard LED outputs

2.4.2 IP Configuration

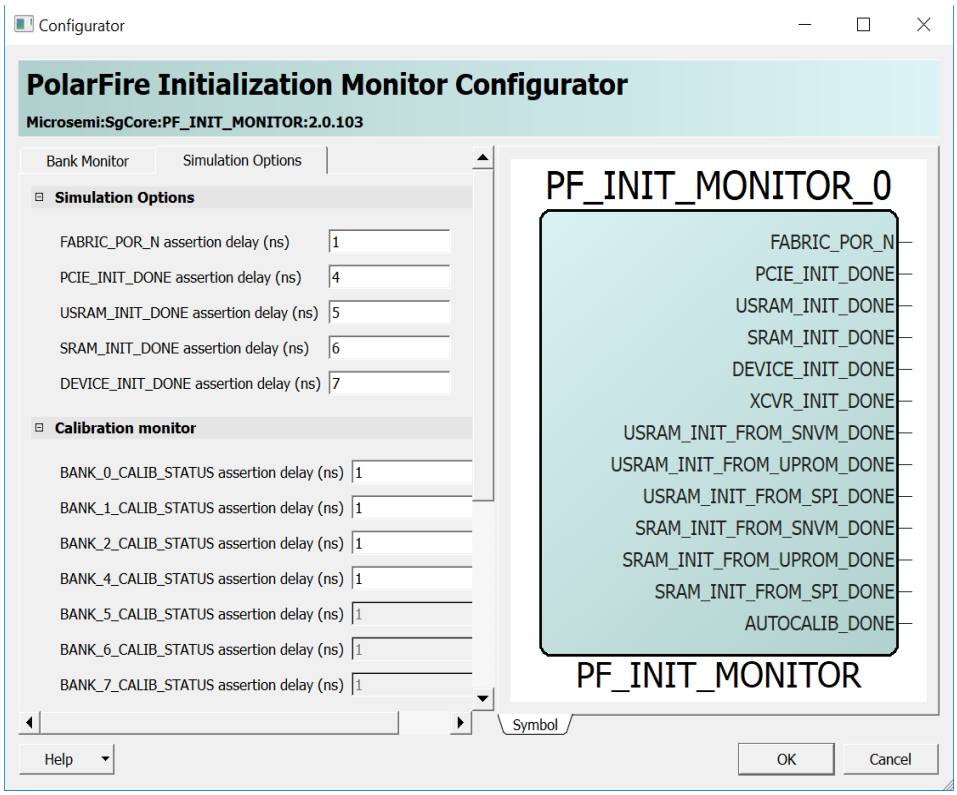
The following sections describe the IP cores used in the design and their configurations.

The other IP cores retain the default configuration.

2.4.2.1 PF_INIT_MONITOR

The PolarFire Initialization Monitor gets the status of device initialization including the LSRAM initialization. The following figure shows PF_INIT_MONITOR configuration.

Figure 5 • **PF_INIT_MONITOR Configuration**



2.4.2.2 PF_CCC_0 Configuration

The PolarFire Clock Conditioning Circuitry (CCC) block takes an input clock of 50 MHz from the onboard oscillator through CLKINT and generates a 100 MHz fabric clock to the Mi-V processor subsystem and other peripherals.

The following figures show the input and output clock configurations.

Figure 6 • **PF_CCC_0 Input Clock Configuration**

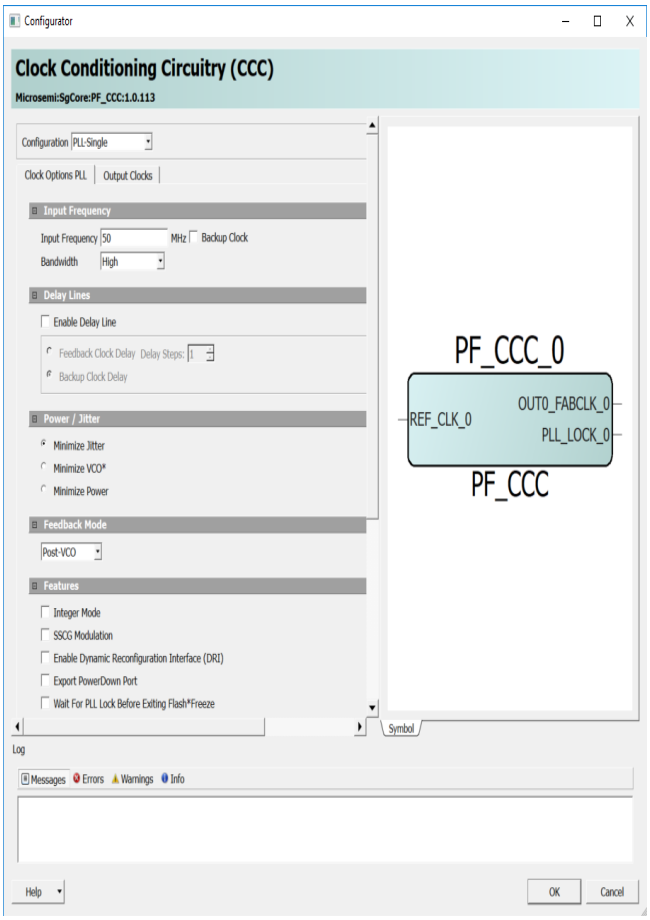
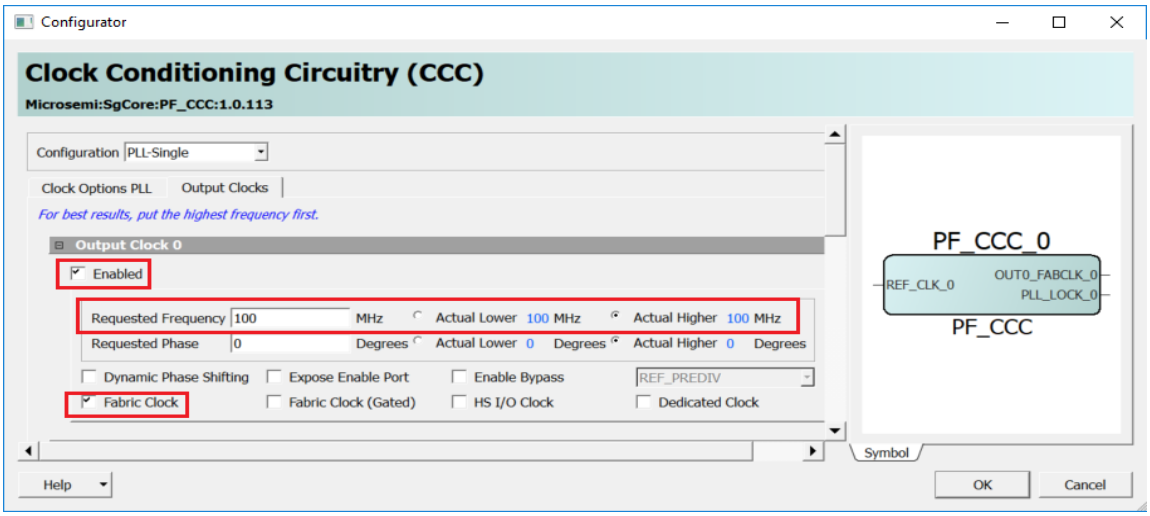


Figure 7 • **PF_CCC_0 Output Clock Configuration**

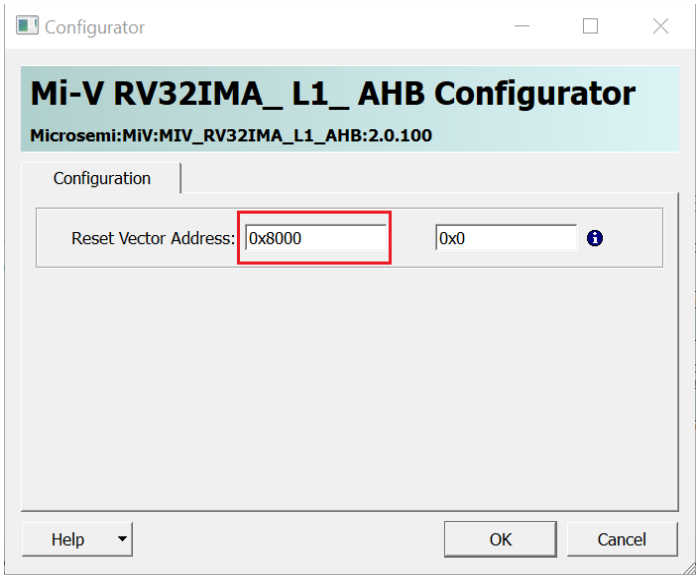


2.4.2.3 Mi-V Soft Processor Configuration

The Reset Vector Address of the Mi-V soft processor is set to 0x8000_0000 from 0x6000_0000. After the device reset, the processor executes the application from LSRAM, which is mapped to 0x80000000, Hence, the Reset Vector Address is set to 0x80000000 as shown in Figure 8, page 10.

In the Mi-V processor memory map, the 0x8000_0000 to 0x8FFF_FFFC range is defined for AHB memory interface and the 0x6000_0000 to 0x7FFF_FFFF range is defined for AHB I/O interface.

Figure 8 • **Mi-V Configuration**



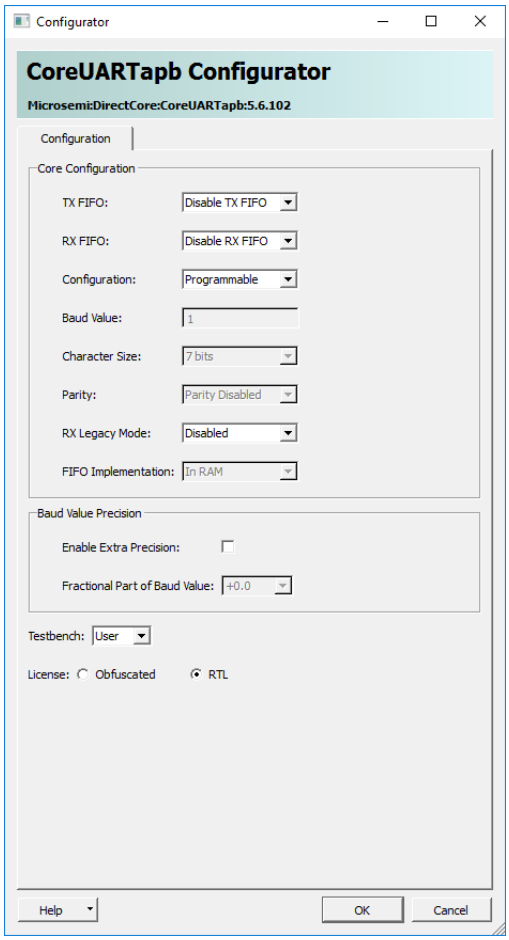
2.4.2.4 CoreUARTapb

The CoreUARTapb IP is connected tothe Mi-V soft processor as an APB slave. It interfaces with the host PC for UART communication. Figure 9, page 11 shows the configuration settings of the CoreUARTapb IP:

- **TX FIFO:** Disabled by default.
  The UART transmit state machine immediately begins to transmit data and continues transmission until the data buffer is empty in normal mode. If **TX FIFO** is enabled, it continues to transmit until **TX FIFO** is empty. In this design, normal mode (without FIFO) is selected.
- **RX FIFO**: Disabled by default.
  The UART receive state machine stores the data in receive data buffer if FIFO is not enabled.
- **Configuration**: Set to **Programmable** by default.

The user application programs the baud rate, character size, and the parity configuration using the UART driver. If the **Fixed** option is selected, the user application can not overwrite these parameters.

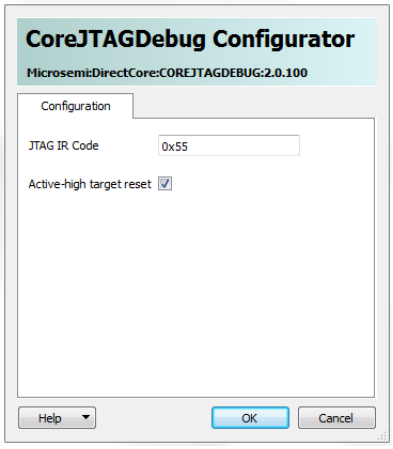
Figure 9 • **CoreUARTapb Configuration**



2.4.2.5 CoreJTAGDebug

The CoreJTAGDebug IP connects the Mi-V soft processor to the JTAG header for debugging. The following figure shows the configuration of the CoreJTAGDebug IP core.

Figure 10 • **CoreJTAGDebug Configuration**

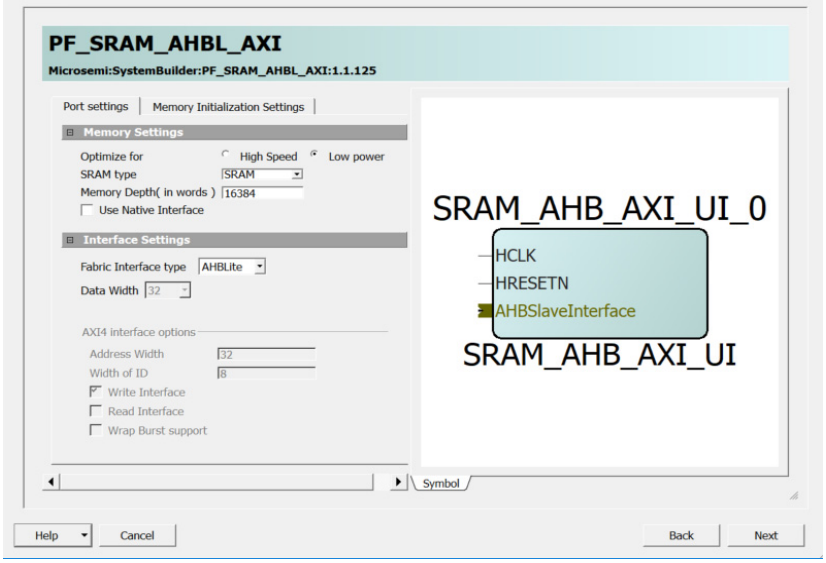


2.4.2.6 PF_SRAM_AHBL_AXI Configuration

The PF_SRAM_AHBL_AXI IP is the main memory of the Mi-V processor, and it gets initialized with the user application from µPROM. It is connected to Mi-V soft processor as an AHB slave. LSRAM is configured for the following settings:

- **Optimize for:** By default, Low power is selected. It optimizes the LSRAM macro for low power. If design demands high speed memory access, High Speed can be selected.
- **Fabric Interface type**: By default, AHBLite is selected. The Mi-V soft processor is AHB based, so the SRAM is interfaced to the processor using AHB bus for code execution.
- **Memory depth:** This field is set to 16384 words to accommodate an application of up to 65 KB into LSRAM. The present application is below 50 KB so this can fit into either sNVM or µPROM. In this demo, µPROM is selected as data storage client. The following figure shows the PF_SRAM_AHBL_AXI (LSRAM_0) IP configuration.

Figure 11 • **PF_SRAM_AHBL_AXI Configuration**



2.4.2.7 CoreGPIO_0 Configuration

The CoreGPIO IP controls the on-board LEDs using GPIOs. It is connected to Mi-V soft processor as an APB slave.

The configuration settings of the COREGPIO_0 IP are as follows:

In the **Global Configurations** pane:

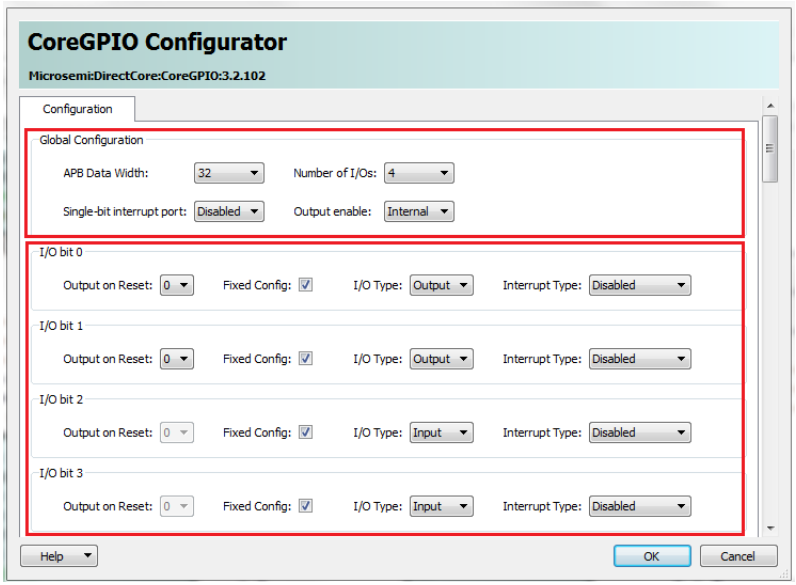
- **APB Data width** is set to 32
  The design uses 32-bit data width for APB read and write data.
- **Number of I/Os** is set to 4
  The design controls 2 onboard LEDs for output and 2 DIP Switches for input.
- **I/O Bit:** The following list shows the sub-options under I/O Bit option.
  - **Output on reset:** Set to 0.
  - **Fixed Config:** Yes
  - **I/O type:** As shown in the following figure, first two I/Os are configured as output and the last two I/Os are configured as input.

Note: The first two I/Os configured as output are used by the design and last two I/Os are not used. The I/Os are interfaced to on-board LEDS to control the LED states.

- **Interrupt Type:** Disabled
  When I/O states change, no interrupt is required for the application as these are used for only LEDs.

The following figure shows the CoreGPIO_0 configuration.

Figure 12 • **CoreGPIO_0 Configuration**



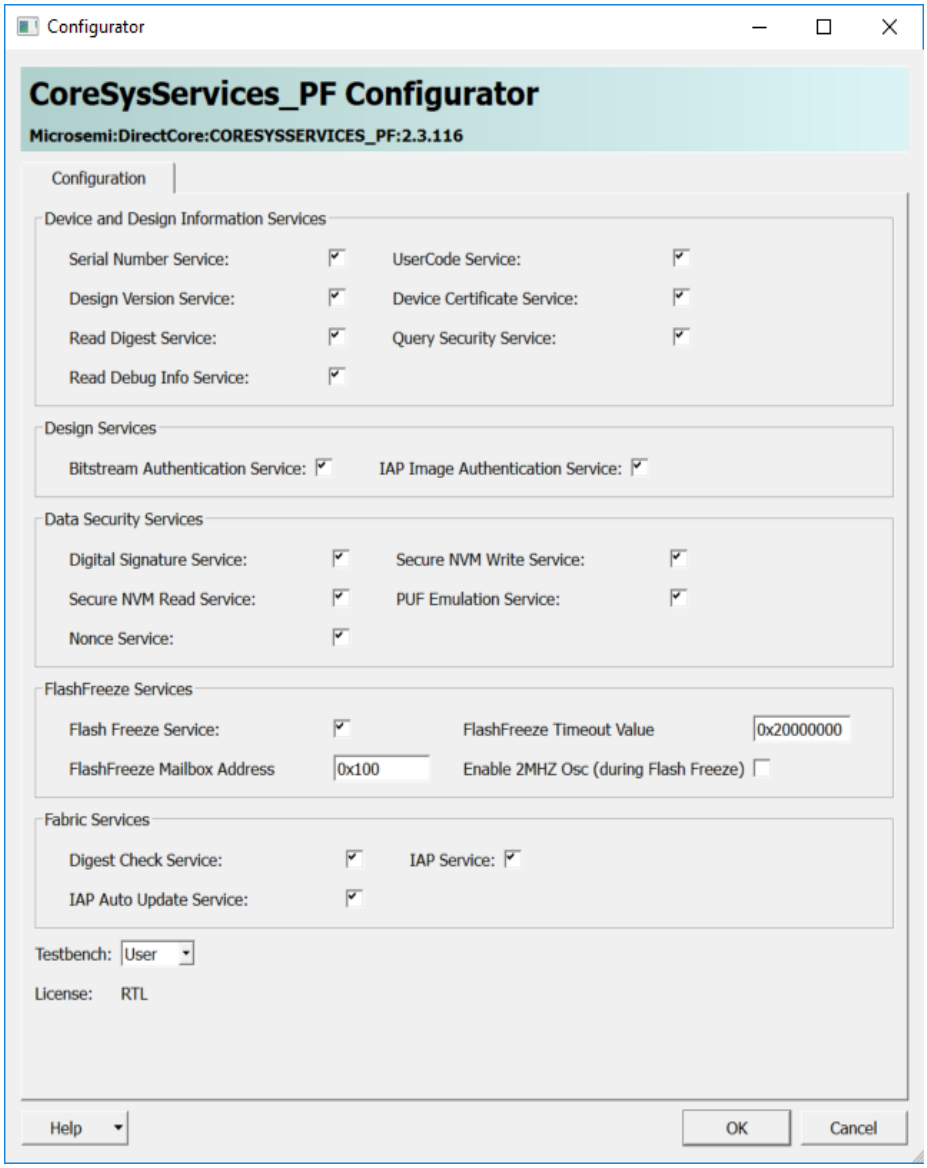
2.4.2.8 CoreSysServices_PF Configuration

CoreSysServices IP provides access to the system controller. It is connected to Mi-V soft processor as an APB slave. The configuration settings of CoreSysServices_PF IP are as follows:

- By default, all the service check boxes are selected. The application can initiate these selected services.
- **FlashFreeze time out:** The default value is 0x20000000 ms. This value can be overwritten by the application before initiating the Flash*Freeze service. When device enters the Flash*Freeze state, the device automatically exits the Flash*Freeze state after 0x20000000 ms.
- **FlashFreeze Mailbox Address:** It is the mailbox address location where time out value is passed as input in the service request. The default value is 0x100. The allowed value is between 0 to 511.

CoreSysServices IP is configured as shown in the following figure.

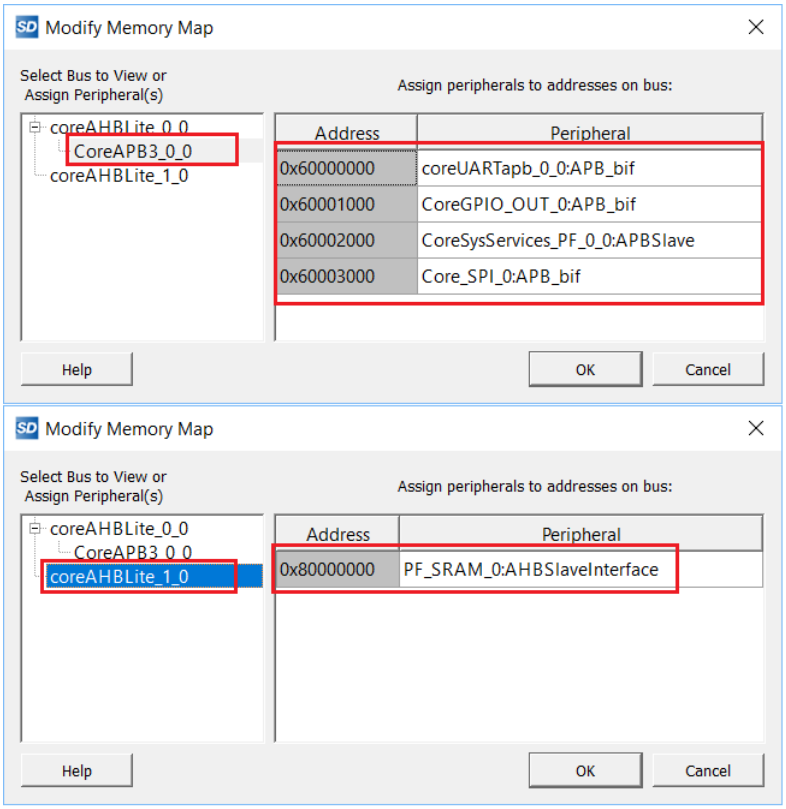
Figure 13 • **CoreSysServices_PF Configuration**



2.4.2.9 Design Memory Map

The Mi-V processor bus interface memory map is shown in the following figure.

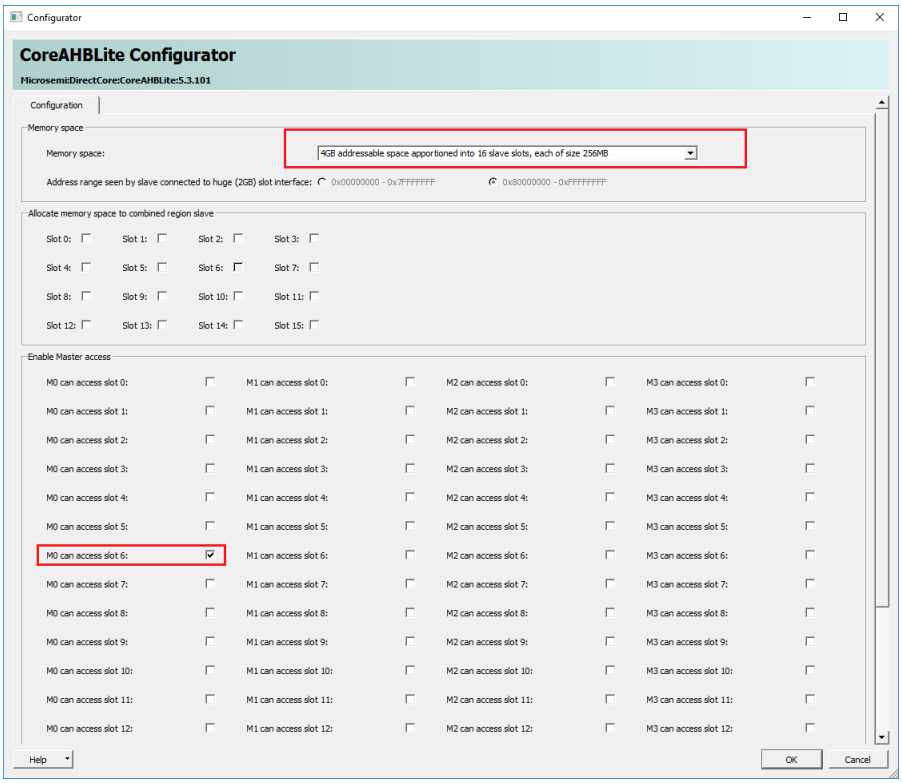
Figure 14 • **Memory Map**



2.4.2.9.1 CoreAHBLite Configuration

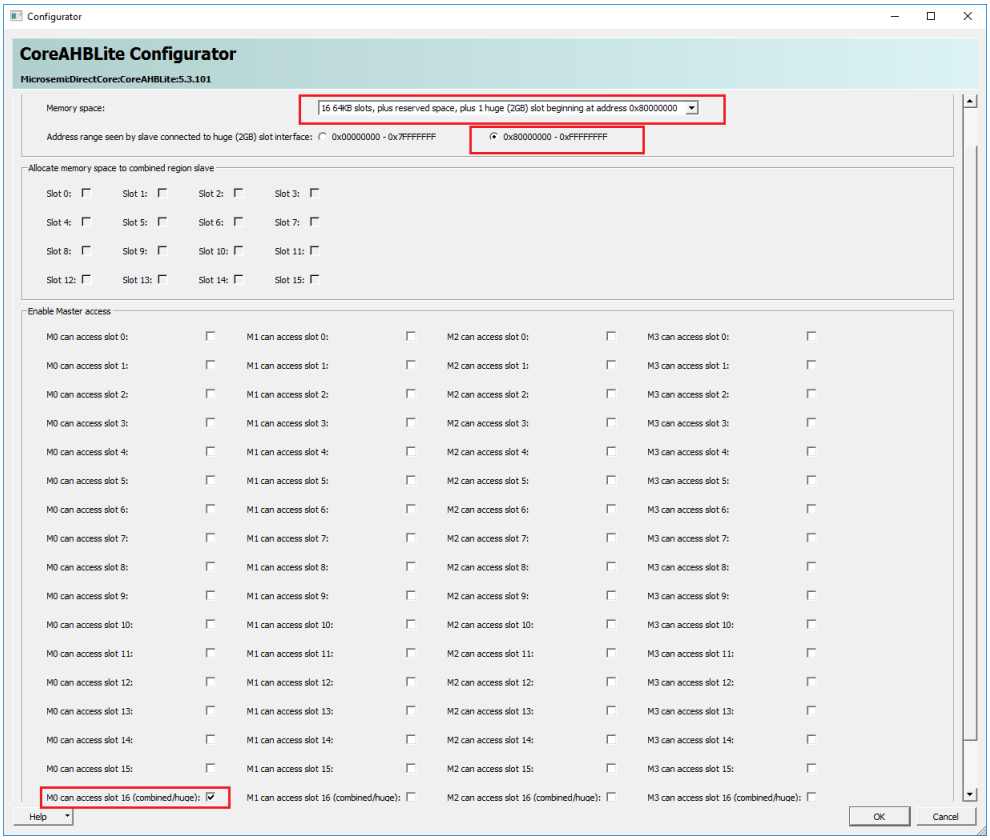
Two instances of CoreAHBLite are used in this design. The following figures show the configurations of CoreAHBLite_0 and CoreAHBLite_1 IP cores.The CoreAHBLite_0 interfaces with the APB peripherals to the Mi-V processor at 0x6000_0000.

Figure 15 • **CoreAHBLite_0 Configuration**



The CoreAHBLite_1 interfaces PF_SRAM with Mi-V soft processor for accessing the LSRAM at memory address 0x8000_0000. This configuration is required as the Mi-V processor executes the code from 0x8000_0000.

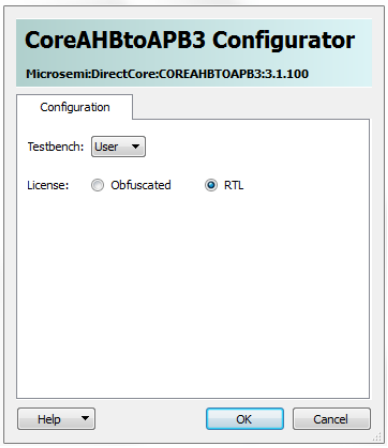
Figure 16 • **CoreAHBLite_1 Configuration**



2.4.2.9.2 COREAHBTOAPB3

The CoreAHBtoAPB3 works as a bridge in between the AHB and the APB domains. CoreAHBtoAPB3 interfaces with CoreAHBLite through its AHB interface and with CoreAPB3 through its APB interface. The following figure shows configuration of COREAHBTOAPB3 IP core.

Figure 17 • **COREAHBTOAPB3 Configuration**



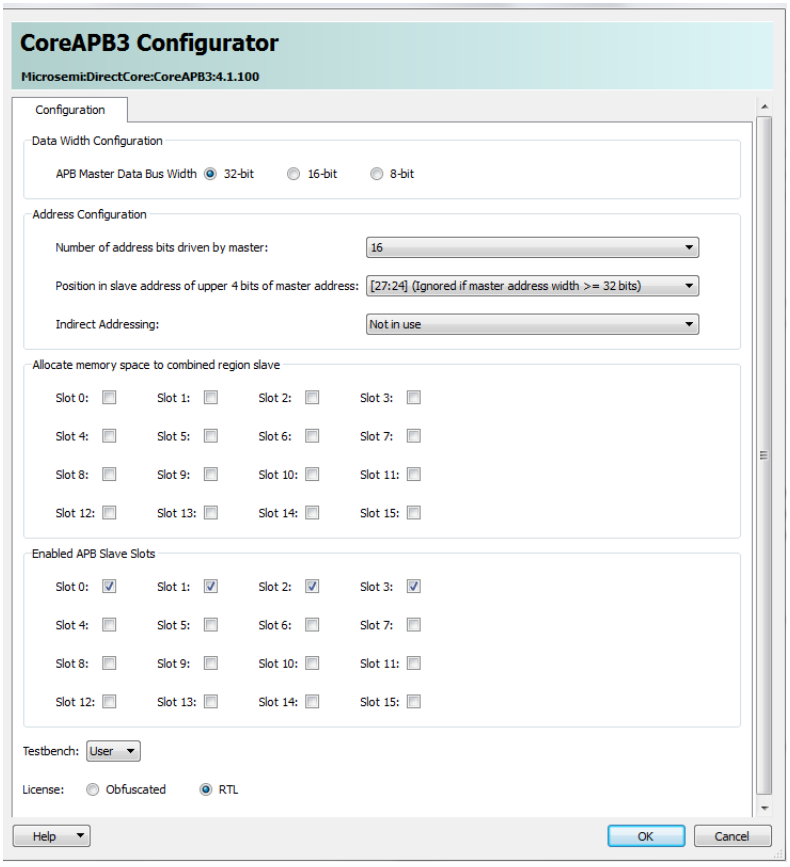
2.4.2.9.3 CoreAPB3 Configuration

The CoreAPB3 IP connects the peripherals, CoreSysServices_PF, CoreSPI, CoreGPIO and CoreUARTapb as slaves.The configuration settings of **COREAPB3** are as follows:

- **APB Master Data bus width:** 32-bit
  The design uses 32-bit data width for APB read and write data.
- **Number of address bits driven by master:** 16
  The Mi-V processor accesses the slaves using the 16-bit. The final addresses for these slaves are translated into 0x6000_0000, 0x6000_1000, 0x6000_2000 and 0x6000_3000.
- **Enabled APB slave slots:** Slot 0 for CoreUARTapb, Slot 1 for CoreGPIO, Slot 2 for CoreSysServices_PF, and Slot 3 for CoreSPI.

The following figure shows the CoreAPB3 configuration.

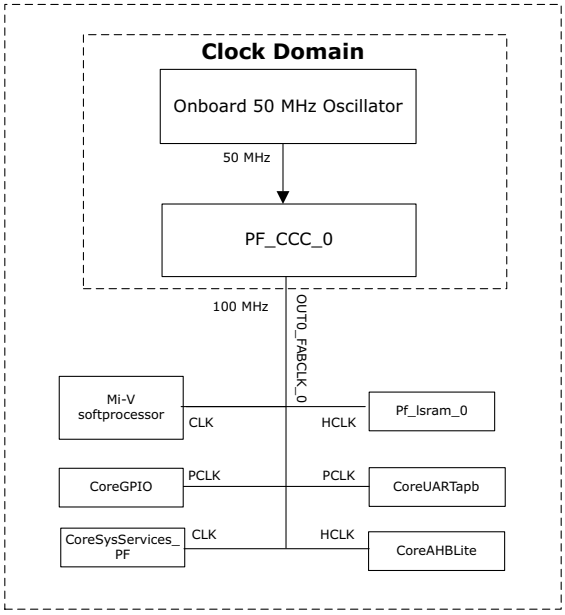
Figure 18 • **CoreAPB3_0 Configuration**



2.5 Clocking Structure

The following figure shows the clocking structure of the demo design. The Mi-V processor supports a clock rate of up to 120 MHz. This design uses 100 MHz system clock.

Figure 19 • **Clocking Structure**



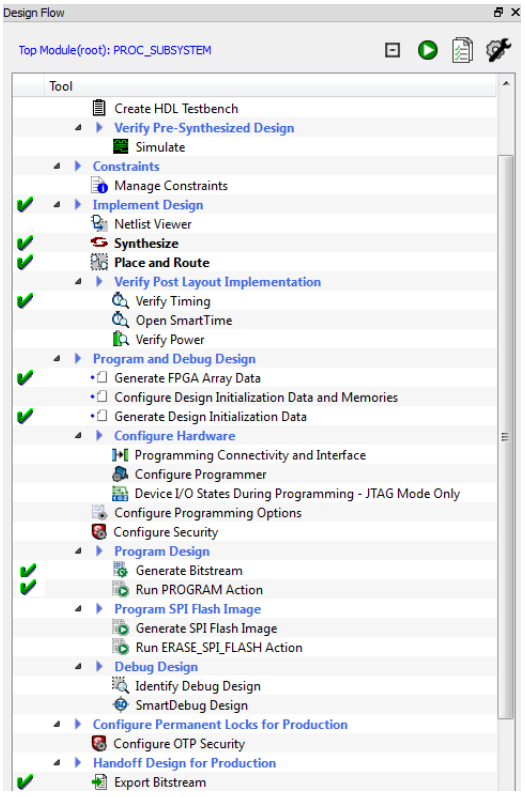
3 Libero Design Flow

The Libero design flow involves running the following processes in the Libero SoC PolarFire:

- Synthesize, page 21
- Place and Route, page 21
- Verify Timing, page 21
- Generate FPGA Array Data, page 21
- Configure Design Initialization Data and Memories, page 22
- Configure Programming Options, page 25
- Generate Bitstream, page 26
- Run PROGRAM Action, page 26

The following figure shows these options in the **Design Flow** tab.

Figure 20 • **Libero Design Flow Options**



3.1 Synthesize

To synthesize the design:

1. Double-click **Synthesize** from the **Design Flow** tab.
   When the synthesis is successful, a green tick mark appears as shown in Figure 20, page 20.
2. Right-click **Synthesize** and select **View Report** to view the synthesis report and log files in the **Reports** tab.

Note: `PROC_SUBSYSTEM.srr` and the `PROC_SUBSYSTEM_compile_netlist.log` files are recommended to be viewed for debugging synthesis and compile errors.

3.2 Place and Route

The Place and Route process requires the I/O, timing, and floor planner constraints. The demo design includes following constraint files in the **Constraint Manager** window:

- The io.pdc and the user.pdc file for the I/O assignments
- The PROC_SUBSYSTEM_derived_constaints.sdc file for timing constraints
- Th pll_placement.pdc file for PLL placement

To Place and Route, double-click **Place and Route** from the **Design Flow** window.

When place and route is successful, a green tick mark appears next to Place and Route.

Note: The file, `PROC_SUBSYSTEM_place_and_route_constraint_coverage.xml` is recommended to be viewed for place and route constraint coverage.

3.2.1 Resource Utilization

The resource utilization report is written to the `PROC_SUBSYSTEM_layout_log.log` file in the `Reports tab -> PROC_SUBSYSTEM reports -> Place and Route`. It lists the resource utilization of the design after place and route. These values may vary slightly for different Libero runs, settings, and seed values.

Table 5 • **Resource Utilization**

Type	Used	Total	Percentage
4LUT	14471	299544	4.83
DFF	7457	299544	2.49
I/O Register	0	242	0.00
Logic Element	15153	299544	5.06

3.3 Verify Timing

To verify timing:

1. Double-click **Verify Timing** from the **Design Flow** tab.
   When the design successfully meets the timing requirements, a green tick mark appears as shown in Figure 20, page 20.
2. Right-click **Verify Timing** and select **View Report,** to view the verify timing report and log files in the **Reports** tab.

3.4 Generate FPGA Array Data

To generate the FPGA array data:

1. Double-click **Generate FPGA Array Data** from the **Design Flow** window.
2. A green tick mark is displayed after the successful generation of the FPGA array data as shown in Figure 20, page 20.

3.5 Configure Design Initialization Data and Memories

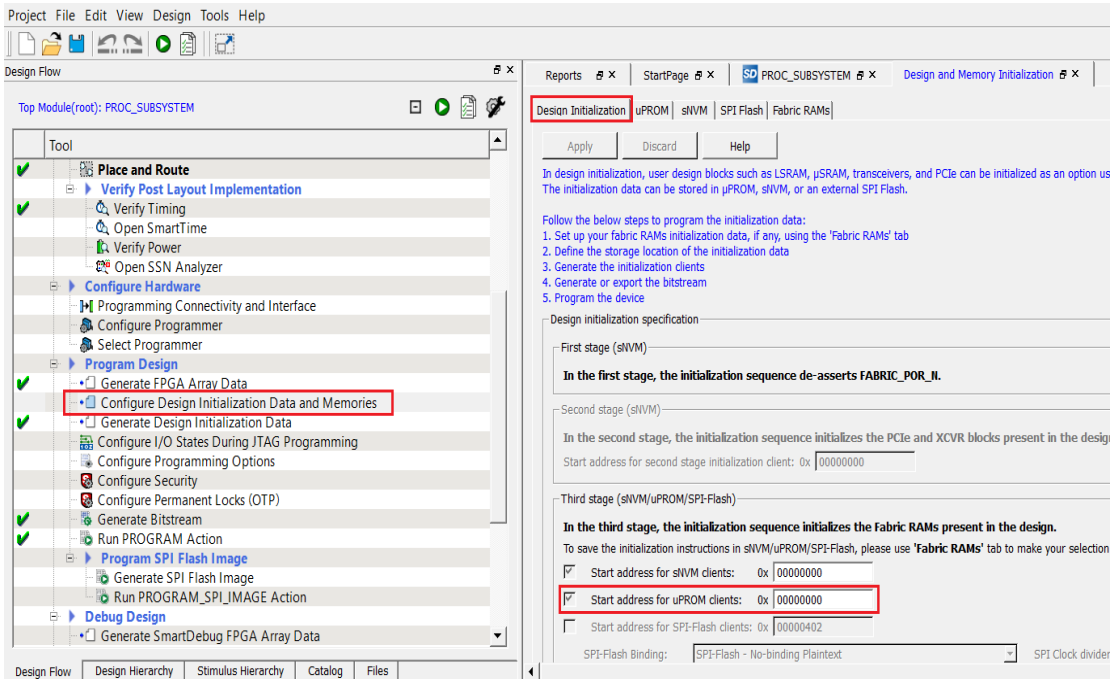
The **Configure Design Initialization Data and Memories** step generates the LSRAM initialization client and adds it to sNVM, µPROM, or an external SPI flash, based on the type of non-volatile memory selected. In the demo, the LSRAM initialization client is stored in the µPROM.

This process requires the user application executable file (hex file) to initialize the LSRAM blocks on device power-up. The hex file (`application.hex`) is available in the `DesignFiles_Directory\Libero_Project\hw_project` folder. When the hex file is imported, a memory initialization client is generated for LSRAM blocks.

Follow these steps:

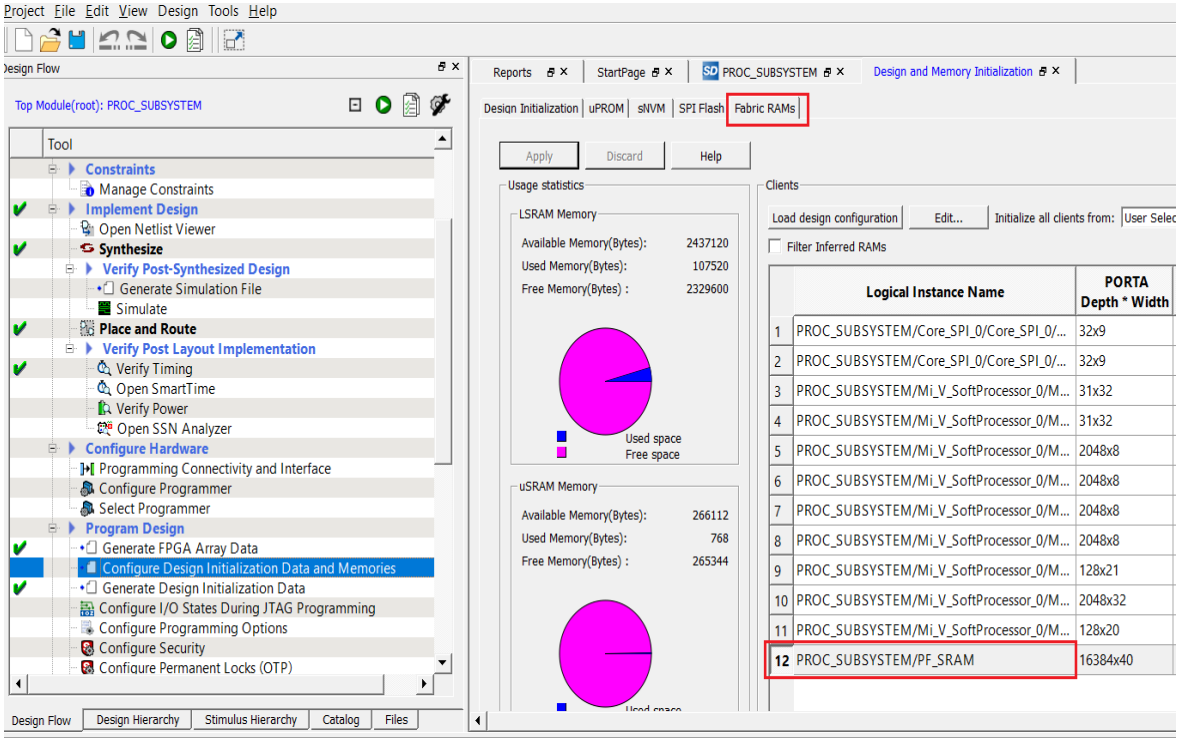
1. Double-click **Configure Design Initialization Data and Memories** from the **Design Flow** window. The **Design and Memory Initialization** window opens as shown in the following figure.

Figure 21 • **Design and Memory Initialization**



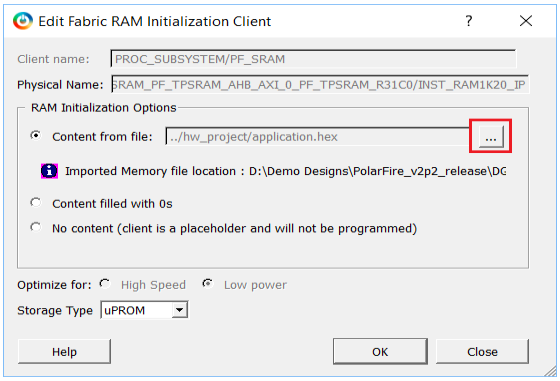
2. Select the **Fabric RAMs** tab and select the **PF_SRAM** client from the list and click **Edit** as shown in the following figure.

Figure 22 • **Fabric RAMs Tab**



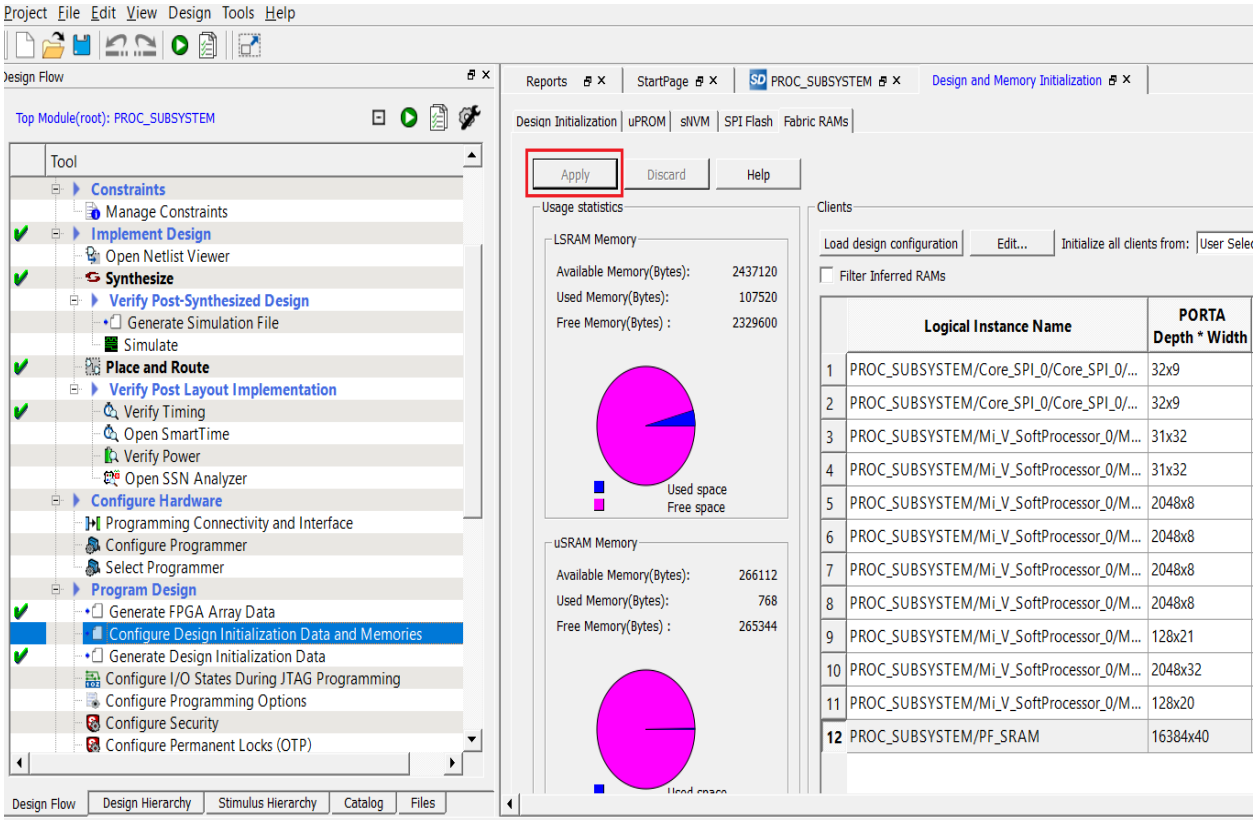
3. In the **Edit Fabric RAM Initialization Client** dialog box, select the **Content from file** option, and locate the `application.hex` file from `DesignFiles_directory\Libero_Project\hw_project` folder and Click **OK** as shown in the following figure.

Figure 23 • **Edit Fabric RAM Initialization Client**



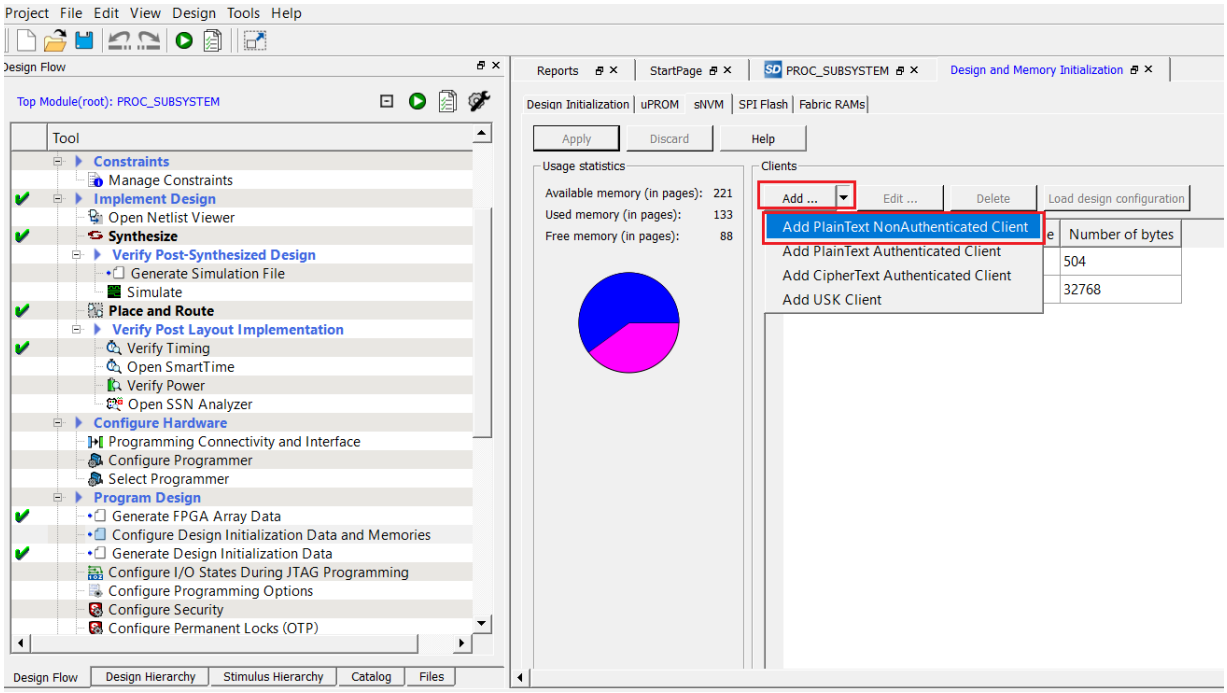
4. Click **Apply** as shown in the following figure.

Figure 24 • **Apply Fabric RAM Content**



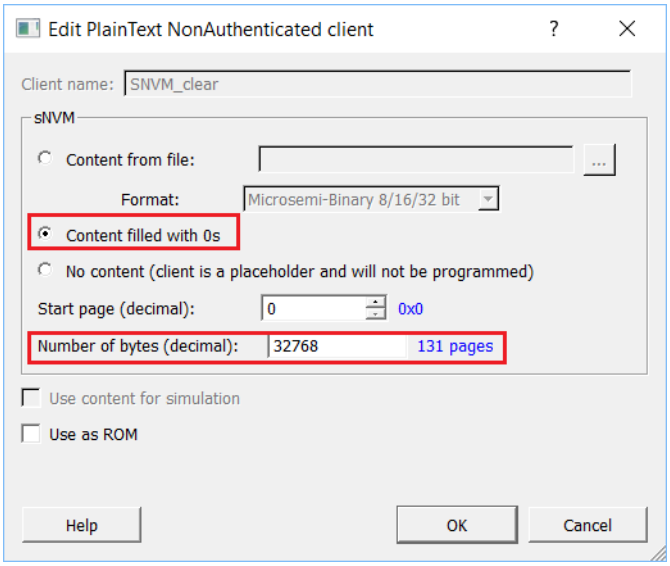
5. Select the **sNVM** tab and select the **Add** from the list. Click **Add PlainText NonAuthenticated** Client as shown in the following figure.

Figure 25 • **Add PlainText NonAuthenticated Option**



6. In the **Edit PlainText NonAuthenticated** client dialog box, select the **Content filled with 0's** option, and provide the **Number of bytes** and Click **OK** as shown in the following figure.

Figure 26 • **Edit PlainText NonAuthenticated Client**



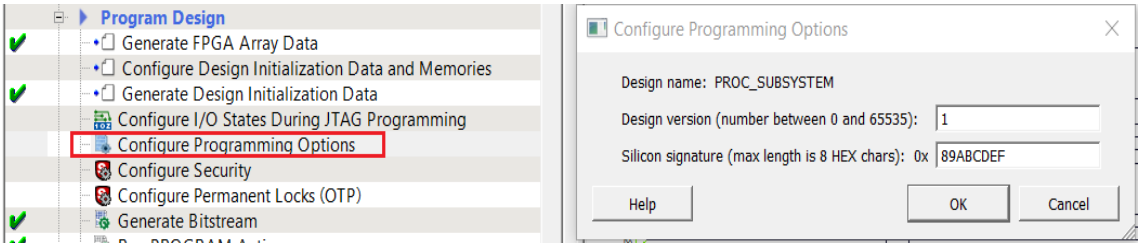
7. Click **Apply** in the **Design Initialization** tab.
8. In Libero Design Flow, click **Generate Initialization Data** to generate design initialization data.

After successful generation of the Initialization data, a green tick mark appears next to **Generate Initialization Data** option as shown in the Figure 20, page 20.

3.6 Configure Programming Options

The Design version and user code (Silicon signature) are configured in this step. Double click `Design flow->Program and Debug Design->Configure Programming Options` to give values as shown in the following figure.

Figure 27 • **Configure Programming Options**



3.7 Generate Bitstream

To generate the bitstream:

1. Double-click **Generate Bitstream** from the **Design Flow** tab. When the bitstream is successfully generated, a green tick mark appears as shown in Figure 20, page 20
2. Right-click **Generate Bitstream** and select **View Report** to view the corresponding log file in the **Reports** tab.

3.8 Run PROGRAM Action

After generating the bitstream, the PolarFire device must be programmed with the system services design.

Follow these steps to program the PolarFire device:

1. Ensure that the following jumper settings are set on the board.

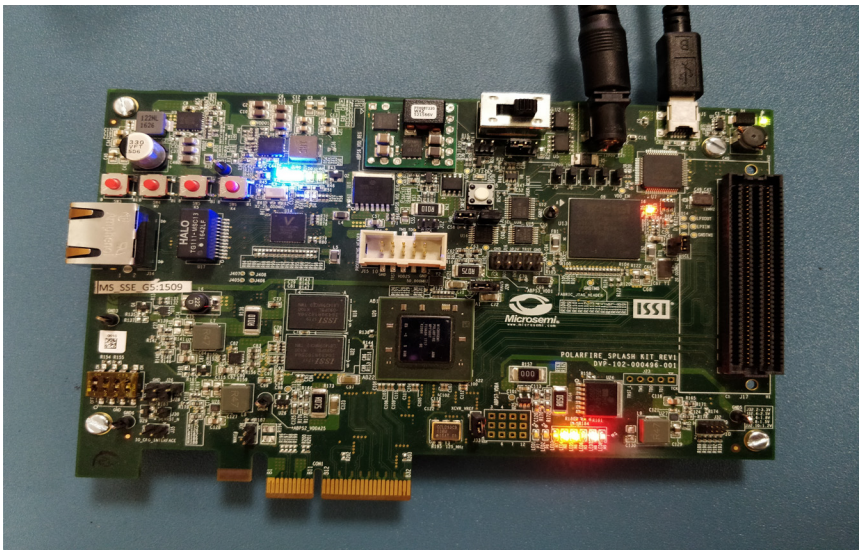
Table 6 • **Jumper Settings for PolarFire Device Programming**

Jumper	Description
J5, J6,J7, J8,J9	Close pin 2 and 3 for programming the PolarFire FPGA through FTDI
J11	Close pin 1 and 2 for programming through FTDI chip
J10	Close pin 1 and 2 for programming through FTDI SPI
J4	Close pin 1 and 2 for manual power switching using SW1
J3	Open pin 1 and 2 for 1.0 V

2. Connect the power supply cable to the **J2** connector on the board.
3. Connect the USB cable from the host PC to the **J1** (FTDI port) on the board.
4. Power on the board using the **SW1** slide switch.

The following figure shows the board setup after these connections are made.

Figure 28 • **Board Setup**



5. Double-click **Run PROGRAM Action** from the **Libero Design Flow**.

The device is successfully programmed and the onboard LEDs 4, 5, 6, 7, and 8 glow. A green tick mark appears next to **Run PROGRAM Action** as shown in Figure 20, page 20.

4 Programming the Device Using FlashPro

This chapter describes how to program the PolarFire device with the `.stp` programming file using FlashPro software without opening Libero SoC PolarFire. The `system_services.stp` file is available at the following design files folder location:

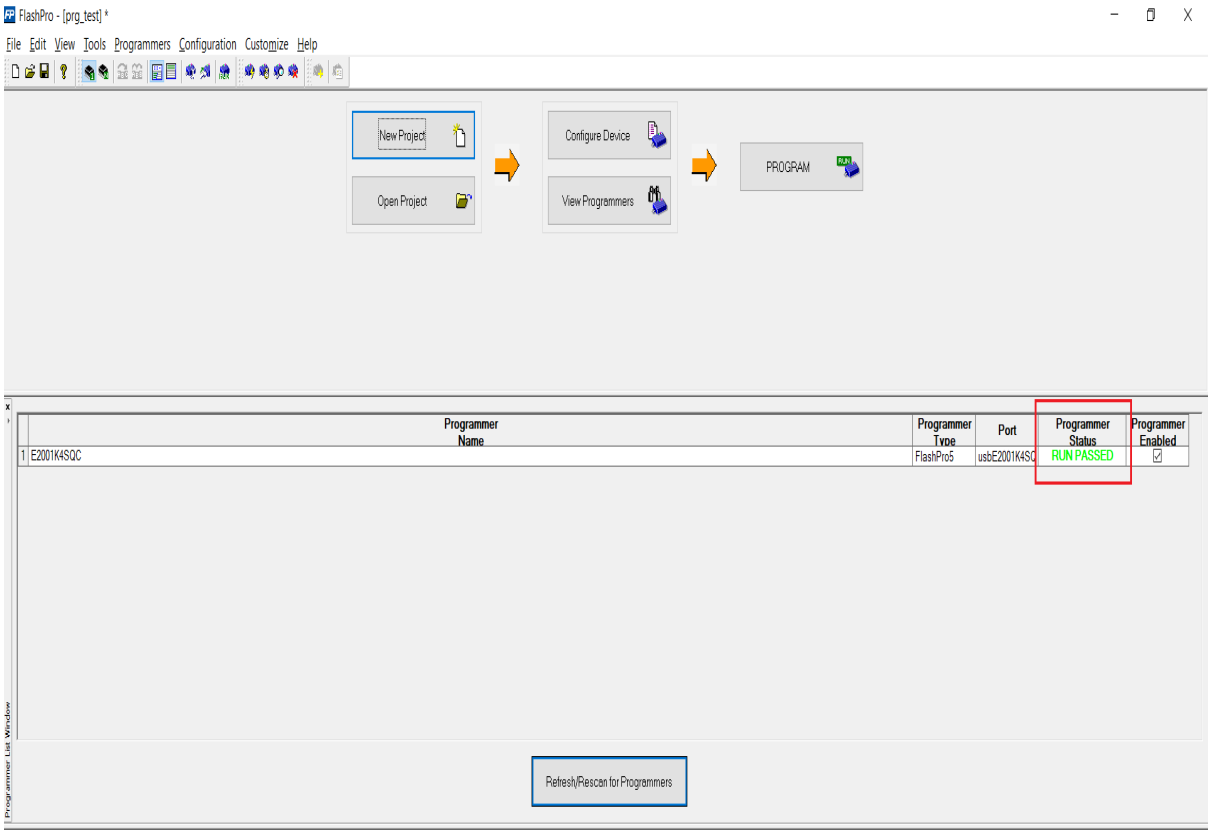
`mpf_dg0824_liberosocpolarfirev2p2_df\Programming_file`

Follow these steps to program the device:

1. Connect the jumpers and set up the PolarFire Splash Kit board as described in steps 1 to 4 of Run PROGRAM Action, page 26
2. Figure 28, page 26 shows the board setup after these connections are made.
3. On the host PC, start the FlashPro software.
4. Click **New Project** to create a new project.
5. In the **New Project** window, do the following, and click **OK**:
   - Enter a project name
   - Select **Single device** as the programming mode
6. Click **Configure Device**.
7. Click **Browse**, and locate the `system_services.stp` file from the following location.
   `mpf_dg0824_liberosocpolarfirev2p2_df\Programming_file`
8. Click **Program** to program the PolarFire Device.

The device is successfully programmed and the onboard LEDs 4, 5, 6, 7, and 8 glow. The **RUN PASSED** message is displayed as shown in the following figure.

Figure 29 • **Run Passed**



After the device is programmed, PuTTY must be launched to run the system services demo. See, Setting up the Serial Terminal Program - PuTTY, page 28.

5 Setting up the Serial Terminal Program - PuTTY

The user application receives the system service commands on the serial terminal through the UART interface. This chapter describes how to set up the serial terminal program.

To Setup PuTTY, perform the following steps:

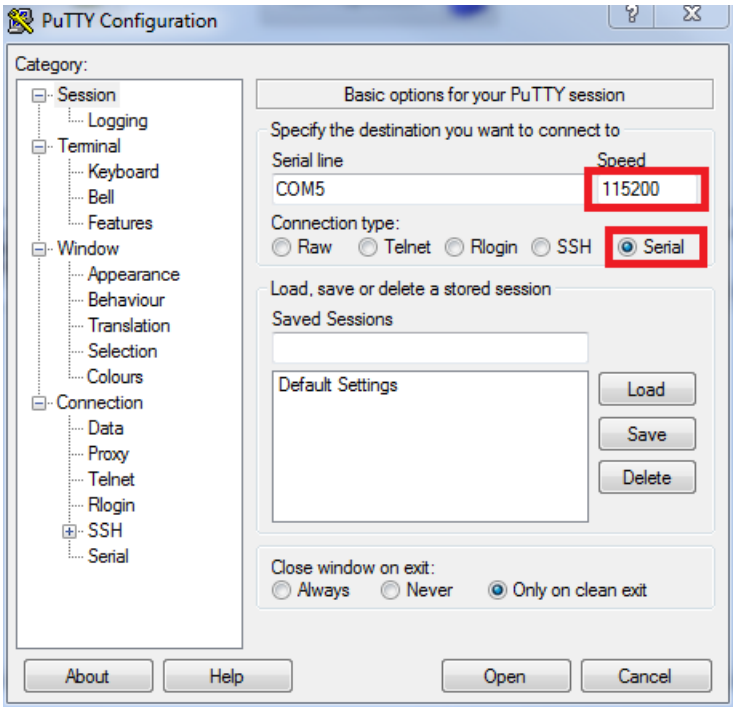
1. Connect the USB cable from the host PC to the **J1** (USB) port on the board.
2. Connect the power supply cable to the **J2** connector on the board.
3. Power on the board using the **SW1** slide switch.
4. From the host PC, click **Start** and open **Device Manager** to note the second highest COM Port number and use that in the PuTTY configuration. In this example, COM Port 5 (**COM5)** is selected as shown in the following figure. COM Port-numbers may vary.

Figure 30 • **Finding the COM Port**



5. From the host PC, click **Start,** and then find and select the PuTTY program.
6. Select **Serial** as the **Connection type** as shown in the following figure.

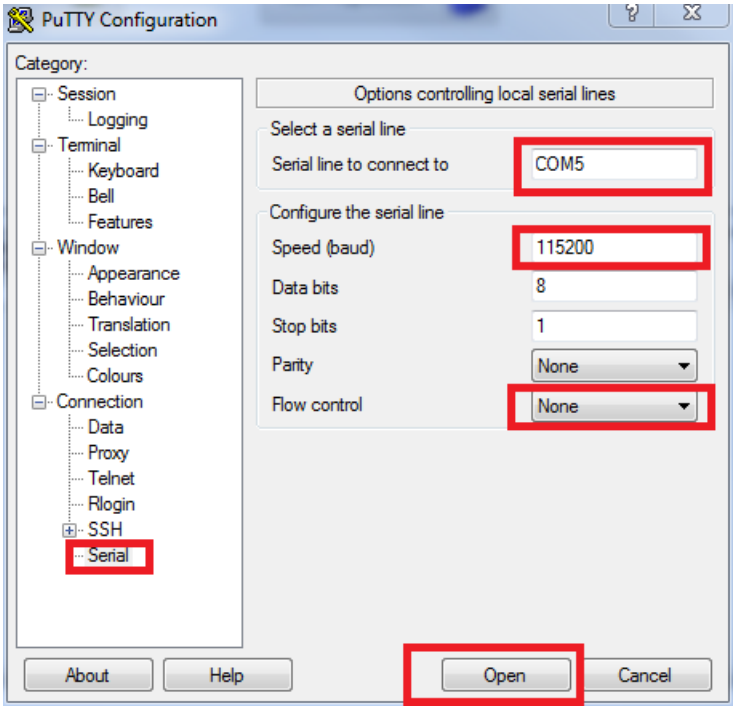
Figure 31 • **Select Serial as the Connection Type**



7. Set the **Serial line to connect to** COM port number noted in step 3.

8. Set the **Speed (baud)** to 115200 as shown in the following figure.

Figure 32 • **PuTTY Configuration**



9. Set the **Flow control** to **None** as shown in the following figure and click **Open**.

PuTTY opens successfully, and this completes the serial terminal emulation program setup. See Running the Demo, page 30.

6 Running the Demo

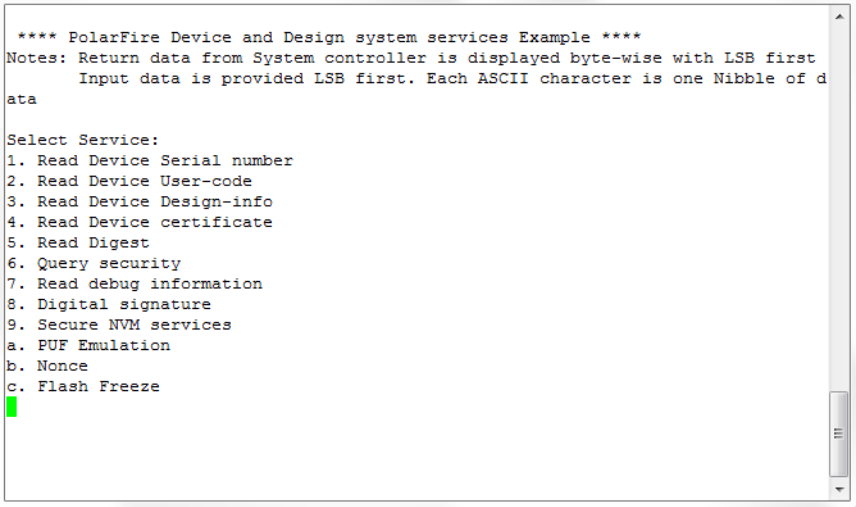
This chapter describes how to run the system services demo using the serial terminal program (PuTTY). Before running the demo, ensure the device is programmed and serial terminal is set up. For more information on setting up the serial terminal, see Setting up the Serial Terminal Program - PuTTY, page 28.

To run the demo, perform the following steps:

1. Power on the board using the **SW1** slide switch.

System services options are displayed on the PuTTY as shown in the following figure.

Figure 33 • **System Services Options**



2. Enter 1 to select **Read Device Serial number**.

The 128-bit device serial number (DSN) is displayed as shown in the following figure.

Figure 34 • **Device Serial Number**

```
------------------------------------------------------------------------
Device serial number:  42ED5BFF9B98F4B55FDDF4DEE8898F7B
------------------------------------------------------------------------
```

Each PolarFire FPGA device has a unique, publicly readable, 128-bit device serial number (DSN). The DSN can be used in cryptographic protocols to uniquely identify the device. The DSN comprises two 64-bit fields: FSN and SNM.

FSN—The first (most significant) field is the factory serial number (FSN). FSN is a pseudo-random per-device unique value assigned during Microsemi's manufacturing test, and persists for the lifetime of the device.

SNM—The second component is the serial number modifier (SNM). SNM is initialized during factory test and is destroyed during the recoverable zeroization action. If the device is subsequently recovered, a new SNM is assigned such that each SNM generated for a given FSN, is unique.

Note: The system services main menu is displayed after the execution of any of the options.

3. Enter 2 to select **Read Device User-code**.

The 32-bit device USERCODE/Silicon signature is displayed as shown in the following figure.

Figure 35 • **Device User-code**

```
------------------------------------------------------------------------
32bit USERCODE/Silicon signature (MSB first):  89ABCDEF
------------------------------------------------------------------------
```

This can be configured from `Design flow->Program and Debug Design->Configure Programming Options.`

4. Enter 3 to select **Read Device Design-info**.

As shown in Figure 36, page 31, the device design information consists of:

- 256-bit user-defined Design ID
- 16-bit design version
  This can be configured from `Design flow->Program and Debug Design->Configure Programming Options.` In auto update programming, the current design version is compared with the available images in external SPI flash to initiate the auto update on power up.
- 16-bit design back-level
  This can be configured from `Design flow->Program and Debug Design->Configure Security`. When back level protection is enabled, the device can only be programmed if the target design version is more than the back level value.

Figure 36 • **Device Design Information**

```
-------------------------------------------
Design ID:
 2D8150524F435F53554253595354454D
 00000000000000000000000000000000
Design Version:  0100
Design Back-Level:  0000
-------------------------------------------
```

5. Enter 4 to select **Read Device certificate**.

The device supply chain assurance certificate is displayed as shown in Figure 37, page 32 and Figure , page 32.

For more information about decoding the device certificate see, Appendix: Device Certificate Information, page 37.

Figure 37 • **Device Certificate**

```
------------------------------------------------------------------------
Device Certificate:
 308203CB30820352A003020102021240
 D1AE3A0C5A6BB8C77EA69268DC77CFD3
 01300A06082A8648CE3D040303304931
 0B30090603550406130255533310C300A
 060355040B1303536F43310D300B0603
 55040A13044D534343311D301B060355
 040313143162336462333162363031 30
 313135353037376530201 70D31353130
 31343030303734375A180F3939393931
 32333132333539353 95A304E314C300B
 060355042C1304303030303013060355
 04040C0C506F6C617246697265202020
 3028060355042A0C214D504633303054
 2020202020202020202020202020202020
 20202020202020202020307630100607
 2A8648CE3D020106052B810400220362
 000400014CD8E496F4F65B5C280DC824
 3AB5C734593539B77C13A6B3B21EC992
 E371C77AB0EE49112D0307FDAB63521D
 C060043719A62D6A803E56E552DEFBB0
 988A834504D06D9313B731EBD08FFD88
 6816E029142C8E380E07002FD316C704
 7FF981090040364A662911B600821100
 42ED5BFF9B98F4B55FDDF4DEE8898F7B
 A38201D6308201D23012060A2B060104
 0182BD6401010404000000030818F06
 0A2B0601040182BD640102048180FFFF
 FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
 FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
 FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
 FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
 FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
 FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
 FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
 FFFFFFFFFFFFFFFFFFFFFFFF003012
 060A2B0601040182BD6401030404FFFF
 FFFF308186060A2B0601040182BD6401
 040478000000000000000000000000000
 000000000000000000000000000000000
 000000000000000000000000000000000
 000000000000000000000000000000000
 000000000000000000000000000000000
 000000000000000000000000000000000
 000000000000000000000000000000000
 000000000000000000000030818C060A
 2B0601040182BD640105047E00000000
 000000000000000000000000000000000
 000000000000000000000000000000000
 000000000000000000000000000000000
 000000000000000000000000000000000
 000000000000000000000000000000000
 000000000000000000000000000000000
 000000000000000000000000000000000
 000000000000000000000300A06082A86
 48CE3D040303036700306402307 03ECE
 0D8E3BC74E752F363F31F106F276088D
 798D0EBCC176A4AB9F0D4D5EF0621652
 236F389F068AB81724B1A68616023031
 34F47534CE7D9D127F004794B9A8EF67
 872A1CA0358DC8B34977C964A9857F6C
 A752CE2684808E76FF71E91299232000
 000000000000000000000000000000000
 000000000000000000000000000000000
 000000000000000000000000000000000
------------------------------------------------------------------------
```

6. Enter 5 to select **Read Digest**.

The 416 byte Digest contains the fabric digest, sNVM digest and user key digests. The Digest protects the data integrity. The following figure shows the 416 byte digest displayed. For more information about decoding the digest output see, Appendix: Digest Information, page 41

Figure 38 • **Digest**

```
------------------------------------------------------------------------
Read Digest:
 01067C0FA01C52CAAB4F853E811F2E53
 5A35A03121DBBE5F5D1B32376A7C4ACE
 70B00F45420C2CA6838C1BAB947B892A
 3B070F7327A0B88FE2CBE3B974284C6D
 E3B0C44298FC1C149AFBF4C8996FB924
 27AE41E4649B934CA495991B7852B855
 223186F913839429B186AC19F222D712
 2212B305ACAAE0725D96B9FB918F34BC
 00000000000000000000000000000000
 00000000000000000000000000000000
 00000000000000000000000000000000
 00000000000000000000000000000000
 2EA9AB9198D1638007400CD2C3BEF1CC
 745B864B76011A0E1BC52180AC6452D4
 F5A5FD42D16A20302798EF6ED309979B
 43003D2320D9F0E8EA9831A92759FB4B
 00000000000000000000000000000000
 00000000000000000000000000000000
 2EA9AB9198D1638007400CD2C3BEF1CC
 745B864B76011A0E1BC52180AC6452D4
 F5A5FD42D16A20302798EF6ED309979B
 43003D2320D9F0E8EA9831A92759FB4B
 00000000000000000000000000000000
 00000000000000000000000000000000
 B108CF3C9D97E2EE1AA651BC2DAC018E
 CD456DB9F3CD6F52566E272BF58968A6
------------------------------------------------------------------------
```

7. Enter 6 to select **Query Security**.

The non-volatile states of user security locks are displayed as shown in the following figure.

Figure 39 • **Security Locks Information**

```
------------------------------------------------------------------------
Security locks:  0000000000000000
------------------------------------------------------------------------
```

The design does not include any security settings for device safety. Security locks can be configured from `Design flow->Program and Debug Design->Configure Security.` For more information about security locks, see Appendix: Query Security, page 39.

Note: If security locks are not handled properly, the device can go into the locked state.

8. Enter 7 to select **Read debug information**.

The debug information is displayed as shown in the following figure.

Figure 40 • **Debug Information**

```
------------------------------------------------------------------------
Debug info:
 0000003B2D0027002700B809AC09B409
 FE064302A2110000000000000000000
 0102BC4D01062EFA83001BAB00000000
 00010F7300000000402400004200000
 00B0C44298FC1C1400000000
------------------------------------------------------------------------
```

In Figure 40, page 33, the highlighted 4 bytes 42000000 (LSB first) indicate the number of times (in this example,00000042), the device was programmed (programming cycles). For more information about the Debug Info fields, see Appendix: Debug Information, page 40.

9. Enter 8 to select **Digital Signature**.

The digital signature in both Raw and DER formats are displayed in the following figure.

Figure 41 • **Digital Signature**

```
------------------------------------------------------------------------
Digital Signature service:
48 byte hash value:
 9C38A6E8D06004199740984213C554E6
 C71F7F0EC8A42800FEC72D43899167F8
 9C38A6E8D06004199740984213C554E6
Raw format:
Digital Signature service successful.
Output Digital Signature - Raw format:
 92189AAA8573E2BAEDF4DCA99E6A853F
 CB6819278CA550CF7C01657D26066B53
 8757032485D06C887F6FDCACDFF6A9C1
 C31F0288B15404873C9D4536BE91A4AD
 9D2D883D0B94FB1F81F0DC85F3858464
 BE61794645585F24E56E8B7AE21CB92CB
DER format:

Digital Signature service successful.
Output Digital Signature - DER format:
 3066023100FD090760FC8EFA8B4D6612
 67030BEEE9F8F3D692994AA006B4E272
 7697D582D01E3586149C892600A0300F
 F6EA6EC83A023100D0AEE07D47CDEA42
 273F605A27FB61403D4959E84D413A88
 C82BA33F936255256FA3D2E529B9EA2D
 CD04F0D8822A40AE
------------------------------------------------------------------------
```

The digital signature service takes a user-supplied SHA384 hash and signs it with the device private key. The application randomly generates the SHA384 hash value. The Digital Signature service sends the hash value to the system controller. The Athena core runs elliptic curve digital signature algorithm (ECDSA) using the hash and the device private key to generate the signature.

10. Enter 9 to select **Secure NVM.**

When the sNVM page/ module address is entered (in this case, 0), the randomly generated 60 byte data is written to the specified sNVM page and read back, as shown in Figure 42, page 35.

Figure 42 • **Secure NVM Services**

```
------------------------------------------------------------------------
Secure NVM (sNVM) Functions
sNVM write format
    Non-authenticated plain-text

------------------------------------------------------------------------------
Enter the sNVM page/module address (1 HEX byte. Maximum Page value is 0xDC) and Press Enter key:
0
Input Data(60 Byte):
A5ADC27946476F0FE3D0751B4F90120D
E075DD37A5ADC27946476F0FE3D0751B
4F90120DE075DD37A5ADC27946476F0F
E3D0751B4F90120DE075DD37
Secure NVM write successful.
------------------------------------------------------------------------

Data read from sNVM region:
A5ADC27946476F0FE3D0751B4F90120D
E075DD37A5ADC27946476F0FE3D0751B
4F90120DE075DD37A5ADC27946476F0F
E3D0751B4F90120DE075DD3700000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000
```

11. Enter ‘a’ (without quotes) to select **PUF Emulation_service**

The PUF emulation service provides a mechanism for authenticating a device, or for generating a psuedo random bit strings that can be used for different purposes. When this service is selected, the service by default accepts a 128-bit challenge and an 8-bit optype, and returns a 256-bit response unique to the challenge and the optype as shown in the following figure.

Figure 43 • **PUF Emulation Service**

```
------------------------------------------------------------------------
The challenge OPTYPE range(0x0 to 0xFF):  5B
16 byte challenge:
 5B695B9C6D4507A3FD356667FEF42301
PUF emulation service successful.Generated Response:
 94F3C480D0177D643998D9AC4A8CB25E
 AD176DF50D7D794A795B2A4A4A0C7CCC
------------------------------------------------------------------------
```

12. Enter ‘b’ to select **Nonce service**.

The 32 byte nonce value is displayed as shown in the following figure. The nonce service provides the ability to strengthen the deterministic random bit generator (DRBG) of the Athena by providing an alternate entropy source to use as additional seed data in its DRBG functions.

Figure 44 • **Generated Nonce**

```
------------------------------------------------------------------------
Generated Nonce:
 F8144860BF0320BD4021727E05BA4062
 5F12D647D5D13ED6DAB63EAC2344169A
------------------------------------------------------------------------
```

13. Enter ‘c’ to select **Flash*Freeze service**.

The device enters Flash*Freeze state for 5 seconds and exits automatically. The Flash*Freeze duration is shown in Figure 45, page 36.

Figure 45 • **Flash Freeze**

```
------------------------------------------------------------------------

Device will exit Flash Freeze after the 5 seconds.
```

In the Flash*Freeze state LED7 glows. Because, the last value is selected for I/O state in Flash*Freeze mode in the Libero I/O Editor as shown in the following figure

Figure 46 • **I/O State in Flash*Freeze**

Port View [active] | Pin View | Memory View | IOD View | XCVR View | Package View | Floorplanner View

	Port Name	Direction	I/O Standard	Pin Number	Locked	Macro Cell	Bank Name	User I/O Lock Down	I/O state in Flash*Freeze mode	Clamp Diode
1	CLK	Output	--	G3	☑	ADLIB:PF_SPI	--	☐	--	--
2	DI	Input	--	G10	☑	ADLIB:PF_SPI	--	☐	--	--
3	DO	Output	--	G5	☑	ADLIB:PF_SPI	--	☐	--	--
4	FLASH	Input	--	H10	☑	ADLIB:PF_SPI	--	☐	--	--
5	▼ GPIO_IN	Input	LVCMOS18		☑	ADLIB:INBUF	Bank4	☐	LAST_VALUE	ON
6	GPIO_IN[2]	Input	LVCMOS18	L3	☑	ADLIB:INBUF	Bank4	☐	LAST_VALUE	ON
7	GPIO_IN[3]	Input	LVCMOS18	M4	☑	ADLIB:INBUF	Bank4	☐	LAST_VALUE	ON
8	▼ GPIO_OUT	Output	LVCMOS18		☐	ADLIB:OUTBUF		☐	LAST_VALUE	ON
9	GPIO_OUT[0]	Output	LVCMOS18	P7	☑	ADLIB:OUTBUF	Bank4	☐	LAST_VALUE	ON
10	GPIO_OUT[1]	Output	LVCMOS18	P8	☑	ADLIB:OUTBUF	Bank4	☐	LAST_VALUE	ON
11	GPIO_OUT[2]	Output	LVCMOS18	M2	☐	ADLIB:OUTBUF	Bank4	☐	LAST_VALUE	ON
12	GPIO_OUT[3]	Output	LVCMOS18	Y3	☐	ADLIB:OUTBUF	Bank0	☐	LAST_VALUE	ON
13	IFACE	Input	--	G9	☑	ADLIB:PF_SPI	--	☐	--	--
14	LED6	Output	LVCMOS18	N7	☑	ADLIB:OUTBUF	Bank4	☐	LAST_VALUE	ON
15	LED7	Output	LVCMOS18	N8	☑	ADLIB:OUTBUF	Bank4	☐	LAST_VALUE	ON
16	REF_CLK_0	Input	LVCMOS18	H7	☑	ADLIB:INBUF	Bank5	☐	LAST_VALUE	ON
17	resetn	Input	LVCMOS18	N4	☑	ADLIB:INBUF	Bank4	☐	LAST_VALUE	ON
18	RX	Input	LVCMOS18	R5	☑	ADLIB:INBUF	Bank4	☐	LAST_VALUE	ON
19	SS	Output	--	G4	☑	ADLIB:PF_SPI	--	☐	--	--
20	TCK	Input	--	F8	☑	ADLIB:UJTAG	--	☐	--	--
21	TDI	Input	--	G8	☑	ADLIB:UJTAG	--	☐	--	--
22	TDO	Output	--	F6	☑	ADLIB:UJTAG	--	☐	--	--
23	TMS	Input	--	F7	☑	ADLIB:UJTAG	--	☐	--	--
24	TRSTB	Input	--	G7	☑	ADLIB:UJTAG	--	☐	--	--
25	TX	Output	LVCMOS18	R4	☑	ADLIB:OUTBUF	Bank4	☐	LAST_VALUE	ON

With this step the system services demo is successfully completed. Power cycle the board to run the system services options again.

7 Appendix: Device Certificate Information

The Device Certificate is a 1024-byte Microsemi-signed X-509 certificate programmed during manufacturing. This certificate is used to guarantee the authenticity of a device and its characteristics. The following table lists the main fields of the device certificate.

Table 7 • **Device Certificate Fields (1024 bytes)**

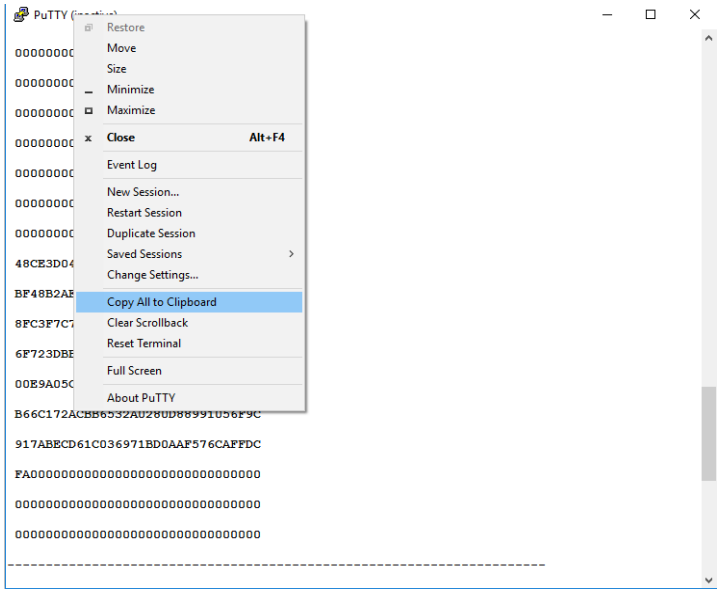
Offset (Byte)	Length (Bytes)	Data
4	854	Signed region of certificate
234	120	Device Public Key
368	16	DSN

The device certificate is encoded in the ASN.1 format. To view the content, the certificate must be decoded to a user readable format using the online JAVA tool: *http://lapo.it/asn1js/#*.

Follow these steps for decoding the certificate:

1. Right click PuTTy and select **Copy All to Clipboard,** and paste the same to notepad as shown in the following figure.

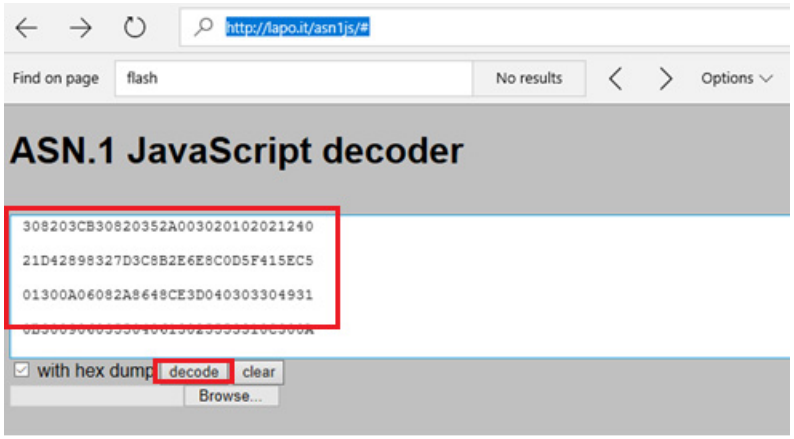
Figure 47 • **Copy Device Certificate**



2. Copy the 1024 bytes of device certificate from notepad to ASN.1 decoder as shown in the following figure and click **decode** button.

Note: The sample device certificate(1024 bytes) is provided at `mpf_dg0824_liberosocpolarfirev2p2_df\Device_certificate\sample.txt`.

Figure 48 • **Certificate Decoding Using Java Script**



3. The web page displays all the fields in certificate as shown in the following figure.

Figure 49 • **Decoded Certificate**



8 Appendix: Query Security

The following table lists each security lock bit and its features.

Table 8 • **Security Locks Fields**

Byte	Bit	Lock	Description
0			
	0	UL_DEBUG	Debug instructions disabled
	1	UL_SNVM_DEBUG	sNVM debug disabled
	2	UL_LIVEPROBE	Live probes disabled
	3	UJTAG_DISABLE	User JTAG interface disabled
	4	JTAG_BS_DISABLE	JTAG boundary scan disabled
	5	UL_TVS_MONITOR	External access to system temperature and voltage sensor (TVS) disabled
	6	JTAG_MONITOR	JTAG fabric monitor enabled
	7	JTAG_TAP	JTAG TAP disabled
1			
	0	UL_PLAINTEXT	Plain text passcode unlock disabled
	1	UL_FAB_PROTECT	Fabric erase/write disabled
	2	UL_EXT_DIGEST	External digest check disable
	3	UL_VERSION	Replay protection enabled
	4	UL_FACT_UNLOCK	Factory test disabled
	5	UL_IAP	IAP disabled
	6	UL_EXT_ZEROIZE	External zeroization disabled
	7	SPI_SLAVE_DISABLE	SPI port disabled
2-8		Reserved	Reserved

9 Appendix: Debug Information

The following table lists the bit fields of debug information.

Table 9 • **Debug Info Fields**

Byte Offset	Size (Bytes)	Parameter	Description
0	32	RESERVED	
32	4	TOOL_INFO	Reflects the TOOL_INFO passed in during ISC_ENABLE prior to programming IAP sets this to 0
36	1	TOOL_TYPE	Tool type used to program device 1=JTAG, 2=IAP, 3=SPI_SLAVE
37	4	RESERVED	
41	7	RESERVED	
48	1	UIC_STATUS	Design initialization status
49	1	UIC_SOURCE_TYPE	Design initialization Data source type when execution finished or halted
50	2	RESERVED	
52	4	UIC_START_ADDRESS	Design initialization Data source address when execution finished or halted
56	4	UIC_INSTR_ADDRESS	Design initialization Data instruction count from the start of Design initialization execution
60	4	CYCLECOUNT	Programming cycle count
64	1	IAP_ERROR_CODE	IAP error information
65	7	RESERVED	
72	4	IAP_LOCATION	External SPI flash memory address that was used during IAP

10 Appendix: Digest Information

The following table lists the bit fields of digest information.

Table 10 • **Digest Information Bit Fields**

Offset (byte)	Size (bytes)	Value	Note
0	32	CFD	Fabric digest
32	32	CCDIGEST	Fabric Configuration segment digest
64	32	SNVMDIGEST	sNVM Digest
96	32	ULDIGEST	User lock segment
128	32	UKDIGEST0	User Key Digest 0 in User Key segment (includes SRAM PUF activation code and device Integrity bit)
160	32	UKDIGEST1	User Key Digest 1 in User Key segment
192	32	UKDIGEST2	User Key Digest 2 in User Key segment (UPK1)
224	32	UKDIGEST3	User Key Digest 3 in User Key segment (UEK1)
256	32	UKDIGEST4	User Key Digest 4 in User Key segment (DPK)
288	32	UKDIGEST5	User Key Digest 5 in User Key segment (UPK2)
320	32	UKDIGEST6	User Key Digest 6 in User Key segment (UEK2)
352	32	UPDIGEST	User Permanent lock (UPERM) segment
384	32	FDIGEST	Digest for Factory Key Segments
Total	416 bytes		

11 Appendix: References

This section lists the documents that provide more information about system services and other IP cores used to build the demo design.

- For more information on Design and Data Security Services, see the *UG0753: PolarFire FPGA Security User Guide.*
- For more information about the CoreJTAGDEBUG IP core, see CoreJTAGDebug_HB.pdf from **Libero**->**Catalog**.
- For more information about the CoreAHBtoAPB3 IP core, see *CoreAHBtoAPB3_HB.pdf*.
- For more information about the CoreUARTapb IP core, see *CoreUARTapb_HB.pdf*.
- For more information about the CoreAHBLite IP core, see *CoreAHBLite_HB.pdf*.
- For more information about the CoreAPB3 IP core, see *CoreAPB3_HB.pdf*.
- For more information about the CoreGPIO IP core, see *CoreGPIO_HB.pdf*.
- For more information about the PolarFire initialization monitor, see *UG0725: PolarFire FPGA Device Power-Up and Resets User Guide*.
- For more information about how to build a Mi-V processor subsystem for PolarFire devices, see *TU0775: PolarFire FPGA: Building a Mi-V Processor Subsystem Tutorial*.
- For more information about the PF_CCC IP core, see *UG0684: PolarFire FPGA Clocking Resources User Guide*.
- For more information about the SRAM buffer, see *UG0680: PolarFire FPGA Fabric User Guide*.
- For more information about Libero, ModelSim, and Synplify, see the *Microsemi Libero SoC PolarFire webpage.*
- For more information about SoftConsole migration from v5.1 to v5.2,see *Migrating a SoftConsole v5.1 Project to SoftConsole v5.2 Application Note.*