AC471
Application Note
PolarFire FPGA Auto Update and In-Application Programming Using Splash Kit

a Microchip company

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

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.

51900471. 2.0 7/18

Contents

Figures

Tables

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

The document was updated for Libero SoC PolarFire v2.2 release.

1.2 Revision 1.0

The first publication of this document.

2 PolarFire FPGA Auto Update and In-Application Programming using Splash Kit

PolarFire® FPGAs support the SPI master programming mode for auto update and in-application programming (IAP). In this programming mode, the programming images are stored in an external SPI flash memory.

Auto update—on power-up, if the version of the update image is found to be different from the current programmed version, the System Controller reads the update image bitstream from the external SPI flash memory and programs the device.

IAP—the user application initiates the program action and the System Controller reads the bitstream from the external SPI flash memory to program the device.

The System Controller supports fetching programming images from SPI Flash device based on the Index value or direct addressing. The SPI directory contains the start addresses of the programming images.

The following components of PolarFire devices are programmable:

- FPGA fabric
- Secure non-volatile memory (sNVM)
- User security settings (keys, passcodes, and locks)

This document explains how to use the accompanying design to demonstrate the auto update and IAP features on the PolarFire Splash kit.

The on-board 1 GB Micron SPI flash device is connected to System Controller SPI and can be programmed using the fabric logic or Libero® SoC PolarFire software. For more information about programming the on-board SPI flash using Libero, see Appendix: Programming On-board SPI Flash Using Libero, page 37.

This application note includes the Mi-V soft processor, which initiates the system service requests for the device programming and enables the CoreSysService_PF IP core to access the System Controller. For more information about the design implementation, and the necessary blocks and IP cores instantiated in Libero SoC PolarFire, see Demo Design, page 6.

This 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 design, see Programming the Device Using FlashPro Software, page 29.
- **Using Libero SoC PolarFire:** To program the device using Libero SoC PolarFire, see Libero Design Flow, page 22.

This design can be used as a reference to build a fabric design with programming features.

2.1 CoreSysService_PF IP Overview

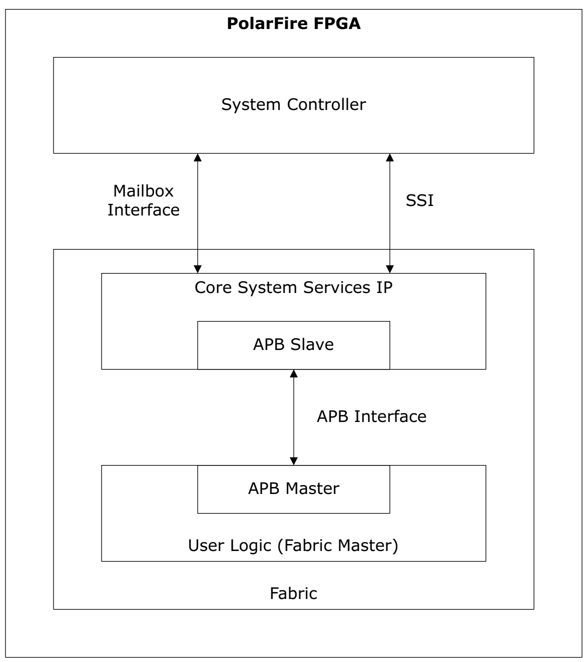
System Controller actions are initiated by the fabric logic through the system service interface (SSI) of the System Controller. The fabric logic requires the CoreSysService_PF IP for initiating the system services. A service request interrupt to the System Controller is triggered when the fabric user 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 1 • **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 CoreSysService_PF IP Interfaces with the fabric logic.

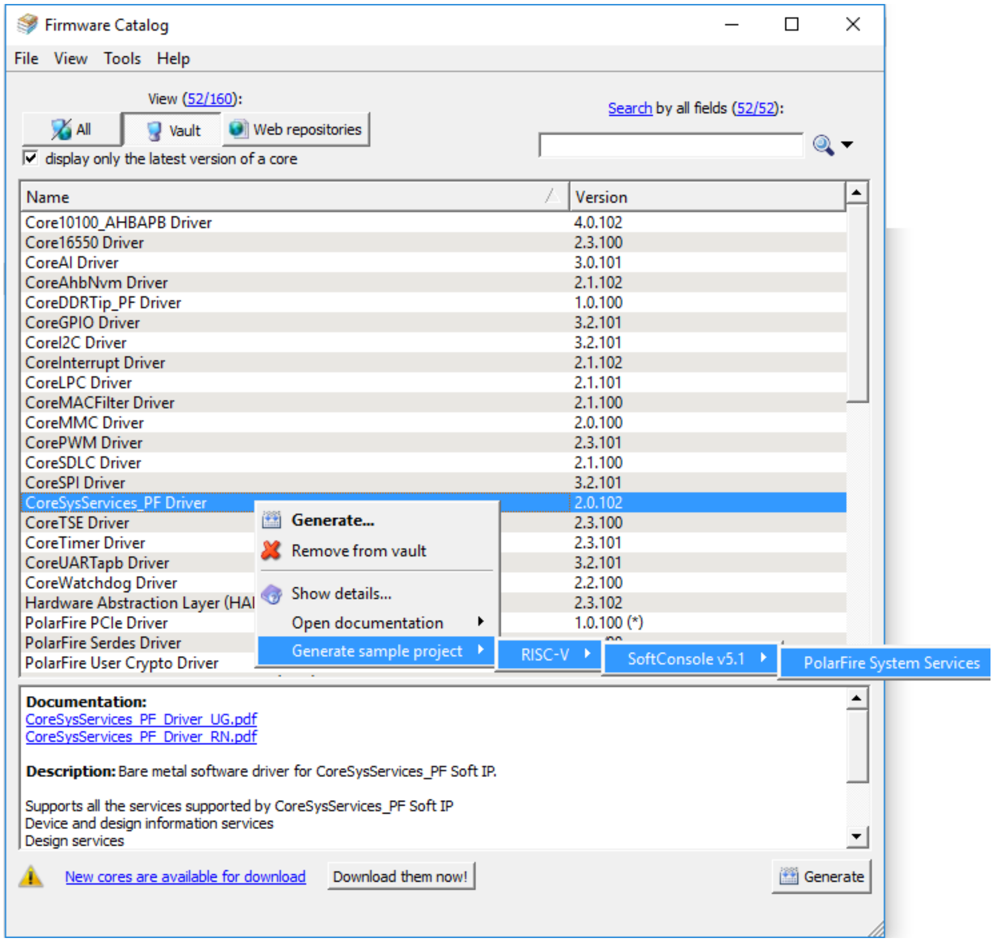
Figure 1 • **Core System Services IP Interfacing with Fabric User Logic**



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

In this design, the sample SoftConsole project is migrated to SoftConsole v5.2. The Mi-V soft processor is compatible with only SoftConsole v5.2 or later. The application files `main.c` and `hw_platform.h` are modified to provide the programming user options, system clock frequency, and APB peripheral addresses.

Figure 2 • **Firmware catalog**



2.2 Design Requirements

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

Table 2 • **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
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
CoreSysService_PF	2.3.116
CORESPI	5.1.104
PF_SRAM_AHBL_AXI	1.1.125
PF_SPI macro	
CLKINT	1.0

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

2.3 Prerequisites

Before you start:

1. Download the design files from the following location:
   http://soc.microsemi.com/download/rsc/?f=mpf_ac471_liberosocpolarfirev2p2_df
2. 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.

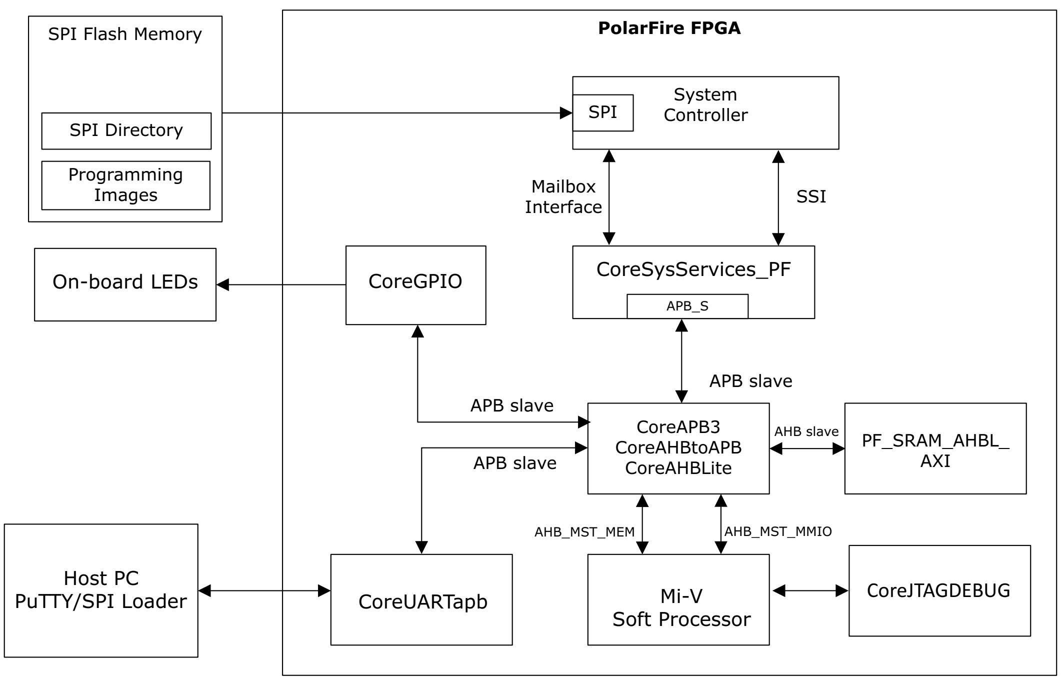
2.4 Demo Design

The following steps describe the data flow in the 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 CoreSysService_PF IP and sends the requested system service command to the System Controller.
3. The System Controller executes the system service command by reading the bitstream images from the external SPI flash and sends the relevant response to the CoreSysService_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 PolarFire programming design.

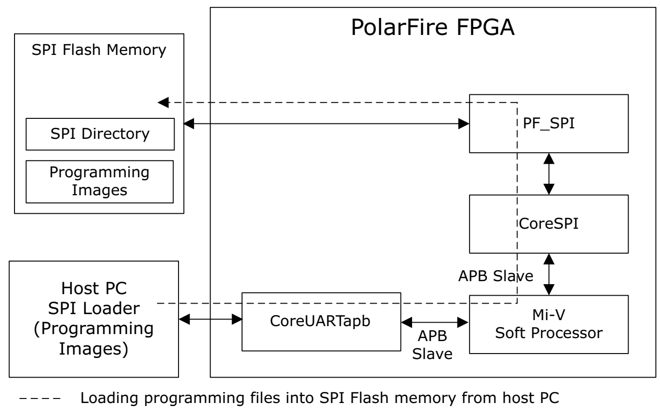
Figure 3 • **PolarFire Programming Design Block Diagram**



To initiate auto update or IAP system service request, the on-board SPI flash must be programmed with programming images. The fabric logic interfaces to the on-board SPI flash using SPI controller and PF_SPI macro. When the System Controller's SPI is enabled and configured as master, the System Controller hands over the control of the SPI to the fabric on device power-up. The fabric logic programs the on-board SPI flash with flash directory and programming images using UART interface. The programming images are transfered from the host PC using SPI flash loader (spi_loader.exe).

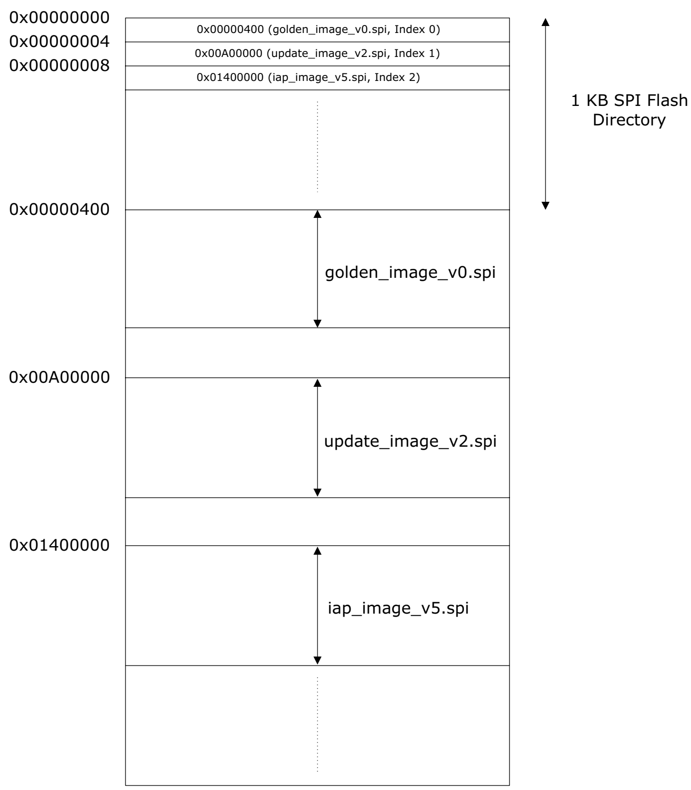
The on-board SPI flash can be programmed using fabric logic as shown in the following figure.

Figure 4 • **Accessing On-board SPI Flash Using Fabric**



The following figure shows the SPI flash memory with directory and programming images.

Figure 5 • **SPI Flash Memory**



When System Controller receives programming or authentication system service from fabric user logic, the System Controller fetches the programming images from the on-board SPI flash to execute the service request. In this application note, the following system services are initiated on user request.

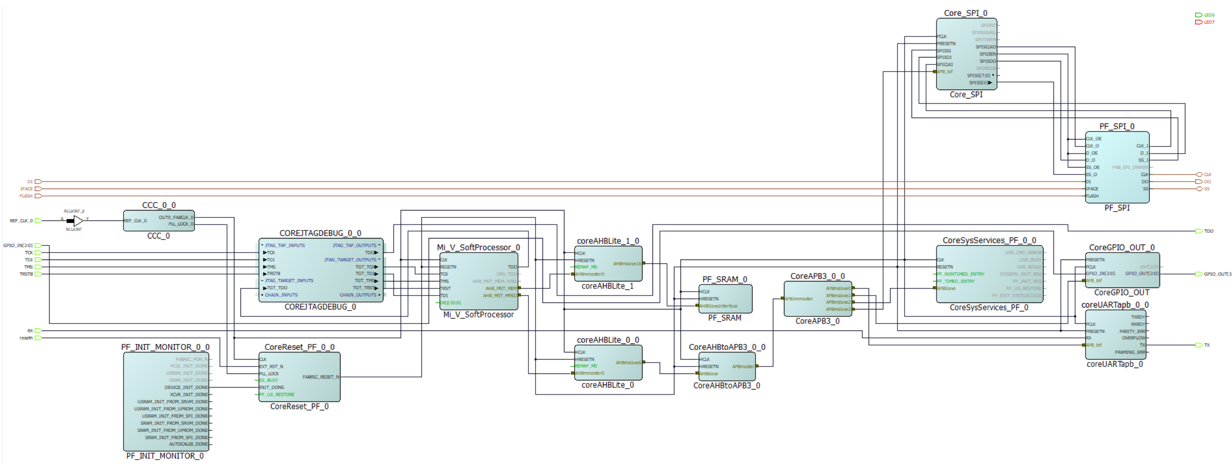
- Bitstream authentication
- IAP image authentication
- Auto update
- IAP

For more information about the preceding services, see the *UG0714: PolarFire FPGA Programming User Guide*.

2.4.1 Design Implementation

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

Figure 6 • Top Level Libero Design



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

Table 3 • I/O Signals

Signal	Description
REF_CLK_0	Input 50 MHz clock from the on-board 50 MHz oscillator
resetn	On-board 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]	On-board LED outputs
GPIO_IN[3:0]	To interface on-board DIP switches.

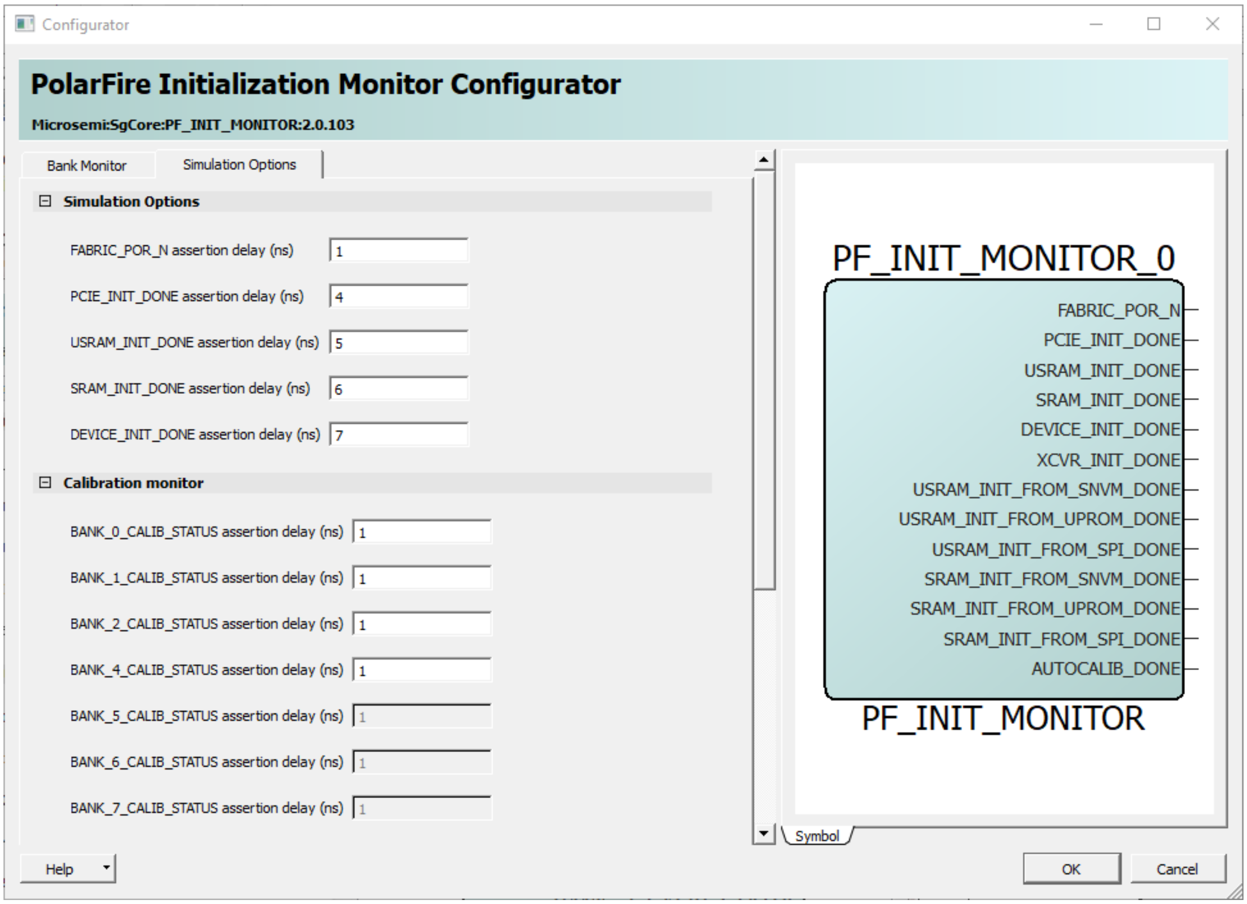
2.4.2 IP Configuration

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 7 • **PF_INIT_MONITOR Configuration**



2.4.2.2 Instantiating CLKINT

From the Catalog, drag the CLKINT macro to SmartDesign. This macro is required as a 50 MHz clock oscillator with an accuracy of +/-50 ppm is available on the board. This clock oscillator is connected to the FPGA fabric to provide a system reference clock. The pin number of the 50 MHz oscillator is H7, and the pin name is GPIO239PB5/CLKIN_W_2/CCC_SW_CLKIN_W_2/CCC_SW_PLL0_OUT0. When the pin is not hardwired to the PLL reference clock input, use CLKINT macro to promote it to global clock network.

2.4.2.3 PF_CCC_0 Configuration

The PolarFire Clock Conditioning Circuitry (CCC) block takes an input clock of 50 MHz from the on-board oscillator passed 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 8 • PF_CCC_0 Input Clock Configuration

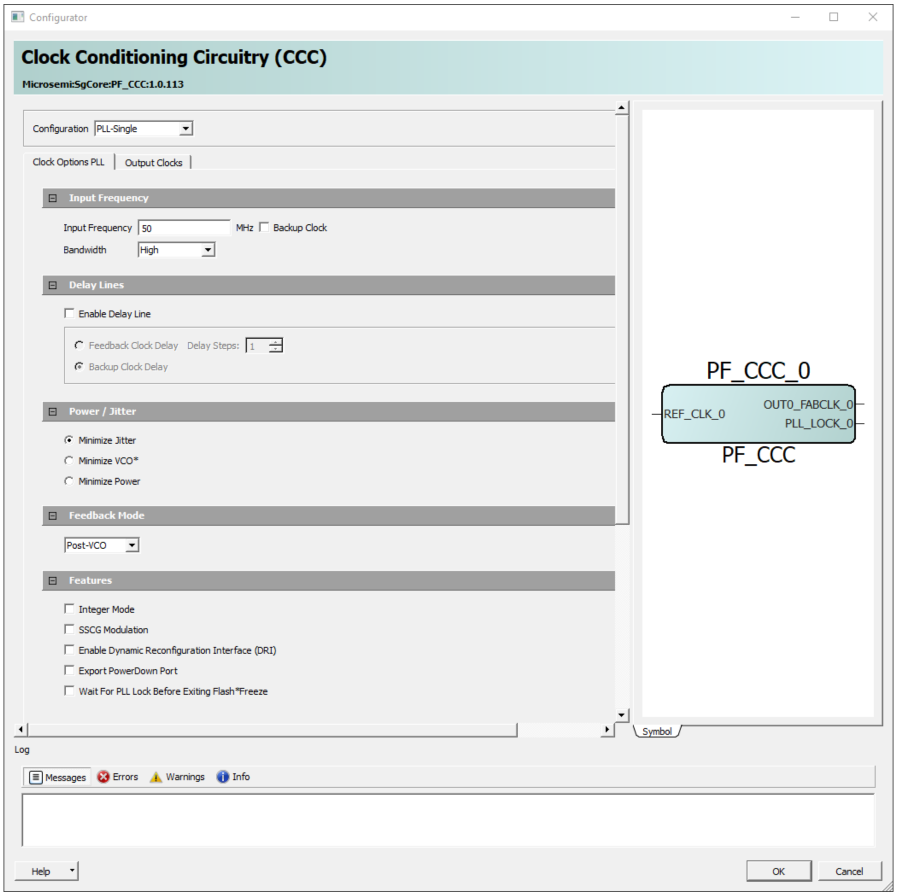
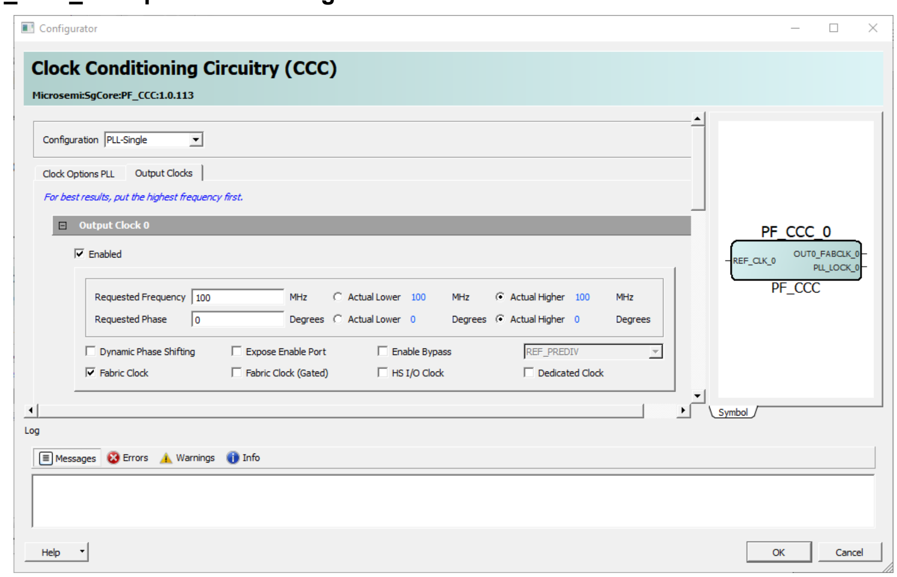


Figure 9 • PF_CCC_0 Output Clock Configuration

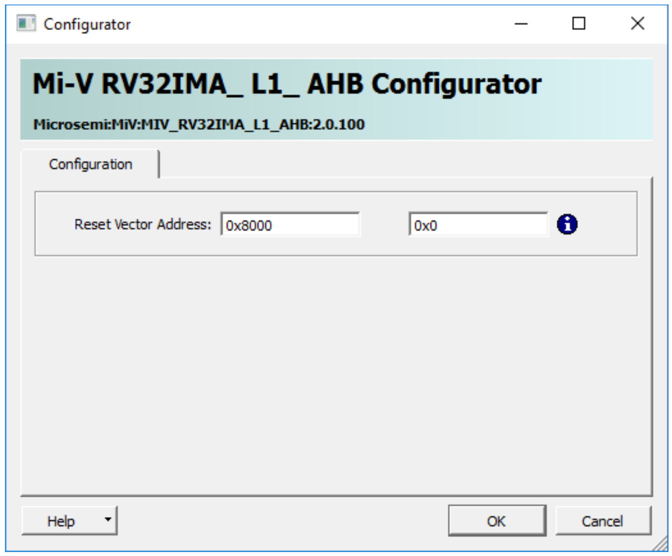


2.4.2.4 Mi-V Soft Processor Configuration

The Mi-V soft processor Reset Vector Address is set to 0x8000_0000 from default value 0x6000_0000. After 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 the following figure.

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 10 • **Mi-V Configuration**

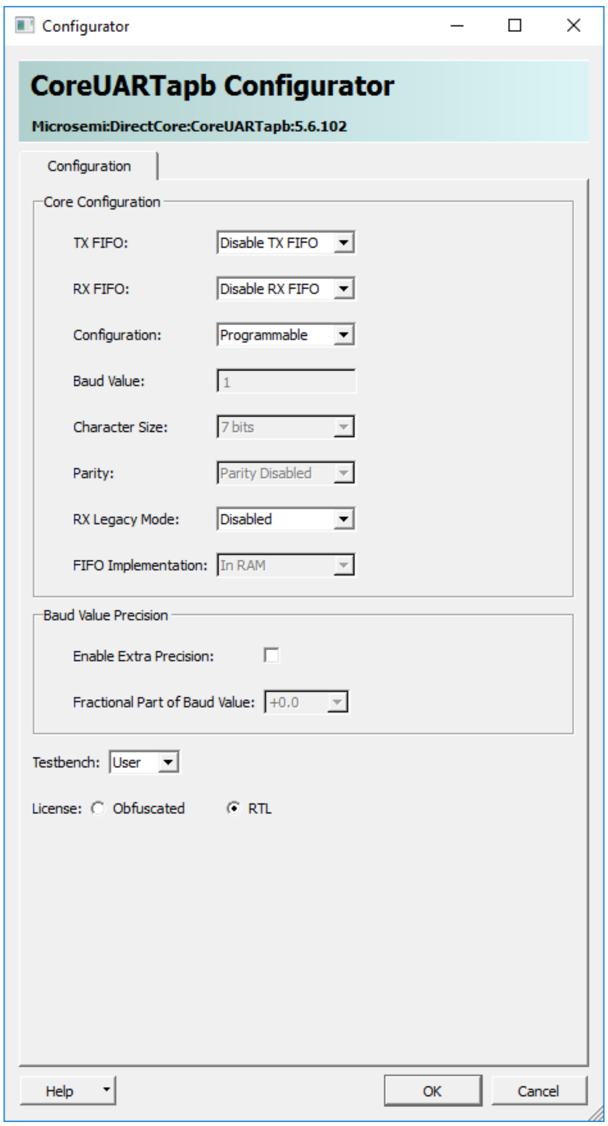


2.4.2.5 CoreUARTapb

The CoreUARTapb IP is connected to Mi-V soft processor as an APB slave. It interfaces with the host PC for UART communication. The default configuration settings of the CoreUARTapb IP are shown in the following figure:

- **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.

Figure 11 • **UART Configuration**



The SoftConsole 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.

2.4.2.6 CoreJTAGDEBUG

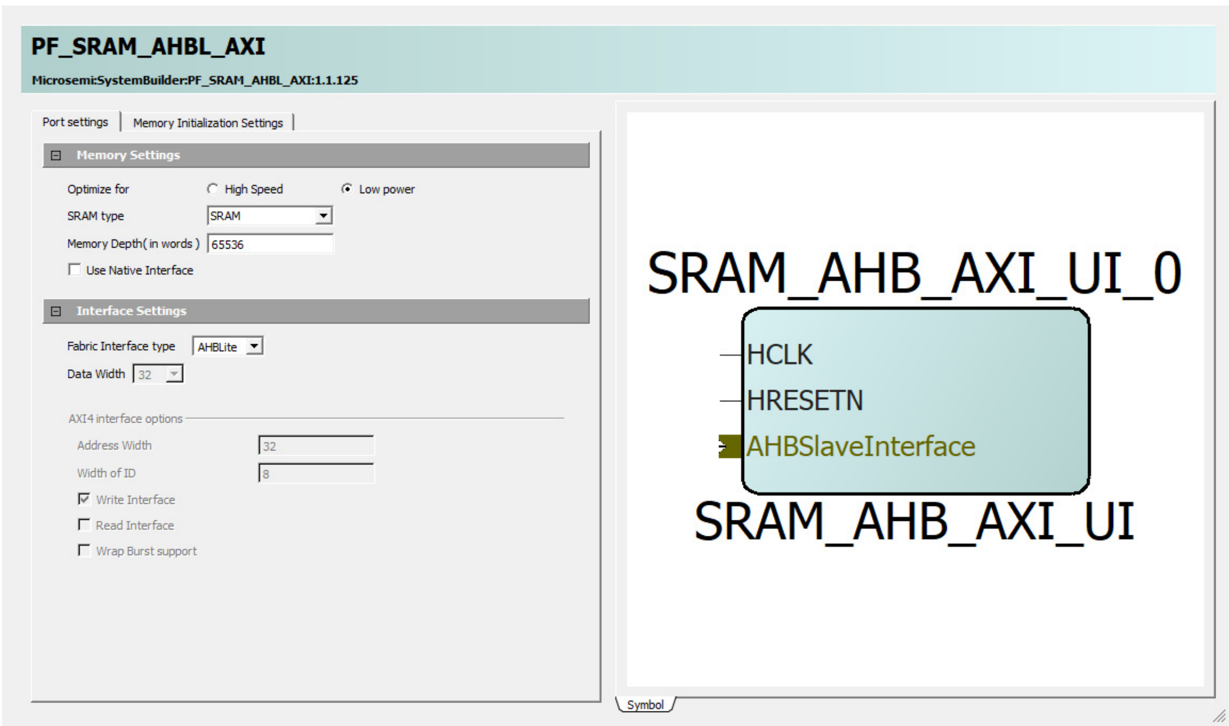
The CoreJTAGDebug IP connects the Mi-V soft processor to the JTAG header for debugging.

2.4.2.7 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 65536 words to accommodate an application of up to 256 KB into LSRAM. The present application is below 50 KB so this can fit into either sNVM or µPROM. In this design, µPROM is selected as data storage client. The following figure shows the PF_SRAM_AHBL_AXI (LSRAM_0) IP configuration.

Figure 12 • **PF_SRAM_AHBL_AXI Configuration**



2.4.2.8 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:

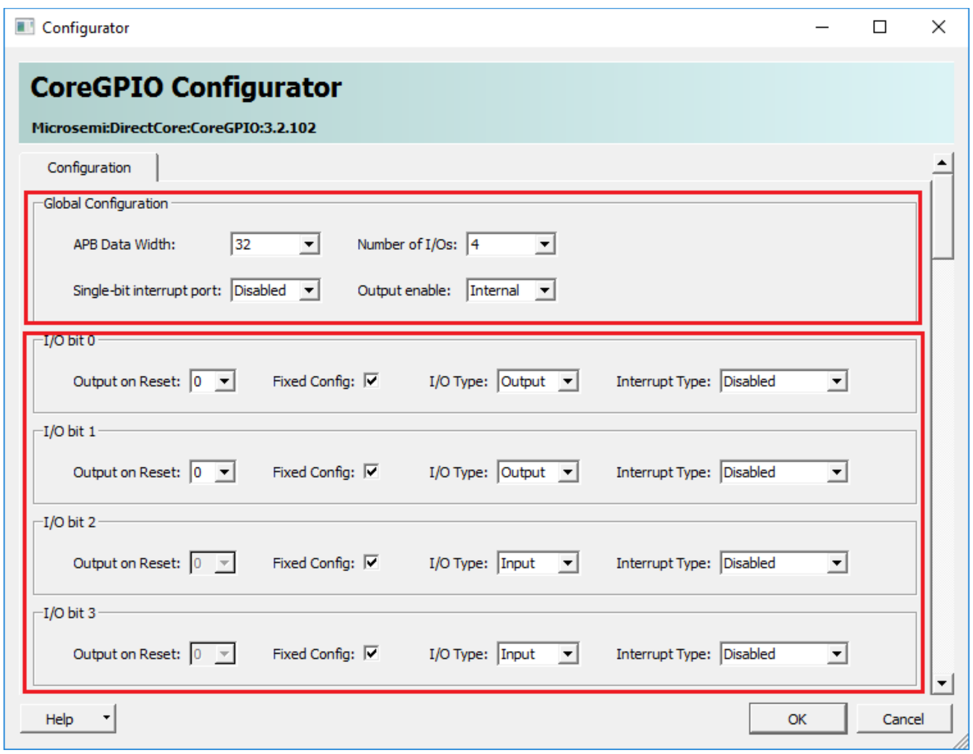
- **APB Data width** is set to 32
  The design uses 32-bit data width for APB read and write data.
- **Number of I/Os** is set to 4
  The design controls 2 on-board LEDs for output and 2 DIP Switches for input.
- **I/O Bit:** The following list shows the sub-options under I/O Bit option.
  - **Output 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 and DIP switches.

  - **Interrupt Type:** Disabled
    When I/O states change, no interrupt is required for the application.

The following figure shows the CoreGPIO_0 configuration.

Figure 13 • **CoreGPIO_0 Configuration**



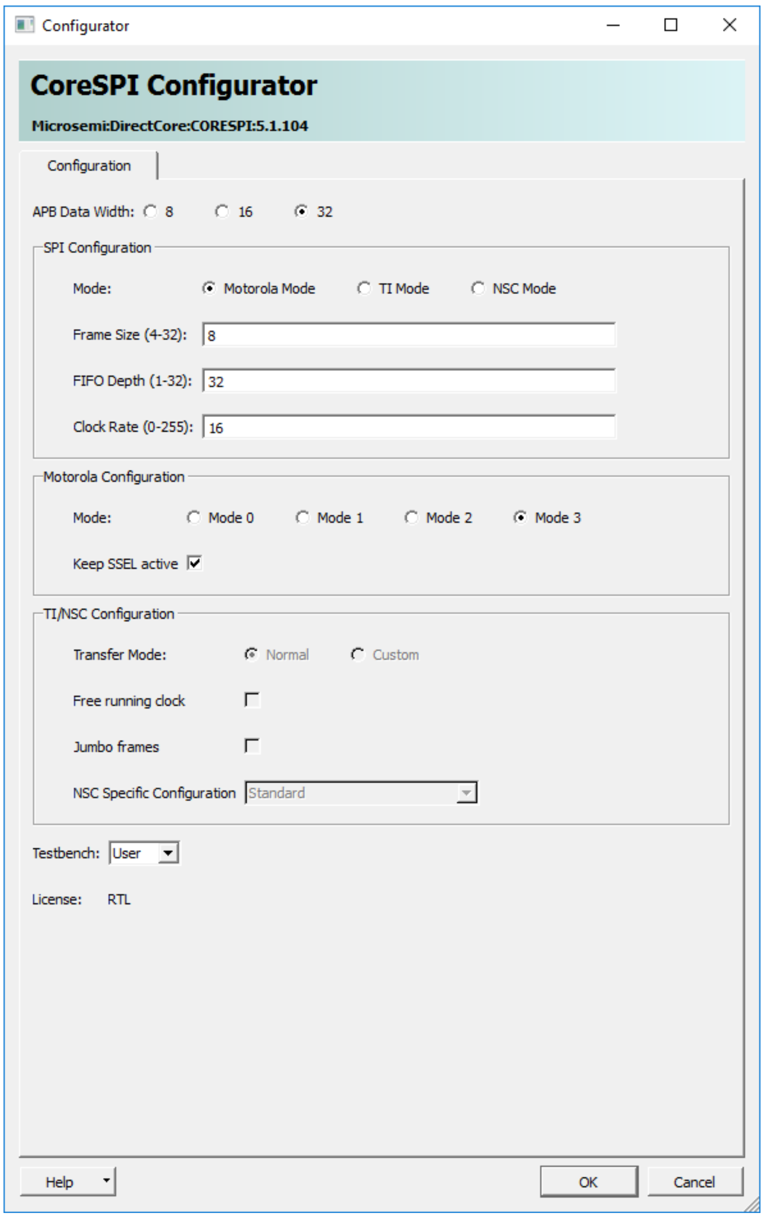
2.4.2.9 CoreSPI Configuration

The CoreSPI is used to program the external SPI flash using Mi-V processor. PF_SPI macro interfaces the fabric logic to the external SPI flash, which is connected to System Controller.

- **APB Data Width**: select 32 as APB data width in the design is 32-bit. The default value is 8.
- **Mode**: select Motorola Mode (default) as the target SPI slave (VSC Phy) supports Motorola mode.
- **Frame Size**: enter 8. The default value is 4.
- **FIFO Depth**: enter 32 to store maximum frames (Tx and Rx) in FIFO. The default value is 4.
- **Clock Rate**: enter 16. The default value is 8.
  The SPI clock becomes system clock/ 2*(16+1).
- **Keep SSEL active**: enabled to keep the slave peripheral active between back to back data transfers.

The following figure shows the CoreSPI configurator.

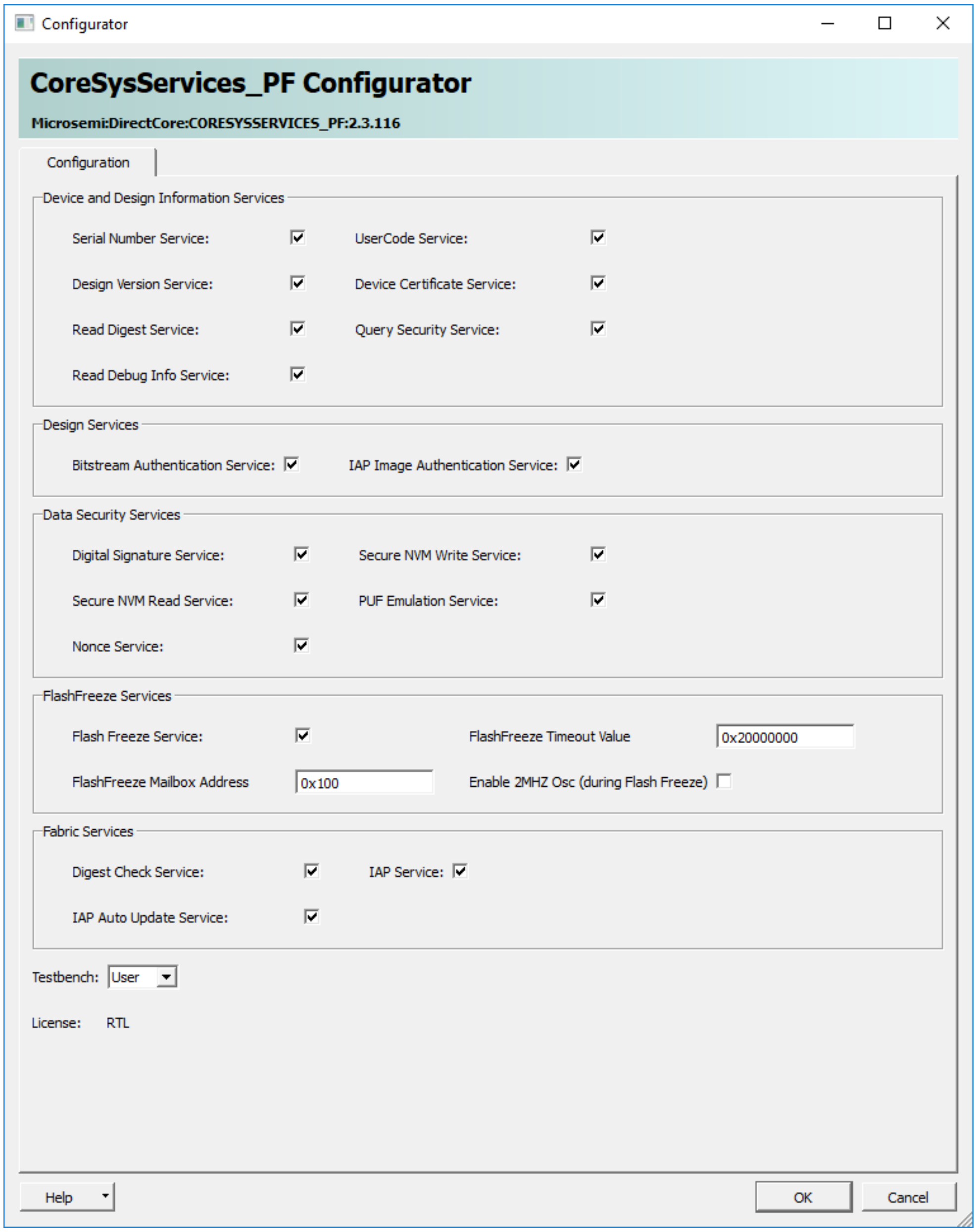
Figure 14 • **CoreSPI Configuration**



2.4.2.10 CoreSysService_PF Configuration

CoreSysServices IP provides access to the System Controller. It is connected to Mi-V soft processor as an APB slave. By default, all the service check boxes are selected. The application can initiate these selected services. CoreSysServices IP is configured as shown in the following figure.

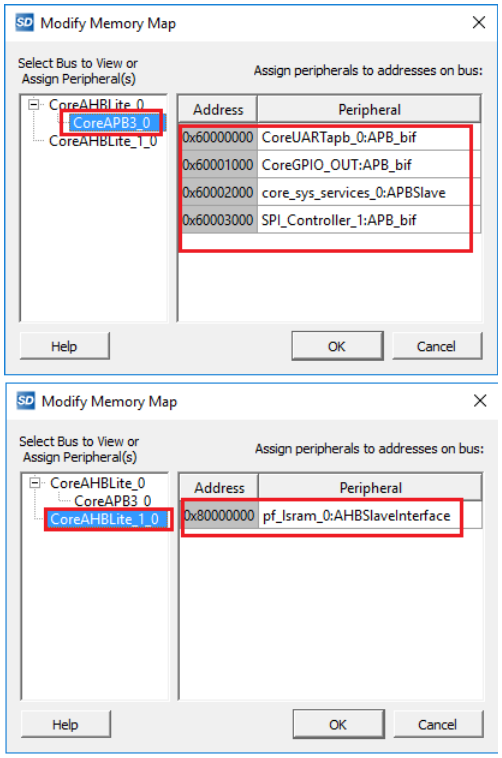
Figure 15 • **CoreSysService_PF Configuration**



2.4.2.11 Design Memory Map

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

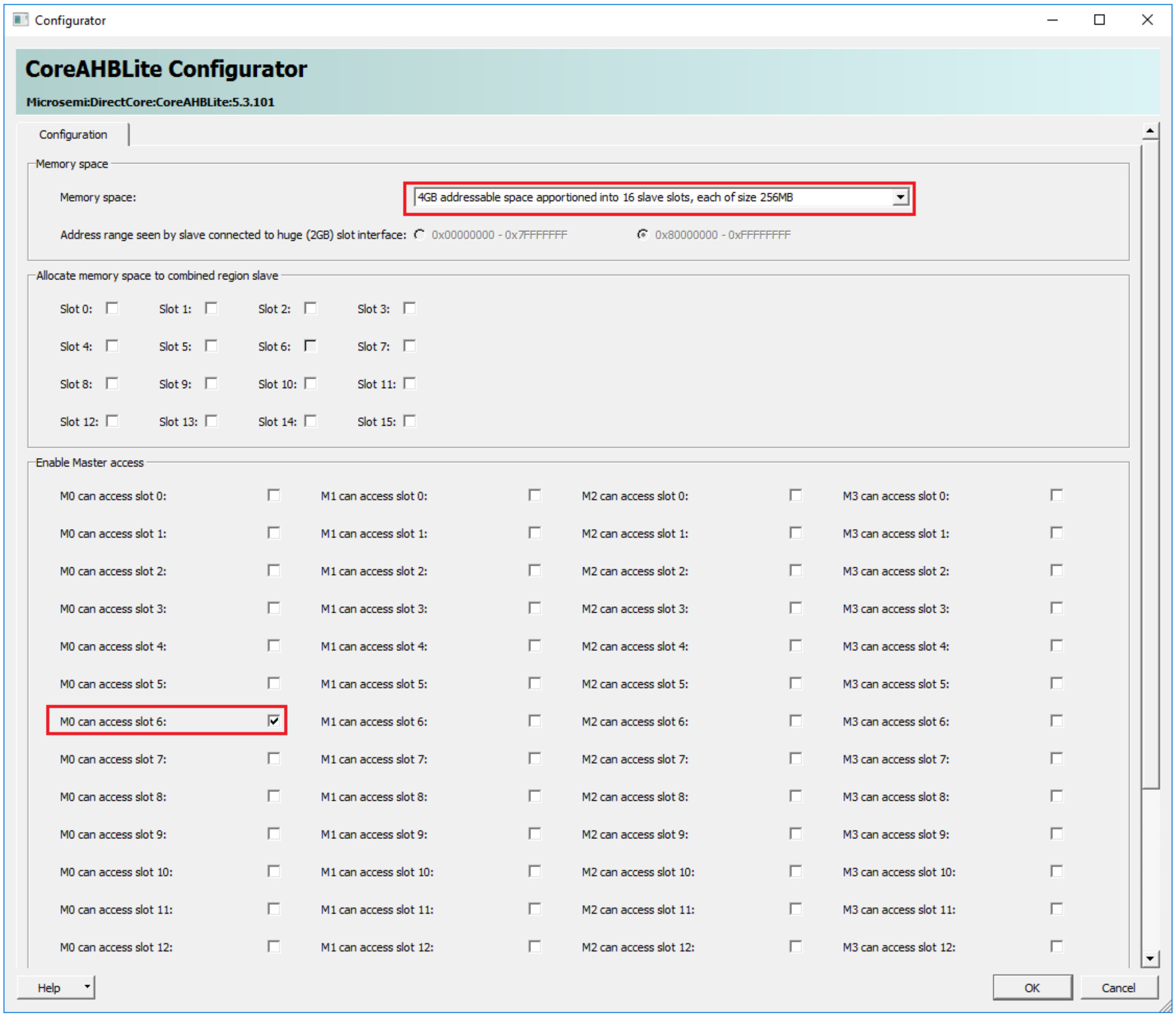
Figure 16 • **Memory Map**



2.4.2.11.1 CoreAHBLite Configuration

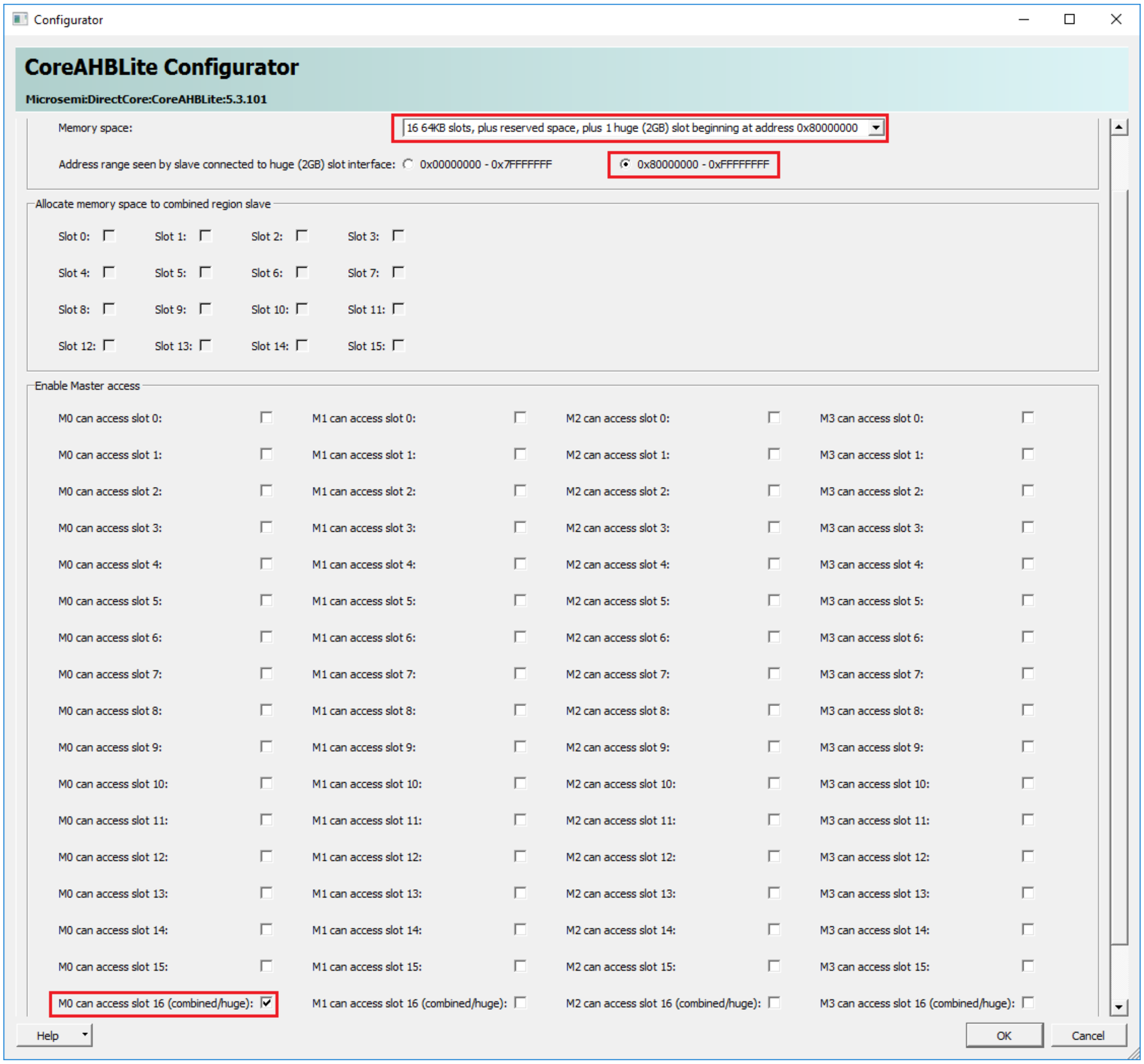
Two instances of CoreAHBLite are used in this design. The following figures show the configurations of CoreAHBLite_0 and CoreAHBLite_1 IP cores.The CoreAHBLite_0 interfaces with the APB peripherals to the Mi-V processor at 0x6000_0000.

Figure 17 • **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 18 • **CoreAHBLite_1 Configuration**



2.4.2.11.2 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.

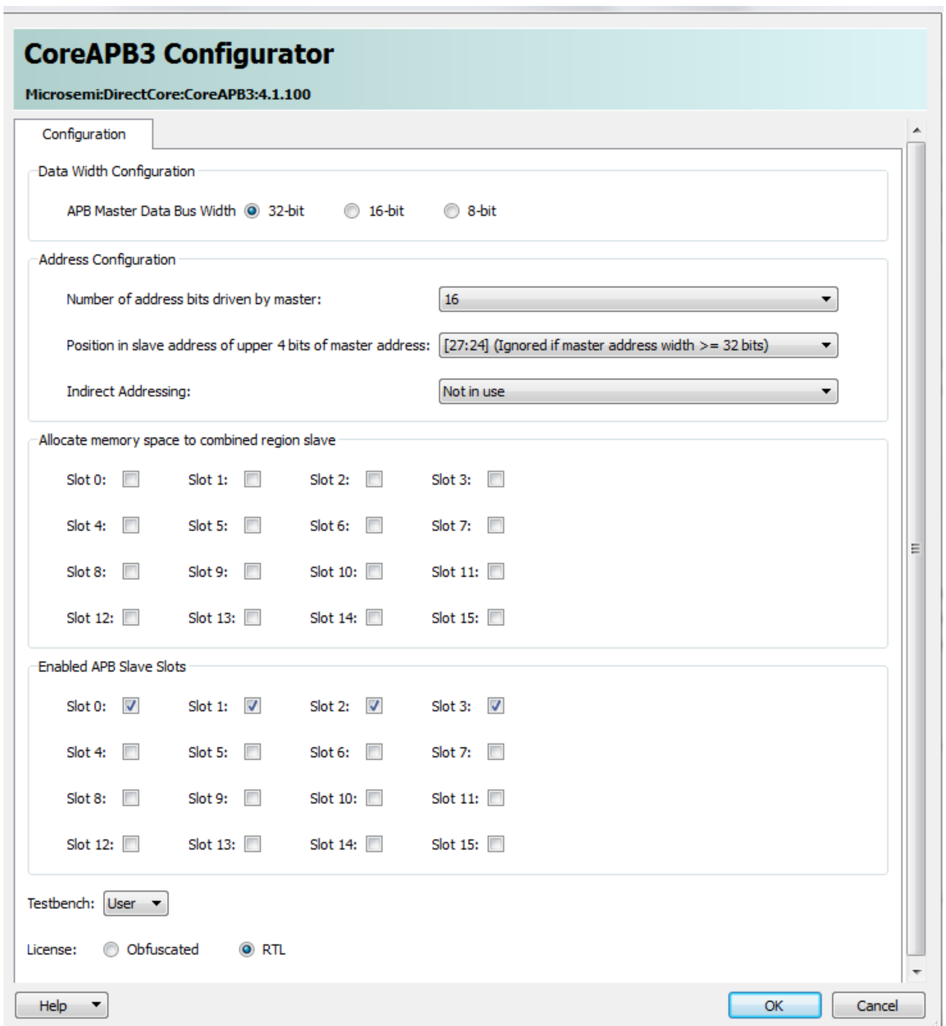
2.4.2.11.3 CoreAPB3 Configuration

The CoreAPB3 IP connects the peripherals, CoreSysService_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 CoreSysService_PF, and Slot 3 for CoreSPI.

The following figure shows the CoreAPB3 configuration.

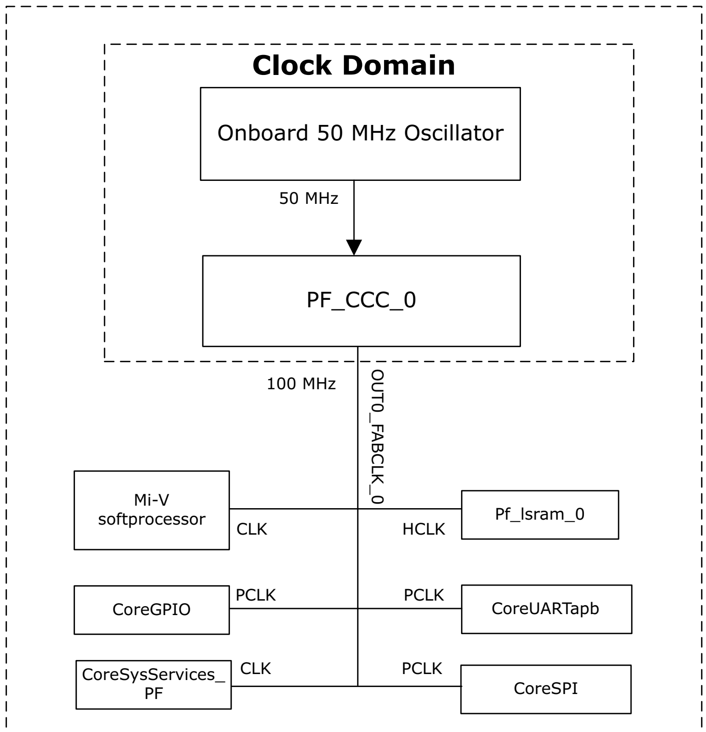
Figure 19 • **CoreAPB3_0 Configuration**



2.5 Clocking Structure

The following figure shows the clocking structure of this design. The Mi-V processor supports up to 120 MHz and this design uses 100 MHz system clock.

Figure 20 • **Clocking Structure**



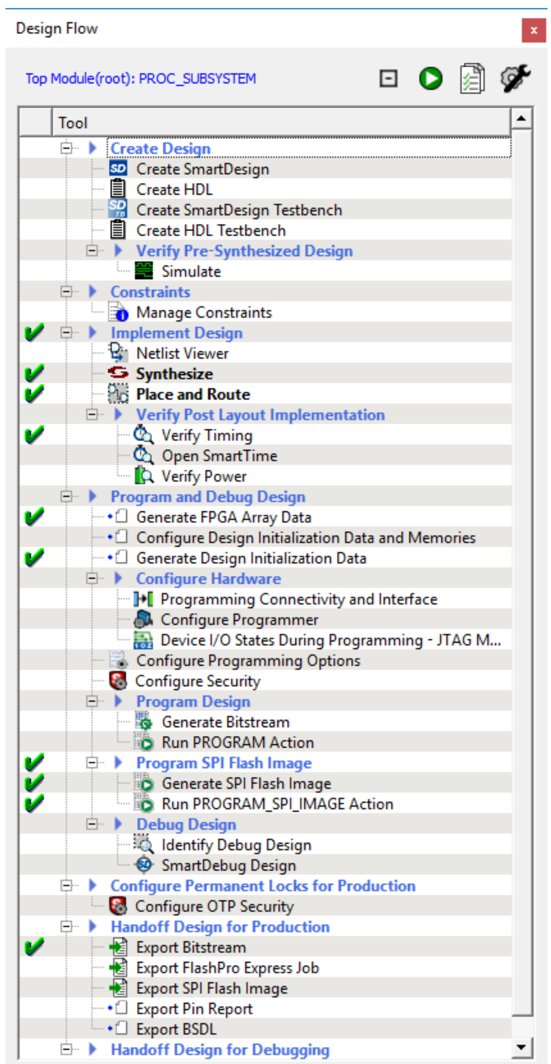
3 Libero Design Flow

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

- Synthesize, page 23
- Place and Route, page 23
- Verify Timing, page 23
- Generate FPGA Array Data, page 23
- Configure Design Initialization Data and Memories, page 24
- Configure Programming Options, page 26
- Generate Bitstream, page 27
- Run PROGRAM Action, page 27

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

Figure 21 • **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 21, page 22.
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. This 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
- JTAG_constraint.sdc file for creating the JTAG clock with 30 MHz frequency.
- The Async_Clock_groups.sdc file defines that the CCC_0 output clock and the JTAG clock as asynchronous clocks.

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 4 • **Resource Utilization**

Type	Used	Total	Percentage
4LUT	17822	299544	5.95
DFF	10918	299544	3.64
I/O Register	0	242	0.00
Logic Element	18529	299544	6.19

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 21, page 22.
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 21, page 22.

3.5 Configure Design Initialization Data and Memories

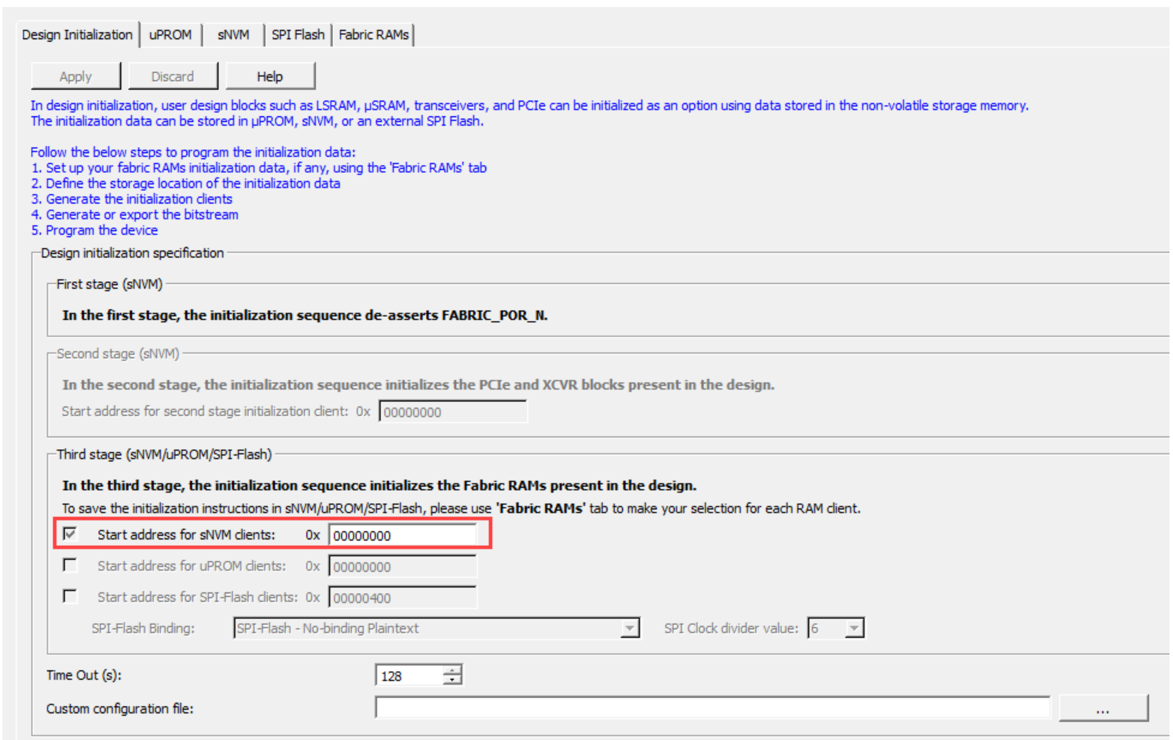
The **Configure Design Initialization Data and Memories** step generates the LSRAM initialization client and adds it to sNVM, µPROM, or an external SPI flash, based on the type of non-volatile memory selected. In this design, the LSRAM initialization client is stored in the sNVM.

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:

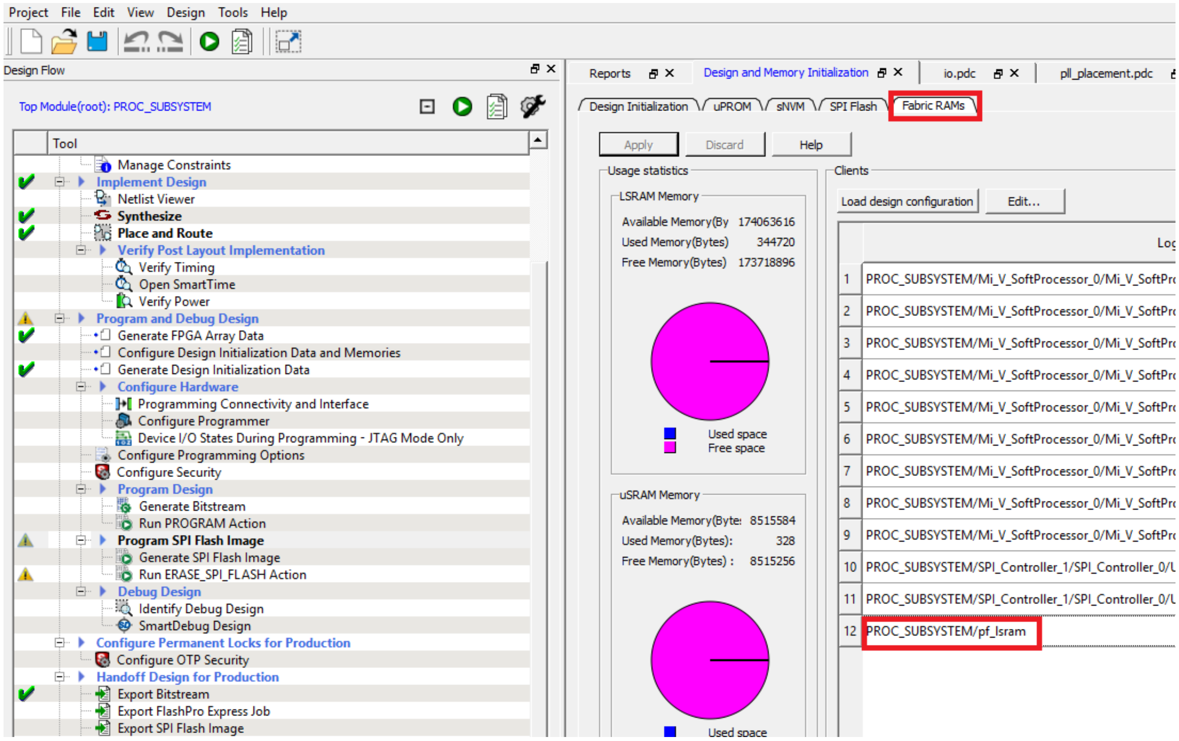
1. Double-click **Configure Design Initialization Data and Memories** from the **Design Flow** window. The **Design and Memory Initialization** window opens as shown in the following figure.

Figure 22 • **Design and Memory Initialization**



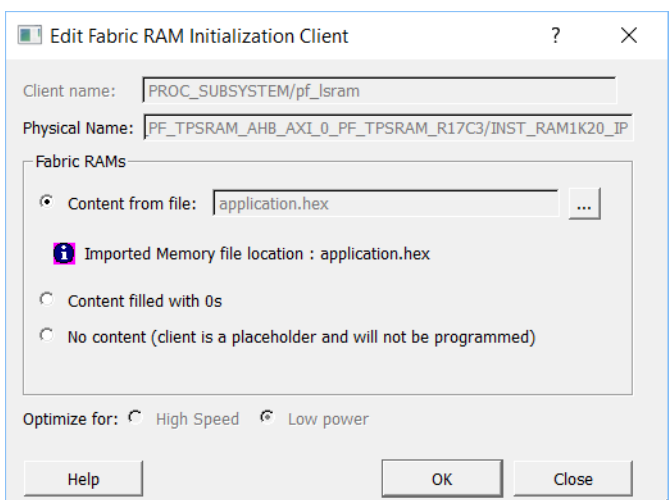
2. Select the **Fabric RAMs** tab and select the **pf_lsram** client from the list and click **Edit** as shown in the following figure.

Figure 23 • **Fabric RAMs Tab**



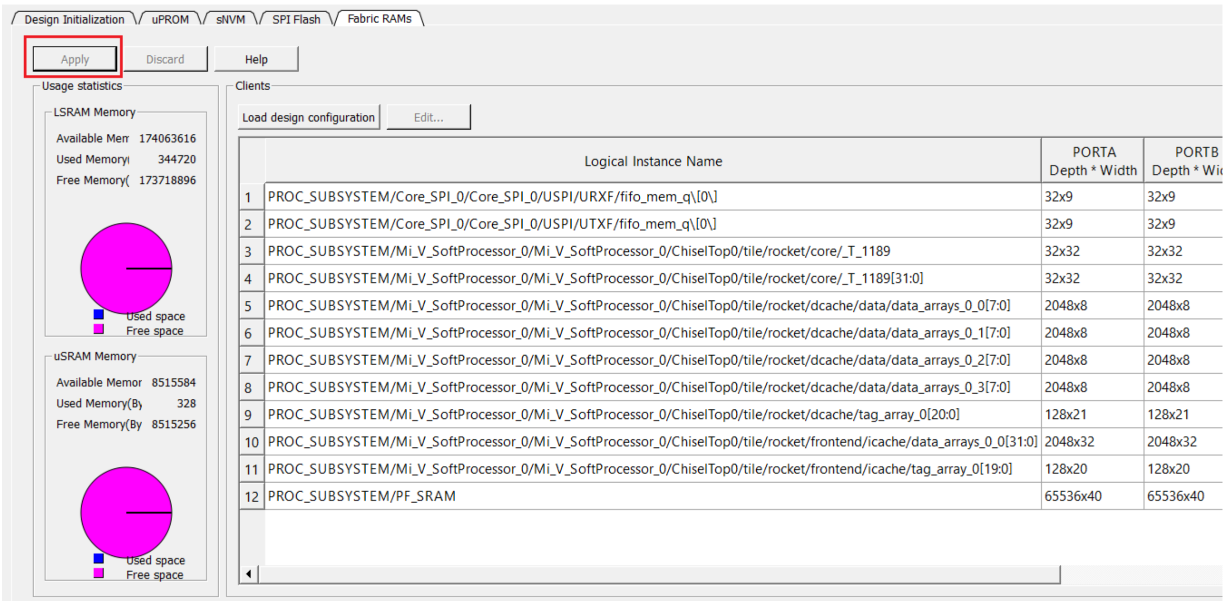
3. In the **Edit Fabric RAM Initialization Client** dialog box, select the **Content from file** option, and locate the `application.hex` file from `DesignFiles_directory\Libero_Project\hw_project` folder and Click **OK** as shown in the following figure.

Figure 24 • **Edit Fabric RAM Initialization Client**



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

Figure 25 • **Apply Fabric RAM Content**



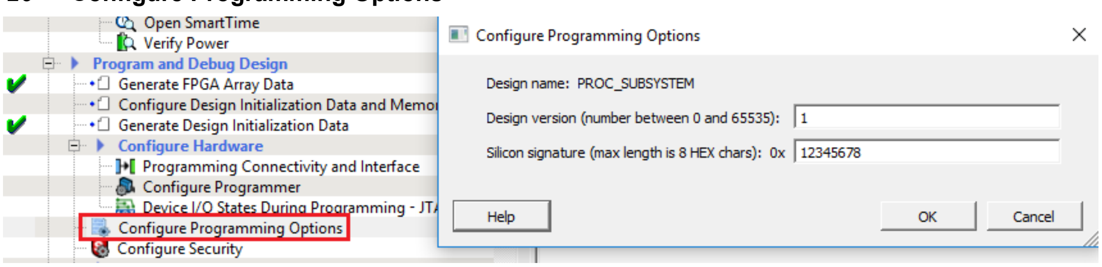
5. Click **Apply** in the **Design Initialization** tab.
6. From Libero Design Flow, click **Generate Initialization Data** to generate design initialization data.

After successful generation of the Initialization data, a green tick mark appears next to **Generate Initialization Data** option as shown in the Figure 21, page 22.

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 26 • **Configure Programming Options**

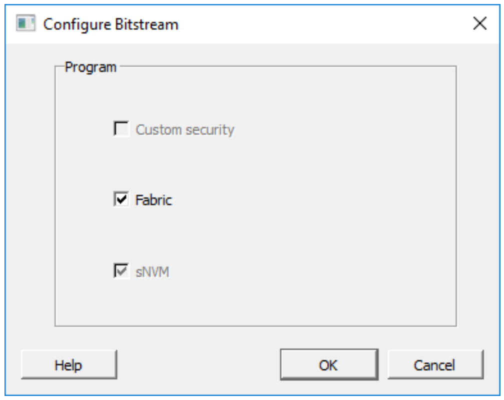


3.7 Generate Bitstream

To generate the bitstream:

1. Right-click **Generate Bitstream** and select **Configure Options...** to select the bitstream components—Custom security, Fabric, and sNVM.

Figure 27 • **Generate Bitstream—Configure Bitstream Options**



2. Double-click **Generate Bitstream** from the **Design Flow** tab. When the bitstream is successfully generated, a green tick mark appears as shown in Figure 21, page 22
3. 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 Auto Update and IAP design.

Follow these steps to program the PolarFire device:

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

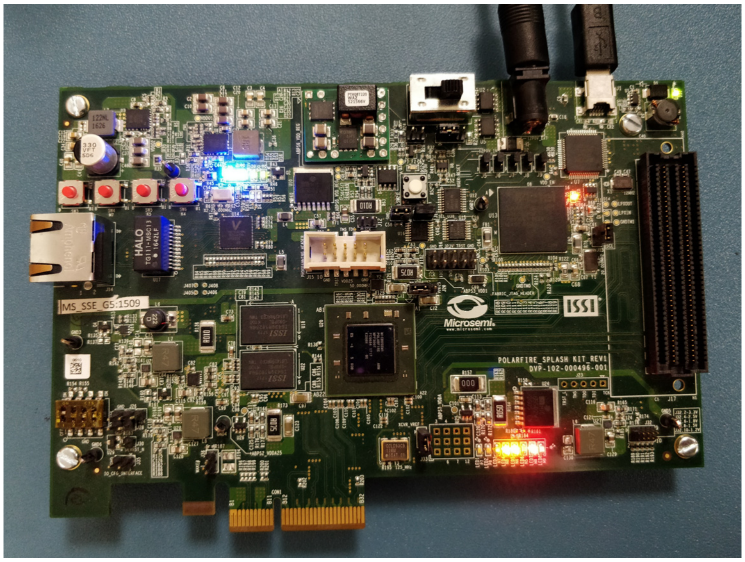
Table 5 • **Jumper Settings for PolarFire Device Programming**

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 on-board LEDs glow. A green tick mark appears next to **Run PROGRAM Action** as shown in Figure 21, page 22.

4 Programming the Device Using FlashPro Software

This section describes how to program the PolarFire device with the .stp programming file using FlashPro. The .stp file is available at the following design files folder location:

```
mpf_ac471_liberosocpolarfirev2p2_df\Programming_File
```

To program the PolarFire device using FlashPro, complete the following steps:

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

Note: The power supply switch must be switched off while making the jumper connections.

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.
5. On the host PC, launch the FlashPro software.
6. Click **New Project** to create a new project.
   In the New Project window, enter a project name.
7. Click **Browse** and navigate to the location where you want to save the project.
8. Select **Single device** as the programming mode and click **OK** to save the project.
9. Click **Configure Device**.
10. Click **Browse**, and select the progamming_appnote_v1.stp file from the following folder:
    <$design file directory>\mpf_ac471_liberosocpolarfirev2p2_df\Programming_File
11. Click **Open**. The required programming file is selected and ready to be programmed in the device.
12. Click **PROGRAM** to program the device.

When the device is programmed successfully, a **Run PASSED** status is displayed.

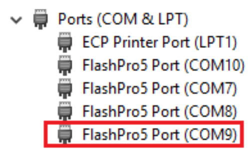
5 Serial Terminal Emulation Program Setup

The user application receives programming 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:

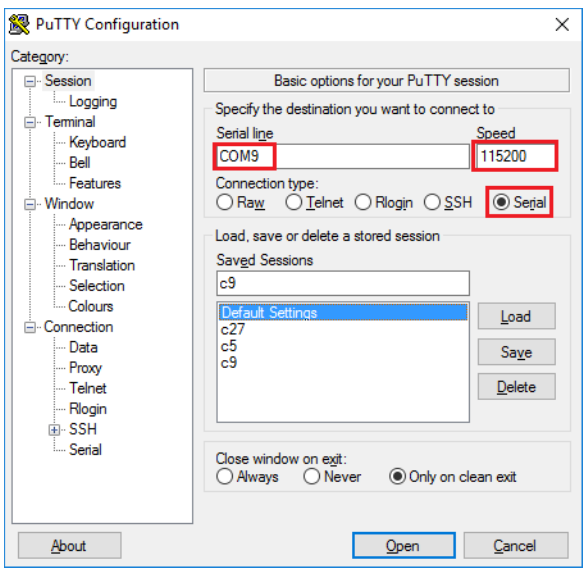
1. Connect the USB cable from the host PC to the **J1** (USB) port on the board.
2. Connect the power supply cable to the **J2** connector on the board.
3. Power on the board using the **SW1** slide switch.
4. From the host PC, click Start and open **Device Manager** to note the second highest COM Port number and use that in the PuTTY configuration. In this example, COM Port 9 (COM9) is selected as shown in the following figure. COM Port-numbers may vary.

Figure 29 • **COM Port Number**



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.

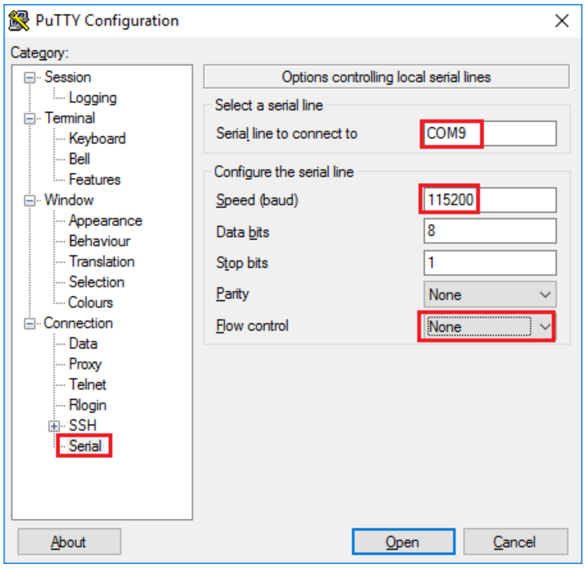
Figure 30 • **Select Serial as the Connection Type**



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.

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

Figure 31 • **PuTTY Configuration**



PuTTY opens successfully, and this completes the serial terminal emulation program setup. See Running the Demo, page 32.

6 Running the Demo

This section describes how to run the authentication, auto update and IAP. The following procedure assumes that the serial terminal is setup, for more information about setting up the serial terminal, see Serial Terminal Emulation Program Setup, page 30.

The on-board 1 GB Micron SPI flash device is connected to System Controller SPI and can be programmed using the fabric logic or Libero SoC PolarFire software. For more information about programming the on-board SPI flash using Libero, see Appendix: Programming On-board SPI Flash Using Libero, page 37.

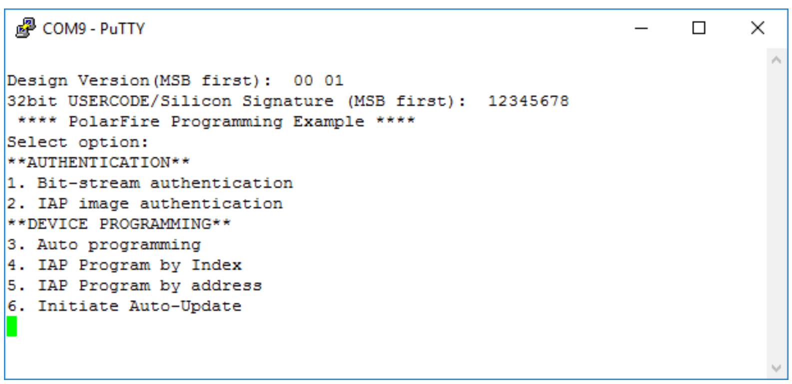
Before you start:

1. Ensure that the device is programmed with the `programming_appnote_v1.stp` file.
2. Connect the power supply cable to the **J2** connector on the board.
3. Connect the USB cable from the host PC to **J1** (FTDI port) on the board.
4. Ensure that on-board **SW8** DIP 1 is set to Off.
5. Power-up the board using the **SW1** slide switch.

6.1 Programming the SPI Flash Using Fabric Logic

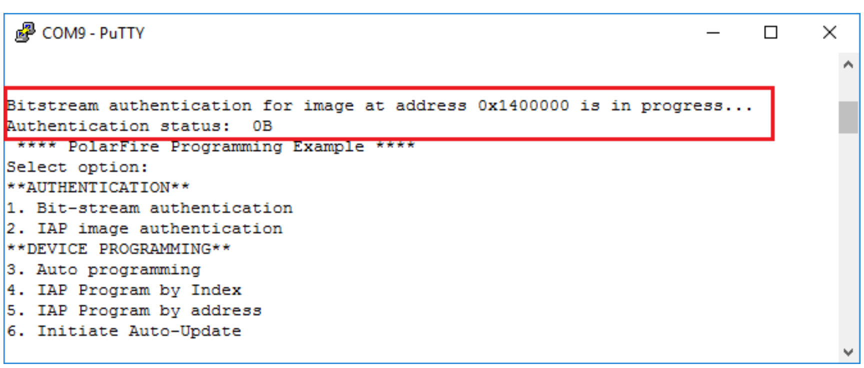
After power-up, PuTTY displays the options as shown in the following figure. Observe the design version **01** in the device.

Figure 32 • **Authentication and Programming Options**



At this point, the on-board SPI Flash device is empty. Hence, selecting Option 1 or 2 returns unsuccessful status codes as shown in the following figure.

Figure 33 • **Authentication Error**



Selecting option 4, 5, or 6 does not initiate any program operation as the on-board SPI flash is empty. Power cycle the board. Observe the design version **01** in the device. This indicates auto update is not initiated and the device is not updated.

To program the SPI flash:

1. Power off the board using the **SW1** slide switch. Close the PuTTY and set the on-board **SW8** DIP 1 to On.
2. Disconnect and connect the USB cable from the host PC to **J1** (FTDI port) on the board. This ensures clearing off UART buffers.
3. Power on the board using the **SW1** slide switch.
4. Locate the `load_spi_flash.bat` batch file from the `$DesignFiles_Folder\host_pc_tool_pf` folder.
5. Right-click `load_spi_flash.bat` batch file and edit it as follows to match the COM port number. For example, COM Port 9 in this instance.

```
spi_loader.exe 54 golden_image_v0.spi update_image_v2.spi iap_image_v5.spi
```

6. Double-click the `load_spi_flash.bat` file to load the programming images—listed in the following table—into external SPI flash.The application firmware writes the flash directory contents into the external SPI flash along with programming images.

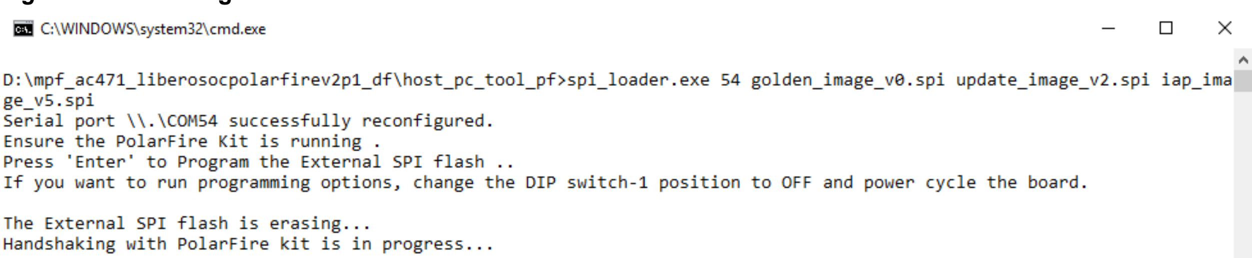
Table 7 • **Programming Images**

Image Name	Version	Silicon Signature/ User Code	Image Index in SPI Flash Directory	Image Address in SPI Flash Memory
`golden_image_v0.spi`	0	0x01234567	0	0x00000400
`update_image_v2.spi`	2	0x23456789	1	0x00A00000
`iap_image_v5.spi`	5	0x56789ABC	2	0x01400000

The command window prompts to press enter to erase and program the SPI Flash with programming images.

The LED 4 blinks to indicate that the SPI Flash Erase operation is in progress. The command prompt displays the status as shown in the following figure.

Figure 34 • **Erasing SPI Flash**



7. The SPI Flash programming operation starts and takes 20-30 minutes to complete. LED 5 blinks to indicate that the SPI Flash programming operation is in progress.
   When the SPI Flash programming operation completes successfully, LED 5 starts to glow.
   The Command prompt shows the status and the time taken as shown in the following figure.

Figure 35 • **Command Prompt Status**

```
============Begin transaction Ack 'b' is received from the target============
Requested address from the target =9527296
Requested returnbytes from the target =1296
bytes read from the file=1296
Remaining bytes =0
Sending the data to the target...............................................
End of one transaction:Ack 'a' received from target for the data from the host


start time 22:54:23

end time 23:24:28


DONE press ctrl+c to terminate the application.
_
```

8. Close the application.

This concludes programming the on-board SPI flash memory.

6.2 Running Auto Update

To run auto update:

1. Set the on-board **SW8** DIP 1 to Off.
2. Start the PuTTY and power-cycle the board. The auto update is initiated and update image (update_image_v2.spi) gets programmed into the device.

Observe the design version **02** as shown in the following figure.

Figure 36 • **Auto Update**

```
COM9 - PuTTY

Design Version(MSB first): 00 02
32bit USERCODE/Silicon Signature (MSB first):  23456789
 **** PolarFire Programming Example ****
Select option:
**AUTHENTICATION**
1. Bit-stream authentication
2. IAP image authentication
**DEVICE PROGRAMMING**
3. Auto programming
4. IAP Program by Index
5. IAP Program by address
6. Initiate Auto-Update
```

6.3 Running Authentication

To run bitstream authentication:

1. Press 1 to initiate the bitstream authentication.

After successful authentication, PuTTY displays the status code as shown in the following figure.

Figure 37 • **Successful Bitstream Authentication**

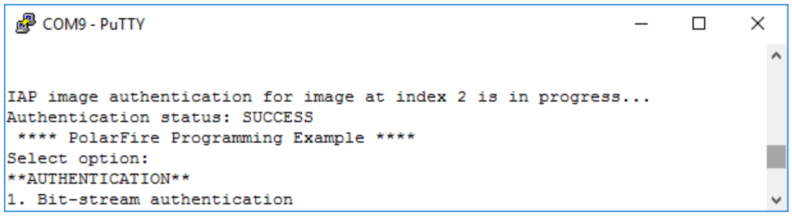
```
COM9 - PuTTY

Bitstream authentication for image at address 0x1400000 is in progress...
Authentication status: SUCCESS
 **** PolarFire Programming Example ****
Select option:
**AUTHENTICATION**
1. Bit-stream authentication
2. IAP image authentication
```

2. Press 2 to initiate the IAP image authentication.

After successful authentication, PuTTY displays the status code, as shown in the following figure.

Figure 38 • **Successful IAP Image Authentication**



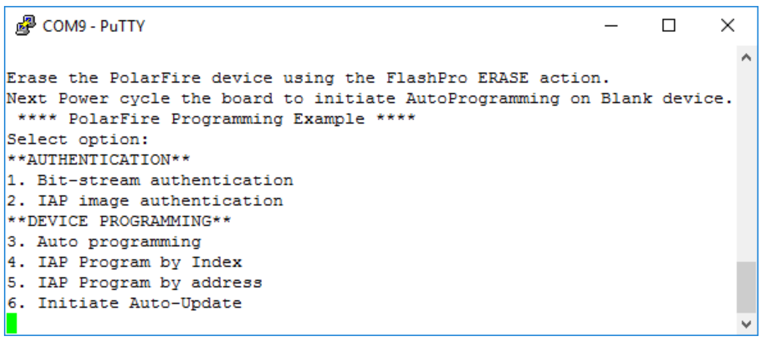
This concludes the bitstream and IAP image authentication.

6.4 Running Auto Programming

To run Auto programming:

1. Press 3 in PuTTY. The PuTTY notifies to erase the device using FlashPro and power-cycle the board as shown in the following figure.

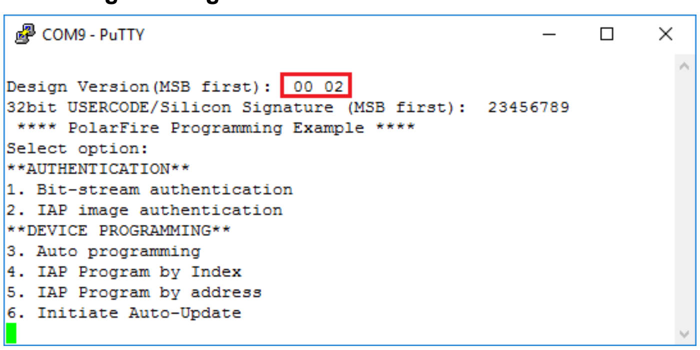
Figure 39 • **Notifying ERASE Action**



2. Using FlashPro, erase the device and power-cycle the board.
   All the LEDs stop glowing for few seconds, which indicates that the auto programming is in progress. The highest programming image version is selected from first two available images in external SPI Flash for auto programming. In this case, it is version 2 (update_image_v2.spi).

   PuTTY displays the updated design version, as shown in the following figure.

Figure 40 • **Successful Auto Programming**



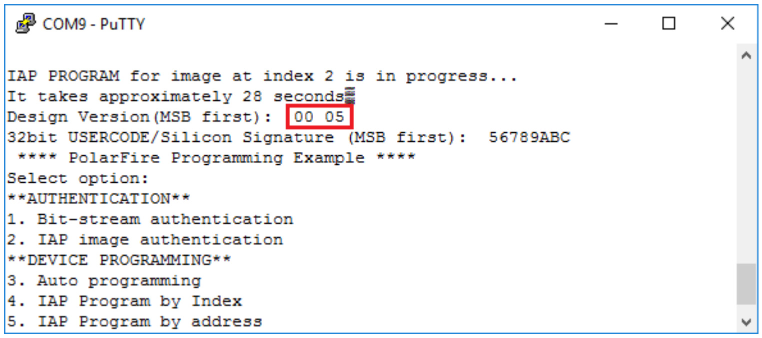
This concludes running the Auto programming feature.

6.5 Running IAP

To run IAP:

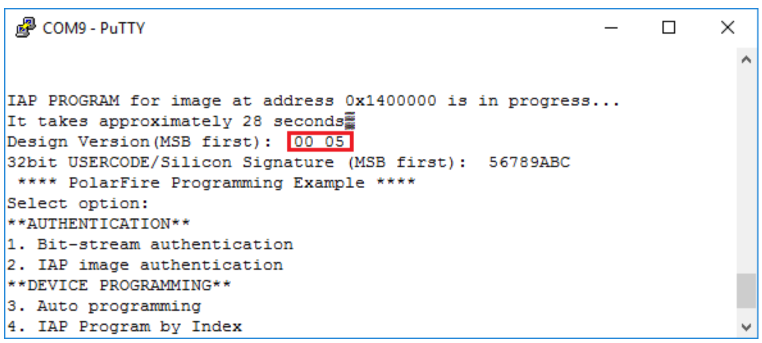
1. Press 4, IAP program by Index. After around 28 seconds, the IAP with image at index 2 is executed successfully and the design version **05** is displayed as shown in the following figure.

Figure 41 • **Successful IAP at Index 2**



2. Press 5, IAP program by address. After around 28 seconds, the IAP with image at address 0x1400000 is executed successfully and the design version **05** is displayed as shown in the following figure.

Figure 42 • **Successful IAP by Address**



This concludes running the IAP feature.

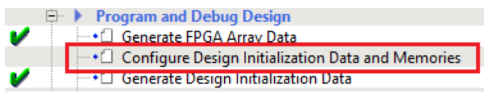
7 Appendix: Programming On-board SPI Flash Using Libero

Libero SoC PolarFire Design Suite supports the on-board SPI Flash programming using JTAG. For more information about the SPI Flash programming modes, see *UG0714: PolarFire FPGA Programming User Guide*.

To program the SPI flash using JTAG:

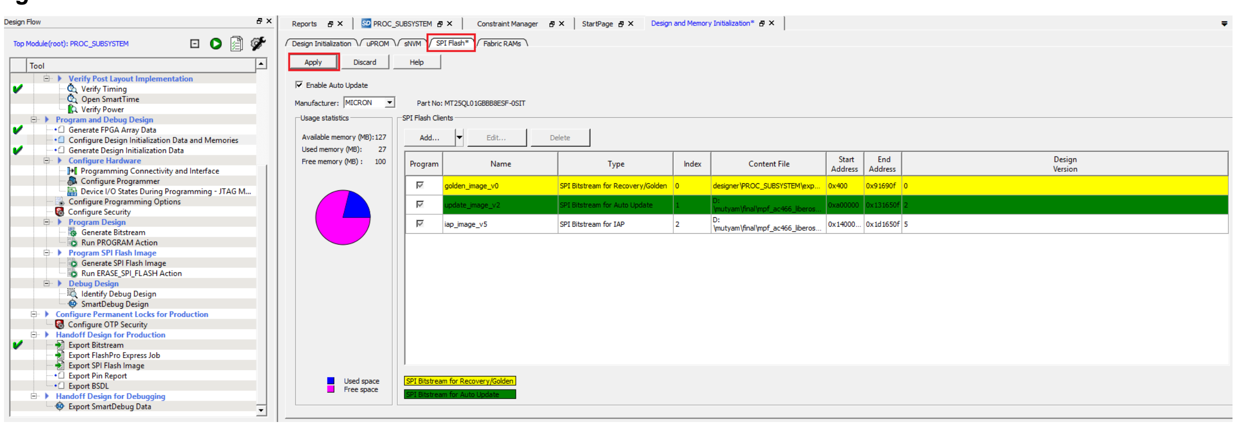
1. Ensure that the jumper settings on the board are the same as those listed in Table 5, page 27.
2. In the **Design Flow** window, select **Program and Debug Design** and then double-click **Configure Design Initialization Data and Memories**.

Figure 43 • **Configure Design Initialization Data and Memories Option**



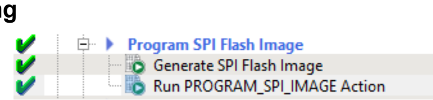
3. In the Design and Memory Initialization page, select the **SPI Flash** tab, as shown in Figure 44, page 37.
4. In SPI Flash Clients pane, add the required programming images (.spi images), and click **Apply**. These images are provided at *mpf_ac471_liberosocpolarfirev2p2_df\Libero_Project\hw_project\designer\PROC_SUBSYSTEM\export*.

Figure 44 • **SPI Flash Tab**



5. Connect the power supply cable to the **J2** connector on the board.
6. Connect the USB cable from the host PC to **J1** (FTDI port) on the board.
7. Double-click **Generate SPI Flash Image** and double-click **Run PROGRAM_SPI_IMAGE Action** to get the SPI flash programmed with the programming images as shown in the following figure.

Figure 45 • **SPI Flash Programming**



8. Power-cycle the board once you program the device.

Note: If you program the external SPI flash using Libero, set the on-board **SW8** DIP 1 to **On** because the fabric design is not required to program the SPI flash. Libero takes approximately 30 minutes to program the three programming files into SPI Flash.

This concludes the on-board SPI Flash Programming.

8 Appendix: References

This section lists documents that provide more information about programming and other IP cores used.

- For more information about PolarFire FPGA programming, see the *UG0714: PolarFire FPGA Programming 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 migration of SoftConsole v5.1 project to SoftConsole v5.2, see *AC465: Migrating a SoftConsole v5.1 Project to SoftConsole v5.2 Application Note*.
- 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 web page.