

PolarFire FPGA Auto Update and In-Application Programming Application Note

AN4660



Introduction [\(Ask a Question\)](#)

PolarFire[®] FPGAs support the Serial Peripheral Interface (SPI) Initiator 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 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 the 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 the PolarFire devices are programmable:

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

Table of Contents

Introduction.....	1
1. Overview.....	3
1.1. PF_SYSTEM_SERVICES Overview.....	3
1.2. Design Requirements.....	4
1.3. Prerequisites.....	5
1.4. Demo Design.....	5
1.5. Instantiating MiV ESS Core.....	12
1.6. Clocking Structure.....	14
2. Libero Design Flow.....	15
2.1. Synthesis.....	16
2.2. Place and Route.....	16
2.3. Verify Timing.....	17
2.4. Generate FPGA Array Data.....	17
2.5. Configure Design Initialization Data and Memories.....	17
2.6. Configure Programming Options.....	20
2.7. Generate Bitstream.....	20
2.8. Generating the SPI programming Images.....	20
2.9. Export FlashPro Express Job.....	21
2.10. Programming the Device on the Evaluation Board.....	22
3. Serial Terminal Emulation Program Setup.....	24
4. Running the Demo.....	26
4.1. Programming On-board SPI Flash Using Libero.....	27
4.2. Running Auto Update.....	28
4.3. Running Authentication.....	28
4.4. Running Auto Programming.....	29
4.5. Running IAP.....	30
5. Appendix 1: Programming On-board SPI Flash using the Fabric Logic through the Host Loader.....	31
6. Appendix 2: Programming the Device and External SPI Flash Using FlashPro Express.....	33
7. Appendix 3: Running the TCL Script.....	36
8. Appendix 4: References.....	37
9. Revision History.....	38
Microchip FPGA Support.....	39
Microchip Information.....	39
Trademarks.....	39
Legal Notice.....	39
Microchip Devices Code Protection Feature.....	40

1. Overview [\(Ask a Question\)](#)

This application note explains how to use the accompanying design to demonstrate the auto update and IAP features on the PolarFire® Evaluation board.

The on-board 1 GB Micron SPI Flash device is connected to System Controller SPI which is programmed using the fabric logic or Libero® SoC software.

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

This design is programmed using any of the following options:

- Using the pre-generated .job file: To program the device using the .job file provided along with the design, see [Appendix 2: Programming the Device and External SPI Flash Using FlashPro Express](#).
- Using Libero SoC: To program the device using Libero SoC, see [Libero Design Flow](#).



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

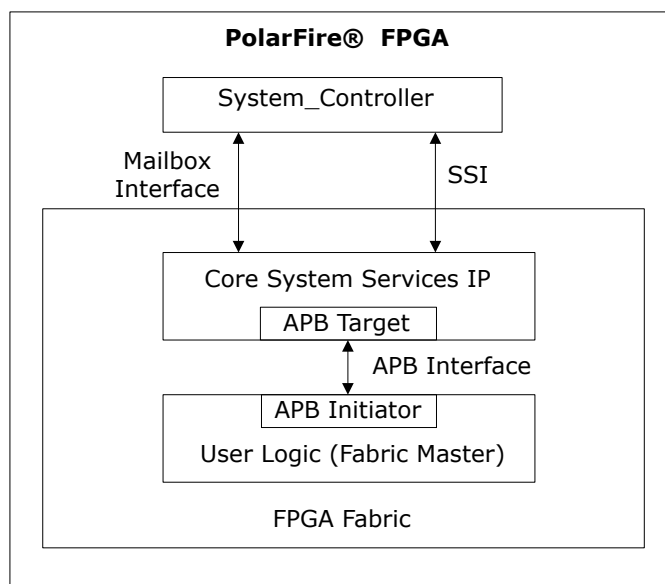
1.1. PF_SYSTEM_SERVICES Overview [\(Ask a Question\)](#)

The System Controller actions are initiated by the fabric logic through the System Service Interface (SSI) of the System Controller. The fabric logic requires PF_SYSTEM_SERVICES 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 1-1. 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 initiator to the System Controller and the status from the System Controller to the fabric initiator. The following figure shows how the PF_SYSTEM_SERVICES Interfaces with the fabric logic.

Figure 1-1. PF_SYSTEM_SERVICES IP Interfacing with Fabric User Logic

1.2. Design Requirements [\(Ask a Question\)](#)

The following table lists the hardware and software requirements for this design.

Table 1-2. Design Requirements

Requirements	Description
Hardware Requirements	
PolarFire® Evaluation Kit (MPF300TS-1FCG1152I-EVAL-KIT)	Rev D or later
Host PC	—
Software Requirements	
FlashPro Express	See the <code>readme.txt</code> file provided in the design files for all software versions needed to create this reference design.
Libero® SoC	
SoftConsole	
Serial Terminal Emulation Program	PuTTY or HyperTerminal
Operating System	Windows® 10



Important:

- Any serial terminal emulation program can be used. PuTTY is used in this application note.
- Libero SmartDesign and configuration screen shots shown in this application note are for illustration purpose only. For the latest updates, open the Libero design.

1.3. Prerequisites [\(Ask a Question\)](#)

Before you begin, ensure that the following components are in place:

1. For information about the Evaluation kit, see [PolarFire Evaluation Kit \(MPF300TS-1FCG1152I-EVAL-KIT\)](#).
2. Download the demo design files from [PolarFire FPGA Auto Update and In-Application Programming Design Files](#).
3. Download and install Libero SoC from the following link: [Libero SoC Documentation](#).

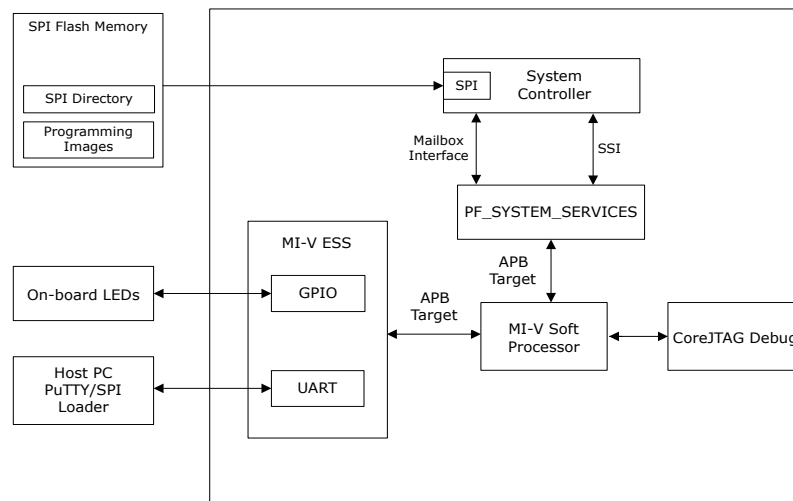
1.4. Demo Design [\(Ask a Question\)](#)

The following steps describe the data flow in the design:

1. The host PC sends the system service requests to UART in Mi-V ESS through the UART Interface.
2. The Mi-V soft processor initializes the System Controller using the PF_SYSTEM_SERVICES 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 PF_SYSTEM_SERVICES over the mailbox interface.
4. The Mi-V processor receives the service response and forwards the data to the UART interface.

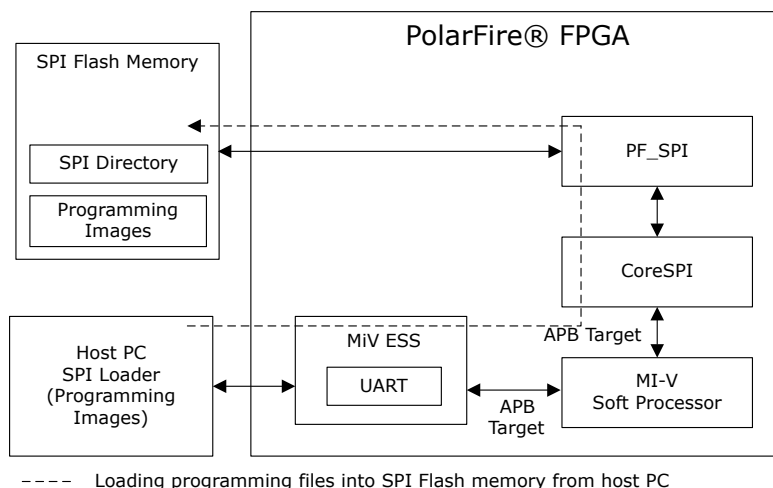
The following figure shows the block diagram of the PolarFire programming design.

Figure 1-2. PolarFire Programming Design Block Diagram

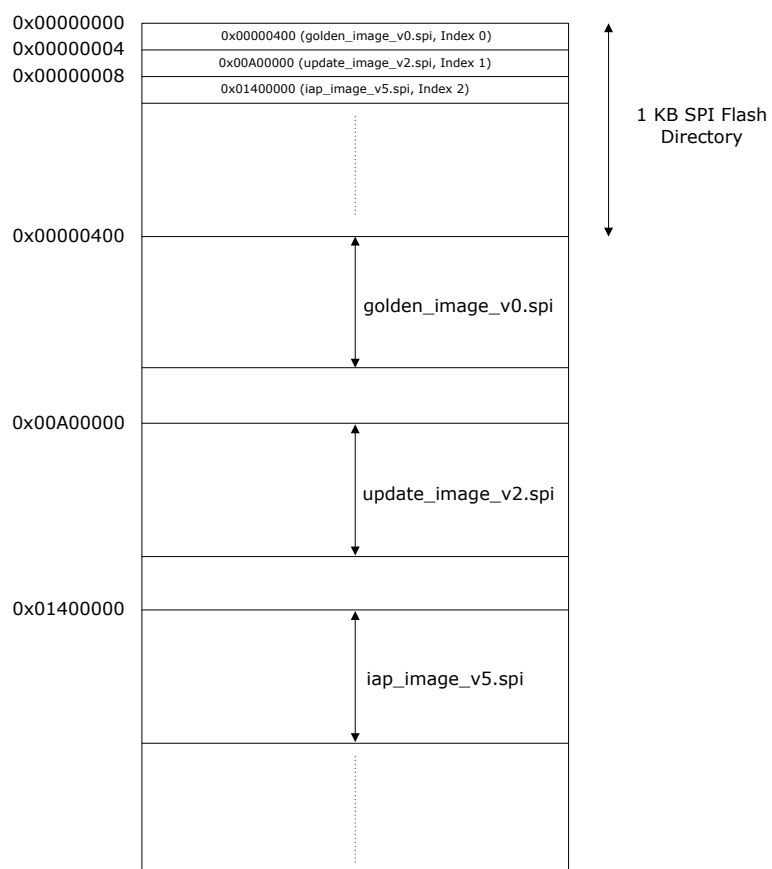


To initiate an auto update or IAP system service request, the on-board SPI Flash must be programmed with programming images. The fabric logic interfaces to the on-board SPI Flash using the SPI controller and PF_SPI macro. When the System Controller's SPI is enabled and configured as master, the System 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 the UART interface. The programming images are transferred from the host PC using SPI Flash loader (`spi_loader.exe`).

The on-board SPI Flash is programmed using fabric logic as shown in the following figure. For more information, see [Appendix 1: Programming On-board SPI Flash using the Fabric Logic through the Host Loader](#).

Figure 1-3. Accessing On-board SPI Flash Using Fabric

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

Figure 1-4. SPI Flash Memory

When the 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:

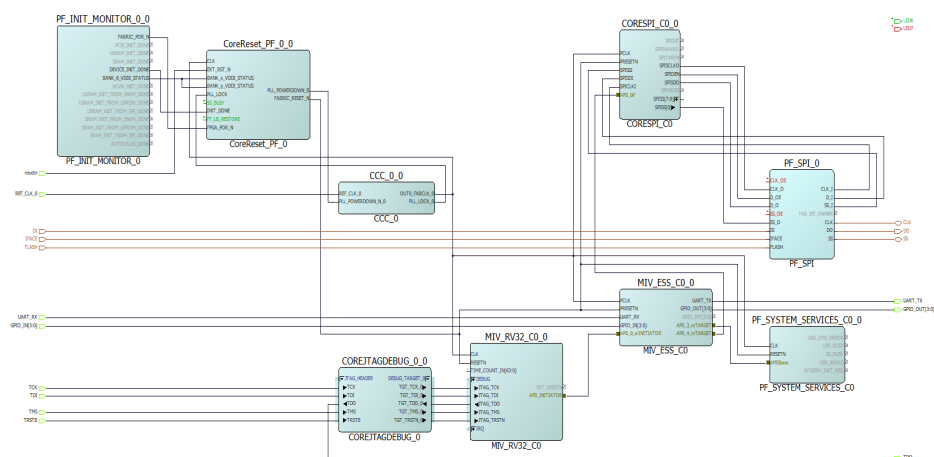
- Bitstream authentication
- IAP image authentication
- Auto update
- IAP

For more information about the preceding services, see the [PolarFire FPGA and PolarFire SoC FPGA Programming User Guide](#).

1.4.1. Design Implementation [\(Ask a Question\)](#)

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

Figure 1-5. Top Level Libero Design



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

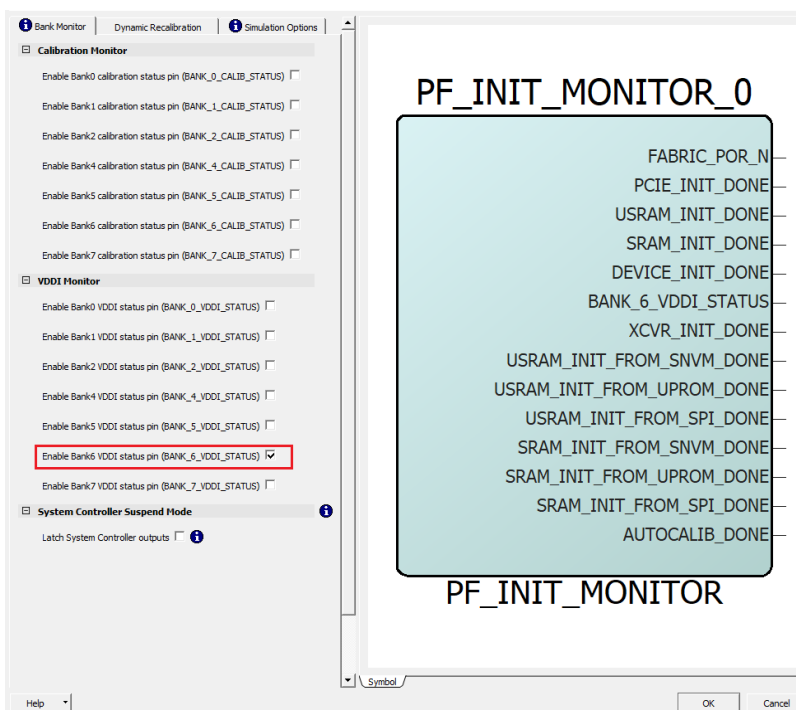
Table 1-3. Input and Output Signals

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
UART_RX	Input signals received from the serial UART terminal
UART_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

1.4.1.1. PF_INIT_MONITOR [\(Ask a Question\)](#)

The PolarFire Initialization Monitor gets the status of device initialization. In PF_INIT_MONITOR configurator, under **Bank Monitor** tab, ensure that all the calibration status pins are deselected, and then select "Enable Bank6 VDDI status pin (BANK_6_VDDI_STATUS)" as shown in the following figure.

Figure 1-6. PF_INIT_MONITOR Configuration



1.4.1.2. PF_CCC_0 Configuration [\(Ask a Question\)](#)

The PolarFire Clock Conditioning Circuitry (CCC) block takes an input clock of 50 MHz from the on-board oscillator and generates a 83.33 MHz fabric clock to the Mi-V processor subsystem and other peripherals. The following figures show the input and output clock configurations.

Figure 1-7. PF_CCC_0 Input Clock Configuration

Configuration: PLL-Single

Clock Options: PLL | Output Clocks

Input Frequency

Input Frequency: 50 MHz ☐ Backup Clock

Bandwidth: High = 0.893 MHz

Delay Line

☐ Enable Delay Line

☒ Reference Clock Delay ☐ Feedback Clock Delay

Delay Steps: 1

Power / Jitter

☒ Maximize VCO for Lowest Jitter VCO = 4800 MHz

☐ Minimize VCO for Lowest Power

Feedback Mode

Post-VCO

Features

Log

Messages ☒ Errors ☒ Warnings ☒ Info ☒

Help

OK Cancel

Symbol

PF_CCC_0

REF_CLK_0 OUT0_FABCLK_0

PLL_LOCK_0

Figure 1-8. PF_CCC_0 Output Clock Configuration

Configuration: PLL-Single

Clock Options: PLL | Output Clocks

For best results, put the highest frequency first.

Output Clock 0

☒ Enabled

Requested Frequency: 83.333 MHz ☐ Actual Lower: 83.333 MHz ☒ Actual Higher: 83.333 MHz

Requested Phase: 0 Degrees ☐ Actual Lower: 0 Degrees ☒ Actual Higher: 0 Degrees

☐ Dynamic Phase Shifting ☐ Expose Enable Port ☐ HS LO Clock ☐ Dedicated Clock

☒ Global Clock ☒ Global Clock (Sated)

Output Clock 1

☐ Enabled

Requested Frequency: 100 MHz ☐ Actual Lower: MHz ☒ Actual Higher: MHz

Requested Phase: 0 Degrees ☐ Actual Lower: Degrees ☒ Actual Higher: Degrees

☐ Dynamic Phase Shifting ☐ Expose Enable Port ☐ HS LO Clock ☐ Dedicated Clock

☒ Global Clock ☒ Global Clock (Sated)

Log

Messages ☒ Errors ☒ Warnings ☒ Info ☒

Help

OK Cancel

Symbol

PF_CCC_0

REF_CLK_0 OUT0_FABCLK_0

PLL_POWERDOWN_0 PLL_LOCK_0

1.4.1.3. Mi-V Soft Processor Configuration [\(Ask a Question\)](#)

The Mi-V soft processor default Reset Vector Address value is 0x8000_0000. After the device reset, the processor executes the application from TCM, which is mapped to 0x8000_0000. Hence, the Reset Vector Address is set to 0x8000_0000, as shown in the following figure.

TCM is the main memory of the Mi-V processor. It gets initialized with the user application from SNVM.

In the Mi-V processor memory map, the 0x8000_0000 to 0x8000_FFFF address range is defined for TCM memory interface and the 0x6000_0000 to 0x6FFF_FFFF address range is defined for APB interface.

Figure 1-9. Mi-V Configuration

The image shows a 'Mi-V Configuration' dialog box with two tabs: 'Configuration' (selected) and 'Memory Map'. The dialog is organized into several sections, each with a title bar and a list of options. The 'Extension Options' section includes checkboxes for 'C:', 'F:', and 'M:', with 'C:' and 'M:' checked, and a 'Multiplier' dropdown set to 'Fabric'. The 'Interface Options' section includes dropdowns for 'AHB Initiator' (None), 'APB Initiator' (APB3), and 'AXI Initiator' (None), along with checkboxes for 'AHB Mirrored I/F:', 'APB Mirrored I/F:', 'AXI Mirrored I/F:', 'ICACHE:', and 'Multi-Interface IM:'. The 'Reset Vector Address' section has two text boxes for 'Upper 16bits (Hex):' (0x8000) and 'Lower 16bits (Hex):' (0x0). The 'BootROM Option' section has checkboxes for 'BootROM:' and 'Reconfigurable:'. The 'Tightly Coupled Memory (TCM) Options' section has a checked 'TCM:' checkbox and an unchecked 'TCM Access Support (TAS):' checkbox. The 'Interrupt Options' section has a dropdown for 'External System IRQs:' (0) and an unchecked 'Vectored Interrupts:' checkbox. The 'Timer Options' section has checkboxes for 'Internal MTIME:', 'Internal MTIME IRQ:', and a 'MTIME Prescaler:' text box (100). The 'Debug Options' section has a checked 'Debug:' checkbox, a 'Trace Interface:' checkbox, and a 'Hart ID:' text box (0x0). The 'Performance and Reliability Options' section is currently empty. At the bottom, there is a 'Help' dropdown, 'OK', and 'Cancel' buttons.

Configuration | Memory Map

Extension Options

C: ☒ F: ☐ M: ☒ Multiplier: Fabric

Interface Options

AHB Initiator: None AHB Mirrored I/F: ☐

APB Initiator: APB3 APB Mirrored I/F: ☐

AXI Initiator: None AXI Mirrored I/F: ☐

ICACHE: ☐ Multi-Interface IM: ☐

Reset Vector Address

Upper 16bits (Hex): 0x8000 Lower 16bits (Hex): 0x0

BootROM Option

BootROM: ☐ Reconfigurable: ☐

Tightly Coupled Memory (TCM) Options

TCM: ☒ TCM Access Support (TAS): ☐

Interrupt Options

External System IRQs: 0

Vectored Interrupts: ☐

Timer Options

Internal MTIME: ☐ MTIME Prescaler: 100

Internal MTIME IRQ: ☐

Debug Options

Debug: ☒ Trace Interface: ☐ Hart ID: 0x0

Performance and Reliability Options

Help OK Cancel

Memory depth: This field is set to 16384 words to accommodate an application of up to 64 KB into TCM. The present application is below 50 KB, so this fits into either sNVM or μ PROM. In this design, sNVM is selected as data storage client, as shown in the following figure.

Figure 1-10. Mi-V RV32 Configuration

The screenshot shows the 'Configuration' tab of the Mi-V RV32 Configuration dialog. The fields are as follows:

- AHB Initiator Address:** Start Address: Upper 16bits (Hex): 0x8000, Lower 16bits (Hex): 0x0; End Address: Upper 16bits (Hex): 0x8fff, Lower 16bits (Hex): 0xffff.
- APB Initiator Address:** Start Address: Upper 16bits (Hex): 0x6000, Lower 16bits (Hex): 0x0; End Address: Upper 16bits (Hex): 0x6fff, Lower 16bits (Hex): 0xffff.
- AXI Initiator Address:** Start Address: Upper 16bits (Hex): 0x6000, Lower 16bits (Hex): 0x0; End Address: Upper 16bits (Hex): 0x6fff, Lower 16bits (Hex): 0xffff.
- TCM Address:** Start Address: Upper 16bits (Hex): 0x8000, Lower 16bits (Hex): 0x0; End Address: Upper 16bits (Hex): 0x8000, Lower 16bits (Hex): 0xffff.
- TCM Access Support (TAS) Address:** Start Address: Upper 16bits (Hex): 0x4000, Lower 16bits (Hex): 0x0; End Address: Upper 16bits (Hex): 0x4000, Lower 16bits (Hex): 0x3fff.
- BootROM Address:** Source Start Address: Upper 16bits (Hex): 0x8000, Lower 16bits (Hex): 0x0; Source End Address: Upper 16bits (Hex): 0x8000, Lower 16bits (Hex): 0x3fff; Destination Address: Upper 16bits (Hex): 0x4000, Lower 16bits (Hex): 0x0.

1.4.1.4. CoreSPI Configuration [\(Ask a Question\)](#)

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. The default value is 8.
- **Mode:** Select Motorola Mode (default) as the target SPI target supports Motorola mode. **Mode 3** is selected under **Motorola Configuration**.
- **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 target peripheral active between back-to-back data transfers.

The following figure shows the CoreSPI configurator.

Figure 1-11. CoreSPI Configuration

The image shows the 'Configuration' dialog box for CoreSPI. It has three main sections: SPI Configuration, Motorola Configuration, and TI/NSC Configuration. At the bottom are Testbench and License settings, and buttons for Help, OK, and Cancel.

- SPI Configuration:**
 - APB Data Width: ☐ 8 ☐ 16 ☒ 32
 - Mode: ☒ Motorola Mode ☐ TI Mode ☐ NSC Mode
 - Frame Size (4-32):
 - FIFO Depth (1-32):
 - Clock Rate (0-255):
- Motorola Configuration:**
 - Mode: ☐ Mode 0 ☐ Mode 1 ☐ Mode 2 ☒ Mode 3
 - Keep SSEL active: ☒
- TI/NSC Configuration:**
 - Transfer Mode: ☒ Normal ☐ Custom
 - Free running clock: ☐
 - Jumbo frames: ☐
 - NSC Specific Configuration:
- Testbench:**
- License:**

1.5. Instantiating MiV ESS Core [\(Ask a Question\)](#)

To instantiate the MiV ESS core, perform the following steps:

1. From the **Catalog**, drag the **MiV ESS IP** core to **SmartDesign**.
2. In the **Create Component** dialog box, enter MIV_ESS_C0 as the component name, and click **OK**.
3. In the **MiV ESS Configurator** screen, perform the following configurations:
 - a. Navigate to **General** tab, and make sure that the configurations are same as shown in the following figure.
 - i. Deselect **Bootstrap**
 - ii. Select the following peripherals: **GPIO** and **UART** and deselect all others.

Figure 1-12. General Configurator

The image shows the 'General' tab of the MiV ESS Configurator. It includes tabs for General, Bootstrap, APB, uDMA, GPIO, PLIC, SPI, Timer, and UART. The 'Family' section shows 'PolarFire'. The 'Bootstrap' section has 'Bootstrap' unchecked and 'Bootstrap Source' set to 'SPI'. The 'Peripherals' section shows 'GPIO' and 'UART' checked, while others are unchecked.

- Family:**
 - FPGA Family:
- Bootstrap:**
 - Bootstrap: ☐
 - Bootstrap Source:
- Peripherals:**
 - uDMA: ☐
 - GPIO: ☒
 - I2C: ☐
 - PLIC: ☐
 - SPI: ☐
 - Timer: ☐
 - UART: ☒
 - Watchdog: ☐

- b. Navigate to **APB** tab, and make sure that the configurations are same as shown in the following figure.
 - i. Select **APB Mirrored I/F**
 - ii. External APB Target: **Slot 3** and **Slot 4**

Figure 1-13. APB Configurator

The screenshot shows the APB Configurator interface. The 'General' tab is selected. Under 'External APB Initiator', the 'APB Mirrored I/F' checkbox is checked. Under 'External APB Target', the 'Slot 3' and 'Slot 4' checkboxes are checked, while 'Slot 11', 'Slot 12', 'Slot 13', 'Slot 14', and 'Slot 15' are unchecked.

- c. Navigate to **GPIO** tab, and make sure that the configurations are same as shown in the following figure.

Figure 1-14. GPIO Configurator

The screenshot shows the GPIO Configurator interface. The 'GPIO' tab is selected. Under 'Global Configuration', 'APB Data Width' is 32, 'Number of I/Os' is 4, 'Single-bit interrupt port' is Disabled, and 'Output enable' is Internal. A red box highlights the I/O bit configurations for bits 0 through 4. Bit 0 is Output, Bit 1 is Output, Bit 2 is Input, Bit 3 is Input, and Bit 4 is Input. Bits 5 through 8 are also Input.

I/O bit	Output on Reset	Fixed Config	I/O Type	Interrupt Type
I/O bit 0	0	✓	Output	Disabled
I/O bit 1	0	✓	Output	Disabled
I/O bit 2	0	✓	Input	Disabled
I/O bit 3	0	✓	Input	Disabled
I/O bit 4	0	✗	Input	Disabled
I/O bit 5	0	✗	Input	Disabled
I/O bit 6	0	✗	Input	Disabled
I/O bit 7	0	✗	Input	Disabled
I/O bit 8	0	✗	Input	Disabled

- d. Navigate to **UART** tab, and make sure that the configurations are same as shown in the following figure.

Figure 1-15. UART Configurator

General | Bootstrap | APB | **UART** | GPIO | PLIC | SPI | Timer

Core Configuration

TX FIFO:

RX FIFO:

Configuration:

Baud Value:

Character Size:

Parity:

RX Legacy Mode:

FIFO Implementation:

Status Flags: ☐

Baud Value Precision

Enable Extra Precision: ☐

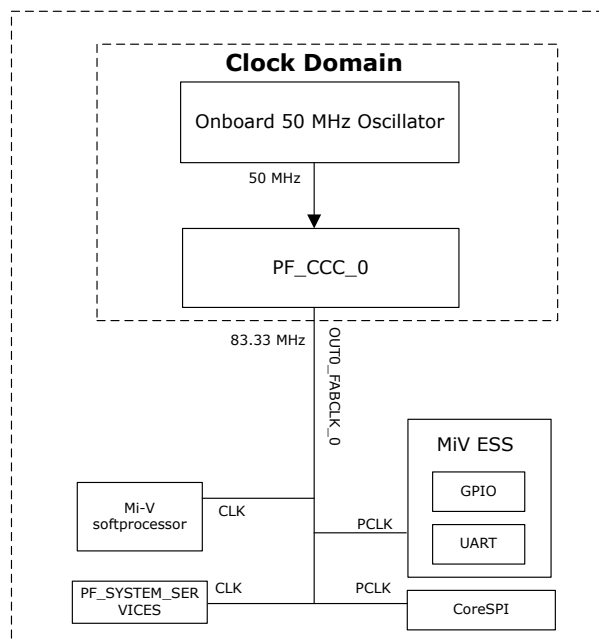
Fractional Part of Baud Value:

4. To generate the component, click **OK**.

1.6. Clocking Structure [\(Ask a Question\)](#)

The following figure shows the clocking structure of this design. This design uses a 83.33 MHz system clock for configuring the APB peripherals.

Figure 1-16. Clocking Structure



2. Libero Design Flow [\(Ask a Question\)](#)

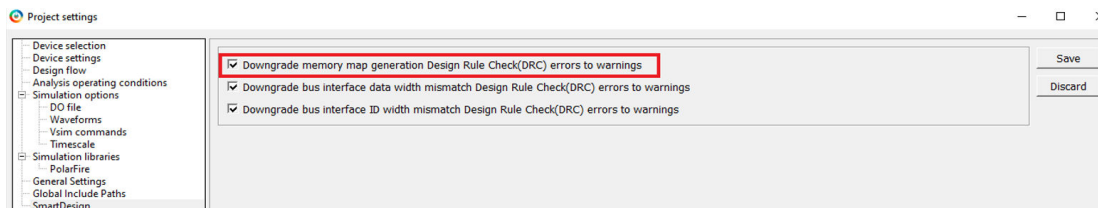
The Libero design flow involves running the following processes in the Libero SoC.



Important:

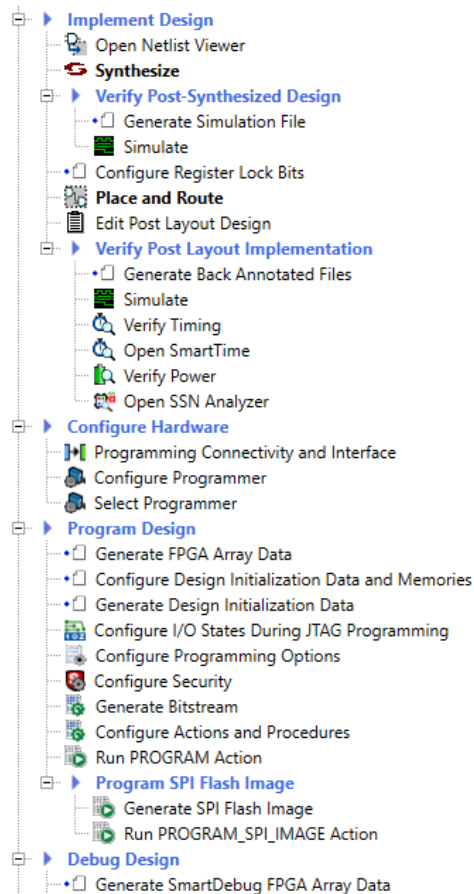
- To initialize the TCM in PolarFire using the System Controller, a local parameter `l_cfg_hard_tcm0_en`, in the `miv_rv32_subsys_pkg.v` file must be changed to `1'b1` prior to synthesis. For more information, see the 2.7 TCM section in the [MIV_RV32 Handbook](#). This user guide can be downloaded from the Libero SoC Catalog.
- In case any error is observed related to address space access issue, ensure to perform this step: Navigate to **Project > Project settings > SmartDesign**, and then Enable "**Downgrade memory map generation DRC errors to warnings**" as shown in the following figure.

Figure 2-1. Project Settings



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

Figure 2-2. Libero Design Flow Options



2.1. Synthesis [\(Ask a Question\)](#)

To synthesize the design, perform the following steps:

1. In the **Design Flow** window, double-click **Synthesis**.
When the synthesis is successful, a green tick mark appears as shown in [Figure 2-2](#).
2. To view the synthesis report and log files on the **Reports** tab, right-click **Synthesis** and click **View Report**.



Important:

- Set the correct tool profile before you start Synthesis.
- `top.srr` and the `top_compile_netlist.log` files are recommended to be viewed for debugging synthesis and compile errors.

2.2. Place and Route [\(Ask a Question\)](#)

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

- The `io_constraints.pdc` file for the I/O assignments
- The `top_derived_constraints.sdc` file for timing constraints

- The `timing_user_constraints.sdc` file for creating the JTAG clock with 30 MHz frequency

To Place and Route, perform the following step:

- In the **Design Flow** window, double-click **Place and Route**.
When place and route is successful, a green tick mark appears next to Place and Route.



Important: The `top_place_and_route_constraint_coverage.xml` file is recommended to be viewed for place and route constraint coverage.

2.2.1. Resource Utilization [\(Ask a Question\)](#)

The resource utilization report is written to the `top_layout_log.log` file in the **Reports tab > Top 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 2-1. Resource Utilization—Evaluation Board

Type	Used	Total	Percentage
4LUT	10463	299544	3.49
DFF	3895	299544	1.30
I/O Register	0	510	0.00
Logic Element	10463	299544	3.49

2.3. Verify Timing [\(Ask a Question\)](#)

To verify timing, perform the following steps:

1. In the **Design Flow** window, double-click **Verify Timing**.
When the design successfully meets the timing requirements, a green tick mark appears, see [Figure 2-2](#).
2. To view the verify timing report and log files on the **Reports** tab, right-click **Verify Timing**, and then click **View Report**.

2.4. Generate FPGA Array Data [\(Ask a Question\)](#)

To generate the FPGA array data, perform the following steps:

1. In the **Design Flow** window, double-click **Generate FPGA Array Data**.
2. A green tick mark is displayed after the successful generation of the FPGA array data. See [Figure 2-2](#).

2.5. Configure Design Initialization Data and Memories [\(Ask a Question\)](#)

The **Configure Design Initialization Data and Memories** step generates the TCM 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 TCM initialization client is stored in the sNVM.

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

To configure design initialization data and memories, perform the following steps:

1. In the **Design Flow** window, double-click **Configure Design Initialization Data and Memories**. The **Design and Memory Initialization** window opens, see the following figure.

Figure 2-3. Design and Memory Initialization

Design Initialization | uPROM | sNVM | SPI Flash | Fabric RAMs

Apply Discard Help

In design initialization, user design blocks such as LSRAM, uSRAM, transceivers, and PCIe can be initialized as an option using data stored in the non-volatile storage memory. The initialization data can be stored in uPROM, sNVM, or an external SPI Flash.

Follow the below steps to program the initialization data:

1. Set up your fabric RAMs initialization data, if any, using the 'Fabric RAMs' tab
2. Define the storage location of the initialization data
3. Generate the initialization clients
4. Generate or export the bitstream
5. Program the device

Design initialization specification

First stage (sNVM)

In the first stage, the initialization sequence de-asserts FABRIC_POR_N.

Second stage (sNVM)

In the second stage, the initialization sequence initializes the PCIe and XCVR blocks present in the design.

Start address for second stage initialization client: 0x 00000000

Third stage (sNVM/uPROM/SPI-Flash)

In the third stage, the initialization sequence initializes the fabric RAMs present in the design.

To save the initialization instructions in sNVM/uPROM/SPI-Flash, please use 'Fabric RAMs' tab to make your selection for each RAM client.

☒ Start address for sNVM clients: 0x 00000000 sNVM start page: 0

☐ Start address for uPROM clients: 0x 00000000

☐ Start address for SPI-Flash clients: 0x 00000400

SPI-Flash Binding: SPI-Flash - No-binding Plaintext SPI Clock divider value: 6

Time Out (s): 128

Auto Calibration Time Out (ms): 3000

Custom configuration file:

2. On the **Fabric RAMs** tab, click **tcm_ram** client from the list, and then click **Edit**, see the following figure.

Figure 2-4. Fabric RAMs Tab

Apply Discard Help

RAM statistics

SRAM Memory

Available Memory(B): 2437120

Used Memory(Bytes): 81920

Free Memory(Bytes): 2355200

SRAM Memory

Available Memory(B): 266112

Used Memory(Bytes): 864

Free Memory(Bytes): 265248

Clients

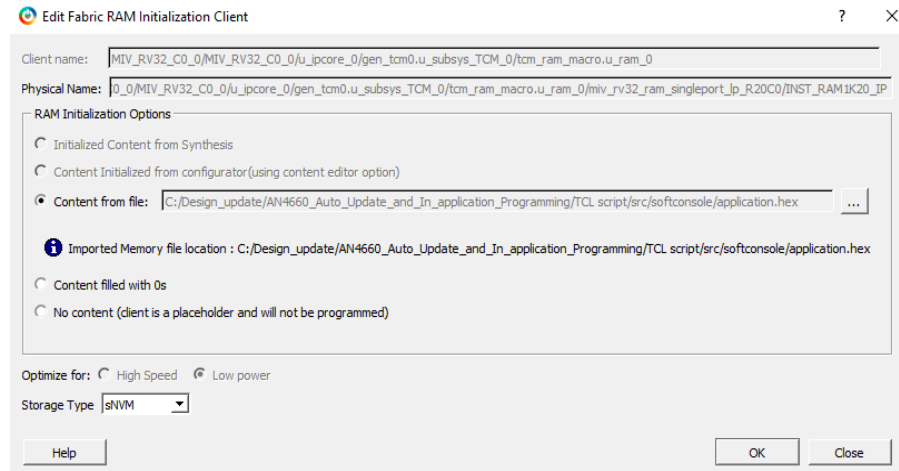
add design configuration Edit... Initialize all clients from Initialize all Clients from sNVM

☐ Filter out Inferred RAMs

Logical Instance Name	PORTA enth * Width	PORTB enth * Width
1 CORESPI_C0_0/CORESPI_C0_0/USPI/URXF/ffifo_mem_q[4]	4x5	4x5
2 CORESPI_C0_0/CORESPI_C0_0/USPI/UTXF/ffifo_mem_q[4]	4x5	4x5
3 MIV_RV32_C0_0/MIV_RV32_C0_0/u_ipcore_0/gen_tcm0.u_subsys_TCM_0/tcm_ram_macro.u_ram_0	16384x32	16384x32
4 MIV_RV32_C0_0/MIV_RV32_C0_0/u_ipcore_0/u_hart_0/u_expipe_0/gen_gpr_ram.u_gpr_0/gen_gpr_u_gpr_array_0/mem_xf[31:0]	32x32	32x32
5 MIV_RV32_C0_0/MIV_RV32_C0_0/u_ipcore_0/u_hart_0/u_expipe_0/gen_gpr_ram.u_gpr_0/gen_gpr_u_gpr_array_0/mem_xf_1[31:0]	32x32	32x32
6 MIV_RV32_C0_0/MIV_RV32_C0_0/u_ipcore_0/u_subsys_interconnect_0/u_subsys_regs/u_req_buffer/gen_buff_loop[0].buff_data[60]	2x6	2x6

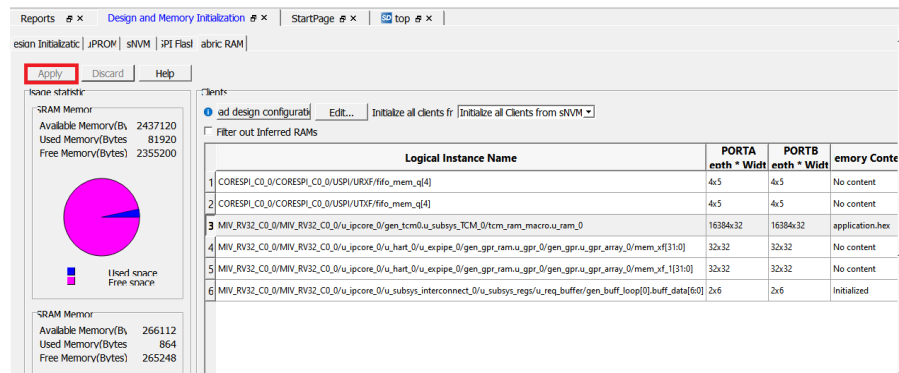
3. In the **Edit Fabric RAM Initialization Client** dialog box, select the **Content from file** option, locate the application.hex file from the DesignFiles_directory\HW\src\softconsole folder, and then click **OK**, as shown in the following figure.

Figure 2-5. Edit Fabric RAM Initialization Client



- Click **Apply**, see the following figure.

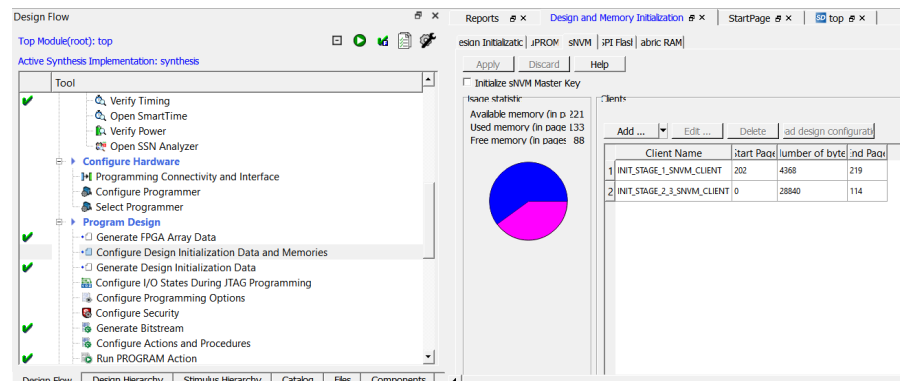
Figure 2-6. Apply Fabric RAM Content



- On the **Design Initialization** tab, click **Apply**.
- From Libero Design Flow, double-click **Generate Design 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. See the [Figure 2-2](#).

The following figure shows the client in the sNVM after **Generate Design Initialization Data**.

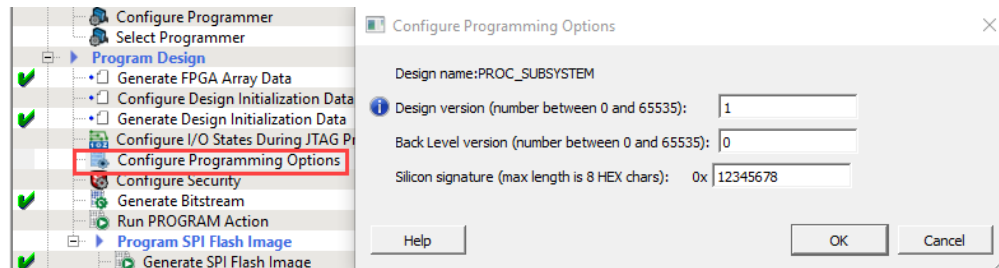
Figure 2-7. Client in the sNVM Option



2.6. Configure Programming Options [\(Ask a Question\)](#)

The Design version and user code (Silicon signature) are configured in this step. Double-click **Configure Programming Options** to give values, see the following figure.

Figure 2-8. Configure Programming Options

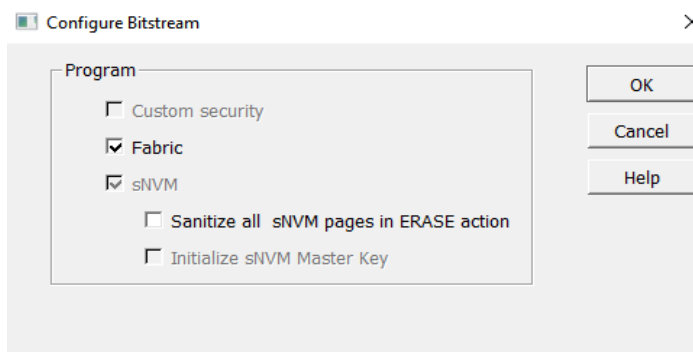


2.7. Generate Bitstream [\(Ask a Question\)](#)

To generate the bitstream, perform the following steps:

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

Figure 2-9. Generate Bitstream—Configure Bitstream Options



2. In the **Design Flow** window, double-click **Generate Bitstream**. When the bitstream is successfully generated, a green tick mark appears, see [Figure 2-2](#).
3. To view the corresponding log file in the **Reports** tab, right-click **Generate Bitstream**, and then click **View Report**.

2.8. Generating the SPI programming Images [\(Ask a Question\)](#)

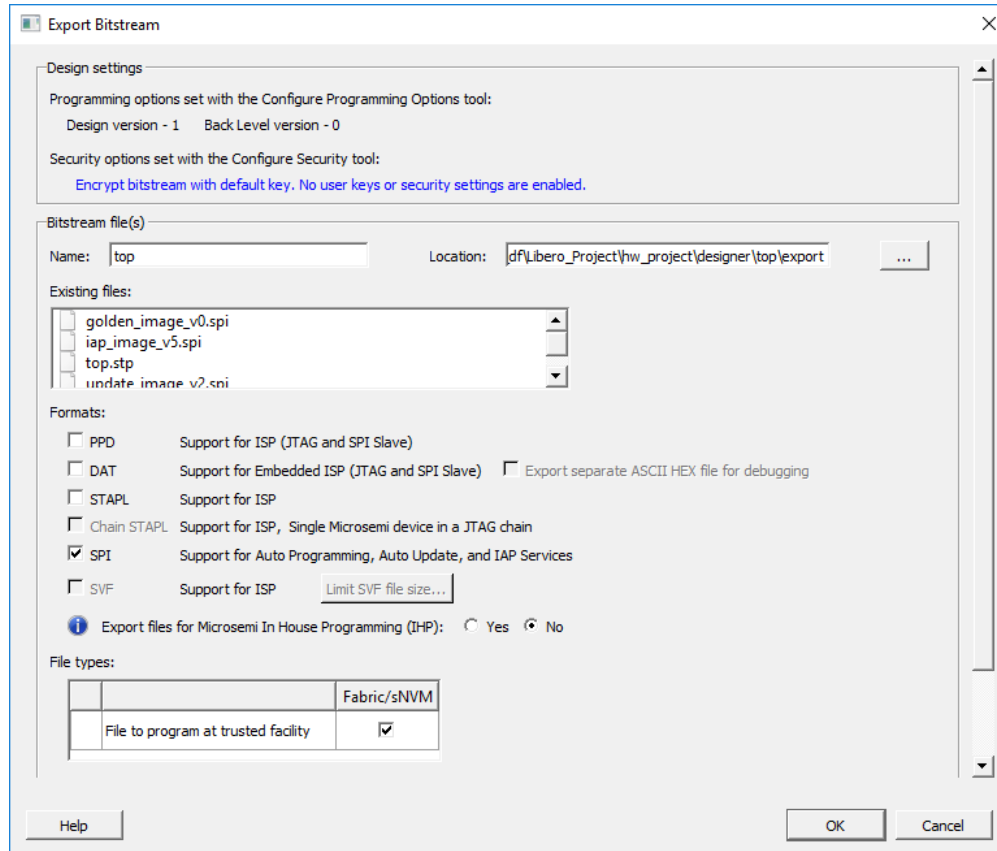
The following programming images are generated and copied to external SPI Flash memory.

Table 2-2. 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	N/A	0	0x00000400
update_image_v2.spi	2	0x23456789	1	0x00A00000
iap_image_v5.spi	5	0x56789ABC	2	0x01400000



Important: The golden image does not contain any security or Silicon signature information.

Figure 2-10. Generating the .SPI Programming Images

➔ Important: The golden image does not contain any security or Silicon signature information.

The .SPI programming images are generated using Export Bitstream option, see [Figure 2-10](#). Before generating the .SPI images, modify the Design version and Silicon Signature, see [Configure Programming Options](#).

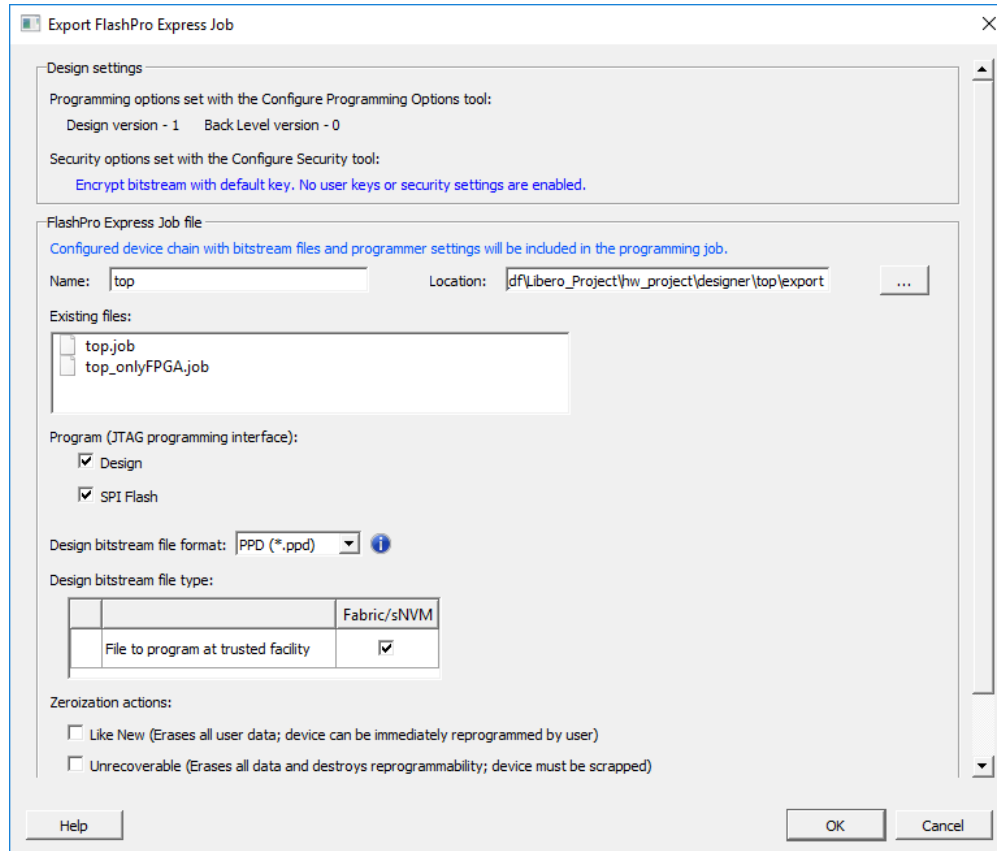
2.9. Export FlashPro Express Job [\(Ask a Question\)](#)

To generate .job file, perform the following steps:

- On the **Design Flow** tab, double-click **Export FlashPro Express Job**, and then click **Design and SPI Flash**, see the following figure.

Note: The exported job file contains the data contents to be programmed into PolarFire FPGA and external SPI Flash. This Job file is utilized in FlashPro Express software to program both Device and external SPI Flash, see [Appendix 2: Programming the Device and External SPI Flash Using FlashPro Express](#).

Figure 2-11. Export FlashPro Express Job



2.10. Programming the Device on the Evaluation Board [\(Ask a Question\)](#)

After generating the bitstream, the PolarFire device must be programmed with the Auto Update and IAP design.

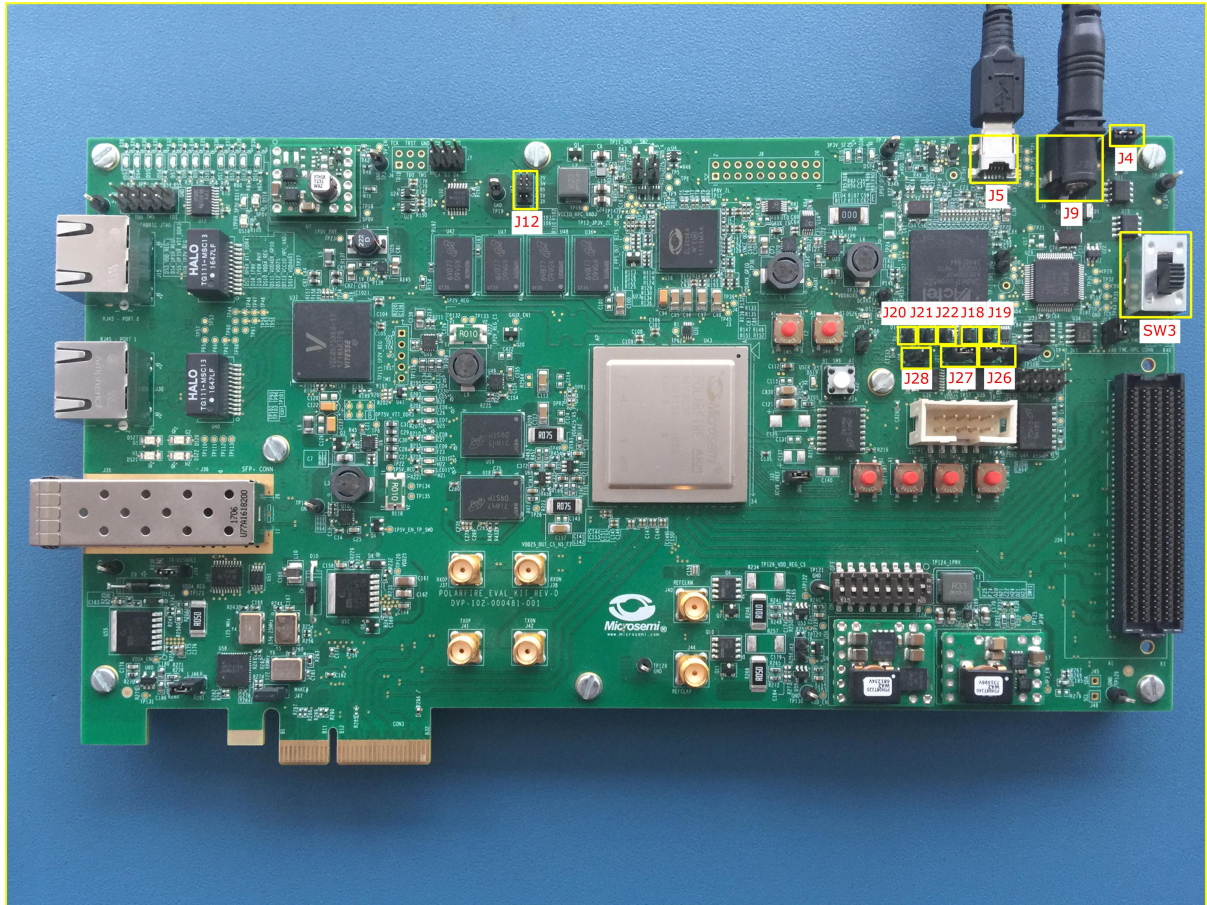
To program the PolarFire device, perform the following steps:

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

Table 2-3. Jumper Settings—Evaluation Board

Jumper	Description
J18, J19, J20, J21, and J22	Close pin 2 and 3 for programming the PolarFire FPGA through FTDI
J28	Close pin 1 and 2 for programming through the on-board FlashPro5
J23	Open pin 1 and 2 for accessing external SPI Flash
J4	Close pin 1 and 2 for manual power switching using SW3
J12	Close pin 3 and 4 for 2.5 V

2. Connect the power supply cable to the **J9** connector on the board.
 3. Connect the USB cable from the host PC to the **J5** (FTDI port) on the board.
 4. Power ON the board using the **SW3** slide switch.
- The following figure shows the board setup after these connections are made.

Figure 2-12. Board Setup—Evaluation Kit

5. On the **Libero Design Flow**, double-click **Run PROGRAM Action**.

The device is successfully programmed and the on-board LEDs glow. A green tick mark appears next to **Run PROGRAM Action**, see [Figure 2-2](#).

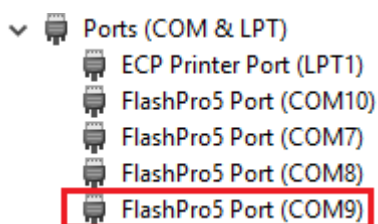
3. Serial Terminal Emulation Program Setup [\(Ask a Question\)](#)

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

To setup PuTTY, perform the following steps:

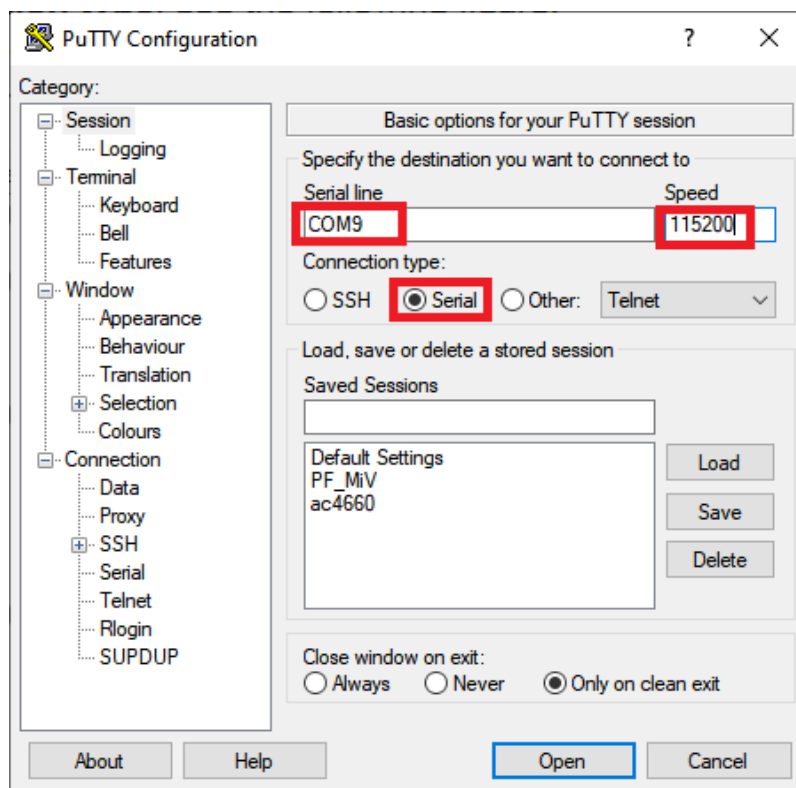
1. Connect the USB cable from the host PC to the **J5** (USB) port on the Evaluation board.
2. Connect the power supply cable to the **J9** connector on the Evaluation board.
3. Power on the Evaluation board using the **SW3**.
4. From the host PC, click **Start** and open **Device Manager** to note the second highest COM Port number and use that in the PuTTY configuration. In this example, COM Port 9 (COM9) is selected, see the following figure. COM Port-numbers may vary.

Figure 3-1. COM Port Number



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

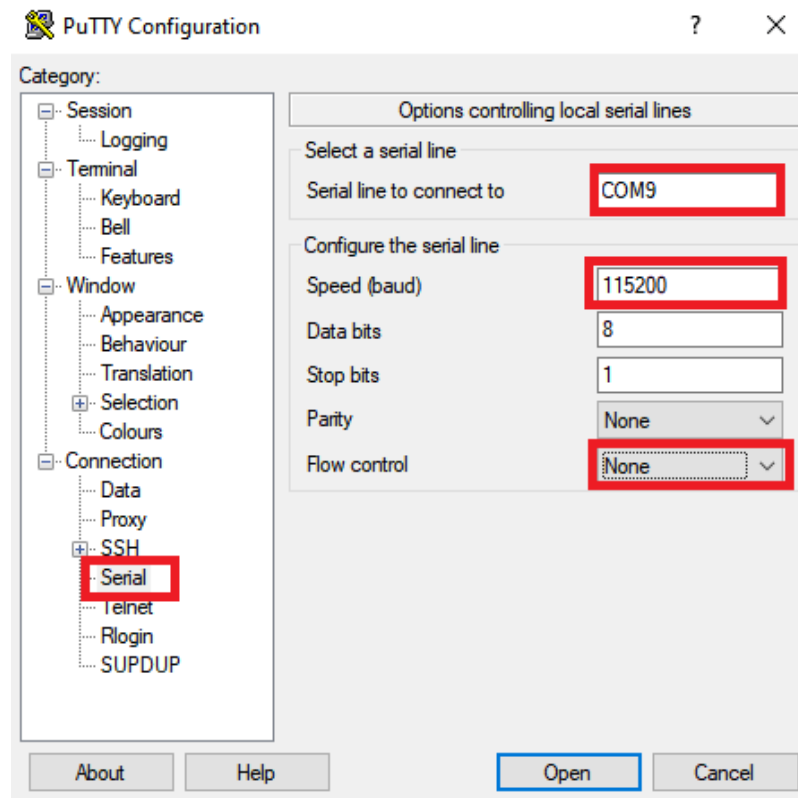
Figure 3-2. Select Serial as the Connection Type



7. Set the **Serial line to connect** to COM port number as noted in Step 4.

8. Set the **Speed (baud)** to **115200**, see the following figure.
9. Set the **Flow control** to **None**, and then click **Open**.

Figure 3-3. PuTTY Configuration



PuTTY opens successfully, and this completes the serial terminal emulation program setup. See [Running the Demo](#).

4. Running the Demo [\(Ask a Question\)](#)

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](#).

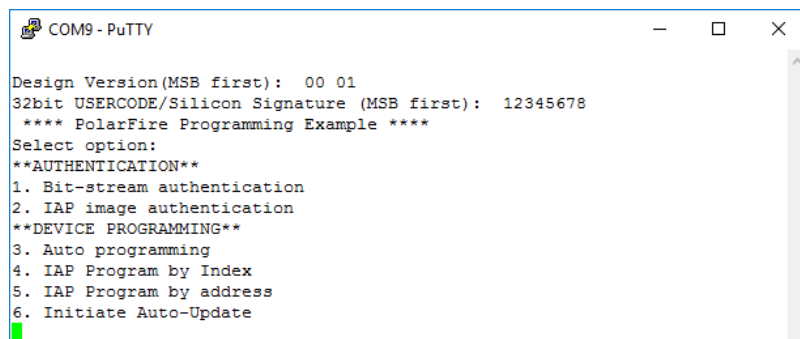
The on-board 1 GB Micron SPI Flash device is connected to System Controller SPI and can be programmed using the Libero SoC PolarFire software or fabric logic. For more information about programming the on-board SPI Flash using Fabric Logic, see [Appendix 1: Programming On-board SPI Flash using the Fabric Logic through the Host Loader](#).

Before you begin, ensure that the following configurations are in place:

1. Ensure that the device is programmed with the `programming_appnote_only_FPGA_v1.job` file. See [Appendix 2: Programming the Device and External SPI Flash Using FlashPro Express](#).
2. Connect the power supply cable to the **J9** (Evaluation board) connector on the board.
3. Connect the USB cable from the host PC to FTDI port **J5** (Evaluation board) on the board.
4. Ensure that on-board **SW11** (Evaluation board) DIP 1 is set to OFF.
5. Open pin 1 and 2 of the **J23** jumper.
6. Power ON the Evaluation board using the **SW3**.

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

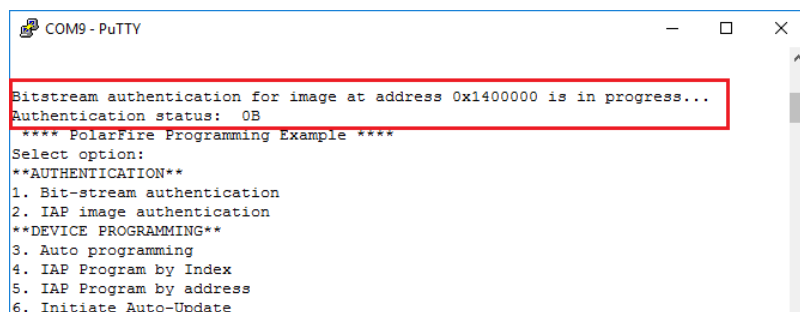
Figure 4-1. Authentication and Programming Options



```
COM9 - PuTTY
Design Version(MSB first): 00 01
32bit USERCODE/Silicon Signature (MSB first): 12345678
**** 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
```

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 4-2. Authentication Error



```
COM9 - PuTTY
Bitstream authentication for image at address 0x1400000 is in progress...
Authentication status: 0B
**** 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
```

Selecting option 4, 5, or 6 does not initiate any program operation as the on-board SPI Flash is empty.

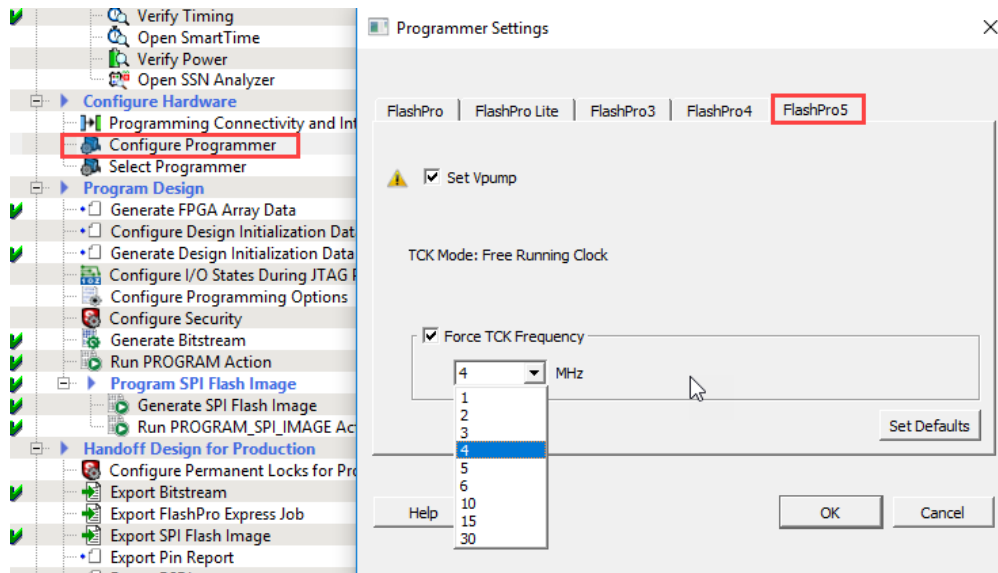
4.1. Programming On-board SPI Flash Using Libero [\(Ask a Question\)](#)

Libero SoC Design Suite supports the on-board SPI Flash programming using JTAG. For more information about the SPI Flash programming modes, see [PolarFire FPGA and PolarFire SoC FPGA Programming User Guide](#).

The external SPI Flash can also be programmed through FlashPro Express software. For more information, see [Appendix 2: Programming the Device and External SPI Flash Using FlashPro Express](#).

To optimize the SPI Flash programming time, change the TCK frequency value under **Configure Programmer**, as shown in the following figure.

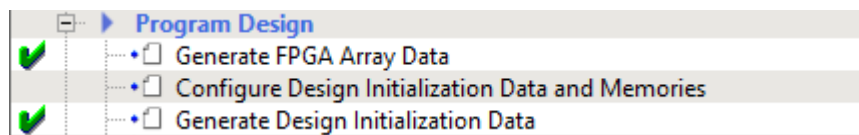
Figure 4-3. Configure Programming Settings



To program the SPI Flash using JTAG, perform the following steps:

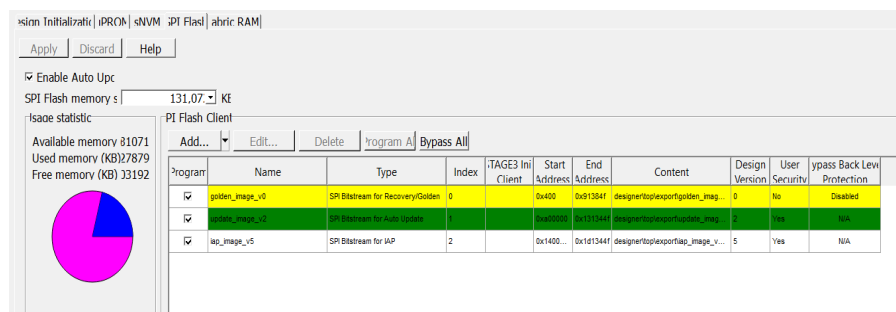
1. Ensure that the jumper settings on the board are the same as those listed in [Table 2-3](#) (for Evaluation board).
2. In the **Design Flow** window, select **Program Design**, and then double-click **Configure Design Initialization Data and Memories**.

Figure 4-4. Configure Design Initialization Data and Memories Option



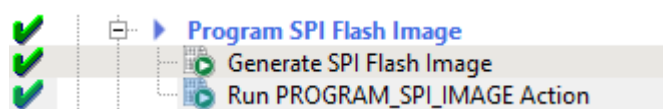
3. In the **Design and Memory Initialization** page, click the **SPI Flash** tab, see [Figure 4-5](#).
4. In the **SPI Flash Clients** pane, click **Add SPI Bitstream Client** to add the required programming images (.spi images), and click **Apply**. These images are provided at `\mpf_an4660_df\host_pc_tool_pf\images`.

Figure 4-5. SPI Flash Tab



- To get the SPI Flash programmed with the programming images, double-click **Generate SPI Flash Image**, and then double-click **Run PROGRAM_SPI_IMAGE Action**, see the following figure.

Figure 4-6. SPI Flash Programming



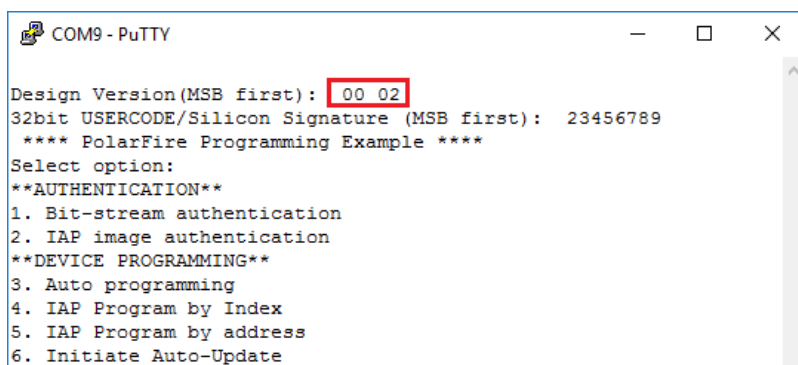
At this point, the on-board SPI Flash device is programmed with the images.

4.2. Running Auto Update [\(Ask a Question\)](#)

To run auto update, 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.

The following figure shows the design version **02**.

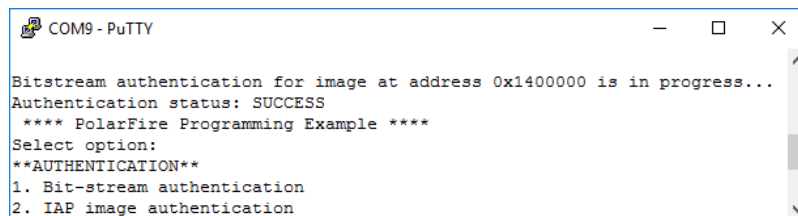
Figure 4-7. Auto Update



4.3. Running Authentication [\(Ask a Question\)](#)

To run bitstream authentication, perform the following steps:

- To initiate the bitstream authentication, press **1**.
After successful authentication, PuTTY displays the status code, see the following figure.

Figure 4-8. Successful Bitstream Authentication


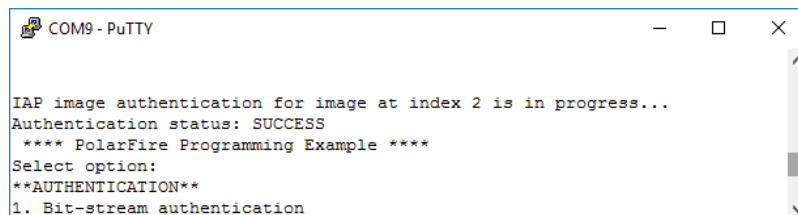
```

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

```

- To initiate the IAP image authentication, press **2**.
After successful authentication, PuTTY displays the status code, see the following figure.

Figure 4-9. Successful IAP Image Authentication


```

COM9 - PuTTY

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

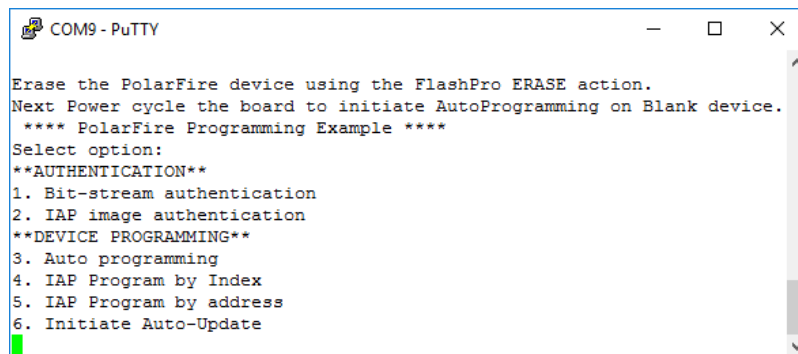
```

This concludes the bitstream and IAP image authentication.

4.4. Running Auto Programming [\(Ask a Question\)](#)

To run Auto programming, perform the following steps:

- Press **3** in PuTTY. The PuTTY notifies to erase the device using FlashPro and power-cycle the board, see the following figure.

Figure 4-10. Notifying ERASE Action


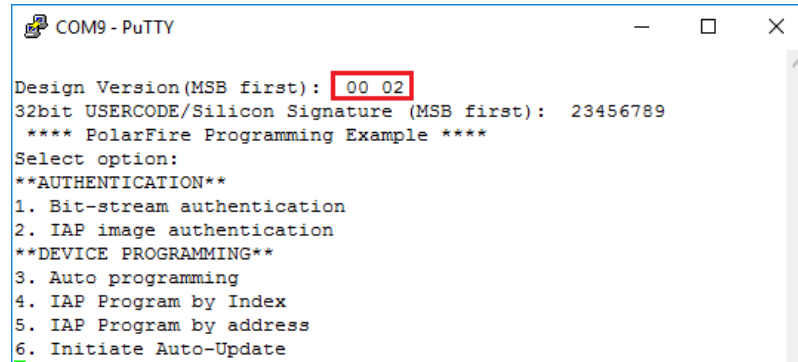
```

COM9 - PuTTY

Erase the PolarFire device using the FlashPro ERASE action.
Next Power cycle the board to initiate AutoProgramming on Blank device.
**** 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

```

- Using FlashPro, erase the device and power-cycle the board.
All the LEDs stop glowing for a few seconds, which indicates that the auto programming is in progress. The highest programming image version is selected from the 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, see the following figure.

Figure 4-11. Successful Auto Programming

```
COM9 - PuTTY

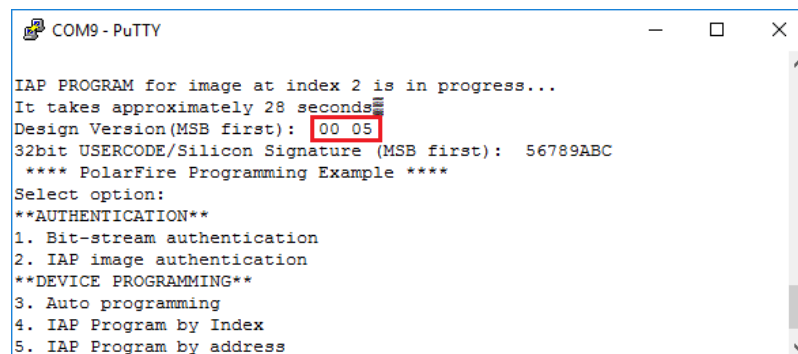
Design Version(MSB first): 00 02
32bit USERCODE/Silicon Signature (MSB first): 23456789
**** PolarFire Programming Example ****
Select option:
**AUTHENTICATION**
1. Bit-stream authentication
2. IAP image authentication
**DEVICE PROGRAMMING**
3. Auto programming
4. IAP Program by Index
5. IAP Program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

This concludes running the Auto programming feature.

4.5. Running IAP [\(Ask a Question\)](#)

To run IAP, perform the following steps:

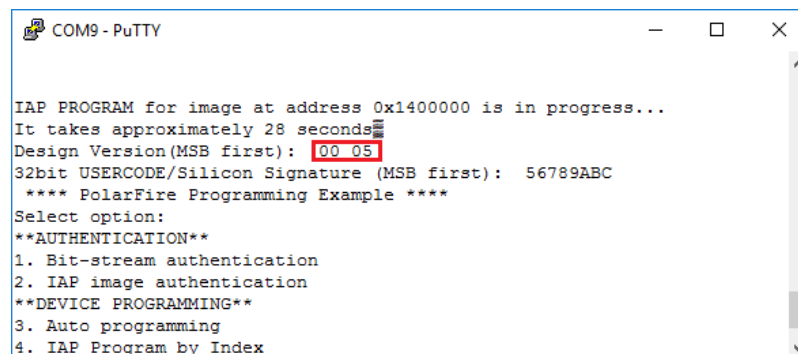
1. Press **4**, IAP program by Index. After around 28 seconds, the IAP with image at index 2 is executed successfully and the design version **05** is displayed, see the following figure.

Figure 4-12. Successful IAP at Index 2

```
COM9 - PuTTY

IAP PROGRAM for image at index 2 is in progress...
It takes approximately 28 seconds
Design Version(MSB first): 00 05
32bit USERCODE/Silicon Signature (MSB first): 56789ABC
**** 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
```

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

Figure 4-13. Successful IAP by Address

```
COM9 - PuTTY

IAP PROGRAM for image at address 0x1400000 is in progress...
It takes approximately 28 seconds
Design Version(MSB first): 00 05
32bit USERCODE/Silicon Signature (MSB first): 56789ABC
**** PolarFire Programming Example ****
Select option:
**AUTHENTICATION**
1. Bit-stream authentication
2. IAP image authentication
**DEVICE PROGRAMMING**
3. Auto programming
4. IAP Program by Index
```

This concludes running the IAP feature.

For information about programming the on-board SPI Flash using the fabric logic, see [Appendix 1: Programming On-board SPI Flash using the Fabric Logic through the Host Loader](#).

5. Appendix 1: Programming On-board SPI Flash using the Fabric Logic through the Host Loader [\(Ask a Question\)](#)

To program the SPI Flash, perform the following steps:

1. Power off the Evaluation board using the **SW3** slide switch. Close the PuTTY and set the on-board **SW11** (Evaluation board) DIP 1 to ON state.
2. Disconnect and connect the USB cable from the host PC to FTDI port **J5** on the Evaluation board. This ensures clearing off UART buffers.
3. Power on the Evaluation board using the **SW3** slide switch.
4. Locate the `load_spi_flash.bat` batch file from the `DesignFiles_Folder\host_pc_tool_pf` folder.
5. Open or edit the `load_spi_flash.bat` batch file and ensure that it matches the COM port number. For example, COM Port 9 in this instance.

```
spi_loader.exe 9 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 as 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.
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 5-1. Erasing SPI Flash

```
C:\WINDOWS\system32\cmd.exe
D:\> \host_pc_tool_pf>spi_loader.exe 9 golden_image_v0.spi update_image_v2.spi iap_image_v5.spi
Serial port \\.\COM9 successfully reconfigured.
Ensure the PolarFire Kit is running.
Press 'Enter' to Program the External SPI flash ..
If you want to run programming options, change the DIP switch-1 position to OFF and power cycle the board.
The External SPI flash is erasing...
Handshaking with PolarFire kit is in progress...
```

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 5-2. 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.
9. Set the on-board **SW11** (Evaluation board) DIP1 to OFF state and open the PuTTY terminal. To select the programming options, power cycle the board.

6. Appendix 2: Programming the Device and External SPI Flash Using FlashPro Express [\(Ask a Question\)](#)

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

```
mpf_an4660_df\Programming_Job
```

programming_appnote_FPGA_SPI_images_v1 contains both PolarFire device contents and SPI images. When the file is selected for programming, the FlashPro express software programs the PolarFire FPGA device and the external SPI Flash memory with programming images. The programming takes nearly 30 minutes to complete.

programming_appnote_only_FPGA_v1 contains only PolarFire device contents. When the file is selected for programming, the FlashPro express software programs the PolarFire FPGA device only.

To program the device and external SPI Flash, perform the following steps:

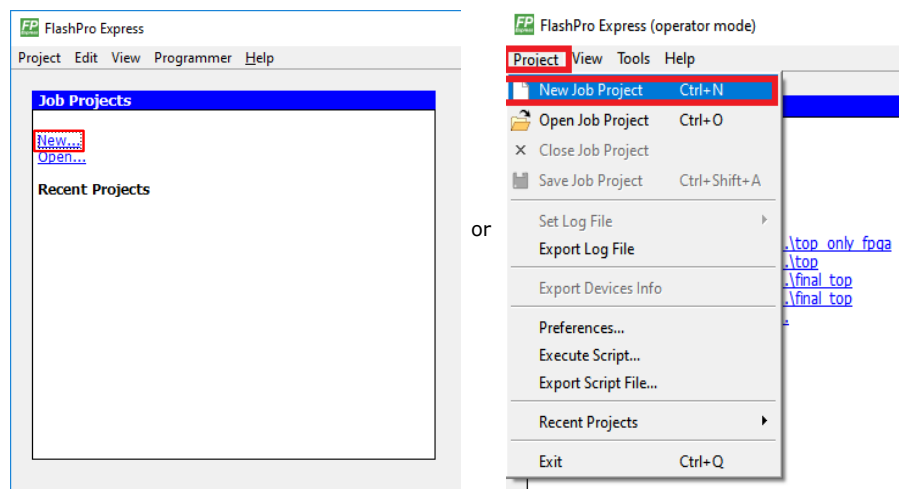
1. Ensure that the jumper settings on the board are the same as those listed in [Table 2-3](#) (for Evaluation board).



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

2. Connect the power supply cable to the **J9** connector on the Evaluation board.
3. Connect the USB cable from the host PC to the **J5** (FTDI port) on the Evaluation board.
4. Power on the Evaluation board using the **SW3** slide switch.
5. On the host PC, launch the **FlashPro Express** software.
6. To create a new job project, click **New** or select **New Job Project** from FlashPro Express Job, as shown the following figure.

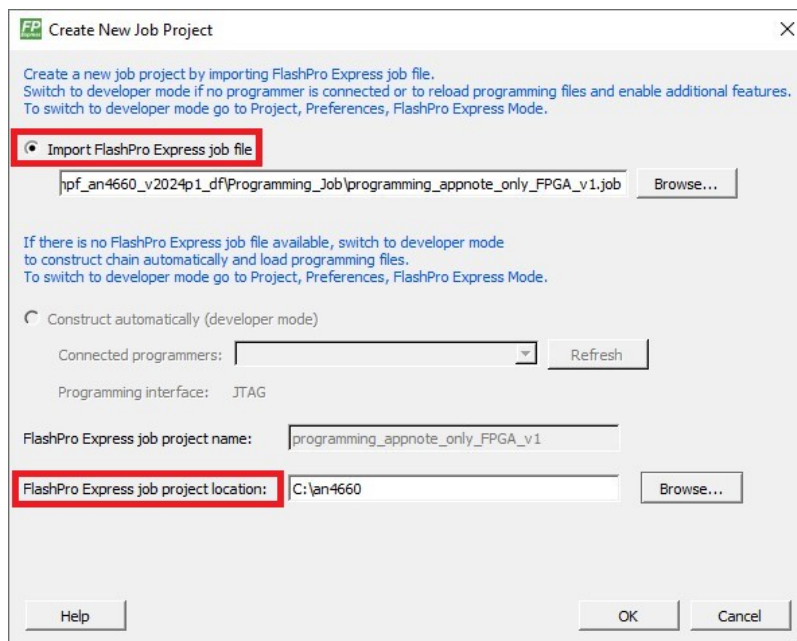
Figure 6-1. FlashPro Express Job Project



7. Enter the following in the **New Job Project from FlashPro Express Job** dialog box:
 - **Programming job file:** Click **Browse** and navigate to the location where the .job file is located and select the file. The default location is:
`<download_folder>\mpf_an4660_df\Programming_Job.`

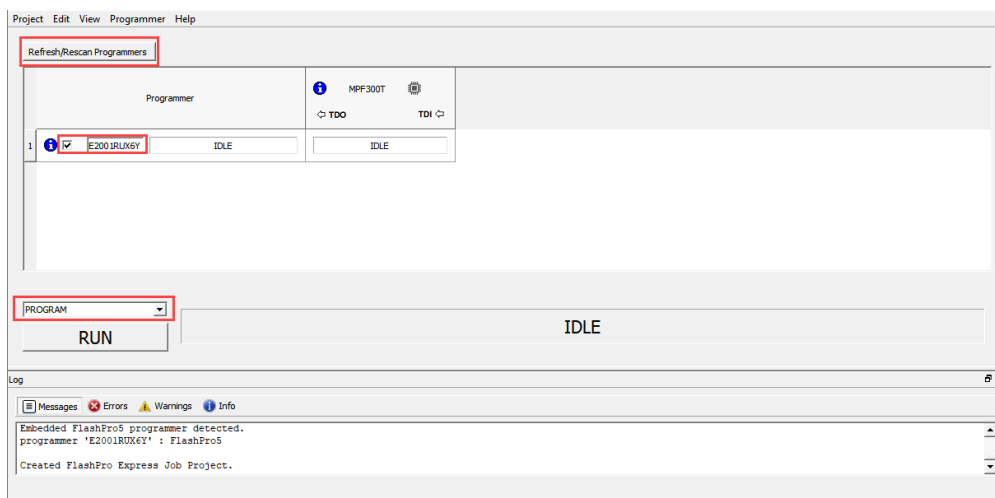
- **FlashPro Express job project location:** Click **Browse** and navigate to the location where you want to save the project.

Figure 6-2. New Job Project from FlashPro Express Job



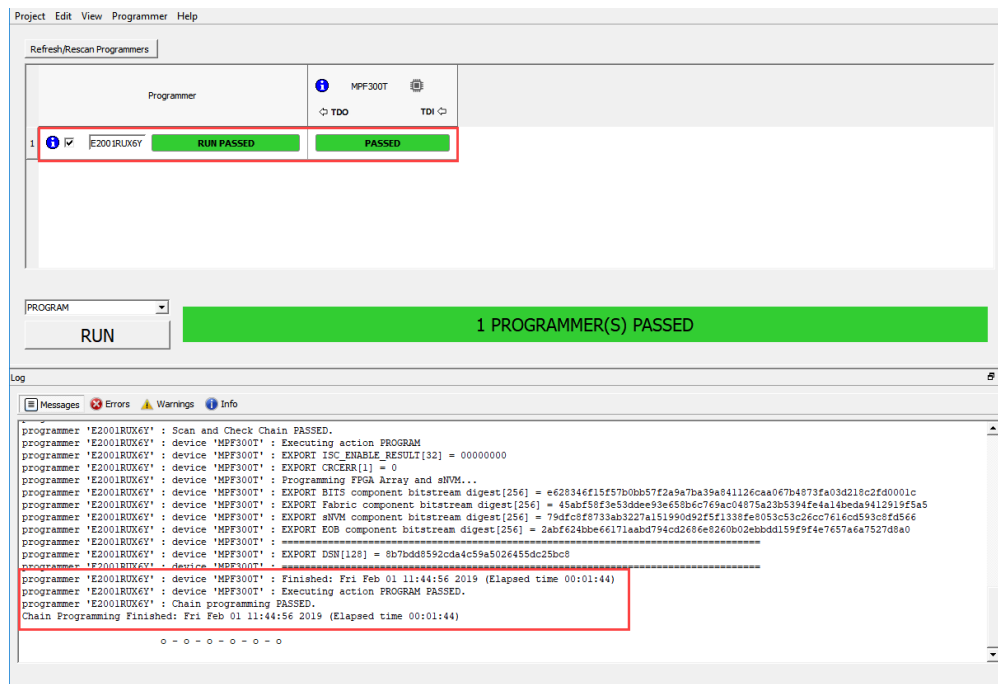
8. Click **OK**. The required programming file is selected and ready to be programmed in the device.
9. The **FlashPro Express** window appears. Confirm that a programmer number appears in the Programmer field. If it does not, confirm the board connections and click **Refresh/Rescan** **Programmers**.

Figure 6-3. Programming the Device



10. Click **RUN**. When the device is programmed successfully, a **RUN PASSED** status is displayed, as shown in the following figure. To run the demo, see [Running the Demo](#).

Figure 6-4. FlashPro Express—RUN PASSED



11. Close **FlashPro Express**, **Project > Exit**.

7. Appendix 3: Running the TCL Script [\(Ask a Question\)](#)

TCL scripts are provided in the design files folder under directory `HW`. If required, the design flow can be reproduced from the Design Implementation before the job file is generated.

To run the TCL, perform the following steps:

1. Launch the Libero[®] software.
2. Select **Project > Execute Script....**
3. Click **Browse** and from the downloaded `HW` directory, select `script.tcl`
4. Click **Run**.

After successful execution of TCL script, Libero project is created within `HW` directory.

For more information about TCL scripts, see `mpf_an4660_df/HW/TCL_Scripts_readme.txt`.

For more details on TCL commands, see [Tcl Commands Reference Guide](#). Contact Technical Support for any queries encountered when running the TCL script.

8. Appendix 4: References [\(Ask a Question\)](#)

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

- For more information about PolarFire FPGA programming, see [PolarFire FPGA and PolarFire SoC FPGA Programming User Guide](#).
- For more information about the CoreJTAGDEBUG IP core, see CoreJTAGDebug_HB.pdf. This user guide can be downloaded from the Libero SoC Catalog.
- For more information about the MIV_RV32 IP core, see MIV_RV32 Handbook. This user guide can be downloaded from the Libero SoC Catalog.
- For more information about the CoreUARTapb IP core, see CoreUARTapb_HB.pdf. This user guide can be downloaded from the Libero SoC Catalog.
- For more information about the CoreAPB3 IP core, see CoreAPB3_HB.pdf. This user guide can be downloaded from the Libero SoC Catalog.
- For more information about the CoreGPIO IP core, see CoreGPIO_HB.pdf. This user guide can be downloaded from the Libero SoC Catalog.
- For more information about the PolarFire initialization monitor, see [PolarFire Family Power-Up and Resets User Guide](#).
- For more information about how to build a Mi-V processor subsystem for PolarFire devices, see [AN4997: PolarFire FPGA Building a Mi-V Processor Subsystem](#).
- For more information about the PF_CCC IP core, see [PolarFire Family Clocking Resources User Guide](#).
- For more information about Libero, ModelSim, and Synplify, see [Libero SoC Documentation](#).

9. Revision History [\(Ask a Question\)](#)

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

Table 9-1. Revision History

Revision	Date	Description
D	05/2025	Updated the TCL filepath in the Appendix 3: Running the TCL Script section
C	04/2025	The following is the list of changes in revision C of the document: <ul style="list-style-type: none"> Updated the IAP design with the TCK frequency constraint in the SDC file Updated the .job filepath throughout the document
B	07/2024	The following is the list of changes in revision B of the document: <ul style="list-style-type: none"> Updated Figure 1-2 in Demo Design section Updated Figure 1-5 and Table 1-3 in Design Implementation section Updated "100 MHz" to "83.33 MHz" and Figure 1-8 in PF_CCC_0 Configuration section Updated Reset Vector Address values, Figure 1-9, and Figure 1-10 in Mi-V Soft Processor Configuration section Removed "CoreGPIO_0 Configuration" and "CoreAPB3 Configuration" section Added Instantiating MiV ESS Core section Updated "100 MHz" to "83.33 MHz" in Figure 1-16 in Clocking Structure section Updated Figure 2-6 and Figure 2-7 in Configure Design Initialization Data and Memories section Updated Figure 3-2 and Figure 3-3 in Serial Terminal Emulation Program Setup section Updated Figure 6-1 in Appendix 2: Programming the Device and External SPI Flash Using FlashPro Express section
A	08/2022	The following is the list of changes in revision A of the document: <ul style="list-style-type: none"> Migrated the document to the Microchip template Updated the document number to DS00004660A from 51900466 Updated the document ID to AN4660 from AC466 Removed all Splash Board related content from document Replaced all Microsemi links with Microchip links Updated hex file location throughout the document Removed "Programming the Device" section Updated .job programming file location in Appendix 2: Programming the Device and External SPI Flash Using FlashPro Express and Appendix 3: Running the TCL Script
7.0	—	Added Appendix 3: Running the TCL Script .
6.0	—	The following is a summary of the changes made in this revision. <ul style="list-style-type: none"> Updated the document for Libero SoC v12.2 Removed the references to Libero version numbers
5.0	—	The document was updated for Libero SoC v12.0
4.0	—	Merged Splash kit related content and updated the document for Libero SoC PolarFire v2.3 release
3.0	—	The document was updated for Libero SoC PolarFire v2.2 release
2.0	—	The document was updated for Libero SoC PolarFire v2.1 release
1.0	—	The first publication of this document

Microchip FPGA Support

Microchip FPGA products group backs its products with various support services, including Customer Service, Customer Technical Support Center, a website, and worldwide sales offices. Customers are suggested to visit Microchip online resources prior to contacting support as it is very likely that their queries have been already answered.

Contact Technical Support Center through the website at www.microchip.com/support. Mention the FPGA Device Part number, select appropriate case category, and upload design files while creating a technical support case.

Contact Customer Service for non-technical product support, such as product pricing, product upgrades, update information, order status, and authorization.

- From North America, call **800.262.1060**
- From the rest of the world, call **650.318.4460**
- Fax, from anywhere in the world, **650.318.8044**

Microchip Information

Trademarks

The “Microchip” name and logo, the “M” logo, and other names, logos, and brands are registered and unregistered trademarks of Microchip Technology Incorporated or its affiliates and/or subsidiaries in the United States and/or other countries (“Microchip Trademarks”). Information regarding Microchip Trademarks can be found at <https://www.microchip.com/en-us/about/legal-information/microchip-trademarks>.

ISBN: 979-8-3371-1231-2

Legal Notice

This publication and the information herein may be used only with Microchip products, including to design, test, and integrate Microchip products with your application. Use of this information in any other manner violates these terms. Information regarding device applications is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. Contact your local Microchip sales office for additional support or, obtain additional support at www.microchip.com/en-us/support/design-help/client-support-services.

THIS INFORMATION IS PROVIDED BY MICROCHIP “AS IS”. MICROCHIP MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION INCLUDING BUT NOT LIMITED TO ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE, OR WARRANTIES RELATED TO ITS CONDITION, QUALITY, OR PERFORMANCE.

IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT, SPECIAL, PUNITIVE, INCIDENTAL, OR CONSEQUENTIAL LOSS, DAMAGE, COST, OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE INFORMATION OR ITS USE, HOWEVER CAUSED, EVEN IF MICROCHIP HAS BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE FULLEST EXTENT ALLOWED BY LAW, MICROCHIP’S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THE INFORMATION OR ITS USE WILL NOT EXCEED THE AMOUNT OF FEES, IF ANY, THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THE INFORMATION.

Use of Microchip devices in life support and/or safety applications is entirely at the buyer’s risk, and the buyer agrees to defend, indemnify and hold harmless Microchip from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any Microchip intellectual property rights unless otherwise stated.

Microchip Devices Code Protection Feature

Note the following details of the code protection feature on Microchip products:

- Microchip products meet the specifications contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is secure when used in the intended manner, within operating specifications, and under normal conditions.
- Microchip values and aggressively protects its intellectual property rights. Attempts to breach the code protection features of Microchip products are strictly prohibited and may violate the Digital Millennium Copyright Act.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of its code. Code protection does not mean that we are guaranteeing the product is “unbreakable”. Code protection is constantly evolving. Microchip is committed to continuously improving the code protection features of our products.