PolarFire FPGA and PolarFire SoC FPGA Programming User Guide

Introduction (Ask a Question)

Microchip's PolarFire® FPGAs are the fifth-generation family of non-volatile FPGA devices, built on state-of-the-art 28 nm non-volatile process technology. PolarFire FPGAs deliver the lowest power at mid-range densities. PolarFire FPGAs lower the cost of mid-range FPGAs by integrating the industry's lowest power FPGA fabric, lowest power 12.7 Gbps transceiver lane, built-in low power dual PCI Express® Gen2 (EP/RP), and, on select data security (S) devices, an integrated low-power crypto co-processor.

The PolarFire SoC family offers industry's first RISC-V based SoC FPGAs capable of running Linux®. It combines a powerful 64-bit 5x core RISC-V Microprocessor Subsystem (MSS), based on SiFive's U54-MC family, with the PolarFire FPGA fabric in a single device.

PolarFire FPGAs and PolarFire SoC FPGAs offer a variety of programming options to diverse end-user applications. The following table lists the components that are programmable in PolarFire FPGA and PolarFire SoC FPGA.

Table 1. Programming Components

Components	PolarFire® FPGA (MPF)	PolarFire® SoC FPGA (MPFS)
FPGA fabric	✓	✓
Secure Non-Volatile Memory (sNVM)	✓	✓
Embedded Non-Volatile Memory (eNVM)	—	✓
User security settings (keys, passcodes, and locks)	✓	✓

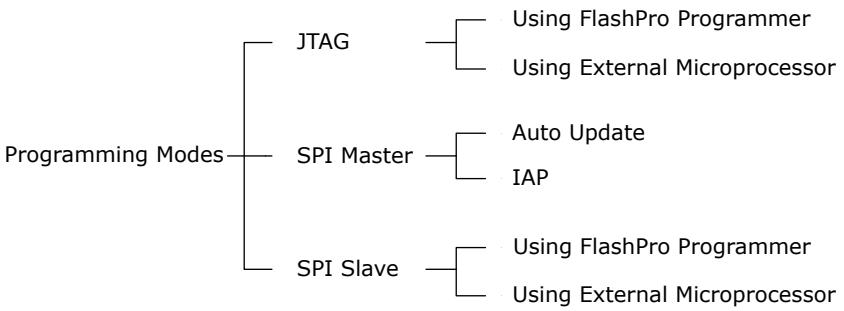
Both the device families can be programmed using on-chip system controller through its dedicated JTAG or SPI interface. Based on the interface used, the following three programming modes are supported:

- JTAG
- SPI master
- SPI slave

In JTAG and SPI slave programming modes, the device can be programmed either using an external master, such as a microprocessor or a Microchip FlashPro programmer (version 5 or later). The external master fetches the programming data (bitstream) from an external memory.

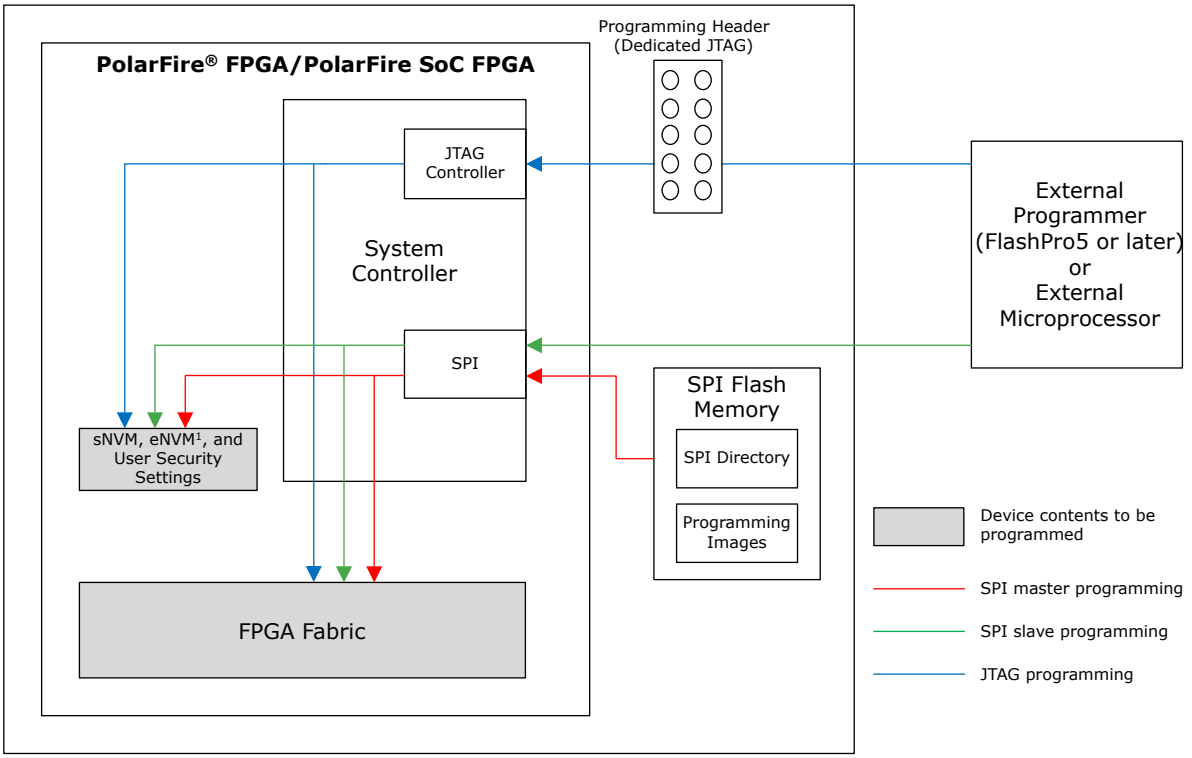
In SPI master programming mode, the system controller acts as the master and fetches the bitstream from an external SPI flash memory to program the device. This mode supports two programming features—Auto Update and In-Application Programming (IAP). In auto update, the device reprograms itself on power-up, and in IAP, the device is programmed when the user application initiates programming. The following figure shows the programming modes.

Figure 1. Programming Modes



The following block diagram shows the device programming modes and the associated interfaces.

Figure 2. Device Programming Modes and Interfaces



[1] *Applicable for PolarFire SoC FPGA only.*

Note: When device is used in System Controller Suspend mode, device programming is disabled to protect the device from unintended programming because of single event upsets. After device initialization, the system controller is held in reset state and cannot provide system services such as security, IAP, or auto update programming. After the device exits System Controller Suspend mode, JTAG and SPI Slave programming modes are available.

References (Ask a Question)

- For information about sNVM, eNVM, and Security Settings, see PolarFire FPGA and PolarFire SoC FPGA Security User Guide.
- For information about programming cycle count, see PolarFire FPGA and PolarFire SoC FPGA System Services User Guide.
- For information about design initialization, see PolarFire FPGA and PolarFire SoC FPGA Device Power-Up and Resets User Guide.
- For information about power supply requirement and filtering capacitors, see respective UG0726: PolarFire FPGA Board Design User Guide or PolarFire SoC Board Design Guidelines User Guide.
- For information about using Libero® SoC for PolarFire FPGA and PolarFire SoC FPGA, see Libero SoC Documentation.
- For information about MSS, see PolarFire SoC FPGA MSS Technical Reference Manual.

Table of Contents

1. Bitstream Generation (Ask a Question)

The Libero® SoC design suite generates the programming bitstream required for various programming modes. Depending on the requirement, the programming bitstream may contain one or more of the following components:

- FPGA fabric logic
- sNVM data
- eNVM data (for PolarFire SoC FPGA only)
- User security settings

The following table lists the programming interfaces used in various programming modes and the associated bitstream formats.

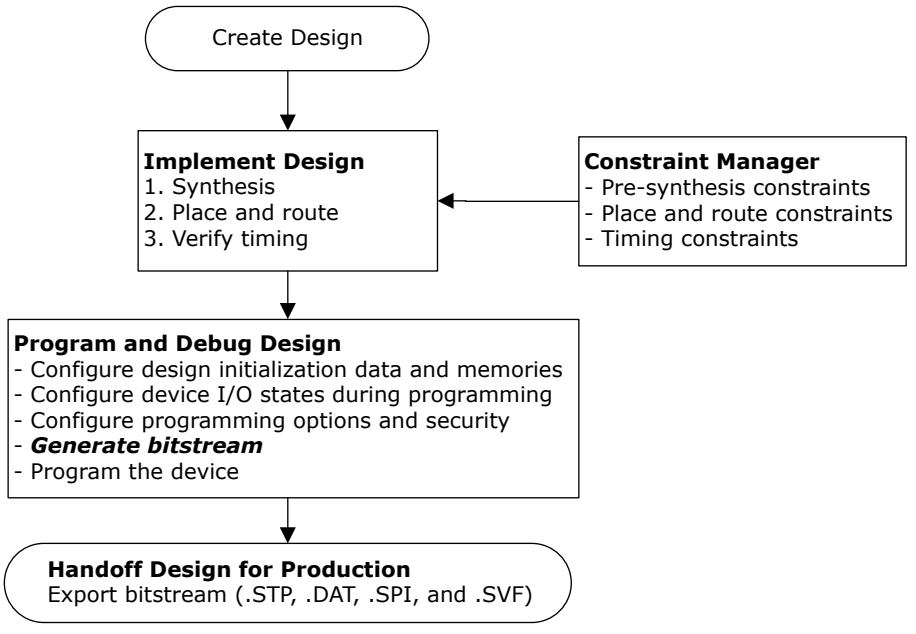
Table 1-1. PolarFire and PolarFire SoC FPGA Programming Interfaces and Bitstream Formats

Programming Mode	Interface	Master	Bitstream Format
JTAG programming	System controller's dedicated JTAG	FlashPro programmer	STP/JOB
JTAG programming	System controller's dedicated JTAG	External microprocessor	DAT
SPI slave programming	System controller's dedicated SPI	FlashPro programmer	DAT
SPI slave programming	System controller's dedicated SPI	External microprocessor	DAT
SPI master programming	System controller's dedicated SPI	System controller	SPI

1.1. Bitstream Generation Flow (Ask a Question)

The following figure shows where the bitstream is generated in the Libero SoC.

Figure 1-1. Bitstream Generation in Libero® Design Flow

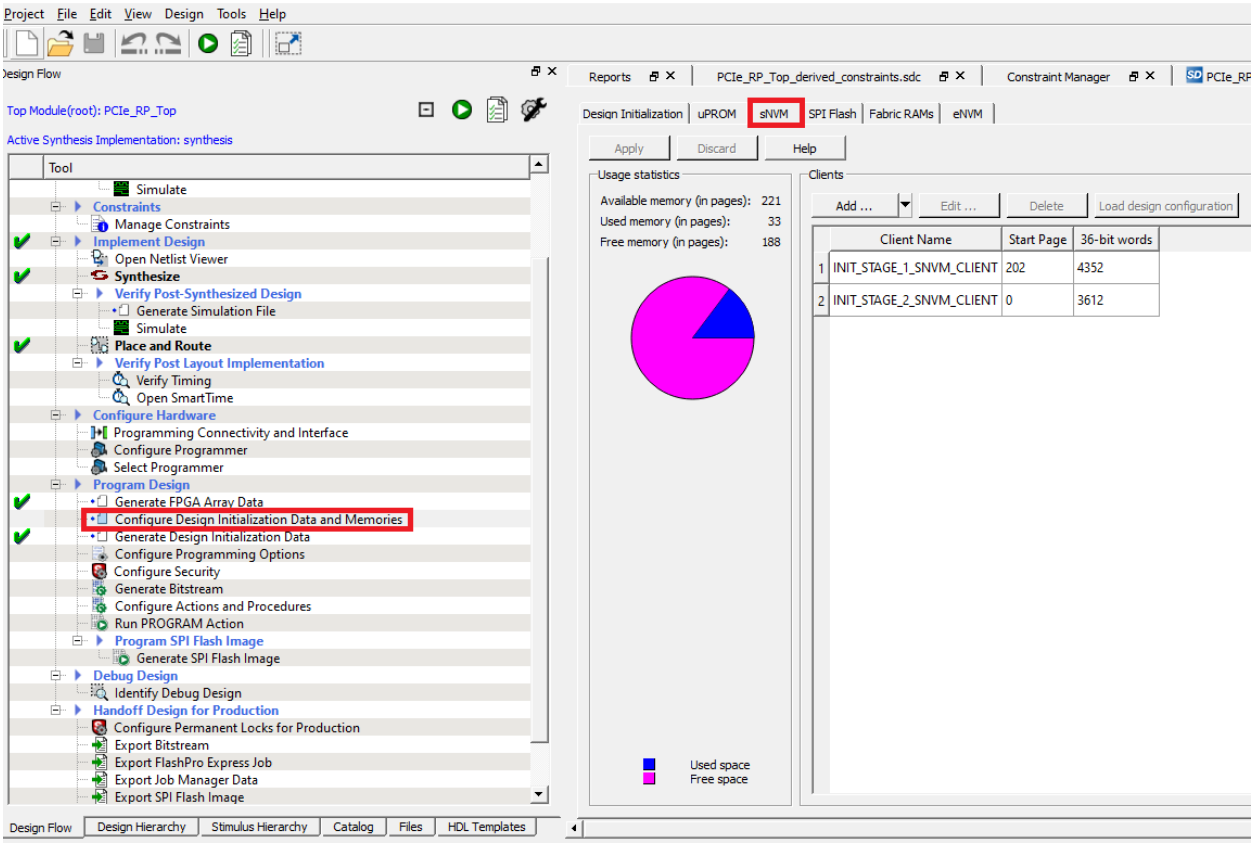


1.2. Adding sNVM Data to the Bitstream (Ask a Question)

The sNVM is a user non-volatile flash memory that can be programmed independently. Each device has 56 KB of sNVM.

To add multiple sNVM data clients to the bitstream in Libero SoC, go to **Design Flow** > **Program Design** > **Configure Design Initialization Data and Memories**, as shown in the following figure.

Figure 1-2. Design and Memory Initialization



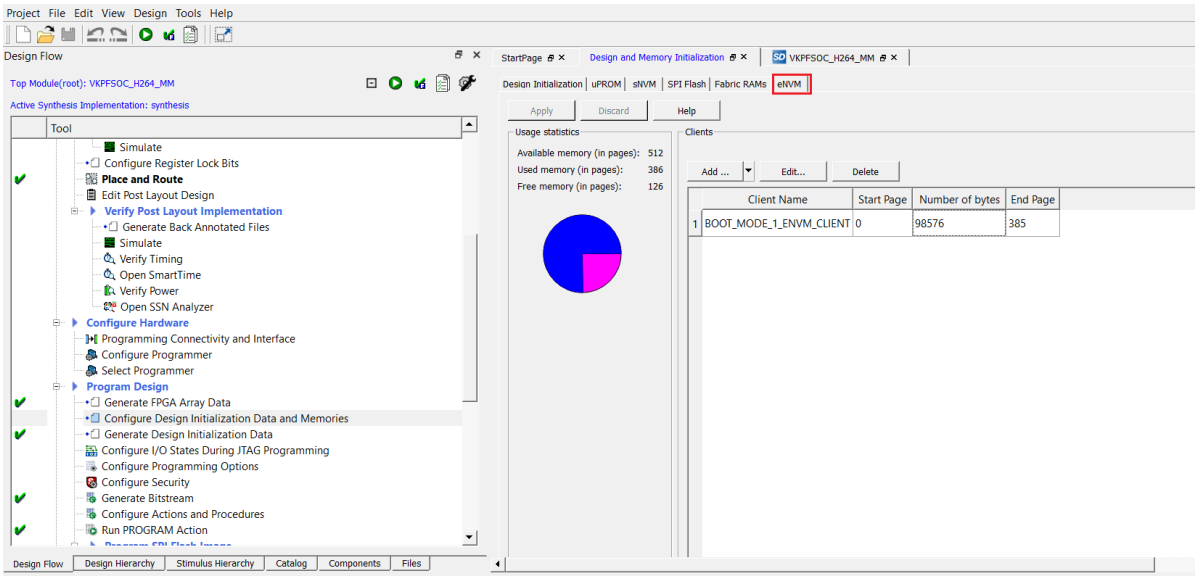
Important: In PolarFire devices, when internal RAMs are initialized from sNVM , the Design and Memory Initialization configurator limits the available sNVM size from 56 KB depending on the internal RAM size. This is due to user data initialization overheads and physical structure of the logical RAMs, which are mapped to physical blocks. If sNVM is initialized for user data storage, 56 KB of sNVM is available.

1.3. Adding eNVM Data to the Bitstream (For PolarFire SoC FPGA Only) (Ask a Question)

The eNVM is a user non-volatile flash memory that can be programmed independently. Each device has 128 KB of eNVM.

To add multiple eNVM data clients to the bitstream in Libero SoC, go to **Design Flow** > **Program Design** > **Configure Design Initialization Data and Memories**, as shown in the following figure.

Figure 1-3. Design and Memory Initialization



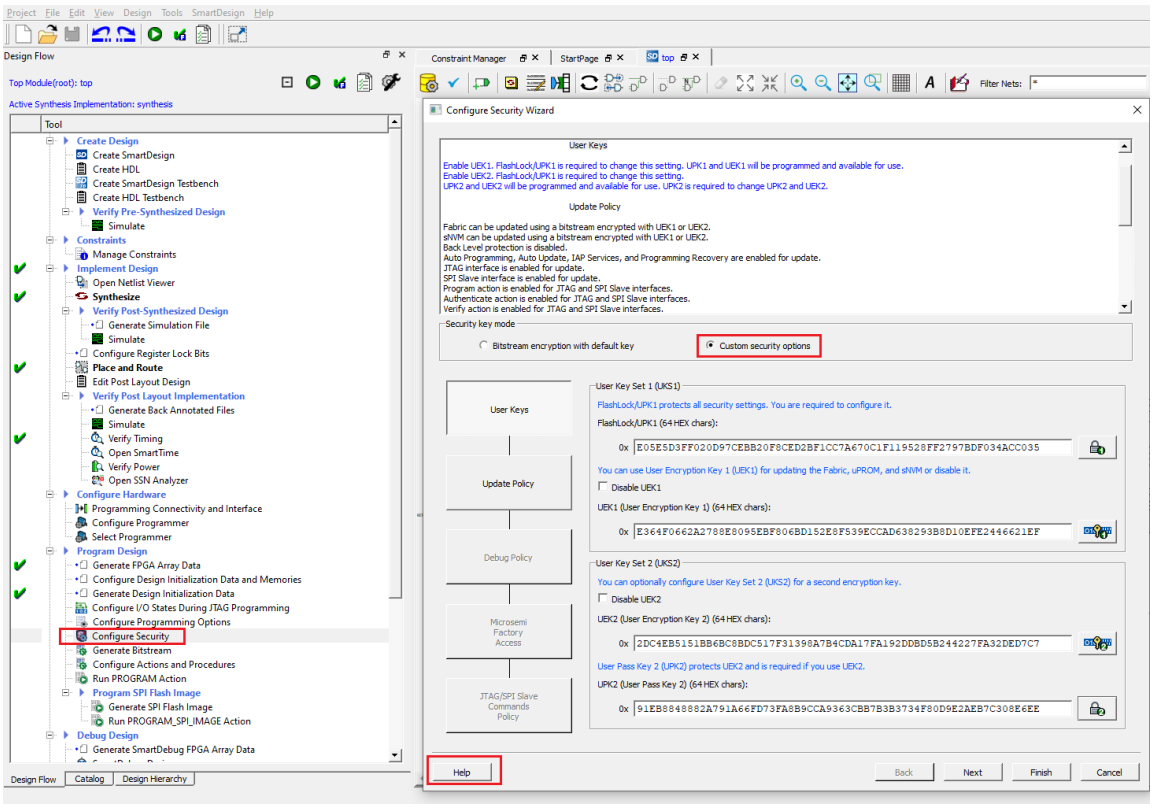
1.4. Adding User Security Settings to Bitstream (Ask a Question)

Both the device families are provisioned with a set of unique factory keys. In addition, the end users can also enroll their own security keys, thus providing complete independence from using Microchip provided keys. The user encryption key1 (UEK1) and user encryption key2 (UEK2) are user-defined AES-2 symmetric keys. Either of these keys can be used as the root key for encrypting and decrypting bitstreams, and to authenticate them.

To add user security settings in the bitstream:

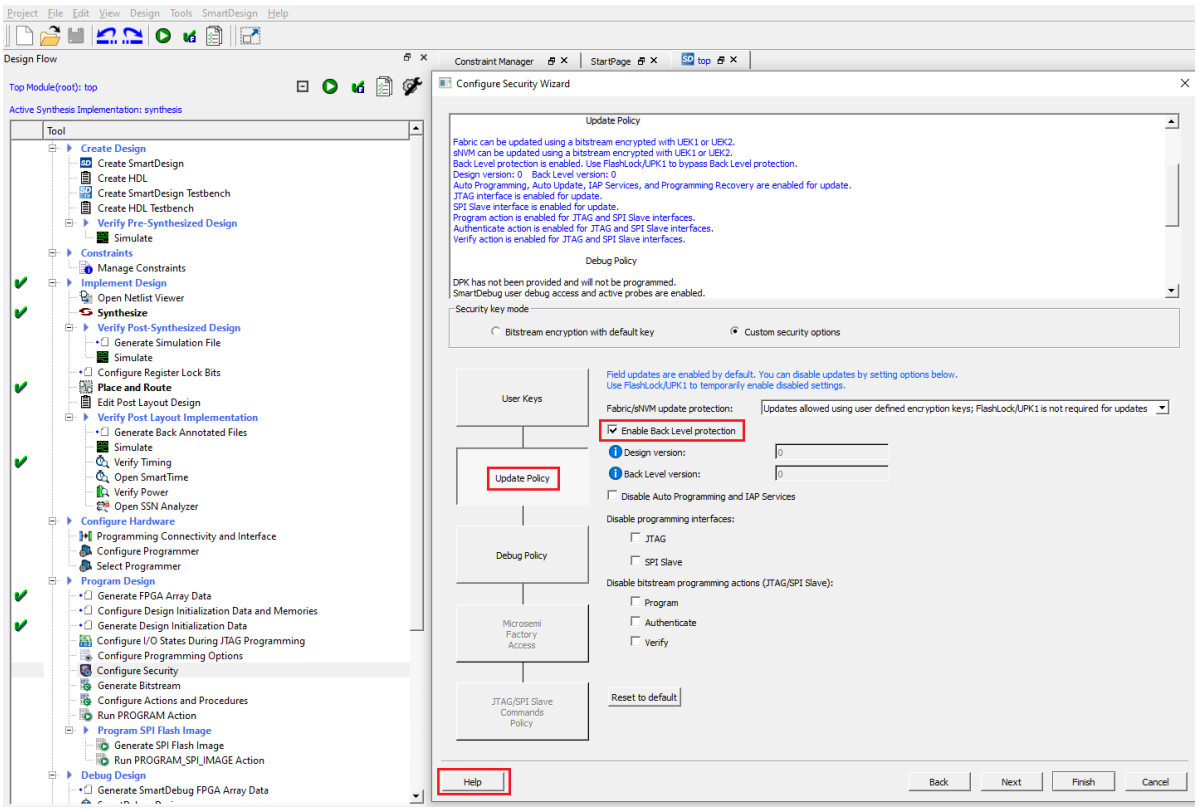
1. In Libero SoC, go to **Design Flow** > **Program Design** > **Configure Security** > **Custom security options**, as shown in the following figure.

Figure 1-4. Configure Security—Custom Security Options



2. Click **Next** to modify Update policy. The Configure Security wizard appears as shown in the following figure.

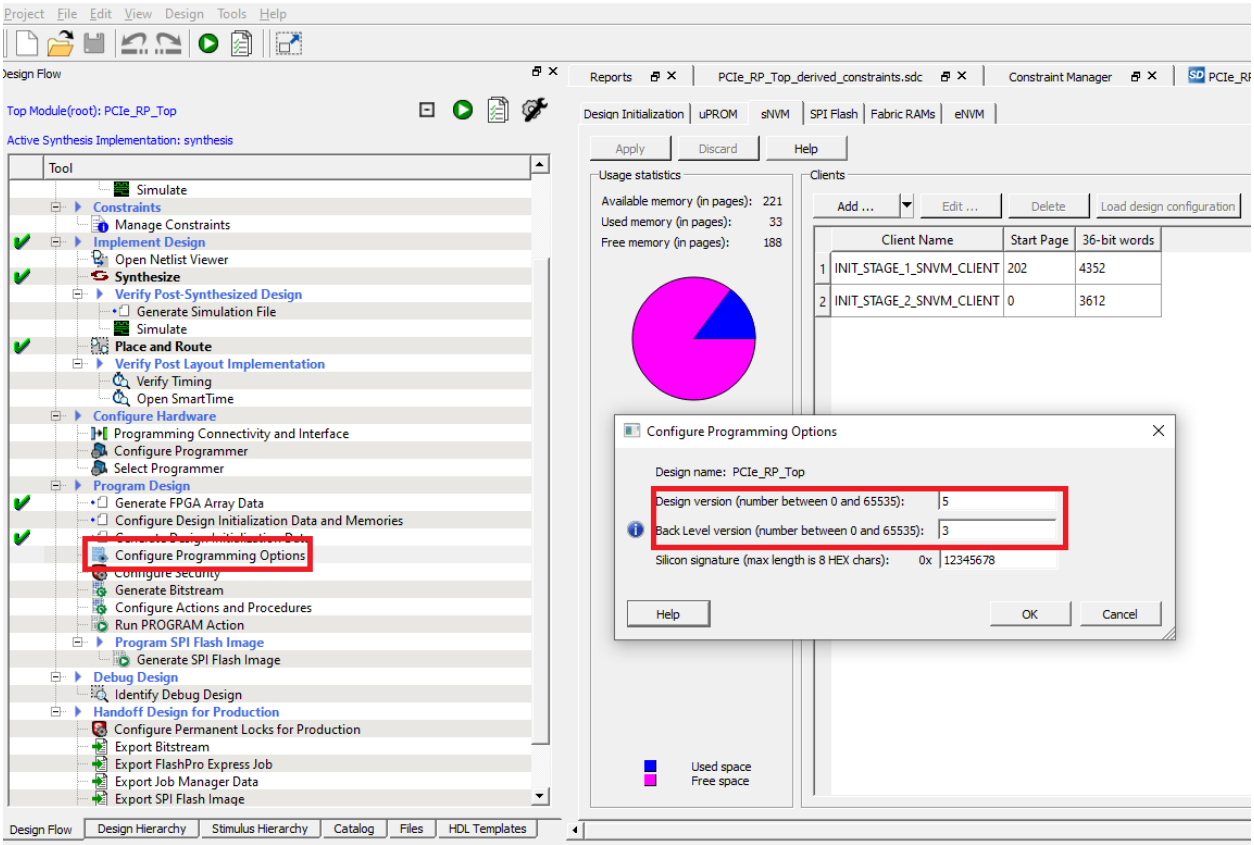
Figure 1-5. Configure Security Wizard—Update Policy



If **Back Level protection** is enabled, the **Back Level version** must be lower than the version of the design being programmed. For more information about the fields, click **Help**. The back-level version value restricts the design version that the device accepts as an update. Only (new) programming bitstreams with a Design Version strictly greater than the current Back Level Version previously stored in the device are allowed for programming. Back-level protection is secured by FlashLock/UPK1, which can be bypassed. The back level version and design version can be modified in the configure programming options tool. For more information about sNVM and security settings, see PolarFire FPGA and PolarFire SoC FPGA Security User Guide.

The following figure shows the configuration of programming options.

Figure 1-6. Configure Programming Options



For more information about the bypass back level protection, see Bypass Back Level Protection Use Case.

1.5. Configuring Bitstream Components (Ask a Question)

To configure security settings, and bitstream components such as fabric, sNVM, and eNVM (for PolarFire SoC FPGA only), follow these steps:

In Libero SoC, go to **Design Flow** > **Program Design** > **Program Design**.

1. Right-click **Generate Bitstream**, and select **Configure Options...**.
2. The **Configure Bitstream** window opens.
3. Select **Custom security**, **Fabric**, **sNVM**, and **eNVM** (for PolarFire SoC FPGA only).
4. Click **OK**.

Figure 1-7. PolarFire® FPGA Configure Bitstream Window

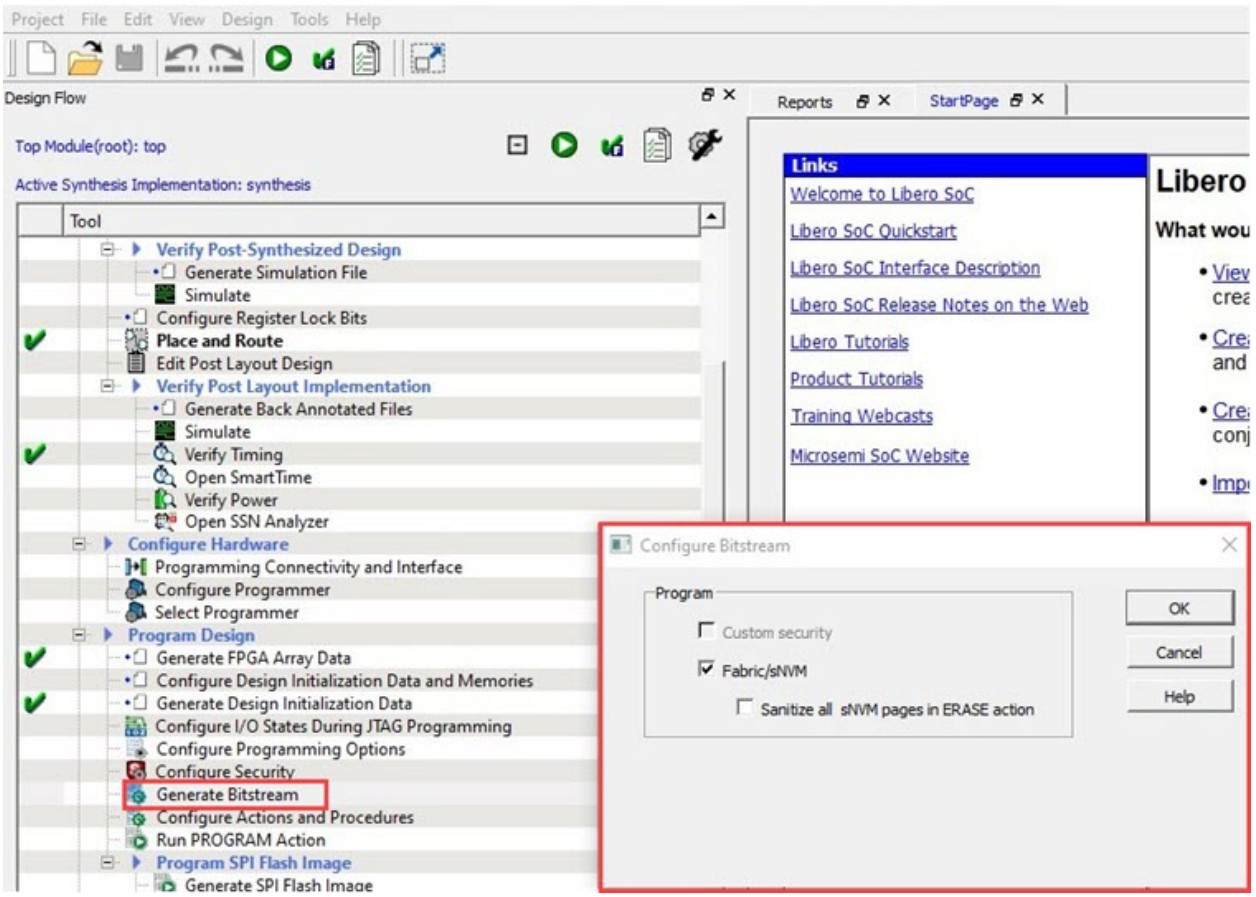
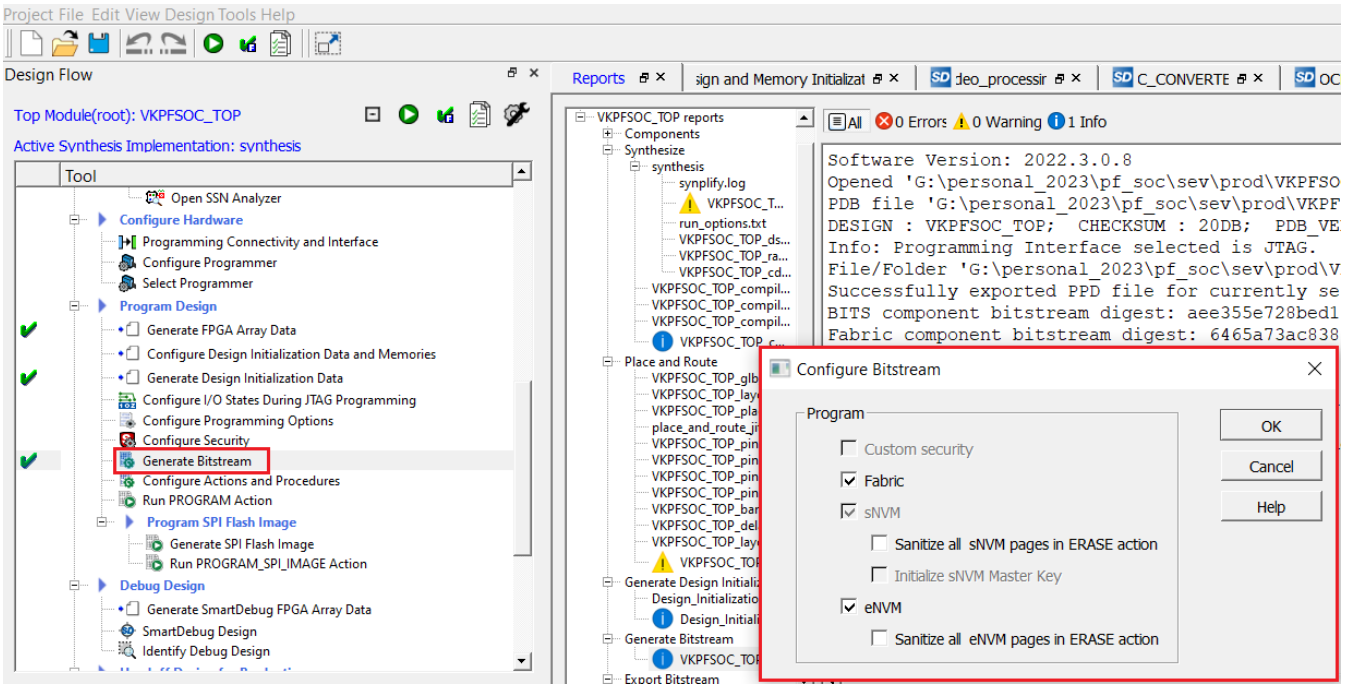


Figure 1-8. PolarFire® SoC FPGA Configure Bitstream Window



To export bitstream files, go to **Design Flow** > **Handoff Design for Production** > **Export Bitstream**.

1.6. Programming File Size (Ask a Question)

Programming files are encrypted with factory key or user key. So, the file (`.dat` or `.spi`) cannot be compressed to reduce the file size. The following table lists the PolarFire FPGA programming file sizes when custom security is disabled.

Table 1-2. PolarFire FPGA Programming Files Sizes—Custom Security Disabled

		PolarFire® FPGA				
		MPF050	MPF100	MPF200	MPF300	MPF500
Fabric and sNVM	JOB	2113 kB	4482 kB	7742 kB	11896 kB	18895 kB
	STAPL	2668 kB	5599 kB	9625 kB	14788 kB	23460 kB
	DAT	1645 kB	3497 kB	6043 kB	9307 kB	14789 kB
	SPI	1643 kB	3496 kB	6041 kB	9305 kB	14788 kB

The following tables list the PolarFire FPGA programming file sizes when custom security is enabled.

Table 1-3. PolarFire FPGA Programming Files Sizes—Custom Security Enabled

			PolarFire® FPGA				
			MPF050	MPF100	MPF200	MPF300	MPF500
Custom Security, Fabric, and sNVM	JOB	Master File	2119 kB	4487 kB	7747 kB	11901 kB	18902 kB
		UEK1/UEK2	2113 kB	4482 kB	7742 kB	11896 kB	18894 kB
	STAPL	Master File	2677 kB	5608 kB	9634 kB	14796 kB	23469 kB
		UEK1/UEK2	2669 kB	5600 kB	9626 kB	14788 kB	23460 kB
	DAT	Master File	1650 kB	3502 kB	6047 kB	9312 kB	14794 kB
		UEK1/UEK2	1645 kB	3497 kB	6043 kB	9307 kB	14789 kB
	SPI	Master File	NA	NA	NA	NA	NA
		UEK1/UEK2	1643 kB	3496 kB	6041 kB	9305 kB	14788 kB
Custom Security	JOB	Master File	16 kB	16 kB	16 kB	16 kB	16 kB
	STAPL	Master File	84 kB	84 kB	84 kB	84 kB	84 kB
	DAT	Master File	8 kB	8 kB	8 kB	8 kB	8 kB

For example, MPF200 programming file size (SPI) that contains Security, Fabric, and sNVM is 6041 Kbytes or ~ 6 MB. The following table lists examples of external SPI Flash memory densities that are required based on the number of programming images stored.

Table 1-4. PolarFire FPGA—Approximate External SPI Flash Memory Size

Number of Images	External SPI Flash Size
1	6 MB
2	12 MB
3	18 MB
4	24 MB

The following table lists the PolarFire SoC FPGA programming file sizes when custom security is disabled.

Table 1-5. PolarFire SoC FPGA Programming Files Sizes—Custom Security Disabled

		PolarFire® SoC FPGA				
		MPFS025T	MPFS095T	MPFS160T	MPFS250T	MPFS460T
Fabric, sNVM, and eNVM	JOB	3040 kB	5820 kB	8453 kB	12039 kB	20022 kB
	STAPL	3810 kB	7264 kB	10523 kB	14966 kB	24865 kB
	DAT	2365 kB	4549 kB	6609 kB	9418 kB	15674 kB
	SPI	2361 kB	4545 kB	6605 kB	9414 kB	15670 kB

The following tables list the PolarFire SoC FPGA programming file sizes when custom security is enabled.

Table 1-6. PolarFire SoC FPGA Programming Files Sizes—Custom Security Enabled

			PolarFire® SoC FPGA				
			MPFS025T	MPFS095T	MPFS160T	MPFS250T	MPFS460T
Custom Security, Fabric, sNVM, and eNVM	JOB	Master File	3040 kB	5820 kB	8452 kB	12038 kB	20023 kB
		UEK1/UEK2	3040 kB	5820 kB	8452 kB	12037 kB	20022 kB
	STAPL	Master File	3811 kB	7266 kB	10525 kB	14968 kB	24866 kB
		UEK1/UEK2	3810 kB	7264 kB	10523 kB	14967 kB	24865 kB
	DAT	Master File	2365 kB	4549 kB	6609 kB	9418 kB	15674 kB
		UEK1/UEK2	2365 kB	4549 kB	6609 kB	9418 kB	15674 kB
	SPI	Master File	NA	NA	NA	NA	NA
		UEK1/UEK2	2361 kB	4545 kB	6605 kB	9414 kB	15670 kB
Custom Security	JOB	Master File	15 kB	15 kB	16 kB	16 kB	18 kB
	STAPL	Master File	85 kB	85 kB	85 kB	86 kB	87 kB
	DAT	Master File	8 kB	8 kB	8 kB	8 kB	8 kB

Important: Master SPI file cannot be used for SPI bitstream clients and Libero will not generate the same.

Custom Security is always programmed in the Master file. The master bitstream file is used to program the FPGA at a trusted facility.

Bitstream file encrypted with UEK1 or UEK2 are used to program at an untrusted facility or for a Broadcast field update to ensure that the intellectual property contained within the bitstream is not exposed. The Security-only programming must be performed only on erased or new devices. If performed on a device with fabric programmed, the fabric is disabled after performing security-only programming. You must reprogram the fabric to re-enable it. For more information about the Security-only programming, refer to Libero SoC Documentation.

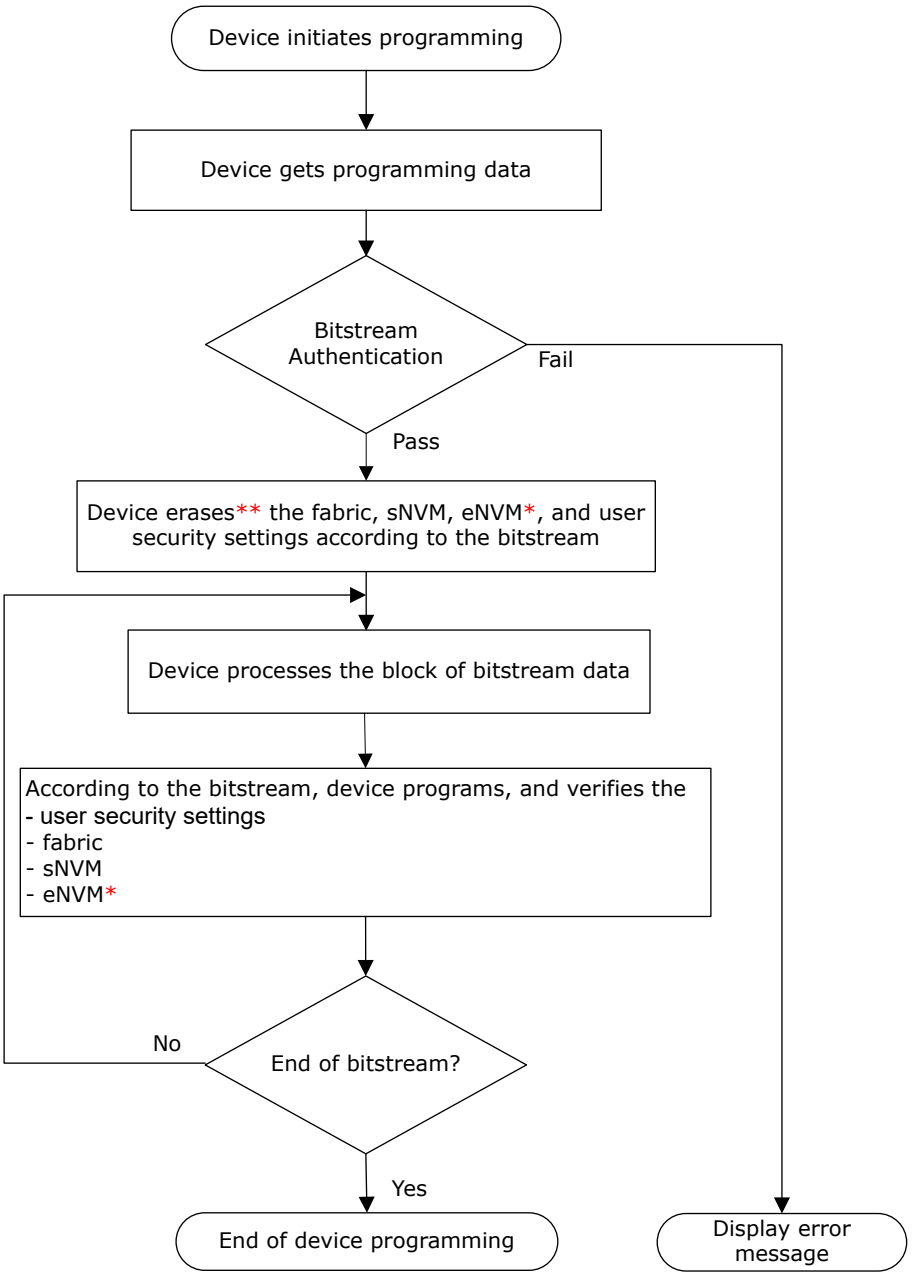
When a programming bitstream is exported by Libero SoC, digests of the selected components is printed in the Libero SoC log window and saved in a log file under the `/export` folder. These digests can be used to verify whether intended bitstream was used during programming or not. For more information, see the "Digest" section of the PolarFire Family FPGA Security User Guide.

2. Device Programming Flow (Ask a Question)

The device programming flow starts when the system controller receives or initiates device programming and ends, when the bitstream data are fully transferred and verified. The system controller fetches the bitstream data block-by-block to program the device. Authentication of the bitstream and verification of the programmed contents are part of the programming flow. The security settings are enabled either after erasing the device contents or on completion of device programming. On successful completion of programming, the system controller resets the device to run the programmed design. This programming flow is common to all the programming modes.

The following figure summarizes the device programming flow.

Figure 2-1. Programming Flow



* *Applicable for PolarFire® SoC FPGA only*

** *When bitstream is configured using **Generate Bitstream** in the Libero® SoC Design Flow, enable **Sanitize all sNVM/eNVM pages in Erase action**. This ensures sNVMs/eNVMs are erased when device is programmed.*

Important: Programming cycle count is incremented for both programming and erase operations, as erase is internally a programming scheme. For more information about programming cycle count, see the PolarFire FPGA and PolarFire SoC FPGA System Services User Guide.

2.1. Programming Time (Ask a Question)

Programming time is the time taken to erase the existing contents of the device, process bitstream data, program the device, and verify the programmed contents. The programmed content is verified as the next block of data is loaded for programming. The simultaneous programming and verification mechanism considerably reduces the total programming time.

The total programming time of both the device families is less than 60 seconds. For information about programming time for specific devices and programming modes, see respective PolarFire FPGA Datasheet or PolarFire SoC FPGA Advance Datasheet.

3. Programming Modes (Ask a Question)

This section describes the three programming modes in detail.

3.1. JTAG Programming (Ask a Question)

Both PolarFire FPGA and PolarFire SoC FPGA have a built-in JTAG controller that is compliant with the IEEE® 1149.1 and IEEE 1532 standards. The JTAG controller communicates with the system controller using a command register that sends the JTAG instruction to be executed and a 128-bit data buffer that transfers any associated data.

3.1.1. JTAG Programming Interface (Ask a Question)

In both the PolarFire FPGA and PolarFire SoC FPGA, the JTAG pins are located in a dedicated I/O bank 3 VDDI. For information about the I/O states during JTAG programming, see I/O States During Programming.

The JTAG bank voltages can be set to operate at 1.8V, 2.5V, or 3.3V. The following table lists the JTAG pins.

Table 3-1. JTAG Pins

Pin Name	Direction	Weak Pull-Up/Unused Condition	Description
TMS	Input	Yes/DNC	JTAG test mode select.
TRSTB	Input	Yes [1]	JTAG test reset. Must be held low during device operation.
TDI	Input	Yes/DNC	JTAG test data in. In ATPG or test mode, when using a 4-bit tdi bus, this I/O is used as tdi[0].
TCK	Input	No[2]	JTAG test clock
TDO	Output	No/DNC	JTAG test data out.

Important:

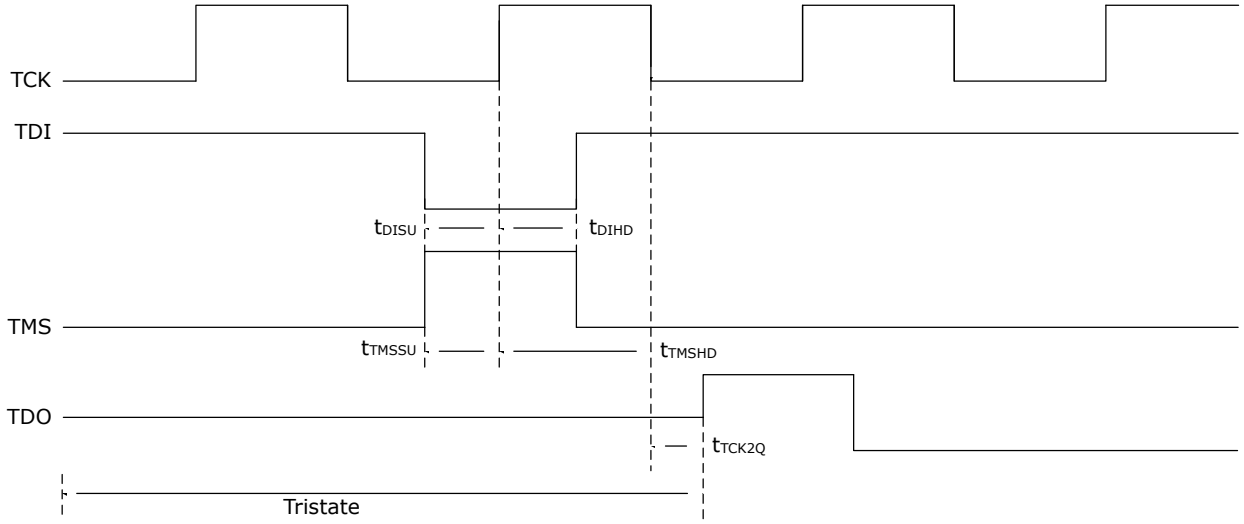
1. If TRSTB is unused and in the avionics mode, either an external 1 kΩ pull-down resistor must be connected to it to override the weak internal pull-up or it must be driven low from the external source.
2. In unused condition, must be connected to VSS through 10 kΩ resistor.

Important: A glitch can occur on GPIO pins during JTAG programming if power is disrupted. The glitch can be mitigated by powering down VDDI before VDDAUX, VDD, and VDDI3.

3.1.2. JTAG Timing (Ask a Question)

Proper operation of JTAG programming depends on the timing relationship between JTAG pins, as shown in the following figure. For recommended timing values, see JTAG switching characteristics in respective PolarFire FPGA Datasheet or PolarFire SoC FPGA Advance Datasheet.

Figure 3-1. JTAG Signals Timing Diagram

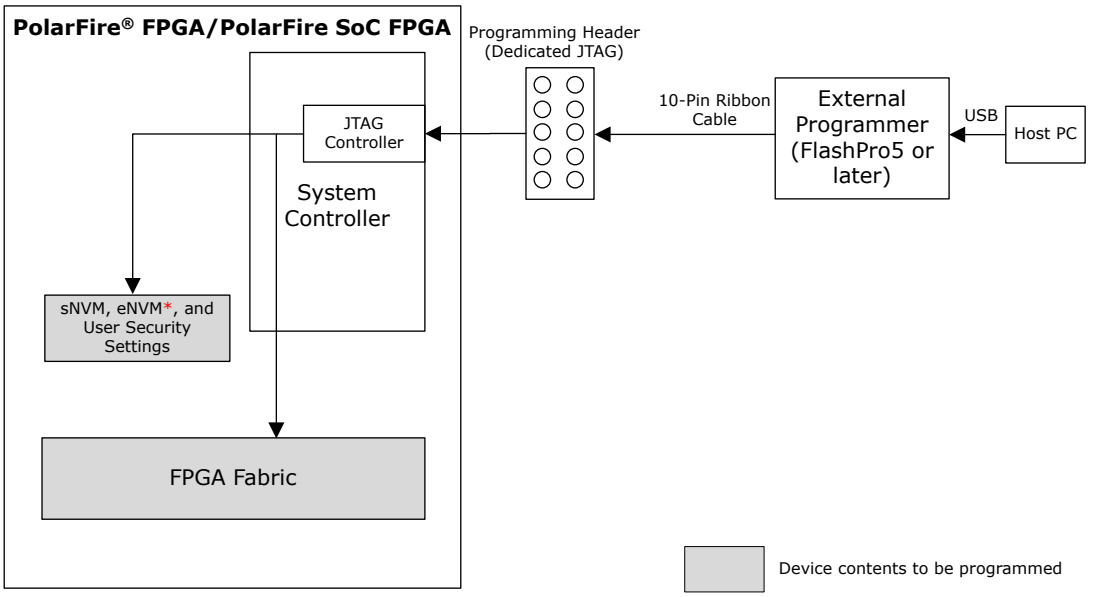


3.1.3. JTAG Programming Using FlashPro Programmer (Ask a Question)

Microchip FlashPro programmer (version 5 or later) can be used to program both the device families through the dedicated JTAG interface. This can be done either using the Libero SoC or a standalone FlashPro Express.

The FlashPro programmer connects to the device via a 10-pin programming header using a FlashPro cable (10-pin ribbon), as shown in the following figure.

Figure 3-2. JTAG Programming Using External Programmer



* *Applicable for PolarFire SoC FPGA only.*

The following table lists the FlashPro header signals.

Table 3-2. FlashPro Header Signals

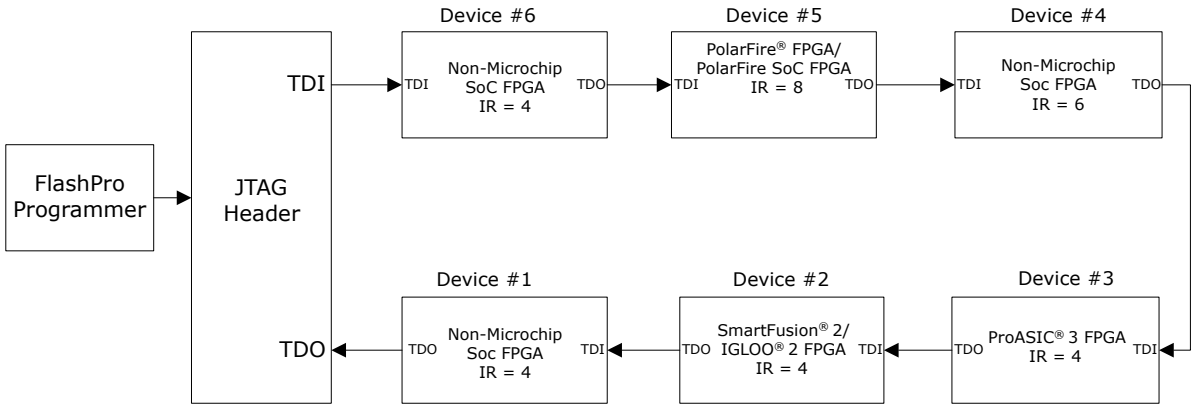
Pin Number	Signal	Direction to FlashPro Programmer	Description
1	TCK/SCK	Output	JTAG/SPI clock.
2	GND	—	Signal reference. GND pins must be connected.
3	TDO/MISO	Input	JTAG/SPI data output from programming device.
4	PROG_MODE	Not connected	Unused
5	TMS/SS	Output	JTAG test-mode select/SPI slave select.
6	VJTAG/VSPI	—	Target interface voltage input.
7	VPUMP	Not connected	Unused
8	TRSTB	Output	JTAG test reset.
9	TDI/MOSI	Output	JTAG/SPI data input to programming device.
10	GND	—	GND

A single FlashPro programmer can program multiple Microchip FPGAs from the same family or from different families in a single JTAG chain. The TDO pin of the JTAG header represents the beginning of the chain. The TDI pin of the last device connects back to the JTAG header, thus completing the JTAG chain. The following types of FPGAs can be added to a JTAG chain:

- Microchip devices targeted for programming
- Microchip bypass devices not targeted for programming
- Non-Microchip bypass devices

When a device is in bypass mode, the device's data register length is automatically set to 1 and the device stops responding to any programming instructions. To place a device in bypass mode, the Instruction Register (IR) length must be known. For Microchip FPGAs, the IR length is obtained automatically by the FlashPro Express. For non-Microchip FPGAs, the Boundary Scan Description Language (BSDL) file, which contains a sequence of boundary scan commands and data, must be loaded, or the IR length must be manually entered in the FlashPro Express. For more information about JTAG chain programming, see FlashPro User Guide.

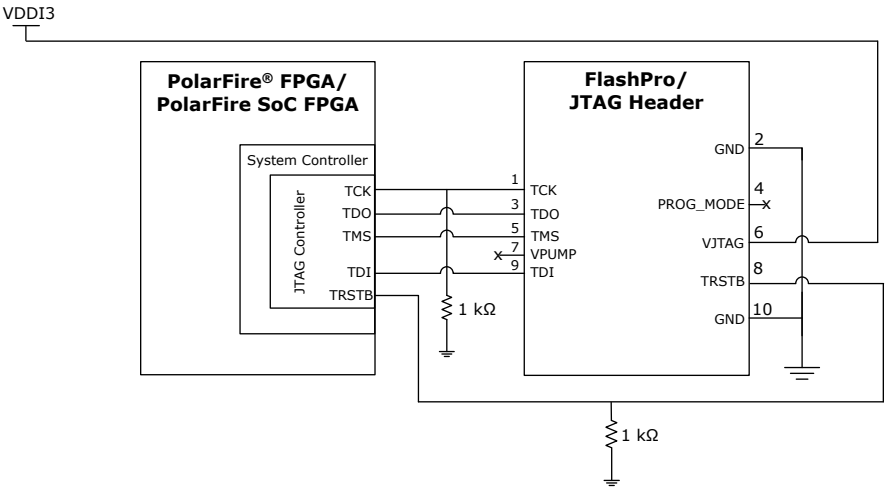
Figure 3-3. Device Programming in JTAG Chain



For information about power supply requirement and filtering capacitors, see respective UG0726: PolarFire FPGA Board Design User Guide or PolarFire SoC Board Design Guidelines User Guide.

The following figure shows the connections between the programming header and the device.

Figure 3-4. Connecting FlashPro Programmer to a Device



3.1.4. JTAG Programming Using External Microprocessor (Ask a Question)

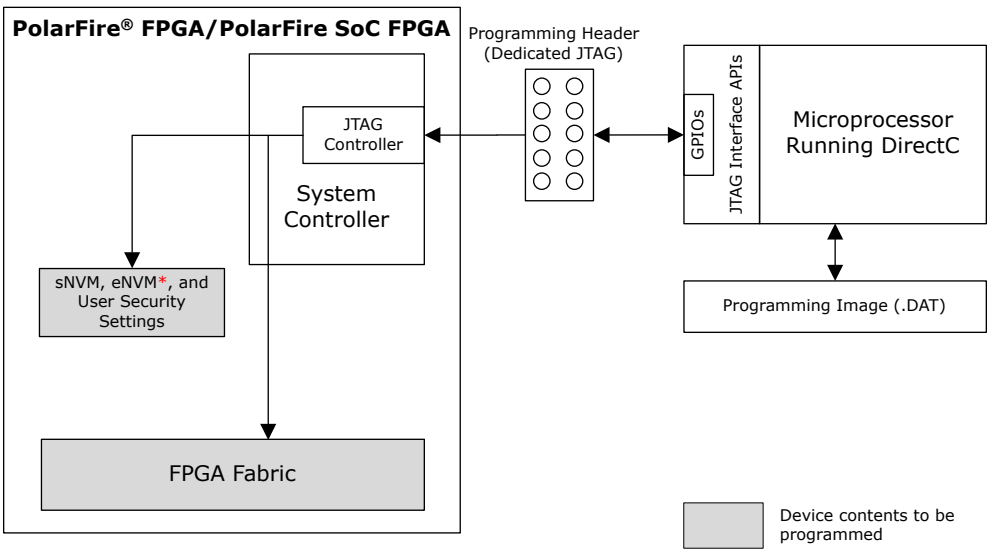
An external microprocessor can be used to program the device through the dedicated JTAG interface. This type of programming requires that the external microprocessor run DirectC, a Microchip programming solution for FPGAs, and the microprocessor's GPIO ports drive the JTAG interface.

Note: The DirectC solution supports programming of the FPGA fabric, sNVM, eNVM (for PolarFire SoC FPGA only), and user security settings. DirectC is used by adding the necessary APIs and compiling the source code to create a binary executable. The binary executable is downloaded to the external microprocessor along with the programming data file. For more information, see the latest version of the *DirectC User Guide* available on the DirectC solution webpage.

Security only bitstream must be programmed only on erased or blank devices. If the security bitstream is used to program a previously programmed FPGA, it disables the FPGA Array. The fabric must be re-programmed to enable it.

The following figure shows a sample implementation of device programming using an external microprocessor running DirectC.

Figure 3-5. Programming Using External Microprocessor



* *Applicable for PolarFire SoC FPGA only.*

3.2. SPI Slave Programming (Ask a Question)

Both the device families can be programmed using an external SPI master, such as an external microprocessor or a FlashPro programmer through the SPI interface. See Table 3-4 for the pin settings that must be used to configure the system controller SPI in slave mode.

The SPI slave or master mode is determined by IO_CFG_INTF SPI pin at device Power-on-Reset (POR) and cannot be switched dynamically. A power cycle or DEVRST is required to change the SPI configuration from Slave to Master or vice-versa by configuring the IO_CFG_INTF pin, as mentioned in Table 3-3.

When SPI is in Slave mode, fabric has no access to SPI and the SPI interface is dedicated to the system controller.

Design initialization from an external SPI flash is not supported when the device is in SPI slave programming mode. For information about design initialization, see PolarFire FPGA and PolarFire SoC FPGA Power-up and Reset User Guide.

3.2.1. SPI Slave Programming Interface (Ask a Question)

In addition to the standard SPI signals, both the device families provide two pins—SPI_EN and IO_CFG_INTF—for configuring the SPI controller.

The following table lists the system controller's SPI pins and specifies what must be done if a pin is not in use (unused condition). For information about unused conditions and power sequence, see respective UG0726: PolarFire FPGA Board Design User Guide or PolarFire SoC FPGA Board Design Guidelines User Guide.

Table 3-3. System Controller SPI Pins

SPI Pin Name	Direction	Description	Unused Condition
SCK	Bidirectional	SPI clock[1]	Connect to VSS through a 10 kΩ resistor

Table 3-3. System Controller SPI Pins (continued)

SPI Pin Name	Direction	Description	Unused Condition
SS[2]	Bidirectional	SPI slave select[1]	Connect to VSS through a 10 kΩ resistor
SDI	Input	SDI input[1]	Connect to VDDI3 through a 10 kΩ resistor
SDO	Output	SDO output[1]	DNC
SPI_EN	Input	SPI enable 0: SPI output tristated 1: Enabled Pulled up or down through a resistor or driven dynamically from an external source to enable or tristate the SPI I/O.	Connect to VSS through a 10 kΩ resistor
IO_CFG_INTF	Input	SPI I/O configuration 0: SPI slave interface 1: SPI master interface Pulled up or down through a resistor.	Connect to VSS through a 10 kΩ resistor

Important:

1. Shared between the system controller and the FPGA fabric/MSS (for PolarFire SoC FPGA only). 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 (after device power-up)/MSS (for PolarFire SoC FPGA only). When the Suspend mode is enabled for a device, the system controller hands over the control of SPI to the fabric (during the power-up and initilization) before goes into the Suspend mode. When the SPI_EN pin is disabled (driven low) or when SS is driven high, the SPI outputs of system controller are tristated.
2. The system controller SS pin is an active-low signal. In unused condition, the pin must be tied to VSS to avoid a floating pin on the device.

The SPI_EN and IO_CFG_INTF pins must be configured external to the device. This can be done by using jumpers on the board or by bootstrapping. The following table lists the SPI_EN and IO_CFG_INTF configuration for SPI slave programming.

Table 3-4. System Controller's SPI Configuration - SPI Slave

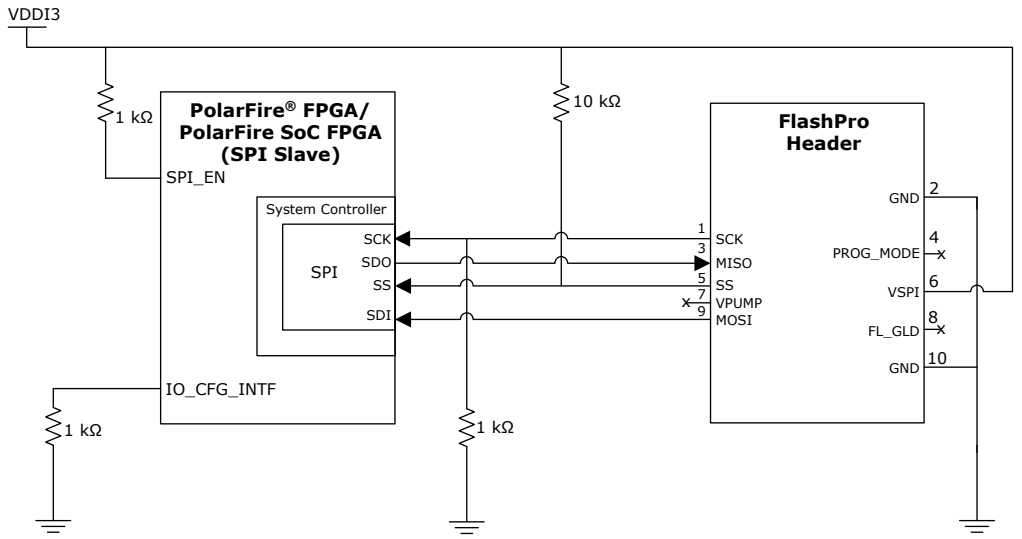
SPI Pins		SPI Slave Programming	Description
SPI_EN	IO_CFG_INTF		
0	x	No	Dynamic switching from slave to master or vice-versa is not allowed. A power-cycle or device reset (DEVRST_N) is required to change the SPI configuration from Slave to Master or vice-versa by configuring the IO_CFG_INTF pin.
1	0 (SPI slave mode)	Yes	
1	1 (SPI master mode)	No	

3.2.2. SPI Slave Programming Using FlashPro Programmer (Ask a Question)

Microchip FlashPro programmer (version 5 or later) can be used to program device through the dedicated SPI. This can be done using either the Libero SoC or a standalone FlashPro Express. The FlashPro programmer is connected to the device SPI ports, as shown in the following figure.

The target board must provide power to the VDD, VDD18, VDD25, and VDDI3.

Figure 3-6. SPI Slave Programming Using External Programmer



Device Programming using SPI Slave can be selected in **Libero SoC Design Flow** > **Configure Hardware** > **Programming Connectivity and Interface**.

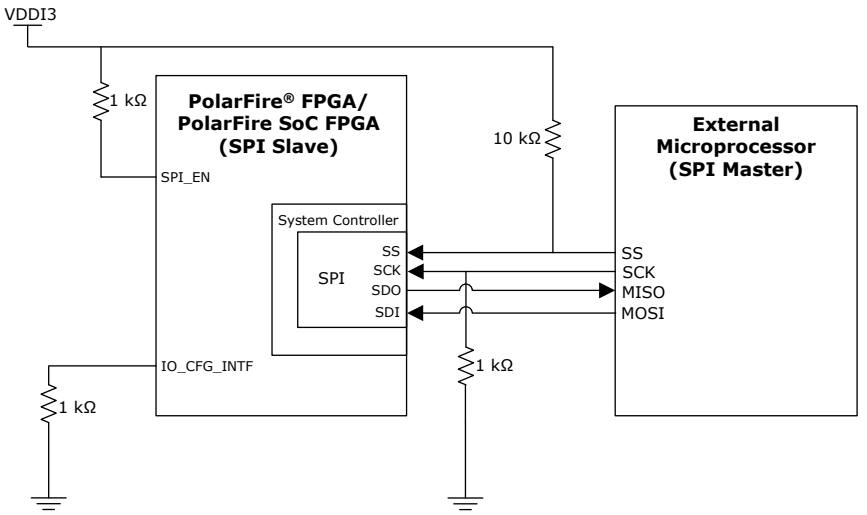
3.2.3. SPI Slave Programming Using External Microprocessor (Ask a Question)

An external microprocessor (such as a host PC or another Microchip FPGA) can be used to program the device through the dedicated SPI port, as shown in the following figure. This type of programming requires that the external microprocessor run the Microchip SPI-DirectC solution. The external microprocessor can also control the SPI_EN, IO_CNF_INTF, and DEVRST_N pins to program the device.

SPI-DirectC supports programming of the FPGA fabric, sNVM, eNVM (for PolarFire SoC FPGA only), and user security settings. SPI-DirectC is used by adding the necessary APIs and compiling the source code to create a binary executable. The binary executable is downloaded to the external microprocessor along with the programming data file. For more information, see the latest version of the *SPI-DirectC User Guide* available on the DirectC solution webpage. The example project (Direct-C installer) is also available on the **Downloads** tab.

For information about FlashPro header signals, see Table 3-2.

Figure 3-7. SPI Slave Programming Using External Microprocessor



3.3. SPI Master Programming (Ask a Question)

When the system controller SPI is configured as a master, a device can program itself. In SPI master programming, the programming images are stored in the external SPI flash memory using the SPI directory. For more information about the SPI directory and about programming the external SPI flash memory, see Programming the External SPI Flash.

SPI master programming supports auto update and IAP. In auto update programming, if the version of the update image is different from the currently programmed version, the system controller reads the update image bitstream from the external SPI flash memory and programs the device on power-up. In IAP, the user application initiates the device program, and the system controller reads the bitstream from the external SPI flash memory to program the device. The auto update and IAP operations are atomic and cannot be interrupted by JTAG or SPI slave commands.

The Auto Update feature is not enabled by default and if required, this needs to be enabled using Libero SoC. SPI Master mode also supports Auto Programming and Auto Recovery, see Table 3-5. These two features are enabled by default and do not require user configuration.

For information about the I/O states during SPI master programming, see I/O States During Programming.

The following table lists the initiation sources for the features supported by SPI master programming.

Table 3-5. Device Program Initiation Sources

Programming Feature	Description	Initiation Source
Auto programming	Programs a blank device	Device reset or power-cycle
Auto update	Updates device contents automatically	Device reset, power-cycle, or system service request
IAP	Updates device contents upon user request	System service request
Auto recovery	Automatically recovers the device from programming failure	Device power failure during programming

Field updates performed in unpredictable conditions carry associated risks of programming failures. For example, risks include a power brownout/outage event or internal system controller Single Event Upsets (SEUs) when re-programmed in high altitude applications. The following table lists IAP/auto update recommendations to minimize these risks for various bitstream components when performing field updates to PolarFire or PolarFire SoC devices. For information about SEU rates, see PolarFire Radiation Test reports.

Table 3-6. Recommendations for Field Updates (IAP and Auto Update)

Device	eNVM Only Bitstream (For PolarFire® SoC FPGA Only)	sNVM Only Bitstream	Security Only Bitstream	Fabric and eNVM or sNVM Bitstream
PolarFire	N/A	N/A	No[1]	Yes
PolarFire SoC	IAP: No [2] Auto Update: Yes [3]	IAP: No [2] Auto Update: Yes [3]	No[1]	Yes

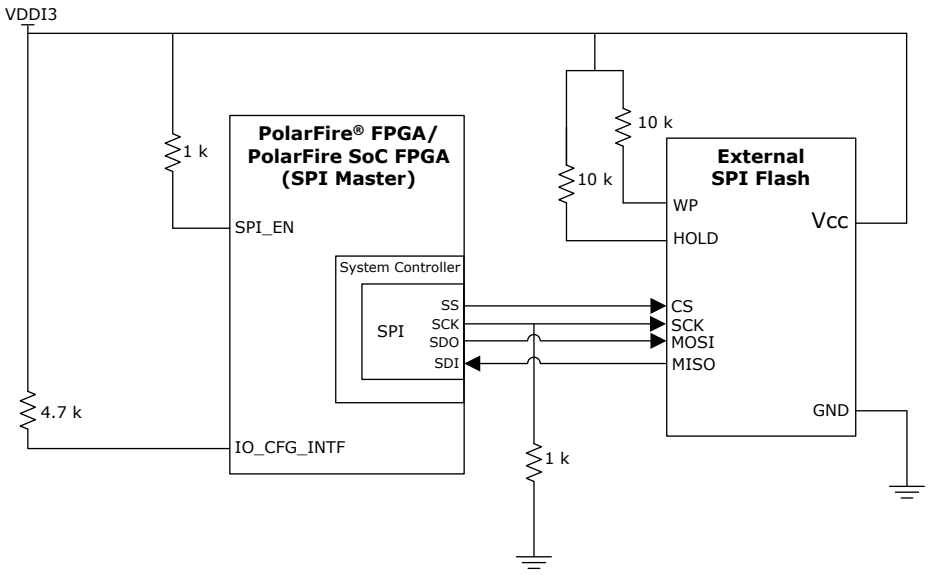
Important:

1. Reprogramming security settings, where power interruption or SEUs are possible, is not recommended to prevent the risk of accidental device lockout. Device security settings and locks must be programmed in a terrestrial environment. The ability to perform field updates of user cryptographic keys will be provided in a future tool release to enable **key rolling** functionality.
2. Power interruption or brown-out during IAP programming of eNVM or sNVM component bitstreams can corrupt eNVM or sNVM and prevent subsequent device boot.
3. Supported only by bitstreams generated using LiberoSoC v2023.1 and later.

For information about implementing Auto update and IAP, see PolarFire FPGA Auto Update and In-Application Programming Application Note.

The following figure shows the recommended board configuration for SPI master programming. The VDDI3 must match the voltage specified in the datasheet associated with the external SPI flash.

Figure 3-8. Recommended Board Configuration for SPI Master Programming



3.3.1. SPI Master Programming Interface (Ask a Question)

The SPI_EN and IO_CFG_INTF pins must be configured external to the device by using jumpers on the board or by bootstrapping. The following table provides the SPI_EN and IO_CFG_INTF pin configuration details for SPI master programming.

Table 3-7. System Controller's SPI Configuration—SPI Master

SPI Pins		SPI Master Programming	
SPI_EN	IO_CFG_INTF	IAP	Auto Update
0	x	No	No
1	0 (SPI slave mode)	No	No
1	1 (SPI master mode)	Yes	Yes

The SPI_EN input must be asserted on device power-up or reset for the SPI Master programming. If the input is sensed low during device power-up, the SPI controller is not available for FPGA fabric.

3.3.1.1. System Controller SPI Mode and Clock (Ask a Question)

The system controller SPI operates in data transfer mode 3 (SPI mode 3) for SPI flash read operations. Both the clock polarity (SPO/CPOL) and clock phase (SPH/CPHA) for this data transfer mode must be set to HIGH. The system controller's SPI operates at a fixed clock of 20 MHz.

3.3.2. System Services (Ask a Question)

Both PolarFire FPGA and PolarFire SoC FPGA devices include a System Controller, which accepts and responds to system service requests from the user.

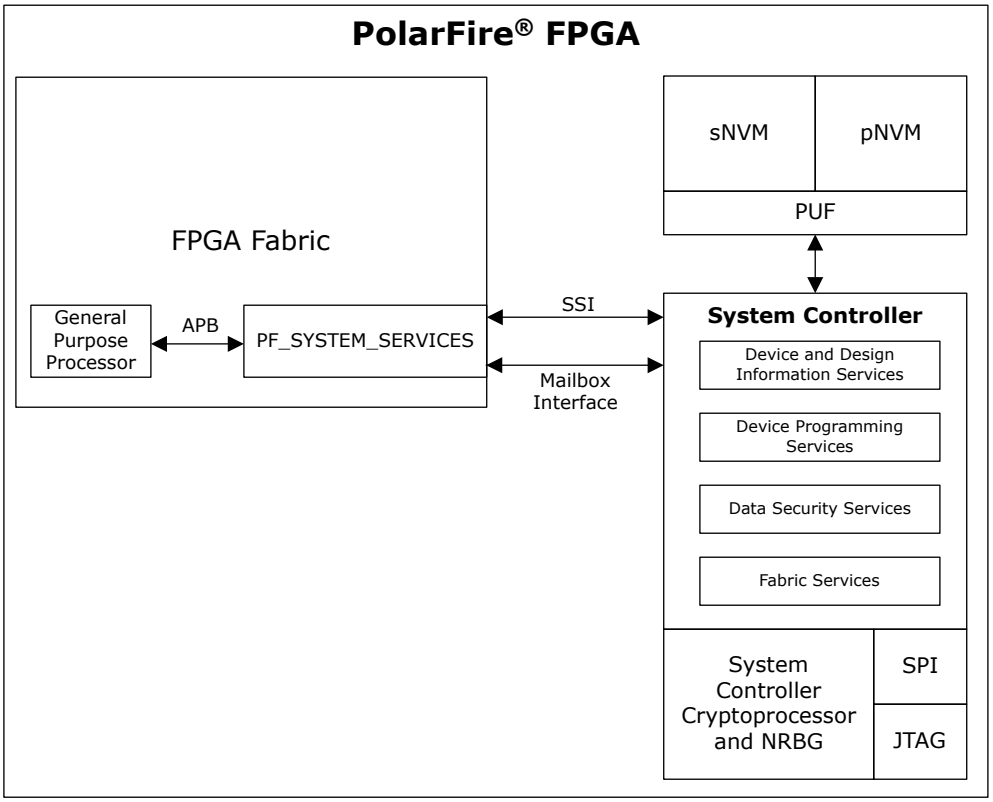
The user application can initiate the following programming related system services:

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

3.3.2.1. PolarFire FPGA System Services (Ask a Question)

In PolarFire FPGA, system services are system controller actions initiated by the fabric user logic through the system controller's System Service Interface (SSI). For initiating the system services, the fabric user logic requires the PF_SYSTEM_SERVICES SgCore IP available in the Libero catalog. The following figure shows the design interface between fabric and System Controller.

Figure 3-9. Design Interface Between Fabric and System Controller

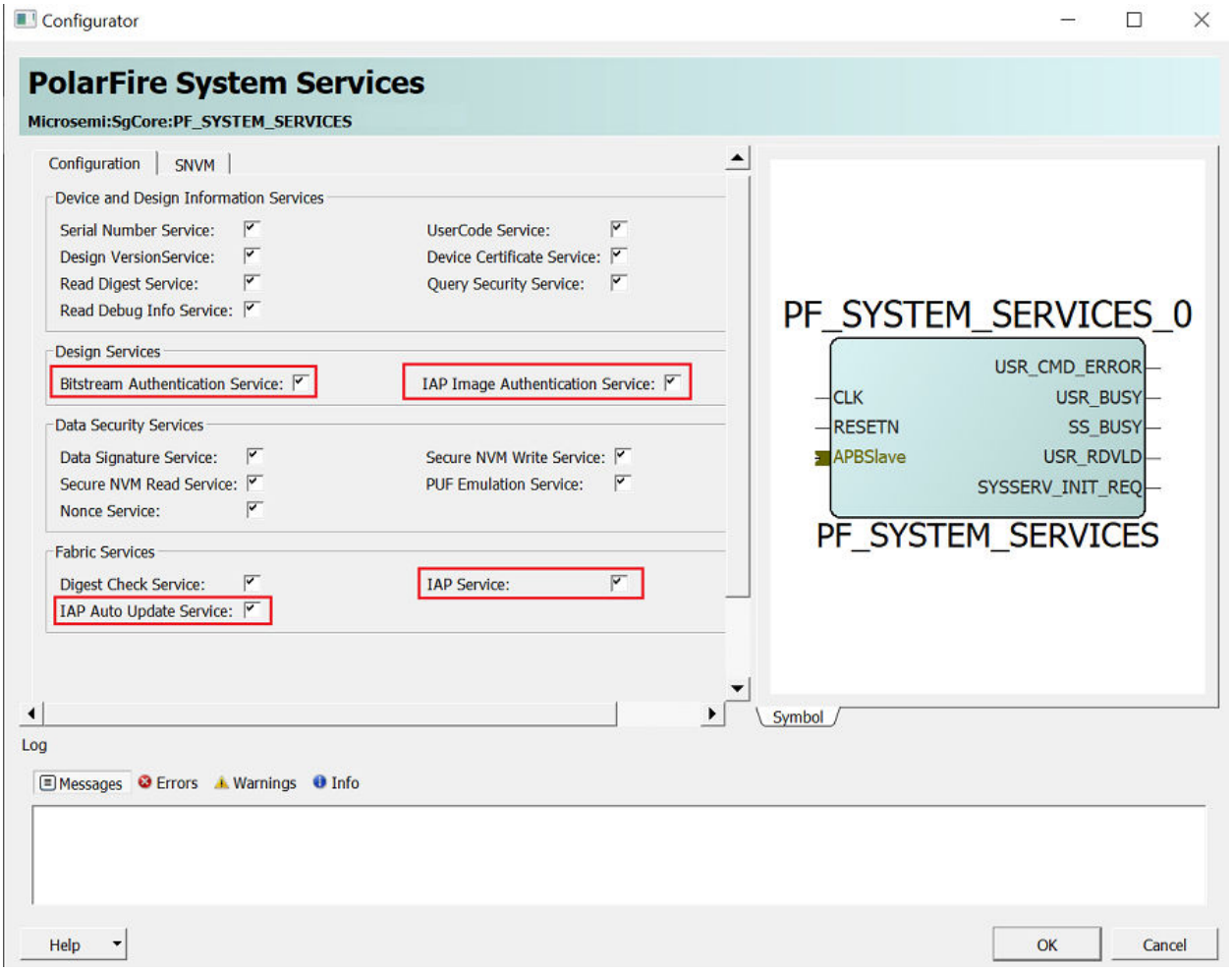


For information about PolarFire FPGA system services driver and example SoftConsole project, see *Firmware Catalog*, which is available in the Libero SoC installation package.

3.3.2.1.1. PolarFire System Services Configurator (Ask a Question)

The following figure shows the PolarFire System Services Configurator.

Figure 3-10. PolarFire® FPGA Core System Services Configurator

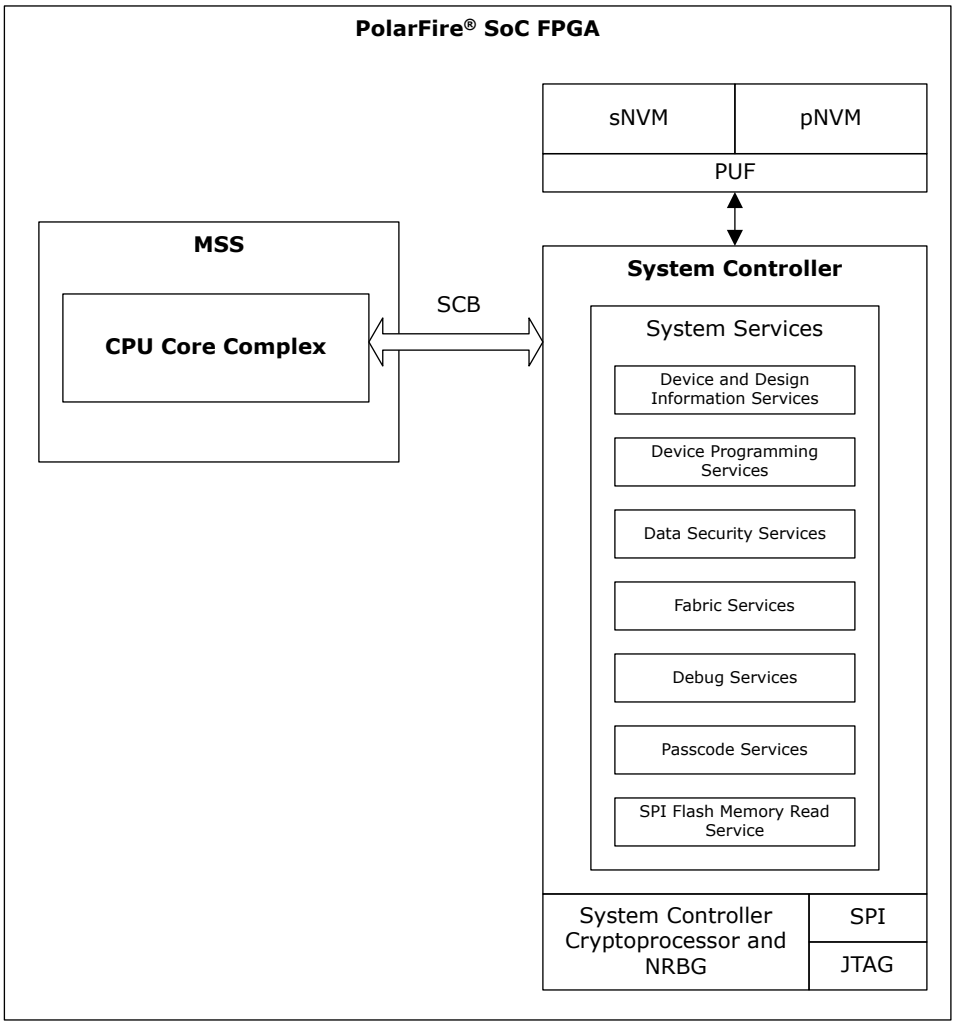


The fabric master is connected to the PF_SYSTEM_SERVICES core using the APB interface. The PF_SYSTEM_SERVICES core can be configured using the PolarFire System Services configurator in Libero SoC, as shown in Figure 3-10. For more information, see PolarFire System Services SgCore User Guide.

3.3.2.2. PolarFire SoC FPGA System Services (Ask a Question)

In PolarFire SoC FPGA, system services are System Controller actions initiated by PolarFire SoC MSS. MSS communicates with the System Controller over System Controller Bridge (SCB) bus. The following figure shows the design interface between MSS and System Controller.

Figure 3-11. Design Interface Between MSS and System Controller



For information about PolarFire SoC FPGA MSS system services driver and example SoftConsole project, see GitHub.

3.3.2.3. System Service Request (Ask a Question)

In both PolarFire FPGA and PolarFire SoC FPGA, the system service request is initiated by passing a 16-bit system service descriptor to the System Controller. The lower seven bits of the descriptor specify the service to be performed and the upper nine bits specify address offset. There is a 2 KB internal mailbox RAM memory space. This space is used for passing the input data and storing the service request output that is returned by the System Controller. The mailbox address specifies the service-specific data structure that is used for any additional inputs to or outputs from the service. On completion of service, the System Controller writes a status code indicating the successful completion of the system service or an error code. The following table lists the system service request descriptor bits. For information about mailbox read/write communication from Fabric, see PolarFire System Services SgCore User Guide.

Table 3-8. PolarFire® FPGA and PolarFire SoC FPGA System Service Request Descriptor

System Service Descriptor Bit Field	Value	Description
15:7	MBOXADDR[10:2]	Specifies the address offset in mailbox RAM to access minimum four bytes of memory. Mailbox addresses are specified using a word offset (0–511).
6:0	SERVICECMD	Service command for System Controller to execute the request.

For more information about system services, see PolarFire FPGA and PolarFire SoC FPGA System Services User Guide.

3.3.2.4. Bitstream and IAP Image Authentication System Services (Ask a Question)

For security and reliability reasons, the programming bitstream must be authenticated and validated before the device is programmed. Successful authentication of the bitstream prevents auto recovery. While the authentication is in progress, the fabric user logic in PolarFire FPGA and MSS user application in PolarFire SoC FPGA continues to operate normally, though without access to SPI flash and system services. Before the device is programmed using auto update or IAP, the user application can run the authentication system service.

Note: If the bitstream authentication system service is initiated while a new bitstream is being loaded through the JTAG interface, the system service takes precedence, and the JTAG operation fails.

3.3.2.4.1. Bitstream Authentication System Service (Ask a Question)

The bitstream authentication system service parses a bitstream image stored in the SPI flash and verifies the integrity of the bitstream. The following table lists the fields in a bitstream authentication service request.

Table 3-9. Bitstream Authentication Service Request

System Service Descriptor Bit Field	Value	Description
15:7	MBOXADDR[10:2]	Mailbox address. For the format, see Table 3-10.
6:0	23H	Bitstream authentication command code.

The following table describes the bitstream authentication service mailbox format.

Table 3-10. Bitstream Authentication Service Mailbox Format

Offset	Length (bytes)	Parameter	Direction	Description
0	4	SPIADDR	Input	Address of the bitstream in SPI flash. If the external SPI flash device does not support 32-bit addresses, SPIADDR[31:24] is ignored.

3.3.2.4.2. IAP Image Authentication System Service (Ask a Question)

The IAP image authentication system service parses an image stored in the SPI flash and verifies the integrity of the image descriptor, bitstream, and design initialization data.

The following table lists the fields in an IAP image authentication service request.

Table 3-11. IAP Image Authentication Service Request

System Service Descriptor Bit Field	Value	Description
15	—	Reserved
14:7	IMAGEID[7:0]	Identifies the image index in the SPI directory for image authentication.
6:0	22H	Authenticates image command.

3.3.2.4.3. Authentication Service Status Codes (Ask a Question)

If bitstream authentication or IAP image authentication is successful, the status code 0 is generated. If bitstream authentication or IAP image authentication fails, an 8-bit error code is generated. For the detailed information about error codes, see Appendix: Error Codes.

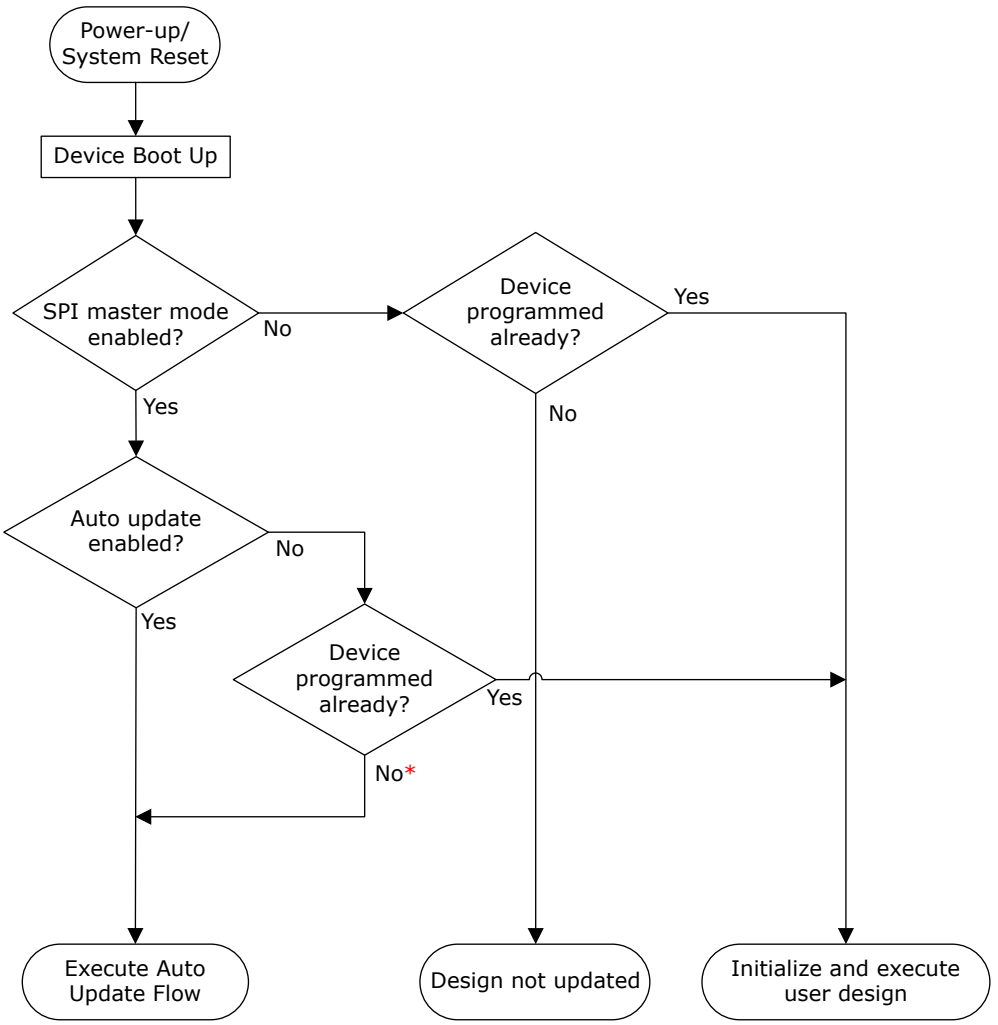
3.3.2.4.4. Usage of Authentication System Services (Ask a Question)

The programming image contains the image descriptor, bitstream, and optional design initialization data. The bitstream authentication system service can be used to authenticate the bitstream only. The IAP image authentication system service, however, can be used to authenticate the entire programming image, including the image descriptor, bitstream, and optional design initialization data.

3.3.3. Auto Update (Ask a Question)

For auto update to occur, the auto update feature needs to be enabled in the user design. On power-up, the device selects the newer version of the first two images stored in the SPI directory. If the version of the newer image does not match that of the currently programmed image, then auto update occurs. The following figure shows the high-level flow of auto update programming.

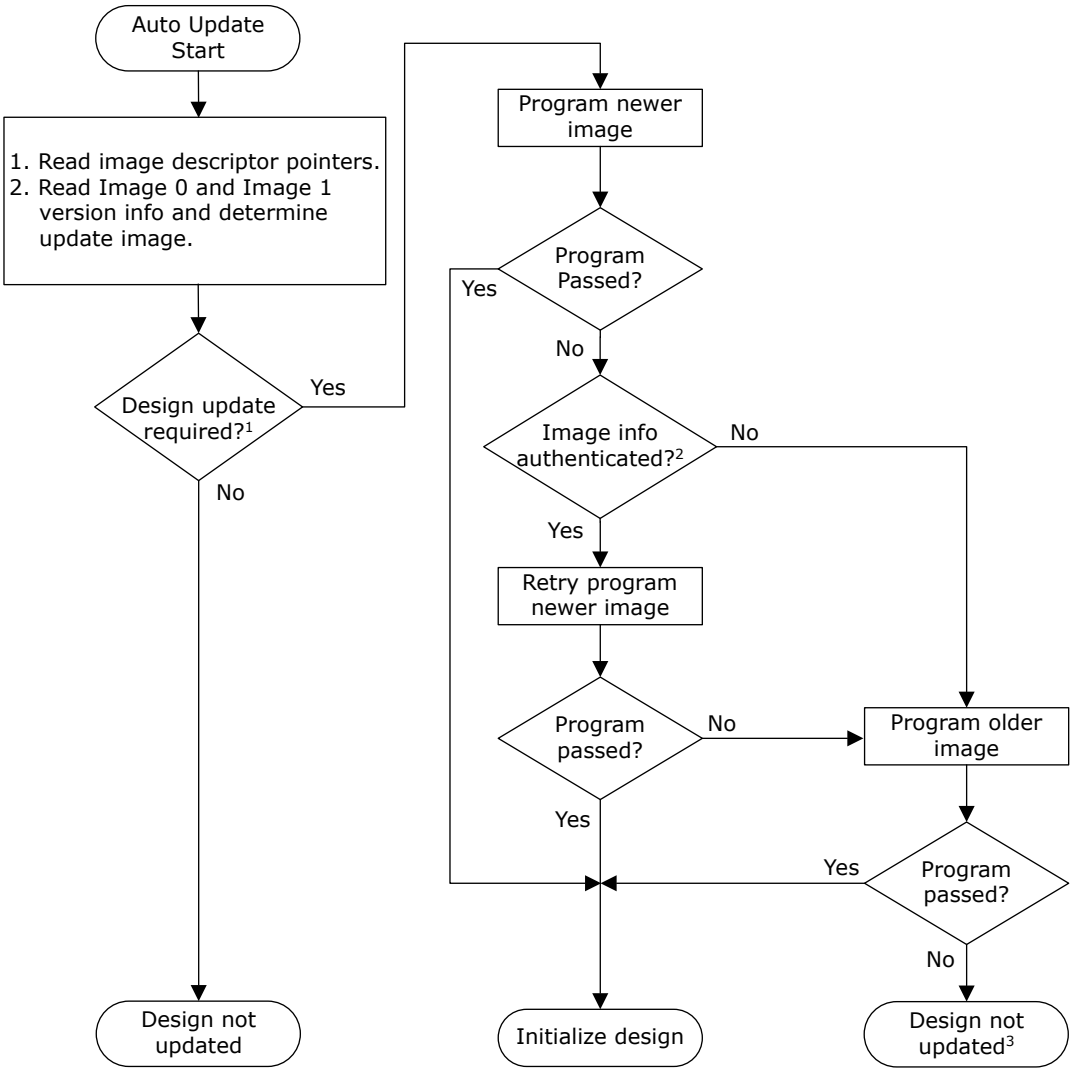
Figure 3-12. Auto Update High-Level Flowchart



Different scenarios to reach here:
- *Device is blank and auto update is initiated to program the device*
- *As part of IAP recovery when power fails during IAP or partially programmed with an invalid image*
- *As part of auto update recovery when power fails during auto update*

The following figure shows the detailed flow of auto update programming.

Figure 3-13. Auto Update Detailed Flow



1. Condition for update: version of the design differs from the update image or the device is blank.
2. Device checks only BITS (starting bits of the bitstream) and AUTH (encryption keys information) components of the bitstream as part of the programming.
3. The device is not programmed, and user intervention is required.

The following table lists example auto update conditions when different image versions are available in the SPI flash.

Table 3-12. Example Auto Update Conditions

Version Running on the Device	First Two Image Versions Available in SPI Flash	Back Level Protection	Image Version Selected for Auto Update
Blank device	2, 3	Disabled	3
3	2, 3	Disabled	No auto update
3	1, 2	Disabled	2
2	1, 2	Disabled	No auto update
1	1, 2	Disabled	2
2	3, 4	Enabled and set to 4	No auto update
3	3, 5	Enabled and set to 4	5
2	3, 5	Enabled and set to 4	5
5	2, 3	Enabled and set to 4	No auto update

3.3.3.1. Auto Update on a Blank Device (Auto Programming) (Ask a Question)

When a blank device is powered up or reset (with SPI master mode enabled), the device programs itself using the newest version of the image. This process is known as auto programming.

When the device is blank and programmed using the auto programming method with security-enabled bitstream, subsequent programming can only be done using a custom security-enabled bitstream file (UEK1/UEK2). For more information about generating security enabled bitstream, see Adding User Security Settings to Bitstream.

3.3.3.2. Auto Update on a Pre-programmed Device (Ask a Question)

Auto update is also initiated through system services on a pre-programmed device. If the device is preprogrammed, it compares the update image with the currently programmed image. If the version of the update image is found to be different from the currently programmed version, auto update programming is initiated.

To perform auto update on a preprogrammed device, the user application must initiate a system service request. The system controller executes the system service request and programs the device.

The user application cannot obtain the status code in the following scenarios:

- If the auto update program is successful, the device is automatically restarted to initialize the new version of the design.
- If the auto update program fails, the auto update recovery procedure attempts to program the device with the valid image again.

The following table lists the fields in an auto update system service request.

Table 3-13. Auto Update System Service Request

System Service Descriptor Bit Field	Value	Description
15:7	—	Reserved
6:0	46H	Auto update programming command.

When auto update is not enabled in the user design, the auto update system service can be used to update the device with the newest image using the user application.

Note: Auto update system service does not generate an error if SPI controller is not in the master mode.

3.3.3.3. Recovery on Auto Update Programming Failure (Ask a Question)

When power fails during auto update programming, the auto update programming flow is initiated on the next boot cycle to program the device with the newest image.

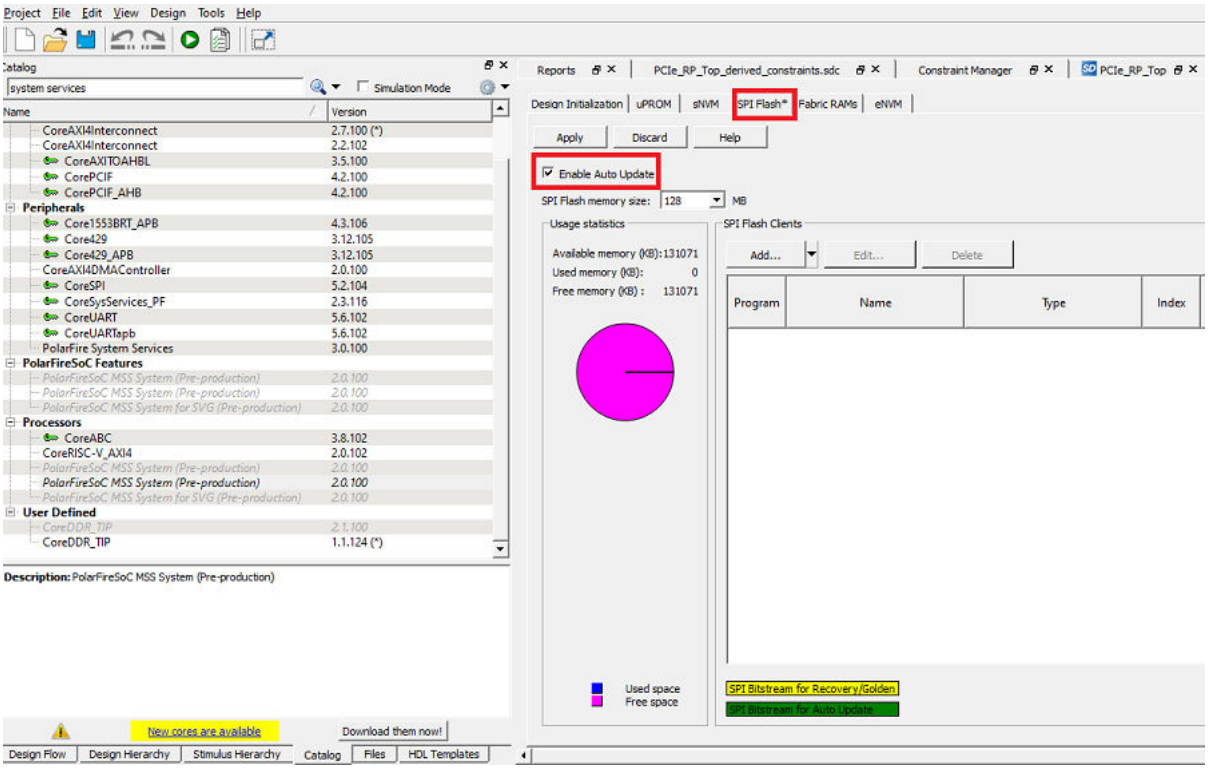
Note: If the device fails to program the newer image, it retries once before programming itself with the older version of the image. If the device remains blank at the end of auto update, there is no indication through I/O and user intervention is required.

3.3.3.4. Enabling Auto Update Option in User Design (Ask a Question)

To enable auto update, follow these steps:

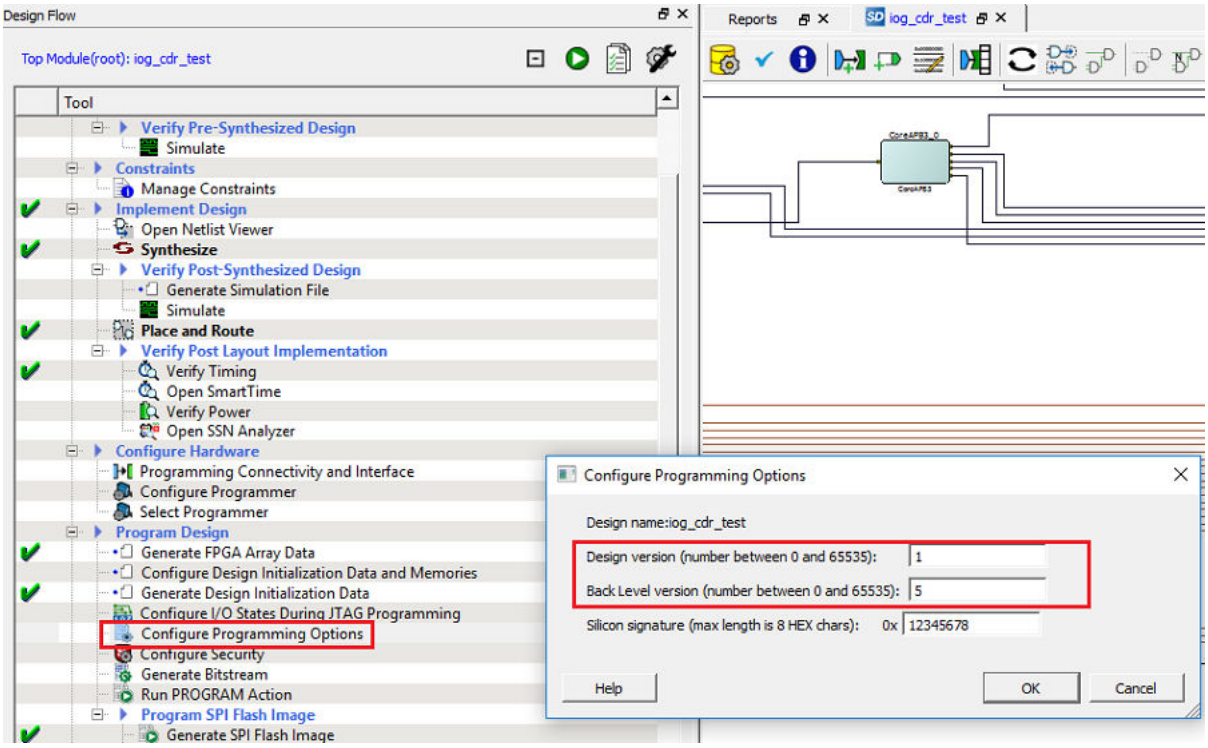
1. Click **Configure Design Initialization Data and Memories** and select the **SPI Flash** tab.
2. Select the **Enable Auto Update** checkbox.

Figure 3-14. Auto Update Setting



3. Click **Configure Programming Options**, and specify the design version and back level version, as shown in the following figure.

Figure 3-15. Design Version



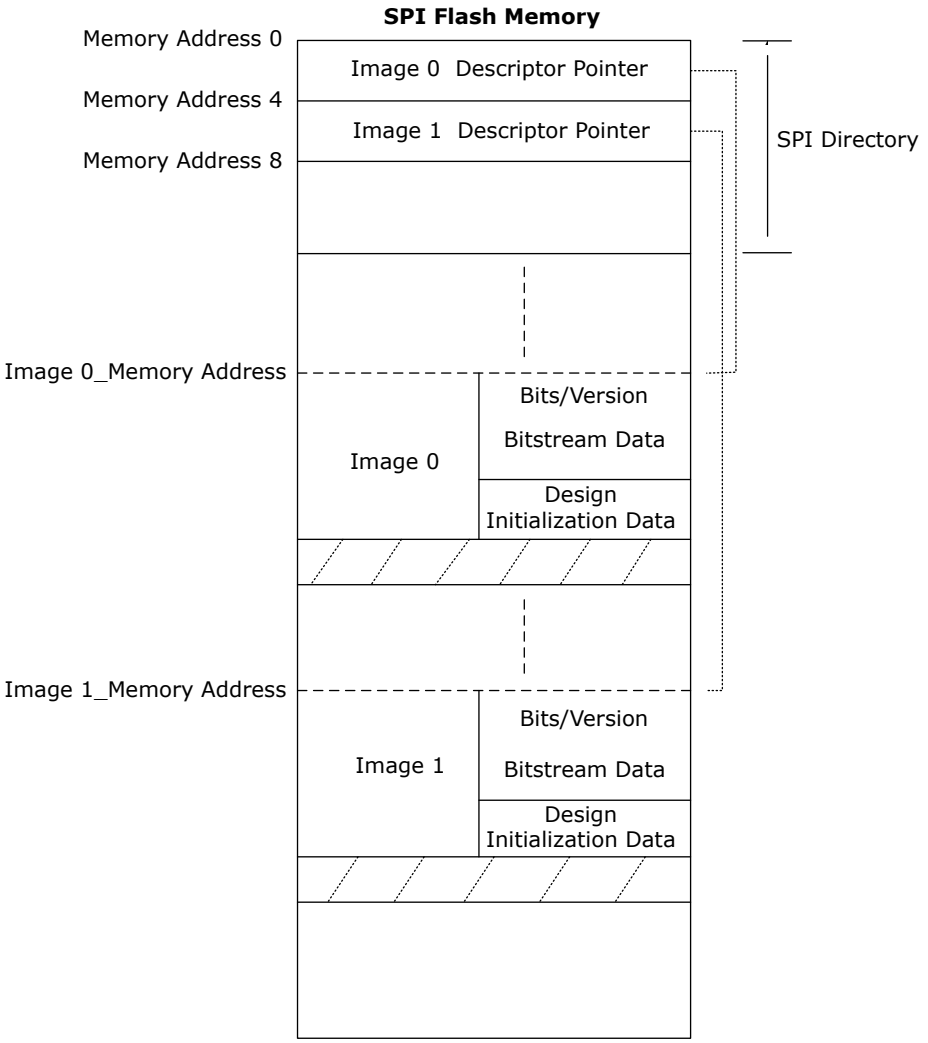
3.3.3.5. Auto Update Use Models (Ask a Question)

Auto update is initiated when a different version of the programming image is available in the SPI flash memory. For more information, see SPI Directory. The device uses the Bits/Version component of the programming image to determine the version. The Bits/Version component appears at the beginning of a bitstream and contains version information. This section describes three auto update use models—ping pong, golden image, and single image. Based on the design requirement, any of these models can be used.

Ping Pong

Auto update uses the newer of the first two images on the SPI flash memory. When a new image is written to the SPI flash memory, the older of the two images is overwritten with the new image. This is known as the ping pong model and is used when the previous image version needs to be retained along with the newer image. This facilitates an automatic rollback to the previous image if the new image fails. The following figure shows the ping pong use model.

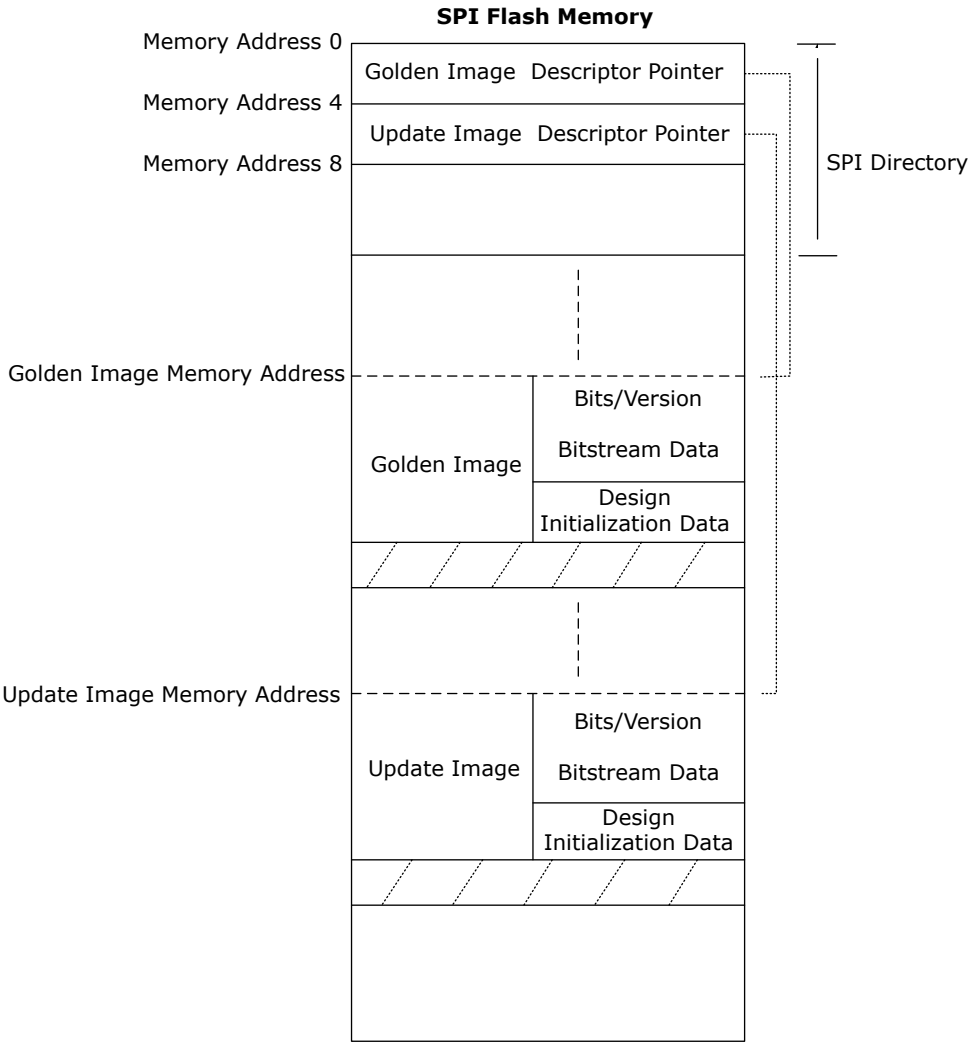
Figure 3-16. Ping Pong Use Model



Golden Image

When auto update fails with a newer version of the image, the device needs to be updated safely using a working image. This image is known as the golden image. When a new image is written to the SPI flash memory, it must not overwrite the golden image. The following figure shows the golden image use model.

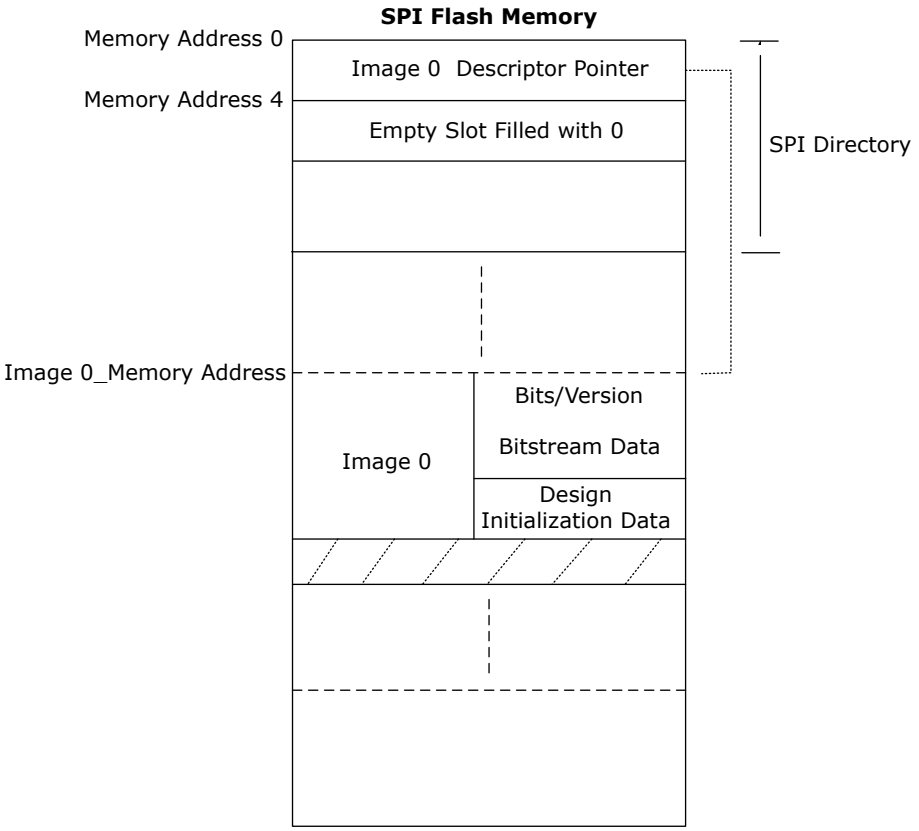
Figure 3-17. Golden Image Use Model



Single Image

This model is used when only one image is available for updating the device. The following figure shows the single image use model.

Figure 3-18. Single Image Use Model



3.3.4. IAP (Ask a Question)

IAP reprograms the device with a specific programming image. In IAP, regardless of the image version, the device chooses the programming image based on either the image index or the SPI image address. The fabric user logic in PolarFire FPGA and MSS user application in PolarFire SoC FPGA specifies the programming image and initiates reprogramming of the device using the IAP system service.

3.3.4.1. IAP Using System Service (Ask a Question)

The user application initiates an IAP system service request using fabric user logic in PolarFire FPGA and MSS user application in PolarFire SoC FPGA. The system service specifies whether the image is used for verification or programming. The system controller automatically reads the bitstream from the SPI flash to verify or program the device contents.

Verify Operation

The verify operation compares the specified programming image contents with the device contents. The following table lists the fields in an IAP system service request using the image index.

Table 3-14. IAP Verify Request by Image Index

System Service Descriptor Bit Field	Value	Description
15	—	Reserved
14:7	SPI_IDX[7:0]	Identifies the image index in the SPI directory for IAP operation.
6:0	44H	IAP verify operation

An SPI flash memory address can be specified instead of the image index within the SPI directory, as shown in the following table.

Table 3-15. IAP Verify Request by Image Address

System Service Descriptor Bit Field	Value	Description
15:7	MBOXADDR[10:2]	Mailbox address. For the format, see Table 3-18.
6:0	45H	IAP verify operation

If the IAP verification is successful, the status code 0 is generated. If the IP verification fails, an 8-bit error code is generated. For the detailed information about error codes, see Appendix: Error Codes.

Digest Check system service is recommended to verify the integrity of the device contents instead of IAP verify operation. For more information, see PolarFire Family System Services User Guide

Program Operation

The program operation updates the device contents using a specified programming image. The IAP program operation does not authenticate the image before executing the program. The image can be authenticated using the IAP image authentication system service. For more information, see IAP Image Authentication System Service.

The user application cannot obtain the status code in the following scenarios:

- If IAP is successful, the device is automatically restarted to initialize the new design.
- If IAP fails, the IAP recovery procedure attempts to program the device with image 0.

Important: IAP recovery considers image 0 when the pointer to image 1 in the SPI directory is null. For more information, see SPI Directory.

The following table lists the fields in an IAP system service request using the image index.

Table 3-16. IAP Program Request by Image Index

System Service Descriptor Bit Field	Value	Description
15	—	Reserved
14:7	SPI_IDX[7:0]	Identifies the image index in the SPI directory for IAP operation.
6:0	42H	IAP program operation

An SPI flash memory address can be specified instead of the image index within the SPI directory, as specified in the following table.

Table 3-17. IAP Request by Image Address

System Service Descriptor Bit Field	Value	Description
15:7	MBOXADDR[10:2]	For the mailbox format, see the following table.
6:0	43H	IAP program operation

The following table describes the mailbox format.

Table 3-18. Mailbox Format

Offset	Length (bytes)	Parameter	Direction	Description
0	4	SPIADDR	Input	Programming image address in SPI flash memory. If the attached SPI flash device does not support 32-bit addresses, SPIADDR[31:24] is ignored.

3.3.4.2. Recovery on Programming Failure (Ask a Question)

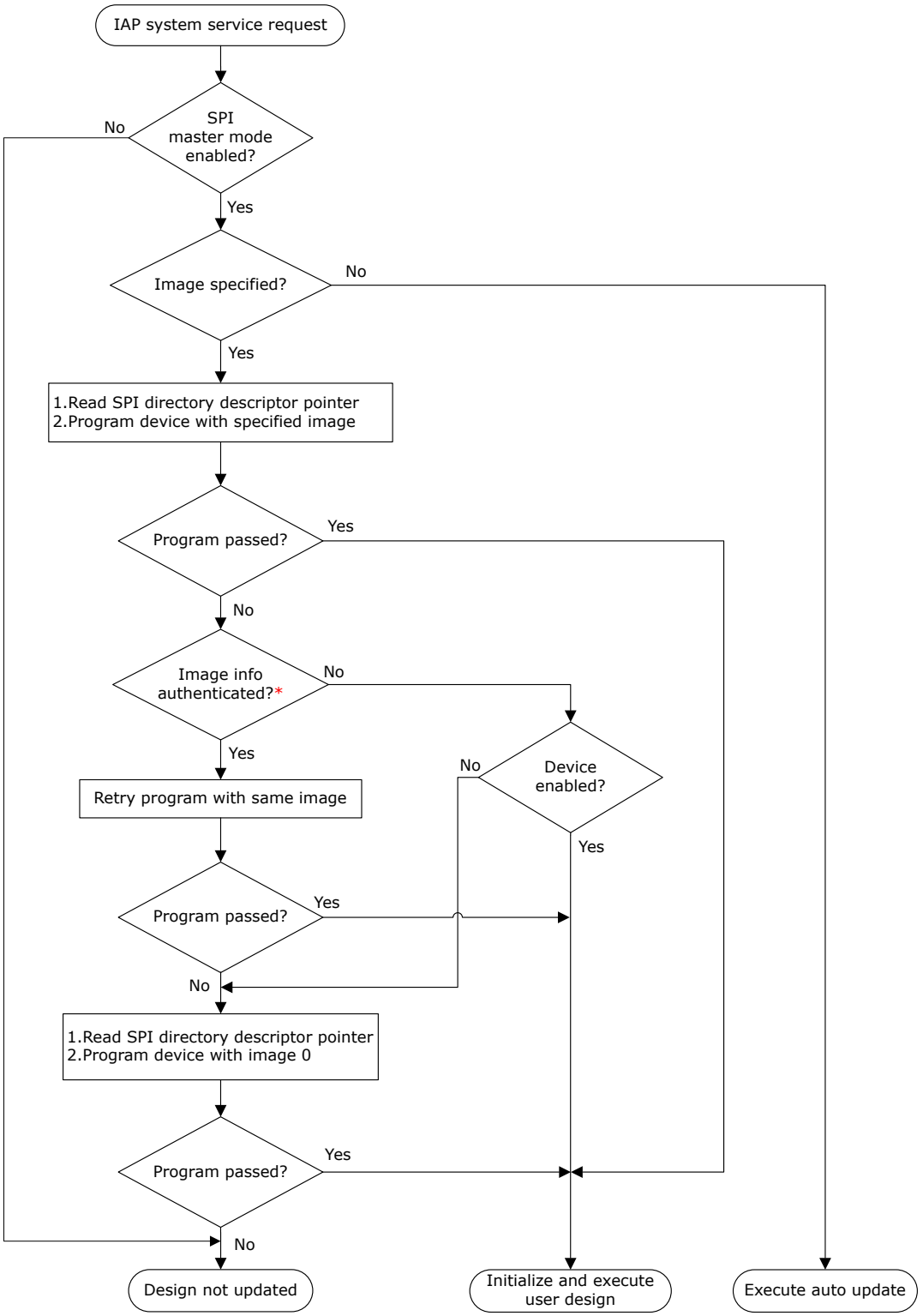
When power fails during IAP, the device programs itself with image 0.

Note: When the device fails to program the specific image, it retries once before programming itself with image 0. If the device is still blank at the end of IAP, there is no indication through I/O and user intervention is required.

3.3.4.3. IAP Flow (Ask a Question)

The following figure shows the IAP flow.

Figure 3-19. IAP Flowchart

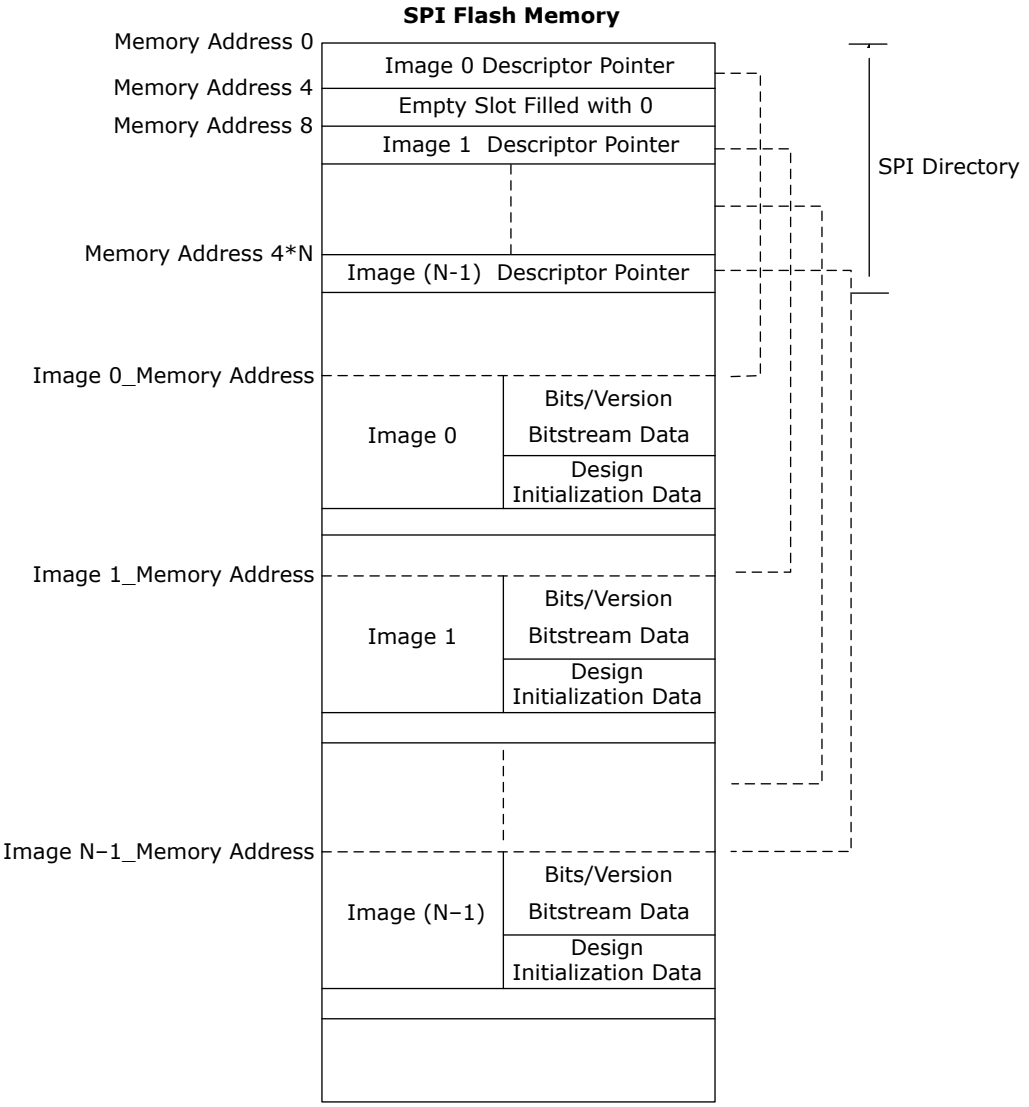


* *Device checks only BITS (starting bits of the bitstream) and AUTH (encryption keys information) components of the bitstream as part of the programming.*

3.3.4.4. IAP Use Model (Ask a Question)

Both the device families support the multi-image IAP use model, which allows up to 255 images to be stored in the SPI flash memory. The image descriptor pointers are in Sector 0 of the SPI flash memory. The device can be programmed with any image; however, if the program fails, the device is programmed with image 0. The programming image pointer next to the image 0 pointer must be null (empty slot). This model is used when the device needs to be updated with a specific image from among the available images. Figure 3-20 shows the multi-image use model.

Figure 3-20. Multi-Image Use Model



4. Bypassing the Back Level Protection (Ask a Question)

If Back Level protection is enabled in the Configure Security tool, the back level protection can be bypassed for SPI bitstreams while exporting the bitstream using Libero. To prevent Programming Recovery failures, enable the **Bypass the Back Level Protection for Recovery/Golden bitstream (SPI files only)**, as shown in the following figures.

Figure 4-1. PolarFire® FPGA—Selecting Bypass Back Level Protection Feature

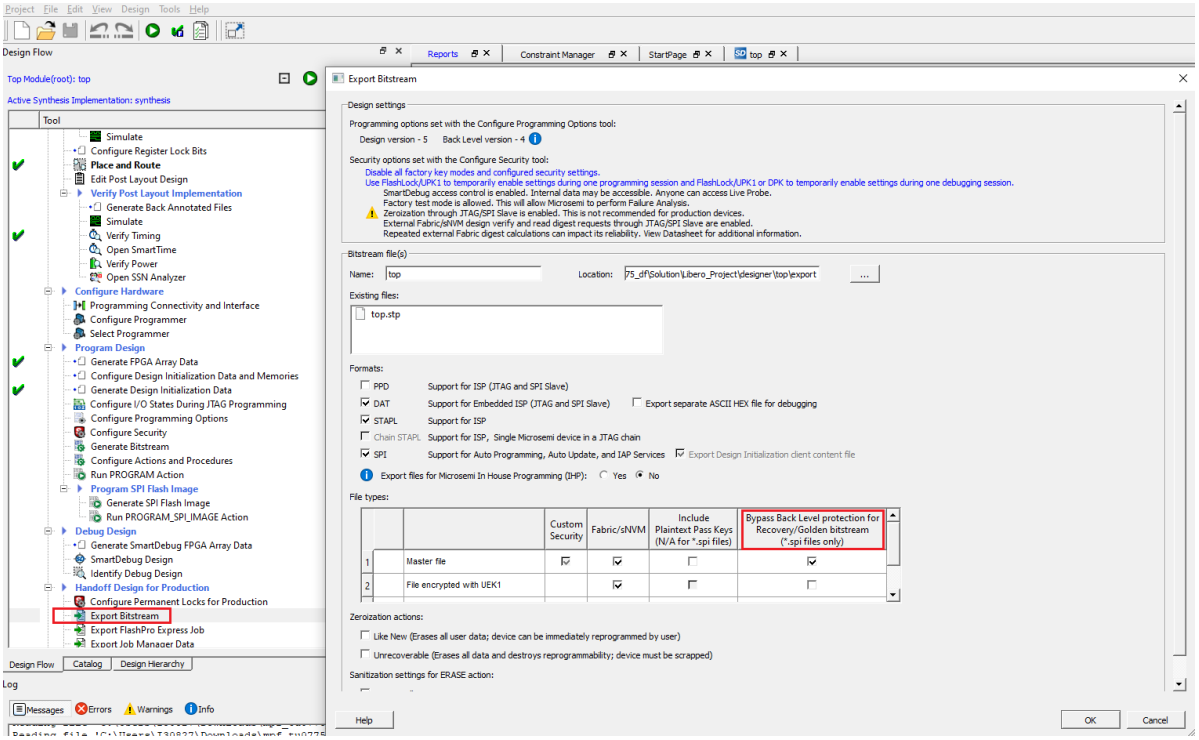
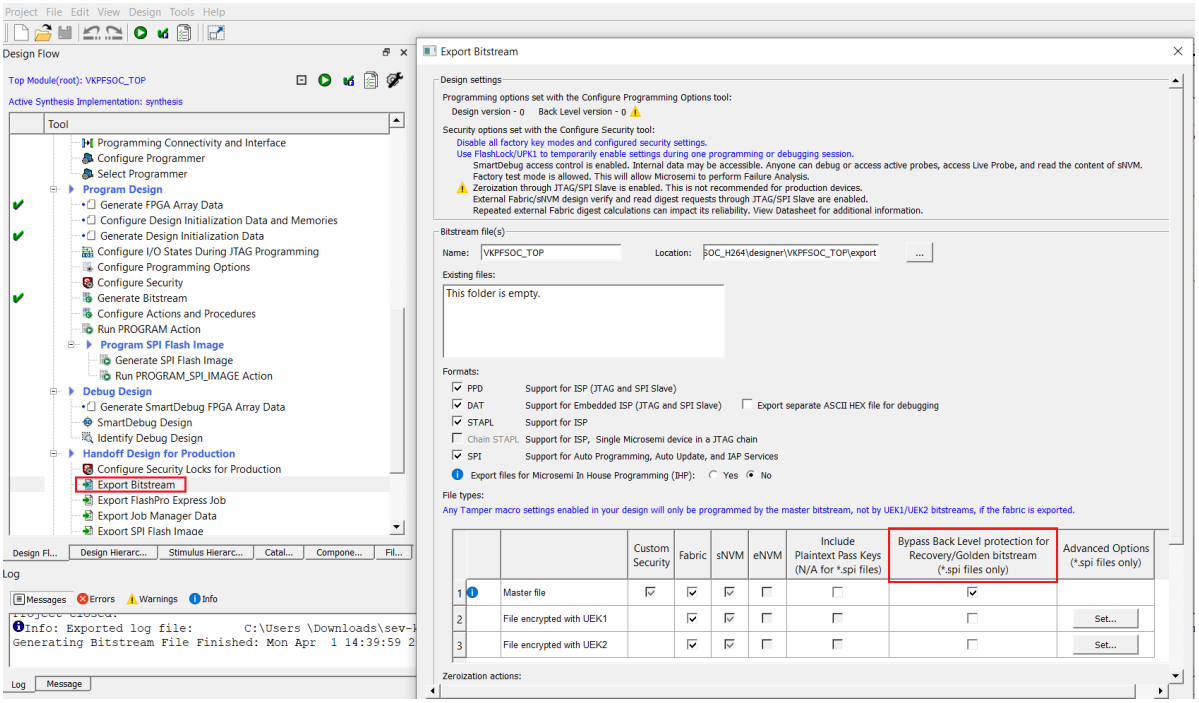
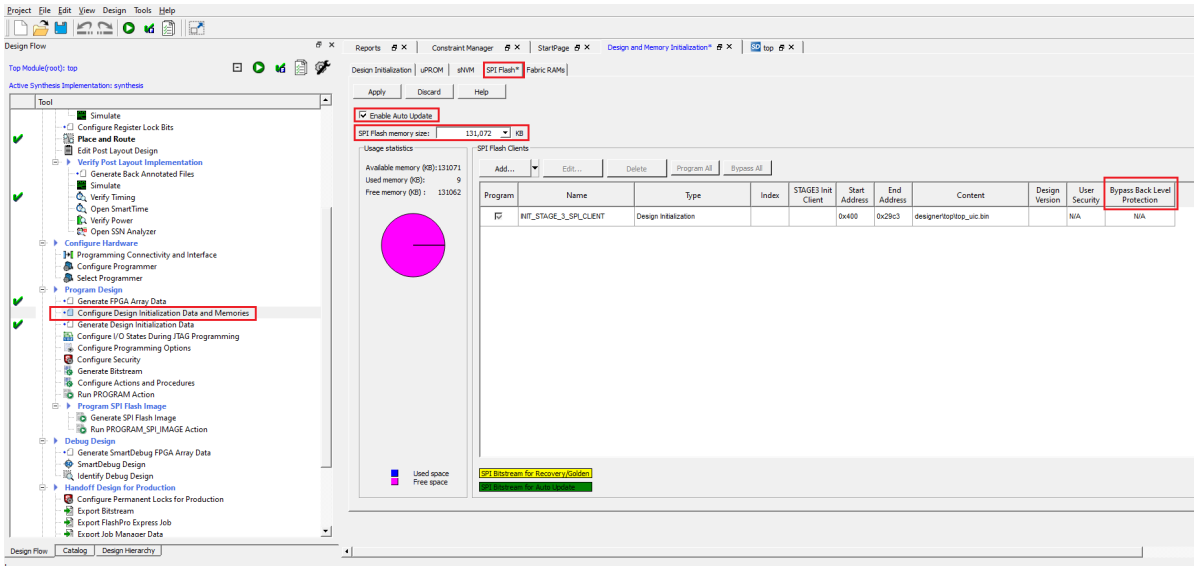


Figure 4-2. PolarFire® SoC FPGA—Selecting Bypass Back Level Protection Feature



When the SPI bitstream is added to the SPI flash using design and memory initialization data, the tool shows back level protection bypass feature in bitstream, as shown in the following figure.

Figure 4-3. Status of Bypass Back Level Protection



4.1. Bypass Back Level Protection Use Case (Ask a Question)

The following table lists the user case for Bypass Back Level Protection.

Table 4-1. Bypass Back Level Protection Use Case

Step	SPI Bitstream	Action	Result	Design Version	Design Back Level Version	Device Back Level Version
1	Golden/Recovery	Auto Programming	Pass	2	1	1
2	IAP/Update Bitstream	Auto Update/IAP	Pass	3	2	2
3	IAP/Update Bitstream	Auto Update/IAP	Fail, Attempt Programming Recovery	4	Not Enabled	2

The steps are described as follows:

1. The device programs with a bitstream version 2 and back level version 1. The current device back level version is set to 1.
2. The device then updates with a bitstream version 3 and back level version 2. The current device back level version is set to 2.
3. The device attempts to update itself with a bitstream version 4 and fails to update. In this case, the device attempts to recover using a golden/recovery bitstream version 2. This recovery also fails, as the current device back level protection is set to version 2 and the golden/recovery bitstream version is equal to the back level version. The **Bypass Back Level Protection** must be enabled (see Figure 4-1) for Golden/Recovery bitstream to avoid programming recovery failures because of back level protection.

5. I/O States During Programming (Ask a Question)

The following table lists the I/O states that apply during various stages of programming.

Table 5-1. I/O States for Various Programming Modes

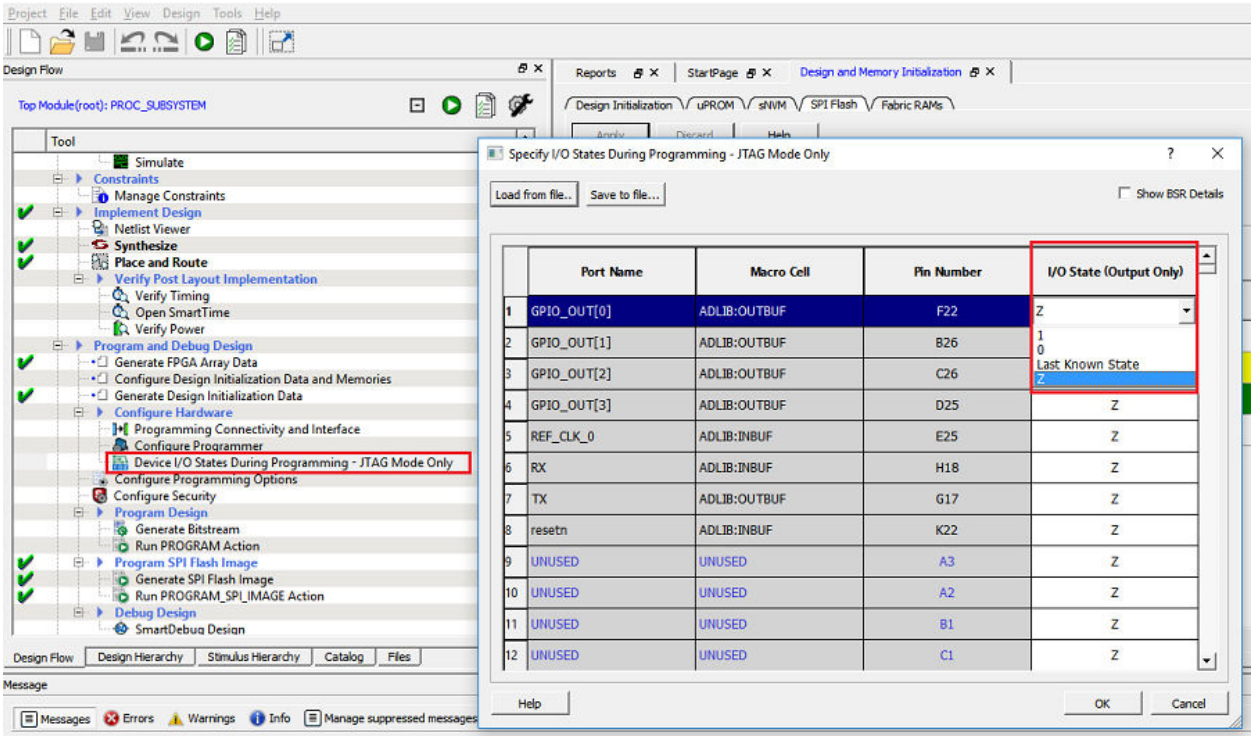
I/O Type	I/O States		
	JTAG Programming	SPI Slave Programming	SPI Master Programming (IAP/Auto Update)
System controller I/O	Enabled	Enabled	Enabled.
XCVR reference clock inputs	Not affected	Not affected	Not affected. May be kept alive during IAP using loopback mode, allowing the XCVR link to be kept active.
XCVR data I/O	As set by the boundary scan cell	Not affected	Not affected. May be kept alive during IAP using loopback mode, allowing the XCVR link to be kept active.
GPIO and HSIO	I/Os are enabled, but the I/O state can be set using the boundary scan cell.	Weakly pulled up by default (no user control)	• During Auto update, I/Os are tristated. • During IAP, I/Os are Weakly pulled down.
MSS I/Os for PolarFire SoC	I/Os are enabled, but the I/O state can be set using the boundary scan cell.	Weakly pulled up by default (no user control)	Outputs are tristated and not in weakly pulled up state.

In Libero SoC, the I/O states can be set before JTAG programming, and these I/O states are held at the set values during JTAG programming. The following are the I/O output state settings:

- 1: I/O is set to drive out logic HIGH
- 0: I/O is set to drive out logic LOW
- Last Known State: I/O is set to the last value that was driven out before entering the programming mode and then held at that value during programming
- Z: I/O is tri-stated with weak pull-up (10 kΩ)

The I/O output states can be set as shown in the following figure.

Figure 5-1. I/O States During Programming (JTAG Mode Only)



6. Fabric and MSS State During Programming (Ask a Question)

The following table lists the Fabric and MSS state during programming.

Table 6-1. Fabric and MSS State During Programming [1]

Action Name	Method	FPGA Fabric	MSS (For PolarFire® SoC FPGA Only)
Programming	JTAG	Off	Off
	IAP/Auto update	Off. However, for PolarFire SoC FPGA, if bitstream does not contain the fabric component, the fabric is operational during IAP program or verify mode.	On (MSS is in operational state during IAP programming using system services)
	Auto-programming	Off	Off (Auto-programming happens on blank parts)
	Recovery	Recovery during auto-programming is Off. However, for PolarFire SoC FPGA, if bitstream does not contain the fabric component, the fabric is operational during IAP program or verify mode.	Off (Recovery during auto-programming) On (Recovery during IAP)
Verify (Standalone)	JTAG	Off	Off
	IAP/Auto update	OFF. However, for PolarFire SoC FPGA, if bitstream does not contain the fabric component, the fabric is operational during IAP program or verify mode.	On
	Auto-programming	N/A	N/A
	Recovery	N/A	N/A
Erase	JTAG	Off	Off (MSS loses its configuration)
	IAP/Auto update	N/A	N/A
	Auto-programming	N/A	N/A
	Recovery	N/A	N/A

Important: Off indicates non-operational and On indicates operational.

7. Programming Recommendations (Ask a Question)

To ensure successful programming, the following guidelines are recommended:

- Authenticate the bitstream before programming the device.
- Do not assert the reset pin (DEVRST_N) during programming because this may corrupt the device configuration.
- Use the correct configuration and programming interface based on the selected programming mode.
- Configure the device I/O states (before JTAG programming) based on the design requirements. For more information, see I/O States During Programming.
- For the external SPI flash Serial Data Out (SDO) and Serial Data In (SDI) traces, it is recommended to implement a series impedance of approximately 20Ω when utilizing the MPF050T device to ensure proper functionality of the auto programming and auto update features.

8. Brownout During Programming (Ask a Question)

Brownout is a condition that occurs when the power supplies fall below the recommended levels. If brownout occurs during programming, the device automatically recovers from the programming failure (as auto recovery is enabled by default) and programs the device with a valid programming image stored in the external SPI flash. IAP of sNVM component only or eNVM component only bitstreams is not recommended. A power failure during IAP programming of these types of bitstreams can lead to an incorrectly programmed device. To avoid this, IAP program with bitstreams that also contain the FPGA fabric component, which ensures automatic recovery, executes, if a brownout occurs. IAP/Auto-update programming of security settings is not recommended to prevent the risk of accidental device lockout due to a brownout event.

9. Zeroization (Ask a Question)

Both the device families have a built-in capability that can zeroize (clear and verify) any or all configuration storage elements as per the user setting. Internal volatile memories, such as LSRAMs, µSRAMs, and system controller RAMs are cleared and verified. Once the zeroization is complete, a zeroization certificate can be retrieved using a JTAG/SPI slave instruction to confirm that the zeroization process is successful. For more information about zeroization, see PolarFire FPGA and PolarFire SoC FPGA Security User Guide.

10. Programming the External SPI Flash (Ask a Question)

To perform IAP or auto update, an external SPI flash memory is required. This SPI flash memory interfaces with the system controller's SPI and stores the programming images.

The SPI flash memory is divided into several sectors. The 1 KB memory in first sector (sector 0) is used as the SPI directory, and it contains the programming image indexes (descriptor pointers). The remaining flash memory stores the programming images.

10.1. Supported SPI Flash Devices (Ask a Question)

SPI flash devices from various vendors implement a standard instruction set for read operations. The system controller firmware executes the following command to identify the addressing mode (3-byte or 4-byte):

```
READ SERIAL FLASH DISCOVERY PARAMETER (5AH)
```

The System Controller supports read and write operations using FlashPro6 programmer and Libero software. However, if any SPI Flash devices that are JESD216 compliant, support 3-byte or 4-byte addressing and 0BH fast read, support only read operation by system controller and subjected to specific memory size. For more information, see FlashPro6 SPI Flash Support.

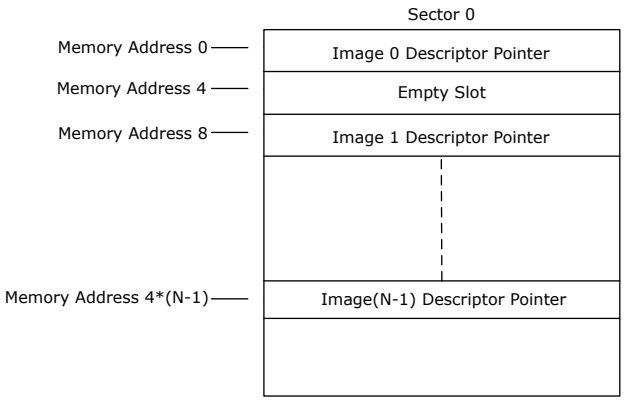
Important: FlashPro5 programmer supports read and write operation for only Micron device part number: MT25QL01G, which is limited to first 128 MB of the Flash memory.

10.2. SPI Directory (Ask a Question)

The SPI directory is a collection of image descriptor pointers that point to the beginning of the programming image. Each pointer uses four bytes. If the SPI flash memory device supports only the 3-byte addressing mode, the first three bytes are used.

For IAP recovery to choose image 0 on power-up, the programming image pointer next to the image 0 pointer must be null (empty slot), otherwise auto update is chosen. The following figure shows the SPI flash directory with the programming image descriptor pointers.

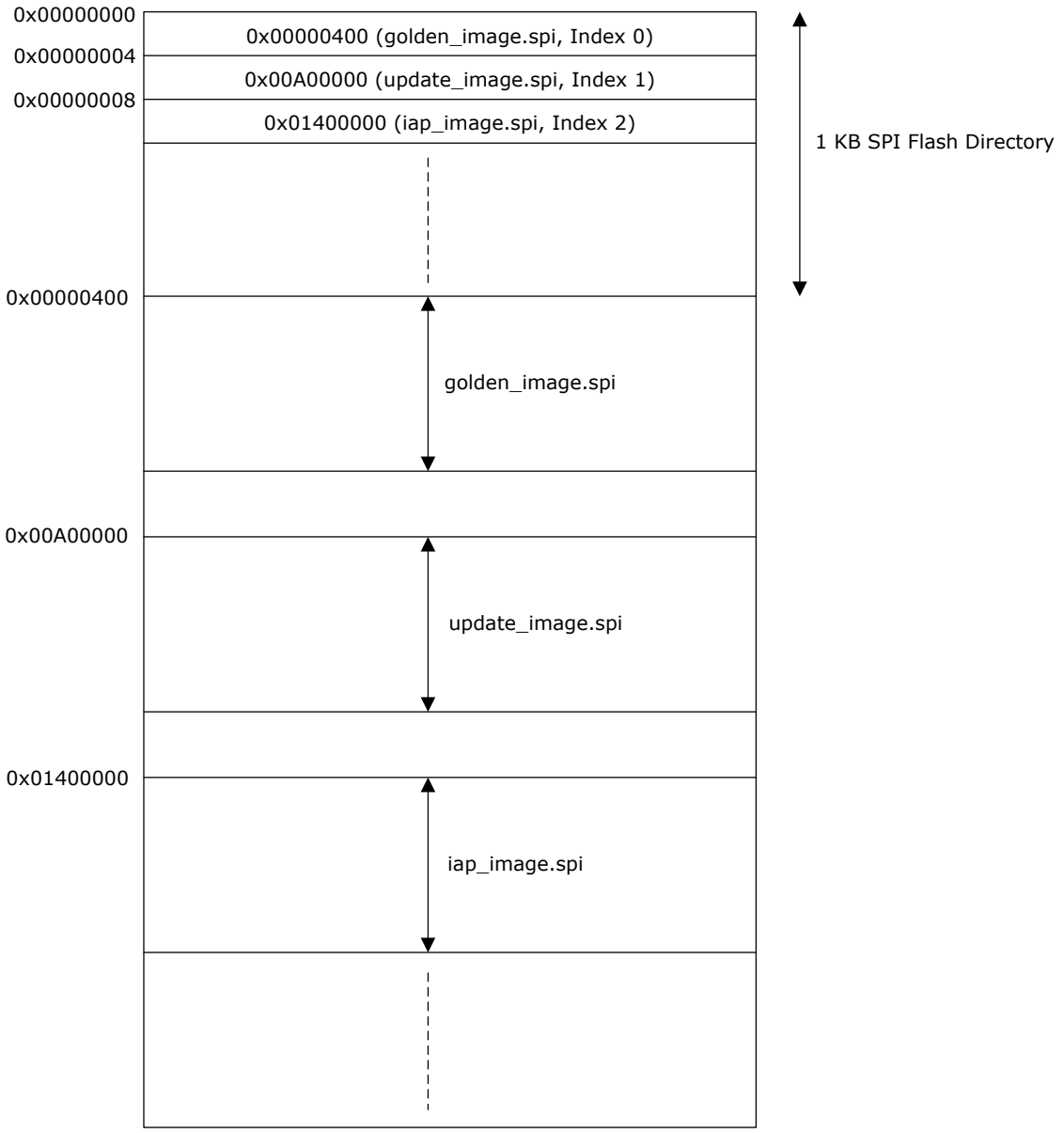
Figure 10-1. SPI Flash Directory



The SPI directory contains the start addresses of the programming images. The SPI directory occupies 1 KB memory from sector 0 of external SPI flash memory. For example, if the external SPI flash contains three images: golden image, update image, and IAP image, then these images are stored at memory with starting addresses: 0x400, 0xA00000, and 0x1400000. If the Libero configurator is used to program SPI flash with programming images, then the Libero configurator

takes care of the programming SPI directory automatically. If the user application programs the external SPI flash with programming images, then the application must write starting addresses of each image into SPI directory starting from SPI flash address 0, as shown in the following figure.

Figure 10-2. SPI Flash Memory



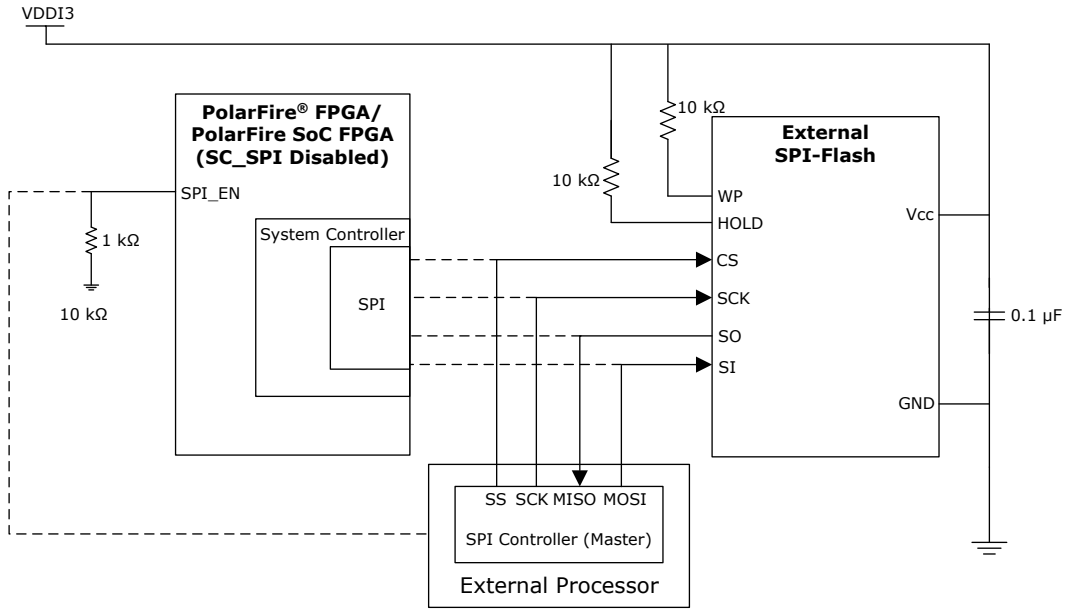
10.3. Use Models for Programming SPI Flash (Ask a Question)

The external SPI flash can be programmed using either JTAG or the system controller's SPI. When the system controller's SPI is enabled and configured in SPI master mode, the system controller's SPI port is shared between the system controller and either the FPGA fabric master/MSS (for PolarFire SoC FPGA only) or JTAG. This section describes the use models for programming the external SPI flash.

10.3.1. Programming the SPI Flash Using External Processor (Ask a Question)

When the SPI_EN pin is disabled (driven LOW), the system controller's SPI outputs are tri-stated, and the external processor can drive the SPI pins to program the SPI flash. Neither the system controller nor the fabric/MSS (for PolarFire SoC FPGA only) can drive the SPI interface. The external processor can drive the SPI_EN pin LOW to program the external SPI flash. The SPI_EN pin can also be configured external to the device using the jumpers on the board. The SPI flash is programmed using an external processor SPI master SCK frequency. The SCK frequency is configured using external processor application. The following figure shows the connections required for programming the SPI flash using an external processor.

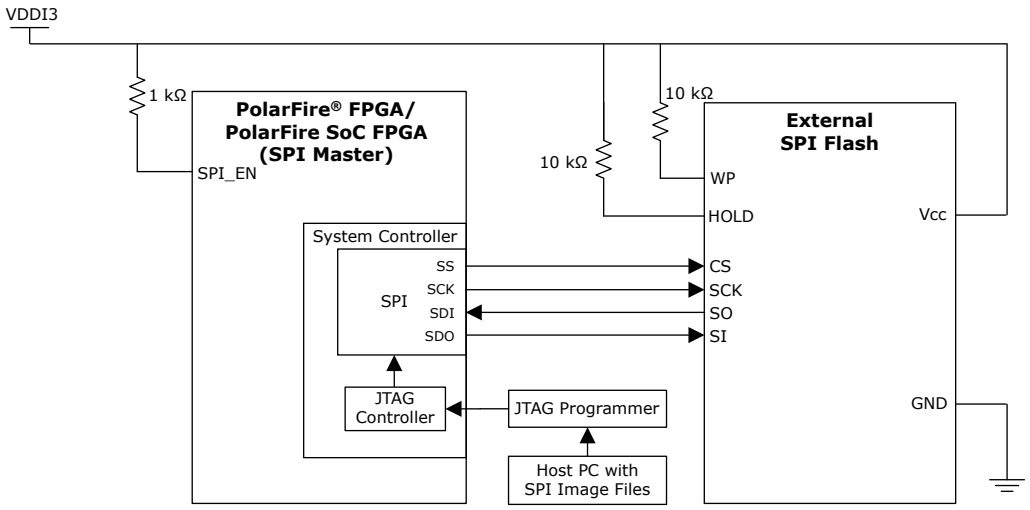
Figure 10-3. SPI Flash Programming Using External Processor



10.3.2. Programming the SPI Flash Using JTAG (Ask a Question)

The external SPI flash can be programmed using a FlashPro programmer (version 5 or later) through the system controller's JTAG interface. The JTAG controller uses a special JTAG instruction—`SPIPROG (IR=0xb0)`—to interface with the external SPI flash through the system controller's SPI. The JTAG controller in both the device families supports this instruction to directly drive the system controller's SPI outputs. The following figure shows the connections required for programming the SPI flash using JTAG.

Figure 10-4. SPI Flash Programming Using JTAG

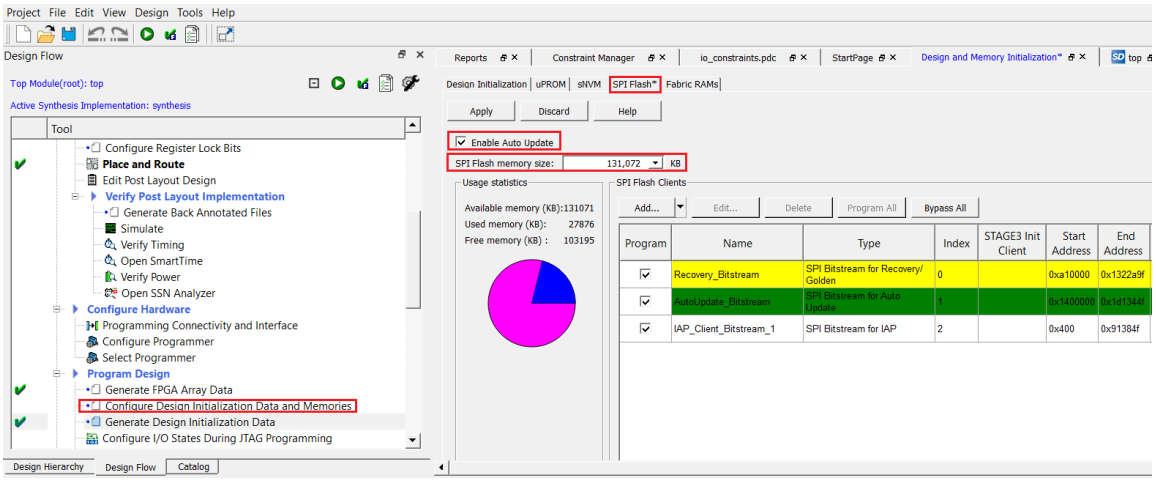


10.3.2.1. Programming External SPI Flash Using Libero (Ask a Question)

The Libero SoC software allows you to program the external SPI flash memory with programming images. To program the SPI flash memory:

1. Go to **Design Flow** > **Program and Debug Design** > **Configure Design Initialization Data and Memories**, and select the **SPI Flash** tab, as shown in following figure.

Figure 10-5. SPI Flash Programming in Libero® SoC

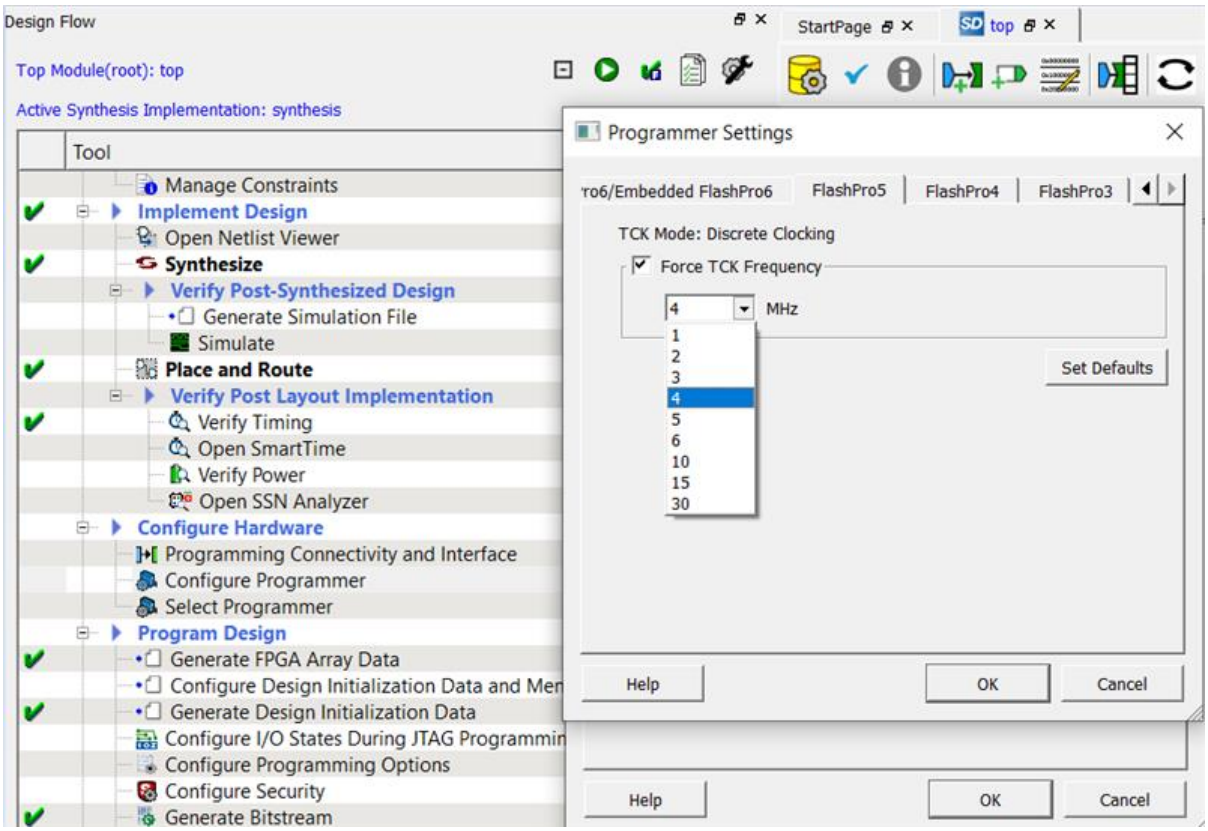


Important: For PolarFire FPGA, to streamline the SPI-Flash Programming support with FlashPro6, effective from Libero SoC v12.4, the vendor information is replaced with the density of the target memory.

2. Under **SPI Flash Clients**, add the required programming images, and click **Apply**. For more information about values to be entered in the fields, click **Help**.

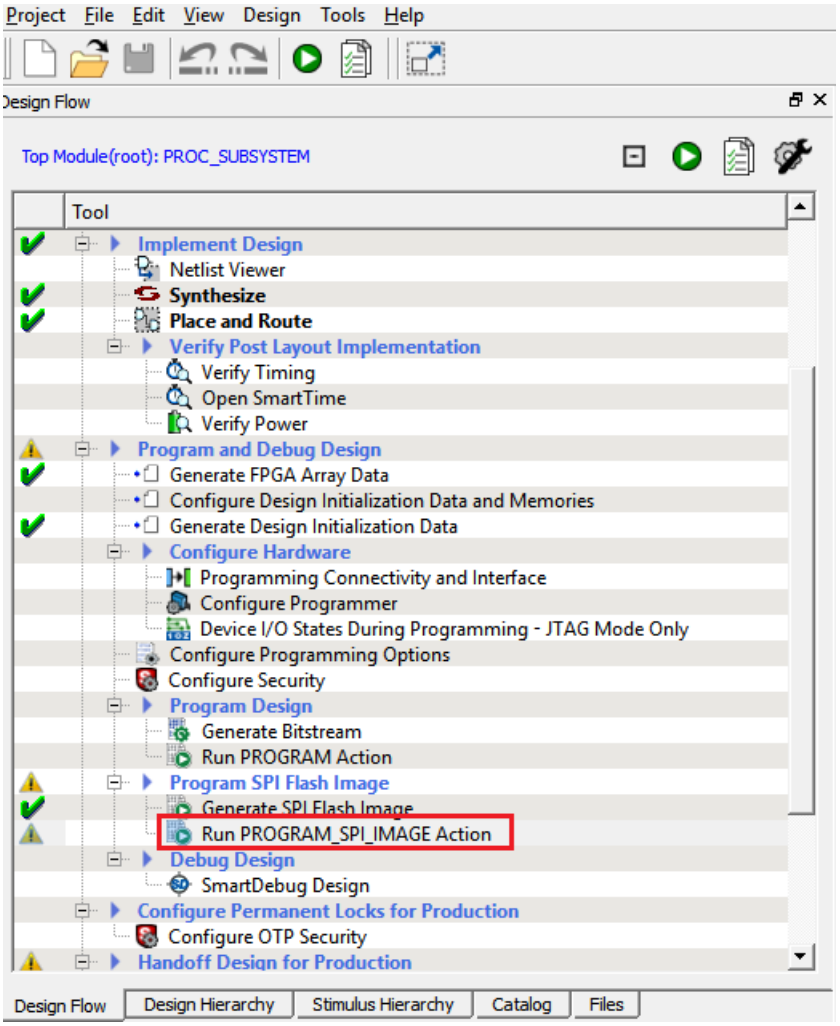
3. Go to **Design Flow** > **Configure Hardware** > **Configure Programmer** > right-click and select **Programmer Settings** in the FlashPro tabs. You can modify the TCK frequency by checking and selecting the Force TCK Frequency to enhance the SPI flash programming time.

Figure 10-6. Programmer Settings



4. Double-click **Run PROGRAM_SPI_IMAGE Action** to get the SPI flash programmed with the SPI directory and the programming images.

Figure 10-7. Run PROGRAM_SPI_IMAGE Action



For more information about design initialization data and memories, see PolarFire FPGA and PolarFire SoC FPGA Device Power-up and Reset User Guide.

Notes: The following are the recommendations for SPI Flash Programming Using Libero.

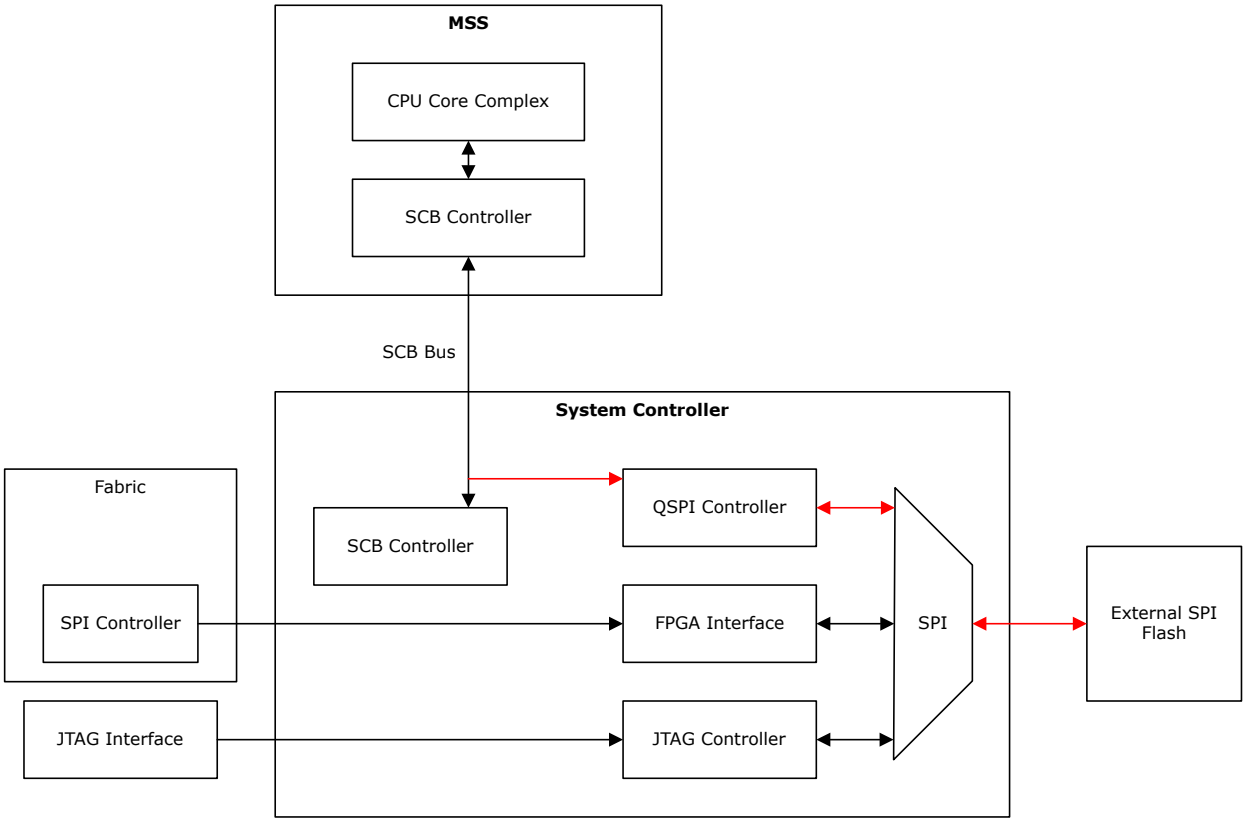
- This tool erases the SPI Flash prior to programming. It is recommended to program the SPI Flash with Libero SoC prior to programming other data on the SPI Flash using non-Libero programming solutions.
- Partial update of the SPI Flash is currently not supported.
- It is not recommended to have large gaps between clients in the SPI Flash, as gaps are currently programmed with 1s and increases programming time.

10.3.3. Programming the SPI Flash Using MSS (For PolarFire SoC FPGA Only) (Ask a Question)

MSS connects to the system controller QSPI using SCB bus. The QSPI controller allows MSS to access system controller SPI flash memory. Use the MSS QSPI driver in Normal mode to access an external SPI flash memory using QSPI base address as 0x37020100 (this address is part of IOSCB data).

The following figure shows the interface between MSS and system controller QSPI controller.

Figure 10-8. Interface between MSS and System Controller QSPI Controller



10.3.4. Copying Contents from External SPI Flash to MSS User Application (for PolarFire SoC FPGA Only) (Ask a Question)

The MSS SPI copy system service allows data to be copied from the external SPI flash to the MSS user application memory. The `mss_system_services` driver includes the method to copy data from external SPI flash to the MSS user application memory.

For information about `mss_system_services` driver and example SoftConsole project, see GitHub. This MSS SPI copy system service is only useful for reading contents from the External SPI flash memory.

10.3.5. Programming the SPI Flash Using Fabric User Logic (Ask a Question)

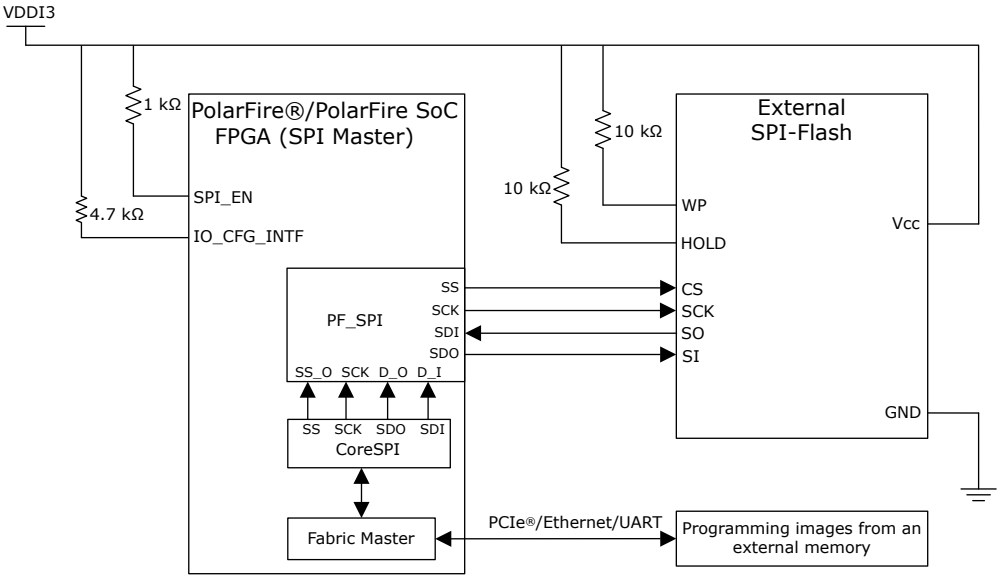
When the system controller's SPI is enabled and configured as master, the system controller hands over the control of the SPI to the fabric (after device power-up). The JTAG controller that starts programming the SPI flash, or any system service request from the fabric user logic, can take over the control of SPI from the fabric.

The fabric user logic gets the programming images from an external memory source, as shown in the following figure. The fabric user logic accesses the external SPI flash using the CoreSPIcontroller and PF_SPI macro provided in Libero Catalog. The external SPI flash is programmed using SPI master SCK frequency. The SCK frequency can be configured in user logic.

System controller can only access dedicated SPI I/Os (SPI Interface pins). System Controller cannot access the fabric IOs. As a result, all the services from the system controller using SPI (that is, programming) can only use the dedicated SPI I/Os. The user can use PF_SPI, a macro provided in the Libero Catalog to get access to the dedicated SPI I/Os from the fabric (that is, once the system controller releases them) to access the SPI flash memory.

Note: To fetch the programming images and write to the external SPI flash, both the device families must be preprogrammed with a design. For more information, see PolarFire FPGA Auto Update and In-Application Programming Application Note.

Figure 10-9. SPI Flash Programming Using Fabric User Logic



11. System Controller Suspend Mode (Ask a Question)

PolarFire FPGA and PolarFire SoC FPGA devices have a System Controller Suspend mode feature that can be used to force the system controller into reset after the device initialization is complete. This mode is essential for safety critical applications to protect the device from unintended device programming or zeroization of the device due to Single Event Upset (SEU) events. When the user program the System Controller to be in the Suspend mode (using the Libero SoC tool), some device programming options are disabled, and others are enabled/disabled by controlling the JTAG_TRST_B pin. The following table lists the availability of device programming options when the device is programmed with System Controller Suspend mode enabled. For a full listing of device features availability in suspend mode, and suspend mode operation, see PolarFire FPGA and PolarFire SoC FPGA System Services User Guide.

Table 11-1. PolarFire® FPGA and PolarFire SoC FPGA Programming Availability in System Controller Suspend Mode

Programming Mode	System Controller Suspend Mode		Description
	Enabled	Disabled	
JTAG	Yes[1]	Yes	—
SPI Slave	Yes[1]	Yes	—
Auto-update (POR/DEVRSTn)	Yes	Yes	Executes after power-up or DEVRSTn, if this feature is enabled in the device.
Auto-update (System Service)	No	Yes	Auto-update requested through System Services is not available in suspend mode.
IAP	No	Yes	—

Note:

1. Requires the JTAG_TRST_B pin to be driven logic high prior to executing operation and remain high until operation is complete.

A device is configured to be in System Controller Suspend mode when the System Controller Suspend mode bit is programmed into a device during FPGA programming. Controlling the JTAG_TRST_B pin affects the availability of JTAG and SPI Slave programming mode and has no effect on the other features listed in Table 11-1.

Note: For PolarFire FPGA devices, when the system controller is forced from suspend mode reset by asserting JTAG_TRST_B = 1, the outputs of the PF_INIT_MONITOR macro are forced = 0. This macro is often used for resetting the user logic design, so the appropriate user design considerations must be made for this operational case.

12. Appendix: Error Codes (Ask a Question)

The system controller executes system service requests from the design. When a service is completed, a status code is returned to the user application. This status code can be 0 (success) or an 8-bit error code. The following table lists the error codes.

Table 12-1. Error Codes

Error Code	Error Text	Error Description
1	Validator or hash chaining mismatch	Bitstream is constructed incorrectly, or a wrong security key is used.
2	Unexpected data received	Additional data is received after the End of the Bitstream (EOB) component.
3	Invalid/corrupt encryption key	Requested key mode is disabled, or the key could not be read or reconstructed.
4	Invalid component header	Bitstream contains invalid component data.
5	Back level not satisfied	Bitstream version is older than that of the current back level in the device.
6	Illegal bitstream mode	Requested bitstream mode is disabled by user security.
7	DSN binding mismatch	Bitstream is rejected because the Device Serial Number (DSN) in the bitstream does not match the DSN on the device.
8	Illegal component sequence	Bitstream ends in the ERR state, meaning it is an illegal bitstream. Every bitstream begins in the BEGIN state, but only a legal bitstream ends in the END state.
9	Insufficient device capabilities	Bitstream is rejected because the capabilities specified in the bitstream do not match the target device's capabilities.
10	Incorrect DEVICEID	Bitstream is rejected because an attempt by the DEVICEID specified in the bitstream does not match the part identification field of the target device.
11	Unsupported bitstream protocol version (regeneration required)	Bitstream is rejected because of an attempt made by the old device to decode the new version of bitstream or by the new device to decode the old version of the bitstream.
12	Verify not permitted on this bitstream	When the device programs the bitstream with encryption keys, it is not possible to use the bitstream later to verify the device contents because the device refers to the modified encryption keys.
13	Invalid device certificate	Device certificate is missing or invalid.
14	Invalid DIB	Device integrity bits are invalid.
21	Device not in SPI master mode	Bitstream is executed in IAP mode, but the device is not configured as SPI master.
22	No valid images found (auto update)	Bitstream is executed through auto update mode, but no valid image pointers are found.
23	No valid images found (IAP)	Bitstream is executed through IAP via index mode, but no valid image pointers are found.
24	Programmed design version newer than auto update image	Bitstream is executed through auto update mode, and the design version is the latest.
25	Reserved	
26	Selected image invalid and no recovery performed because the device is running a valid design	Bitstream is executed through auto update or IAP mode, and the selected image is invalid.
27	Selected recovery image failed to program	Bitstream is executed through auto update or IAP mode, and the selected recovery image failed to program the device.
127	Abort	A non-bitstream instruction is executed during bitstream loading.
128	NVMVERIFY	Fabric/security key segment verification failed.
129	PROTECTED	The device non-volatile memory cannot be modified because of device security settings.
130	NOTENA	Programming mode is not enabled.

Table 12-1. Error Codes (continued)

Error Code	Error Text	Error Description
131	SNVMVERIFY	The sNVM verify operation failed.
132	SYSTEM	An error occurred in the system hardware (PUF or DRBG).
133	BADCOMPONENT	An error is detected in a component's payload.
134	HVPROGERR	The HV programming subsystem has failed.
135	HVSTATE	The HV programming subsystem is in an unexpected state because of an error.

13. Revision History (Ask a Question)

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

Revision	Date	Description
H	05/2025	The following is a summary of the changes made in this revision. • Updated Table 1-1 in the Bitstream Generation section. • Updated the Programming File Size section for the programming file sizes. Also, the following tables are updated: – Table 1-2 – Table 1-3 – Table 1-5 – Table 1-6 • Updated Table 5-1 in I/O States During Programming. Also added that the I/O is tristated with weak pull-up (10 kΩ). • Updated Table 6-1 in the Fabric and MSS State During Programming section for the auto update feature. • Updated the Programming Recommendations section.
G	10/2024	The following is a summary of the changes made in this revision. • Updated the following figures with better quality images: – Figure 1-3 – Figure 1-4 – Figure 1-5 – Figure 4-1 – Figure 4-2 – Figure 4-3 – Figure 10-5 • Updated Programming File Size by adding the information about programming bitstream export. • Revised the note about GPIO glitch during JTAG programming by mentioning the power-down sequence. • Updated and refined the sentence for better clarity in IAP Using System Service • Updated Table 5-1 by changing the state of GPIO/HSIO during Auto-update and IAP.
F	02/2024	Removed Supported SPI Flash Devices table and added a reference to FlashPro6 SPI Flash support. See Supported SPI Flash Devices.
E	07/2023	The following is a summary of the changes made in this revision. • Information about Supported SPI Flash Devices was updated. • Information about SPI_EN input was added. See SPI Master Programming Interface.
D	04/2023	The following is a summary of the changes made in this revision. • Updated information about sNVM capacity. See Adding sNVM Data to the Bitstream. • Updated information about SPI Master Programming. • Added information about Fabric and MSS State During Programming. • Updated information about Brownout During Programming. • Added information about Programming the SPI Flash Using MSS (For PolarFire SoC FPGA Only).

Revision History (continued)		
Revision	**Date**	**Description**
C	01/2023	The following is a summary of the changes made in this revision. • Updated the GitHub links in the document. • Added a note to mention about control of SPI to the fabric during Suspend mode in SPI Slave Programming Interface.
B	04/2022	The following is a summary of the changes made in this revision. • Added information about System Controller Suspend mode. See System Controller Suspend Mode. • Added Information about programmer and SPI flash part number. See Programming the SPI Flash Using JTAG.
A	08/2021	The first publication of the document. This user guide was created by merging the following documents: • UG0714: PolarFire FPGA Programming User Guide • UG0914: PolarFire SoC FPGA Programming User Guide The revision history tables of both the user guides are retained here for the future reference. For information, see Table 13-1 and Table 13-2.

The following revision history table describes the changes that were implemented in the UG0714: PolarFire FPGA Programming User Guide document. The changes are listed by revision.

UG0714: PolarFire FPGA Programming User Guide document is now obsolete and the information in the document has been migrated to PolarFire® FPGA and PolarFire SoC FPGA Programming User Guide.

Table 13-1. Revision History of UG0714: PolarFire FPGA Programming User Guide

Revision	Date	Description
Revision 9.0	6/21	The following is a summary of changes made in this revision. • Updated the Auto Update on a Blank Device (Auto Programming). • Updated the SPI Directory. • Updated the Programming the SPI Flash Using External Processor. • Updated the Programming External SPI Flash Using Libero®. • Updated the Programming the SPI Flash Using Fabric User Logic.
Revision 8.0	3/21	Removed information about the Dynamic Reconfiguration Interface (PF_DRI) section.
Revision 7.0	8/20	The following is a summary of the changes made in this revision. • Updated Recommended Board Configuration for SPI Master Programming figure. • Updated the Configuring Bitstream Components. • Updated the Programming the SPI Flash Using Fabric User Logic. • Updated the Programming External SPI Flash Using Libero. • Updated Flash Programming in Libero SoC configurator screen shot. • Updated Connecting FlashPro Programmer to a PolarFire FPGA figure. • Updated the Programming File Size. • Updated the SPI Slave Programming Interface. • Updated the Device Programming Flow. • Updated the JTAG Programming Interface.
Revision 6.0	5/20	Updated the document for Libero SoC v12.1.

Table 13-1. Revision History of UG0714: PolarFire FPGA Programming User Guide (continued)

Revision	Date	Description
Revision 5.0	10/18	The following is a summary of the changes made in this revision. • Updated the document for Libero SoC PolarFire v2.3. • The flash freeze functionality is de-featured. • Added information about Bypass the back level protection feature. For more information, see Bypassing the Back Level Protection. • Updated the information about zeroization. For more information, see Zeroization. • Update the Programming file sizes table.
Revision 4.0	4/18	The following is a summary of the changes made in this revision. • Updated the programming file size details for MPF300 devices. • Updated the port details of SPI block in SPI Flash Programming Using JTAG figure and SPI Flash Programming Using Fabric User Logic figure.
Revision 3.0	11/17	The following is a summary of the changes made in this revision. • Updated the document for Libero SoC PolarFire v2.0. • Updated the programming file size details for MPF300 devices. For more information, see Programming File Size. • Updated the information about I/O states for XCVR reference clock inputs, XCVR data I/O, GPIO, and HSIO. • Added the procedure to program an external SPI flash. For more information, see Programming External SPI Flash Using Libero. • Updated JTAG pin details. • Updated the diagram showing the recommended configuration for SPI master programming. • Replaced VDDIx with VDDI3 throughout the document.
Revision 2.0	6/17	The following is a summary of the changes made in this revision. • Updated the information about bitstream generation. • Updated the information about auto update.
Revision 1.0	2/17	The first publication of UG0714: PolarFire FPGA Programming User Guide.

The following revision history table describes the changes that were implemented in the UG0914: PolarFire SoC FPGA Programming User Guide document. The changes are listed by revision.

UG0914: PolarFire SoC FPGA Programming User Guide document is now obsolete and the information in the document has been migrated to PolarFire® FPGA and PolarFire SoC FPGA Programming User Guide.

Table 13-2. Revision History of UG0914: PolarFire SoC FPGA Programming User Guide

Revision	Date	Description
Revision 4.0	6/21	The following is a summary of changes made in this revision. • Updated the Auto Update on a Blank Device (Auto Programming). • Updated the SPI Directory. • Updated the Programming the SPI Flash Using External Processor. • Updated the Programming External SPI Flash Using Libero. • Updated the Programming the SPI Flash Using Fabric User Logic.
Revision 3.0	3/21	Updated the Programming the SPI Flash Using MSS.

Table 13-2. Revision History of UG0914: PolarFire SoC FPGA Programming User Guide (continued)

Revision	Date	Description
Revision 2.0	10/20	The following is a summary of the changes made in this revision. • Updated Recommended Board Configuration for SPI Master Programming figure. • Updated the Configuring Bitstream Components. • Updated the Programming the SPI Flash Using MSS. • Updated Connecting FlashPro Programmer to a PolarFire SoC FPGA figure. • Updated the SPI Slave Programming Interface. • Updated the Device Programming Flow. • Updated the JTAG Programming Interface.
Revision 1.0	4/20	The first publication of UG0914: PolarFire SoC FPGA Programming User Guide.

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: 979-8-3371-1303-6

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.