RT PolarFire® FPGA Programming User Guide

Introduction

RT PolarFire® FPGAs offer a variety of programming options to diverse end-user applications. The following components of RT PolarFire devices are programmable:

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

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

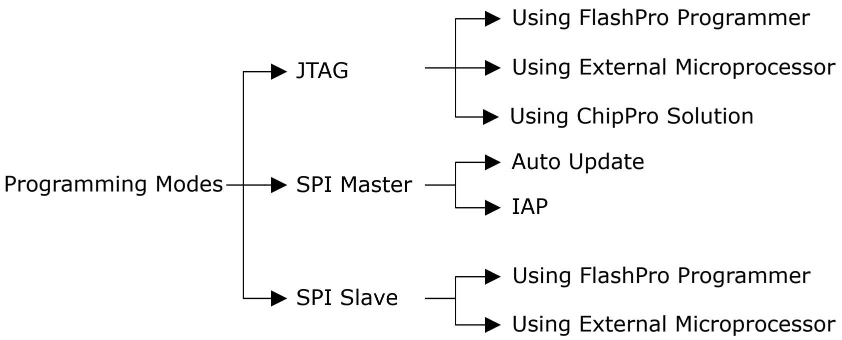
- JTAG
- SPI master
- SPI slave

If System Controller Suspend Mode is enabled, SPI master (In-Application Programming (IAP) and Auto Update) mode is not available to program the device.

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 v5 or later. The external master fetches the programming data (bitstream) from an external memory to program the device.

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 IAP. In auto update, the device reprograms itself on power-up, and in IAP, the device is programmed when the user application initiates programming.

Figure 1. Programming Modes


Programming Modes
JTAG
Using FlashPro Programmer
Using External Microprocessor
Using ChipPro Solution
SPI Master
Auto Update
IAP
SPI Slave
Using FlashPro Programmer
Using External Microprocessor

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

Figure 2. Device Programming Modes and Interfaces

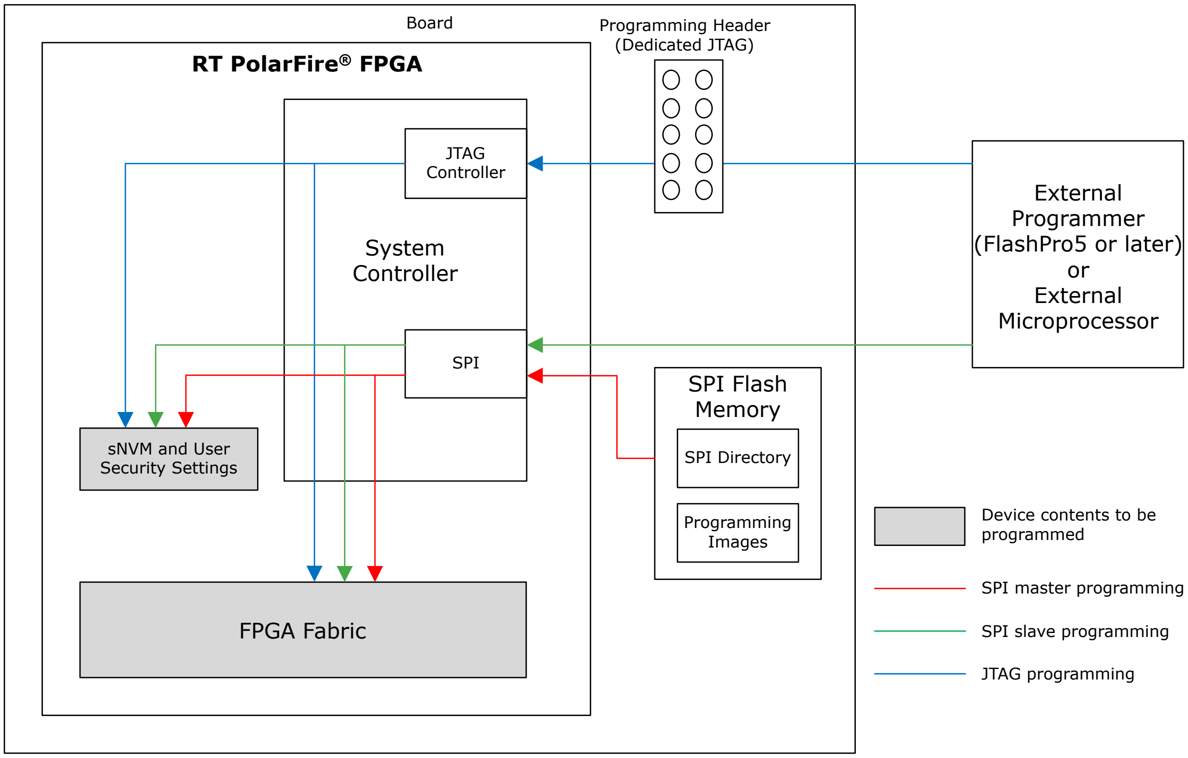


Table of Contents

1. Bitstream Generation

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
- User security settings

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

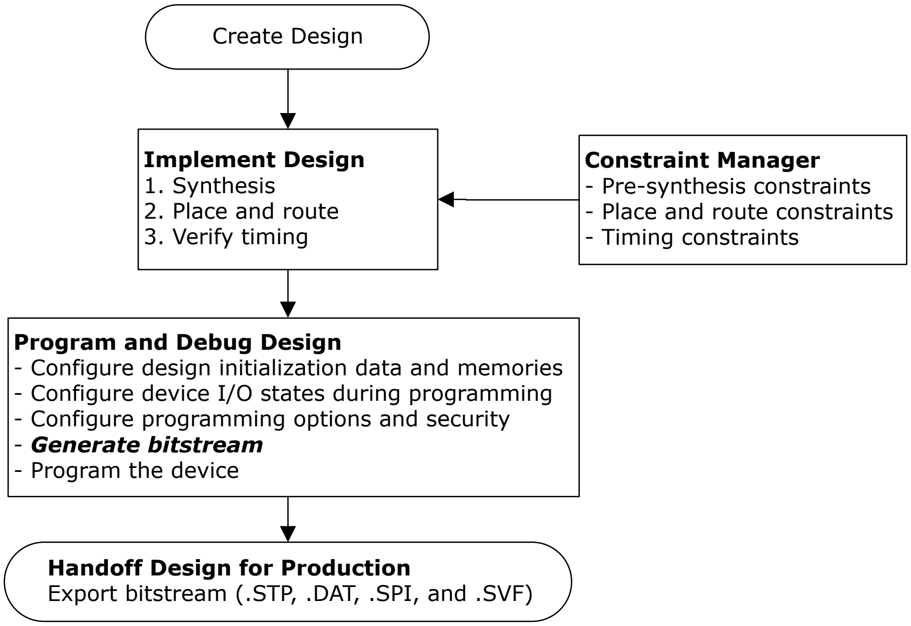
Table 1-1. RT PolarFire FPGA Programming Interfaces and Bitstream Formats

Programming Mode	Interface	Master	Bitstream Format
JTAG programming	System controller's dedicated JTAG	FlashPro programmer	STP
JTAG programming	System controller's dedicated JTAG	External microprocessor	DAT
JTAG programming	System controller's dedicated JTAG	ChipPro solution using FlashPro6	STAPL
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

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

Figure 1-1. Bitstream Generation in Libero Design Flow

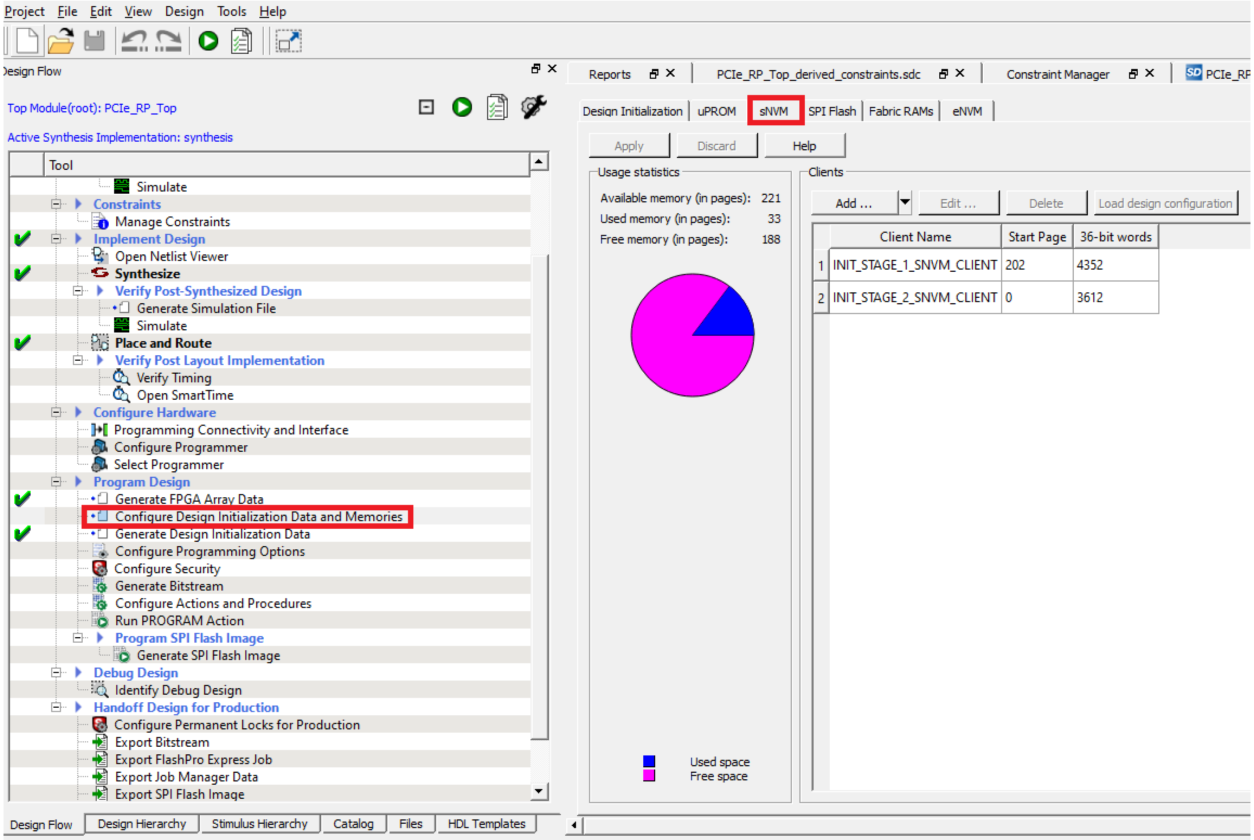

Create Design
Implement Design
1. Synthesis
2. Place and route
3. Verify timing
Constraint Manager
- Pre-synthesis constraints
- Place and route constraints
- Timing constraints
Program and Debug Design
- Configure design initialization data and memories
- Configure device I/O states during programming
- Configure programming options and security
- Generate bitstream
- Program the device
Handoff Design for Production
Export bitstream (.STP, .DAT, .SPI, and .SVF)

1.2 Adding sNVM Data to the Bitstream

The sNVM is a user non-volatile flash memory that can be programmed independently. Each RT PolarFire FPGA has 56 Kbytes 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



Available memory (in pages): 221
Used memory (in pages): 33
Free memory (in pages): 188

	Client Name	Start Page	36-bit words
1	INIT_STAGE_1_SNVM_CLIENT	202	4352
2	INIT_STAGE_2_SNVM_CLIENT	0	3612

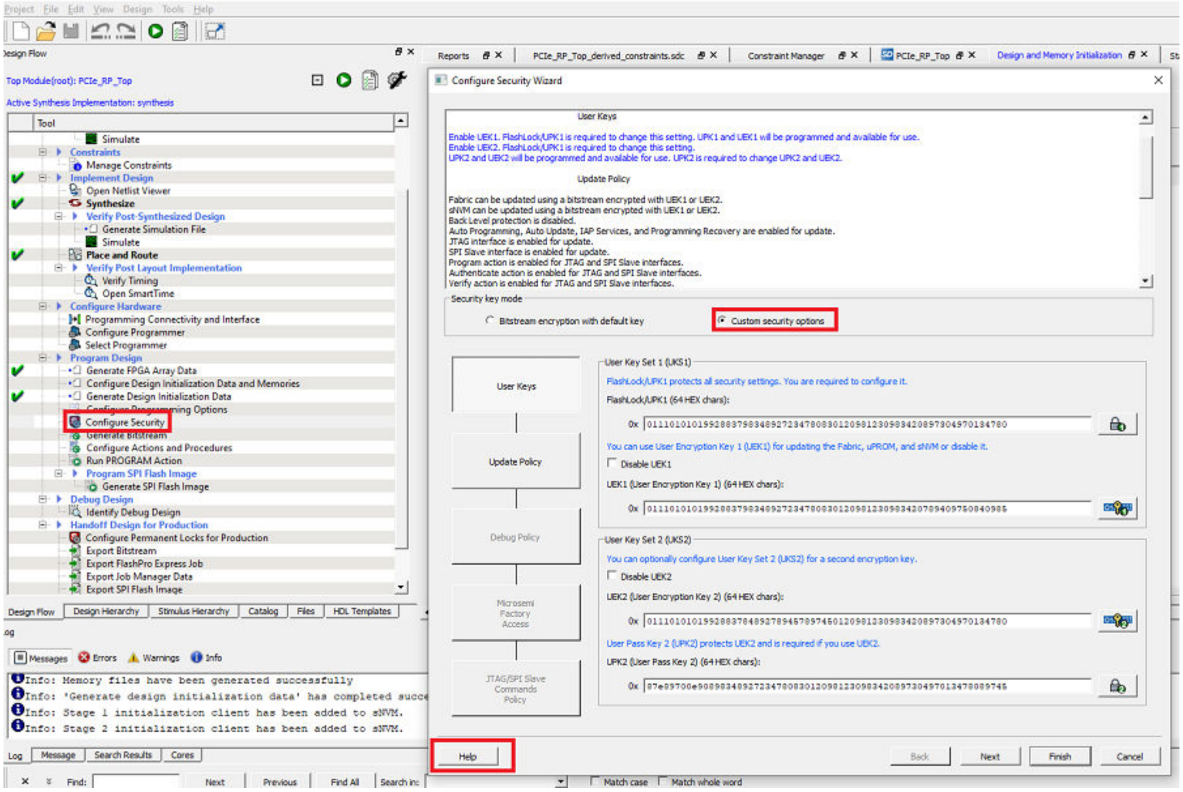
1.3 Adding User Security Settings to the Bitstream

RT PolarFire FPGA 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 1-3. Configure Security—Custom Security Options



Configure Security Wizard

User Keys

Enable UEK1. FlashLock/UPK1 is required to change this setting. UPK1 and UEK1 will be programmed and available for use.
Enable UEK2. FlashLock/UPK1 is required to change this setting.
UPK2 and UEK2 will be programmed and available for use. UPK2 is required to change UPK2 and UEK2.

Update Policy

Fabric can be updated using a bitstream encrypted with UEK1 or UEK2.
sNVM can be updated using a bitstream encrypted with UEK1 or UEK2.
Back Level protection is disabled.
Auto Programming, Auto Update, IAP Services, and Programming Recovery are enabled for update.
JTAG interface is enabled for update.
SPI Slave interface is enabled for update.
Program action is enabled for JTAG and SPI Slave interfaces.
Authenticate action is enabled for JTAG and SPI Slave interfaces.
Verify action is enabled for JTAG and SPI Slave interfaces.

Security key mode

○ Bitstream encryption with default key ● Custom security options

User Key Set 1 (UKS1)

FlashLock/UPK1 protects all security settings. You are required to configure it.

FlashLock/UPK1 (64 HEX chars):

0x 0111010101992883798348927234780830120981230983420897304970134780

You can use User Encryption Key 1 (UEK1) for updating the Fabric, uPROM, and sNVM or disable it.

□ Disable UEK1

UEK1 (User Encryption Key 1) (64 HEX chars):

0x 0111010101992883798348927234780830120981230983420789409750840985

User Key Set 2 (UKS2)

You can optionally configure User Key Set 2 (UKS2) for a second encryption key.

□ Disable UEK2

UEK2 (User Encryption Key 2) (64 HEX chars):

0x 0111010101992883784892789457897460120981230983420897304970134780

User Pass Key 2 (UPK2) protects UEK2 and is required if you use UEK2.

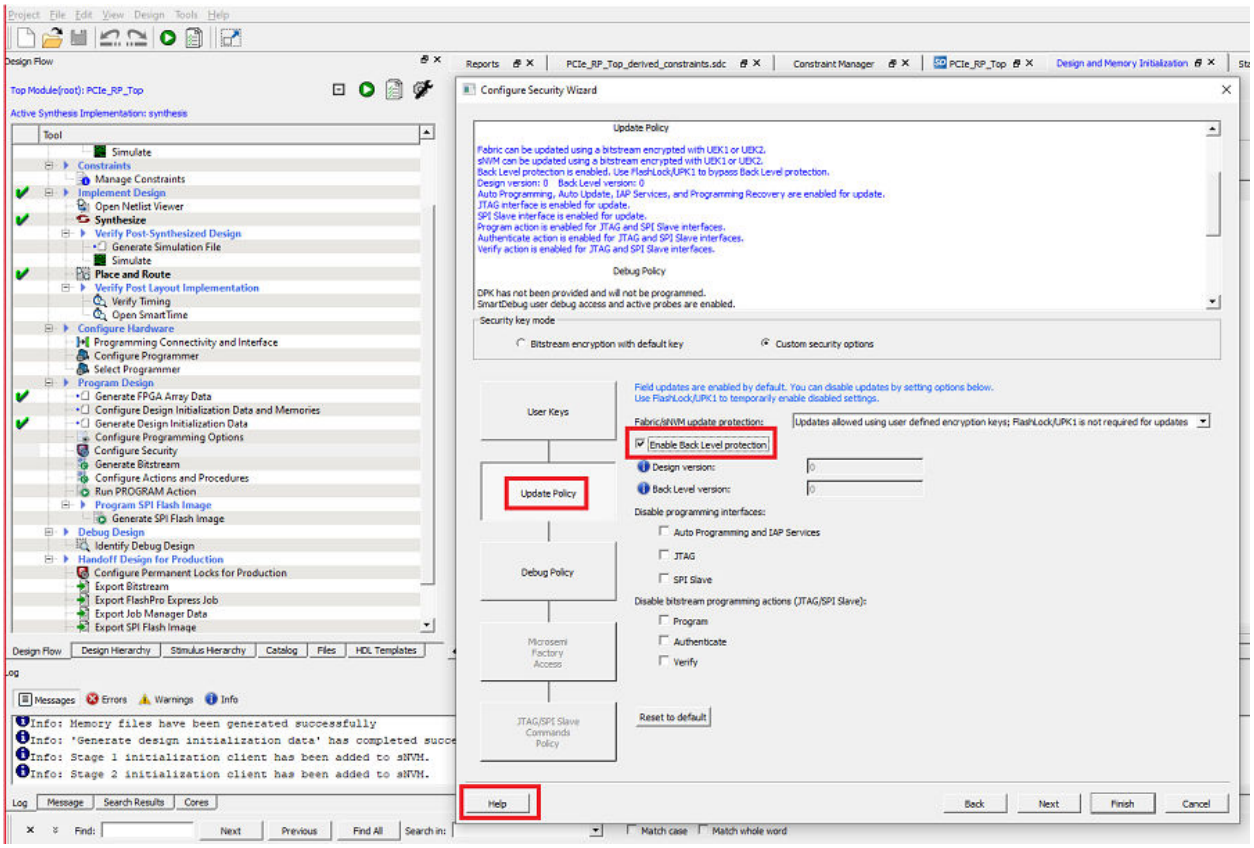
UPK2 (User Pass Key 2) (64 HEX chars):

0x 87e09700e908983489272347808301209812309834208973049701347808974 5

```
Info: Memory files have been generated successfully
Info: 'Generate design initialization data' has completed succe
Info: Stage 1 initialization client has been added to sNVM.
Info: Stage 2 initialization client has been added to sNVM.
```

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

Figure 1-4. Configure Security Wizard—Update Policy



Configure Security Wizard

Update Policy

Fabric can be updated using a bitstream encrypted with UEK1 or UEK2.
sNVM can be updated using a bitstream encrypted with UEK1 or UEK2.
Back Level protection is enabled. Use FlashLock/UPK1 to bypass Back Level protection.
Design version: 0   Back Level version: 0
Auto Programming, Auto Update, IAP Services, and Programming Recovery are enabled for update.
JTAG interface is enabled for update.
SPI Slave interface is enabled for update.
Program action is enabled for JTAG and SPI Slave interfaces.
Authenticate action is enabled for JTAG and SPI Slave interfaces.
Verify action is enabled for JTAG and SPI Slave interfaces.

Debug Policy

DPK has not been provided and will not be programmed.
SmartDebug user debug access and active probes are enabled.

Security key mode

○ Bitstream encryption with default key   ● Custom security options

Field updates are enabled by default. You can disable updates by setting options below.
Use FlashLock/UPK1 to temporarily enable disabled settings.

Fabric/sNVM update protection: Updates allowed using user defined encryption keys; FlashLock/UPK1 is not required for updates

☑ Enable Back Level protection

Design version: 0

Back Level version: 0

Disable programming interfaces:

□ Auto Programming and IAP Services

□ JTAG

□ SPI Slave

Disable bitstream programming actions (JTAG/SPI Slave):

□ Program

□ Authenticate

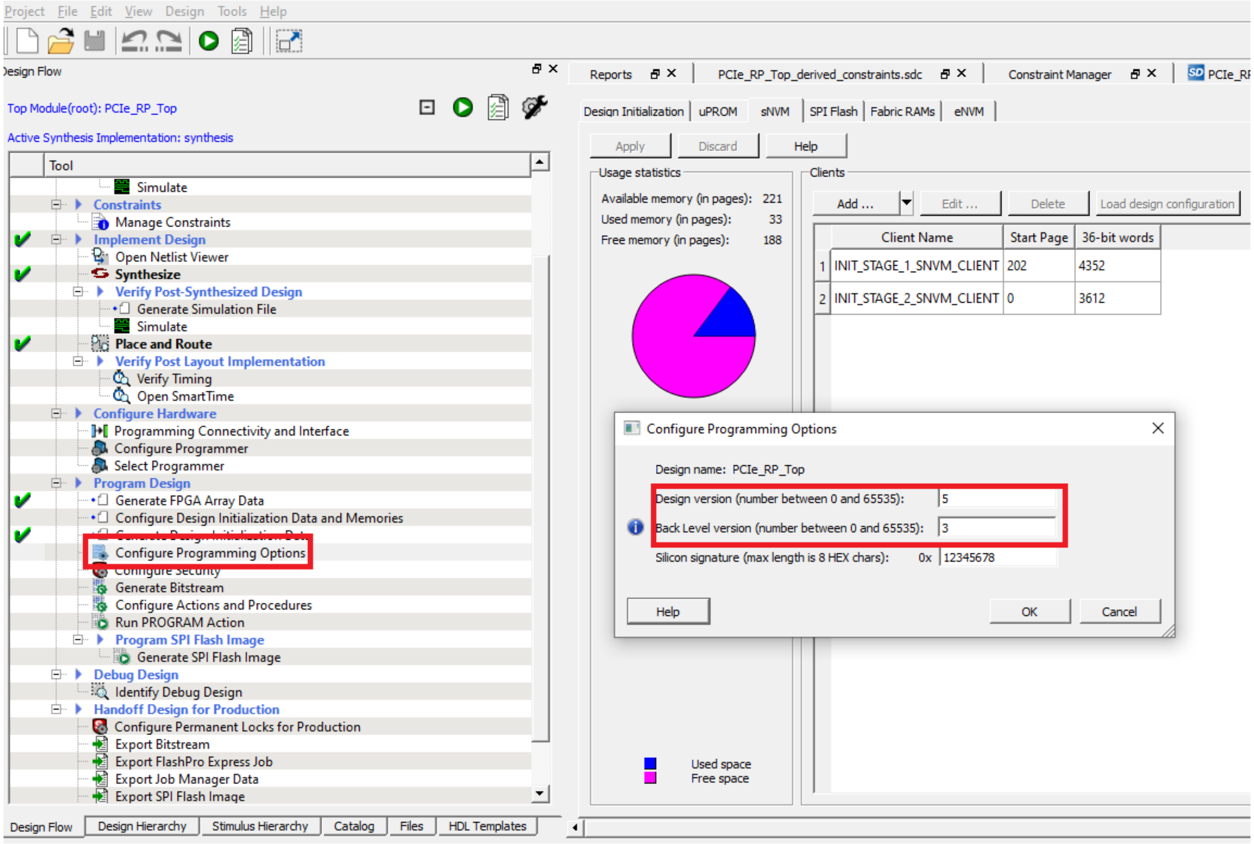
□ Verify

```
Info: Memory files have been generated successfully
Info: 'Generate design initialization data' has completed succe
Info: Stage 1 initialization client has been added to sNVM.
Info: Stage 2 initialization client has been added to sNVM.
```

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 1-5. Configure Programming Options



Usage statistics

Available memory (in pages): 221
Used memory (in pages): 33
Free memory (in pages): 188

	Client Name	Start Page	36-bit words
1	INIT_STAGE_1_SNVM_CLIENT	202	4352
2	INIT_STAGE_2_SNVM_CLIENT	0	3612

Design name: PCIe_RP_Top

Design version (number between 0 and 65535): 5

Back Level version (number between 0 and 65535): 3

Silicon signature (max length is 8 HEX chars): 0x 12345678

For more information about the bypass back-level protection, see 6.1. Bypass Back Level Protection Use Case.

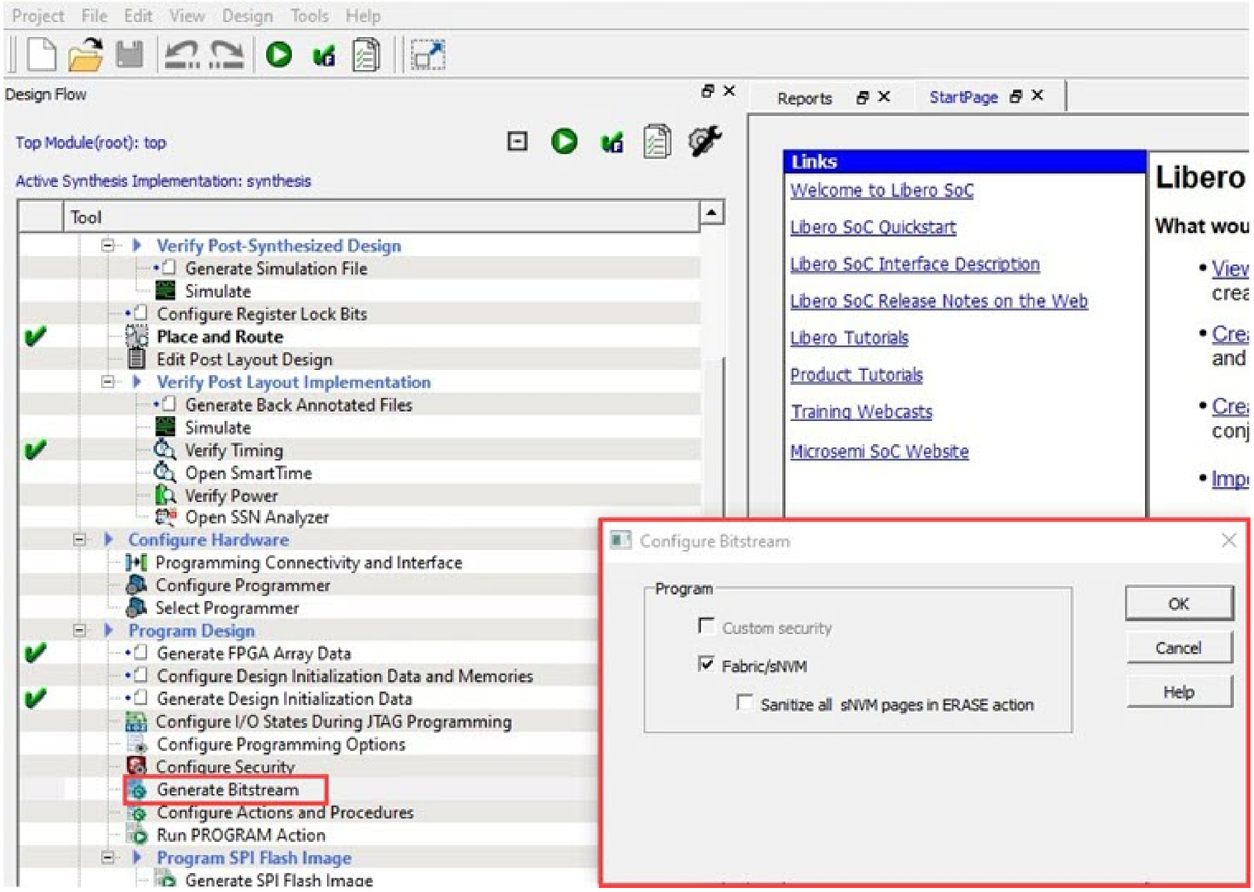
1.4 Configuring Bitstream Components

To configure security settings, and bitstream components such as fabric and sNVM, 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**, and **sNVM**.
4. Click **OK**.

Figure 1-6. Configure Bitstream Window



Program

□ Custom security

☑ Fabric/sNVM

□ Sanitize all sNVM pages in ERASE action

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

Note: 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.

1.5 Programming File Size

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 programming file sizes when custom security is disabled.

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

		RT PolarFire FPGA
		RTPF500
Fabric and sNVM (kB)	STAPL	23446
	DAT	14780
	SPI	14776

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

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

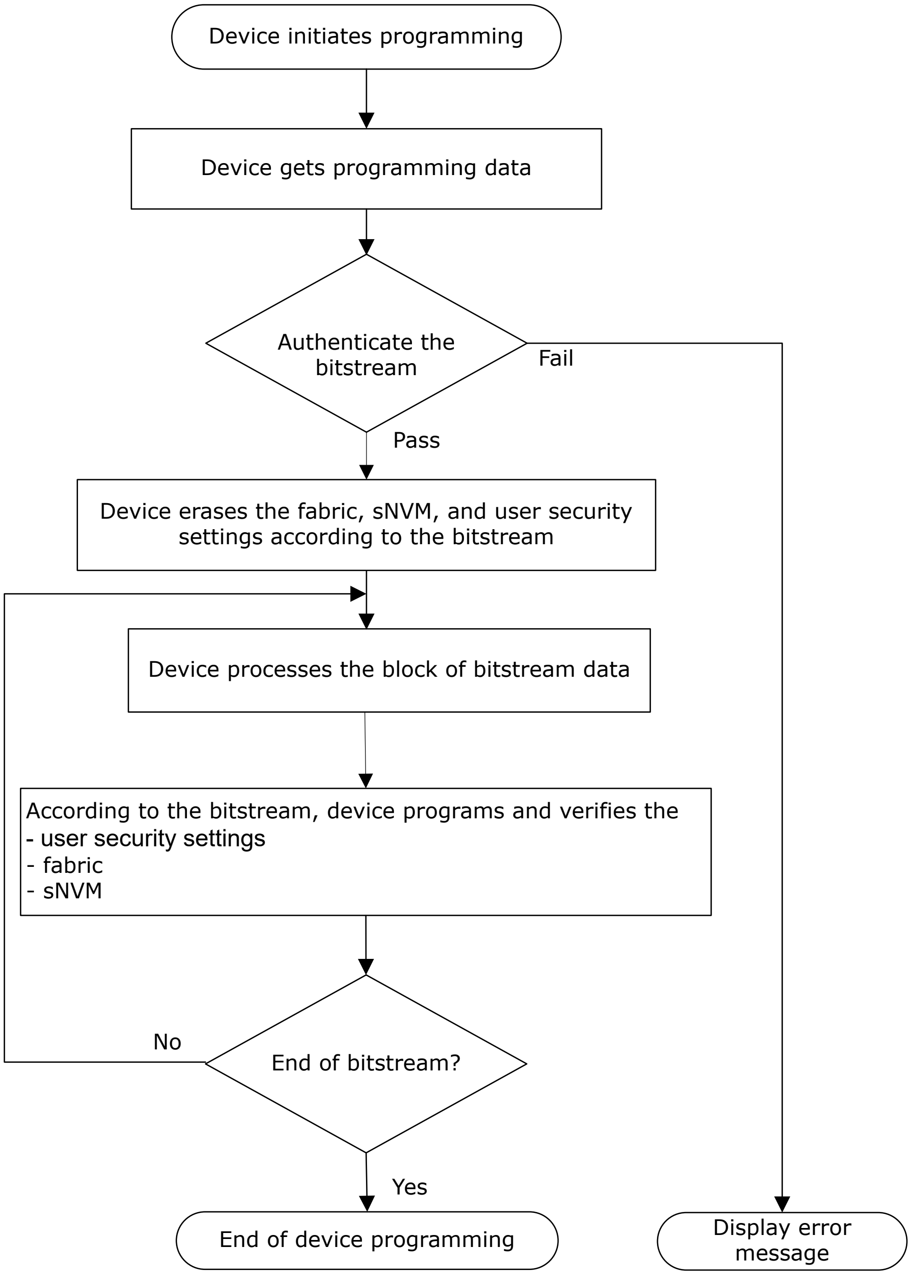
			RT PolarFire FPGA
			RTPF500
Custom Security, Fabric , and sNVM (kB)	STAPL	Master Files	23446
		UEK1/UEK2	23439
	DAT	Master Files	14780
		UEK1/UEK2	14775
	SPI	Master Files	14776
		UEK1/UEK2	14773
Custom Security (kB)	STAPL	NA	87
	DAT	NA	8
	SPI	NA	4

2. Device Programming Flow

The device programming flow starts when the system controller receives or initiates device programming and ends when the bitstream data is 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


Device initiates programming
Device gets programming data
Authenticate the bitstream
Fail
Pass
Device erases the fabric, sNVM, and user security settings according to the bitstream
Device processes the block of bitstream data
According to the bitstream, device programs and verifies the
- user security settings
- fabric
- sNVM
No
End of bitstream?
Yes
End of device programming
Display error message

Note: Programming cycle count is incremented for both programming and erase operations, since 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

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 JTAG programming time of RT PolarFire FPGA is less than 147 seconds. For information about programming time for specific devices and programming modes, see RT PolarFire Datasheet.

3. System Controller Suspend Mode

To protect the device from unintended behavior due to single event upset (SEU), the system controller can be held in Suspend mode after device initialization. The system controller is active if the device is power-cycled or if a hard reset is applied. But it returns to Suspend Mode, once the initialization cycle is completed. A Sonos bit that is programmed during device programming controls the System Controller Suspend Mode. This Sonos bit is not accessible from the customer design or by any external pin. The flash bit is only accessible through the programming file loaded into the device.

As the control bit is stored in a flash cell, it is immune to radiation effects due to the following:

- Neutrons or alpha particles in terrestrial and airborne applications
- Heavy ions in space applications

While the FPGA is in System Controller Suspend Mode, programming via JTAG or SPI-Slave mode can be accomplished, by asserting the JTAG_TRSTB pin HIGH, to temporarily remove the FPGA from System Controller Suspend Mode. If the JTAG_TRSTB pin is LOW, all the other JTAG input signals are blocked from activating the system controller.

For programming, run **Scan Chain** using FlashPro software (from the **Menu** bar, click **Programmers** > **Scan Chain**), which keeps the JTAG_TRSTB pin HIGH. Keeping the JTAG_TRSTB pin HIGH causes system controller to exit from Suspend Mode and then program the device.

When in space, the JTAG_TRSTB pin must be held LOW using one of the following methods:

- Hardwired to ground
- Connected to ground through a jumper
- Tied to ground through a pull-down, if an active device is included in the circuit to allow on-orbit reprogramming

To restore normal operation, the device must be reprogrammed using the JTAG port with the System Controller Suspend Mode bit turned off, that is, disable the System Controller Suspend Mode in Libero SoC software, regenerate the bitstream, and reprogram the device.

When RT PolarFire FPGAs are used in System Controller Suspend Mode, device programming is disabled to protect the device from unintended programming because of SEUs. 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 the System Controller Suspend Mode, it can be programmed as usual.

If the System Controller Suspended Mode is disabled, it increases vulnerability to radiation single event effects (SEEs) in the System Controller.

The following table lists the programming support when System Controller Suspend Mode is enabled or disabled.

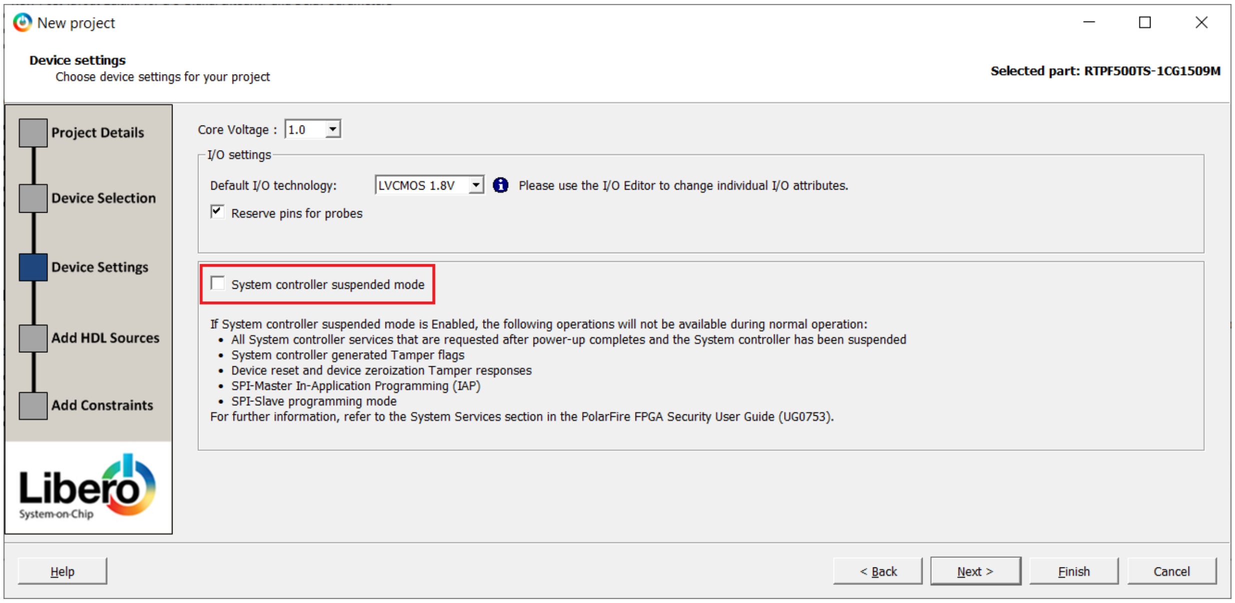
Table 3-1. Programming Support

Programming Mode	System Controller Suspend Mode	Programming Support
JTAG	Disabled	Supported
JTAG	Enabled	Supported – requires System Controller Suspend Mode to be temporarily disabled by asserting JTAG_TRSTB.
SPI Slave	Disabled	Supported
SPI Slave	Enabled	Supported – requires System Controller Suspend Mode to be temporarily disabled by asserting JTAG_TRSTB.
SPI Master	Disabled	Supported
SPI Master	Enabled	Not supported

3.1 Configuring System Controller Suspend Mode

The System Controller Suspend Mode can be configured, that is, enabled or disabled in the Libero SoC software, as shown in the following figure.

Figure 3-1. System Controller Suspend Mode



New project

Device settings

Choose device settings for your project

Selected part: RTPF500TS-1CG1509M

Core Voltage : 1.0

I/O settings

Default I/O technology: LVCMOS 1.8V

Please use the I/O Editor to change individual I/O attributes.

☑ Reserve pins for probes

□ System controller suspended mode

If System controller suspended mode is Enabled, the following operations will not be available during normal operation:

- All System controller services that are requested after power-up completes and the System controller has been suspended
- System controller generated Tamper flags
- Device reset and device zeroization Tamper responses
- SPI-Master In-Application Programming (IAP)
- SPI-Slave programming mode

For further information, refer to the System Services section in the PolarFire FPGA Security User Guide (UG0753).

If System Controller Suspend Mode is enabled, the following operations are not available during normal operation:

- SPI master programming mode (IAP and Auto Update).
- SPI slave programming mode.
- All system controller services that are requested after power-up completes and the system controller is suspended.
- System controller generated Tamper flags.
- Device reset and device zeroization Tamper responses.

4. Programming Modes

This section describes the three programming modes in detail.

4.1 JTAG Programming

RT PolarFire FPGA device has 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.

4.1.1 JTAG Programming Interface

In RT PolarFire FPGA, the JTAG pins are located in a dedicated I/O Bank 3. For information about the I/O states during JTAG programming, see 7. I/O States During Programming.

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

Table 4-1. JTAG Pins

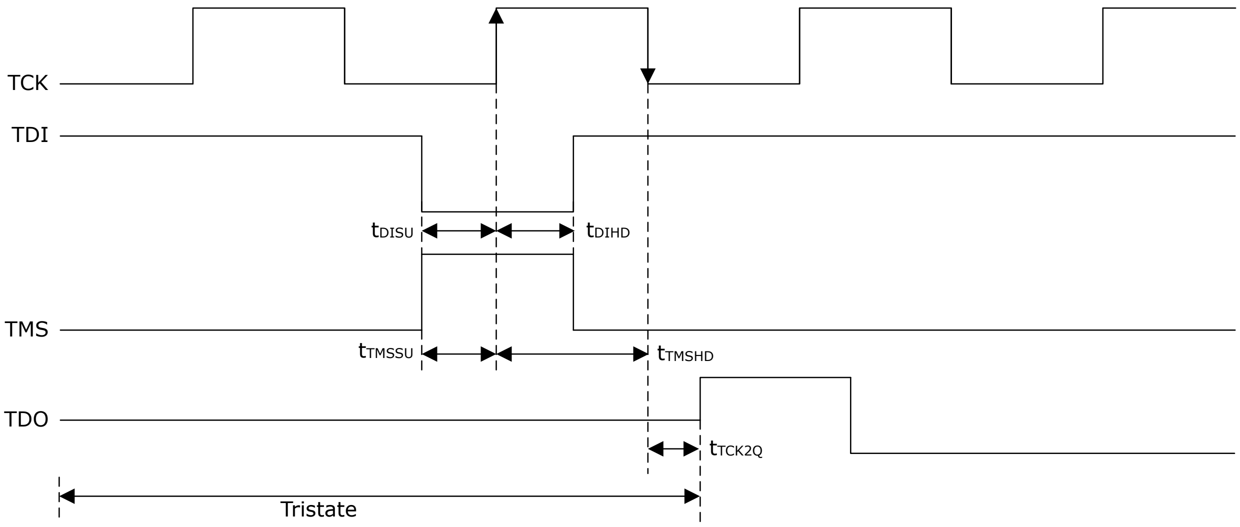
Pin Name	Direction	Weak Pull-Up/Unused Condition	Description
TMS	Input	Yes/DNC	JTAG test mode select.
TRSTB	Input	Yes[1]	JTAG test reset. Must be held low during device operation.
TDI	Input	Yes/DNC	JTAG test data in. In ATPG or test mode, when using a 4-bit TDI bus, this I/O is used as tdi[0].
TCK	Input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

[1] If TRSTB is unused and the System Controller is in suspend 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 an external source.
[2] In unused condition, must be connected to VSS through 10 kΩ resistor.

4.1.2 JTAG Timing

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 the RT PolarFire FPGA Datasheet.

Figure 4-1. JTAG Signals Timing Diagram

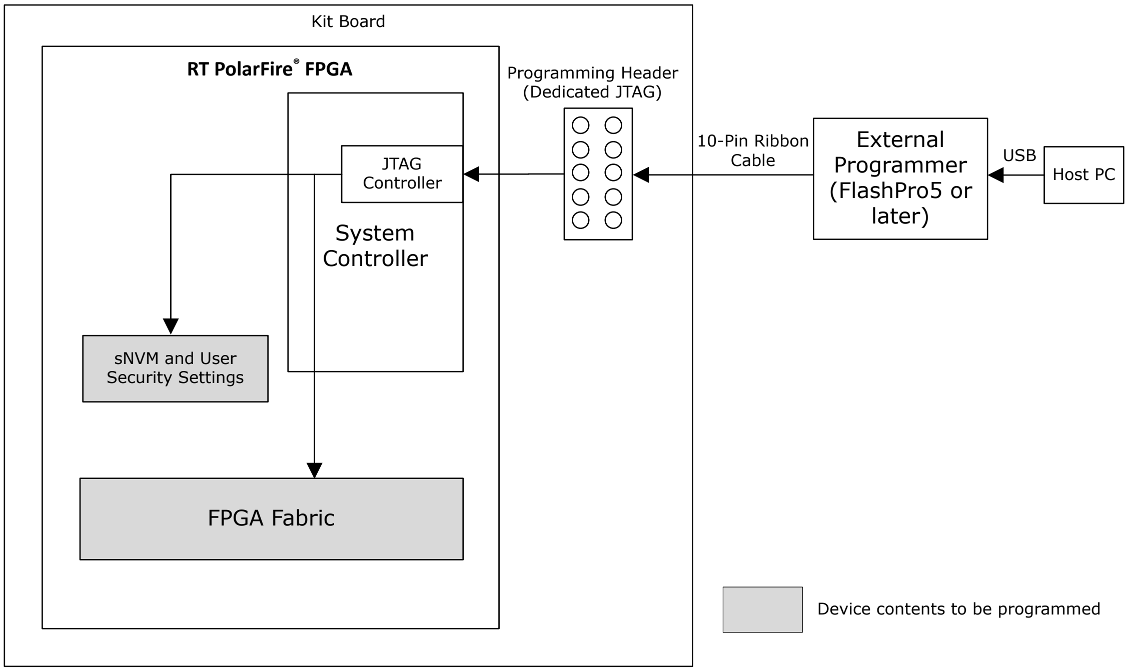


4.1.3 JTAG Programming Using FlashPro Programmer

Microchip FlashPro programmer v5 or later can be used to program RT PolarFire FPGA 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 4-2. JTAG Programming Using External Programmer


Kit Board
RT PolarFire® FPGA
JTAG Controller
System Controller
sNVM and User Security Settings
FPGA Fabric
Programming Header (Dedicated JTAG)
10-Pin Ribbon Cable
External Programmer (FlashPro5 or later)
USB
Host PC
Device contents to be programmed

The following table lists the FlashPro header signals.

Table 4-2. FlashPro Header Signals

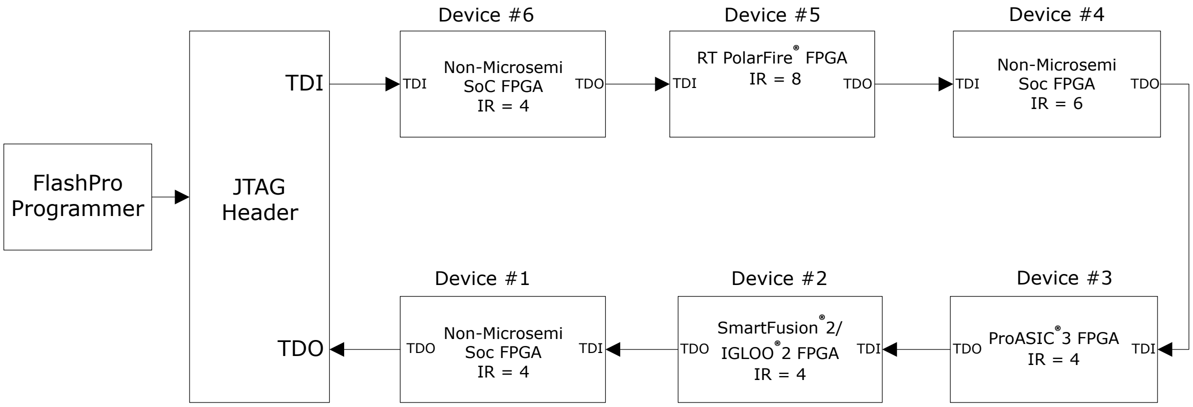
Pin Number	Signal	Direction to FlashPro Programmer	Description
1	TCK/SCK	Output	JTAG/SPI clock.
2	GND	—	Signal reference. GND pins must be connected.
3	TDO/MISO	Input	JTAG/SPI data output from programming device.
4	PROG_MODE	Not connected	Unused
5	TMS/SS	Output	JTAG test-mode select/SPI slave select.
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's Guide.

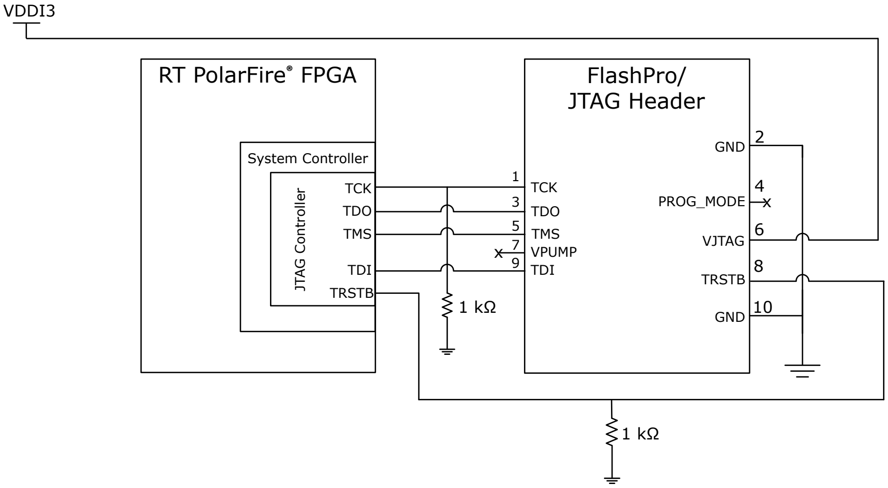
Figure 4-3. Device Programming in JTAG Chain


Device #6
Device #5
Device #4
TDI
Non-Microsemi SoC FPGA IR = 4
RT PolarFire® FPGA IR = 8
Non-Microsemi Soc FPGA IR = 6
TDO
FlashPro Programmer
JTAG Header
Device #1
Device #2
Device #3
Non-Microsemi Soc FPGA IR = 4
SmartFusion®2/ IGLOO®2 FPGA IR = 4
ProASIC®3 FPGA IR = 4

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

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

Figure 4-4. Connecting FlashPro Programmer to a Device



4.1.4 JTAG Programming Using External Microprocessor

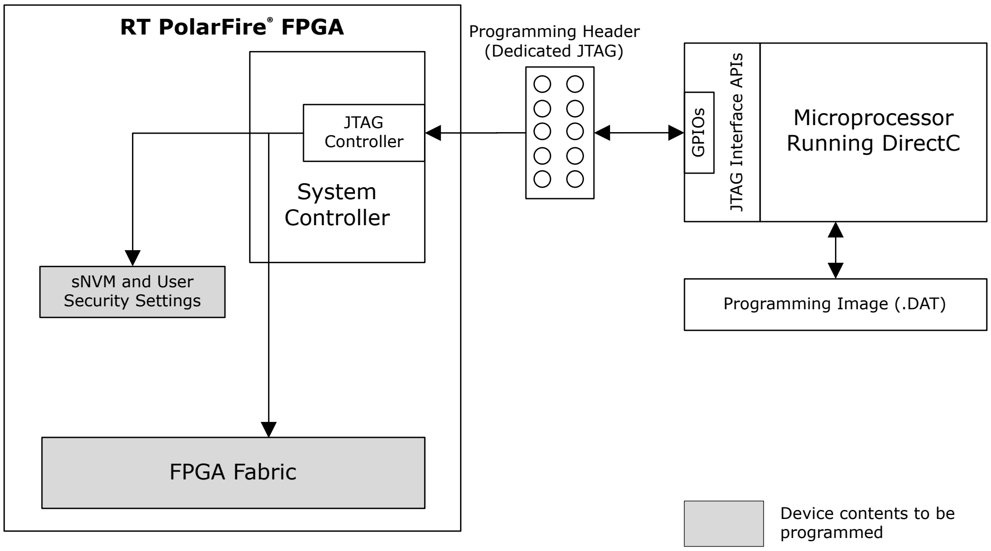
An external microprocessor can be used to program the device through the dedicated JTAG interface. This type of programming requires that the external microprocessor run DirectC, a Microchip programming solution for FPGAs, and the microprocessor's GPIO ports drive the JTAG interface.

Note: The DirectC solution supports programming of the FPGA fabric, sNVM, 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 Microchip 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 4-5. Programming Using External Microprocessor


RT PolarFire® FPGA
JTAG Controller
System Controller
sNVM and User Security Settings
FPGA Fabric
Programming Header (Dedicated JTAG)
GPIOs
JTAG Interface APIs
Microprocessor Running DirectC
Programming Image (.DAT)
Device contents to be programmed

4.1.5 JTAG Programming Using ChipPro Solution

The ChipPro programmer baseboard with FlashPro6 can be used to program the device through the dedicated JTAG interface. This can be done either using the Libero SoC or a standalone FlashPro Express. For information about ChipPro, see CP-PROG-BASE.

4.2 SPI Slave Programming

RT PolarFire FPGA can be programmed using an external SPI master such as an external microprocessor or a FlashPro programmer through the SPI interface. See Table 4-4 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 4-3.

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.

Note: SPI-Slave programming can be accomplished while System Controller Suspend Mode is enabled, by temporarily exiting System Controller Suspend Mode by holding JTAG_TRSTB HIGH.

4.2.1 SPI Slave Programming Interface

In addition to the standard SPI signals, RT PolarFire FPGA 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 UG0726: PolarFire FPGA Board Design User Guide.

Table 4-3. System Controller SPI Pins

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

[1] Shared between the system controller and the FPGA fabric. 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). When the SPI_EN pin is disabled (driven low) or when the SS is driven HIGH, the system controller's SPI outputs 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 4-4. System Controller's SPI Configuration - SPI Slave

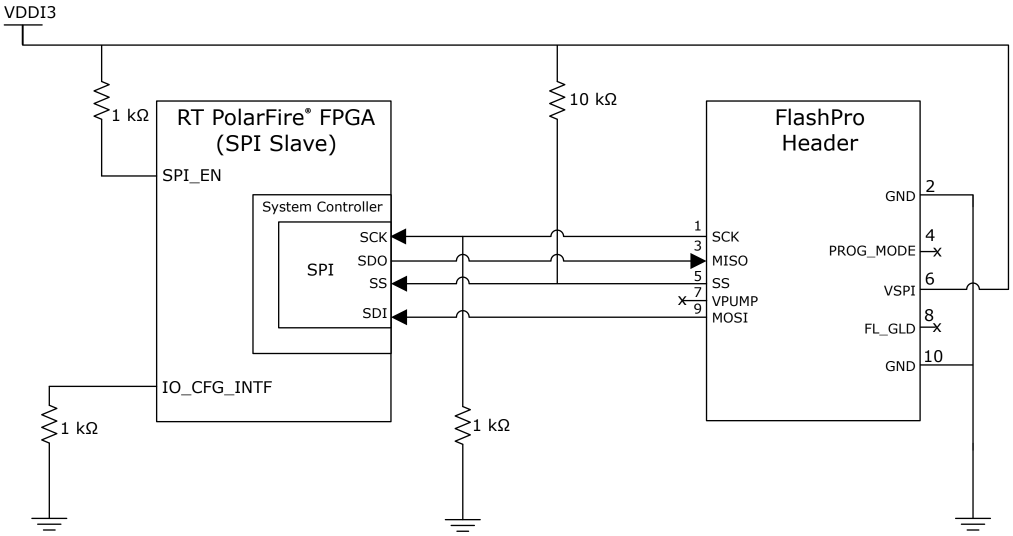
SPI Pins		SPI Slave Programming	Description
SPI_EN	IO_CFG_INTF		
0	x	No	Dynamic switching from Slave to Master or vice-versa is not allowed. A power-cycle or device reset (DEVRST_N) is required to change the SPI configuration from Slave to Master or vice-versa by configuring the IO_CFG_INTF pin.
1	0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

4.2.2 SPI Slave Programming Using FlashPro Programmer

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 4-6. 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**.

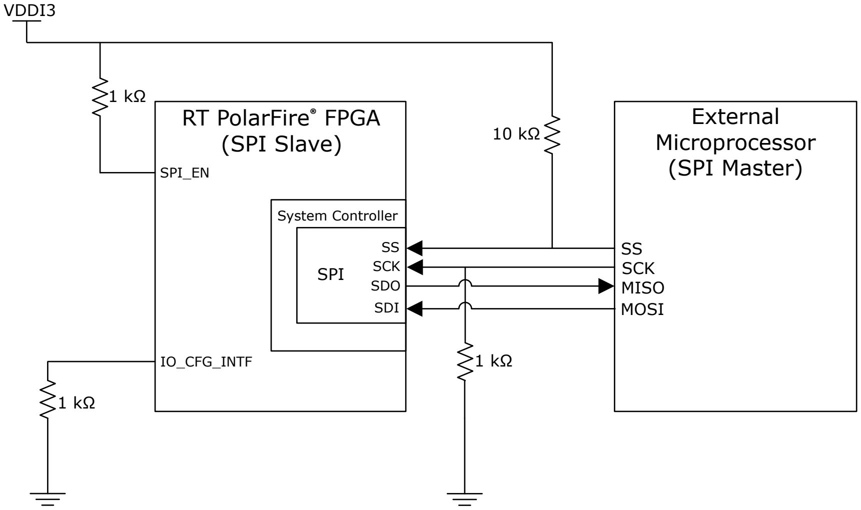
4.2.3 SPI Slave Programming Using External Microprocessor

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, 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 Microchip DirectC solution webpage. The example project (Direct-C installer) is also available on the Downloads tab.

For information about FlashPro header signals, see Table 4-2.

Figure 4-7. SPI Slave Programming Using External Microprocessor



4.3 SPI Master Programming

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 11. 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 found to be 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 4-5. 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 7. I/O States During Programming.

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

Table 4-5. 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[1]	Automatically recovers the device from programming failure	Device power failure during programming

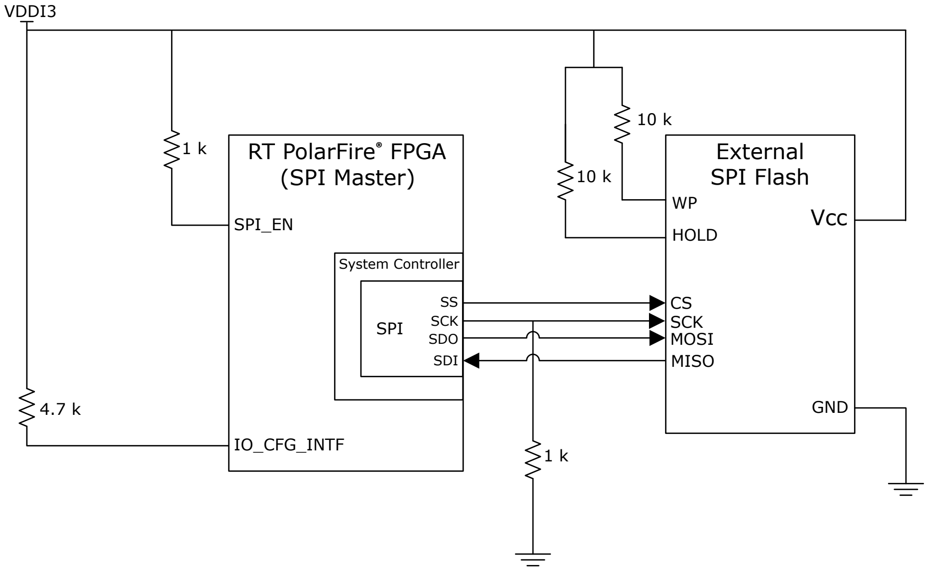
[1] If there is a power interruption while Auto update or IAP is updating the sNVM, then the auto-recovery is not triggered. Though, if the sNVM is not updated completely because of a power interruption, the device starts up and will attempt to boot as normal. However, the partially programmed sNVM causes the user design to malfunction. In this case, the user needs to use the VERIFY action or Digest Check to determine if the programming is successful.

Note: If System Controller Suspend Mode is enabled, SPI master (IAP and Auto Update) programming mode is not available to program the device.

For information about implementing Auto update and IAP, see AC466: 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 4-8. Recommended Board Configuration for SPI Master Programming



4.3.1 SPI Master Programming Interface

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 4-6. 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

4.3.1.1 System Controller SPI Mode and Clock

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.

4.3.2 System Services

RT PolarFire 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:

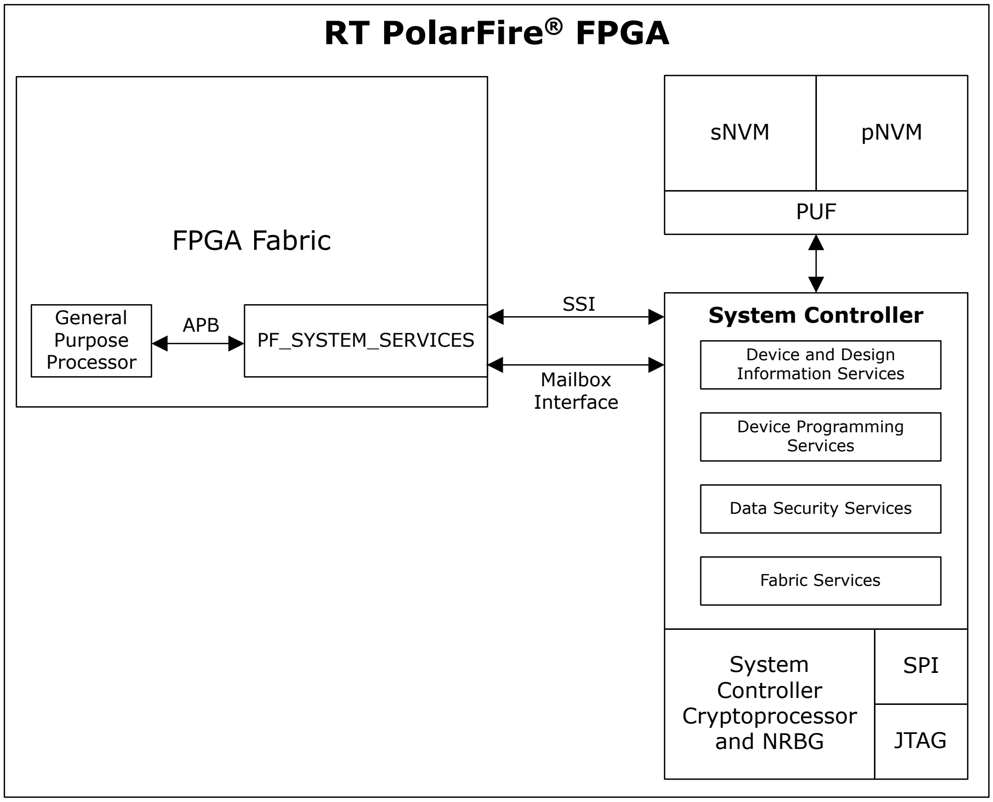
- Bitstream authentication
- IAP image authentication
- Auto update
- IAP

4.3.2.1 System Services

In RT 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 4-9. Design Interface Between Fabric and System Controller

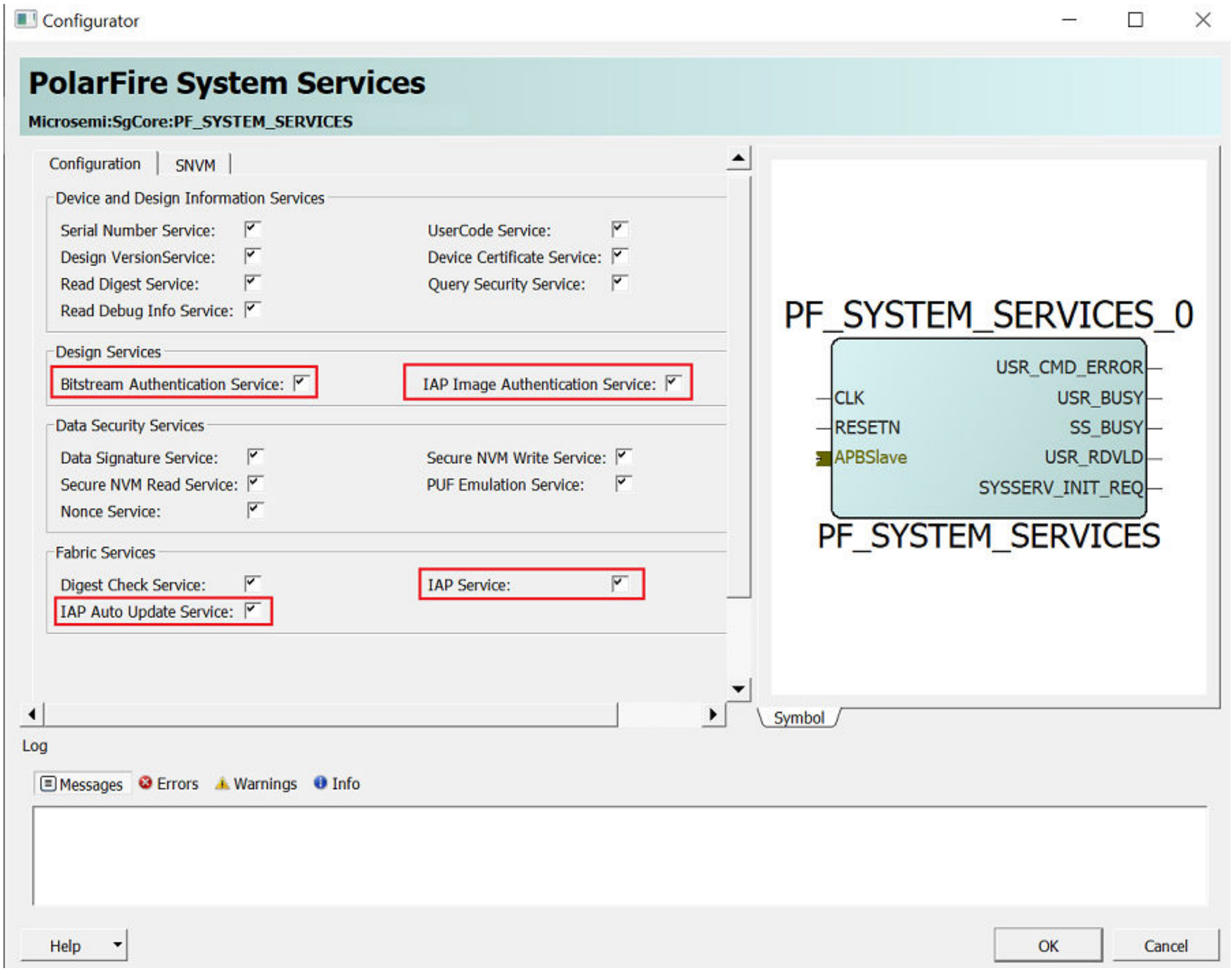


For information about system services driver and example SoftConsole project, see *Firmware Catalog*, which is available in the Libero SoC installation package.

4.3.2.1.1 RT PolarFire System Services Configurator

The following figure shows the RT PolarFire System Services Configurator.

Figure 4-10. PolarFire FPGA Core System Services Configurator



Device and Design Information Services

- Serial Number Service: ☑
- Design VersionService: ☑
- Read Digest Service: ☑
- Read Debug Info Service: ☑
- UserCode Service: ☑
- Device Certificate Service: ☑
- Query Security Service: ☑

Design Services

- Bitstream Authentication Service: ☑
- IAP Image Authentication Service: ☑

Data Security Services

- Data Signature Service: ☑
- Secure NVM Read Service: ☑
- Nonce Service: ☑
- Secure NVM Write Service: ☑
- PUF Emulation Service: ☑

Fabric Services

- Digest Check Service: ☑
- IAP Auto Update Service: ☑
- IAP Service: ☑

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 4-10. For more information, see UG0848: PolarFire System Services User Guide.

4.3.2.2 System Service Request

In RT PolarFire 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 Kbytes 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 UG0848 PolarFire System Services User Guide.

Table 4-7. RT PolarFire 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.

4.3.2.3 Bitstream and IAP Image Authentication System Services

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 RT PolarFire 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.

4.3.2.3.1 Bitstream Authentication System Service

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 4-8. Bitstream Authentication Service Request

System Service Descriptor Bit Field	Value	Description
15:7	MBOXADDR[10:2]	Mailbox address. For the format, see Table 4-9.
6:0	23H	Bitstream authentication command code.

The following table describes the bitstream authentication service mailbox format.

Table 4-9. 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.

4.3.2.3.2 IAP Image Authentication System Service

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 4-10. 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.

4.3.2.3.3 Authentication Service Status Codes

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 12. Appendix: Error Codes.

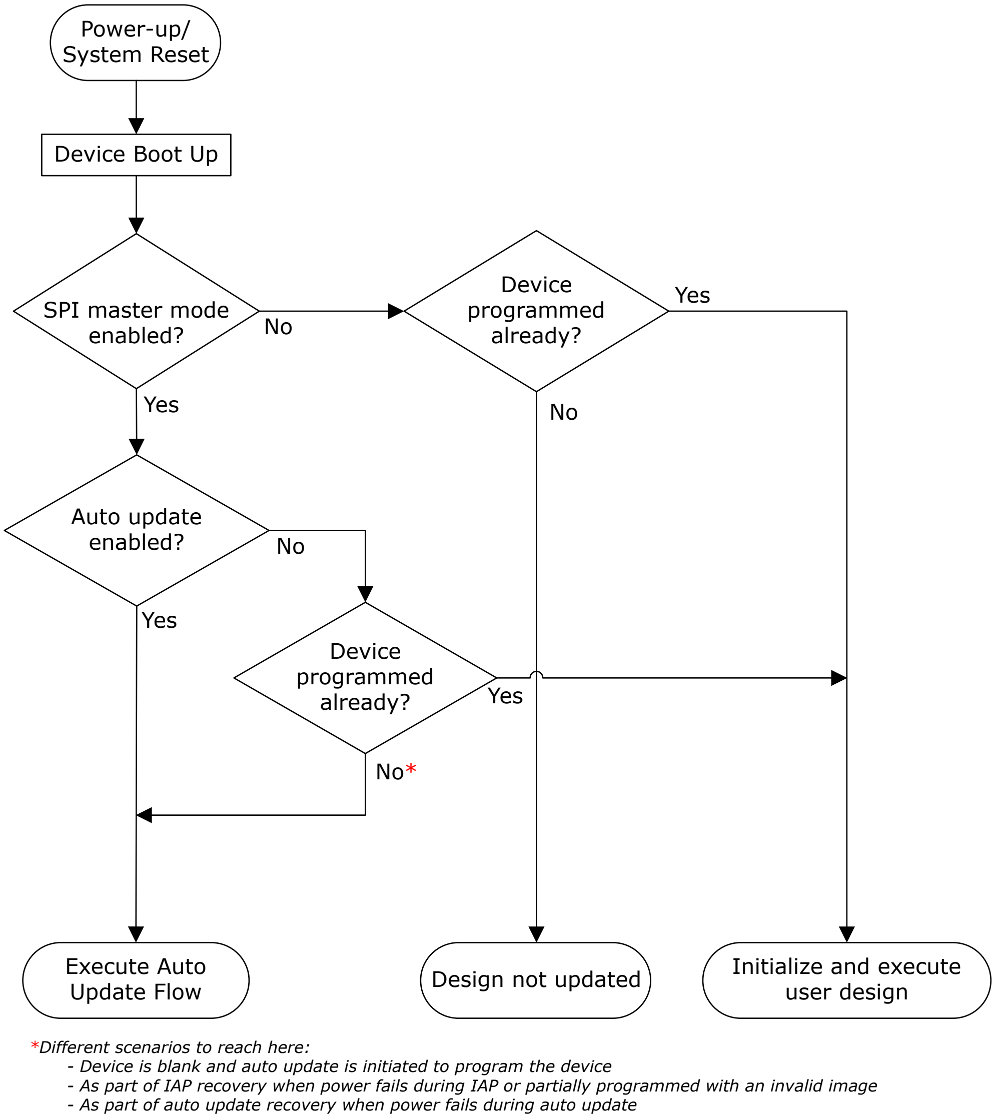
4.3.2.3.4 Usage of Authentication System Services

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.

4.3.3 Auto Update

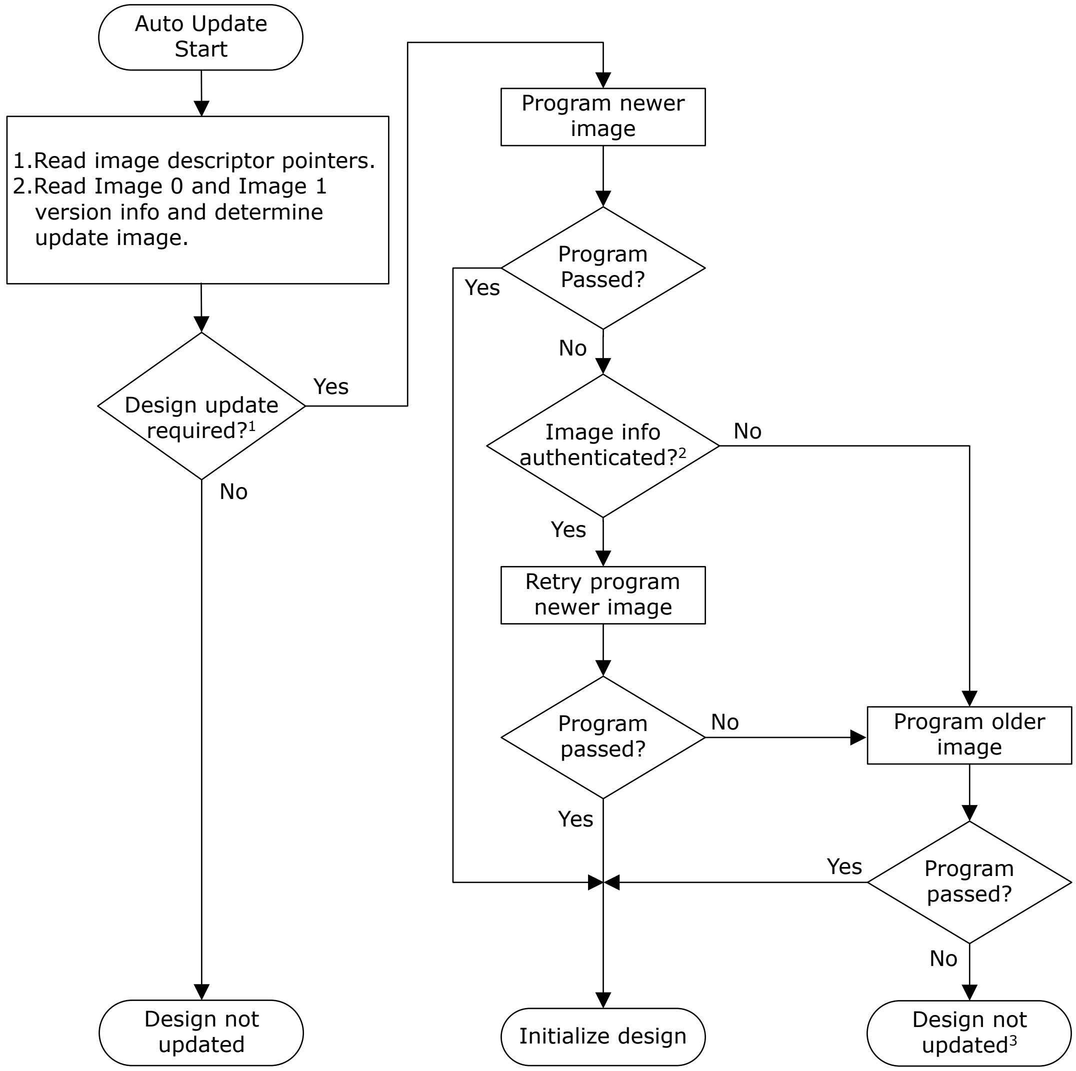
For auto update to occur, the auto update feature needs to be enabled in the user design. On power-up, the device selects the newer version of the first two images stored in the SPI directory. If the version of the newer image does not match that of the currently programmed image, then auto update occurs.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

Figure 4-11. Auto Update High-Level Flowchart


Power-up/ System Reset
Device Boot Up
SPI master mode enabled?
No
Yes
Device programmed already?
Yes
No
Auto update enabled?
No
Yes
Device programmed already?
Yes
No*
Execute Auto Update Flow
Design not updated
Initialize and execute user design
*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 4-12. 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 4-11. 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

4.3.3.1 Auto Update on a Blank Device (Auto Programming)

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 1.3. Adding User Security Settings to the Bitstream.

4.3.3.2 Auto Update on a Pre-programmed Device

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 4-12. 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.

4.3.3.3 Recovery on Auto Update Programming Failure

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.

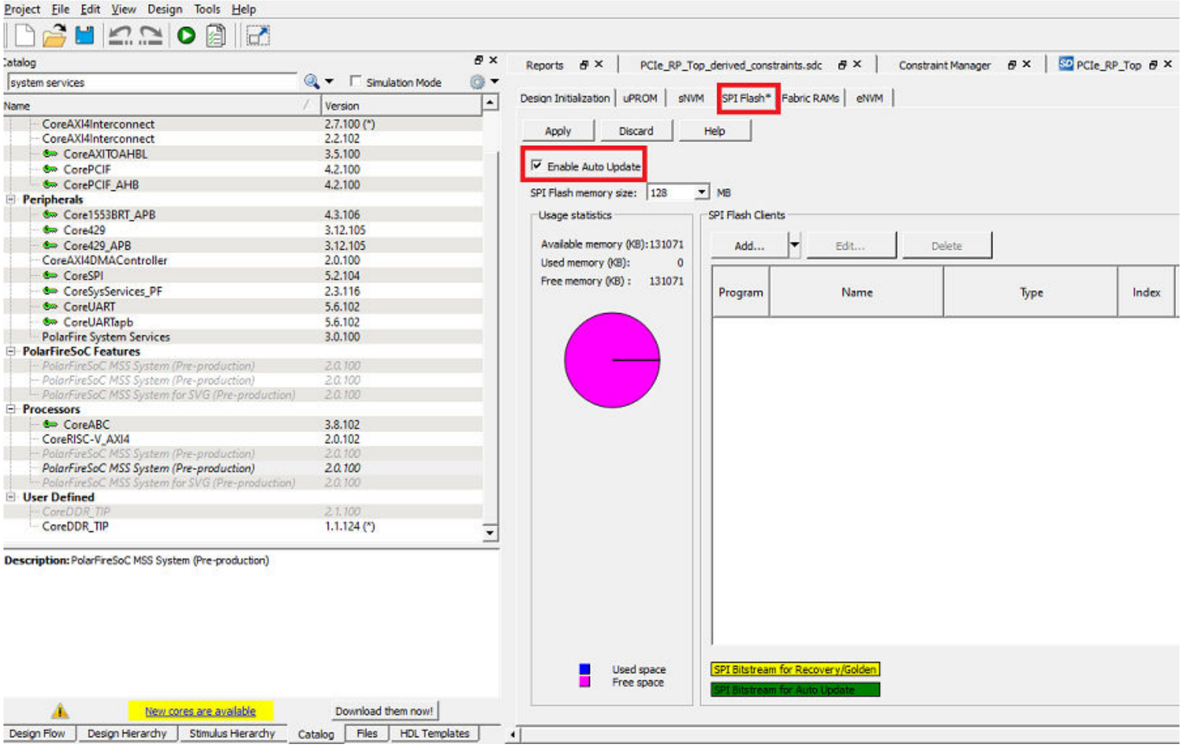
Note: If the device fails to program the newer image, it retries once before programming itself with the older version of the image. If the device remains blank at the end of auto update, there is no indication through I/O and user intervention is required.

4.3.3.4 Enabling Auto Update Option in User Design

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 4-13. Auto Update Setting



system services

Name	Version
CoreAXI4Interconnect	2.7.100 (*)
CoreAXI4Interconnect	2.2.102
CoreAXITOAHBL	3.5.100
CorePCIF	4.2.100
CorePCIF_AHB	4.2.100
Peripherals	
Core1553BRT_APB	4.3.106
Core429	3.12.105
Core429_APB	3.12.105
CoreAXI4DMAController	2.0.100
CoreSPI	5.2.104
CoreSysServices_PF	2.3.116
CoreUART	5.6.102
CoreUARTapb	5.6.102
PolarFire System Services	3.0.100
PolarFireSoC Features	
PolarFireSoC MSS System (Pre-production)	2.0.100
PolarFireSoC MSS System (Pre-production)	2.0.100
PolarFireSoC MSS System for SVG (Pre-production)	2.0.100
Processors	
CoreABC	3.8.102
CoreRISC-V_AXI4	2.0.102
PolarFireSoC MSS System (Pre-production)	2.0.100
PolarFireSoC MSS System (Pre-production)	2.0.100
PolarFireSoC MSS System for SVG (Pre-production)	2.0.100
User Defined	
CoreDDR_TIP	2.1.100
CoreDDR_TIP	1.1.124 (*)

Description: PolarFireSoC MSS System (Pre-production)

☑ Enable Auto Update

SPI Flash memory size: 128 MB

Usage statistics

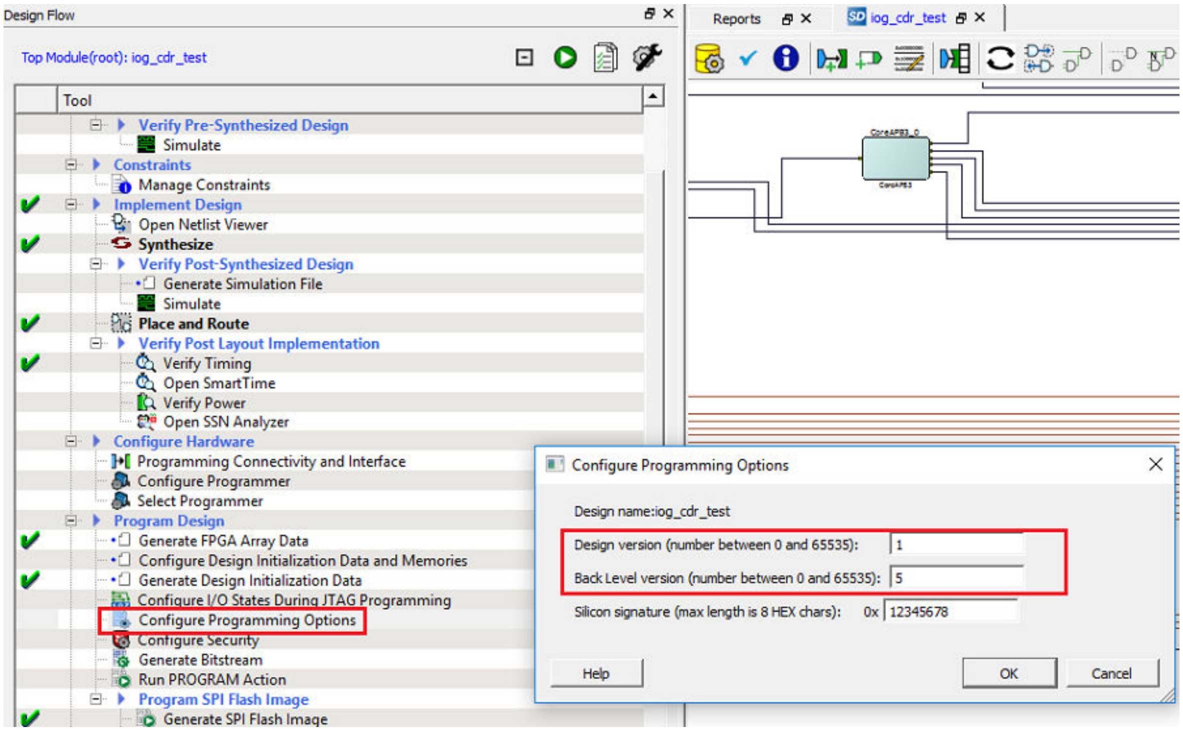
Available memory (KB): 131071
Used memory (KB): 0
Free memory (KB) : 131071

SPI Flash Clients

Program	Name	Type	Index

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

Figure 4-14. Design Version



Design name:iog_cdr_test

Design version (number between 0 and 65535): 1

Back Level version (number between 0 and 65535): 5

Silicon signature (max length is 8 HEX chars): 0x 12345678

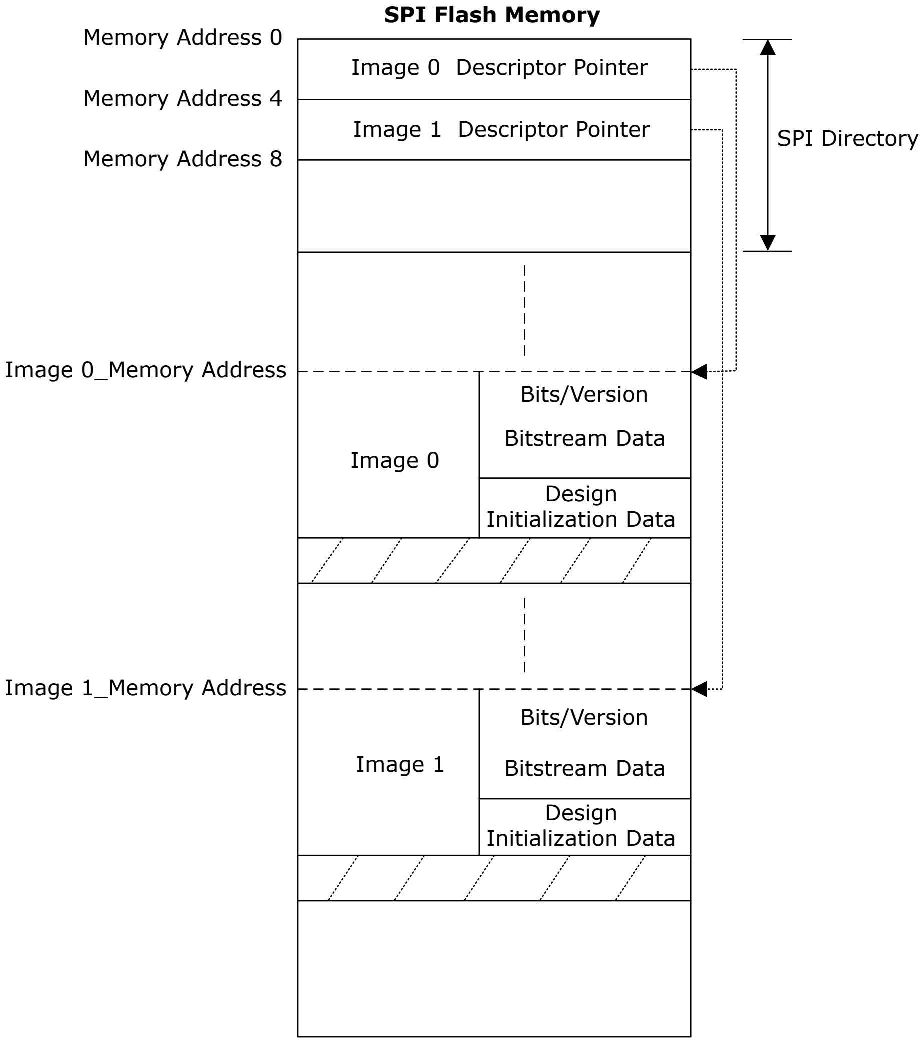
4.3.3.5 Auto Update Use Models

Auto update is initiated when a different version of the programming image is available in the SPI flash memory. For more information, see 11.2. 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.

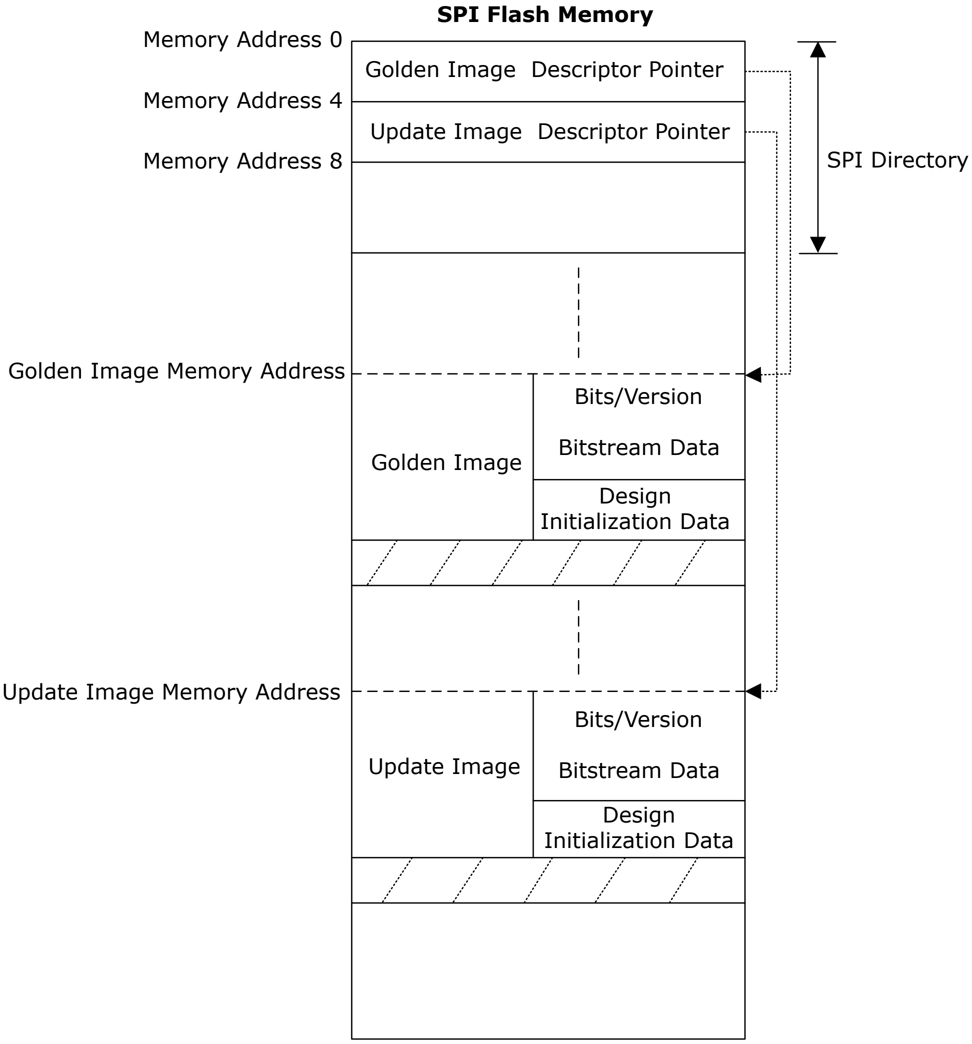
Figure 4-15. Ping Pong Use Model



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.

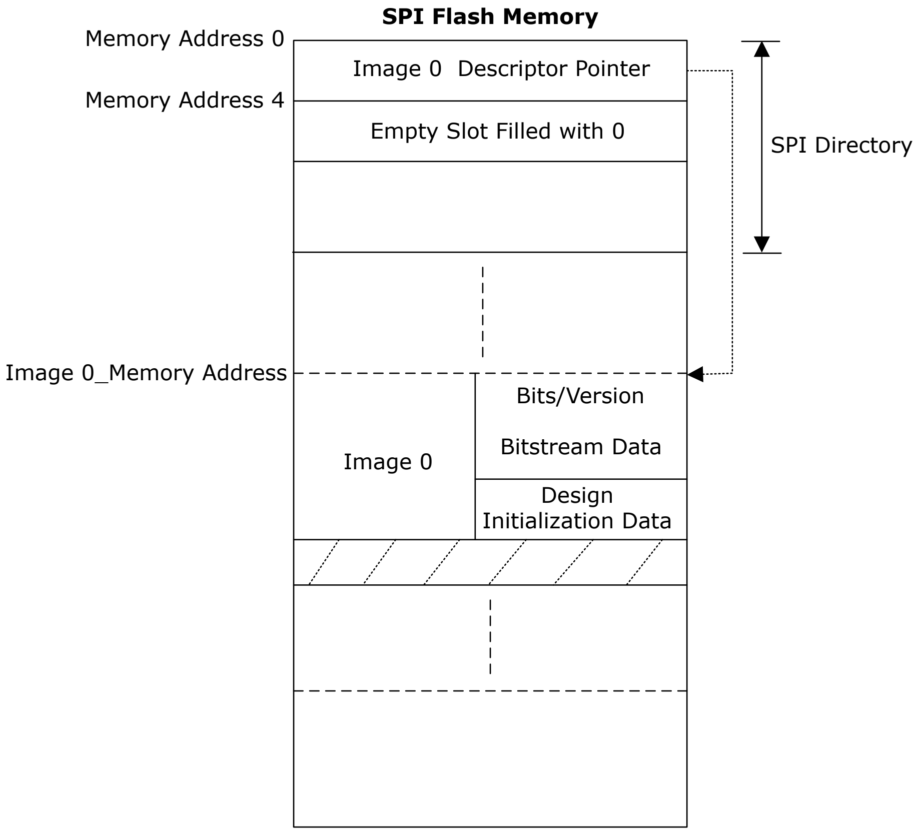
Figure 4-16. Golden Image Use Model


SPI Flash Memory
Memory Address 0
Memory Address 4
Memory Address 8
Golden Image Descriptor Pointer
Update Image Descriptor Pointer
SPI Directory
Golden Image Memory Address
Golden Image
Bits/Version
Bitstream Data
Design Initialization Data
Update Image Memory Address
Update Image
Bits/Version
Bitstream Data
Design Initialization Data

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 4-17. Single Image Use Model



4.3.4 IAP

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 specifies the programming image and initiates reprogramming of the device using the IAP system service.

4.3.4.1 IAP Using System Service

The user application initiates an IAP system service request using fabric user logic. 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 4-13. 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 4-14. 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 4-17.
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 12. 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 RT PolarFire FPGA Datasheet.

Note: Digest printed during programming (same as in `*.digest file`) is bitstream payload digest. It is meant for device to confirm that it receives the correct bitstream payload. Digest exported from DEVICE_INFO is the digest of the actual memory content. It does not have other metadata that is included in the encrypted bitstream payload, so it is different than one generated during programming.

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 4.3.2.3.2. 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.

Note: IAP recovery considers image 0 when the pointer to image 1 in the SPI directory is null. For more information, see 11.2. SPI Directory.

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

Table 4-15. 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 4-16. 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 4-17. 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.

4.3.4.2 Recovery on Programming Failure

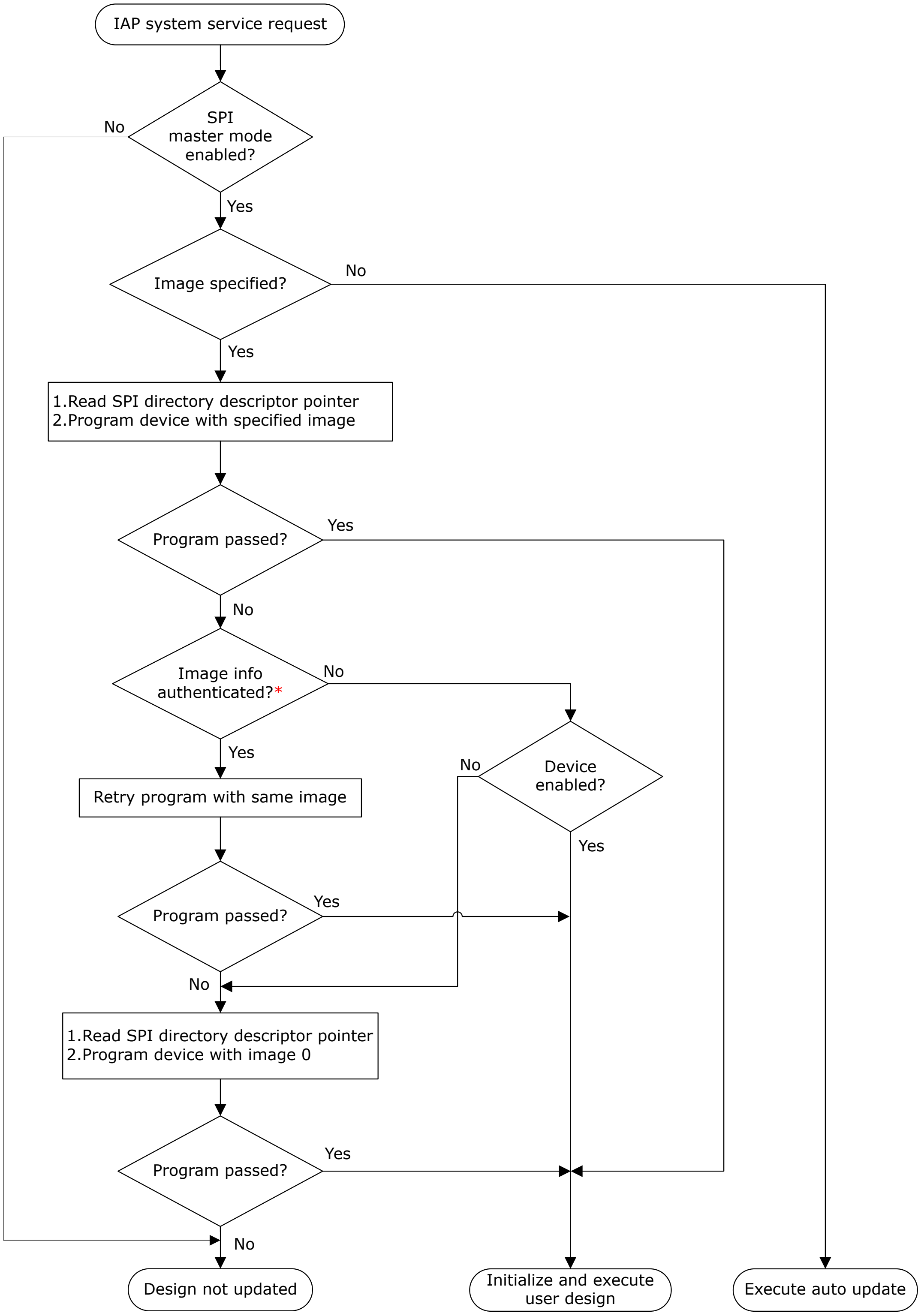
When power fails during IAP, the device programs itself with image 0.

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.

4.3.4.3 IAP Flow

The following figure shows the IAP flow.

Figure 4-18. IAP Flowchart

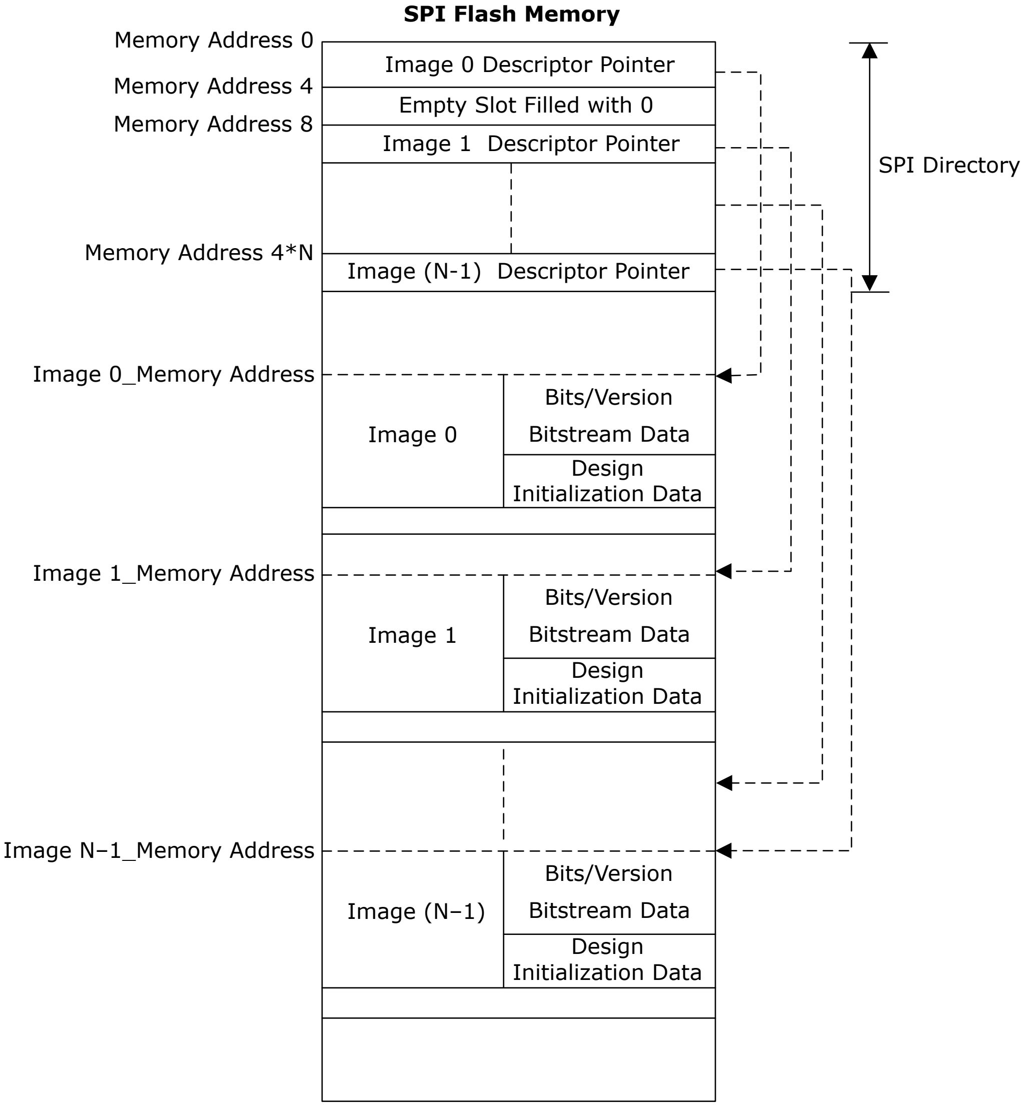

IAP system service request
SPI master mode enabled?
No
Yes
Image specified?
No
Yes
1.Read SPI directory descriptor pointer
2.Program device with specified image
Program passed?
Yes
No
Image info authenticated?*
No
Yes
Device enabled?
No
Yes
Retry program with same image
Program passed?
Yes
No
1.Read SPI directory descriptor pointer
2.Program device with image 0
Program passed?
Yes
No
Design not updated
Initialize and execute user design
Execute auto update

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

4.3.4.4 IAP Use Model

RT PolarFire FPGA 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. The following figure shows the multi-image use model.

Figure 4-19. Multi-Image Use Model



5. In-Flight Reprogramming

Reprogramming on orbit is increasingly becoming a hard requirement for space payload electronics. Satellite payload electronics' complexity have evolved so much that it is impossible to identify hardware bugs until the satellite is launched. The ability to reprogram an FPGA in space can be used to fix critical bugs. Furthermore, re-programmability in the FPGAs enables tuning of data processing algorithms to give the optimum results for new science missions. When the primary mission of the satellite is accomplished, the satellite hardware can be re-purposed to accomplish additional objectives if the FPGAs can be reprogrammed.

5.1 In-Flight Reprogramming Guidance

Microchip has performed several sets of radiation tests on RT PolarFire FPGAs and determined that the FPGAs can be programmed in space, with a greater-than-99% probability of success in space.

RT PolarFire reprogramming in space is supported using JTAG with DirectC, with the following guidance:

- Single Event Effects (SEE) Impact
  - Probability of first-time success for programming in GEO-synchronous orbit with Solar-Min conditions is calculated to be greater than 99%. If radiation disrupts programming, it is likely that the next programming attempt will be successful. Heavy ION test results can be obtained from Microchip, on request.
  - Probability of programming success in LEO is very high. No programming or verify failure was observed in accelerated ground testing. Proton test data is available from Microchip, on request.
  - In-beam reprogramming and verify is non-destructive as seen in accelerated ground testing.
  - It is unlikely that an ION will disrupt programming, since the flux in space is many orders of magnitude lower than the flux tested during accelerated ground testing.
- Total Ionizing Dose (TID) Impact
  - No in-flight programming constraints are identified after TID testing to 100 Krad.

5.2 In-Flight Reprogramming Sequence

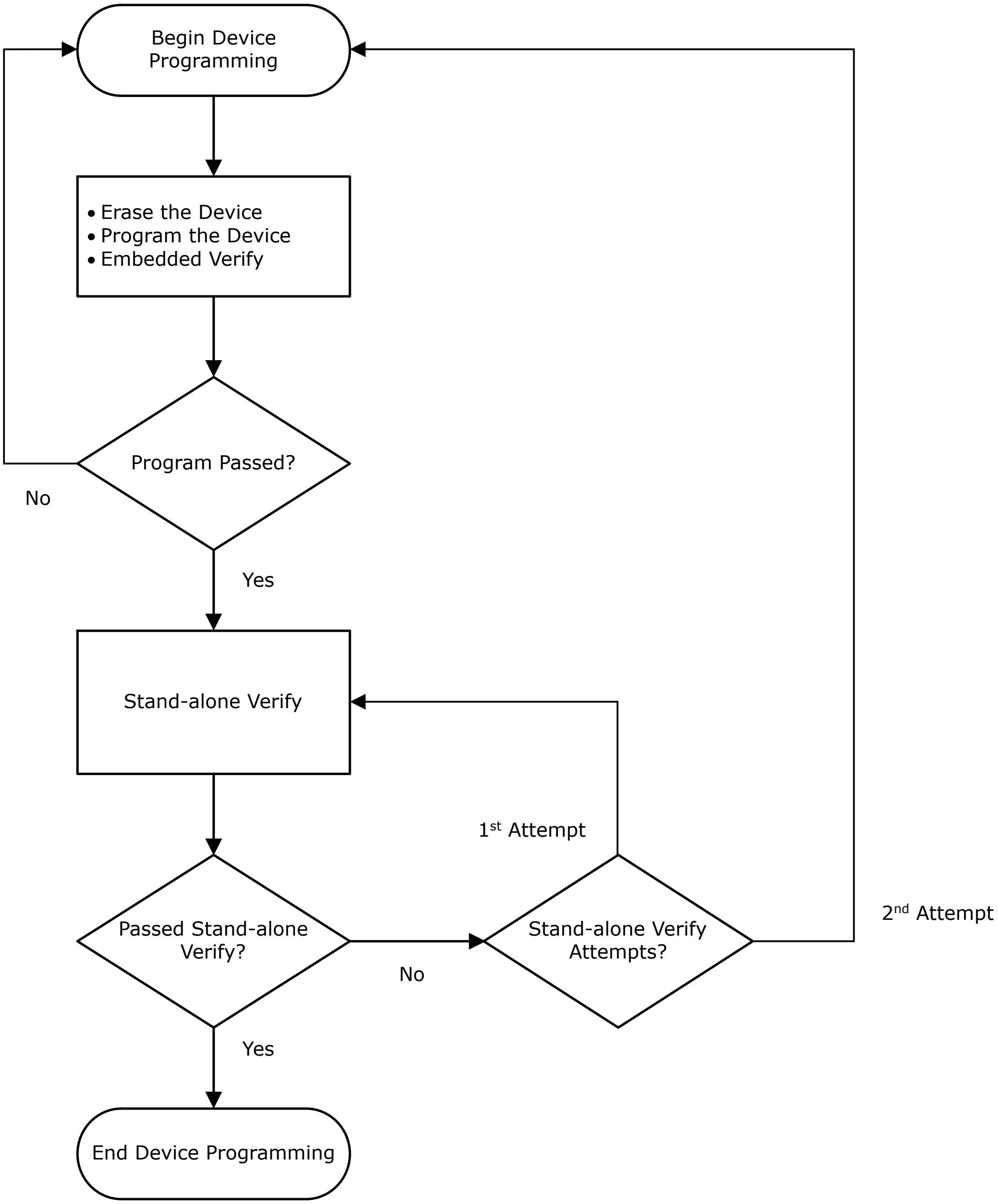
When reprogramming is initiated, the FPGA is erased. Therefore, the programming and stand-alone verify operations must be completed to success, otherwise the FPGA will be inoperable. The following sequence must be followed when performing in-flight reprogramming:

- Programming must be followed by stand-alone verify.
- If programming fails, programming must be attempted again.
- If stand-alone verify fails, the stand-alone verify action must be attempted again.
- If stand-alone verify fails second time, reprogramming of the device must be initiated again.

For information about timing requirements and interface to the RT PolarFire FPGA JTAG pins, see 4.1. JTAG Programming.

The following figure shows the sequence of in-flight reprogramming.

Figure 5-1. In-Flight Reprogramming Flow Chart



5.3 In-Flight Reprogramming Solutions

A programming controller is required to retrieve the new programming code from an external memory and to upload the new code into the target RT PolarFire FPGA, which is to be programmed. Some of the viable options for a programming controller include (but not limited to):

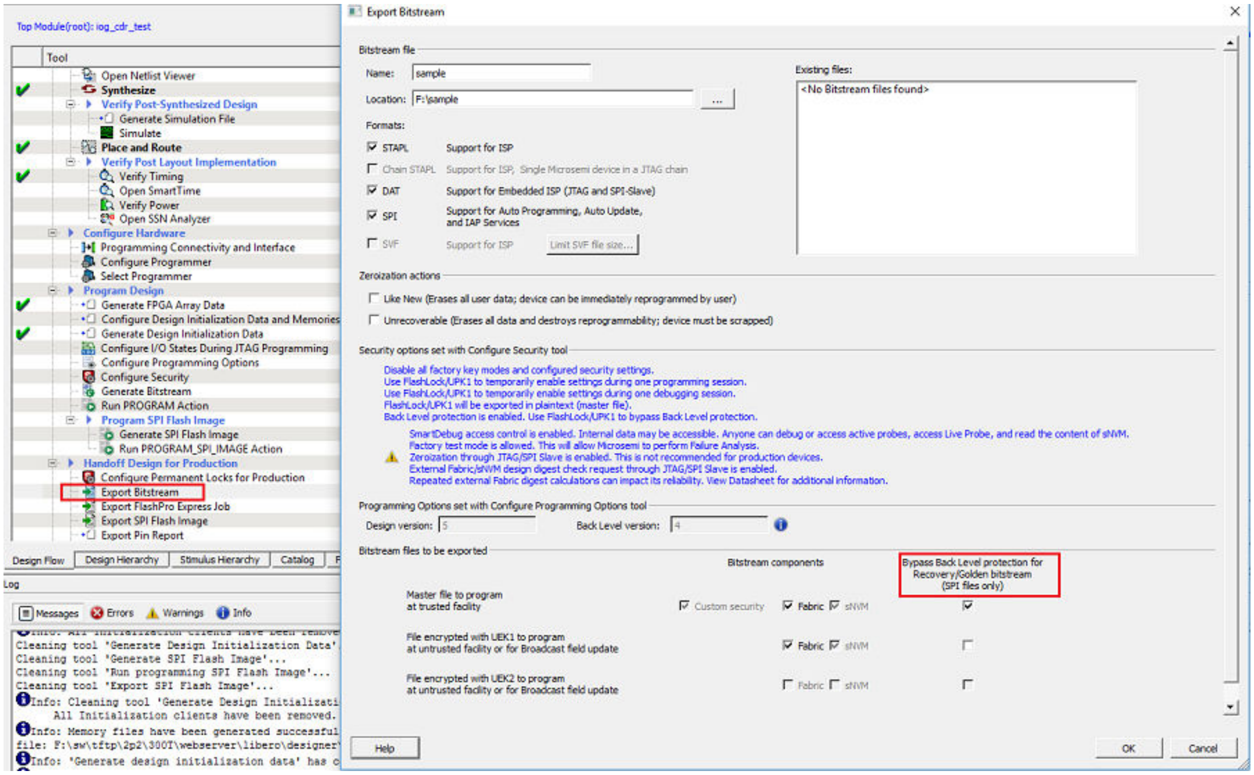
- Standalone radiation-tolerant microcontroller such as Microchip's SAMRH71F20. A reference design is available at GitHub. A demonstration video for RTG4 FPGA reprogramming is available here; a demonstration video for RT PolarFire will be provided in the later version of the document.

- Soft microprocessor IP such as RISC-V core implemented in a FPGA. A reference design using RISC-V core in RTG4 FPGA to program another RTG4 FPGA can be found in the *DirectC Installer* > *SpaceForum.zip* file. Similar principles can be applied to program an RT PolarFire FPGA by an RTG4 FPGA or by another RT PolarFire FPGA. Download the DirectC Installer. The *SpaceForum.zip* file can be found in the DirectC installation folder. For example, *\DirectC_Suite_v1.1\JTAG-DirectC_v4.1\SpaceForum.zip*. This reference design has not been updated to the latest Libero software versions.
- Other possible solutions, which include implementing a JTAG player in RTL or using a different soft processor on another FPGA. These solutions have not been tested by Microchip.

6. Bypassing the Back Level Protection

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 6-1. Selecting Bypass Back Level Protection Feature



```
Cleaning tool 'Generate Design Initialization Data'
Cleaning tool 'Generate SPI Flash Image'...
Cleaning tool 'Run programming SPI Flash Image'...
Cleaning tool 'Export SPI Flash Image'...
Info: Cleaning tool 'Generate Design Initializati
    All Initialization clients have been removed.
Info: Memory files have been generated successful
file: F:\sw\tftp\2p2\300T\webserver\libero\designer
Info: 'Generate design initialization data' has c
```

Export Bitstream

Bitstream file

Name: sample

Location: F:\sample

Existing files: <No Bitstream files found>

Formats:

- ☑ STAPL — Support for ISP
- □ Chain STAPL — Support for ISP, Single Microsemi device in a JTAG chain
- ☑ DAT — Support for Embedded ISP (JTAG and SPI-Slave)
- ☑ SPI — Support for Auto Programming, Auto Update, and IAP Services
- □ SVF — Support for ISP — Limit SVF file size...

Zeroization actions

- □ Like New (Erases all user data; device can be immediately reprogrammed by user)
- □ Unrecoverable (Erases all data and destroys reprogrammability; device must be scrapped)

Security options set with Configure Security tool

Disable all factory key modes and configured security settings.
Use FlashLock/UPK1 to temporarily enable settings during one programming session.
Use FlashLock/UPK1 to temporarily enable settings during one debugging session.
FlashLock/UPK1 will be exported in plaintext (master file).
Back Level protection is enabled. Use FlashLock/UPK1 to bypass Back Level protection.

⚠ SmartDebug access control is enabled. Internal data may be accessible. Anyone can debug or access active probes, access Live Probe, and read the content of sNVM.
Factory test mode is allowed. This will allow Microsemi to perform Failure Analysis.
Zeroization through JTAG/SPI Slave is enabled. This is not recommended for production devices.
External Fabric/sNVM design digest check request through JTAG/SPI Slave is enabled.
Repeated external Fabric digest calculations can impact its reliability. View Datasheet for additional information.

Programming Options set with Configure Programming Options tool

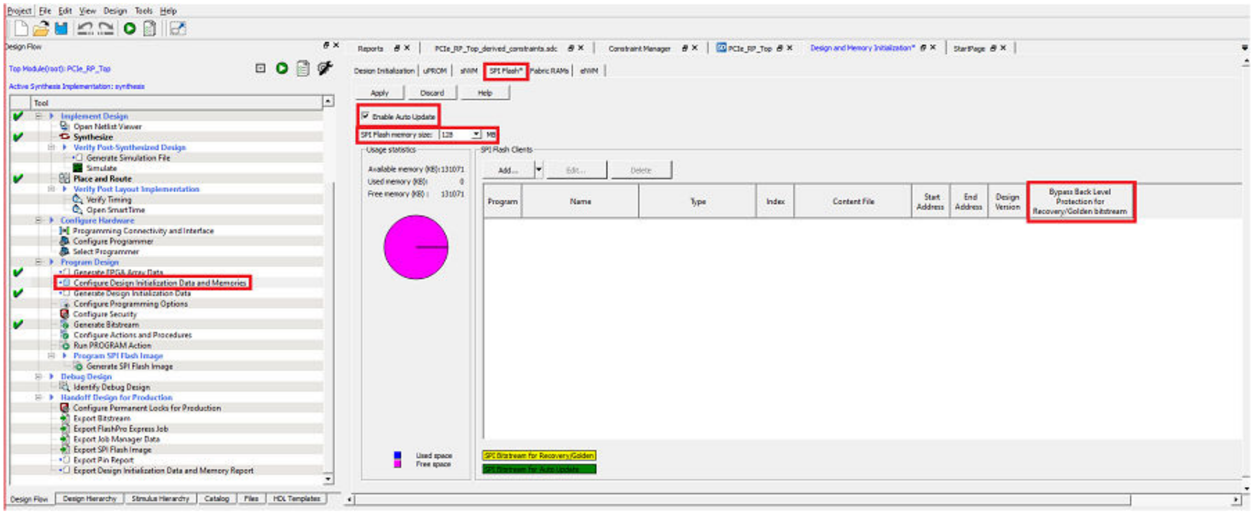
Design version: 5 Back Level version: 4

Bitstream files to be exported

		Bitstream components	Bypass Back Level protection for Recovery/Golden bitstream (SPI files only)
Master file to program at trusted facility	☑ Custom security	☑ Fabric ☑ sNVM	☑
File encrypted with UEK1 to program at untrusted facility or for Broadcast field update		☑ Fabric ☑ sNVM	□
File encrypted with UEK2 to program at untrusted facility or for Broadcast field update		□ Fabric □ sNVM	□

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 6-2. Status of Bypass Back Level Protection


Project File Edit View Design Tools Help
Design Flow
Top Module(root): PCIe_RP_Top
Active Synthesis Implementation: synthesis
Tool
Configure Design Initialization Data and Memories
Design Flow
Design Hierarchy
Stimulus Hierarchy
Catalog
Files
HDL Templates
Reports
PCIe_RP_Top_derived_constraints.sdc
Constraint Manager
PCIe_RP_Top
Design and Memory Initialization*
StartPage
Design Initialization
uPROM
sNVM
SPI Flash*
Fabric RAMs
eNVM
Apply
Discard
Help
Add...
Edit...
Delete
Used space
Free space
SPI Bitstream for Recovery/Golden
SPI Bitstream for Auto Update

☑ Enable Auto Update

SPI Flash memory size: 128 MB

Usage statistics

Available memory (KB): 131071
Used memory (KB): 0
Free memory (KB): 131071

SPI Flash Clients

Program	Name	Type	Index	Content File	Start Address	End Address	Design Version	Bypass Back Level Protection for Recovery/Golden bitstream

6.1 Bypass Back Level Protection Use Case

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

Table 6-1. 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. But the 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 6-1) for Golden/Recovery bitstream to avoid programming recovery failures because of back level protection.

7. I/O States During Programming

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

Table 7-1. I/O States for Various Programming Modes

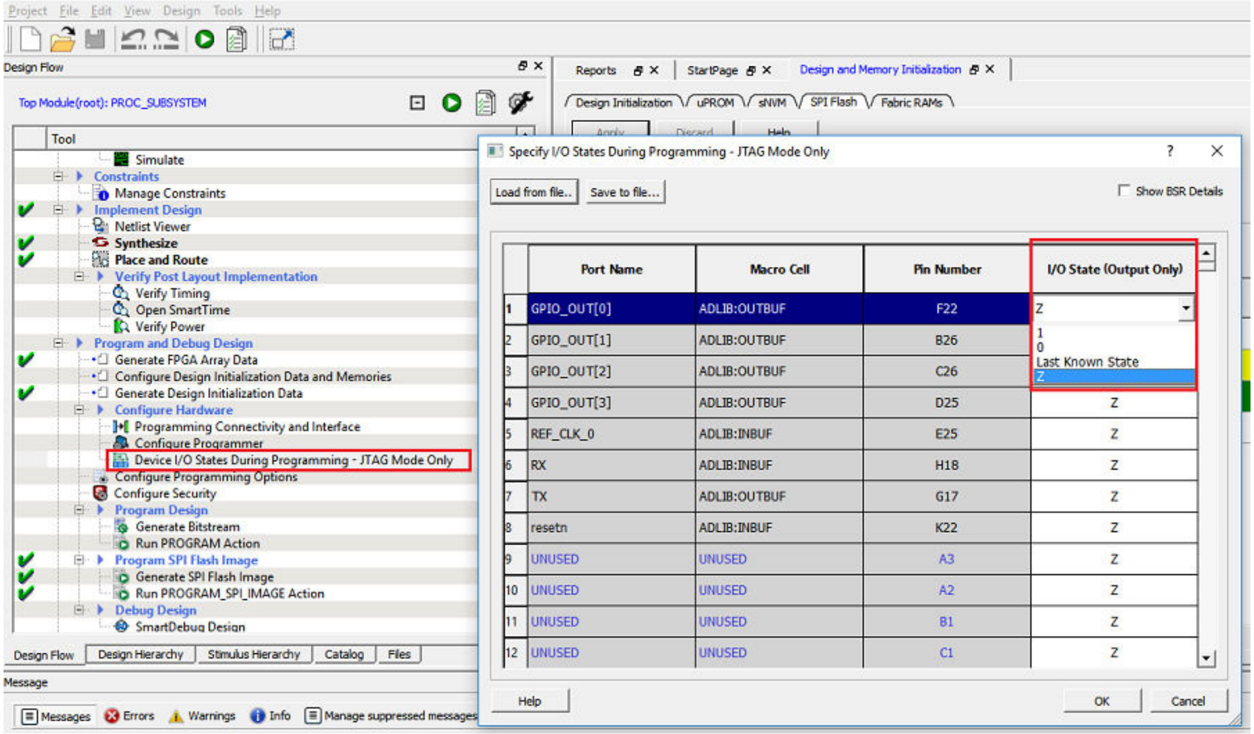
I/O Type	I/O States		
	JTAG Programming	**SPI Slave Programming**	**SPI Master Programming (IAP/Auto Update)**
System controller I/O	Enabled.	Enabled.	Enabled.
XCVR reference clock inputs	Not affected.	Not affected.	Not affected. May be kept alive during IAP using loopback mode, allowing the XCVR link 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.	Can be weakly pulled up using the SPI slave instruction ISC_ENABLE.	Outputs are tristated and weakly pulled up.

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

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

Figure 7-1. I/O States During Programming (JTAG Mode Only)



	Port Name	Macro Cell	Pin Number	I/O State (Output Only)
1	GPIO_OUT[0]	ADLIB:OUTBUF	F22	Z
2	GPIO_OUT[1]	ADLIB:OUTBUF	B26	
3	GPIO_OUT[2]	ADLIB:OUTBUF	C26	
4	GPIO_OUT[3]	ADLIB:OUTBUF	D25	Z
5	REF_CLK_0	ADLIB:INBUF	E25	Z
6	RX	ADLIB:INBUF	H18	Z
7	TX	ADLIB:OUTBUF	G17	Z
8	resetn	ADLIB:INBUF	K22	Z
9	UNUSED	UNUSED	A3	Z
10	UNUSED	UNUSED	A2	Z
11	UNUSED	UNUSED	B1	Z
12	UNUSED	UNUSED	C1	Z

8. Programming Recommendations

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 7. I/O States During Programming.

9. Brownout During Programming

Brownout is a condition that occurs when the power supplies fall below recommended levels. If brownout occurs during programming, the device automatically recovers from the programming failure (since auto recovery is enabled by default) and programs the device with a valid programming image stored in the external SPI flash.

10. Zeroization

RT PolarFire FPGAs 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.

11. Programming the External SPI Flash

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.

11.1 Supported SPI Flash Devices

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)
```

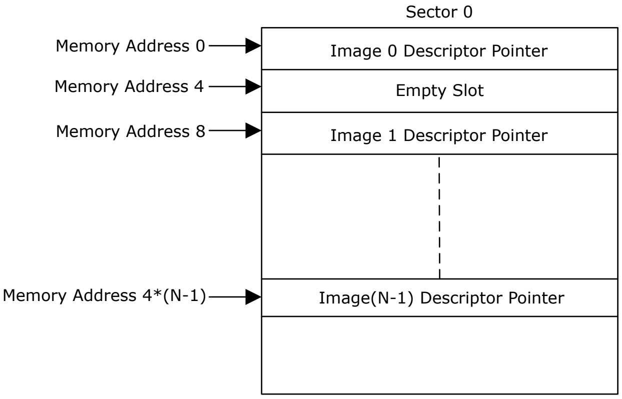
The system controller supports devices from Micron, Winbond, Macronix, and Spansion. However, any other device compatible with the JESD216 standard may also be used. Devices that are not JESD216-compliant may still be used if they support the FAST READ (0BH) command with 3-byte addressing. Such devices are limited to using only the first 128 Mb of the flash memory.

11.2 SPI Directory

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.

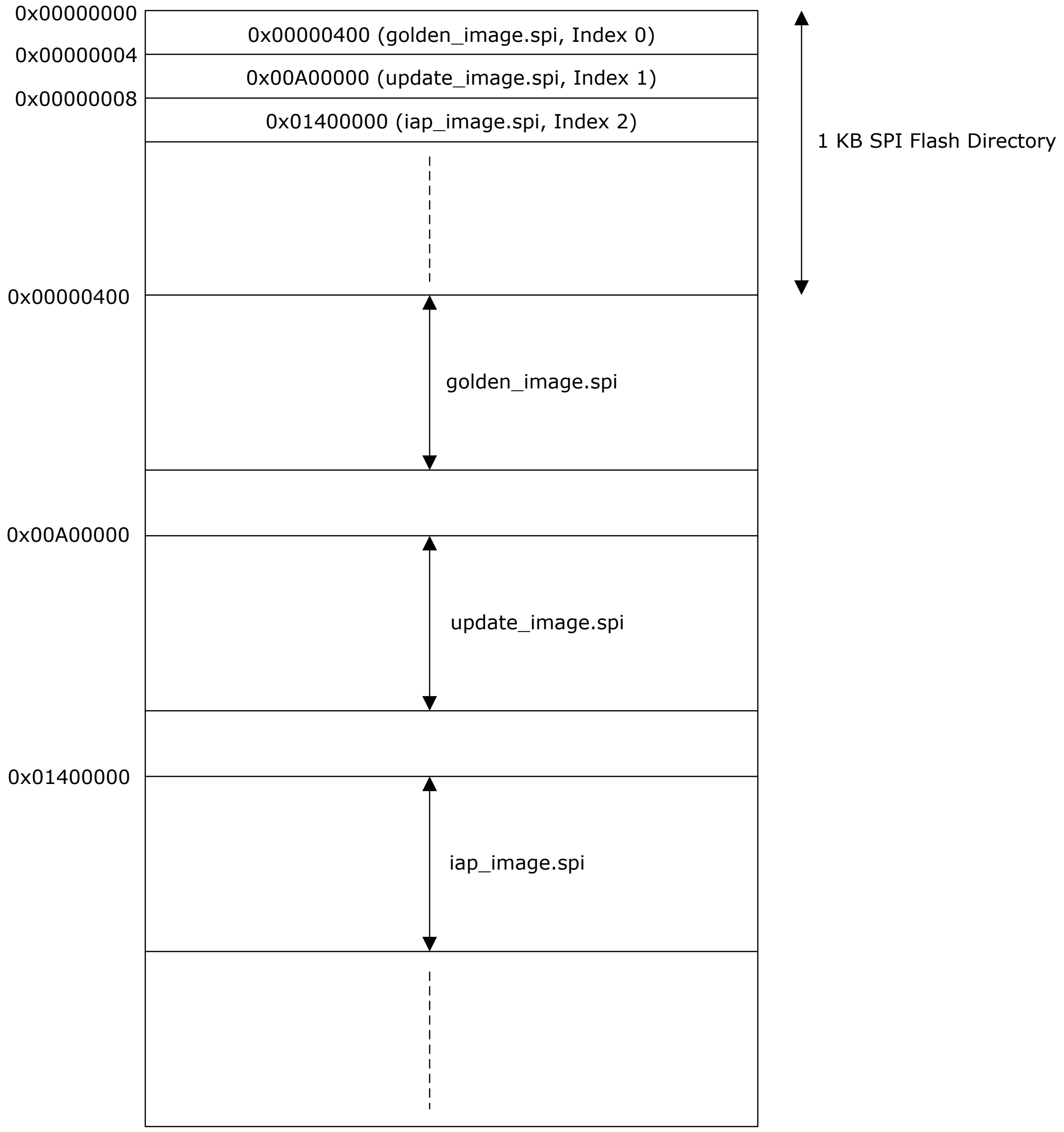
For IAP recovery to choose image 0 on power-up, the programming image pointer next to the image 0 pointer must be null (empty slot), otherwise auto update is chosen. The following figure shows the SPI flash directory with the programming image descriptor pointers.

Figure 11-1. 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 the 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 11-2. SPI Flash Memory



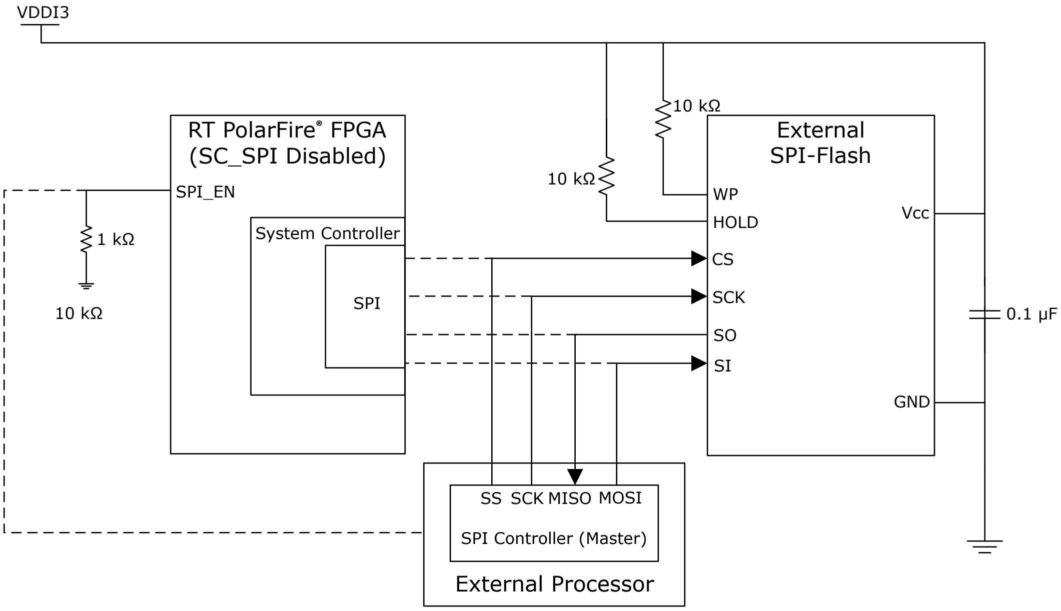
11.3 Use Models for Programming SPI Flash

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 or JTAG. This section describes the use models for programming the external SPI flash.

11.3.1 Programming the SPI Flash Using External Processor

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 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.

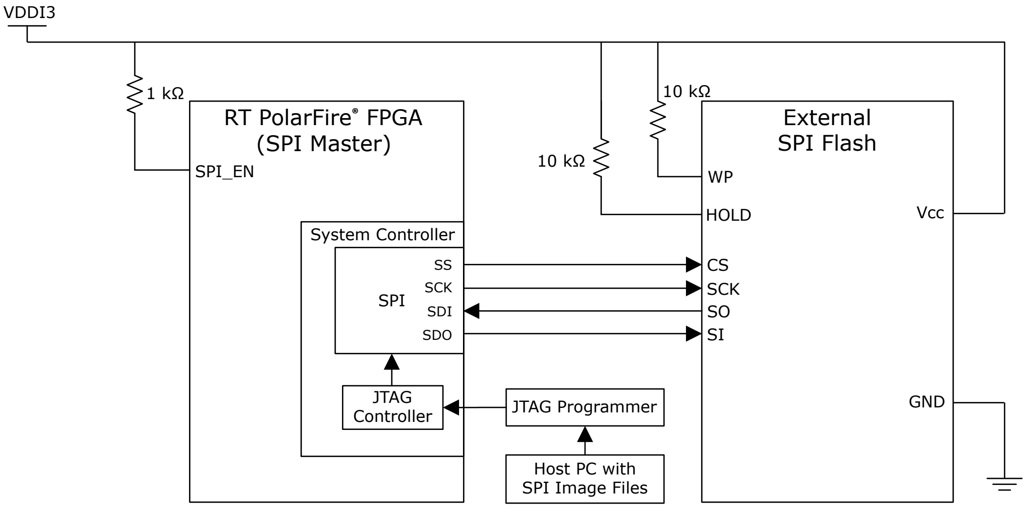
Figure 11-3. SPI Flash Programming Using External Processor



11.3.2 Programming the SPI Flash Using JTAG

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 support 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 11-4. SPI Flash Programming Using JTAG

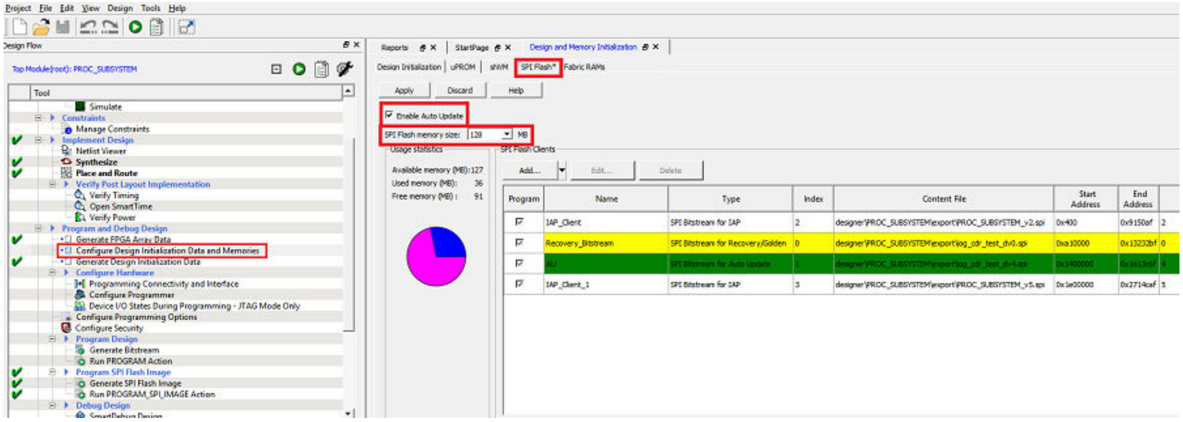


11.3.2.1 Programming External SPI Flash Using Libero

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 11-5. SPI Flash Programming in Libero SoC

   ☑ Enable Auto Update

   SPI Flash memory size: 128 MB

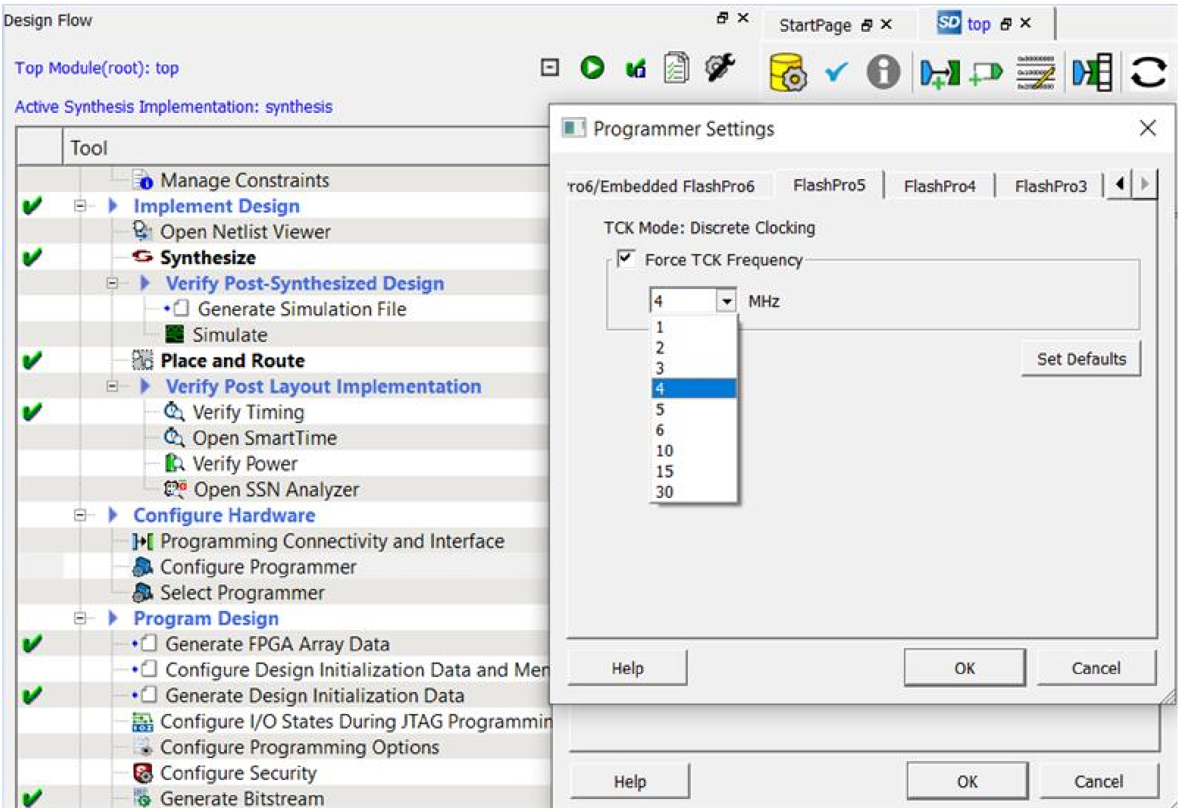
   Available memory (MB): 127
   Used memory (MB): 36
   Free memory (MB) : 91

Program	Name	Type	Index	Content File	Start Address	End Address	
☑	IAP_Client	SPI Bitstream for IAP	2	designer\PROC_SUBSYSTEM\export\PROC_SUBSYSTEM_v2.spi	0x400	0x9150af	2
☑	Recovery_Bitstream	SPI Bitstream for Recovery/Golden	0	designer\PROC_SUBSYSTEM\export\log_cdr_test_dv0.spi	0xa10000	0x13232bf	0
☑	AU	SPI Bitstream for Auto Update	1	designer\PROC_SUBSYSTEM\export\log_cdr_test_dv4.spi	0x1400000	0x1d13c6f	4
☑	IAP_Client_1	SPI Bitstream for IAP	3	designer\PROC_SUBSYSTEM\export\PROC_SUBSYSTEM_v5.spi	0x1e00000	0x2714caf	5

   Note: For RT 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. User can modify the TCK frequency by checking and selecting the Force TCK Frequency to enhance the SPI flash programming time.

   Figure 11-6. Programmer Settings

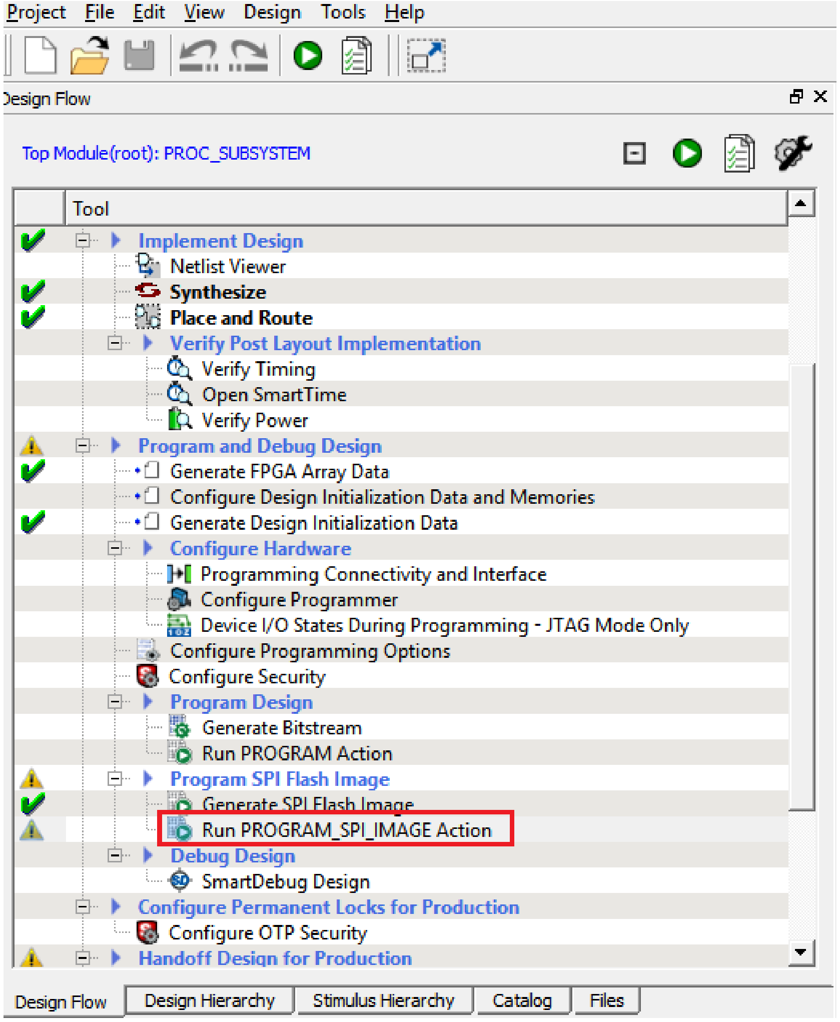
   TCK Mode: Discrete Clocking

   ☑ Force TCK Frequency

   4 MHz

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

Figure 11-7. 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, since gaps are currently programmed with 1's and increases programming time.

11.3.3 Programming the SPI Flash Using Fabric User Logic

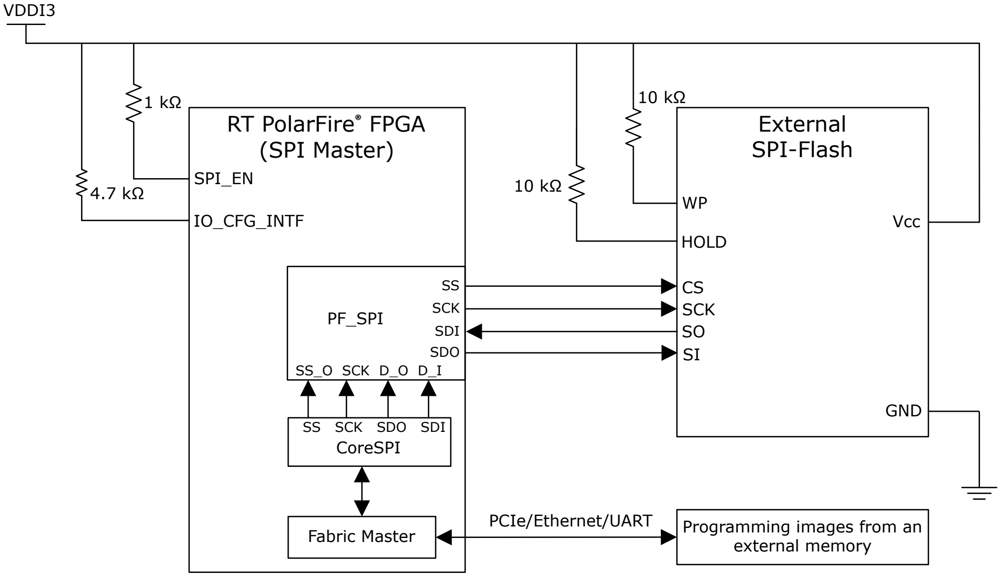
When the system controller's SPI is enabled and configured as master, the system controller hands over the control of the SPI to the fabric (after device power-up). The JTAG controller that starts programming the SPI flash, or any system service request from the fabric user logic, can take over the control 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 AC466: PolarFire FPGA Auto Update and In-Application Programming Application Note.

Figure 11-8. SPI Flash Programming Using Fabric User Logic



12. Appendix: Error Codes

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	Description	Explanation
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.

...........continued

Error Code	Description	Explanation
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.
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

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.

Table 13-1. Revision History

Revision	Date	Description
A	12/2021	The first publication of the document.

Microchip FPGA Support

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

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

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

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

The Microchip Website

Microchip provides online support via our website at www.microchip.com/. This website is used to make files and information easily available to customers. Some of the content available includes:

- **Product Support** – Data sheets and errata, application notes and sample programs, design resources, user's guides and hardware support documents, latest software releases and archived software
- **General Technical Support** – Frequently Asked Questions (FAQs), technical support requests, online discussion groups, Microchip design partner program member listing
- **Business of Microchip** – Product selector and ordering guides, latest Microchip press releases, listing of seminars and events, listings of Microchip sales offices, distributors and factory representatives

Product Change Notification Service

Microchip's product change notification service helps keep customers current on Microchip products. Subscribers will receive email notification whenever there are changes, updates, revisions or errata related to a specified product family or development tool of interest.

To register, go to www.microchip.com/pcn and follow the registration instructions.

Customer Support

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office
- Embedded Solutions Engineer (ESE)
- Technical Support

Customers should contact their distributor, representative or ESE for support. Local sales offices are also available to help customers. A listing of sales offices and locations is included in this document.

Technical support is available through the website at: www.microchip.com/support

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 product is 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.

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.

Trademarks

The Microchip name and logo, the Microchip logo, Adaptec, AnyRate, AVR, AVR logo, AVR Freaks, BesTime, BitCloud, CryptoMemory, CryptoRF, dsPIC, flexPWR, HELDO, IGLOO, JukeBlox, KeeLoq, Kleer, LANCheck, LinkMD, maXStylus, maXTouch, MediaLB, megaAVR, Microsemi, Microsemi logo, MOST, MOST logo, MPLAB, OptoLyzer, PIC, picoPower, PICSTART, PIC32 logo, PolarFire, Prochip Designer, QTouch, SAM-BA, SenGenuity, SpyNIC, SST, SST Logo, SuperFlash, Symmetricom, SyncServer, Tachyon, TimeSource, tinyAVR, UNI/O, Vectron, and XMEGA are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

AgileSwitch, APT, ClockWorks, The Embedded Control Solutions Company, EtherSynch, Flashtec, Hyper Speed Control, HyperLight Load, IntelliMOS, Libero, motorBench, mTouch, Powermite 3, Precision Edge, ProASIC, ProASIC Plus, ProASIC Plus logo, Quiet- Wire, SmartFusion, SyncWorld, Temux, TimeCesium, TimeHub, TimePictra, TimeProvider, TrueTime, WinPath, and ZL are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Adjacent Key Suppression, AKS, Analog-for-the-Digital Age, Any Capacitor, AnyIn, AnyOut, Augmented Switching, BlueSky, BodyCom, CodeGuard, CryptoAuthentication, CryptoAutomotive, CryptoCompanion, CryptoController, dsPICDEM, dsPICDEM.net, Dynamic Average Matching, DAM, ECAN, Espresso T1S, EtherGREEN, GridTime, IdealBridge, In-Circuit Serial Programming, ICSP, INICnet, Intelligent Paralleling, Inter-Chip Connectivity, JitterBlocker, Knob-on-Display, maxCrypto, maxView, memBrain, Mindi, MiWi, MPASM, MPF, MPLAB Certified logo, MPLIB, MPLINK, MultiTRAK, NetDetach, NVM Express, NVMe, Omniscient Code Generation, PICDEM, PICDEM.net, PICkit, PICtail, PowerSmart, PureSilicon, QMatrix, REAL ICE, Ripple Blocker, RTAX, RTG4, SAM-ICE, Serial Quad I/O, simpleMAP, SimpliPHY, SmartBuffer, SmartHLS, SMART-I.S., storClad, SQI, SuperSwitcher, SuperSwitcher II, Switchtec, SynchroPHY, Total Endurance, TSHARC, USBCheck, VariSense, VectorBlox, VeriPHY,

ISBN: 978-1-5224-9500-0

Quality Management System

For information regarding Microchip’s Quality Management Systems, please visit www.microchip.com/quality.

Worldwide Sales and Service

AMERICAS

Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200
Fax: 480-792-7277
Technical Support:
www.microchip.com/support
Web Address:
www.microchip.com

Atlanta
Duluth, GA
Tel: 678-957-9614
Fax: 678-957-1455

Austin, TX
Tel: 512-257-3370

Boston
Westborough, MA
Tel: 774-760-0087
Fax: 774-760-0088

Chicago
Itasca, IL
Tel: 630-285-0071
Fax: 630-285-0075

Dallas
Addison, TX
Tel: 972-818-7423
Fax: 972-818-2924

Detroit
Novi, MI
Tel: 248-848-4000

Houston, TX
Tel: 281-894-5983

Indianapolis
Noblesville, IN
Tel: 317-773-8323
Fax: 317-773-5453
Tel: 317-536-2380

Los Angeles
Mission Viejo, CA
Tel: 949-462-9523
Fax: 949-462-9608
Tel: 951-273-7800

Raleigh, NC
Tel: 919-844-7510

New York, NY
Tel: 631-435-6000

San Jose, CA
Tel: 408-735-9110
Tel: 408-436-4270

Canada - Toronto
Tel: 905-695-1980
Fax: 905-695-2078

ASIA/PACIFIC

Australia - Sydney
Tel: 61-2-9868-6733

China - Beijing
Tel: 86-10-8569-7000

China - Chengdu
Tel: 86-28-8665-5511

China - Chongqing
Tel: 86-23-8980-9588

China - Dongguan
Tel: 86-769-8702-9880

China - Guangzhou
Tel: 86-20-8755-8029

China - Hangzhou
Tel: 86-571-8792-8115

China - Hong Kong SAR
Tel: 852-2943-5100

China - Nanjing
Tel: 86-25-8473-2460

China - Qingdao
Tel: 86-532-8502-7355

China - Shanghai
Tel: 86-21-3326-8000

China - Shenyang
Tel: 86-24-2334-2829

China - Shenzhen
Tel: 86-755-8864-2200

China - Suzhou
Tel: 86-186-6233-1526

China - Wuhan
Tel: 86-27-5980-5300

China - Xian
Tel: 86-29-8833-7252

China - Xiamen
Tel: 86-592-2388138

China - Zhuhai
Tel: 86-756-3210040

ASIA/PACIFIC

India - Bangalore
Tel: 91-80-3090-4444

India - New Delhi
Tel: 91-11-4160-8631

India - Pune
Tel: 91-20-4121-0141

Japan - Osaka
Tel: 81-6-6152-7160

Japan - Tokyo
Tel: 81-3-6880- 3770

Korea - Daegu
Tel: 82-53-744-4301

Korea - Seoul
Tel: 82-2-554-7200

Malaysia - Kuala Lumpur
Tel: 60-3-7651-7906

Malaysia - Penang
Tel: 60-4-227-8870

Philippines - Manila
Tel: 63-2-634-9065

Singapore
Tel: 65-6334-8870

Taiwan - Hsin Chu
Tel: 886-3-577-8366

Taiwan - Kaohsiung
Tel: 886-7-213-7830

Taiwan - Taipei
Tel: 886-2-2508-8600

Thailand - Bangkok
Tel: 66-2-694-1351

Vietnam - Ho Chi Minh
Tel: 84-28-5448-2100

EUROPE

Austria - Wels
Tel: 43-7242-2244-39
Fax: 43-7242-2244-393

Denmark - Copenhagen
Tel: 45-4485-5910
Fax: 45-4485-2829

Finland - Espoo
Tel: 358-9-4520-820

France - Paris
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

Germany - Garching
Tel: 49-8931-9700

Germany - Haan
Tel: 49-2129-3766400

Germany - Heilbronn
Tel: 49-7131-72400

Germany - Karlsruhe
Tel: 49-721-625370

Germany - Munich
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

Germany - Rosenheim
Tel: 49-8031-354-560

Israel - Ra'anana
Tel: 972-9-744-7705

Italy - Milan
Tel: 39-0331-742611
Fax: 39-0331-466781

Italy - Padova
Tel: 39-049-7625286

Netherlands - Drunen
Tel: 31-416-690399
Fax: 31-416-690340

Norway - Trondheim
Tel: 47-72884388

Poland - Warsaw
Tel: 48-22-3325737

Romania - Bucharest
Tel: 40-21-407-87-50

Spain - Madrid
Tel: 34-91-708-08-90
Fax: 34-91-708-08-91

Sweden - Gothenberg
Tel: 46-31-704-60-40

Sweden - Stockholm
Tel: 46-8-5090-4654

UK - Wokingham
Tel: 44-118-921-5800
Fax: 44-118-921-5820