

**UG0451**  
**User Guide**  
**SmartFusion2 and IGLOO2 Programming**



a  MICROCHIP company

**Microsemi Headquarters**

One Enterprise, Aliso Viejo,  
CA 92656 USA

Within the USA: +1 (800) 713-4113

Outside the USA: +1 (949) 380-6100

Sales: +1 (949) 380-6136

Fax: +1 (949) 215-4996

Email: [sales.support@microsemi.com](mailto:sales.support@microsemi.com)

[www.microsemi.com](http://www.microsemi.com)

©2020 Microsemi, a wholly owned subsidiary of Microchip Technology Inc. All rights reserved. Microsemi and the Microsemi logo are registered trademarks of Microsemi Corporation. All other trademarks and service marks are the property of their respective owners.

Microsemi makes no warranty, representation, or guarantee regarding the information contained herein or the suitability of its products and services for any particular purpose, nor does Microsemi assume any liability whatsoever arising out of the application or use of any product or circuit. The products sold hereunder and any other products sold by Microsemi have been subject to limited testing and should not be used in conjunction with mission-critical equipment or applications. Any performance specifications are believed to be reliable but are not verified, and Buyer must conduct and complete all performance and other testing of the products, alone and together with, or installed in, any end-products. Buyer shall not rely on any data and performance specifications or parameters provided by Microsemi. It is the Buyer's responsibility to independently determine suitability of any products and to test and verify the same. The information provided by Microsemi hereunder is provided "as is, where is" and with all faults, and the entire risk associated with such information is entirely with the Buyer. Microsemi does not grant, explicitly or implicitly, to any party any patent rights, licenses, or any other IP rights, whether with regard to such information itself or anything described by such information. Information provided in this document is proprietary to Microsemi, and Microsemi reserves the right to make any changes to the information in this document or to any products and services at any time without notice.

### About Microsemi

Microsemi, a wholly owned subsidiary of Microchip Technology Inc. (Nasdaq: MCHP), offers a comprehensive portfolio of semiconductor and system solutions for aerospace & defense, communications, data center and industrial markets. Products include high-performance and radiation-hardened analog mixed-signal integrated circuits, FPGAs, SoCs and ASICs; power management products; timing and synchronization devices and precise time solutions, setting the world's standard for time; voice processing devices; RF solutions; discrete components; enterprise storage and communication solutions, security technologies and scalable anti-tamper products; Ethernet solutions; Power-over-Ethernet ICs and midspans; as well as custom design capabilities and services. Learn more at [www.microsemi.com](http://www.microsemi.com).

# Contents

---

<b>1</b>	<b>Revision History</b>	<b>1</b>
1.1	Revision 9.0	1
1.2	Revision 8.0	1
1.3	Revision 7.0	1
1.4	Revision 6.0	1
1.5	Revision 5.0	1
1.6	Revision 4.0	2
1.7	Revision 3.0	2
1.8	Revision 2.0	2
1.9	Revision 1.0	2
1.10	Revision 0	2
1.11	SmartFusion2 List of Changes Table for Reference	2
<b>2</b>	<b>Programming Overview</b>	<b>3</b>
2.1	Programming Interface	7
2.2	Programming Bitstream Generation	8
2.2.1	Programming Bitstream	9
2.3	Programming Flow	11
<b>3</b>	<b>JTAG Programming</b>	<b>12</b>
3.1	Programming Interface Overview	12
3.1.1	JTAG Timing Diagram	13
3.1.2	Design Implementation	14
3.2	Programming Using an External Programmer	15
3.2.1	Power Supply Requirements for Programming	15
3.3	Programming Using an External Microprocessor	17
<b>4</b>	<b>SPI Slave Programming</b>	<b>18</b>
4.1	Programming Interface Overview	18
4.1.1	Design Implementation	18
<b>5</b>	<b>Auto Programming</b>	<b>20</b>
5.1	Programming Interface Overview	20
5.1.1	Design Implementation	20
5.1.2	Auto Programming of M2S/M2GL050 Device	22
<b>6</b>	<b>MSS ISP (SmartFusion2 Only)</b>	<b>23</b>
6.1	Design Implementation	23
<b>7</b>	<b>In-Application Programming</b>	<b>28</b>
7.1	Design Implementation	29
7.1.1	Authenticate	29
7.1.2	Verify	29
7.1.3	Program	30
<b>8</b>	<b>Auto Update</b>	<b>35</b>
8.1	Configuring the Device for Auto Update	36
<b>9</b>	<b>Programming Recovery</b>	<b>37</b>

9.1	Programming Recovery Implementation . . . . .	37
9.2	SPI Flash Configuration and Image Selection . . . . .	39
9.2.1	MSS/HPMS SPI_0 Port Configuration . . . . .	39
9.3	Back Level Protection . . . . .	41
10	Production Programming . . . . .	43
11	State of SmartFusion2 and IGLOO2 Components During Programming . . . . .	44
11.1	Use of Flash Freeze Mechanism in Device Programming . . . . .	47

# Figures

Figure 1	Libero SoC Programming Bitstream Generation Flow	9
Figure 2	JTAG Signals Timing Diagram	13
Figure 3	JTAG Programming Mode	14
Figure 4	Programming Microsemi Devices in a JTAG Chain	15
Figure 5	JTAG Programming using External Programmer	15
Figure 6	JTAG Programming of a SmartFusion2 Device	16
Figure 7	Programming Using an External Microprocessor	17
Figure 8	SPI Slave Programming by External Microprocessor	19
Figure 9	SPI Slave Programming by External Programmer	19
Figure 10	SmartFusion2/IGLOO2 MSS/HPMS SPI_0 Port Configured for Auto Programming (Except 050 Device)	21
Figure 11	Timing Relationship of Reset and FLASH_GOLDEN_N Pin	22
Figure 12	Auto Programming Scheme for M2S/M2GL050 Devices	22
Figure 13	MSS ISP Update Process	24
Figure 14	MSS ISP Update Flow	26
Figure 15	IGLOO2 In-Application Programming Interface	28
Figure 16	SmartFusion2 In-Application Programming	30
Figure 17	IAP Flow	33
Figure 18	Auto Update Programming Ports	35
Figure 19	Enabling Auto Update	36
Figure 20	Enabling Programming Recovery	38
Figure 21	Programming Recovery Configuration for SmartFusion2	40
Figure 22	Back Level Protection	41
Figure 23	Flow Chart of Back Level Protection during Auto-Update and Programming Recovery Mode	42
Figure 24	I/O States During JTAG Programming	46
Figure 25	Setting I/O States	47

# Tables

Table 1	SmartFusion2 and IGLOO2 Programming Modes (Except M2S/M2GL050 Device) . . . . .	3
Table 2	M2S/M2GL050 Programming Modes . . . . .	6
Table 3	Available Programming Interfaces . . . . .	7
Table 4	State of Programming Interface During Reset . . . . .	8
Table 5	Programming Bitstream Size (All Variations T/S/TS) . . . . .	10
Table 6	JTAG Pin Names and Description . . . . .	12
Table 7	Dedicated SC_SPI Pins . . . . .	18
Table 8	MSS/HPMS SPI_0 Signals . . . . .	20
Table 9	ISP Programming Service Request . . . . .	24
Table 10	ISP Programming Options . . . . .	24
Table 11	ISP Programming Modes . . . . .	24
Table 12	ISP Responses . . . . .	25
Table 13	ISP Programming Service Status Codes . . . . .	25
Table 14	IAP Service Requests . . . . .	31
Table 15	IAP Programming Options . . . . .	31
Table 16	IAP Programming Modes . . . . .	31
Table 17	IAP Service Responses . . . . .	31
Table 18	IAP Programming Service Status Codes . . . . .	31
Table 19	Error Codes . . . . .	32
Table 20	IAP Bitstream Authorization Error Codes . . . . .	32
Table 21	Programming Recovery Configuration Settings (UCNFG[16:12]) . . . . .	38
Table 22	SPI Directory . . . . .	39
Table 23	User Lock Row Strobe Bits for Programming Recovery . . . . .	39
Table 24	SPI Signal Polarity Modes . . . . .	39
Table 25	Programming Recovery and Auto Update Truth Table . . . . .	40
Table 26	ASIC Block and I/O State During Programming . . . . .	44
Table 27	ASIC Block and I/O State During Programming Recovery/Auto Update . . . . .	46

# 1 Revision History

---

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

## 1.1 Revision 9.0

The following is a summary of the changes in revision 9.0 of this document.

- Information about [Back Level Protection](#), page 41 was added.
- Information about MSS Clock Frequency during IAP was updated. See [Use of Flash Freeze Mechanism in Device Programming](#), page 47.
- Information about I/O state during programming was updated. See [Table 26](#), page 44 and [Table 27](#), page 46.

## 1.2 Revision 8.0

Updated the document for Libero SoC v12.1.

## 1.3 Revision 7.0

The following is a summary of the changes in revision 7.0 of this document.

- The programming method for auto programming SPI Directory was updated in [Table 1](#), page 3.
- Throughout the document the term user security was changed to custom security.
- A reference was added for digest based verification in [Programming Bitstream Generation](#), page 8.
- A precautionary note was added in [Design Implementation](#), page 14.
- Additional references were added in [Programming Recovery Implementation](#), page 37.

## 1.4 Revision 6.0

The following is a summary of the changes in revision 6.0 of this document.

- Information about IAP flow was updated. For more information, see [Figure 17](#), page 33.
- Information about MSS ISP update flow was updated. For more information, see [Figure 14](#), page 26.
- Information about IAP error codes was updated. For more information, see [Figure 19](#), page 32.
- Information about programming recovery and auto update was updated. For more information, see [Figure 25](#), page 40.

## 1.5 Revision 5.0

The following is a summary of the changes in revision 5.0 of this document.

- Information about programming overview was updated. For more information, see [Table 1](#), page 3 and [Table 2](#), page 6.
- Information about programming flow was updated. For more information, see [Programming Flow](#), page 11.
- Information about design implementation for JTAG programming was updated. For more information, see [Design Implementation](#), page 14.
- Information about programming interface for SPI slave programming was updated. For more information, [Programming Interface Overview](#), page 12.
- Information about design implementation for auto programming was updated. For more information, see [Design Implementation](#), page 20.
- Information about auto programming of M2S/M2GL050 devices was added. For more information, see [Auto Programming of M2S/M2GL050 Device](#), page 22.
- Information about design implementation for MSS IP was updated. For more information, see [Design Implementation](#), page 23.
- Information about auto update was updated. For more information, see [Auto Update](#), page 35.
- Information about configuring the MSS/HPMS SPI\_0 port was updated. For more information, see [MSS/HPMS SPI\\_0 Port Configuration](#), page 39.

- Information about production programming was updated. For more information, see [Production Programming](#), page 43.
- Information about I/O sates was updated. For more information, see [State of SmartFusion2 and IGLOO2 Components During Programming](#), page 44.
- Changed the document to the new template.

## 1.6 Revision 4.0

Updated introduction of Auto Programming chapter (SAR 67871).

## 1.7 Revision 3.0

The following is a summary of the changes in revision 3.0 of this document.

- Merged SmartFusion2 and IGLOO2 user guide.
- Updated Table 3-1 (SAR 57679 and 63009).
- Updated Table 1-1 (SAR 59429).
- Updated Programming Bitstream Generation section (SAR 62487).

## 1.8 Revision 2.0

The M2GL005 device was added to Table 1-2. Figure 2-5, Figure 2-7, and Figure 3-1 were revised. SPI content added throughout.

## 1.9 Revision 1.0

Corrected the FlashPro4 pin number for TMS/VPUMP in Figure 2-5 (SAR 51653).

## 1.10 Revision 0

Revision 0 was the first publication of this document.

## 1.11 SmartFusion2 List of Changes Table for Reference

Date	Changed Chapters
Revision 5 (May 2014)	The M2S005 device was added to Table 1-2. Figure 2-5 and Figure 2-7 were revised. SPI content updated throughout.
Revision 4 (February 2014)	Revamped the User Guide (SAR 51186).
Revision 3 (December 2013)	Updated the JTAG Programming chapter (SAR 51652).
Revision 2 (September 2013)	Updated the JTAG Programming chapter (SAR 48287).
Revision 1 (April 2013)	Restructured the user guide.
Revision 0 (October 2012)	Initial release.



## 2 Programming Overview

The SmartFusion<sup>®</sup>2 and IGLOO<sup>®</sup>2 devices support multiple programming modes and can address various platform requirements. Specific programming modes can be implemented based on:

- End user application
- System requirements
- Package selected
- Design phase

For information about limitations on programming capabilities by package, see [Programming Interface](#), page 7.

SmartFusion2 and IGLOO2 devices support programming via an external master as well as self-programming. An external master such as a microprocessor or a programmer accesses either the system controller's dedicated JTAG or SPI port (SC\_SPI) to program the device.

Programming modes using these ports are:

- JTAG programming
- SPI-Slave programming

SmartFusion2 and IGLOO2 devices also support self-programming and those programming modes are:

- Auto programming
- Auto update
- Microcontroller Subsystem (MSS) In-System Programming (ISP) (SmartFusion2 only)
- In-Application programming

Additionally, these devices can be programmed by a standalone programmer or automatic test equipment (ATE) before mounting them on the board. This programming mode is called production programming. The following tables list the different programming modes and interfaces/ports used for these modes. In the following chapters these programming methods are discussed in detail.

**Table 1 • SmartFusion2 and IGLOO2 Programming Modes (Except M2S/M2GL050 Device)**

Description/ Programming Mode	JTAG/ SPI-Slave Programming	Auto Programming	Auto Update	MSS ISP	2 Step IAP	Production Programming
<b>Definition</b>	An external programmer or processor programs the device mounted on the board	System controller programs a blank device when the flash golden pin is asserted at device power up	System controller programs a pre-configured device with a newer image stored in SPI flash when the device is powered up	Cortex-M3 in the MSS fetches bitstream from outside the chip and calls ISP system service, which programs the FPGA	Cortex-M3 or user logic programs the bitstream into the SPI flash connected to SPI_0 port and calls IAP service call, which programs the FPGA	A standalone programmer or test equipment programs the FPGA before it is surface mount using an adapter module or a test fixture
<b>Port used</b>	JTAG or SC_SPI	MSS/HPMS SPI_0	MSS/HPMS SPI_0	Any communication peripheral	MSS/HPMS SPI_0	JTAG
<b>Bitstream location</b>	PC or on-board flash device	External SPI flash device connected to SPI_0	External SPI flash device connected to SPI_0	Remote host or on-board flash device	External SPI flash device connected to SPI_0	PC

**Table 1 • SmartFusion2 and IGLOO2 Programming Modes (Except M2S/M2GL050 Device)**

<b>Description/ Programming Mode</b>	<b>JTAG/ SPI-Slave Programming</b>	<b>Auto Programming</b>	<b>Auto Update</b>	<b>MSS ISP</b>	<b>2 Step IAP</b>	<b>Production Programming</b>
<b>Bitstream version</b>	Any bitstream (no version dependency)	Any bitstream (no version dependency)	Bitstream version has to be greater than existing bitstream version programmed in the FPGA. Otherwise, auto update does not initiate, and the device powers up with the existing bitstream.	Any bitstream (no version dependency)	Any bitstream (no version dependency)	Any bitstream (no version dependency)
<b>Bitstream address location in external SPI flash</b>	Not applicable	Non-zero address location of the external SPI flash device	SPI directory defines where the update and/or golden image is located. Both images have to be in non-zero address location, as address zero is reserved for SPI directory.	Not applicable	SPI directory defines where the update and/or golden image is located. Both images have to be in non-zero address location, as address zero is reserved for SPI directory.	Not applicable

**Table 1 • SmartFusion2 and IGLOO2 Programming Modes (Except M2S/M2GL050 Device)**

<b>Description/ Programming Mode</b>	<b>JTAG/ SPI-Slave Programming</b>	<b>Auto Programming</b>	<b>Auto Update</b>	<b>MSS ISP</b>	<b>2 Step IAP</b>	<b>Production Programming</b>
<b>SPI directory</b>	Not applicable	You must export SPI directory while exporting bitstream and program it into the external flash device in address zero location.	You must export the SPI directory while exporting bitstream and program it into the external flash device in address zero location. System controller reads the SPI directory for the image version. Every time a new image is programmed, you need to export SPI directory (by setting appropriate design version and/or address) and program the flash device before initiating auto update.	Not applicable	You must export SPI directory while exporting bitstream and program it into the external flash device in address zero location.	Not applicable
<b>Programming recovery support</b>	Not supported	Not supported since the device is blank (recovery bit has not been programmed into the device)	Supported. If you turn on the auto update switch in Libero SoC, recovery is turned on by default. If power fails, the system controller uses the golden image defined in the SPI directory.	Supported. First, the FPGA needs to be programmed with recovery setting enabled in Libero SoC.	Supported. First, the FPGA needs to be programmed with recovery setting enabled in Libero SoC.	Not supported
<b>Bitstream format</b>	JTAG Programming— Libero SoC default/STAPL/ DAT SPI-Slave-DAT	SPI	SPI	SPI	SPI	STAPLE, SVF

**Table 2 • M2S/M2GL050 Programming Modes**

<b>Description/ Programming Mode</b>	<b>JTAG/SPI-Slave Programming</b>	<b>Auto Programming</b>	<b>MSS ISP</b>	<b>2 Step IAP</b>	<b>Production Programming</b>
<b>Definition</b>	An external programmer or processor programs the device mounted on the board	System controller programs a blank device when the flash golden pin is asserted at device power up	Cortex-M3 in the MSS fetches the bitstream from outside the chip and calls ISP system service, which programs the FPGA	Cortex-M3 or user logic programs the bitstream into the SPI flash connected to SPI_0 port and calls IAP service call, which programs the FPGA	A standalone programmer or test equipment programs the FPGA before it is surface mount using an adapter module or a test fixture
<b>Port used</b>	JTAG or SC_SPI	SC_SPI	Any communication peripheral	MSS/HPMS SPI_0	JTAG
<b>Bitstream location</b>	PC or on-board flash	External SPI flash device connected to SC_SPI	Remote host or on-board flash device	External SPI flash device connected to SPI_0	PC
<b>Bitstream version</b>	Any bitstream (no version dependency)	Any bitstream (no version dependency)	Any bitstream (no version dependency)	Any bitstream (no version dependency)	Any bitstream (no version dependency)
<b>Bitstream address location in external SPI flash</b>	Not applicable	Address 0x00 location of the external SPI flash device	Not applicable	SPI directory defines where the update and/or golden image is located. Both images have to be in non-zero address location, as address zero is reserved for SPI directory.	Not applicable
<b>SPI directory</b>	Not applicable	Not needed	Not applicable	You must export SPI directory while exporting the bitstream and program it into the external flash device in the address zero location.	Not applicable
<b>Programming recovery support</b>	Not supported	Not supported	Not supported	Not supported	Not supported
<b>Bitstream format</b>	JTAG Programming— Libero SoC default/STAPL/DAT SPI-Slave -DAT	SPI	SPI	SPI	STAPLE, SVF

## 2.1 Programming Interface

SmartFusion2 and IGLOO2 devices support the programming interfaces listed in the following table. However, some interfaces are not available in all the device packages. The SC\_SPI port is used for auto programming. All other devices use the MSS/HPMS SPI\_0 port for auto programming.

All the packages mentioned in this table are available with lead and lead free. (G) indicates that the package is RoHS 6/6 compliant/Pb-free.

**Table 3 • Available Programming Interfaces**

Devices	Package	JTAG	SPI_0	Flash_GOLDEN_N	System Controller SPI Port (SC_SPI)
M2S005/M2GL005	TQ(G)144	Yes	Yes	No	No
M2S010/M2GL010		Yes	Yes	No	No
M2S005/M2GL005	VF(G)256	Yes	Yes	Yes	Yes
M2S010/M2GL010		Yes	Yes	Yes	No
M2S025/M2GL025		Yes	Yes	Yes	No
M2S025/M2GL025	FCS(G)325	Yes	Yes	No	No
M2S050/M2GL050		Yes	Yes	No	No
M2S060/M2GL060		Yes	Yes	No	No
M2S090/M2GL090		Yes	Yes	No	No
M2S005/M2GL005	VF(G)400	Yes	Yes	Yes	Yes
M2S010/M2GL010		Yes	Yes	Yes	Yes
M2S025/M2GL025		Yes	Yes	Yes	Yes
M2S050/M2GL050		Yes	Yes	Yes	Yes
M2S060/M2GL060		Yes	Yes	Yes	Yes
M2S150/M2GL150	FCV(G)484	Yes	Yes	Yes	Yes
M2S005/M2GL005	FG(G)484	Yes	Yes	Yes	Yes
M2S010/M2GL010		Yes	Yes	Yes	Yes
M2S025/M2GL025		Yes	Yes	Yes	Yes
M2S050/M2GL050		Yes	Yes	Yes	Yes
M2S060/M2GL060		Yes	Yes	Yes	Yes
M2S090/M2GL090		Yes	Yes	Yes	Yes
M2S150/M2GL150	FCS(G)536	Yes	Yes	Yes	Yes
M2S060/M2GL060	FGG676	Yes	Yes	Yes	Yes
M2S090/M2GL090		Yes	Yes	Yes	Yes
M2S050/M2GL050	FG(G)896	Yes	No	Yes	Yes
M2S150/M2GL150	FC(G)1152	Yes	Yes	Yes	Yes

For board level design, it is important to know the state of these interfaces during reset. The following table summarizes the state of these interfaces during different reset.

**Table 4 • State of Programming Interface During Reset**

Programming Interface	During Power on Reset	During DEVRST_N <sup>1</sup>	During System Control Reset	During MSS Reset
JTAG port	JTAG I/Os are enabled	JTAG I/Os are enabled	JTAG I/Os enabled, but not functional as JTAG operation within the system controller relies on TRSTB being negated but held asserted for system controller's suspend mode	Unaffected
System control SPI port (SC_SPI)	Disabled (floating, no pull ups)	Disabled (floating, no pull ups)	Active but not functional	Unaffected
MSS/HPMS SPI_0 port	Disabled (floating, no pull ups)	Disabled (floating, no pull ups)	Depends on user configuration	Depends on user configuration

1. DEVRST\_N is an asynchronous reset pin and must be asserted only when the device is unresponsive because of some unforeseen circumstances. Do not assert the DEVRST\_N pin during programming operation, which might cause severe consequences including corrupting the device configuration.

## 2.2 Programming Bitstream Generation

The Libero SoC software is used to generate the programming bitstream formats required to support the different programming modes. Table 5, page 10 shows the different bitstream file types and sizes for SmartFusion2 and IGLOO2 devices.

Programming is integrated into the Libero SoC software design flow. With the enhanced Libero SoC design flow, you can program the device directly, without launching the programming software tool (FlashPro Express) separately. When the design is ready for production, the FlashPro Express Job file (\*.job) can be exported using the Libero SoC software.

FlashPro Express tool uses \*.job file for programming the FPGAs. See [FlashPro Express user guide](#) for information about programming the FPGAs.

The programming bitstream contains the following components, which can be selected during bitstream generation:

- Images of FPGA fabric only
- Full or partial image of eNVM only
- Custom Security

Custom security can be implemented in the bitstream from Configure Security under Program and Debug Design Options menu of the Design flow in the Libero SoC software.

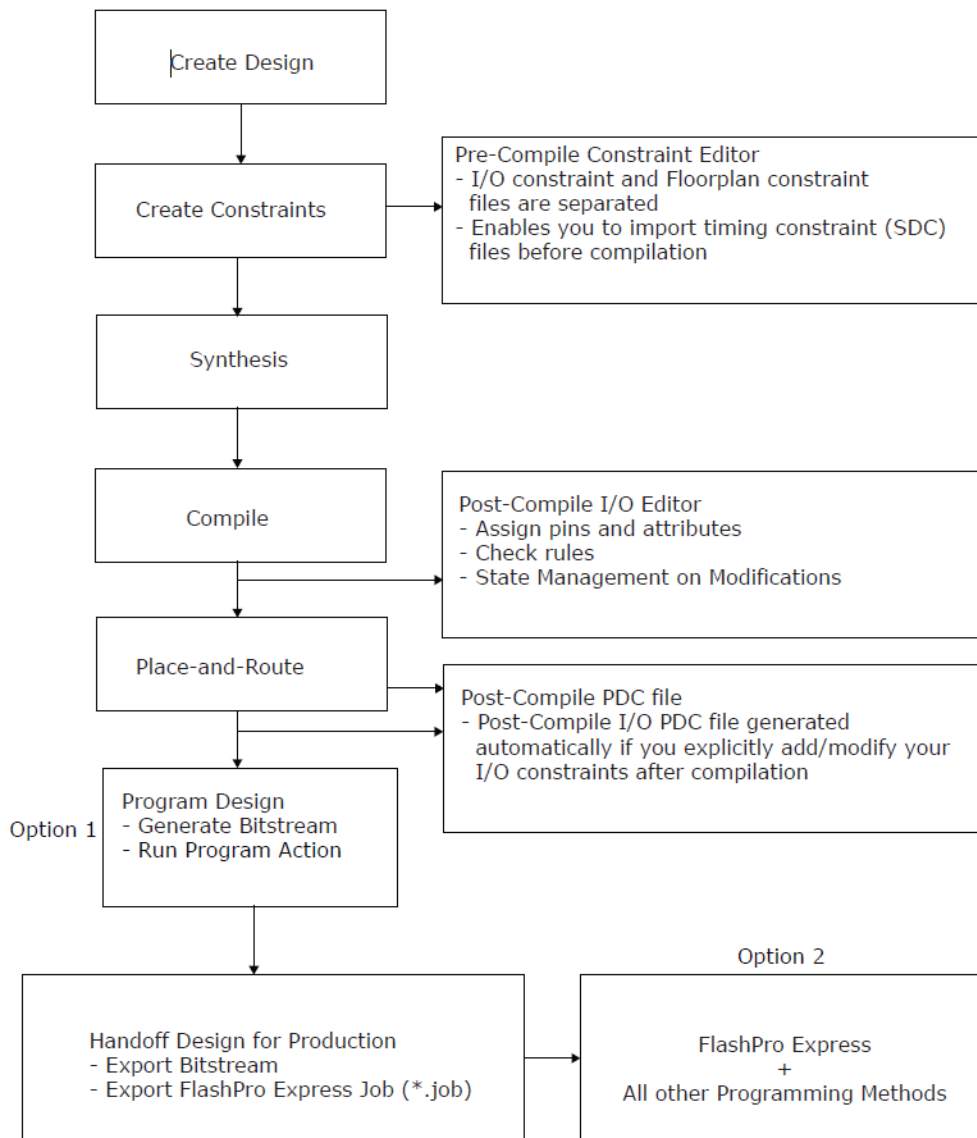
When the bitstream is exported the digest of the selected components is printed in the Libero log window and saved in a file under the export folder. This digest can be used to verify whether intended bitstream was used during programming or not. For more information, see the "Digest-Based Verification Method" section of the [SmartFusion2 and IGLOO2 FPGA Security Best Practices User Guide](#).

The following figure shows the Libero SoC programming bitstream generation flow. There are two programming options:

- Option 1: The programming bitstream file is generated by clicking **Generate Programming Data**. SmartFusion2 and IGLOO2 devices can be programmed from within the tool, provided the target board is connected with the FlashPro4/FlashPro5 programmer. Option 1 is used for JTAG programming mode.

- Option 2: After implementation of the design, the appropriate programming bitstream can be exported and later used in any of the supported programming modes.

**Figure 1 • Libero SoC Programming Bitstream Generation Flow**



## 2.2.1 Programming Bitstream

The programming bitstream contains programming data for FPGA fabric and/or eNVM (partial or full content). Programming bitstream is always encrypted with the default key.

For additional security, custom security keys can be set in the security policy manager (SPM). For more information, see the SPM section in the *Libero SoC User Guide*. If custom security is used default key will be disabled.

The following table lists the bitstream size. Size does not change with device utilization. For information about devices offering with and without transceivers and data security (T, TS, S), see the *SmartFusion2 Product Brief and IGLOO2 Product Brief*.

**Table 5 • Programming Bitstream Size (All Variations T/S/TS)**

Device	File Type	Full Fabric (in KB)	Full eNVM (in KB)	Full Fabric + Full eNVM (in KB)	Full Fabric + Full eNVM + Custom Security (in KB) <sup>1</sup>
M2S005/M2GL005	STAPL	522	270	731	1409
	SPI <sup>2</sup>	296	135	429	429
	DAT	297	135	431	860
M2S010/M2GL010	STAPL	933	482	1354	2655
	SPI	556	269	823	823
	DAT	557	269	825	1648
M2S025/M2GL025	STAPL	1945	482	2367	4679
	SPI	1195	269	1463	1463
	DAT	1197	269	1464	2926
M2S050/M2GL050	STAPL	3792	482	4213	8373
	SPI	2363	269	2630	2630
	DAT	2364	269	2632	5261
M2S060/M2GL060	STAPL	3799	480	4211	8360
	SPI	2363	263	2624	2624
	DAT	2364	263	2625	5248
M2S090/M2GL090	STAPL	5690	900	6526	12996
	SPI	3561	532	4092	4092
	DAT	3564	533	4095	8186
M2S150/M2GL150	STAPL	9539	900	10376	20696
	SPI	5996	532	6527	6527
	DAT	5997	533	6528	13054

1. File size with custom security is larger because of the encryption added to the file.

2. SPI file size with and without custom security option is the same, because the security component in it is small.



## 2.3 Programming Flow

The following steps summarize the SmartFusion2 and IGLOO2 programming flow:

1. Checks if the device ID matches with that of the file loaded. If yes, the device enters programming mode.

**Note:** I/O settings are driven by the BSR during programming.

2. Checks if the security settings of the device matches with that of the programming bitstream. If yes, the bitstream is authenticated.
3. The following operations occur while the SmartFusion2 and IGLOO2 programming bitstream is processed:
  - a. Erases all the features selected (fabric, fabric configuration, and security) to be programmed. eNVM is not erased; it is reprogrammed only when eNVM programming is selected.
  - b. If the custom security features are selected to be programmed, they are programmed before programming the fabric and eNVM.
  - c. If the FPGA fabric is selected for programming in the bitstream, the fabric is programmed while processing the bitstream. Verification is built into the programming. If programming is interrupted or fails in the middle of bitstream, the device is not enabled. The fabric is enabled only after the entire bitstream is successfully programmed.

**Note:** The programming action in STAPL provides an optional DO\_VERIFY step. This is a separate verification process that takes additional programming time. This is disabled by default.

- d. If eNVM programming is selected, the eNVM blocks are programmed (X = 0, 1, depending on the die size). Verification is built into the eNVM programming.

- e. Exits programming mode. I/O control is transferred from BSR to the fabric design.

**Note:** Upon completion of the programming, the system controller resets the entire device to implement the new design. If the Cortex-M3 processor (SmartFusion2 only) is used to update the eNVM, the rest of the device continues to operate.

## 3 JTAG Programming

SmartFusion2 and IGLOO2 devices support programming using a dedicated JTAG port. The system controller implements the functionality of a JTAG slave and complies to IEEE 1532 and IEEE 1149.1 standards. The JTAG port communicates with the system controller using:

- A command register that sends the JTAG instruction to be executed.
- A 128-bit data buffer that transfers any associated data.

To start programming using JTAG, the device must not be in Flash\*Freeze (F\*F) mode. When a device is not in a programmed state, all user I/O pins are disabled, that is, tristated. This is achieved by keeping the global IO\_EN signal internal signal deactivated, which disables the input buffers. I/O states during JTAG programming can be configured in Libero SoC. For more information, see [State of SmartFusion2 and IGLOO2 Components During Programming](#), page 44.

### 3.1 Programming Interface Overview

SmartFusion2 and IGLOO2 devices can be programmed through the dedicated JTAG interface. An external programmer (such as FlashPro4/5) or a microprocessor is used to program the device. The devices can be programmed in both single and chain modes. SmartFusion2 and IGLOO2 devices have JTAG pins in a dedicated bank. The location of the bank varies depending on the package. For more information about the bank and its location, see [SmartFusion2 Pin Descriptions](#).

JTAG signals can be operated at 1.2 V, 1.5 V, 1.8 V, 2.5 V, or 3.3 V. Therefore, the JTAG bank voltage can be set to any of these voltages. The logic level of the JTAG signals depends on the JTAG bank voltage. The following table lists the JTAG pin names, descriptions, and their termination details.

**Note:** If the JTAG programming mode is not used, JTAG pins must be terminated.

**Table 6 • JTAG Pin Names and Description**

Name	Direction	Weak Pull Up	Termination	Description
JTAGSEL	Input	Yes	Terminate with 1 K pull up to JTAG power supply Tie to GND through 1 K	JTAG controller selection. If JTAGSEL is pulled high, an external TAP controller connects the JTAG interface to the system controller TAP. This is the recommended setup for programming and microcontroller debug through SoftConsole. If JTAGSEL is pulled low, an external TAP controller connects to either the Cortex-M3 JTAG TAP (if debug is enabled) or an auxiliary TAP (if debug is disabled). This is the recommended setup for microcontroller debugging using tools such as IAR or KEIL (SmartFusion2 only). For IGLOO2 based designs, this signal must be held high through the 1 K pull-up resistor.
JTAG_TCK/ M3_TCK <sup>1</sup>	Input	No	Microsemi recommends TCK to be tied to VSS or VDDI through a resistor on the board when unused. This is according to IEEE 1532 requirements.	Clock input to JTAG controller and UJTAG <sup>2</sup> . If JTAG is not used, Microsemi recommends tying it off. The tied off resistor must be placed close to the FPGA pin. This prevents creation of totem-pole current due to random oscillations on the input buffer and operation if TMS enters an undesired state.

**Table 6 • JTAG Pin Names and Description (continued)**

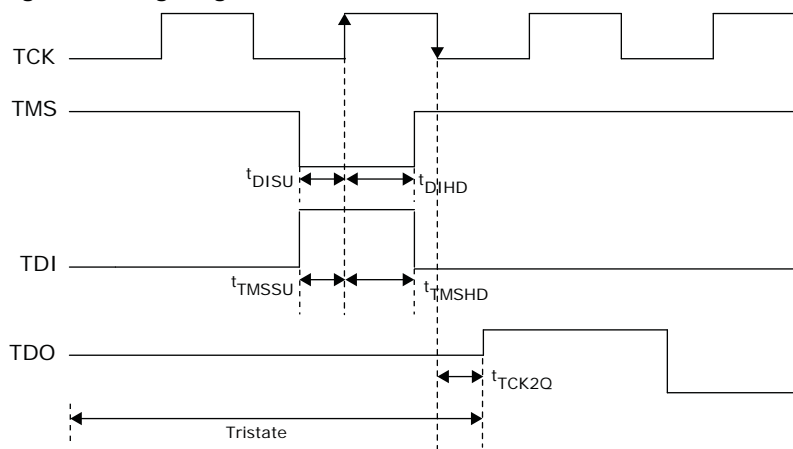
Name	Direction	Weak Pull Up	Termination	Description
JTAG_TDI/ M3_TDI <sup>1</sup>	Input	Yes	Not required	Serial data input to JTAG boundary scan, UJTAG, and ISP. UJTAG uses this pin or this pin aids another component to use JTAG, ISP, and UJTAG.
JTAG_TDO/ M3_TDO <sup>1</sup>	Output	No	Not required	Serial data output from JTAG boundary scan, UJTAG, and ISP. UJTAG uses this pin or this pin aids another component to use JTAG, ISP, and UJTAG.
JTAG_TMS/ M3_TMS <sup>1</sup>	Input	Yes	Not required	The TMS pin controls the use of the following IEEE1532 boundary scan pins: TCK, TDI, TDO, and TRSTB.
JTAG_TRSTB/ M3_TRSTB <sup>1</sup>	Input	Yes	Tie to GND through 1 K	Reset pin for JTAG controller. The TRSTB pin functions as an active low input to asynchronously initialize or reset the boundary scan circuitry. If JTAG is not used, an external pull-down resistor can be included to ensure that the TAP is held in reset mode. In critical applications, an upset in the JTAG circuit could lead to an undesired JTAG state. In such cases, Microsemi recommends tying off TRSTB to GND through a resistor placed close to the FPGA pin.

1. Available only in SmartFusion2.

2. The UJTAG macro is a special purpose macro. It provides access to the user JTAG circuitry on-board the chip. You must instantiate a UJTAG macro in the design to make use of the user JTAG feature. The TMS, TDI, TCK, TRSTB, and TDO pins of the macro must be connected to the top-level ports of the design.

### 3.1.1 JTAG Timing Diagram

The following figure shows the JTAG signals timing diagram. The JTAG 1532 Timing Characteristics table of the *SmartFusion2 and IGLOO2 Datasheet* specifies the timing numbers to be met to ensure proper operation of the JTAG circuitry.

**Figure 2 • JTAG Signals Timing Diagram**

### 3.1.2 Design Implementation

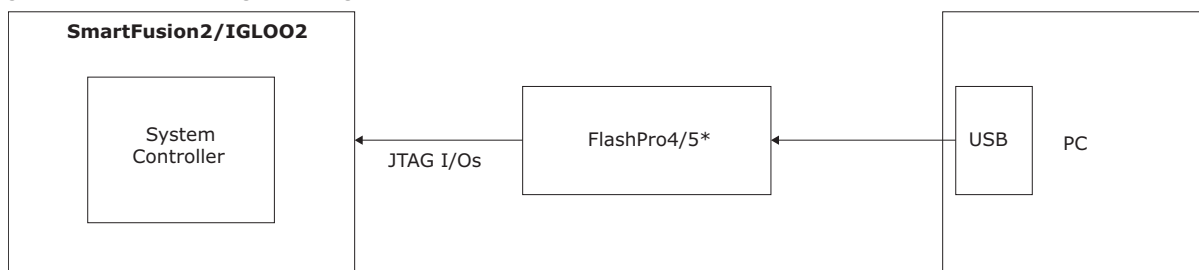
FPGA fabric, embedded non volatile flash memory (eNVM), and custom security settings can be programmed using JTAG programming mode. These can be programmed individually, or at the same time.

**Note:** During JTAG or SPI-Slave programming, do not run any of the AES/DRBG/ECC Point Multiplication system services when System controller is processing the initial component of the bitstream (BITS). The system service may corrupt the bitstream information if the user design requests AES/DRBG/ECC Point Multiplication system services. This issue does not exist in Auto-update, IAP or Programming recovery. If these security system services must be run during programming, you must generate a STAPL/DAT file using Libero 11.8 SP3 or contact [Technical Support](#) to use older Libero versions.

- During programming, the board must provide power to the following pins: VPP, VPPNVM, VDD, and VDDIOx (x: number of the JTAG I/Os). VDDIOx provides power to the JTAG circuitry. For information about the voltage range, see the [SmartFusion2 and IGLOO2 Datasheet](#).
- A USB-based FlashPro4/5 programmer can be used to program the SmartFusion2 and IGLOO2 devices using this dedicated JTAG interface. Libero SoC (or standalone FlashPro Express) software executes the programming from a PC connected to the programmer.

The following figure shows the FlashPro4/5 programmer connected to the JTAG ports of the SmartFusion2 device. Only this programming mode is supported by the current version of the Libero SoC software.

**Figure 3 • JTAG Programming Mode**

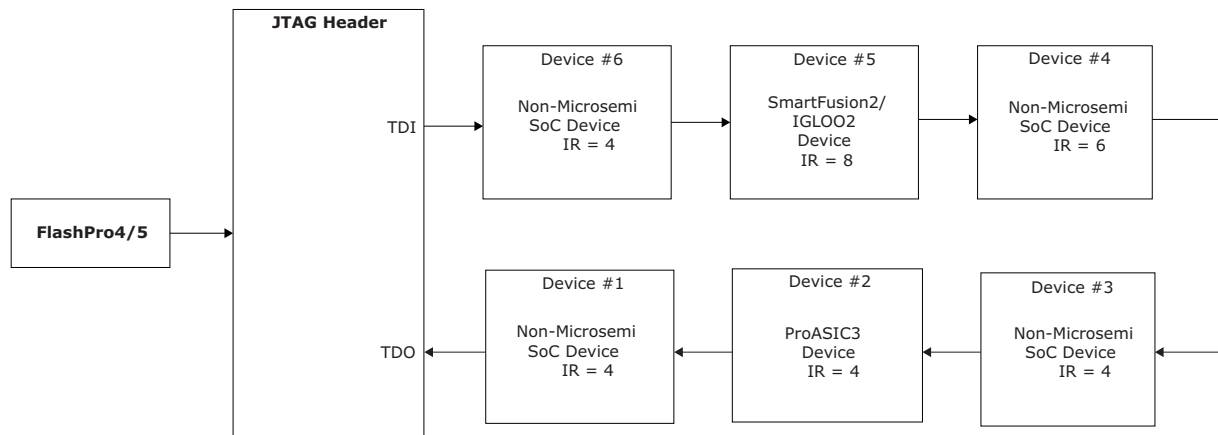


\*The FlashPro3 programmer can be used in JTAG programming mode but it has been discontinued.

A single FlashPro4/5 programmer can program multiple FPGAs (same or different family) with multiple non-Microsemi devices in a single JTAG chain, concurrently, as shown in the following figure.

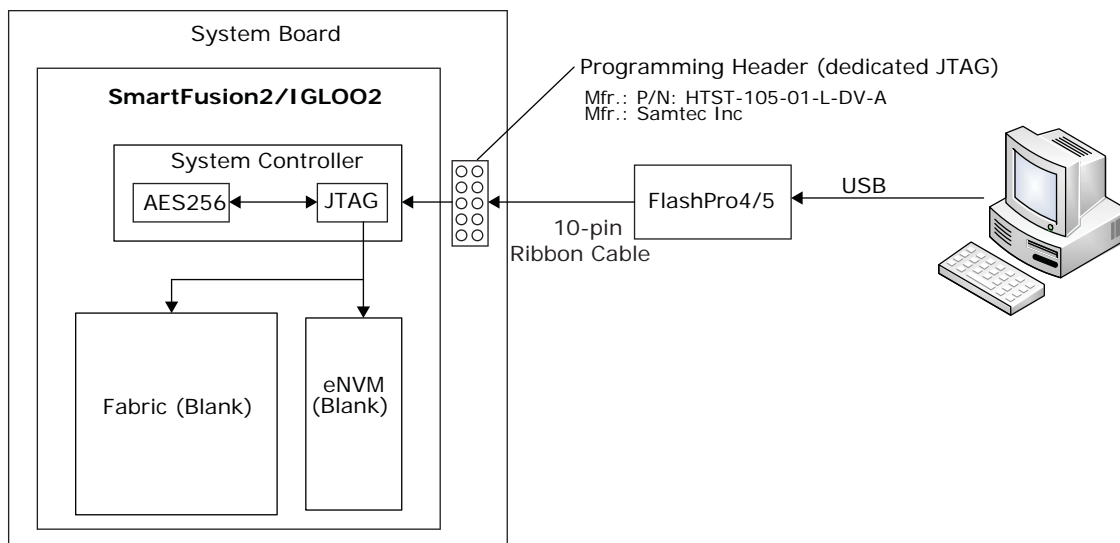
The TDO pin of the JTAG header represents the beginning of the chain and the TDI pin of the last device is connected back to the JTAG header. While programming any Microsemi device in the chain, the Libero SoC (or standalone FlashPro Express) software puts non-Microsemi devices in the chain into bypass mode. Once a device is in bypass mode, its data register length becomes one and does not react to any programming instructions given by the programmer. To put the device into bypass mode, the programmer must know its instruction register (IR) length.

For information about chain programming, see the [FlashPro Express v12.1 User Guide for all the families](#).

**Figure 4 • Programming Microsemi Devices in a JTAG Chain**

## 3.2 Programming Using an External Programmer

SmartFusion2/IGLOO2 devices can be programmed using an external programmer through the dedicated JTAG port. When programming with the Libero SoC or standalone FlashPro Express software, a FlashPro4/FlashPro5 cable must be connected to the JTAG pins of the device through a 10-pin programming header, as shown in the following figures. The target board must provide power to the VPP, VPPNVM, VDD, and VDDIOx (where x = JTAG bank number) pins.

**Figure 5 • JTAG Programming using External Programmer**

### 3.2.1 Power Supply Requirements for Programming

During SmartFusion2 or IGLOO2 programming, The required power supplies for JTAG must be within the prescribed specification and JTAG signals must be toggling within that specification, see [Figure 2](#), page 13. Required signal integrity measures must be taken to ensure that the power supplies and JTAG signals are free from noise. For information about the tolerance levels, see the [SmartFusion2 SoC FPGA Datasheet](#).

- Each power supply required for programming
- Normal device operation as well as the specification for the JTAG signals

SmartFusion2 and IGLOO2 FPGAs require a single programming voltage to be applied at the VPP pin during programming. This voltage must be supplied from the board. Under normal operating conditions, VPP and VPPNVM must be connected to their respective supply range. For the VPP and VPPNVM

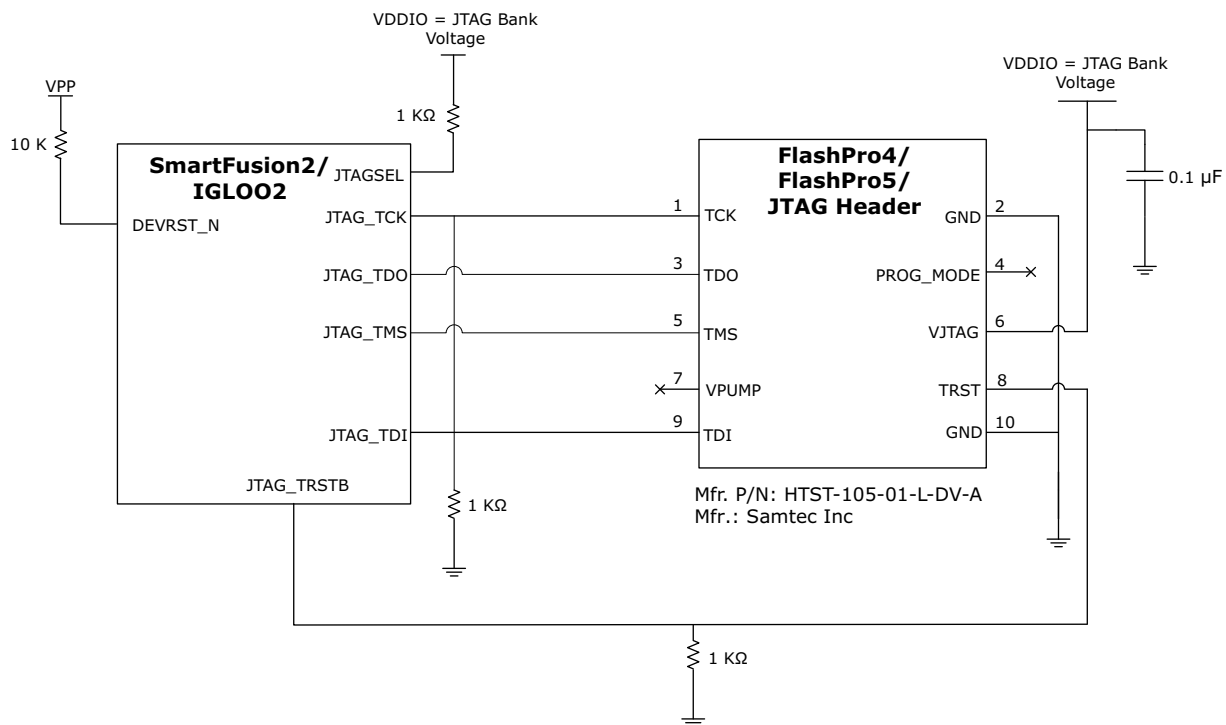
tolerance ranges, see the [SmartFusion2 SoC FPGA Datasheet](#). The board must supply JTAG voltage to the VDDIOx pin of the device and the VJTAG pin of the programmer header.

In Libero SoC 11.8 SP3 and later versions, FlashPro will not detect VPP if the board contains only RTG4/SmartFusion2/IGLOO2 devices. Leave VPUMP pin of the JTAG header floating if the board has only RTG4/SmartFusion2/IGLOO2 devices.

If the board contains ProASIC3/IGLOO/Fusion/SmartFusion devices along with SmartFusion2/IGLOO2 devices in the JTAG chain, connect VPUMP of the ProASIC3/IGLOO/Fusion/SmartFusion device to the JTAG header's VPUMP pin.

Microsemi recommends that VPP and JTAG power supply lines are kept separate with independent filtering capacitors. For capacitor requirements, see the following figure. The bypass capacitor must be placed within 2.5 cm of the device pins. For termination requirements of JTAG pins, see [Table 6](#), page 12.

**Figure 6 • JTAG Programming of a SmartFusion2 Device**



### 3.3 Programming Using an External Microprocessor

Programming using an on-board microprocessor can be accomplished by the following:

- The microprocessor's GPIO driving the JTAG or system controller's SPI port
- An algorithm written in C code known as DirectC

DirectC supports programming the FPGA fabric, eNVM, and security settings individually or at the same time. For more information, see the [DirectC User Guide](#).

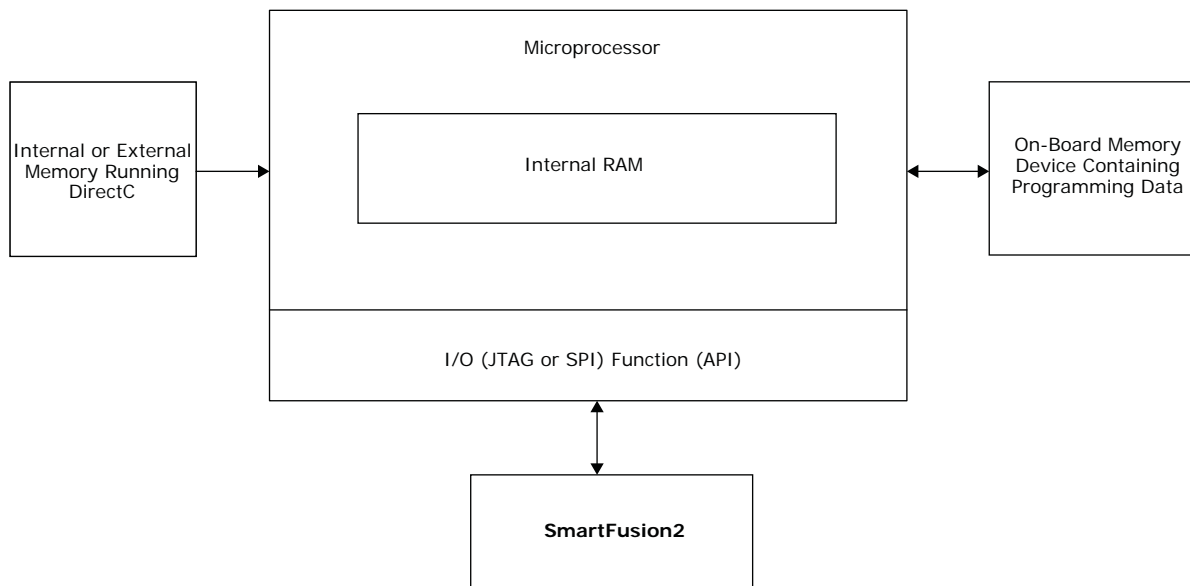
**Note:** Programming through system controller's SPI port using DirectC-SPI is discussed in [SPI Slave Programming](#), page 18.

The following figure shows the programming implementation using:

- An external microprocessor with at least 256 bytes of RAM
- A memory device to store and run DirectC
- Access to the data file containing the programming bitstream
- A JTAG or SPI interface<sup>3</sup> to the target device

An API, I/O functions emulating JTAG or SPI protocols must be created to provide the required functions that act as an interface between DirectC and the hardware. The API is then linked to DirectC and compiled using the target microprocessor's compiler. Once the API is compiled, the resulting binary must be downloaded into the flash memory of microprocessor where it can be accessed to program the device. While programming using the external microprocessor, the same board level considerations must be followed as mentioned in the [Programming Using an External Programmer](#), page 15.

**Figure 7 • Programming Using an External Microprocessor**



## 4 SPI Slave Programming

In the SPI slave programming mode, an external SPI master programs the SmartFusion2 or IGLOO2 devices. The SPI master can be an external microprocessor or a programmer such as FlashPro5. The SPI master interfaces with the system controller through a dedicated SC\_SPI port, which is in SPI slave mode by default. The system controller operates in a fixed SPI mode, 3 (SPO/CPOL= SPH/CPHA =1). For specific device or package support of the SC\_SPI port, see [Table 3](#), page 7. Microsemi DirectC-SPI programming must be used with an external microprocessor that acts as a master to program the SmartFusion2 or IGLOO2 devices in the SPI-Slave mode. For more information, see the [SPI-DirectC Solution User Guide](#).

### 4.1 Programming Interface Overview

The system controller's dedicated SC\_SPI port is located in different bank numbers based on the die or package. See the pin tables of the respective devices for the bank numbers. These ports are part of MSIOs. The bank must be powered up before programming in SPI mode. The following table provides description of the dedicated SC\_SPI pins.

**Table 7 • Dedicated SC\_SPI Pins<sup>1</sup>**

Name	Polarity	Direction	Descriptions
SC_SPI_SS	Low	In	SPI slave select
SC_SPI_SDO	High	Out	SPI data output
SC_SPI_SDI	High	In	SPI data input
SC_SPI_CLK	High	In	SPI clock

1. If unused, these pins must be left floating. These pins have internal pull up activated after the device is powered up.

#### 4.1.1 Design Implementation

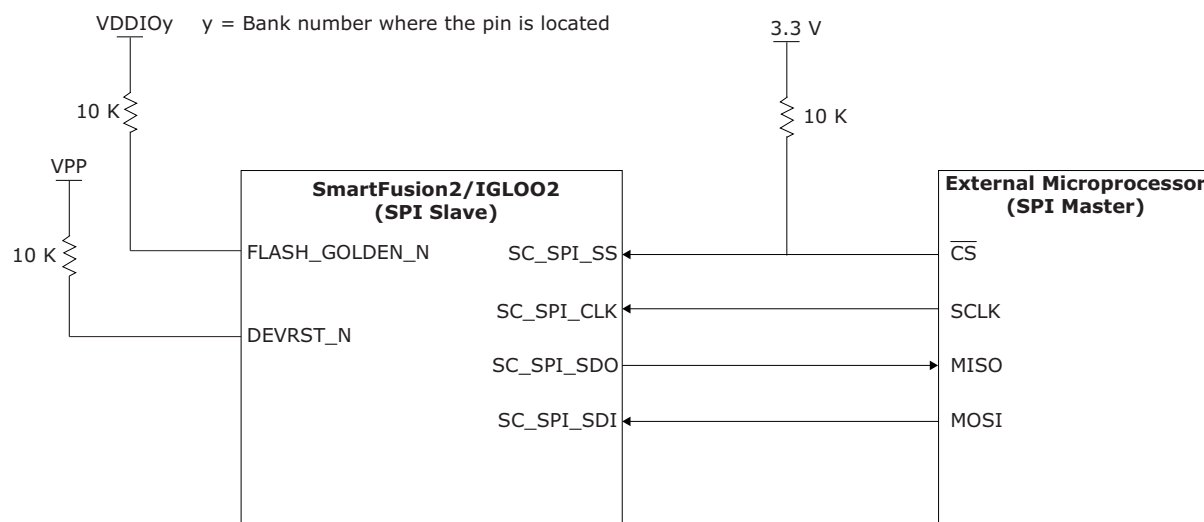
The following figure shows the recommended board configuration for SPI slave programming using an external microprocessor. The command protocol used on the SPI Slave programming is similar to the JTAG command protocol. The target board must provide power to the VPP, VPPNVM, VDD, and VDDIO<sub>x</sub> (where x = SC\_SPI bank number) pins.

For the recommended voltage ranges and pin locations, see the [SmartFusion2 and IGLOO2 Datasheet](#) and the corresponding package pin assignment table.

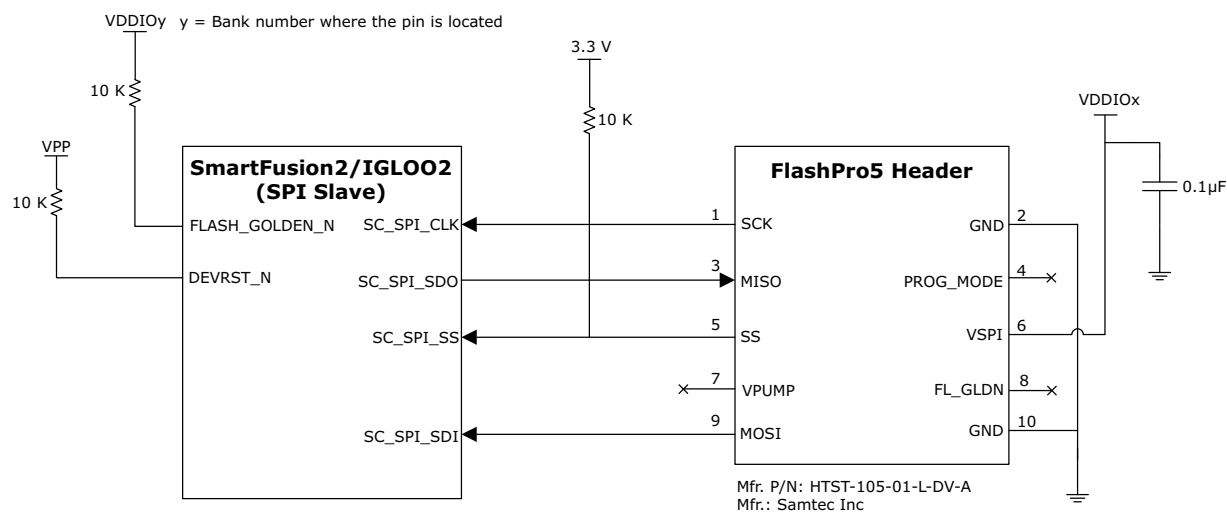
You do not need to reset the FPGA after programming; the DirectC algorithm resets it after the programming is completed. For information about the I/O states during SPI slave programming, see [State of SmartFusion2 and IGLOO2 Components During Programming](#), page 44.

**Note:** During JTAG or SPI-Slave programming, do not run any of the AES/DRBG/ECC Point Multiplication system services when System controller is processing the initial component of the bitstream (BITS). The system service may corrupt the bitstream information if the user design requests AES/DRBG/ECC Point Multiplication system services. This issue does not exist in Auto-update, IAP or Programming recovery. If these security system services must be run during programming, you must generate a STAPL/DAT file using Libero 11.8 SP3 or contact [Technical Support](#) to use older Libero versions.



**Figure 8 • SPI Slave Programming by External Microprocessor**

The following figure shows the recommended board configuration for SPI slave programming using an external programmer.

**Figure 9 • SPI Slave Programming by External Programmer**

## 5 Auto Programming

In auto programming mode, SmartFusion2 and IGLOO2 devices program themselves by downloading the bitstream from an external SPI flash device. The system controller configures the MSS or HPMS SPI\_0 port for the SPI master mode when the dedicated FLASH\_GOLDEN\_N pin is asserted low upon system reboot. An external SPI flash memory device programmed with a bitstream has to be connected to the SPI\_0 port. SPI\_0 port requires serial flash devices supporting the RDID (9F), 24-bit read (0B), and 32-bit read (0C). If RDID indicates a size greater than 2<sup>24</sup> the firmware switches to using the 0C command, otherwise it uses 0B.

**Note:** FLASH\_GOLDEN\_N pin is not available in all packages. For more information, see [Table 3](#), page 7.

### 5.1 Programming Interface Overview

MSS/HPMS SPI\_0 port is used as the programming interface during auto programming. SPI\_0 port is located in a specific bank. See the pin tables of the respective devices for the bank numbers. These ports are part of MSIOs. The bank must be powered up before programming the devices. The following table provides description of the SPI\_0 pins.

**Table 8 • MSS/HPMS SPI\_0 Signals**

Name	Type	Descriptions
SPI_0_SDI	Input	Serial data input
SPI_0_SDO	Output	Serial data output
SPI_0_CLK	Output	Serial clock. It is a serial programmable bit rate clock out signal.
SPI_0_SSO <sup>1</sup>	Output	Slave select
FLASH_GOLDEN_N <sup>1</sup>	Input	If pulled low, the SPI_0 port is put into master mode, which indicates that the device is to be reprogrammed from an image in the external SPI flash attached to the SPI_0 interface.

1. Active low signal.

#### 5.1.1 Design Implementation

In auto programming, an external SPI flash memory is preprogrammed and attached to the MSS/HPMS SPI\_0 port of a blank SmartFusion2 or IGLOO2 device.

The programming bitstream must be programmed at a location other than address 0x00 of the SPI flash connected to the MSS/HPMS SPI\_0 port. Address 0x00 is reserved for the SPI directory.

During auto programming, authentication system service is not available. Microsemi recommends encrypting the golden bitstream with custom security. Custom security is enabled in the Libero SoC software (see Security Policy Manager in the Libero Online Help). For more information about authentication, see [In-Application Programming](#), page 28.

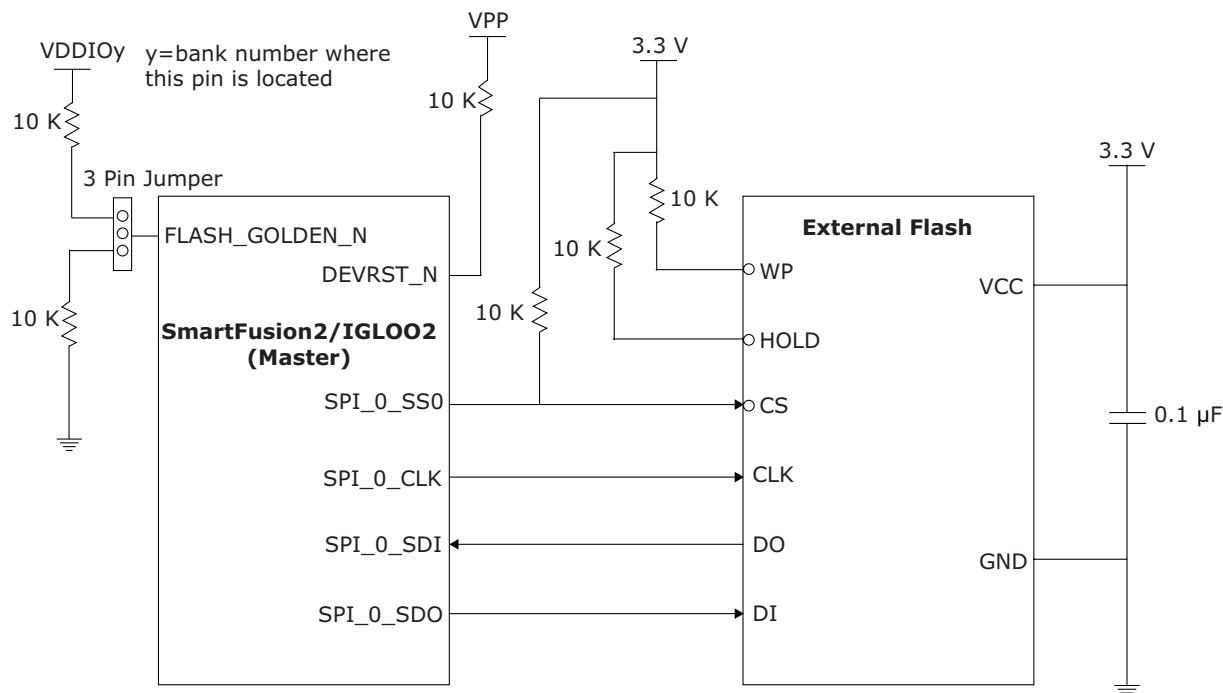
When programming begins, if the system controller detects a corrupted bitstream, programming is aborted. In this case, a valid bitstream must be programmed into the SPI flash and auto programming needs to be reinitiated.

The following figure shows the recommended board configuration to support auto programming. The target board must provide power to the VPP, VPPNVM, VDD, and VDDIOx (where x = SPI\_0 interface bank number) pins.

For the recommended voltage ranges and pin locations, see the [SmartFusion2 and IGLOO2 Datasheet](#) and the corresponding package pin assignment table.

For information about the I/O states during auto programming, see [State of SmartFusion2 and IGLOO2 Components During Programming](#), page 44.

**Figure 10 • SmartFusion2/IGLOO2 MSS/HPMS SPI\_0 Port Configured for Auto Programming (Except 050 Device)**



The MSS/HPMS SPI\_0 port is enabled to operate as SPI master at power-on-reset when the FLASH\_GOLDEN\_N pin is asserted low (pulled low with a 10 K resistor).

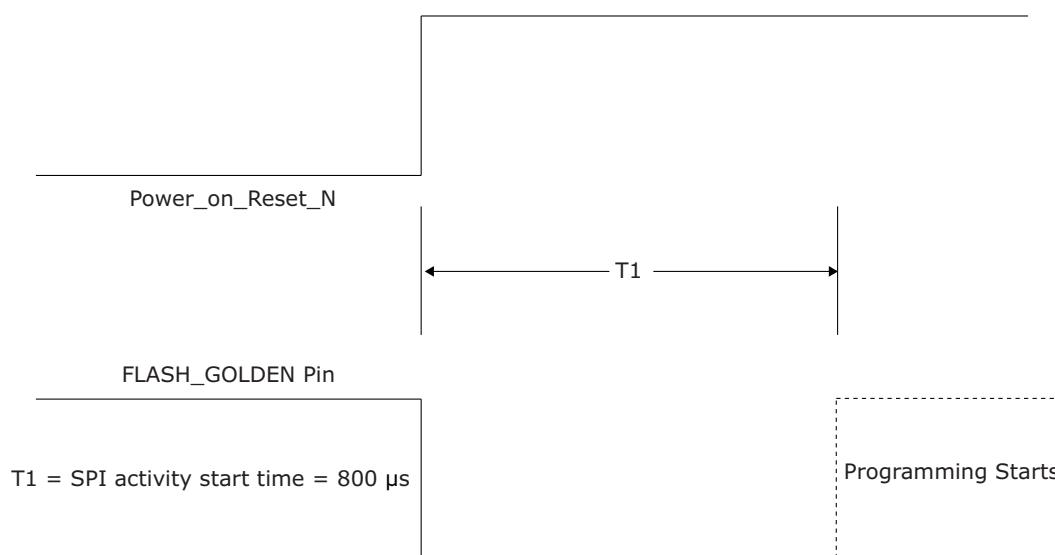
The FLASH\_GOLDEN\_N pin must be sensed by the system controller for a certain amount of time after the DEVRST\_N pin is deasserted or power-on-reset is completed. See the following figure for the timing relations between power-on-reset and the FLASH\_GOLDEN\_N pin.

Figure 10, page 21 shows the control of the FLASH\_GOLDEN\_N pin using a three pin jumper. This pin can be controlled by an external signal or a microprocessor.

Four SPI slave select pins of the SPI\_0 interface (SS4, SS5, SS6, and SS7) drive high during auto programming (except 005 and 010 devices). The remaining SPI\_0 pins are tristated with weak pull up. These four pins that drive high during auto programming must not be used as control pins.

The I/O level will be the same as bank voltage used for SPI\_0 bank and the drive strength will be maximum for this I/O level. For example, if the SPI\_0 bank was powered by 1.8 V, then the drive strength will be the maximum value allowed for 1.8 V LVCMOS I/O. For the maximum drive strength allowed for LVCMOS 1.8 V transmitter, see the [SmartFusion2 and IGLOO2 Datasheet](#).

The system controller reads the SPI flash device ID and device density to determine the read algorithm, and then programs the device with the bitstream. In auto programming, SmartFusion2 and IGLOO2 devices are blank, and hence SPI signal polarity is mode 3 (SPO=SPH=1).

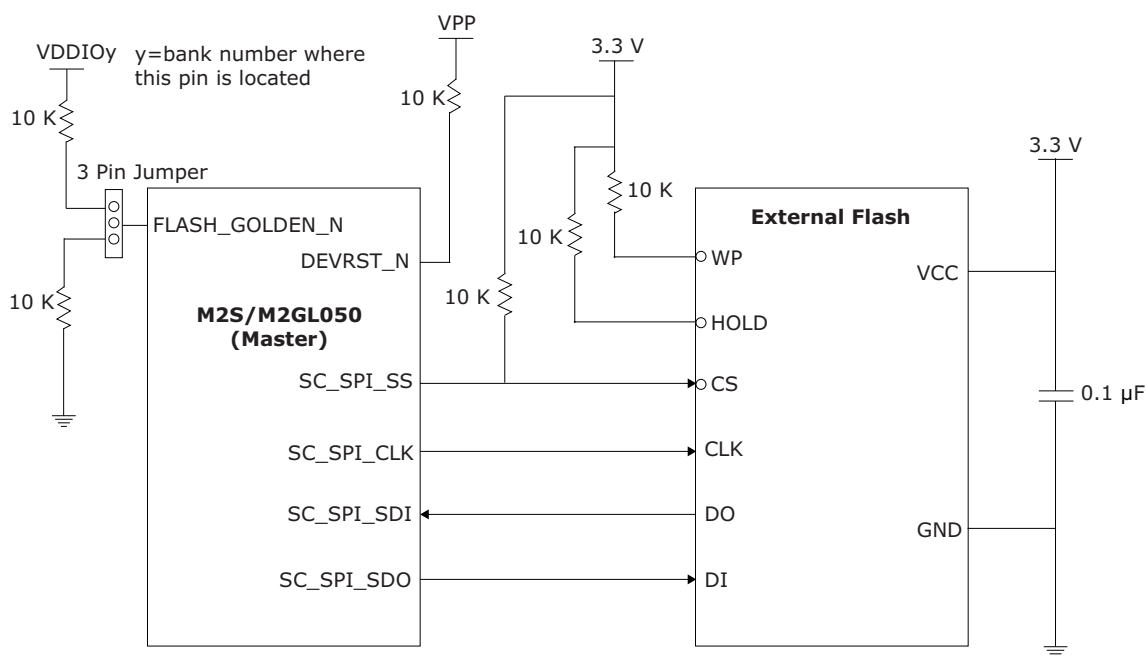
**Figure 11 • Timing Relationship of Reset and FLASH\_GOLDEN\_N Pin**

### 5.1.2 Auto Programming of M2S/M2GL050 Device

Auto programming in M2S/M2GL050 devices is performed slightly different than the other devices. The system controller's dedicated SPI port (SC\_SPI) is used in auto programming of 050 devices, as shown in the following figure.

The SC\_SPI port is enabled to operate as SPI master at power-on-reset or when DEVRST\_N is asserted low, and when the FLASH\_GOLDEN\_N pin is asserted low (pulled low with a 10 K resistor).

Like other devices, an external SPI flash device needs to be connected with the SC\_SPI port, and the standard 0B read command needs to be supported by a serial flash. The bitstream needs to be preprogrammed at 0x00 location of the external SPI flash device. Other requirements remain the same.

**Figure 12 • Auto Programming Scheme for M2S/M2GL050 Devices**

## 6 MSS ISP (SmartFusion2 Only)

---

In this mode, programming is performed under the control of the Cortex-M3 firmware that fetches the programming bitstream directly from one of the communication ports (such as, USB, MMUART, SPI, and I<sup>2</sup>C) and feeds to the system controller through the COMM\_BLK.

The fabric is not operational during programming or verification operations. Downloading the programming bitstream and programming the fabric and/or the eNVM with that bitstream are done in a single step. SmartFusion2 devices must be preprogrammed with the user firmware that calls the ISP service of the system controller to execute programming.

### 6.1 Design Implementation

MSS ISP service enables programming SmartFusion2 devices directly in a single step via any of the dedicated communication ports. As shown in the following figure, the programming bitstream is received from the remote PC through a USB connection. During programming, the fabric enters into F\*F mode. During ISP, the communication I/O port used to fetch the bitstream must be enabled. This option can be set in the I/O Attributes menu in the Libero SoC software (see [Use of Flash Freeze Mechanism in Device Programming](#), page 47). The MSS must continue to operate for the entire duration of the programming operation. The programming recovery features can be implemented to provide a recovery mechanism in case a power failure occurs during programming. For more information, see [Programming Recovery](#), page 37.

The MSS ISP service call has three modes:

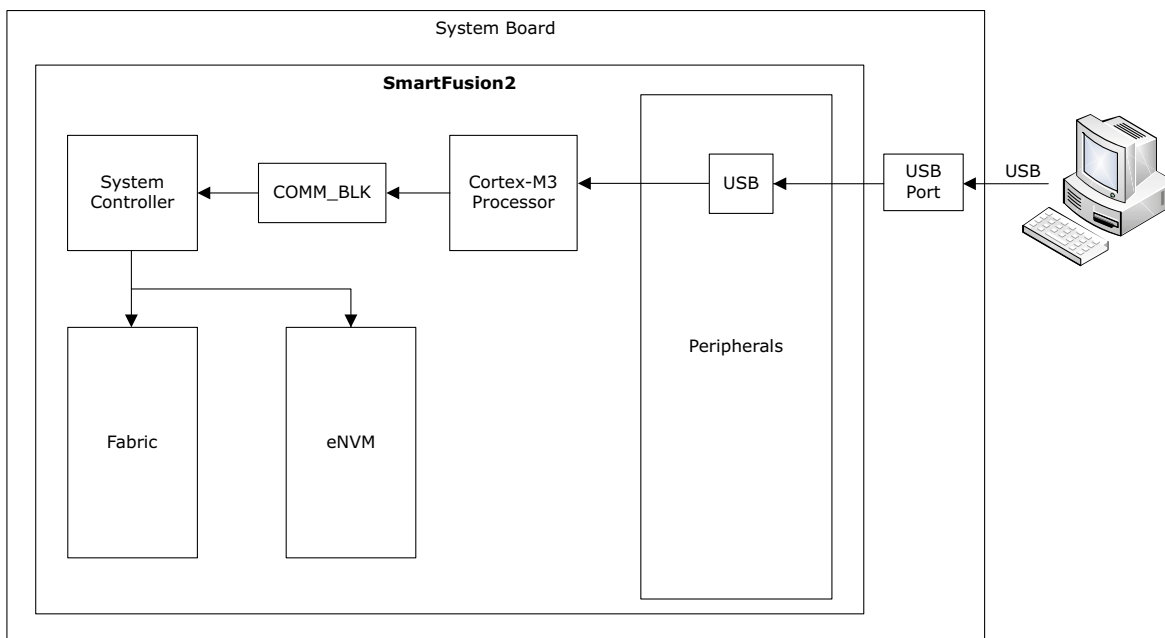
- Authentication
- Programming
- Verification

For more information, see [In-Application Programming](#), page 28.

The target board must provide power to the VPP, VPPNVM, VDD, and VDDIOx (where x = communication port bank number) pins.

For the recommended voltage ranges and pin locations, see the [SmartFusion2 and IGLOO2 Datasheet](#) and the corresponding package pin assignment table.

For information about the I/O states during MSS ISP programming, see [State of SmartFusion2 and IGLOO2 Components During Programming](#), page 44.

**Figure 13 • MSS ISP Update Process**

The application firmware must be written based on the ISP commands shown in the following table. The Libero SoC software provides the ISP system services driver that allows reprogramming the device using the `MSS_SYS_start_isp()` function. For more information, see the *SmartFusion2 MSS System Services Driver User Guide*, which can be download from the Libero SoC firmware catalog.

**Table 9 • ISP Programming Service Request**

Offset	Length (bytes)	Field	Description
0	1	CMD=21	Command
1	1	OPTIONS	See the following table.
2	BSLENGTH	BSDATA	Bitstream data

The following tables list the options and modes.

**Table 10 • ISP Programming Options**

7	6	5	4	3	2	1	0
							MODE

**Table 11 • ISP Programming Modes**

Mode	Operation
0	AUTHENTICATE
1	PROGRAM
2	VERIFY

The following tables describe the ISP response and the corresponding programming service status codes.

**Table 12 • ISP Responses**

Offset	Length (bytes)	Field	Description
0	1	CMD=21	Command
1	1	STATUS	Command status, see the following table

**Table 13 • ISP Programming Service Status Codes**

Status								Description
7	6	5	4	3	2	1	0	
0								Success
0				AUTHERRCODE				Authentication error
1	0			ERRORCODE				Programming error
255								Service protected

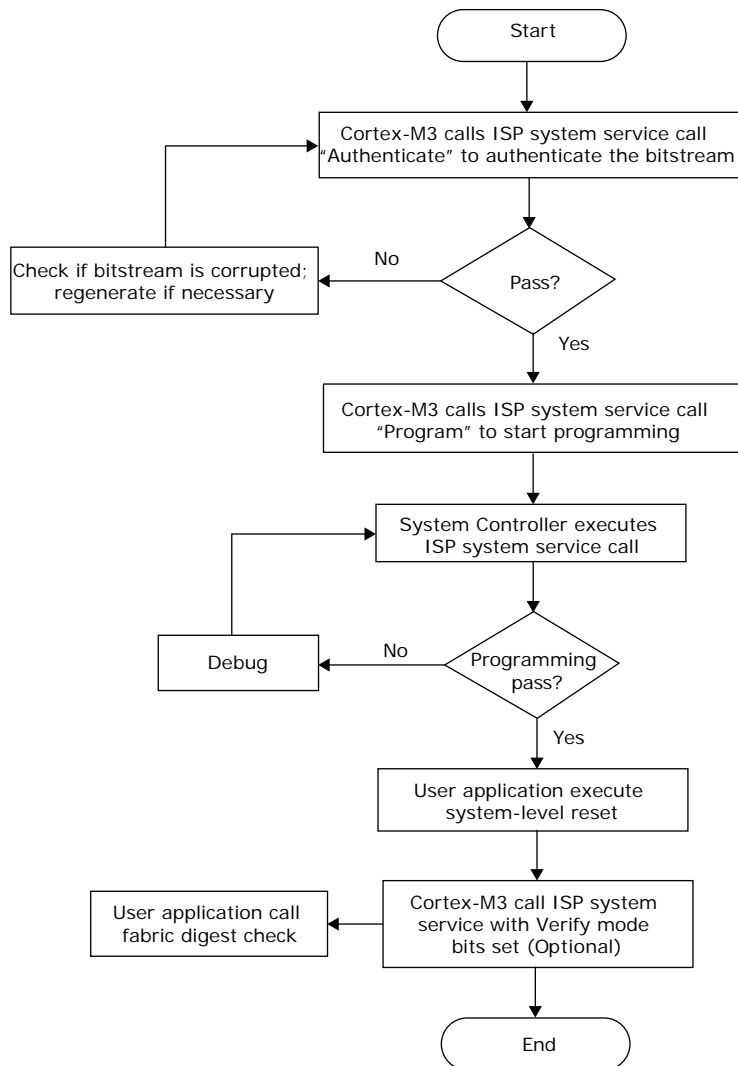
An authentication error might occur in any frame of the bitstream. See [Table 19](#), page 32 and [Table 20](#), page 32 for details on error codes. If the MODE is PROGRAM, and there is a failure, the device might be left in an inconsistent and non-operational state. Therefore, the user application must only commit the bitstream for programming after successful authentication, and thus the integrity of the bitstream is validated.

After the programming operation is complete, an internal reset is generated for the new design to take effect, and the completion handler cannot be called by the system services driver. No exit codes are reported, which is normal and expected. The completion of the programming must be realized by observing the new design functioning immediately after programming. If there is a problem, and the part did not reset, then you can observe the completion handler.

**Note:** The `isp_completion_handler` function must be implemented by the application program, and it is called by the system services driver when an ISP operation initiated by a call to `MSS_SYS_start_isp()` completes.

The following figure illustrates the flow of the MSS ISP update.

**Figure 14 • MSS ISP Update Flow**



The Cortex-M3 processor executes the user application. The user application provides the bitstream through the communication interface required to authenticate, program, and verify the operations.

During the execution of the ISP service, the system controller receives the entire bitstream as a continuous stream of bytes through the COMM\_BLK. If the Cortex-M3 processor is executing code from the eNVM, the system controller firmware must avoid a deadlock scenario, if it attempts to program a region of the eNVM that is used. This is achieved by releasing the eNVM HMASTLOCK between atomic eNVM operations. If the bitstream reprograms the code region that is used, the Cortex-M3 processor must copy its code to eSRAM and execute from there. For more information about HMASTLOCK, see the [SmartFusion2 Microcontroller Subsystem User Guide](#).

Upon completion of programming, the system controller resets the device to ensure that the newly programmed configuration takes effect and the device operates with the new design. However, the user application must execute another system reset in addition to system controller reset as soon as the ISP system service is completed, otherwise the LSRAM block might not be accessible. System reset can be generated using the tamper macro, which can be instantiated from the Libero SoC Catalog. Immediately after ISP, user logic (that is, an FSM module) can check LSRAM access. If the access is denied, then the user logic generates a reset signal that can trigger the tamper macro to execute the system-level reset. The RESET function in the tamper macro configuration window must be enabled. For detailed implementation along with a reference design, see the [SmartFusion2 SoC FPGA In-System](#)



*Programming Using UART Interface Demo Guide or SmartFusion2 SoC FPGA - In-System  
Programming Using USB OTG Controller Interface Demo Guide.*

The system controller runs verification as part of programming operation. However, verification can be executed as a standalone operation to verify the contents of the SmartFusion2 device against the bitstream.

When a standalone verification is required, execute the fabric digest check before calling the VERIFICATION system service. A snippet from the firmware code is given below:

```
if(mode == MSS_SYS_PROG_VERIFY)
{
    //Check digest before IAP verify
    MSS_SYS_check_digest(MSS_SYS_DIGEST_CHECK_FABRIC);
}
```

## 7 In-Application Programming

In-application programming (IAP) is a two-step process:

1. The first step of the IAP is different for SmartFusion2 and IGLOO2 devices.  
SmartFusion2:

The user application in the Cortex-M3 processor receives the new bitstream from any of the dedicated communication ports such as USB, as shown in Figure 16, page 30. The Cortex-M3 processor writes the new bitstream into the external SPI flash that is attached to the MSS SPI\_0 port.

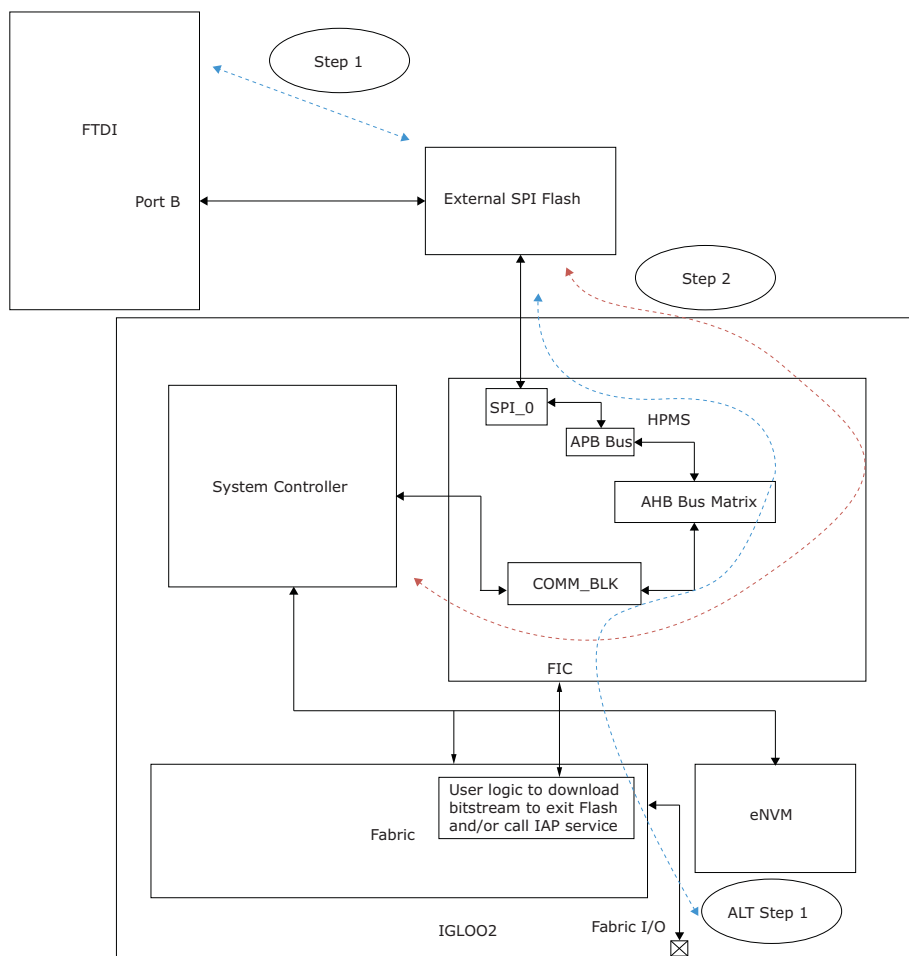
**Note:** The system controller cannot write to the external SPI flash.

IGLOO2:

The first step can be accomplished in one of these two ways for IGLOO2 devices:

- The user application programs the bitstream into the external SPI flash attached to the HPMS SPI\_0 port using an external programmer such as FTDI.
- The user logic in the FPGA fabric implemented using the soft microcontroller such as CoreABC pushes the bitstream from the fabric I/O to the SPI\_0 port, as shown in the following figure.

**Figure 15 • IGLOO2 In-Application Programming Interface**



**Note:** The system controller cannot write to the external SPI flash.

2. In this step, the new bitstream must be verified in the external SPI flash by requesting **AUTHENTICATE** from the IAP system service call. The user application then initiates a

programming cycle by requesting the PROGRAM IAP system service call from the system controller. The system controller fetches the new bitstream from SPI\_0 flash and programs the device and authenticates it on-the-fly. The bitstream must be authenticated before programming to avoid a non-operational state. The SPI\_0 peripheral must be configured before initiating this request.

In SmartFusion2, once the IAP service call is initiated, the Cortex-M3 firmware is no longer involved in the IAP. During IAP, the fabric is usable during the first step of the IAP service call. However, it is not usable during the second, which is the programming stage.

## 7.1 Design Implementation

In IAP, the device must be preprogrammed with an appropriate configuration to fetch data through one of the MSS/HPMS communication peripherals and to access the data from the external SPI flash that is connected to the SPI\_0 port. During programming, the fabric enters into Flash\*Freeze mode. During IAP, the communication I/O port used to fetch the bitstream must be enabled. This option can be set in the I/O Attributes menu in the Libero SoC software (see [Use of Flash Freeze Mechanism in Device Programming](#), page 47). When the IAP system service call is being executed, the system controller requires exclusive access to the SPI\_0 port. During this step, the application must not attempt to access the SPI\_0 port.

Four SPI slave select pins of the SPI\_0 interface (SS4, SS5, SS6, and SS7) drive high during programming (except 005 and 010 devices). The remaining SPI\_0 pins are tristated with weak pull up. These four pins that drive high during programming must not be used as control pins. The I/O level and drive strength are based on the previous settings programmed into the device.

There is no limitation on the number of bitstreams that can be stored in external SPI flash other than the density of the SPI flash itself. As part of the IAP command, the starting address of the bitstream must be given.

The target board must provide power to the VPP, VPPNVM, VDD, and VDDIOx (where x = SPI\_0 interface bank number) pins.

For the recommended voltage ranges and pin locations, see the [SmartFusion2 and IGLOO2 Datasheet](#) and the corresponding package pin assignment table.

For information about the I/O states during IAP programming, see [State of SmartFusion2 and IGLOO2 Components During Programming](#), page 44.

The service call has three modes: authenticate, program, and verify.

### 7.1.1 Authenticate

The FPGA fabric and MSS/HPMS are operational when the bitstream is authenticated.

### 7.1.2 Verify

When standalone verification is required, call the fabric digest check before calling the VERIFICATION system service. A snippet from the firmware code is given below:

```
if(mode == MSS_SYS_PROG_VERIFY)
{
    //Check digest before IAP verify
    MSS_SYS_check_digest(MSS_SYS_DIGEST_CHECK_FABRIC);
}
```

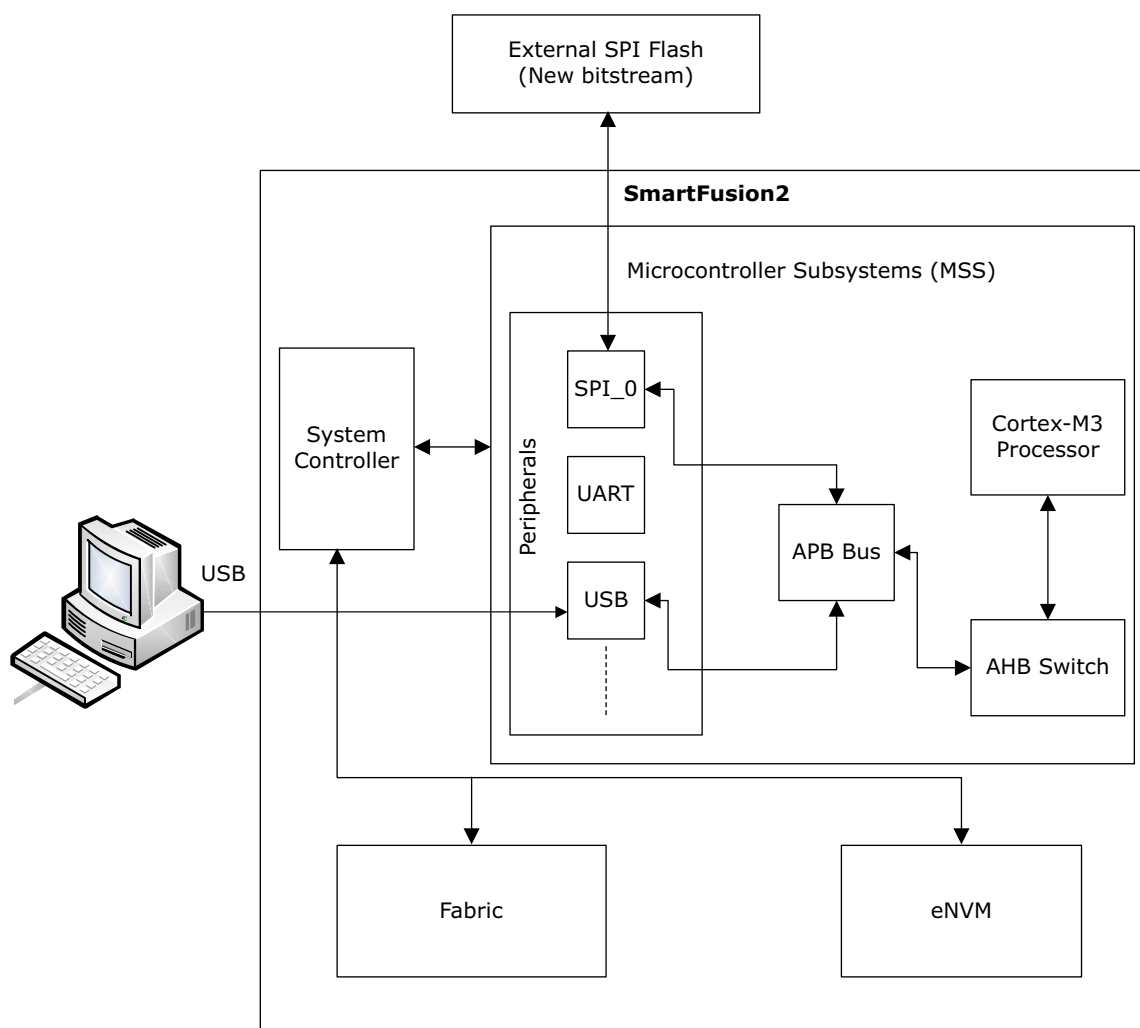
The device is automatically placed in the F\*F state for the duration of the verify operation and automatically awakened upon completion. F\*F entry and exit messages are transmitted before the service response.

### 7.1.3 Program

The device is automatically placed in the F\*F state before programming commences and the F\*F entry message is transmitted. If the programming operation is successful, the device goes through a power-on-reset sequence generated by the system controller.

However, the user application must execute another system reset in addition to system controller reset as soon as the IAP system service is completed, otherwise the LSRAM block might be inaccessible. System reset can be generated using the tamper macro, which can be instantiated from the Libero SoC Catalog. Immediately after IAP, user logic (that is, an FSM module) can check LSRAM access. If the access is denied, then user logic generates a reset signal that can trigger the tamper macro to execute the system-level reset. The RESET function in the tamper macro configuration window must be enabled. For detailed implementation along with reference design, see the *SmartFusion2 SoC FPGA In-Application Programming Using PCIe Interface Demo Guide* or *Implementing Programming Recovery and In-Application Programming Features Using Ethernet Interface for SmartFusion2 Devices Demo Guide*.

**Figure 16 • SmartFusion2 In-Application Programming**



The following tables list the IAP command options. For more information about these commands, see the *SmartFusion2 MSS System Services Driver User Guide*, which can be downloaded from the Libero SoC firmware catalog.

**Table 14 • IAP Service Requests**

Offset	Length (bytes)	Field	Description
0	1	CMD=20	Command
1	1	OPTIONS	See the following table
2	4	SPIADDR	Base address of bitstream in MSS/HPMS SPI 0 (MS byte ignored)

**Table 15 • IAP Programming Options**

7	6	5	4	3	2	1	0
							MODE

**Table 16 • IAP Programming Modes**

Mode	Operation
0	AUTHENTICATE
1	PROGRAM
2	VERIFY

The following tables list the IAP service responses, status codes, error codes, and bitstream authorization error codes receptively.

**Table 17 • IAP Service Responses**

Offset	Length (bytes)	Field	Description
0	1	CMD=20	Command
1	1	STATUS	Command status, see <a href="#">Table 13</a> , page 25

**Table 18 • IAP Programming Service Status Codes**

Status								Description
7	6	5	4	3	2	1	0	
0								Success
0				AUTHERRCODE				Authentication error
1	0			ERRORCODE				Programming error
255								Service protected

**Table 19 • Error Codes**

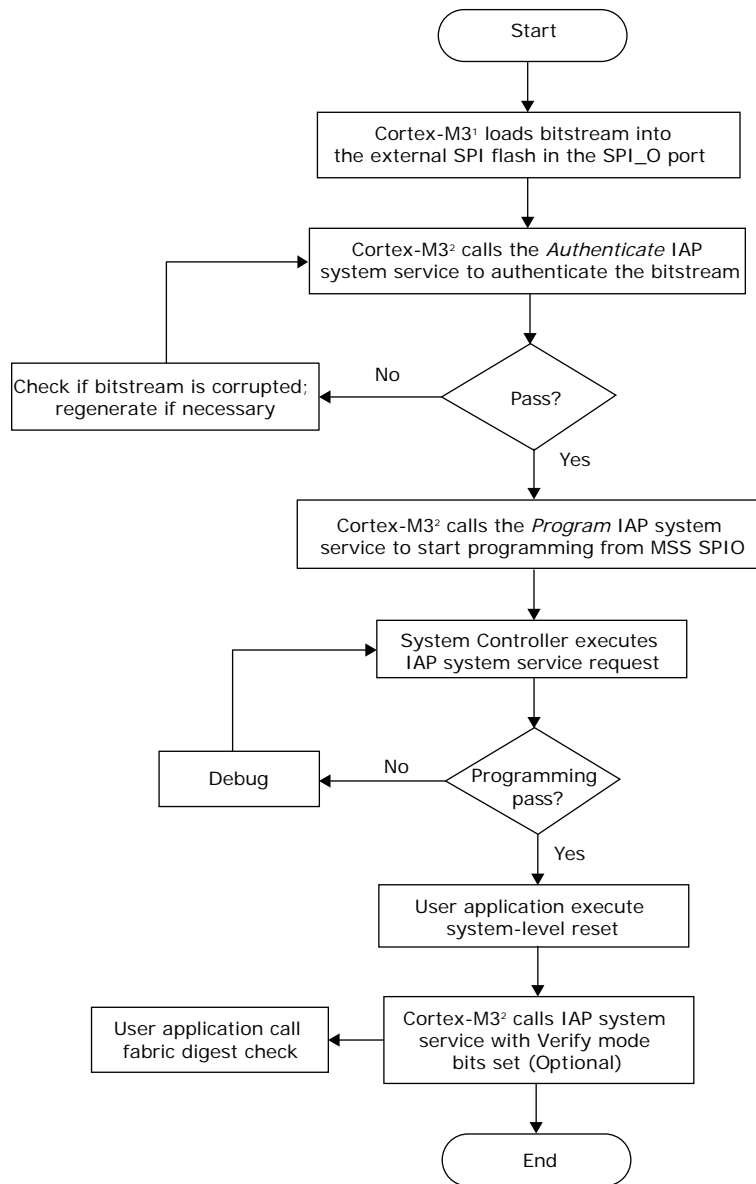
Error Code	Description	Possible Cause	Possible Solution
0	No error/success.		
1	Fabric programming/verification failed.	Device is programmed with a different design or the component is blank.  Unstable voltage level.	Verify if the device is programmed with the correct data/design.  Monitor the related power supplies that cause the issue during programming; check for transients violating Microsemi specifications. For more information about transient specifications, see <a href="#">DS0128: IGLOO2 and SmartFusion2 Datasheet</a> .
2	Device security prevented the operation.	Current device security settings prevents programming.	Security settings can be queried from the device with a STAPL file by running action DEVICE_INFO.  Security settings can be changed with a STAPL file with a passkey (UPK1) that matches the device.
4	eNVM programming operation failed.	Unstable voltage level.	Monitor the related power supplies that cause the issue during programming; check for transients violating Microsemi specifications. For more information about transient specifications, see <a href="#">DS0128: IGLOO2 and SmartFusion2 Datasheet</a> .
5	eNVM verify operation failed.	Device is programmed with a different design. Unstable voltage level.	Verify if the device is programmed with the correct data/design.  Monitor the related power supplies that cause the issue during programming; check for transients violating Microsemi specifications. For more information about transient specifications, see <a href="#">DS0128: IGLOO2 and SmartFusion2 Datasheet</a> .

**Table 20 • IAP Bitstream Authorization Error Codes**

AUTHERRCODE	Description
0	No error
1, 2, 3, 4, 7, 8, 9	Invalid/corrupted bitstream
5	Design version in the bitstream is earlier than the version allowed by the BACKLEVEL specified in the device.
10	Incorrect DEVICEID.
11	Bitstream is outdated; must regenerate.
12	Bitstream does not support verification.

The following figure summarizes the two-step IAP flow.

**Figure 17 • IAP Flow**



<sup>1</sup>For IGLOO2 device, either external programmer or user logic in the fabric loads the bitstream into the external SPI flash in the SPI\_O port.

<sup>2</sup>For IGLOO2 device, user logic calls IAP system service.

The Cortex-M3 processor executes the user application. The user application provides the bitstream through the communication interface required to authenticate, program, and verify the operations.

During the execution of the IAP service, the system controller receives the entire bitstream as a continuous stream of bytes through the COMM\_BLK. If the Cortex-M3 processor is executing code from the eNVM, the system controller firmware must avoid a deadlock scenario, if it attempts to program a region of the eNVM that is used. This is achieved by releasing the eNVM HMASTLOCK between atomic eNVM operations. If the bitstream reprograms the code region that is used, the Cortex-M3 processor must copy its code to eSRAM and execute from there. For more information about HMASTLOCK, see the [SmartFusion2 Microcontroller Subsystem User Guide](#).

Upon completion of programming, the system controller resets the device to ensure that the newly programmed configuration takes effect and the device operates with the new design. However, the user application must execute another system reset in addition to system controller reset as soon as the IAP system service is completed, otherwise the LSRAM block might not be accessible. System reset can be generated using the tamper macro, which can be instantiated from the Libero SoC Catalog. Immediately after ISP, user logic (that is, an FSM module) can check LSRAM access. If the access is denied, then the user logic generates a reset signal that can trigger the tamper macro to execute the system-level reset. The RESET function in the tamper macro configuration window must be enabled. For detailed implementation along with a reference design, see the *SmartFusion2 SoC FPGA In-System Programming Using UART Interface Demo Guide* or *SmartFusion2 SoC FPGA - In-System Programming Using USB OTG Controller Interface Demo Guide*.

The system controller runs verification as part of programming operation. However, verification can be executed as a standalone operation to verify the contents of the SmartFusion2 device against the bitstream.

When a standalone verification is required, execute the fabric digest check before calling the VERIFICATION system service. A snippet from the firmware code is given below:

```
if(mode == MSS_SYS_PROG_VERIFY)
{
    //Check digest before IAP verify
    MSS_SYS_check_digest(MSS_SYS_DIGEST_CHECK_FABRIC);
}
```

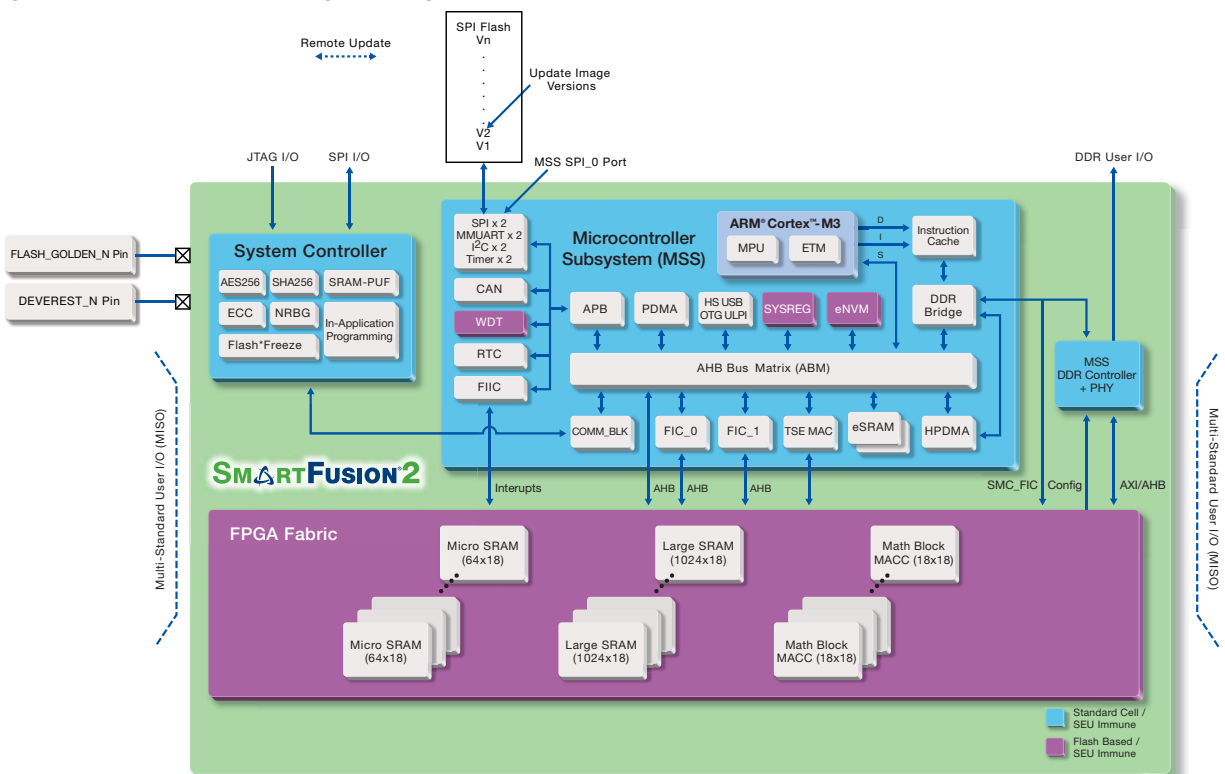


## 8 Auto Update

In auto update programming mode, a device is auto programmed with an update image on power up. The update image is stored in an external flash connected to the MSS/HPMS SPI\_0 port of the device. The design version of the update image must be greater than the image version already programmed in the device.

When auto update is enabled, SmartFusion2/IGLOO2 MSS/HPMS SPI\_0 can be configured to share the SPI\_0 pin with an SPI controller implemented in the FPGA fabric. To share the SPI\_0 port, a multiplexer needs to be implemented in the FPGA fabric to switch the SPI\_0 pins between MSS/HPMS SPI\_0 and the fabric SPI controller. For more information, see *DG0636: Implementing Auto Update and Programming Recovery Features (Using Ethernet Interface) for SmartFusion2 Devices Demo Guide*. Auto update is not supported in M2S/M2GL050 devices.

**Figure 18 • Auto Update Programming Ports**



Auto update mode needs to be turned on in Libero SoC (see *Configuring the Device for Auto Update*, page 36), and the device needs to be programmed first with the exported bitstream. The SPI flash can be preprogrammed and installed on the board or it can be programmed remotely using the update image and the SPI directory. The FPGA can be programmed with only one update image version at a time. Every time auto update of a new update image is desired

- the SPI flash needs to be reprogrammed with that new update image.
- the SPI directory corresponding to the new update image needs to be programmed into the SPI flash before starting the auto update.

During auto update, the system controller reads the version and address of the image from the SPI directory and programs the device with the update image if it is a higher version of the image already programmed in the FPGA.

If auto update is enabled, and the SPI flash is blank or does not contain the SPI directory, the device attempts auto update but fails bitstream authentication and aborts programming. However, the state of

the device is not changed, therefore the device boot sequence continues, and the existing user design powers up.

Four SPI slave select pins of the SPI\_0 interface (SS4, SS5, SS6, and SS7) drive high during auto programming (except 005, 010 devices). The remaining SPI\_0 pins are tristated with weak pull up. These four pins that drive high during auto update must not be used as control pins. The I/O level and drive strength are based on the previous settings programmed into the device.

The target board must provide power to the VPP, VPPNVM, VDD, and VDDIOx (where x = SPI\_0 interface bank number) pins.

For the recommended voltage ranges and pin locations, see the [SmartFusion2 and IGLOO2 Datasheet](#) and the corresponding package pin assignment table.

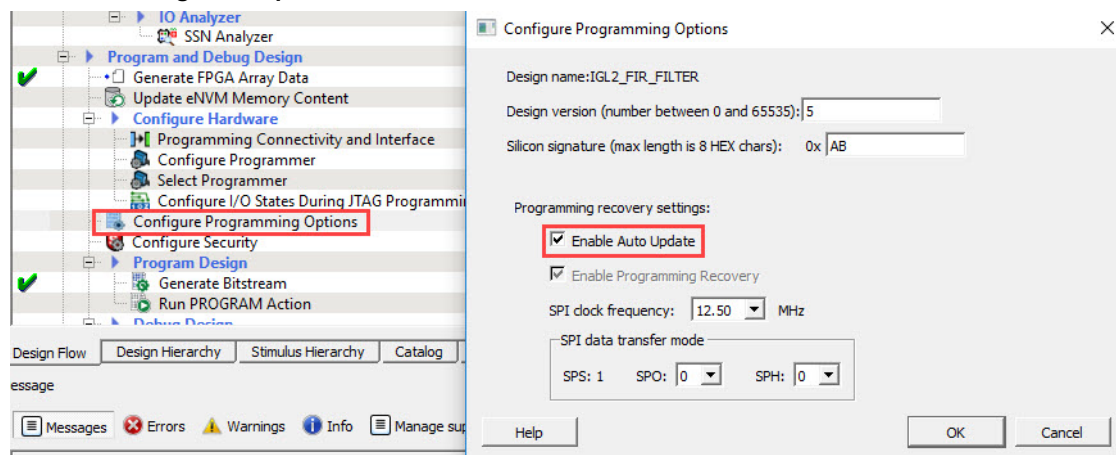
For information about the I/O states during auto update, see [State of SmartFusion2 and IGLOO2 Components During Programming](#), page 44.

## 8.1 Configuring the Device for Auto Update

There are two ways to configure the FPGA for auto update:

- Generate a bitstream with the auto update configuration in the Libero SoC software and program the device to enable this programming mode, as shown in the following figure. For more information on configuring the auto update, see the [Libero SoC User Guide](#).

**Figure 19 • Enabling Auto Update**



- Factory preconfigures the device before shipping. It can be either of the following:
  - In house programming (IHP) at the factory.
  - Programmed with a preconfigured image using a standalone Silicon Sculptor 3 programmer.

## 9 Programming Recovery

Programming recovery, if enabled, allows the device to automatically recover from a power failure during a programming operation. Programming recovery requires an external SPI flash to be connected to SPI\_0 within the MSS/HPMS.

Programming recovery is supported for MSS ISP and IAP and is not intended for recovery from the following events:

- SPI flash error
- Authentication error of programming bitstream
- Loss of communications link during ISP

If there is loss of communications link during programming, the system controller does not time out, but keeps waiting for more data indefinitely. In this case, the communications link must be reestablished.

Programming recovery details:

- If power fails, recovery occurs during power up where neither the FPGA fabric nor the Cortex-M3 processor is active.
- After the recovery is successful, the device is enabled automatically.
- If recovery fails, the device needs to be powered up again.
- If the golden image is corrupted, the recovery occurs each time the device reboots but eventually it fails and the device goes to idle state.

Programming recovery is not supported in M2S/M2GL050 devices.

### 9.1 Programming Recovery Implementation

Programming recovery works through the MSS/HPMS SPI\_0 port. An external SPI flash device needs to be programmed with the golden image and the corresponding SPI directory and connected to the SPI\_0 port.

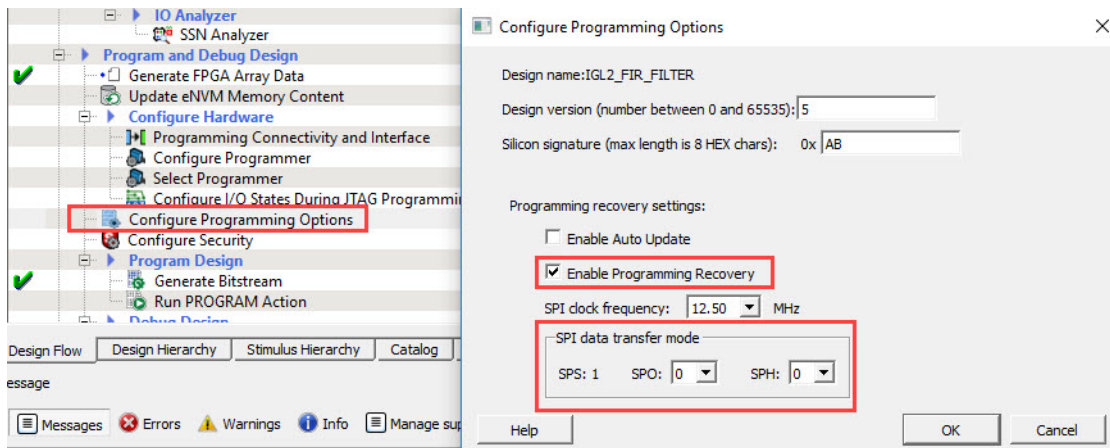
Four SPI slave select pins of the SPI\_0 interface (SS4, SS5, SS6, and SS7) drive high during programming (except 005 and 010 devices). The remaining SPI\_0 pins are tristated with weak pull up. These four pins that drive high during programming must not be used as control pins. The I/O level and drive strength are based on the previous settings programmed into the device.

SmartFusion2 and IGLOO2 devices need to be programmed with programming recovery settings in Libero SoC to enable programming recovery during manufacturing flow, as shown in the following figure. The required security settings are also programmed at that time. After the recovery settings are programmed, they must not be changed or modified during remote updates. The remote update of security or recovery setting in the field poses a risk because if power fails in the middle of the update, the recovery setting might get corrupted. If the recovery setting gets corrupted, the recovery does not take place when power comes back. For more information, see the [Libero SoC User Guide](#).

The system controller is unable to change the oscillator frequency after a successful Programming Recovery. The oscillator frequency remains at 25 MHz instead of 50 MHz. Refer to one the following demo guides for a workaround.

For implementation details, see the following documents:

- [Implementing Auto Update and Programming Recovery Features Using Ethernet Interface for SmartFusion2 Devices Demo Guide](#)
- [Implementing Programming Recovery and In-Application Programming Features Using Ethernet Interface for SmartFusion2 Devices Demo Guide](#)

**Figure 20 • Enabling Programming Recovery**

There are four user-selectable configuration settings for programming recovery, as shown in the following table. These settings are provided as an illustration, actual setting is done through the Libero SoC GUI.

For information about the I/O states during programming recovery, see [State of SmartFusion2 and IGLOO2 Components During Programming](#), page 44.

**Table 21 • Programming Recovery Configuration Settings (UCNFG[16:12])**

Configuration Settings	Description
IAPRECOVERY_ENABLE	0: Enables programming recovery 1: Disables programming recovery [Default]
IAPRECOVERY_PORT_GOLDEN	Specifies the SPI port being used for the golden image. 1: SPI_0: The MSS/HPMS SPI instance 0 [Default and only option]
IAPRECOVERY_IMAGE	Selects the image to be used in recovery. 1: Golden image [Default]
IAPRECOVERY_UPDATE	Turns on the automatic update mode. 0: Checks the update image version on power up, and if the update image version is newer than the design version programmed in the device, performs the update. 1: No action [Default]

## 9.2 SPI Flash Configuration and Image Selection

The image needs to be written to the external SPI flash connected to SPI\_0. The SPI directory contains the addresses of the programming bitstreams stored in the SPI flash. Libero SoC creates the SPI directory (<file\_name>.spidir) as part of the bitstream based on user preference. The following table describes the SPI directory. For more information, see the [Libero SoC User Guide](#).

**Table 22 • SPI Directory**

Offset	Name	Description
0	GOLDEN_IMAGE_ADDRESS[3:0]	Contains the address where the golden image starts.
4	GOLDEN_IMAGE_DESIGNVER[1:0]	Contains the design version of the golden image.
6	UPDATE_IMAGE_ADDRESS[3:0]	Contains the address where the update image starts.
10	UPDATE_IMAGE_DESIGNVER[1:0]	Contains the design version of the update image.

### 9.2.1 MSS/HPMS SPI\_0 Port Configuration

The SPI\_0 port must be configured while enabling programming recovery and/or auto update in the Libero SoC software. Set the SPI clock and SPI data transfer mode in the Libero SoC software in the Configure Programming Recovery window.

**Note:** SPI mode of a blank device is set to 3 (SPO=SPH=1). To operate the SPI bus in other modes, configure them in the Libero SoC software. For more information, see the [Libero SoC User Guide](#).

**Table 23 • User Lock Row Strobe Bits for Programming Recovery**

Name	Description
U_IAPRECOVERY_CLKRATE[7:0]	Sets the SPI clock frequency.
U_IAPRECOVERY_CLKMODE	0: Sets SPICLK using $1/(2^{**}(\text{CLKRATE}+1))$ Where, CLKRATE = 0 to 15 1: Sets SPICLK using $1/(2^{*}(\text{CLKRATE}+1))$ Where, CLKRATE = 0 to 255
U_IAPRECOVERY_SPO	Control SPI signal polarity. See <a href="#">Table 24</a> , page 39.
U_IAPRECOVERY_SPH	Control SPI signal polarity. See <a href="#">Table 24</a> , page 39.
U_IAPRECOVERY_DPC[9:0]	These set the SPI port I/Os settings.

**Table 24 • SPI Signal Polarity Modes**

SPO	SPH	SPI Clock in Idle	Sample Edge	Shift Edge	SPI Select in Idle	SPI Select Between Frames
0	0	Low	Rising	Falling	High	Stays active until all the frames set by frame counter are transmitted.
0	1	Low	Falling	Rising	High	
1	0	High	Falling	Rising	High	
1	1	High	Falling	Falling	High	

**Note:** SPS is set to 1.

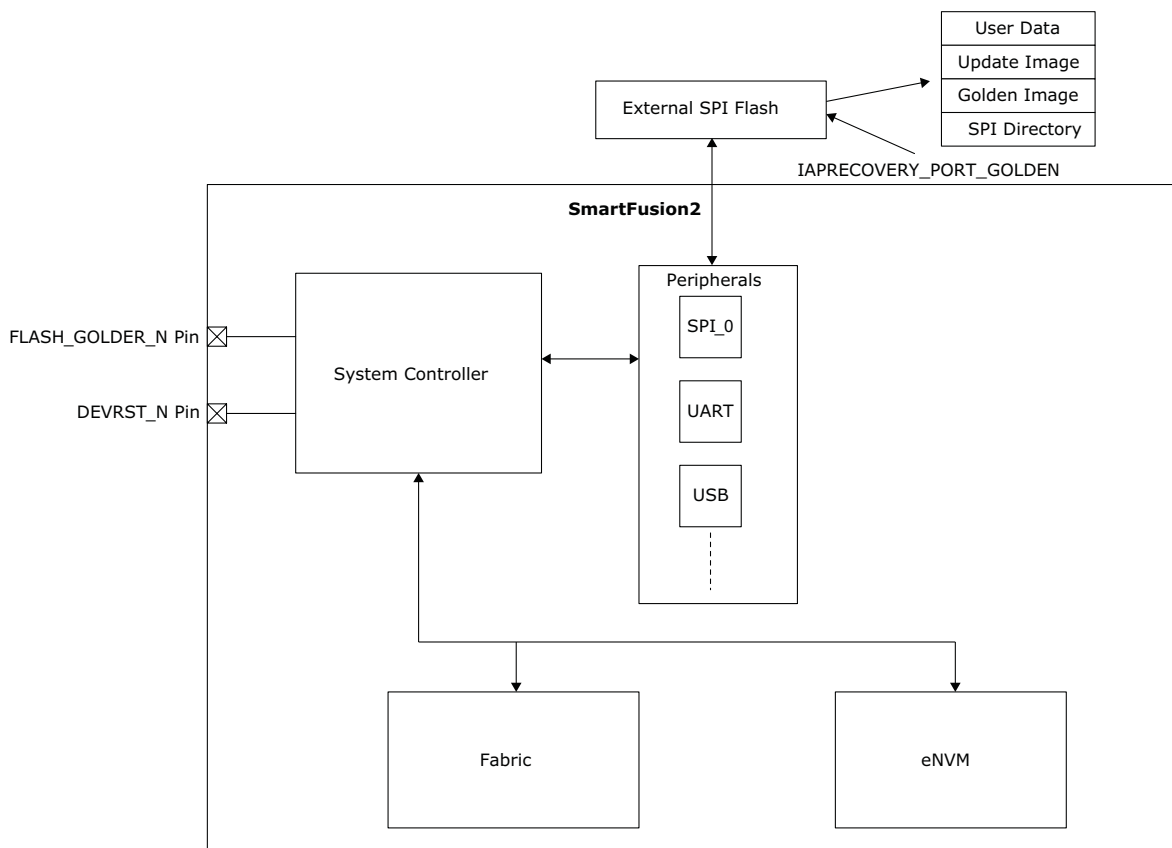
The following table lists the truth table to describe different mechanisms of the programming recovery operation.

**Table 25 • Programming Recovery and Auto Update Truth Table**

Configuration Setting		Events During Assertion of FLASH_GOLDEN_N	Events During Recovery	Events During Power-up
Recovery	Auto Update			
Enable	Enable	Golden image is programmed	Golden image is programmed	Update image is programmed
Enable	Disable	Golden image is programmed	Golden image is programmed	Do nothing
Disable	Disable	Golden image is programmed	Do nothing	Do nothing

The following figure shows the programming recovery configuration for the SmartFusion2 device as described in previous sections. The same settings are applicable for IGLOO2 devices.

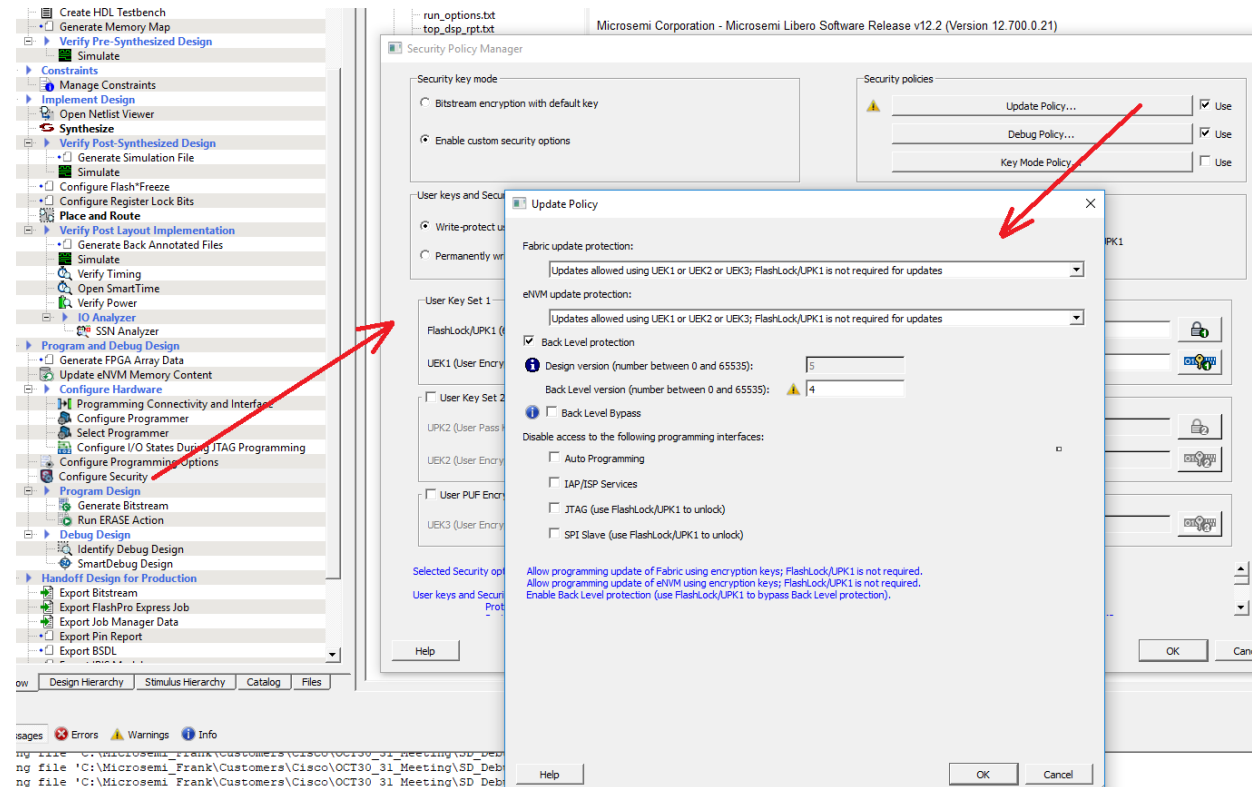
**Figure 21 • Programming Recovery Configuration for SmartFusion2**



## 9.3 Back Level Protection

Bitstream replay protection is provided when back level protection is enabled. It limits the bitstream version that can be programmed into the device. Only programming bitstreams with Designer Version greater than the back level version are allowed for programming. Libero SoC tool's SPM (Security Policy Manager) configures back level protection as shown in the following figure. When enabled, a design being programmed must have its Design version higher than the back level version value already programmed in the device.

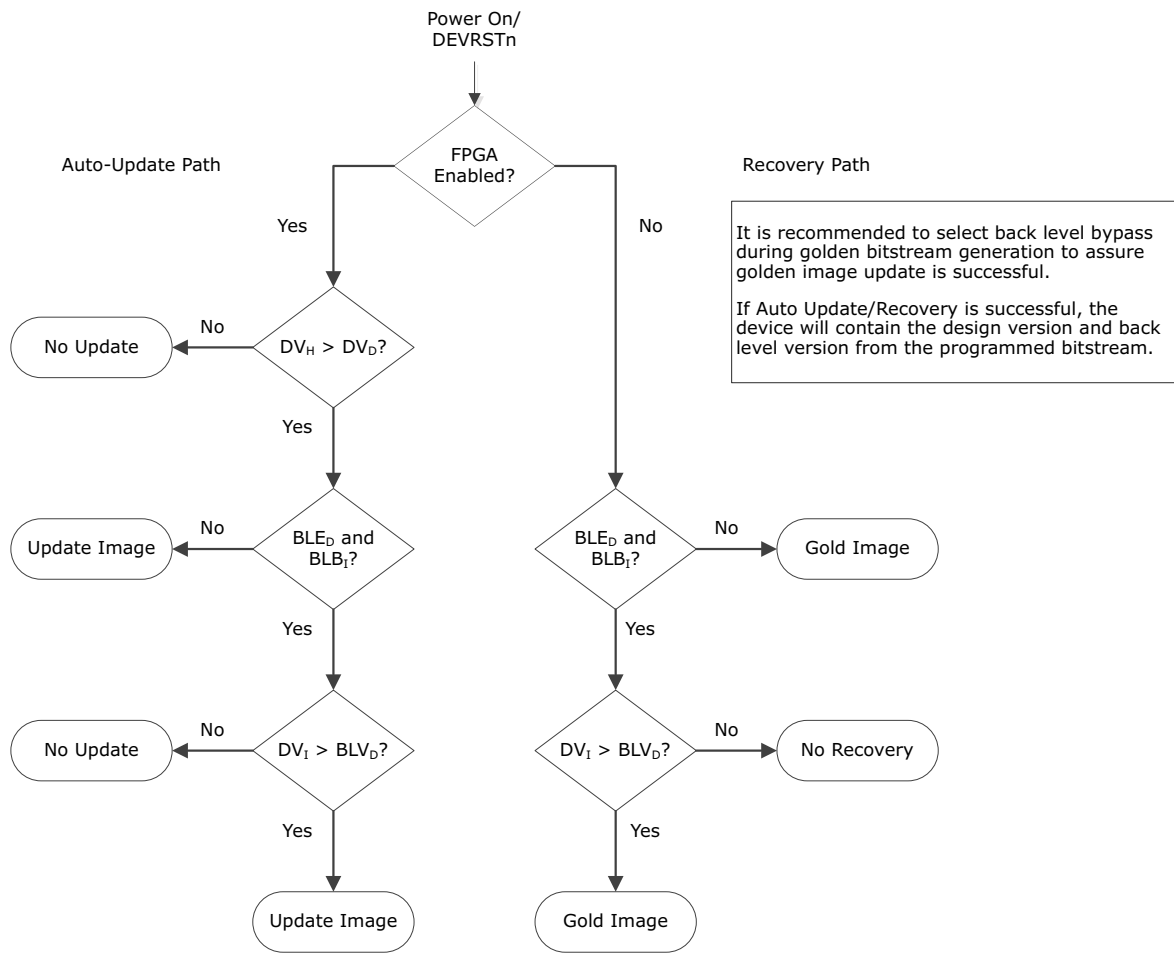
**Figure 22 • Back Level Protection**



**Note:** Back Level Bypass (if selected, design is programmed irrespective of back level version) should be set if you allow programming recover with recovery image lower than the back level version selected. Alternatively, you should update the design version of the recovery image so that it is always greater than the back level version.

The following flow chart explains how back level protection works during Auto Update and Programming Recovery mode.

**Figure 23 • Flow Chart of Back Level Protection during Auto-Update and Programming Recovery Mode**



$DV_H$  = Design Version in Header  
 $DV_D$  = Design Version in Device  
 $DV_I$  = Design Version in update bitstream  
 $BLV_D$  = Back-Level Version in Device  
 $BLB_I$  = Back-Level Bypass in update bitstream  
 $BLE_D$  = Back-Level Enable in Device



## 10 Production Programming

---

The SmartFusion2 and IGLOO2 devices can be programmed before board assembly using Silicon Sculptor 3. The SmartFusion2 or IGLOO2 device is inserted in an FPGA package specific adapter module, which is then plugged into Silicon Sculptor 3.

Microsemi provides Silicon Sculptor 3 software or Sculptw that is used to run the programmer. For more information, see the following web page: <http://www.microsemi.com/products/fpga-soc/design-resources/programming/silicon-sculptor-3>

SmartFusion2 and IGLOO2 devices can also be programmed by automatic test equipment (ATE). The Libero SoC software generates a serial vector format (SVF) file that is used for ATE programming. Support for SVF format will be added in a future release. Contact the [technical support](#) team if you need support for the SVF file format.

As part of production programming, serialization feature can be used. SmartFusion2 and IGLOO2 devices support serialization. For more information, see the Libero SoC or FlashPro Express Online Help.

# 11 State of SmartFusion2 and IGLOO2 Components During Programming

The following table lists the state of each ASIC block and I/O during the programming modes described in this user guide. You can configure the I/O state during JTAG programming in Libero SoC, see [Figure 24](#), page 46. For more information, see the [Libero SoC User Guide](#).

**Note:** I/O state of a blank device is tristated with weak pull up. During reset, the I/Os are floating even if pull up is enabled or I/O has a default pull up. However, for the SC\_SPI and JTAG pins, the pull up is active during reset because they are hardwired. I/O weak pull-up is around 10K.

**Table 26 • ASIC Block and I/O State During Programming**

Component	Programming Method		
	JTAG/SPI Slave	IAP/MSS ISP	Auto Programming
Cortex-M3 processor	Held in reset	Operational, running user firmware	Held in reset
MSS/HPMS (without Cortex-M3)	Active during programming to allow programming access to eNVM.	Active	Active during programming to allow programming access to eNVM.
MDDR controller	DDR I/O calibration block held in reset.	DDR I/O calibration block held in reset.	DDR I/O calibration block held in reset.
Shared I/O (fabric/MSS/HPMS) except SPI_0	I/Os are tristated with a weak pull-up by default during JTAG programming, but can be configured to user-selectable states in Libero SoC (see <a href="#">Figure 24</a> , page 46). During SPI slave programming of a blank device, I/Os are tristated with weak pull up. If the device is already programmed, then the I/O state will be according to the user-configured F*F state. <sup>1</sup>	The device has been programmed already once, therefore, I/O state will be according to the user-configured F*F state. <sup>1</sup>	Tristated with weak pull up.

**Table 26 • ASIC Block and I/O State During Programming (continued)**

Component	Programming Method		
	JTAG/SPI Slave	IAP/MSS ISP	Auto Programming
SPI_0	Same as above	Same as above for MSS ISP. For IAP, slave select pins (SS4, SS5, SS6, and SS7) of the SPI_0 port will drive high (except 005 and 010 devices). The I/O level and drive strength are based on the previous settings programmed into the device. All other SPI_0 I/O will be tristated with weak pull up.	Slave select pins (SS4, SS5, SS6, and SS7) of SPI_0 the port will drive high (except 005 and 010 devices). The I/O level will be the same as bank voltage used for SPI_0 bank and the drive strength will be maximum for this I/O level. For example, if the SPI_0 bank was powered by 1.8 V, then the drive strength will be the maximum value allowed for 1.8 V LVCMOS I/O. For the maximum drive strength allowed for LVCMOS 1.8 V transmitter, see the <a href="#">SmartFusion2 and IGLOO2 Datasheet</a> . All other SPI_0 I/O will be tristated with weak pull up.
Shared I/O (Fabric/MDDR)	Same as above	The device has been programmed already once, therefore, I/O state will be according to the user-configured F*F state. <sup>1</sup>	Tristated with weak pull up
Shared I/O (fabric/FDDR)			Tristated with weak pull up
Dedicated fabric I/O			Tristated with weak pull up
SERDES I/O	Unaffected by programming	Unaffected by programming	Unaffected by programming
FDDR block	Fabric interfaces to this block are gated off. So no transactions can occur at configuring APB, AHB/AXI interfaces to the fabric.	Fabric interfaces to this block are gated off. So no transactions can occur at configuring APB, AHB/AXI interfaces to the fabric.	Fabric interfaces to this block are gated off. So no transactions can occur at configuring APB, AHB/AXI interfaces to the fabric.
SERDESIF block	Fabric interfaces to this block are gated off. So no transactions can occur at configuring APB, AHB/AXI interfaces to the fabric.	Fabric interfaces to this block are gated off. So no transactions can occur at configuring APB, AHB/AXI interfaces to the fabric.	Fabric interfaces to this block are gated off. So no transactions can occur at configuring APB, AHB/AXI interfaces to the fabric.

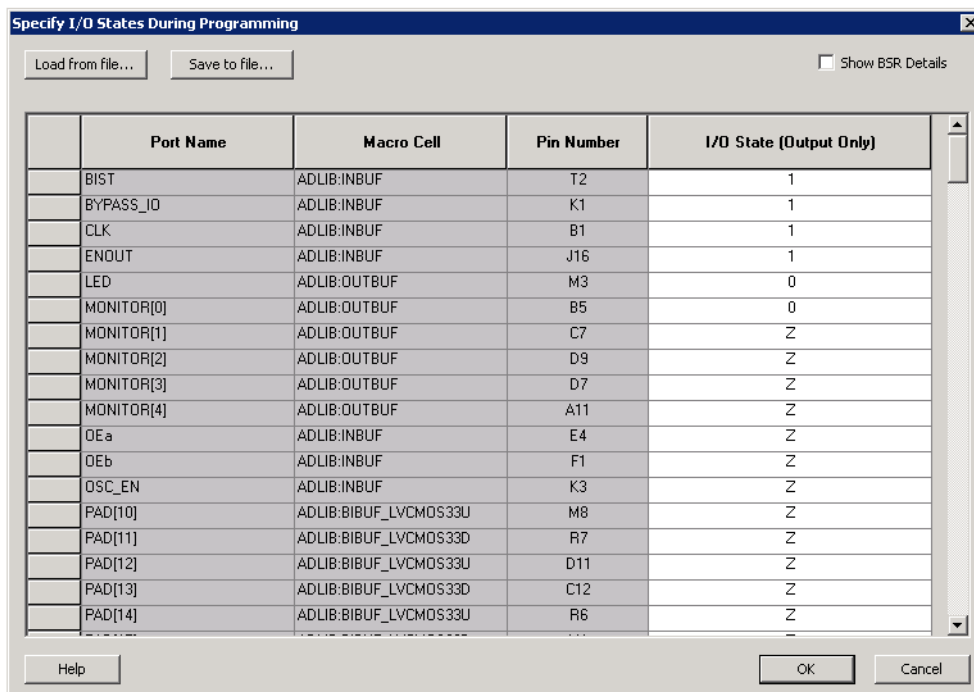
1. I/O state during JTAG programming can be configured differently, see [Figure 24](#), page 46. I/O states during SPI slave programming, MSS ISP, Auto Update, and IAP follow user-configured F\*F state (see [Use of Flash Freeze Mechanism in Device Programming](#), page 47).

The following table lists the state of each ASIC block and each I/O during programming recovery.

**Table 27 • ASIC Block and I/O State During Programming Recovery/Auto Update**

Components	Programming Recovery/Auto Update
Cortex-M3 processor	Held in reset.
MSS/HPMS (without Cortex-M3)	Active during programming to permit programming access to eNVM.
MDDR controller	DDR I/O calibration block held in reset.
Shared I/O (fabric/MSS/HPMS except SPI_0)	Tristated with weak pull up (during programming recovery). User configured I/O state in Flash Freeze mode (during Auto Update).
SPI_0	Slave select pins (SS4, SS5, SS6, and SS7) of the SPI_0 port will drive high (except 005 and 010 devices). The I/O level and drive strength are based on the previous settings programmed into the device. All other SPI_0 I/O will be tristated with weak pull up.
Shared I/O (fabric/MDDR)	Tristated with weak pull up (during programming recovery). User configured I/O state in Flash*Freeze mode (during Auto Update).
Shared I/O (fabric/FDDR)	Tristated with weak pull up (during programming recovery). User configured I/O state in Flash*Freeze mode (during Auto Update).
Dedicated fabric I/O	Tristated with weak pull up (during programming recovery). User configured I/O state in Flash*Freeze mode (during Auto Update).
SERDES I/O	Unaffected by programming.
FDDR block	Fabric interfaces to this block are gated off. So no transactions can occur at config APB, AHB/AXI interfaces to fabric.
SERDESIF block	Fabric interfaces to this block are gated off. So no transactions can occur at config APB, AHB/AXI interfaces to fabric.

**Figure 24 • I/O States During JTAG Programming**



Specify I/O States During Programming

Load from file... Save to file... ☐ Show BSR Details

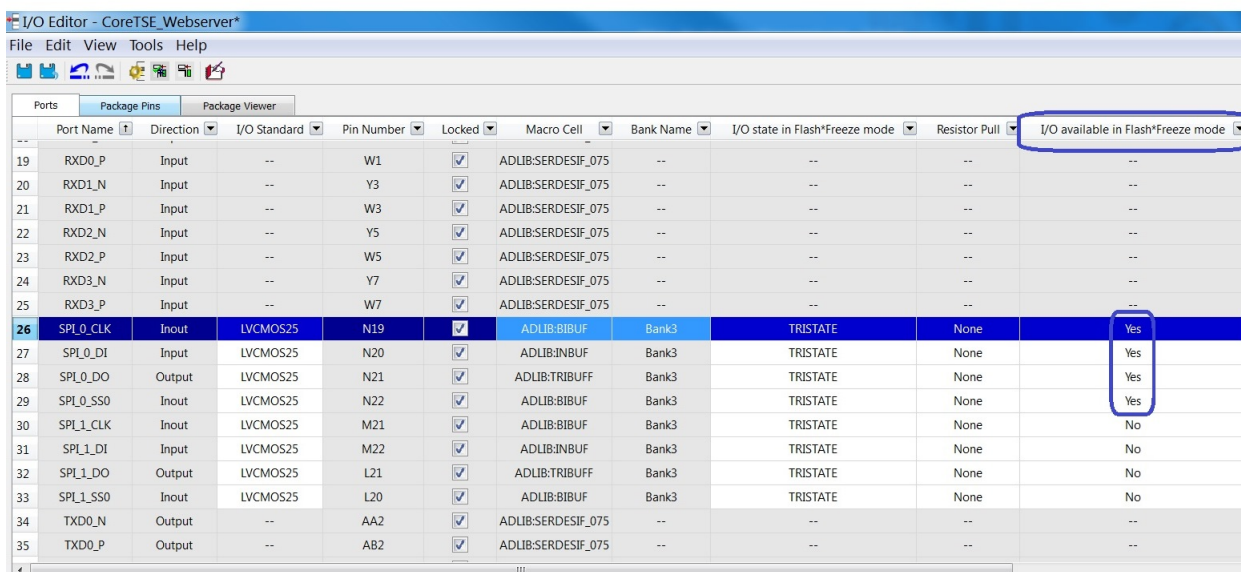
Port Name	Macro Cell	Pin Number	I/O State (Output Only)
BIST	ADLIB:INBUF	T2	1
BYPASS_IO	ADLIB:INBUF	K1	1
CLK	ADLIB:INBUF	B1	1
ENOUT	ADLIB:INBUF	J16	1
LED	ADLIB:OUTBUF	M3	0
MONITOR[0]	ADLIB:OUTBUF	B5	0
MONITOR[1]	ADLIB:OUTBUF	C7	Z
MONITOR[2]	ADLIB:OUTBUF	D9	Z
MONITOR[3]	ADLIB:OUTBUF	D7	Z
MONITOR[4]	ADLIB:OUTBUF	A11	Z
OEa	ADLIB:INBUF	E4	Z
OEb	ADLIB:INBUF	F1	Z
OSC_EN	ADLIB:INBUF	K3	Z
PAD[10]	ADLIB:BIBUF_LVCMOS33U	M8	Z
PAD[11]	ADLIB:BIBUF_LVCMOS33D	R7	Z
PAD[12]	ADLIB:BIBUF_LVCMOS33U	D11	Z
PAD[13]	ADLIB:BIBUF_LVCMOS33D	C12	Z
PAD[14]	ADLIB:BIBUF_LVCMOS33U	R6	Z

Help OK Cancel

## 11.1 Use of Flash Freeze Mechanism in Device Programming

The SPI slave, IAP, Auto Update, and MSS ISP (SmartFusion2 only) programming modes involve the device being put into F\*F mode. In this mode, the I/Os are disabled and the fabric is powered down. However, the I/O state can be set using the I/O Editor of the Libero SoC software, as shown in the following figure. There are two options to select—Tristate and Last known value. Tristate option is actually implemented with weak pull up and Last known value is implemented with weak pull up or down. For example, if the IO is driving high before programming, then it will be tristated with weak pull up during programming when Last Known state is selected for that IO. Similarly, if the IO was driving low, it will be tristated with weak pull down. As part of the entry into Flash Freeze, the system controller switches the clock source of the MSS/HPMS over to the standby clock (options are: 1MHz or 50MHz MSS clock source in the Configure Flash Freeze menu in Libero SoC). To ensure that the MSS is clocked at a consistent speed during IAP/MSS ISP in SmartFusion2, the Cortex-M3 firmware must manually switch the MSS clock source to 50 MHz (if 1MHz is selected in the Configure Flash Freeze menu in Libero SoC) before requesting the IAP/MSS ISP through a system service request.

**Figure 25 • Setting I/O States**



Port Name	Direction	I/O Standard	Pin Number	Locked	Macro Cell	Bank Name	I/O state in Flash*Freeze mode	Resistor Pull	I/O available in Flash*Freeze mode
19 RXD0_P	Input	--	W1	<input checked="" type="checkbox"/>	ADLIB:SERDESIF_075	--	--	--	--
20 RXD1_N	Input	--	Y3	<input checked="" type="checkbox"/>	ADLIB:SERDESIF_075	--	--	--	--
21 RXD1_P	Input	--	W3	<input checked="" type="checkbox"/>	ADLIB:SERDESIF_075	--	--	--	--
22 RXD2_N	Input	--	Y5	<input checked="" type="checkbox"/>	ADLIB:SERDESIF_075	--	--	--	--
23 RXD2_P	Input	--	W5	<input checked="" type="checkbox"/>	ADLIB:SERDESIF_075	--	--	--	--
24 RXD3_N	Input	--	Y7	<input checked="" type="checkbox"/>	ADLIB:SERDESIF_075	--	--	--	--
25 RXD3_P	Input	--	W7	<input checked="" type="checkbox"/>	ADLIB:SERDESIF_075	--	--	--	--
26 SPI0_CLK	Inout	LVC MOS25	N19	<input checked="" type="checkbox"/>	ADLIB:BIBUF	Bank3	TRISTATE	None	Yes
27 SPI0_DI	Input	LVC MOS25	N20	<input checked="" type="checkbox"/>	ADLIB:INBUF	Bank3	TRISTATE	None	Yes
28 SPI0_DO	Output	LVC MOS25	N21	<input checked="" type="checkbox"/>	ADLIB:TRIBUFF	Bank3	TRISTATE	None	Yes
29 SPI0_SS0	Inout	LVC MOS25	N22	<input checked="" type="checkbox"/>	ADLIB:BIBUF	Bank3	TRISTATE	None	Yes
30 SPI1_CLK	Inout	LVC MOS25	M21	<input checked="" type="checkbox"/>	ADLIB:BIBUF	Bank3	TRISTATE	None	No
31 SPI1_DI	Input	LVC MOS25	M22	<input checked="" type="checkbox"/>	ADLIB:INBUF	Bank3	TRISTATE	None	No
32 SPI1_DO	Output	LVC MOS25	L21	<input checked="" type="checkbox"/>	ADLIB:TRIBUFF	Bank3	TRISTATE	None	No
33 SPI1_SS0	Inout	LVC MOS25	L20	<input checked="" type="checkbox"/>	ADLIB:BIBUF	Bank3	TRISTATE	None	No
34 TXD0_N	Output	--	AA2	<input checked="" type="checkbox"/>	ADLIB:SERDESIF_075	--	--	--	--
35 TXD0_P	Output	--	AB2	<input checked="" type="checkbox"/>	ADLIB:SERDESIF_075	--	--	--	--

The system controller, in suspend mode or avionics mode, is held in a reset state and cannot provide system services such as F\*F, security, or IAP programming. The device can only program using the JTAG port. For more information on suspend mode, see the *SmartFusion2 System Controller User Guide*.