

UG0450

User Guide

**SmartFusion2 SoC and IGLOO2 FPGA System
Controller**





a  MICROCHIP company

Microsemi Headquarters

One Enterprise, Aliso Viejo,
CA 92656 USA

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

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

Sales: +1 (949) 380-6136

Fax: +1 (949) 215-4996

Email: sales.support@microsemi.com

www.microsemi.com

©2021 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.

Contents

1	Revision History	1
1.1	Revision 6.0	1
1.2	Revision 5.0	1
1.3	Revision 4.0	1
1.4	Revision 3.0	1
1.5	Revision 2.0	1
1.6	Revision 1.0	1
1.7	Revision 0.0	2
2	System Controller 3	
2.1	Introduction	3
2.2	Functional Description	4
2.2.1	Subsystems	5
2.2.2	Interfaces	5
2.2.3	System IP Interface (SII) Master	5
2.2.4	Communication Block (COMM_BLK)	5
2.2.5	Oscillator Control	6
2.2.6	Random Number Generator	6
2.2.7	Cryptographic Services	6
2.2.8	JTAG	6
2.2.9	User JTAG	8
2.2.10	Dedicated Programming SPI Peripheral	10
2.2.11	Device Reset	10
2.2.12	USI Interface	10
2.2.13	Clock Requirements	11
2.2.14	System Controller Suspend Mode	11
3	System Services	14
3.1	Introduction	14
3.2	Device and Design Information Services	18
3.2.1	Serial Number Service	18
3.2.2	USERCODE Service	18
3.2.3	Device Certificate Service	19
3.2.4	User Design Version Service	19
3.3	Flash*Freeze Service	20
3.4	Cryptographic Services	21
3.4.1	AES Services	21
3.4.2	SHA-256 Services	23
3.5	DPA-Resistant Key-Tree Services	25
3.5.1	Key-Tree Cryptographic Service	26
3.5.2	Challenge-Response Cryptographic Service	27
3.6	Elliptic Curve Cryptography Services	27
3.6.1	ECC Point Multiplication Service	28
3.6.2	ECC Point Addition Service	29
3.7	SRAM-PUF Services	30
3.7.1	Create User Activation Code	30
3.7.2	Key Generation and Enrollment Services	32
3.7.3	Key Reconstruction	36
3.7.4	User-Enrolled SRAM-PUF Keys for Design Security	37
3.7.5	SRAM-PUF Based True Random Number Seed Generation	38

3.8	Non-Deterministic Random Bit Generator (NRBG) Services	40
3.8.1	Self Test Service	41
3.8.2	Instantiate Service	41
3.8.3	Generate Service	42
3.8.4	Uninstantiate Service	44
3.8.5	Reset Service	44
3.9	Zeroization Service	44
3.9.1	FPGA Fabric Configuration NVM	45
3.9.2	User Security Keys and Settings	45
3.9.3	Factory Security Keys and Configuration Settings	45
3.9.4	System Controller Memory	45
3.9.5	Digital Data Path	45
3.9.6	Fabric Registers	45
3.9.7	Fabric SRAM	45
3.9.8	HPMS SRAM	45
3.9.9	eNVM Memory Array and eNVM Registers	45
3.10	Programming Service	46
3.10.1	IAP Service	47
3.10.2	ISP Service	48
3.11	NVM Data Integrity Check Service	49
3.12	Unrecognized Command Response	50
3.13	Asynchronous Messages	50
3.13.1	Power-on-Reset (POR) Digest Error	50
3.14	How to Use System Services in SmartFusion2	51
3.14.1	Use Model 1: Fetching Device and Design Information	52
3.15	How to Use System Services in IGLOO2	55
3.15.1	Configuring System Services	57
3.15.2	Fetching Device and Design information	60
3.15.3	Related Applications	63

Figures

Figure 1	SmartFusion2 - System Controller Interfacing with AHB Bus Matrix	3
Figure 2	IGLOO2 - System Controller Interfacing with AHB Bus Matrix	4
Figure 3	SmartFusion2 - Interfacing of the System Controller with MSS and FPGA Fabric	4
Figure 4	IGLOO2 - Interfacing of the System Controller with HPMS and FPGA Fabric	5
Figure 5	TAP Controller State Machine	7
Figure 6	UJTAG Macro	8
Figure 7	UJTAG Usage Example in Test and Debug Applications	10
Figure 8	FLASH_FREEZE Macro	11
Figure 9	Enabling System Controller Suspended Mode in New Project Window	12
Figure 10	Enabling System Services Suspend mode in Project Settings Window	13
Figure 11	System Services Sample Projects	51
Figure 12	Functional Block Diagram for Accessing System Services	55
Figure 13	System Builder Window	56
Figure 14	System Builder - Device Features Tab	57
Figure 15	CoreSysServices IP to COMM_BLK Path	58
Figure 16	System Builder - Clock Tab	58
Figure 17	System Builder - Memory Map Tab	59
Figure 18	SmartDesign Connections	60
Figure 19	CORESYSSERVICES_0 Configuration	61
Figure 20	HPMS Subsystem and CoreSysServices IP Connections	61

Tables

Table 1	Boundary Scan Opcodes	8
Table 2	UJTAG Port Description	9
Table 3	SmartFusion2 and IGLOO2 System Services	15
Table 4	Device and Design Information Services Status	18
Table 5	Serial Number Service Request	18
Table 6	Serial Number Service Response	18
Table 7	USERCODE Service Request	18
Table 8	USERCODE Service Response	18
Table 9	Device Certificate Service Request	19
Table 10	Device Certificate Service Response	19
Table 11	Design Version Service Request	19
Table 12	Design Version Service Response	19
Table 13	Flash*Freeze Request	20
Table 14	FFOPTIONS	20
Table 15	Flash*Freeze Service Response	20
Table 16	Flash*Freeze Service Status	20
Table 17	Cryptographic Services Status Codes	21
Table 18	128-bit AES Service Request	22
Table 19	AES128DATA Descriptor	22
Table 20	MODE Parameter	22
Table 21	OPMODE Parameter	22
Table 22	128-bit AES Service Response	22
Table 23	256-bit AES Service Request	23
Table 24	AES256DATA Descriptor	23
Table 25	256-bit AES Service Response	23
Table 26	SHA-256 Service Request	24
Table 27	SHA256DATA Structure	24
Table 28	SHA-256 Service Response	24
Table 29	HMAC Service Request	24
Table 30	HMACDATA Structure	24
Table 31	HMAC Service Response	25
Table 32	DPA-Resistant Key-Tree Services Status Codes	26
Table 33	Key-Tree Service Request	26
Table 34	KEYTREEDATA Structure	26
Table 35	KeyTree Service Response	26
Table 36	Challenge-Response Service Request	27
Table 37	CHRESP Structure	27
Table 38	Challenge-Response Service Response	27
Table 39	Challenge-Response Service Response	28
Table 40	Elliptic Curve Point Multiplication Service Request	29
Table 41	ECCPMULT Structure	29
Table 42	ECCPMULT Structure	29
Table 43	Elliptic Curve Point Addition Service Request	29
Table 44	ECCPADD Structure	30
Table 45	Elliptic Curve Point Addition Service Response	30
Table 46	PUFUSERAC Service Request	31
Table 47	PUFUSERAC Structure	31
Table 48	Elliptic Curve Point Addition Service Response	31
Table 49	PUFUSERAC Status	31
Table 50	PUFUSERKC Service Request	33
Table 51	PUFUSERKC Service Request	33
Table 52	PUFUSERACKCEXPORT Memory Layout	34
Table 53	PUFUSERACKCIMPORT	35
Table 54	PUFUSERKC Service Response	35

Table 55	PUFUSERKC Service Response	35
Table 56	PUFUSERKEY Service Request	36
Table 57	PUFUSERKEY Structure	36
Table 58	PUFUSERKC Service Response	37
Table 59	USERPUFKEY Status	37
Table 60	PUFPUBLICKEY Service Request	37
Table 61	PUFPUBLICKEY Structure	38
Table 62	PUFPUBLICKEY Structure	38
Table 63	USERPUFKEY Status	38
Table 64	PUFPUBLICKEY Structure	38
Table 65	PUFPUBLICKEY Structure	39
Table 66	USERPUFKEY Status	39
Table 67	PUFSEED Status	39
Table 68	NRBG Service Response Status Codes	40
Table 69	DRBG Self Test Request	41
Table 70	DRBG Self Test Response	41
Table 71	DRBG Instantiate Request	41
Table 72	DRBGINstantiate Structure	42
Table 73	DRBG Instantiate Service Response	42
Table 74	DRBG Generate Request	42
Table 75	DRBGGENERATE Structure	42
Table 76	DRBG Reseed Request	43
Table 77	DRBGRESEED Structure	43
Table 78	DRBG Reseed Service Response	43
Table 79	DRBG Generate Service Response	43
Table 80	DRBG Uninstantiate Request	44
Table 81	Uninstantiate Response	44
Table 82	DRBG Reset Request	44
Table 83	DRBG Reset Response	44
Table 84	Zeroization Request	44
Table 85	Zeroization Configuration Options	45
Table 86	Programming Service Status Codes	46
Table 87	Autherrcode	46
Table 88	Errorcode	47
Table 89	IAP Programming Service Request	47
Table 90	OPTIONS	47
Table 91	MODE	48
Table 92	IAP Response	48
Table 93	ISP Programming Service Request	48
Table 94	ISP Response	49
Table 95	NVM Data Integrity Check Service Request	49
Table 96	OPTIONS	49
Table 97	NVM Data Integrity Check Response	49
Table 98	DIGESTERR	50
Table 99	Unrecognized Command Message	50
Table 100	POR Digest Error Message	50
Table 101	System Services APIs for Fetching Device and Design Information	52
Table 102	User Interface Signals	62
Table 103	Command Codes for Device and Design Information Services	62

1 Revision History

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

1.1 Revision 6.0

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

- Updated [System Controller Suspend Mode](#), page 11.
- Updated [Figure 5](#).

1.2 Revision 5.0

Updated [Dedicated Programming SPI Peripheral](#), page 10 and [System Controller Suspend Mode](#), page 11.

1.3 Revision 4.0

The following is a summary of the changes in revision 4.0 of this document.

- Added the [Related Applications](#), page 63.
- Updated the [Key-Tree Cryptographic Service](#), page 26.
- Updated the [Challenge-Response Cryptographic Service](#), page 27.
- Added the [Elliptic Curve Cryptography Services](#), page 27.
- Added the [SRAM-PUF Services](#), page 30.
- Updated the [Non-Deterministic Random Bit Generator \(NRBG\) Services](#), page 40.
- Added a note in [Flash*Freeze Service](#), page 20 about deep-power-down action.
- Updated [Table 99](#), page 50 for command change.
- Updated the references as per the standard.

1.4 Revision 3.0

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

- Updated value for OPMODE 3 in [Table 21](#), page 22.
- Updated [Reseed Service](#), page 43.
- Updated [System Controller Suspend Mode](#), page 11.

1.5 Revision 2.0

The following is a summary of the changes in revision 2.0 of this document.

- Updated [Table 3](#), page 15.
- Updated [DPA-Resistant Key-Tree Services](#), page 25.
- Updated [Non-Deterministic Random Bit Generator \(NRBG\) Services](#), page 40.
- Updated the [Instantiate Service](#), page 41.
- Updated the [Generate Service](#), page 42.
- Updated the [Reseed Service](#), page 43.
- Added [Table 87](#), page 46, [Table 88](#), page 47.
- Updated [Figure 2](#), page 4 and [Figure 4](#), page 5.
- Updated [SYSRESET](#), page 10.
- Updated [System Controller Suspend Mode](#), page 11.
- Added [Clock Requirements](#), page 11.

1.6 Revision 1.0

The following is a summary of the changes in revision 1.0 of this document.

- Restructured the user guide.
- Updated [Table 21](#), page 22, [Table 56](#), page 36, [Table 85](#), page 45.

- Added How to Use System Services in IGLOO2, page 54 section.
- Updated [Figure 2](#), page 4.
- Added [System Controller Suspend Mode](#), page 11.

1.7 **Revision 0.0**

Revision 0.0 was the first publication of this document.

2 System Controller

2.1 Introduction

This chapter discusses the subsystems and interfaces in the SmartFusion2 and IGLOO2 System Controller. The System Controller manages the programming of the device and handles system service requests. The System Controller serves as the base on which the following system services (described in the [System Services](#), page 14) are made available with SmartFusion2 and IGLOO2 devices:

- Device and design information services
- Flash*Freeze services
- Cryptographic services
- DPA-resistant key tree services
- Non-deterministic random bit generator services
- Zeroization service
- Programming services

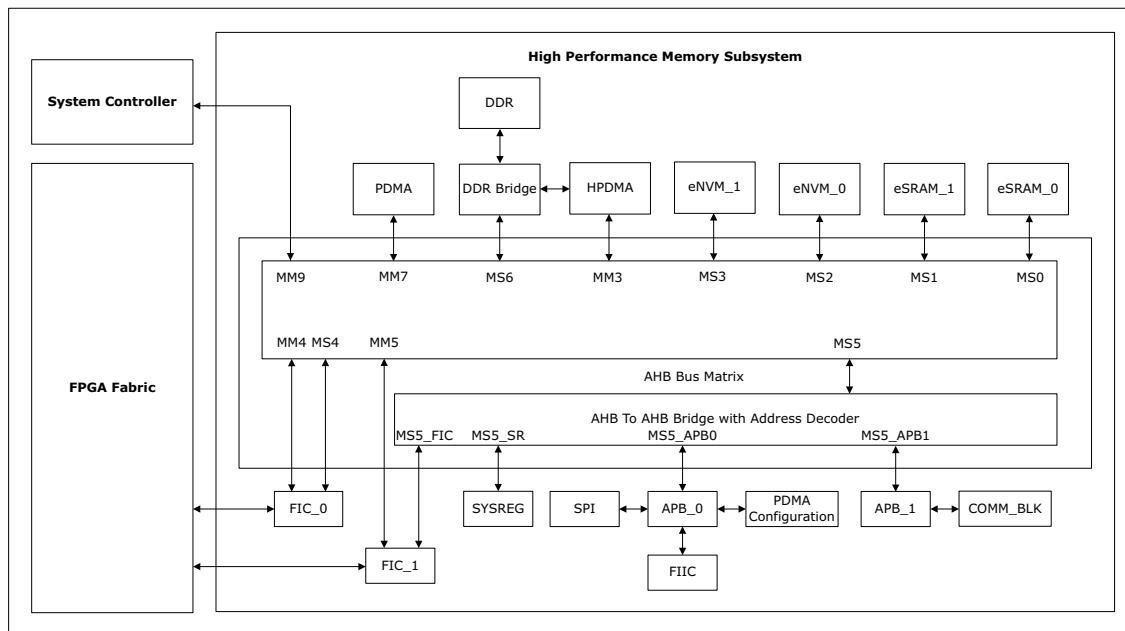
The following figure shows the connectivity of the SmartFusion2 System Controller to the AHB bus matrix.

Figure 1 • SmartFusion2 - System Controller Interfacing with AHB Bus Matrix



The following figure shows the connectivity of the IGLOO2 System Controller to the AHB bus matrix.

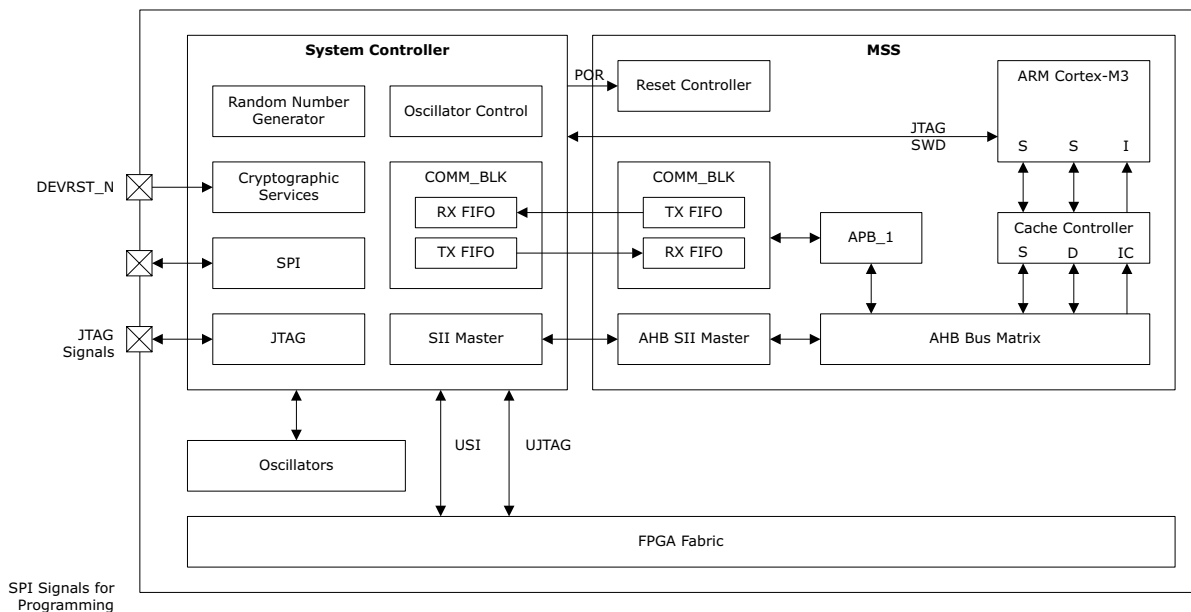
Figure 2 • IGLOO2 - System Controller Interfacing with AHB Bus Matrix



2.2 Functional Description

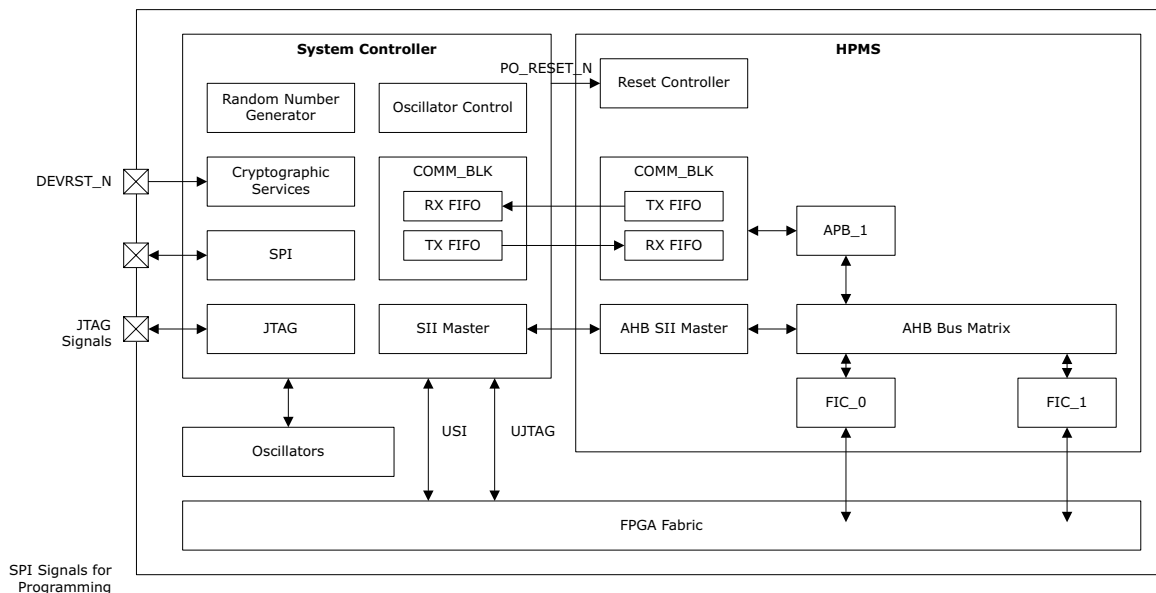
The following figure shows the interfacing of the SmartFusion2 System Controller with MSS and the FPGA fabric.

Figure 3 • SmartFusion2 - Interfacing of the System Controller with MSS and FPGA Fabric



The following figure shows the interfacing of the IGLOO2 System Controller with HPMS and the FPGA fabric.

Figure 4 • IGLOO2 - Interfacing of the System Controller with HPMS and FPGA Fabric



The SmartFusion2 and IGLOO2 System Controllers consist of the following subsystems and interfaces, as shown in Figure 3, page 4 and Figure 4, page 5.

2.2.1 Subsystems

- System IP Interface (SII) Master
- Communication Block (COMM_BLK)
- Oscillator Control
- Random Number Generator
- Cryptographic Services
- JTAG
- Dedicated Programming SPI Peripheral

2.2.2 Interfaces

- DEVRST_N
- SPI signals for programming
- JTAG signals
- User JTAG
- User services interface (USI)

The following sections provide short description of these sub-systems and interfaces.

2.2.3 System IP Interface (SII) Master

The system IP interface (SII) master connects the System Controller with all the internal elements. It is used to transfer data to and from the MSS or HPMS memory space by the System Controller for system services. It is also used for factory tests but not available for customer.

2.2.4 Communication Block (COMM_BLK)

The communication block (COMM_BLK) provides a bidirectional message passing facility between the ARM® Cortex®-M3 processor/Fabric master and the System Controller. It is similar to a mailbox communication channel that allows message bytes to be passed from the Fabric master to the System Controller and vice versa. For more information, refer to the “Communication Block” chapter in the [UG0448: IGLOO2 High Performance Memory Subsystem User Guide](#) and [UG0331: SmartFusion2 Microcontroller Subsystem User Guide](#). The COMM_BLK is used to call system services. System

services can be implemented using API functions. For more information, refer to the [System Services](#), page 14.

2.2.5 Oscillator Control

The oscillator control block manages the on-chip RC oscillators and crystal oscillators. It performs oscillator initialization and control during device start-up. The on-chip oscillators are automatically disabled if the users do not configure the blocks (System Controller, MSS/HPMS, and FPGA fabric logic) that are required for them. If the subsystem is not configured for use, the oscillator is turned off synchronously without generating runt clock pulses. The 50 MHz RC oscillator is the default clock source; it is enabled after power-on reset. For more information about how to set up RC oscillators and the main crystal oscillator as clock sources, refer to the “Oscillator Configuration” section in the [UG0449: SmartFusion2 and IGLOO2 Clocking Resources User Guide](#).

2.2.6 Random Number Generator

The System Controller contains a random number generator block which is available for user cryptographic services. The Deterministic Random Bit Generator (DRBG) is implemented as defined in National Institute of Standards and Technology (NIST) Special Publication 800-90. It provides the following features:

- Designed to support AIS-31 random number generation requirement.
- Uses AES-256 CTR mode per NIST SP800-90 for the DRBG implementation.
- Built-in hardware tests for auto correlation and continuous random number generation testing (CRNGT).

For more information, refer “Non-Deterministic Random Bit Generator (NRBG)” section in the [UG0443: SmartFusion2 and IGLOO2 FPGA Security Best Practices User Guide](#).

2.2.7 Cryptographic Services

The System Controller contains an AES encryption and decryption engine which can be dynamically configured for key lengths of 128 or 256 bits. A NIST-approved SHA-256 authentication algorithm is also provided. For more information, refer to the “Other Cryptographic Services” section in the [UG0443: SmartFusion2 and IGLOO2 FPGA Security Best Practices User Guide](#).

2.2.8 JTAG

The System Controller implements the functionality of a JTAG slave, with IEEE 1532 support, which also implies IEEE 1149.1 compliance. JTAG communicates with the System Controller using a Command register that conveys the JTAG instruction to be executed and a 128-bit data I/O buffer that transfers any associated data. The JTAG interface is used for the following operations:

- In-system programming (ISP)
- For more information, refer [UG0451: SmartFusion2 and IGLOO2 Programming User Guide](#).
- Serial wire JTAG debug port (SWJ-DP)/Serial Wire Debug (SWD) for the Cortex-M3 processor.
- For more information, refer to the “Debug Port” section in the [UG0331: SmartFusion2 Microcontroller Subsystem User Guide](#).
- UJTAG
- Boundary scan

2.2.8.1 Boundary Scan

IEEE Standard 1149.1 defines a hardware architecture and the set of mechanisms for boundary scan testing. JTAG operations are used during boundary scan testing. The basic boundary scan logic circuit is composed of the TAP controller, test data registers, and instruction register.

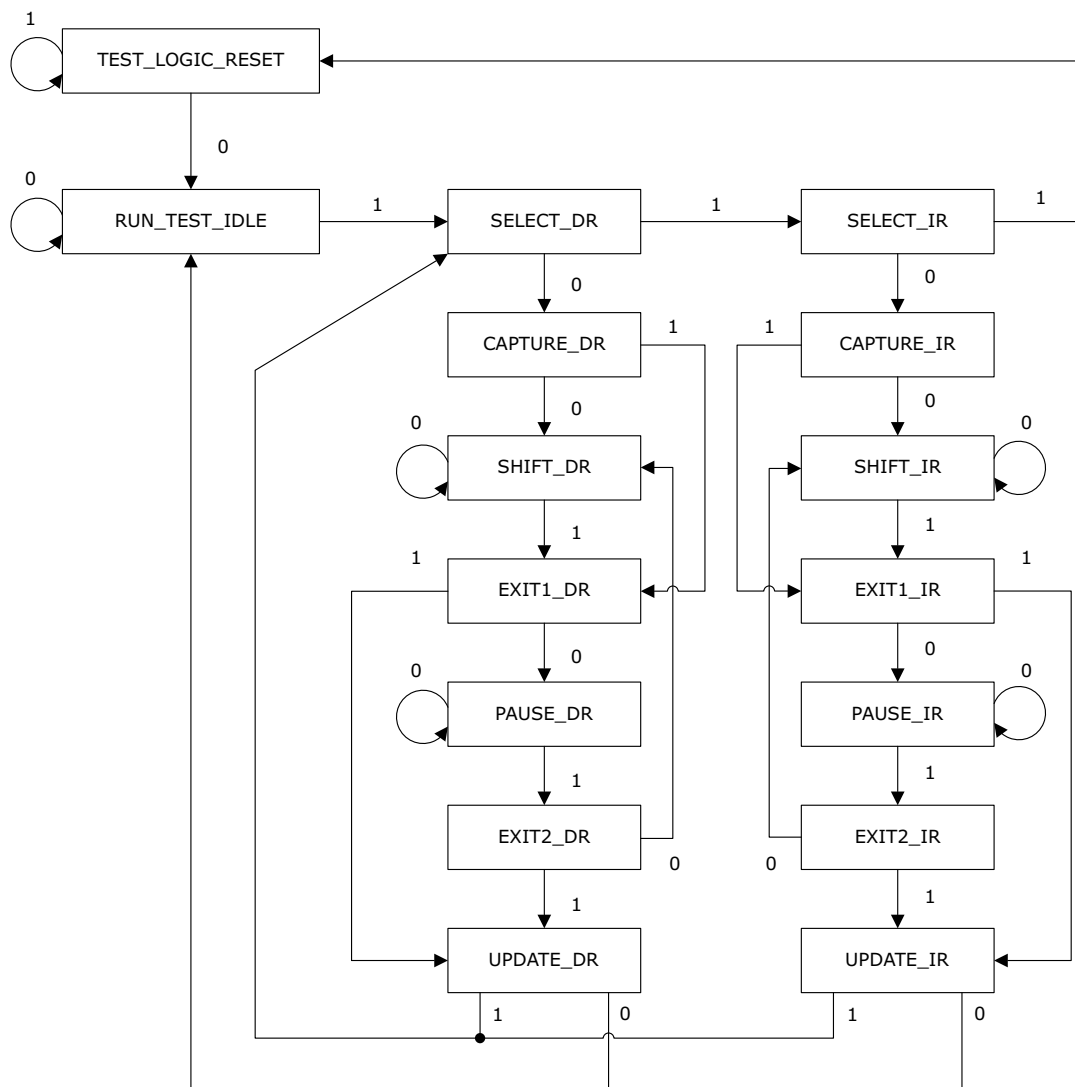
SmartFusion2 and IGLOO2 devices support three types of test data registers: bypass, device identification, and boundary scan. The bypass register is selected when no other register needs to be accessed in a device. This speeds up test data transfer to other devices in a test data path. The 32-bit device identification register is a shift register with four fields (LSB, ID number, part number, and version). The boundary scan register observes and controls the state of each I/O pin, except SERDES I/Os. Each I/O cell has three boundary scan register cells, each with serial-in, serial-out, parallel-in, and parallel-out pins.

2.2.8.1.1 TAP Controller State Machine

The TAP controller is a 4-bit state machine (16 states) that operates as shown in Figure 5. The 1s and 0s represent the values that must be present on TMS at a rising edge of TCK for the given state transition to occur. IR and DR indicate that the instruction register or the data register is operating in that state.

The TAP controller receives two control inputs (TMS and TCK) and generates control and clock signals for the rest of the test logic architecture. On power-up, the TAP controller enters the Test-Logic-Reset state. To guarantee a reset of the controller from any of the possible states, TMS must remain High for five TCK cycles. The TRST pin can also be used to asynchronously place the TAP controller in the Test Logic-Reset state.

Figure 5 • TAP Controller State Machine



2.2.8.2 Boundary Scan Opcodes

SmartFusion2 and IGLOO2 devices support all mandatory IEEE 1149.1 instructions (EXTEST, SAMPLE/PRELOAD, and BYPASS) and the optional IDCODE instruction, as listed in the following table.

Table 1 • Boundary Scan Opcodes

Instruction	Hex Opcode
EXTEST	00
HIGHZ	07
USERCODE	0E
SAMPLE/PRELOAD	01
IDCODE	0F
CLAMP	05
BYPASS	FF

For more information about pin descriptions and board-level recommendations, refer to the “JTAG I/O” section in the [UG0445: IGLOO2 FPGA and SmartFusion2 SoC FPGA Fabric User Guide](#).

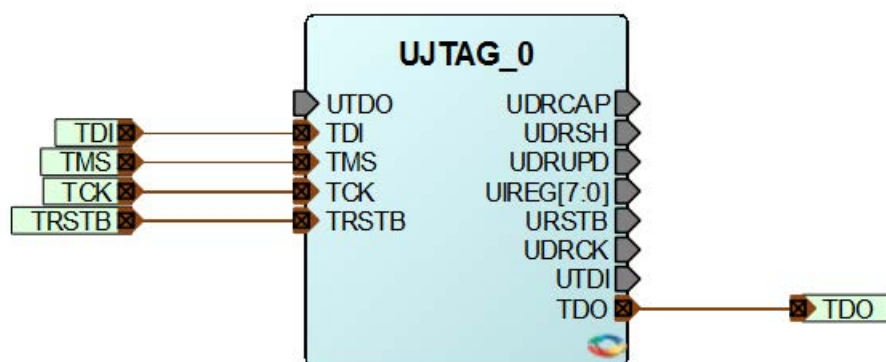
2.2.9 User JTAG

The user JTAG (UJTAG) interface is an extension to the external JTAG port of SmartFusion2 and IGLOO2 devices, controlled by the TAP controller when it is not performing JTAG programming. It can be used to shift in and shift out data/OPCODEs to and from the internal logic. The UJTAG functionality is available by instantiating the UJTAG macro from the Libero® System-on-Chip (SoC) IP catalog in SmartDesign or by instantiating it directly inside an HDL file. Using the UJTAG macro in a design enables real-time updating and monitoring of the internal behavior of FPGA fabric. For more information about UJTAG macro to shift in and shift out data, refer [Flash UJTAG Application Note](#).

2.2.9.1 UJTAG Macro

A block symbol of the UJTAG macro is presented in the following figure. The TDI, TMS, TCK, TRSTB, and TDO ports of the UJTAG macro are directly connected to the JTAG TAP controller and all other ports are accessible by the FPGA fabric.

Figure 6 • UJTAG Macro



The following table lists the descriptions of the ports that can be accessed by the FPGA fabric.

Table 2 • UJTAG Port Description

Port	Direction	Polarity	Description
UIREG [7:0]	Output	–	This 8-bit bus carries the contents of the JTAG instruction register of each device. Instruction register values 16 to 127 are not reserved and can be employed as user-defined instructions.
URSTB	Output	Low	URSTB is an Active Low signal and is asserted when the TAP controller is in Test-Logic-Reset mode. URSTB is asserted at power-up, and a power-on reset signal resets the TAP controller. URSTB is asserted until an external TAP access changes the TAP controller state.
UTDI	Output	–	This port is directly connected to the TAP's TDI signal.
UTDO	Input	–	This port is the user TDO output. Inputs to the UTDO port are sent to the TAP TDO output MUX when the IR address is in user range.
UDRSH	Output	High	Active High signal enabled in the Shift_DR TAP state
UDRCAP	Output	High	Active High signal enabled in the Capture_DR TAP state
UDRCK	Output	–	This port is directly connected to the TAP's TCK signal.
UDRUPD	Output	High	Active High signal enabled in the Update_DR TAP state

2.2.9.2 UJTAG Operation

Understanding a few basic functions of the UJTAG macro is necessary before designing with it. The fundamental concept of the UJTAG design is its connection with the TAP controller state machine. For more information, refer to the [TAP Controller State Machine](#), page 7.

UIREG [7:0] holds the contents of the JTAG instruction register. The UIREG vector value is updated when the TAP controller state machine enters the Update_IR state. Instructions 16 to 127 are user-defined and can be employed to encode multiple applications and commands within an application. Loading new instructions into the UIREG vector requires sending appropriate logic to TMS to put the TAP controller in a full IR cycle starting from the Select IR_Scan state and ending with the Update_IR state.

UTDI, UTDO, and UDRCK are directly connected to the JTAG TDI, TDO, and TCK ports, respectively. The TDI input can be used to provide either data (TAP controller in the Shift_DR state) or the new contents of the instruction register (TAP controller in the Shift_IR state).

UDRSH, UDRUPD, and UDRCAP are High when the TAP controller state machine is in the Shift_DR, Update_DR, and Capture_DR states. Therefore, they act as flags to indicate the stages of the data shift process. These flags are useful for applications in which blocks of data are shifted into the design from JTAG pins. For example, an active UDRSH can indicate that UTDI contains the data bitstream, and UDRUPD is a candidate for the end-of-data-stream flag.

2.2.9.3 Typical UJTAG Applications

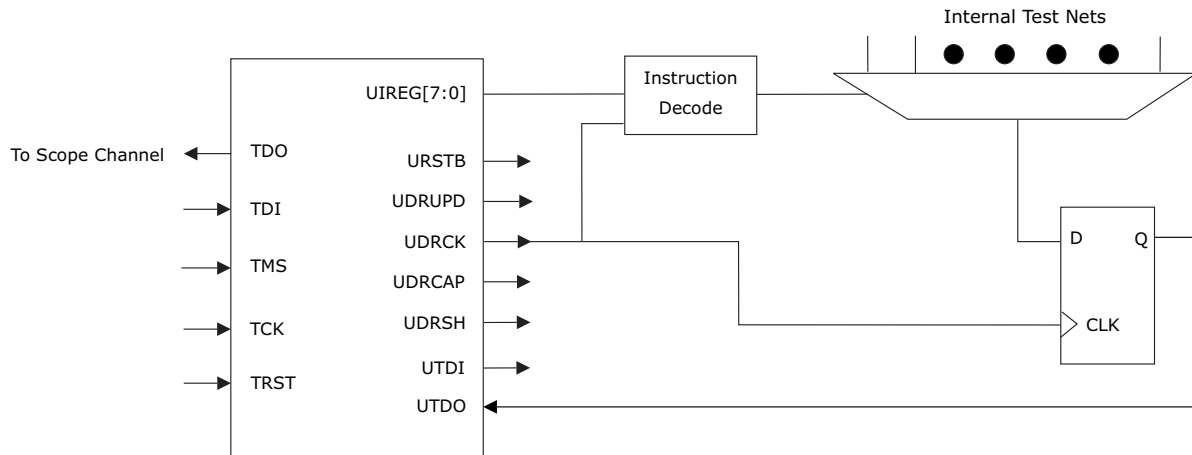
Bi-directional access to the JTAG port from the FPGA fabric—without putting the device into test mode—creates flexibility for implementing a variety of applications. This section describes one of these. This is based on importing/exporting data through the UJTAG macro. However, the possible applications are not limited to what is presented in this section. UJTAG can serve different purposes in many designs as an elementary or auxiliary part of the design.

2.2.9.3.1 Silicon Testing and Debugging

In many applications, the design needs to be tested, debugged, and verified on real silicon or in the final embedded application. To debug and test the functionality of designs, it is necessary to monitor some internal logic (or nets) during device operation. The approach of adding design test pins to monitor the critical internal signals has many disadvantages, such as limiting the number of users I/Os. Furthermore, adding external I/Os for test purposes may require an additional or dedicated board area for testing and debugging.

The UJTAG macro provides a flexible and cost-effective solution for silicon test and debug applications. In this solution, the signals under test are shifted out to the TDO pin of the TAP controller. The main advantage is that all the test signals are monitored from the TDO pin; no pins or additional board-level resources are required. The following figure illustrates this technique.

Figure 7 • UJTAG Usage Example in Test and Debug Applications



Multiple test nets are brought into an internal MUX architecture. The selection of the MUX is done using the contents of the TAP controller instruction register, where individual instructions (values from 16 to 127) correspond to different signals under test. The selected test signal can be synchronized with the rising or falling edge of TCK (optional) and sent out to UTDO to drive the TDO output of JTAG.

2.2.9.4 How to Use UJTAG

Refer to the [How To Use UJTAG](#) application note.

2.2.10 Dedicated Programming SPI Peripheral

The System Controller contains an SPI block that is dedicated for programming. For more information, refer to the “SPI-slave Programming” chapter in the [UG0451: SmartFusion2 and IGLOO2 Programming User Guide](#).

2.2.11 Device Reset

An input-only reset pad (DEV_RST_N) is present on every device, which allows assertion of a full reset to the chip at any time.

For more information, refer to the “Reset Controller” chapter in the [UG0448: IGLOO2 FPGA High Performance Memory Subsystem User Guide](#).

2.2.12 USI Interface

The User Services Interface (USI) is an interface between the FPGA fabric and the System Controller. It consists of the SYSRESET signal and Flash*Freeze signals. The following section describes USI interface signals.

2.2.12.1 SYSRESET

The POWER_ON_RESET_N signal is driven from the System Controller to the FPGA fabric. The System Controller is initiating a reset due to power-on reset, assertion of DEV_RST_N input, completion of programming, or completion of zeroization. This is an active low signal that can be used in the user design as a system power-on-reset for the FPGA fabric.

For more information, refer to the Reset Controller chapter in the [UG0448: IGLOO2 FPGA High Performance Memory Subsystem User Guide](#).

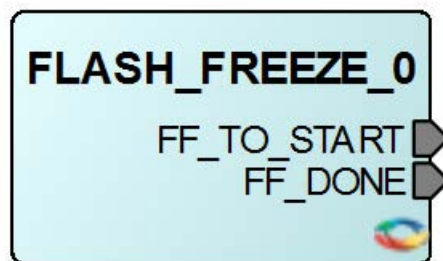
2.2.12.2 Flash*Freeze Signals

The USI interface provides two active high output signals (FF_TO_START and FF_DONE) related to Flash*Freeze to the FPGA Fabric.

- FF_TO_START is asserted by the System Controller to indicate that the Flash*Freeze service is about to start. Only 10 us are available to do housekeeping before the core is powered off. Microsemi recommends the user to use this signal as part of the clock gating process to ensure that any glitches do not cause a sequential element in the design to transition to an unwanted state when entering Flash*Freeze.
- FF_DONE is asserted by the System Controller to indicate the completion of Flash*Freeze.

These signals are made available by instantiating the FLASH_FREEZE macro from the Libero SoC IP catalog in SmartDesign or by instantiating it directly inside an HDL file. A block symbol of the FLASH_FREEZE macro which exposes the FF_TO_START and FF_DONE are presented in the following figure.

Figure 8 • FLASH_FREEZE Macro



For more information about Flash*Freeze, refer [UG0444: SmartFusion2 and IGLOO2 Low Power Design User Guide](#).

2.2.13 Clock Requirements

The System Controller is clocked by the on-chip 50 MHz RC oscillator.

It is powered by the VDD power pins and does not require external components for operation. The Chip Oscillators macro need not to be instantiated in the design for System Controller operation since it has a dedicated hardwired connection from the 50 MHz RC oscillator.

2.2.14 System Controller Suspend Mode

To protect the device from unintended behavior due to Single Event Upset (SEUs), the system controller can be held in suspend mode after device initialization. The system controller is active if the device is power-cycled or if a hard reset is applied. It returns to suspend mode once the initialization cycle is completed. A flash bit that is programmed during device programming controls the system controller suspend mode. This flash bit is not accessible from the customer design or by any external pin. The flash bit is only accessible through the programming file loaded into the device.

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

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

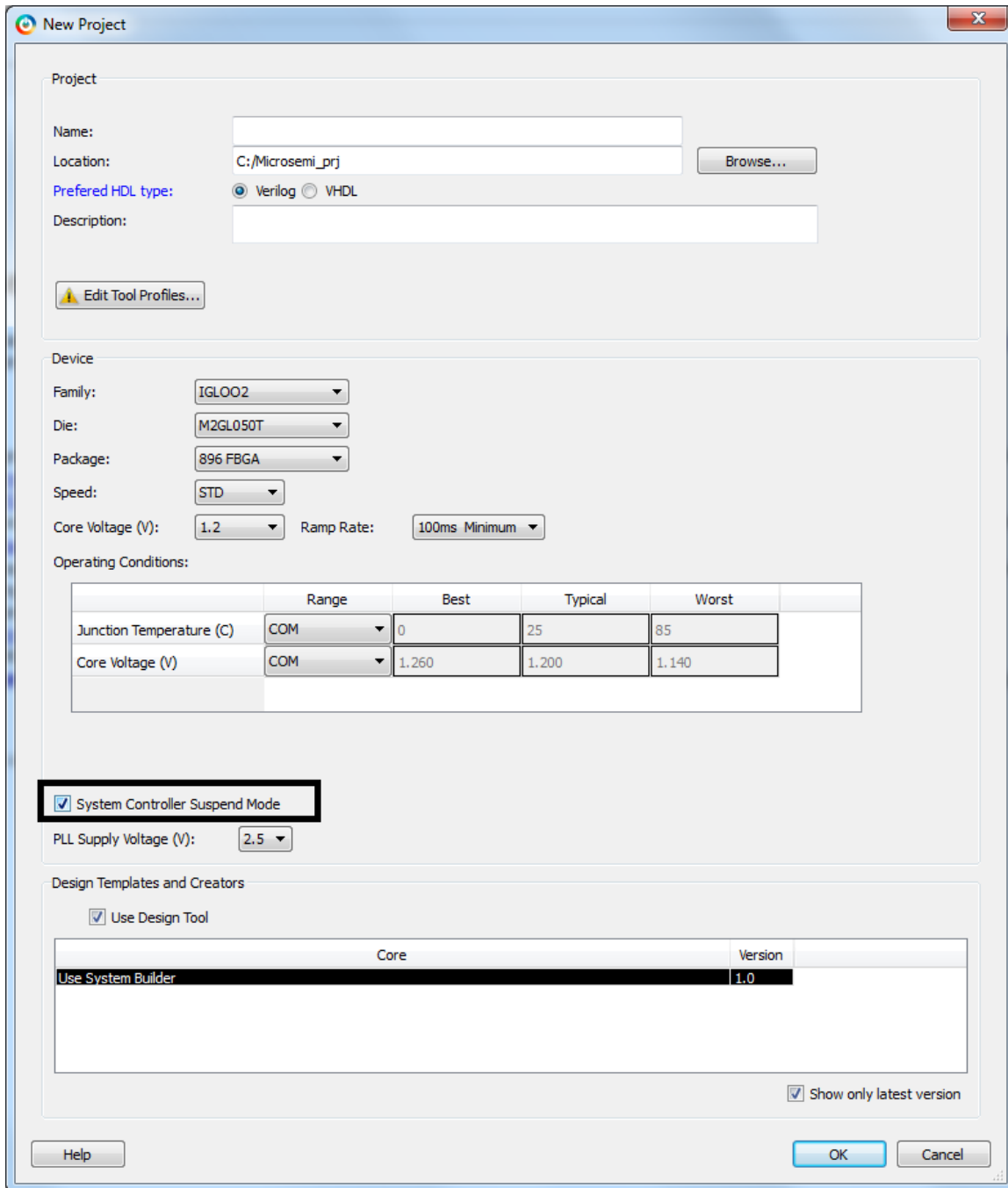
In the system controller suspend mode, the device can be reprogrammed or debugged using the JTAG port if the TRSTB pin is high. If the TRSTB pin is low, all the other JTAG input signals are blocked from activating the system controller. For prototyping or debugging, the device can be forced out of suspend mode by driving TRSTn high.

To restore normal operation, the device must be reprogrammed using the JTAG port with the system controller suspend mode bit turned off, that is, disable the system controller suspend mode in the Libero SoC software, regenerate the bitstream, and reprogram the device.

The System Controller Suspend mode feature can be configured (enable/disable) in Libero SoC software in two ways.

- The following figure shows the **New Project** dialog at the time of project creation.

Figure 9 • Enabling System Controller Suspended Mode in New Project Window



New Project

Project

Name:

Location:

Preferred HDL type: ☒ Verilog ☐ VHDL

Description:

Device

Family:

Die:

Package:

Speed:

Core Voltage (V): Ramp Rate:

Operating Conditions:

	Range	Best	Typical	Worst
Junction Temperature (C)	<input type="text" value="COM"/>	0	25	85
Core Voltage (V)	<input type="text" value="COM"/>	1.260	1.200	1.140

☒ System Controller Suspend Mode

PLL Supply Voltage (V):

Design Templates and Creators

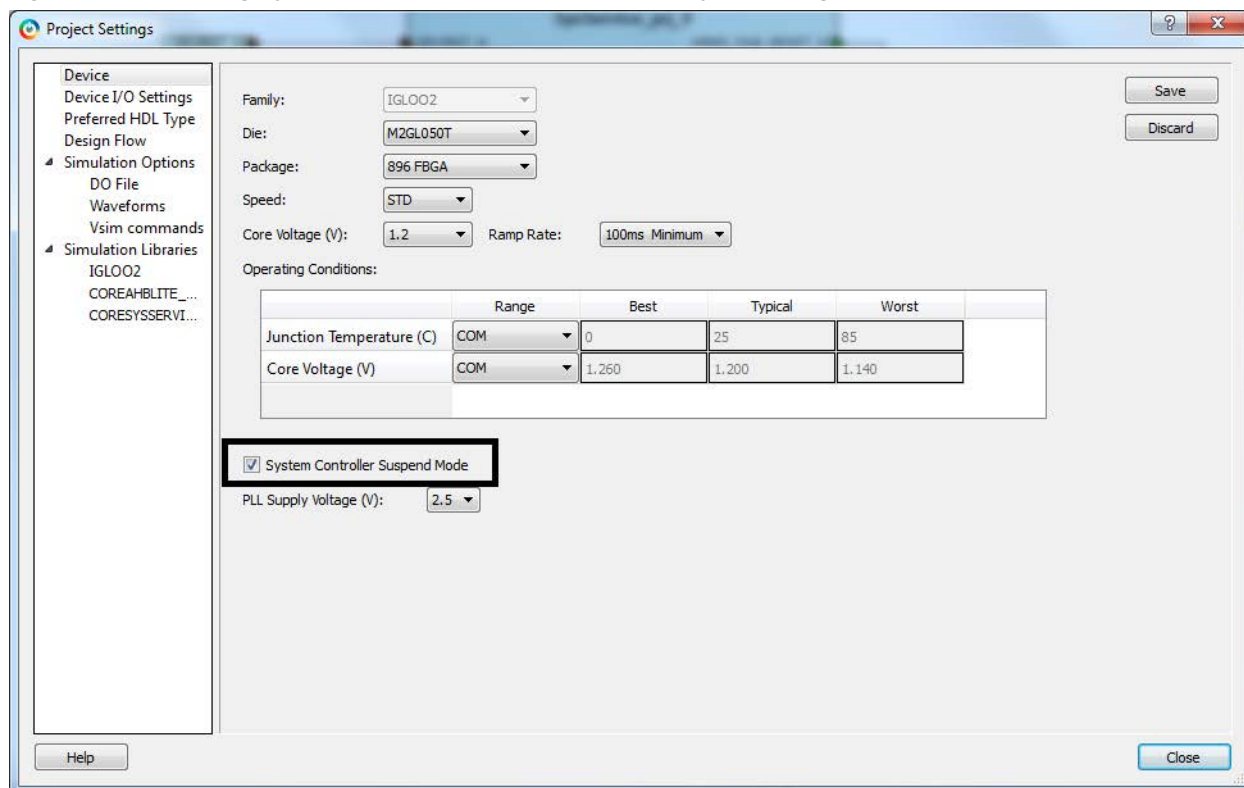
☒ Use Design Tool

Core	Version
Use System Builder	1.0

☒ Show only latest version

Note: Select the device family as SmartFusion2 or IGLOO2.

- The following figure shows the **Project Settings** dialog.

Figure 10 • Enabling System Services Suspend mode in Project Settings Window


By enabling this option, System Controller places itself in a reset state once the device is powered-up. This effectively suspends all the system services from being performed. For a list of system services, refer to [Table 3](#), page 15.

3 System Services

3.1 Introduction

This chapter describes the various system services implemented by the SmartFusion2 and IGLOO2 System Controller. The SmartFusion2 and IGLOO2 system services are System Controller actions initiated by asynchronous events from the ARM Cortex-M3 processor or a master in the FPGA fabric. Communication between the MSS/HPMS and the System Controller occurs through the communication block (COMM_BLK). Refer to the Communication Block chapter in the IGLOO2 High Performance Memory Subsystem User Guide and SmartFusion2 MSS User Guide for more information on COMM_BLK.

System services are requested from the Cortex-M3 processor or fabric master by sending a command byte describing the function to be performed, followed by command-specific sub-commands and/or data through the COMM_BLK interface. Upon completion of the requested service, service responses are sent to the ARM Cortex-M3 processor or the fabric master via the COMM_BLK interface. However, some commands read or write data directly from/to address ranges given by the user in the service request using the system IP interface master (SII Master), similar to the way DMA works. The SII master in the System Controller transfers data to or from the MSS/HPMS memory mapped address located in eSRAM, DDR DRAM, or FPGA fabric SRAM for most of the services. The location is dependent on the address pointers provided in the service request. Each transfer is checked for an AHB bus HRESP error in which MSS or HPMS memory access from the System Controller SII master is failed. If an MSS or a HPMS memory access error occurs, the requested service is aborted and an error is flagged. Commands F0H to FFH is used for high priority services. If a high priority command is received during execution of a low priority command, then the low priority command is aborted and any other commands queued in the COMM_BLK FIFO are discarded. High priority commands are only used for tamper detection purposes.

If a system service command arrives while a previous request is still in progress, the first service is canceled and the new service command is processed. If multiple masters such as the Cortex-M3 and FPGA Fabric master use the same system services, there should be some form of arbitration or co-operation between the masters to ensure that the services are used successfully. All pointers and multi-byte numeric arguments to the system services are little-endian type.

The system services are grouped into the following groups of services:

- Device and Design Information Services
- Flash*Freeze Service
- Cryptographic Services
- DPA-Resistant Key-Tree Services
- Elliptic Curve Cryptography Services
- SRAM-PUF Services
- Non-Deterministic Random Bit Generator (NRBG) Services
- Zeroization Service
- Programming Service
- NVM Data Integrity Check Service
- Asynchronous Messages

The following table lists all the SmartFusion2 and IGLOO2 system services with their command values. Microsemi provides CoreSysServices soft IP core to access the system services. The following

subsections provide the details of each system service including service request format, data descriptor layout, if applicable, and service response.

Table 3 • SmartFusion2 and IGLOO2 System Services¹

Category	System Service Name	Command Value	Response Status
Device and Design Information Services	Serial Number Service	0x1	0: Successful 127: HPMS memory access error (HRESP)
	USERCODE Service	0x4	
	Device Certificate Service	0x0	
	User Design Version Service	0x5	
Flash*Freeze Service	Flash*Freeze Service	0x2	0: Successful 254: Service disabled by factory security 255: Service disabled by user security
Cryptographic Services	256-bit AES Cryptographic Service	0x30x6	0: Successful 127: HPMS memory access error (HRESP) 253: Not licensed
	128-bit AES Cryptographic Service	0x60x3	254: Service disabled by factory security 255: Service disabled by user security
	SHA-256 Cryptographic Service	0xA	
	HMAC Cryptographic Service	0xC	
DPA-Resistant Key-Tree Services	Key-Tree Cryptographic Service	0x9	0: Successful 127: HPMS memory access error (HRESP) 253: Not licensed
	Challenge-Response Cryptographic Service	0xE	254: Service disabled by factory security 255: Service disabled by user security
Elliptic Curve Cryptography Services	ECC Point Multiplication Service	0x10	0: Success Completion 127: HRESP error occurred during MSS/HPMS transfer
	ECC Point Addition Service	0x11	253: License not available in device 254: Service disabled by factory security 255: Service disabled by user security
SRAM-PUF Services	Create or Delete User Activation Code Service	0x19	0: Success completion 1: eNVM MSS/HPMS error 2: PUF error, when creating 3: Invalid subcmd 4: eNVM program error 7: eNVM verify error 127: HRESP error occurred during MSS/HPMS transfer 253: License not available in device 254: Service disabled by factory security 255: Service disabled by user security

Table 3 • SmartFusion2 and IGLOO2 System Services¹ (continued)

Category	System Service Name	Command Value	Response Status
	Key Generation and Enrollment Services	0x1A	0: Success completion 1: eNVM MSS or HPMS error 2: PUF error, when creating 3: Invalid request or KC, when exporting or importing 4: eNVM program error 5: Invalid hash 6: Invalid user AC 7: eNVM verify error 8: Incorrect keysize for renewing a kc 10: Private eNVM user digest mismatch 11: Invalid subcmd 12: DRBG error 127: HRESP error occurred during MSS or HPMS transfer 253: License not available in device 254: Service disabled by factory security 255: Service disabled by user security
	Fetch a User PUF Key Service	0x1B	0: Success completion 2: PUF error, when creating 3: Invalid keynum or argument or exported or invalid key 5: Invalid hash 10: Private eNVM user digest mismatch 127: HRESP error occurred during MSS or HPMS transfer 253: License not available in device 254: Service disabled by factory security 255: Service disabled by user security
	Fetch User PUF ECC Public Key Service	0x1C	0: Success completion 3: No valid public key present in eNVM 10: Private eNVM user digest mismatch 127: HRESP error occurred during MSS/HPMS transfer 253: License not available in device 254: Service disabled by factory security 255: Service disabled by user security
	Get a PUF Seed Service	0x1D	0: Success completion 2: PUF error, when creating 127: HRESP error occurred during MSS/HPMS transfer 253: License not available in device 254: Service disabled by factory security 255: Service disabled by user security

Table 3 • SmartFusion2 and IGLOO2 System Services¹ (continued)

Category	System Service Name	Command Value	Response Status
Non-Deterministic Random Bit Generator (NRBG) Services	Self Test Service	0x28	0: Successful
	Instantiate Service	0x29	1: Fatal error
	Generate Service	0x2A	2: Maximum instantiations exceeded
	Reseed Service	0x2B	3: Invalid handle
	Uninstantiate Service	0x2C	4: Generate request too big
	Reset Service	0x2D	5: Maximum length of additional data exceeded 127: HPMS memory access error (HRESP) 253: Not licensed 254: Service disabled by factory security 255: Service disabled by user security
Zeroization Service	Zeroization Service	0xF0	No response status is sent.
Programming Service	IAP Service	0x14	Refer to Table 85 , page 45
	ISP Service	0x15	Refer to Table 86 , page 46
The ISP Service is applicable only for the SmartFusion2 devices.			
NVM Data Integrity Check Service	NVM Data Integrity Check Service	0x17	Digest error byte Bits [7:3] – Reserved Bits [2] – eNVM1 Error 0: ENV11 digest check passed 1: ENV11 digest check mismatch Bits [1] – eNVM0 Error 0: ENV00 digest check passed 1: ENV00 digest check mismatch Bits [0] – Fabric Error 0: Fabric FPGA configuration digest check passed 1: Fabric FPGA configuration digest check mismatch
Asynchronous Messages	Power-on-Reset (POR) Digest Error	0xF1	Digest error byte Bits [7:3] – Reserved Bits [2] – eNVM1 Error 0: ENV11 digest check passed 1: ENV11 digest check mismatch Bits [1] – eNVM0 Error 0: ENV00 digest check passed 1: ENV00 digest check mismatch Bits [0] – Fabric Error 0: Fabric FPGA configuration digest check passed 1: Fabric FPGA configuration digest check mismatch

1. The command values that are not listed in this table are treated as unrecognized commands. The System Controller's service response includes only the valid status codes as specified in this table.

3.2 Device and Design Information Services

The device and design information services return information about the device and current user design. The service request includes a service command and a pointer to a buffer in MSS or HPMS memory space to receive the result. The requested information is copied to a user-specified buffer whose address is included in the service request. The memory buffer can be located in eSRAM, DDR DRAM, or FPGA fabric SRAM. The return status of these services can be either success or MSS or HPMS memory access error, as listed in the following table.

Table 4 • Device and Design Information Services Status

Status	Description
0	Success
127	MSS or HPMS memory access error (HRESP)

3.2.1 Serial Number Service

This service fetches the 128-bit Device Serial Number (DSN). The DSN is a 128-bit quantity unique to every device, set during manufacturing.

Table 5 • Serial Number Service Request

Offset	Length (bytes)	Field	Description
0	1	CMD = 1	Command
1	4	DSNPTR	Pointer to 16-byte buffer to receive the 128-bit serial number

Table 6 • Serial Number Service Response

Offset	Length (bytes)	Field	Description
0	1	CMD = 1	Command
1	1	STATUS	Command status, see Table 4 , page 18
2	4	DSNPTR	Pointer to original buffer from request

3.2.2 USERCODE Service

This service fetches the 32-bit JTAG USERCODE programmed by the user.

Table 7 • USERCODE Service Request

Offset	Length (bytes)	Field	Description
0	1	CMD = 4	Command
1	4	USERCODEPTR	Pointer to 4-byte buffer to receive the 32-bit USERCODE

Table 8 • USERCODE Service Response

Offset	Length (bytes)	Field	Description
0	1	CMD = 4	Command
1	1	STATUS	Command status, see Table 4 , page 18
2	4	USERCODEPTR	Pointer to original buffer from request

3.2.3 Device Certificate Service

This service fetches the device certificate from eNVM. The device certificate is a digitally-signed X-509 certificate, signed by Microsemi Corp, programmed during the manufacturing process. The certificate is used to guarantee the authenticity of a device and its characteristics. The certificate is cryptographically validated before being delivered. For more information about the certificate and format, refer [AC436: Using Device Certificate System Service in SmartFusion2 Application Note](#).

Table 9 • Device Certificate Service Request

Offset	Length (bytes)	Field	Description
0	1	CMD = 0	Command
1	4	DEVICECERTPTR	Pointer to 768-byte buffer to receive the device certificate

Table 10 • Device Certificate Service Response

Offset	Length (bytes)	Field	Description
0	1	CMD = 0	Command
1	1	STATUS	Command status, see Table 10 , page 19.
2	4	DEVICECERTPTR	Pointer to original buffer from request

3.2.4 User Design Version Service

This service fetches the 16-bit user design version. For more information about design versioning, refer [UG0443: SmartFusion2 and IGLOO2 FPGA Security Best Practices User Guide](#).

Table 11 • Design Version Service Request

Offset	Length (bytes)	Field	Description
0	1	CMD = 5	Command
1	4	DESIGNVERPTR	Pointer to 2-byte buffer to receive the 16-bit design version

Table 12 • Design Version Service Response

Offset	Length (bytes)	Field	Description
0	1	CMD = 5	Command
1	1	STATUS	Command status, see Table 4 , page 18.
2	4	DESIGNVERPTR	Pointer to original buffer from request

3.3 Flash*Freeze Service

This service requests the System Controller to execute the Flash*Freeze entry sequence. For more information about the Flash*Freeze entry sequence, refer “Flash*Freeze” chapter in the [UG0444: SmartFusion2 and IGLOO2 Low Power Design User Guide](#). Note that the Flash*Freeze service is only available if the service has been enabled by the user as part of Libero hardware flow of the design programmed into the device.

Table 13 • Flash*Freeze Request

Offset	Length (bytes)	Field	Description
0	1	CMD = 2	Command
1	1	FFOPTIONS	Flash*Freeze options. Refer to Table 14 , page 20.

Table 14 • FFOPTIONS

7	6	5	4	3	2	1	0
Reserved					MPLLPD	ENVM1PD	ENVM0PD

Note: Reserved bits indicate that even if the user writes these bits, it does not affect the functionality.

The following is a description of the FFOPTIONS mentioned in [Table 14](#), page 20:

- If ENVM0PD is ‘1’ then eNVM module 0 is placed in its deep-power-down state. Refer to the following note.
- If ENVM1PD is ‘1’ then eNVM module 1 is placed in its deep-power-down state (ignored if eNVM module 1 is not present). Refer to the following note.
- If MPLLPD is ‘1’ then the MPLL is powered down for the duration of the Flash*Freeze period.

Note: System Services must not perform deep-power-down of eNVM0 and eNVM1. It is recommended to perform a deep-power-down of eNVMs by System Registers before requesting for the Flash*Freeze service.

When the Flash*Freeze service begins execution, the System Controller informs that a Flash*Freeze shutdown is imminent by sending a command byte E0H to COMM_BLK. The service is stalled until this command byte can be accepted by the COMM_BLK FIFO. Any new requests received during this time are not processed until after the Flash*Freeze state has been exited.

When the Flash*Freeze shutdown sequence is complete, the System Controller responds with a service response, as listed in the following tables.

Table 15 • Flash*Freeze Service Response

Offset	Length (bytes)	Field	Description
0	1	CMD = 2	Command
1	1	STATUS	Command status

Table 16 • Flash*Freeze Service Status

Status	Description
0	Success
254	Service disabled by factory security
255	Service disabled by user security

3.4 Cryptographic Services

Selected devices of the SmartFusion2 or IGLOO2 family can access the built-in Advanced Encryption Standard (AES) and secure hash standard (SHA-256) engines through cryptographic services. These cryptographic services can be used for data security applications by the end user. These services can be disabled by the factory or user security settings. Attempting to execute one of these services on devices that do not support these advanced security features return the status as not licensed. For information about which devices have the cryptographic services, refer to the “Highest Security Devices” section in the [PB0121: IGLOO2 FPGA Product Brief](#) and [PB0115: SmartFusion2 SoC FPGA Product Brief](#). The possible service status values for the cryptographic services are listed in the following table.

Table 17 • Cryptographic Services Status Codes

Status	Description
0	Success
127	HRESP error occurred during HPMS transfer
253	Not licensed
254	Service disabled by factory security
255	Service disabled by user security

3.4.1 AES Services

The System Controller AES engine is implemented as specified in federal information processing standard (FIPS) publication 197, the AES. It is designed to support the following AES cipher operating modes as recommended in National Institute of Standards and Technology (NIST) special publication 800-38A, [Recommendation for Block Cipher Modes of Operation](#).

- Electronic codebook (ECB)
- Cipher-Block chaining (CBC)
- Output feedback (OFB)
- Counter (CTR)

The AES engine can be operated in 128-bit mode or 256-bit mode. The length of the AES key is 128 bits in 128-bit mode and 256 bits in 256-bit mode. An AES service request includes a service command and a pointer to a descriptor in MSS or HPMS memory space describing the transaction to be performed. The descriptor is retrieved via the SII Master.

The referenced key data and plain text/cipher text is then also retrieved via the SII Master. The resultant cipher text/plain text is copied back to the MSS or HPMS memory space using the SII Master. The SmartFusion2 and IGLOO2 AES and AES cipher mode services assume input data is in complete 128-bit blocks, and provide only complete 128-bit output blocks. Adding any padding bits to incomplete plain text blocks before calling the encryption service, and removing any padding bits after receiving the results of the decryption service, is the responsibility of the user. The input and output data format of the AES services is little-endian type. The first byte of the first block is at the lowest address and there are no word alignment requirements.

The MODE parameter defined in the data descriptor specifies the cipher operating mode and whether the source text must be encrypted or decrypted.

3.4.1.1 128-bit AES Cryptographic Service

The 128-bit AES service provides the user design access to the System Controller AES engine in 128-bit mode.

Table 18 • 128-bit AES Service Request

Offset	Length (bytes)	Field	Description
0	1	CMD = 3	Command
1	4	AES128DATAPTR	Pointer to AES128DATA descriptor

The layout of the data descriptor to be passed in this mode is listed in the following table

Table 19 • AES128DATA Descriptor

Offset	Length (bytes)	Field	Description
0	16	KEY	Encryption key to be used
16	16	IV	Initialization vector (Ignored for ECB mode)
32	2	NBLOCKS	Number of 128-bit blocks to process (max 65535)
34	1	MODE	Cipher operating mode. Refer to Table 20, page 22.
35	1	RESERVED	Reserved
36	4	DSTADDRPTR	Pointer to return data buffer
40	4	SRCADDRPTR	Pointer to data to encrypt/decrypt

Table 20 • MODE Parameter

7	6	5	4	3	2	1	0
DECRYPT	Reserve d					OPMODE	E

Note: Reserved bits indicate that even if the user writes these bits, it does not affect the functionality.

In the MODE parameter, if DECRYPT is 0 then the data at SRCADDRPTR field is treated as plain text for encryption. If DECRYPT is 1 then the data at SRCADDRPTR field is treated as cipher text for decryption. The OPMODE field specifies the operating mode for the AES engine.

Table 21 • OPMODE Parameter

OPMODE	Cipher Mode	Note
0	ECB	Initialization vector (IV) parameter is ignored
1	CBC	
2	OFB	
3	CTR	$CTR_0 = IV$. Increment is modulo 2^{128} . Counter is big-endian type.

The IV parameter is a 16-byte array containing the initialization vector that will be used as a part of the requested encryption/decryption operation. Its use is different, depending on the mode. ECB mode ignores the IV parameter and CTR mode uses the content of the IV parameter as its initial counter value.

Table 22 • 128-bit AES Service Response

Offset	Length (bytes)	Field	Description
0	1	CMD = 3	Command

Table 22 • 128-bit AES Service Response (continued)

Offset	Length (bytes)	Field	Description
1	1	STATUS	Command status
1	4	AES128DATAPTR	Pointer to AES128DATA descriptor

3.4.1.2 256-bit AES Cryptographic Service

The 256-bit AES service provides the user design access to the System Controller AES engine in 256-bit mode.

Table 23 • 256-bit AES Service Request

Offset	Length (bytes)	Field	Description
0	1	CMD = 6	Command
1	4	AES256DATAPTR	Pointer to AES256DATA descriptor

The layout of the data descriptor to be passed in this mode is listed in the following table. The description of the MODE parameter is the same as for the 128-bit AES service.

Table 24 • AES256DATA Descriptor

Offset	Length (bytes)	Field	Description
0	32	KEY	Encryption key to be used
32	16	IV	Initialization vector, ignored for ECB mode
48	2	NBLOCKS	Number of 128-bit blocks to process (maximum 65,535)
50	1	MODE	Cipher operating mode. Refer to Table 20 , page 22.
51	1	RESERVED	Reserved
52	4	DSTADDRPTR	Pointer to return data buffer
56	4	SRCADDRPTR	Pointer to data to encrypt/decrypt

Table 25 • 256-bit AES Service Response

Offset	Length (bytes)	Field	Description
0	1	CMD = 6	Command
1	1	STATUS	Service status
2	4	AES256DATAPTR	Pointer to AES256DATA descriptor

3.4.2 SHA-256 Services

The SHA-256 services provide the user design access to the System Controller SHA-256 engine. The System Controller SHA-256 engine is implemented as specified in [FIPS PUB 180-3, SHA](#). The service request includes a service command and a pointer to a descriptor in MSS or HPMS memory space, which includes the associated data.

3.4.2.1 SHA-256 Cryptographic Service

The SHA-256 cryptographic service provides the user design access to the System Controller SHA-256 engine. The service allows for lengths up to 2^{32} bits of data to hash. If the message ends with a partial byte, the significant bits are assumed to be at the LSB end of the byte. The unused MSBs of the final byte is ignored. The input and output data format of the SHA-256 service is little-endian type. Input messages

are automatically padded with a minimum of 65 bits up to a maximum of 576 additional bits, depending on the length of the user input message, as per the SHA-256 standard.

Table 26 • SHA-256 Service Request

Offset	Length (bytes)	Field	Description
0	1	CMD = 10	Command
1	4	SHA256DATAPTR	Pointer to SHA256DATA structure

The layout of the data descriptor to be passed in this service is listed in the following table.

Table 27 • SHA256DATA Structure

Offset	Length (bytes)	Field	Description
0	4	LENGTH	Length of data pointed to by DATAINPTR field in bits (up to 2^{32} bits).
4	4	HASHRESULTPTR	Pointer to 32-byte buffer to receive 256-bit hash result.
8	4	DATAINPTR	Pointer to data to be hashed

Table 28 • SHA-256 Service Response

Offset	Length (bytes)	Field	Description
0	1	CMD = 10	Command
1	1	STATUS	Command status
2	4	SHA256DATAPTR	Pointer to SHA256DATA structure

3.4.2.2 HMAC Cryptographic Service

The Keyed-Hash Message Authentication Code (HMAC) service implements the [FIPS 198 HMAC Algorithm](#) using SHA-256 as the approved hash function. Key length up to 32 bytes (256 bits) can be used to generate the message authentication code. If the key length is less than 256 bits, the unused upper bits should be set to 0. The service allows for lengths up to 2^{32} bits of data to hash. If the message ends with a partial byte, then the significant bits are assumed to be at the LSB end of the byte. The unused MSBs of the final byte is ignored. The input and output data format of the HMAC service is a little-endian type. Input messages are automatically padded with a minimum of 65 bits up to a maximum of 576 additional bits, depending on the length of the user input message, as per the SHA-256 standard.

Table 29 • HMAC Service Request

Offset	Length (bytes)	Field	Description
0	1	CMD = 12	Command
1	4	HMACDATAPTR	Pointer to HMACDATA structure

The layout of the data descriptor to be passed in this service is listed in the following table.

Table 30 • HMACDATA Structure

Offset	Length (bytes)	Field	Description
0	32	KEY	Key to use
32	4	LENGTH	Length of data pointed to by DATAINPTR field in bytes
36	4	DATAINPTR	Pointer to data to be hashed

Table 30 • HMACDATA Structure (continued)

Offset	Length (bytes)	Field	Description
40	4	RESULTPTR	Pointer to 32-byte buffer to receive 256-bit HMAC result.

Table 31 • HMAC Service Response

Offset	Length (bytes)	Field	Description
0	1	CMD = 12	Command
1	1	STATUS	Command status
2	4	HMACDATAPTR	Pointer to HMACDATA structure

3.5 DPA-Resistant Key-Tree Services

Differential Power Analysis (DPA)-resistant services utilize a SHA-256 based key-tree algorithm recommended by cryptography research inc (CRI). These services are only available on the advanced security enabled devices. For information about devices have the DPA-Resistant Key-Tree Services, refer to the Highest Security Devices section in the [PB0121: IGLOO2 FPGA Product Brief](#) and [PB0115: SmartFusion2 SoC FPGA Product Brief](#).

A common CRI-patented protocol-level construct for DPA-safe designs is the key tree. It is useful for mixing a secret value such as a key with a public value such as an initialization vector, nonce, or hash result. It takes as input a 256-bit secret that is hashed with one of two constants in a binary tree arrangement, returning the result of the final hash. The decision as to which branch to take at each level of the tree is determined by one bit of the public input. The public input is comprised of a 7-bit and a 128-bit portion. The side channel information leakage is bounded because each secret value in the tree is used in several ways. At each level the hash operation mixes the secret, making any information an adversary may have learned at one level essentially useless at the next level.

The Key Tree can be used for a number of applications, such as:

- A Message Authentication Code algorithm (from a key and a hash input)
- A key derivation function (from a root key and a key ID)
- To emulate an ideal PUF (from a PUF secret value and a challenge)
- In Challenge-Response protocols (from a key and a challenge)
- To generate pseudo-random bits (from a seed and a counter)

The SmartFusion2 and IGLOO2 devices have the following key-tree based services:

- Generic Key-Tree Cryptographic Service
- Challenge-Response Cryptographic Service

The following table lists the possible service status values for the key-tree services.

Table 32 • DPA-Resistant Key-Tree Services Status Codes

Status	Description
0	Success
127	HRESP error occurred during MSS or HPMS transfer
253	Not licensed
254	Service disabled by factory security
255	Service disabled by user security

3.5.1 Key-Tree Cryptographic Service

The generic key-tree service begins with a user-supplied 256-bit root key and derives a 256-bit output key using the following two input parameters:

- a 7-bit optype value which can be used to separate up to 128 possible uses of the key-tree
- a 128-bit path input that is used to mix the root key pseudo-randomly over a 2^{128} output space

Both the 7-bit input parameter and the 128-bit path variable are assumed to be publicly known, and in any case, it must not be assumed that there is any significant DPA resistance to learn them. The root key, any intermediate keys calculated, and the output key exhibits good DPA resistance. One common use for the output key is as a keyed validator, similar to a message authentication code tag.

Table 33 • Key-Tree Service Request

Offset	Length (bytes)	Field	Description
0	1	CMD = 9	Command
1	4	KEYTREEDATAPTR	Pointer to KEYTREEDATA structure

The layout of the data descriptor to be passed in this service is listed in the following table.

Table 34 • KEYTREEDATA Structure

Offset	Length (bytes)	Field	Description
0	32	KEY	256-bit key (root key) to be modified or generated output key
32	1	OPTYPE	Key-tree optype parameter (7-bits, MSB ignored)
33	16	PATH	Path variable to be used

Table 35 • KeyTree Service Response

Offset	Length (bytes)	Field	Description
0	1	CMD = 9	Command
1	1	STATUS	Command status
2	4	KEYTREEDATAPTR	Pointer to KEYTREEDATA structure

3.5.2 Challenge-Response Cryptographic Service

The challenge-response service performs a similar key-tree calculation using the input challenge with a root key fixed as the internal device-unique key. The service accepts a challenge comprising a 7-bit optype and 128-bit path and returns a 256-bit response unique to the given challenge and the device.

On premium S grade smaller SmartFusion2 or IGLOO2 devices from M2S050S or M2GL050S and below, the challenge-response service is based a unique Pseudo-PUF key. The Pseudo-PUF key based challenge-response service performs the key-tree calculation using a 256-bit static random key. The static random key is generated and stored in the device during its manufacturing process as a starting secret. The static secret key is never revealed during its manufacturing process or during its usage.

On premium S grade SmartFusion2 or IGLOO2 larger devices (M2S060S or M2GL060S, M2S090/M2GL090, and M2S150/M2GL150), the challenge-response service is based on a unique SRAM-PUF ECC private key. Microsemi adds a completely random unique-per-device 384-bit ECC private key during device manufacturing. This key is generated in the Microsemi FIP140-2 level 3 HSM during the key provisioning steps of the device manufacturing process, and is not recorded or re-constructible by Microsemi. It is imported into the device using a fully authenticated and encrypted wrapper, so the private key is never visible in plaintext outside the security boundary of the HSM or the device. This key is protected by SRAM-PUF hardware by enrolling it as an extrinsic key. Since this ECC private key is protected by the intrinsic unclonable nano-scale properties unique to a single device, it provides the strongest integrated circuit device key protection available.

The challenge-response service provides a mechanism for authenticating a device. To use it, several challenges are generated, and the responses are recorded during the device enrollment phase. When the user accesses the device later, the responses are confirmed with the recorded details of the device for proof as it is the same device that was enrolled.

Table 36 • Challenge-Response Service Request

Offset	Length (bytes)	Field	Description
0	1	CMD = 14	Command
1	4	CHRESPPTR	Pointer to CHRESP structure

The following table lists the layout of the data descriptor to be passed in this service.

Table 37 • CHRESP Structure

Offset	Length (bytes)	Field	Description
0	4	KEYADDRPTR	Pointer to 32-byte buffer to receive result
4	1	OPTYPE	Key-tree optype parameter (MSB ignored)
5	16	PATH	Path variable to be used

The following table lists the challenge-response service response.

Table 38 • Challenge-Response Service Response

Offset	Length (bytes)	Field	Description
0	1	CMD = 14	Command
1	1	STATUS	Command status
2	4	CHRESPPTR	Pointer to CHRESP structure

3.6 Elliptic Curve Cryptography Services

The premium S grade SmartFusion2 or IGLOO2 larger devices (M2S060S or M2GL060S, M2S090S or M2GL090S, and M2S150S or M2GL150S) have an ECC hardware accelerator to support asymmetric cryptographic techniques for key establishment.

The ECC hardware accelerator implements the NIST P-384 curve and provides support for point multiplication and point addition with countermeasures against side-channel analysis (SCA). The NIST P-384 curve's domain parameters are defined in NIST FIPS PUB 186-3 specification. The P-384 curve is one of the elliptic curves included in the NIST Suite B list of approved algorithms for protecting classified information up to and including the top secret.

Point Multiplication incorporates the strong intrinsic algorithmic and other DPA countermeasures. Point Addition is not a DPA-resistant. It is designed to be a time-invariant to protect from timing attacks and simple power attacks.

The ECC cryptosystem is based on the apparent difficulty of reversing the point multiplication operation that is, given a scalar and a base point, it is easy to calculate the point resulting from multiplication. But given the base point and the result point, it is very (that is, cryptographically) difficult to determine what the scalar is. Based on the best known attacks, the security strength of an ECC cryptosystem is estimated as half the number of bits in the private key, that is, the security strength of the P-384 system is about 192 bits.

The built-in ECC hardware accelerator does not check that the point(s) given it as input(s) are legal points on the NIST P-384 curve. Supplying illegal X-Y coordinates for a point results in garbage output. However, if the legal input points are given, the accelerator outputs are guaranteed to be correct. If needed, the provided point multiplication and addition services can be used to help check that a point is on the curve.

The ECC system services provide access to the ECC hardware accelerator. Points on the curve are encoded as two 384-bit big-endian numbers (X, Y), and stored in two consecutive blocks of 48-bytes, with the X coordinate first. The point-at-infinity is represented by the point (0, 0). The following table lists the service status codes for ECC services.

Table 39 • Challenge-Response Service Response

Status	Description
0	Success completion
127	HRESP error occurred during MSS/HPMS transfer
253	License not available in device
254	Service disabled by factory security
255	Service disabled by user security

3.6.1 ECC Point Multiplication Service

The ECC point multiplication service multiplies a point (P) by a scalar (d). The scalar is a 384-bit integer, and points are defined by two integers depicting the points X and Y coordinates. All these integers are restricted to the prime Galois Field defined by the P-384 domain parameters. The input point must be on the Elliptic P-384 curve. The point multiplication results in point Q as follows:

$$Q = d * P$$

The ECC point multiplication is used to compute ECC public keys provided in the private key (per NIST FIPS PUB 186-3 Appendix B.4). The ECC point multiplication is also commonly used for establishing a shared secret (the x-coordinate of the resulting point) using the Diffie-Hellman protocol. Two parties generate key pairs (private key and public key), as above, and exchange their public keys. Each multiplies the public key they received (a point) with their own private key (a scalar). The resulting point calculated by both parties is the same.

Other useful ECC public key operations include generating and validating digital signatures. The ECC point multiplication and point addition functions, along with a hash function can be used to implement digital signature operations. Generating an NIST approved digital signature having a security strength of 192 bits requires 384-bit ECC operations (which these devices have), and a 384-bit hash operation. The hash operation is not part of the ECC hardware accelerator, but the built-in SHA-256 services can be used for the hash operation.

The following table lists the ECC point multiplication service request format.

Table 40 • Elliptic Curve Point Multiplication Service Request

Offset	Length (bytes)	Field	Description
0	1	CMD = 16	Command
1	4	ECCMULTPTR	Pointer to ECCPMULT structure

The following table lists the layout of the data descriptor (ECCPMULT) to be passed in the service request.

Table 41 • ECCPMULT Structure

Offset	Length (bytes)	Field	Description
0	4	DPTR	Pointer to 384-bit scalar, d (big endian)
4	4	PPTR	Pointer to (X, Y) coordinates of P
8	4	QPTR	Pointer to (X, Y) coordinates of result Q

In the ECCPMULT structure, if PPTR is 0:

$$Q = d * G$$

Where G is the generator point or base point for the NIST P-384 curve. This form of the service can be used to create a public key from the private key, d. The domain parameters specify the base point to use. The NIST P-384 curve's base point is built into the ECC hardware accelerator and need not be provided. The public key is just the private key times the base point.

The following table lists the ECC point multiplication service response.

Table 42 • ECCPMULT Structure

Offset	Length (bytes)	Field	Description
0	1	CMD = 16	Command
1	1	STATUS	Command status, (see Table 39 , page 28)
2	4	ECCMULTPTR	Pointer to ECCPMULT structure

3.6.2 ECC Point Addition Service

The ECC Point addition is defined as like an arbitrary operation involving the (x, y) coordinates of both input points and the elliptic curve equation, resulting in another (x, y) point on the curve. The inputs are two (x, y) points (P and Q), each lying on the P-384 curve, and the result is another (x, y) point (R), which is guaranteed to be on the curve (or be the point at infinity).

$$R = P + Q$$

The following table lists the ECC point addition service request format.

Table 43 • Elliptic Curve Point Addition Service Request

Offset	Length (bytes)	Field	Description
0	1	CMD = 17	Command
1	4	ECCPADDPTR	Pointer to ECCPADD structure

The following table lists the layout of the data descriptor (ECCPADD) to be passed in the service request.

Table 44 • ECCPADD Structure

Offset	Length (bytes)	Field	Description
0	4	PPTR	Pointer to (X, Y) coordinates of input point P
4	4	QPTR	Pointer to (X, Y) coordinates of input point Q
8	4	RPTR	Pointer to (X, Y) coordinates of result R

The following table lists the ECC point addition service response.

Table 45 • Elliptic Curve Point Addition Service Response

Offset	Length (bytes)	Field	Description
0	1	CMD = 17	Command
1	1	STATUS	Command status, (see Table 39 , page 28)
2	4	ECCPADDPTR	Pointer to ECCPADD structure

3.7 SRAM-PUF Services

The SRAM-PUF services can be used for key generation and storage and device authentication. SRAM-PUF uses the random start-up behavior of a dedicated 2 KB SRAM to determine an intrinsic secret unique to each device. In each device, the SRAM turn-on behavior is essentially independent (even down to the single-bit level), but from turn-on to turn-on in a single device there is sufficient repeatability to reconstruct the same intrinsic secret each time.

The SRAM-PUF services can be used for generating keys in data security applications because of its randomness and device individual fingerprint. The primary advantage of SRAM-PUF for the key generation is that the keys are dynamically reconstructed without storing in memory. Keys are generated only when needed on-the-fly. The SRAM-PUF is also used in design security. Although, the system services are not used for design security, it is possible to retrieve the design security User PUF ECC public key for device authentication purpose. The SRAM-PUF start-up value can also be used to generate a seed for random number generator.

The SRAM-PUF is available in the larger SmartFusion2 or IGLOO2 devices (M2S060 or M2GL060, M2S090 or M2GL090, and M2S150 or M2GL150) for design security applications. On premium S grade larger SmartFusion2 or IGLOO2 devices (M2S060S or M2GL060S, M2S090S or M2GL090S, and M2S150S or M2GL150S), the system services are used to access the SRAM-PUF hardware for data security applications.

The following sections describe how to generate and store a key using SRAM PUF hardware:

- Create User Activation Code
- Key Generation and Enrollment Services
- Key Reconstruction

3.7.1 Create User Activation Code

The initial step for a key generation or key storage using SRAM-PUF is the user activation code creation or device enrollment. A user activation code is created based on the start-up behavior of the dedicated SRAM. The user activation code size is 1192 byte, stored in the eNVM private area. The user activation code is used for key enrollment and key reconstruction. The user activation code is used to eliminate the randomness of the SRAM-PUF power-up content in order to retrieve the PUF secret key.

The user activation code must be generated once, typically when the system containing the SmartFusion2 or IGLOO2 device is commissioned. However, the device can be enrolled multiple times, producing a new user activation code for each enrollment. In this case, previously enrolled keys using the older activation code becomes invalid, making the key reconstruction is impossible. The user activation code value is never exported in clear text from the device. The user activation code can be

destroyed. This function would typically only be used when the system containing SmartFusion2 or IGLOO2 device is decommissioned or re-purposed.

3.7.1.1 Create or Delete User Activation Code Service

The Create or Delete User Activation Code Service can be used to create a user activation code or delete the existing user activation code and key codes by supplying a subcommand value.

In the data descriptor, the SUBCMD filed defines the user activation code creation or deletion. If SUBCMD filed is 0 [CREATE_AC], the SRAM-PUF hardware is requested to enroll a new user activation code. The dedicated SRAM is turned on before executing this command.

If SUBCMD filed is 1 [DELETE_AC], the user AC gets deleted together with all the user key codes and ECC public key. The dedicated SRAM is not turned ON in this case since SRAM-PUF operations are not involved.

The following table lists the service request format.

Table 46 • PUFUSERAC Service Request

Offset	Length (bytes)	Field	Description
0	1	CMD = 25	Command
1	4	PUFUSERACPTR	Pointer to PUFUSERAC structure

The following table lists the layout of the data descriptor (PUFUSERAC) to be passed in the service request.

Table 47 • PUFUSERAC Structure

Offset	Length (bytes)	Field	Description
0	1	SUBCMD	Sub Command 0: CREATE_AC 1: DELETE_AC

The following table lists the PUFUSERAC service response.

Table 48 • Elliptic Curve Point Addition Service Response

Offset	Length (bytes)	Field	Description
0	1	CMD = 25	Command
1	1	STATUS	Command status, (see Table 49 , page 31)

Table 49 • PUFUSERAC Status¹

STATUS	Description
0	Success completion
1	eNVM MSS/HPMS error
2	PUF error, when creating
3	Invalid subcmd
4	eNVM program error
7	eNVM verify error
127	HRESP error occurred during MSS or HPMS transfer
253	License not available in device
254	Service disabled by factory security

Table 49 • PUFUSERAC Status¹ (continued)

STATUS	Description
255	Service disabled by user security

1. Errors can occur when writing to the private eNVM for both subcommands.

3.7.2 Key Generation and Enrollment Services

The SRAM-PUF can be used to store cryptographic keys. The keys are stored in a way that the key's actual value does not appear in the system unless it is retrieved manually. A key code is stored in the eNVM private area instead of the key's value. The key code is generated when a key is enrolled. The key code value is created from the enrolled key value and the user activation code. The key's value can then later be regenerated from the key code value and user activation code on request.

Keys can be either intrinsic keys or extrinsic keys. An intrinsic key is randomly generated by the SRAM-PUF hardware during key enrollment. The intrinsic keys are useful where a security protocol executing on SmartFusion2 and IGLOO2 requires to generate a key value and to store it for later use. For example, you can request a 384-bit long intrinsic key to be enrolled and use it as a private key in an elliptic curve Diffie-Hellman key exchange. An extrinsic key is supplied. For example, you can request an symmetric key obtained from a key exchange protocol to be enrolled for later use. Multiple key codes can be generated from multiple intrinsic or extrinsic keys. Keys are identified by a number and must be enrolled sequentially. The first step in enrolling a new key is to determine how many keys are already enrolled. The maximum number of keys supported is 58 that is, 0 to 57. The key size can vary from 64-bit to 4096-bit in multiples of 64.

The first two key codes—(KC#0 and KC#1)—are reserved for user-enrolled design security keys. KC#0 and KC#1 are purely used by the bit-stream process to protect the design, which is programmed in the device. The KC#0 and KC#1 are called Design Security Keys. These user-enrolled design security keys are a 384-bit intrinsic User PUF ECC private key and a 256-bit extrinsic user symmetric PUF key. Generation of the 384-bit intrinsic key, import of the 256-bit extrinsic key, generation of the user activation code, and the two key codes occur as a result of loading a bit-stream with these options activated. After the true random 384-bit intrinsic User PUF ECC private key is checked to be in the valid P-384 range and enrolled, the device computes the corresponding ECC public key internally and exports the key along with an authentication tag to prevent a man-in-the-middle attack. The User PUF ECC public key is stored in plaintext in the eNVM private area. The User ECC private key is protected by the SRAM-PUF, and neither it never leaves the device nor it is ever exported to the user of the device internally.

The key codes from KC#2 to KC#n are purely used for data security applications. The key codes from KC#2 to KC#n are called data security keys since they protect the customer data. The system services can create or delete data security keys. The data security keys of variable length might also be enrolled using the system services. The data security keys can be reconstructed by SRAM-PUF and exported from the System Controller to the design running in the device. The user activation code is the same for both design security keys and data security keys.

3.7.2.1 Create or Delete User Key Code and Export or Import All Service

Create or Delete User Key Code and Export or Import All Service perform the key generation and enrollment. They can be used for the following functions:

- Get number of enrolled keys
- Enroll an extrinsic user key
- Enroll an intrinsic key
- Export user activation code and all key codes
- Import user activation code and all key codes
- Delete a user key from enrolled keys

The following table lists the PUFUSERKC service request format.

Table 50 • PUFUSERKC Service Request

Offset	Length (bytes)	Field	Description
0	1	CMD = 26	Command
1	4	PUFUSERKCPtr	Pointer to PUFUSERKC structure

The following table lists the layout of the data descriptor (PUFUSERKC) to be passed in the service request. In the data descriptor, the SUBCMD filed defines the function to be requested. Only one option must be used at a time.

Table 51 • PUFUSERKC Service Request

Offset	Length (bytes)	Field	Description
0	1	SUBCMD	Sub Command [INPUT] 0: GET_NUMBER_OF_KC 1: CREATE_EXT_KC 2: CREATE_INT_KC 3: EXPORT_ALL_KC 4: IMPORT_ALL_KC 5: DELETE_KC
1	4	PUFUSERKEYADDR	PUF User Key Fetch address, when creating/enrolling a key or PUF User KC address, to export to or import from [INPUT]
5	4	USEREXTRINSICKEYADDR	User Extrinsic Key address, when creating [INPUT]
9	1	KEYNUM	Key number from 2 to 57 [INPUT] [OUTPUT]
10	1	KEYSIZE	Key size 0 to 64: [INPUT] 0 means 4096 bit 1 means 64 bit 63 means 63*64 = 4032 bit

3.7.2.2 Get Number of Enrolled Keys

If SUBCMD filed is 0 [GET_NUMBER_OF_KC], the total number of user keys enrolled is returned in KEYNUM filed. The number of enrolled keys is from 2 to 58. All valid and invalid keys are counted up to the last valid key. In this context, an invalid key is one that got deleted since it is followed by a valid key. You can only create new keys in the sequence from (KEYNUM) number. Use the SUBCMD = 0 option to get in KEYNUM for creating the next key. The total number of keys is as a minimum of 2 (KC#0 and KC#1), which are used by the bit-stream, and as a maximum of 58 (KC#0 to KC#57). The KEYNUM returned is the total number of keys. The SUBCMD is 0 and the maximum valid key number is always plus one. To create a KC, a key number from 2 to 57 must be provided, since 0 and 1 are reserved.

3.7.2.3 User Key (Extrinsic or Intrinsic) Enrollment

If SUBCMD is 1 [CREATE_EXT_KC] or 2 [CREATE_INT_KC], the SRAM-PUF is requested to generate a new user key code for an extrinsic key or intrinsic key respectively. KC#0 and KC#1 keys can only be generated from the bit-stream. But KC#2 to KC#57 can be created by the Cortex-M3 processor or an FPGA fabric master. Keys can only be created in order. If the number of valid and invalid keys up to the last valid key is M, you can request to create a key with the KEYNUM of M. A key can also be created with a lower KEYNUM. If the KEYSIZE is the same as the original key, the total number of valid keys remains the same. If the KEYSIZE is different, the service is denied.

The PUFUSERKEYADDR address is defined at the time of key enrollment (both intrinsic and extrinsic) and is used when fetching a PUF user key. It must be unique and thus different for each key. When importing the key codes, all keys are automatically generated and stored in memory pointed by these addresses.

An intrinsic key is created by ignoring USEREXTRINSICKEYADDR. An extrinsic key is created by enrolling the key pointed at by USEREXTRINSICKEYADDR. The keys can be managed and removed from the memory, if they are no longer required. Both the KC and the 4 byte address PUFUSERKEYADDR are stored in eNVM private area.

3.7.2.4 Exporting User Activation Code and All Key Codes

If SUBCMD is 3 [EXPORT_ALL_KC], the key codes from 0 to 57 are exported in encrypted form. The stored user activation code and all key codes are first XOR'ed with the one-time pad and copied to contiguous memory space, addressed by PUFUSERKEYADDR.

The following table lists the memory layout of the exported content.

Table 52 • PUFUSERACKCEXPORT Memory Layout

Offset	Length (bytes)	Field	Description
0	1192	User AC	User activation code, encrypted
1192	2	Size KC#0	Size in bytes of KC#0
1194	44	KC#0	KC#0 encrypted
1238	2	Size KC#1	Size in bytes of KC#1
1240	76	KC#1	KC#1 encrypted
1316	2	Size KC#2	Size in bytes of KC#2
1318	??	KC#2	KC#2 encrypted
...
??	2	Size KC#n	Size in bytes of KC#n
??	??	KC#n	KC#n encrypted
??	2	End marker	Is 525, one more than max

The one-time pad is stored in its place in eNVM private area and is composed of a 32 byte hash (SHA-256), and the remainder populated by random bits from the DRBG. Key codes vary in size between 44 and 524 bytes. Therefore, they are preceded by their key code size. If the key code size is 0, it means that the key code is deleted. The following two bytes are the key code size of the next key. If the key code size is 525, one more than the maximum key code size, all valid key codes are exported and this marks the end. The maximum possible size of the complete export record is $1318 + 56 * 46 = 3894$ bytes.

The EXPORT operation can only be successful, if a CREATE operation is successful previously and no prior EXPORT operation is carried out subsequently. The export option can be done only once. After exporting, the importing operation can be done as many times as required. A user key cannot be fetched after export. To get the key, perform import operation which returns all regenerated keys in the addresses provided during create time.

3.7.2.5 Importing User Activation Code and All Key Codes

If SUBCMD is 4 [IMPORT_ALL_KC], the user AC and all KCs are read from a contiguous memory space addressed by PUFUSERKEYADDR, defined in the PUFUSERKC structure. The memory space is identical to the structure, defined in [Table 52](#), page 34.

The user AC and all the KCs are then verified to be last created by this device. This verification is accomplished by first decrypting them (XOR with the one-time pad stored in its place in private eNVM) and then comparing the computed SHA-256 hash result with the one stored in private eNVM. The KC#0 and KC#1 can also be imported, but you cannot do anything with it as they are reserved. The individual private keys for KC#2 to KC#n are regenerated from the SRAM-PUF and are copied into the individual memory address spaces, and PUFUSERKEYADDR is defined by the CREATE_EXT_KC or CREATE_INT_KC SUBCMD.

Importing User Activation Code and All Key Codes operation is successful if an EXPORT operation is successful previously, and no prior CREATE operation is carried out subsequently.

As a result of the import operation, the memory space addressed by PUFUSERKEYADDR has addresses of all the user keys, and regenerated during the import operation for keys 2 to 57. If the address is 0, the key is not imported into SRAM-PUF hardware.

The following table lists the memory layout addressed by PUFUSERKEYADDR.

Table 53 • PUFUSERACKCIMPORT

Offset	Length (bytes)	Field	Description
0	4	Key#2_address	MSS/HPMS address where user Key #2 resides
4	4	Key#3 address	MSS/HPMS address where user Key #3 resides
8	4	Key#4 address	MSS/HPMS address where user Key #4 resides
...
220	4	Key#57 address	MSS/HPMS address where user Key #57 resides

3.7.2.6 Delete a User Key from the Enrolled Keys

If SUBCMD is 5 [DELETE_KC], the KC corresponding to KEYNUM is deleted from eNVM private. It deletes a user key from the enrolled keys. Though the key index is maintained, all other keys that are still valid maintain their index. For this command, the dedicated SRAM is not turned ON, if it is OFF, since SRAM-PUF operations do not perform.

The following table lists the service response for Create or Delete User Key Code and Export or Import All Service.

Table 54 • PUFUSERKC Service Response

Offset	Length (bytes)	Field	Description
0	1	CMD = 26	Command
0	1	STATUS	Command status, (see Table 55 , page 35)
1	4	PUFUSERKCPTR	Pointer to original buffer from request

Table 55 • PUFUSERKC Service Response

STATUS	Description
0	Success completion
1	eNVM MSS or HPMS error
2	PUF error, when creating
3	Invalid request or KC, when exporting or importing

Table 55 • PUFUSERKC Service Response (continued)

STATUS	Description
4	eNVM program error ¹
5	Invalid hash ²
6	Invalid user AC ³
7	eNVM verify error ¹
8	Incorrect keysize for renewing a kc
10	Private eNVM user digest mismatch
11	Invalid subcmd
12	DRBG error ⁴
127	HRESP error occurred during MSS or HPMS transfer
253	License not available in device
254	Service disabled by factory security
255	Service disabled by user security

1. These errors can occur when writing to the private eNVM for sub-commands 1, 2, 3 and 5.
2. Indeed the hash stored in private eNVM must match the SHA-256 of the decrypted AC after import.
3. Indeed an invalid AC could have been imported. Both AC and KC need to be valid.
4. This error can occur when getting the PUF seed for the DRBG initialization.

3.7.3 Key Reconstruction

The keys have to be reconstructed as they are not stored anywhere in the system. The key code along with the activation code and the SRAM-PUF start-up value are used to reconstruct the user enrolled key. The value of the key is protected until it is fetched. It is not possible to reconstruct a user key, if the EXPORT operation is performed or the key is deleted. In both cases, a status code 3 is returned.

3.7.3.1 Fetch a User PUF Key Service

The Key Reconstruction is performed using Fetch a User PUF Key Service.

The following table lists the service request format for Fetch a User PUF Key Service.

Table 56 • PUFUSERKEY Service Request

Offset	Length (bytes)	Field	Description
0	1	CMD = 27	Command
1	4	PUFUSERKEYPTR	Pointer to PUFUSERKEY structure

The following table lists the layout of the data descriptor (PUFUSERKEY) to be passed in the service request. In the data descriptor, the KEYNUM field defines the key to be reconstructed.

Table 57 • PUFUSERKEY Structure

Offset	Length (bytes)	Field	Description
0	4	PUFUSERKEYADDR	PUF User Key address [OUTPUT]
4	1	KEYNUM	Key number from 2 to 57 [INPUT]

If fetch a user PUF key service is successful, the key is written at the address pointed by PUFUSERKEYADDR. The PUFUSERKEYADDR address is returned by the service and it is the same as the one that is defined at the time of key enrollment.

The following table lists the service response for Fetch a User PUF Key Service.

Table 58 • PUFUSERKC Service Response

Offset	Length (bytes)	Field	Description
0	1	CMD = 27	Command
1	1	STATUS	Command status, (see Table 60, page 37)
2	4	PUFUSERKEYPTR	Pointer to original buffer from request

Table 59 • USERPUFKEY Status

STATUS	Description
0	Success completion
2	PUF error, when creating
3	Invalid keynum or argument or exported or invalid key
5	Invalid hash
10	Private eNVM user digest mismatch
127	HRESP error occurred during MSS or HPMS transfer
253	License not available in device
254	Service disabled by factory security
255	Service disabled by user security

3.7.4 User-Enrolled SRAM-PUF Keys for Design Security

The User PUF ECC key pair can be used for initial user key loading or device authentication similar to the Factory ECC keys or the Factory ECC PUF keys. When the keys are used, the appropriate public key is exchanged with the programmer as part of the Elliptic Curve Diffie-Hellman (ECDH) protocol.

3.7.4.1 Fetch User PUF ECC Public Key Service

Fetch a PUF ECC Public Key Service returns the User PUF ECC public key internally to the design running in the device. The public key can be used as appropriate and is certified in Public Key Infrastructure (PKI). Since this key pair can be used with the built-in key confirmation protocol, it can provide a convenient way to authenticate devices (and thus systems) with a key that is part of User-managed PKI.

The Fetch a PUF ECC Public Key Service can be used to fetch User PUF ECC Public Key from eNVM private area. The following table lists the service request format for Fetch User PUF ECC Public Key Service.

Table 60 • PUFPUBLICKEY Service Request

Offset	Length (bytes)	Field	Description
0	1	CMD = 28	Command
1	4	PUFPUBLICKEYPTR	Pointer to PUFPUBLICKEY structure

The following table lists the layout of the data descriptor (PUFPUBLICKEY) to be passed in the service request. If user PUF ECC public key is available and service is successful, it will be stored as 2x384 bit

(96 bytes) in MSS or HPMS memory pointed to by PUFPUBLICKEYADDR. This address is not changed by the call.

Table 61 • PUFPUBLICKEY Structure

Offset	Length (bytes)	Field	Description
1	4	PUFPUBLICKEYADDR	PUF Public Key address

The following table lists the service response and status codes for the Fetch User PUF ECC Public Key Service.

Table 62 • PUFPUBLICKEY Structure

Offset	Length (bytes)	Field	Description
0	1	CMD = 28	Command
1	1	STATUS	Command status (see Table 63 , page 38)
2	4	PUFPUBLICKEYPTR	Pointer to original buffer from request

Table 63 • USERPUFKEY Status

STATUS	Description
0	Success completion
3	No valid public key present in eNVM
10	Private eNVM user digest mismatch
127	HRESP error occurred during MSS or HPMS transfer
253	License not available in device
254	Service disabled by factory security
255	Service disabled by user security

3.7.5 SRAM-PUF Based True Random Number Seed Generation

From turn-on to turn-on, the SRAM generates a large number of repeatable bits-enough to be able to regenerate the static intrinsic secret each time with the help of the base activation code. However, there are also enough noisy bits from which the entropy can be harvested in order to generate a 256-bit full entropy true random seed from each turn-on event. This is done under license from Intrinsic-ID with an iRNG™ function.

3.7.5.1 Get a PUF Seed Service

The Get a PUF Seed system service generates a 256-bit true random seed using SRAM start-up values.

The following table lists the service request format for the Get a PUF Seed Service. If successful, the PUF seed is written at the address pointed to by PUFSEEDADDR.

Table 64 • PUFPUBLICKEY Structure

Offset	Length (bytes)	Field	Description
1	4	PUFPUBLICKEYADDR	PUF Public Key address

The following table lists the service response and status codes for the Fetch User PUF ECC Public Key Service.

Table 65 • PUFPUBLICKEY Structure

Offset	Length (bytes)	Field	Description
0	1	CMD = 28	Command
1	1	STATUS	Command status (see Table 63 , page 38)
2	4	PUFPUBLICKEYPTR	Pointer to original buffer from request

Table 66 • USERPUFKEY Status

STATUS	Description
0	Success completion
3	No valid public key present in eNVM
10	Private eNVM user digest mismatch
127	HRESP error occurred during MSS or HPMS transfer
253	License not available in device
254	Service disabled by factory security
255	Service disabled by user security

Table 67 • PUFSEED Status

Status	Description
0	Success completion
2	PUF error, when creating
127	HRESP error occurred during MSS or HPMS transfer
253	License not available in device
254	Service disabled by factory security
255	Service disabled by user security

3.8 Non-Deterministic Random Bit Generator (NRBG) Services

The NRBG services provide user access to the System Controller Deterministic Random Bit Generator (DRBG), a part of the NRBG. The DRBG post-processes random seeds obtained from the SmartFusion2 and IGLOO2 true random entropy source and other possible user and device supplied seed inputs. In the SmartFusion2 and IGLOO2 devices where the SRAM-PUF is available, the SRAM-PUF hardware can generate a 256-bit full entropy true random seed from each turn-on event. It is done under a license from Intrinsic-ID with an iRNG function. The true random seed is used to strengthen the NRBG by providing it as additional entropy when seeding the DRBG. In the SmartFusion2/IGLOO2 devices, there are two independent and quite different raw sources of entropy seeding the random bit generator, and providing additional security and overall improved resistance to attacks.

The NRBG is designed to be compliant with the [NIST SP800-90](#), [NIST SP800-22](#), and BIS AIS-31 standards, including all required health monitors. All commands defined in the NIST SP800-90, such as creating an instantiation, generating random bits, and reseeding, are supported at a design security strength of 256 bits. Up to 1,024 random bits can be returned per call to an instantiation. The SmartFusion2 and IGLOO2 DRBG mechanism are CTR_DRBG, as defined in [NIST SP800-90A](#). It generates random bits in a similar way that the AES counter mode generates a keystream. In addition, at each call to the generate service, a mixing operation is performed on the instantiations internal state. For more information about the NRBG, refer [UG0443: SmartFusion2 and IGLOO2 FPGA Security Best Practices User Guide](#).

The NRBG can have up to four independent instantiations of the DRBG. The System Controller reserves DRBG instantiations for the following purposes: one DRBG for use in the bitstream and related programming, passcode, and key verification protocols; one for DRBG self tests. Two independent DRBG instantiations are available for user access. Accessing a non-user DRBG instantiation is treated the same as if the DRBG instantiation did not exist that is invalid handle error. The NRBG services share a common set of service response status codes, as shown in the following figure.

Table 68 • NRBG Service Response Status Codes

STATUS	Description
0	Success (DRBGHANDLE is valid)
1	Fatal error
2	Maximum instantiations exceeded
3	Invalid handle
4	Generate request too big
5	Maximum length of additional data exceeded
127	HRESP error occurred during MSS or HPMS transfer
253	Not licensed
254	Service disabled by factory security
255	Service disabled by user security

3.8.1 Self Test Service

This service invokes all DRBG health tests. The health test function determines that the DRBG mechanism continues to function correctly. If any health test fails, a fatal error condition is entered. It requires device reset or user invocation of the DRBG reset service to recover from the fatal error.

Table 69 • DRBG Self Test Request

Offset	Length (bytes)	Field	Description
0	1	CMD = 40	Command

Table 70 • DRBG Self Test Response

Offset	Length (bytes)	Field	Description
0	1	CMD = 40	Command
1	1	STATUS	Command result status

3.8.2 Instantiate Service

The instantiate service instantiates a DRBG instance with optional personalization string. A maximum of two concurrent user instances are available. The instantiate service acquires entropy input from true entropy source, and combines it with a nonce and a personalization string to create a seed from which the DRBG initial internal state is created. The user interacts with the DRBG portion of NRBG only. It is not possible for the user to see the random seed inputs to the DRBG. The personalization string length must be in the range 0-128 bytes inclusive. Microsemi recommends the user to use the personalization string to differentiate a DRBG instantiation from another instantiation that might be created later. The personalization string bits should be unique, and the user can also include secret information, if necessary. If DRBG requires a level of protection that is greater than the intended security strength (256 bits) of the DRBG instantiation, the secret information should not be used in the personalization string. If this field is out of range, an error response is returned from the DRBG with a status code equal to 5.

Table 71 • DRBG Instantiate Request

Offset	Length (bytes)	Field	Description
0	1	CMD = 41	Command
1	4	DRBGINSTATIATEPTR	Pointer to DRBGINSTATIATE structure

The layout of the data descriptor to be passed in this service is shown in the following figure.

Table 72 • DRBGINSTATIATE Structure

Offset	Length (bytes)	Field	Description
0	4	PER_STRING_PTR	Pointer to RBG personalization string
4	1	PER_STRING_LENGTH	Length of personalization string in bytes. Length must be in the range 0-128 bytes inclusive.
5	1	RESERVED	Reserved
6	1	DRBGHANDLE	Returned DRBG handle

Table 73 • DRBG Instantiate Service Response

Offset	Length (bytes)	Field	Description
0	1	CMD = 41	Command
1	1	STATUS	Command status
2	4	DRBGINSTATIATEPTR	Pointer to DRBGINSTATIATE structure

3.8.3 Generate Service

The generate service generates pseudorandom bits upon request, using the current internal state, and generates a new internal state for the next request. It generates a random bit sequence up to 128 bytes long.

Table 74 • DRBG Generate Request

Offset	Length (bytes)	Field	Description
0	1	CMD = 42	Command
1	4	DRBGGENERATEPTR	Pointer to DRBGGENERATE structure

The following table lists the layout of the data descriptor to be passed in this service. The REQUESTEDLENGTH field must be in the range of 0–128 inclusive. An error response is returned from the DRBG with a status code equal to 5 if this field is out of range. If PRREQ is non-zero, prediction resistance is provided.

Prediction resistance means that a compromise of the DRBG internal state has no effect on the security of future DRBG outputs. That is, an adversary who is given access to all of the output sequence after the compromise cannot distinguish it from random output with less work than is associated with the security strength of the instantiation; if the adversary knows only part of the future output sequence, he cannot predict any bit of that future output sequence that he has not already known. When prediction resistance is requested, the DRBG is reseeded at the start of the generate operation.

Table 75 • DRBGGENERATE Structure

Offset	Length (bytes)	Field	Description
0	4	REQUESTEDDATAPTR	Pointer to buffer to receive generated random data
4	4	ADDITIONALINPUTPTR	Pointer to additional input data
8	1	REQUESTEDLENGTH	Number of bytes of random data to generate. Length must be in the range 0-128 bytes inclusive.
9	1	ADDITIONALINPUTLENGTH	Length of additional input in bytes. Length must be in the range 0-128 bytes inclusive.
10	1	PRREQ	Prediction resistance request. If PRREQ is non-zero, prediction resistance is provided.

Table 75 • DRBGGENERATE Structure (continued)

Offset	Length (bytes)	Field	Description
11	1	DRBGHANDLE	DRBG handle specifies which random bit generator instance is to be used to generate the random data. The value of DRBG handle is obtained as a result of a call to the DRBG instantiate service.

Table 76 • DRBG Generate Service Response

Offset	Length (bytes)	Field	Description
0	1	CMD = 42	Command
1	1	STATUS	Command status
2	4	DRBGGENERATEPTR	Pointer to DRBGGENERATE structure

3.8.3.1 Reseed Service

Reseed service acquires a new entropy input and combines it with the current internal state and any additional input that is provided to create a new seed and a new internal state. Reseed service is used to reseed the random bit generator identified by the DRBG handle that is provided in the service request. The reseed service is used to force a reseed operation.

Table 77 • DRBG Reseed Request

Offset	Length (bytes)	Field	Description
0	1	CMD = 43	Command
1	4	DRBGRESEEDPTR	Pointer to DRBGRESEED structure

The layout of the data descriptor to be passed in this service is listed in the following table.

Table 78 • DRBGRESEED Structure

Offset	Length (bytes)	Field	Description
0	4	ADDITIONALINPUTPTR	Pointer to additional input parameter in MSS or HPMS address space
4	1	ADDITIONALINPUTLENGTH	Length of additional input in bytes. Length must be in the range 0-128 bytes inclusive.
5	1	DRBGHANDLE	DRBG handle specifies which random bit generator instance to reseed. The value of DRBG handle is obtained as a result of a call to the DRBG instantiate service.

Table 79 • DRBG Reseed Service Response

Offset	Length (bytes)	Field	Description
0	1	CMD = 43	Command
1	1	STATUS	Command status
2	4	DRBGRESEEDPTR	Pointer to DRBGRESEED structure

3.8.4 Uninstantiate Service

The uninstantiate operation removes a previously instantiated DRBG and releases the associated memory resources for later use by a new instantiation. The working state of the DRBG instantiation is zeroized before being released.

Table 80 • DRBG Uninstantiate Request

Offset	Length (bytes)	Field	Description
0	1	CMD = 44	Command
1	1	DRBGHANDLE	DRBG Handle

Table 81 • Uninstantiate Response

Offset	Length (bytes)	Field	Description
0	1	CMD = 44	Command
1	1	STATUS	Command status
2	1	DRBGHANDLE	DRBG handle uninstantiated

3.8.5 Reset Service

The reset service removes all DRBG instantiations and resets the DRBG. This service is the only mechanism by which to recover from a catastrophic DRBG error without physically resetting the device. All active instantiations are automatically destroyed.

Table 82 • DRBG Reset Request

Offset	Length (bytes)	Field	Description
0	1	CMD = 45	Command

Table 83 • DRBG Reset Response

Offset	Length (bytes)	Field	Description
0	1	CMD = 45	Command
1	1	STATUS	Command status

3.9 Zeroization Service

Zeroization service is a high priority service which destroys sensitive information on the device. The zeroization service is configured by user flash bits as part of the Libero hardware flow of the design programmed into the device. For more information about zeroization, refer [UG0443: SmartFusion2 and IGLOO2 FPGA Security Best Practices User Guide](#).

Table 84 • Zeroization Request

Offset	Length (bytes)	Field	Description
0	1	CMD = F0H	Command

The state of SmartFusion2 and IGLOO2 internal memories affected by zeroization services are described as follows. One of the user configuration options is to disable zeroization. In this case a zeroization command has no effect. If one of the active zeroization configuration options is set, the data in the following memories is destroyed either by setting or clearing all bits in that memory to the same logical and physical state.

3.9.1 FPGA Fabric Configuration NVM

The state of the nonvolatile memory holding the FPGA fabric and I/O configuration is destroyed.

3.9.2 User Security Keys and Settings

The state of the nonvolatile memory holding the user keys and security settings is destroyed.

3.9.3 Factory Security Keys and Configuration Settings

Which factory non-volatile memory segments are zeroized depends upon the user-selected zeroization configuration. There is one option to disable zeroization and three active options as described in the following table.

Table 85 • Zeroization Configuration Options

Option	Description	Notes
No Zeroization	Zeroization is disabled	No effect of Zeroization service
Like New	The factory security keys and factory configuration segments data is preserved and all other internal memories content is destroyed.	Similar to a new part from the factory
Recoverable	The factory configuration segments data only preserved and all other internal memories content is destroyed.	Requires a new factory key file to recover the device
Unrecoverable	Nothing stays	Device is permanently disabled ("bricked") and cannot be recovered.

3.9.4 System Controller Memory

The crypto engine, bitstream frame buffers and all variable storage used for storing secrets are written to zero.

3.9.5 Digital Data Path

All programming/read data latches are reset or written to zero.

3.9.6 Fabric Registers

FPGA register content is destroyed.

3.9.7 Fabric SRAM

Fabric SRAMs are cleared.

3.9.8 HPMS SRAM

eSRAM_0 and eSRAM_1 are cleared to zero.

3.9.9 eNVM Memory Array and eNVM Registers

The eNVM0 and eNVM1 arrays (if present) are cleared. The eNVM registers and read buffers are cleared to zero.

3.10 Programming Service

Programming services include in-application programming (IAP) and in-system programming services. For more information about IAP and ISP, refer [UG0451: SmartFusion2 and IGLOO2 Programming User Guide](#). Programming service status codes are shown in the following table.

Table 86 • Programming Service Status Codes

STA TUS								Description
7	6	5	4	3	2	1	0	
0								Success
0								AUTHERRCODE
1	0							ERRORCODE
255								Service disabled

Table 87 • Autherrcode

Autherrcode	Description
0	No error
1	Validator or hash chaining mismatch
2	Unexpected data received
3	Invalid/corrupt encryption key
4	Invalid component header
5	Back level not satisfied
6	This bit is not used for any IAP service status.
7	DSN binding mismatch
8	Illegal component sequence
9	Insufficient device capabilities
10	Incorrect DEVICEID
11	Unsupported bitstream protocol version (regeneration required)
12	Verify not permitted on this bitstream
13	Invalid (or inaccessible) Device Certificate
127	Abort

Table 88 • Errorcode

Errorcode	Name	Description
0	SUCCESS	No error encountered (ERROR=0)
1	NVMVERIFY	Fabric verification failed
2	PROTECTED	Device security prevented operation
3	NOTENA	Programming mode not enabled
4	ENVMPROG	eNVM programming operation failed
5	ENVMVERIFY	eNVM verify operation failed
6	MSSACCESS	MSS or HPMS access error
7	PUFERROR	PUF access error
8	BADCOMPONENT	An internal error has been detected in a component payload

3.10.1 IAP Service

The IAP service requests the System Controller to reprogram the device using a bitstream already programmed into flash memory connected to MSS SPI0 or HPMS SPI. When the IAP service is invoked, the System Controller automatically reads the bitstream from the SPI flash memory connected to the MSS SPI0 or HPMS SPI and programs the device. The SPI peripheral must be configured by the user before initiating this request (also accounting for any change in clock frequency due to fabric power down). At the time the request is initiated, the user must guarantee exclusive access of MSS SPI0 or HPMS SPI. The MSS SPI0 or HPMS SPI core is accessed directly through the SII Master. A single read operation (SPI command 0BH) is sent to the SPI flash and bytes are then read via the SII Master and passed along to the programming engine.

In SmartFusion2, the Cortex-M3 processor remains active and can continue to operate during IAP service execution, but it requires to execute the code from eSRAM if the eNVMs are being reprogrammed.

Table 89 • IAP Programming Service Request

Offset	Length (bytes)	Field	Description
0	1	CMD = 20	Command
1	1	OPTIONS	Refer to Table 90 , page 47.
2	4	SPIADDR	Base address of bitstream in MSS SPI0 or HPMS SPI (MS byte ignored)

Table 90 • OPTIONS

OPTIONS							
7	6	5	4	3	2	1	0
Reserved						MODE. Refer to Table 91 , page 48.	

Note: Reserved bits indicate that even if the user writes these bits, it does not affect the functionality.

Table 91 • MODE

MODE	Operation
0	AUTHENTICATE
1	PROGRAM
2	VERIFY

If MODE is AUTHENTICATE, the fabric and MSS/HPMS are left operational while the bitstream is authenticated. If MODE is VERIFY, the device is automatically placed in the Flash*Freeze state for the duration of the verify operation and automatically awakened upon completion. The service response is sent after the Flash*Freeze exit stage.

If MODE is PROGRAM, the device is automatically placed in the Flash*Freeze state before programming commences. The MSS and Cortex-M3 processor or HPMS gets reset upon completion of the PROGRAM operation. The service response is sent after the MSS/HPMS reset. If the programming operation is successful, the user design is automatically restarted to initialize the new version of the design. If the programming operation fails, the design is left in the Flash*Freeze configuration and allows to recover from the problem. In this mode, the device components (FPGA fabric, eNVM, and security settings) are programmed based on the payload data available in a bitstream file that is stored in the SPI Flash memory. For example, if the bitstream file contains only FPGA fabric payload data, then this service programs only FPGA fabric portion of the device.

Table 92 • IAP Response

Offset	Length (bytes)	Field	Description
0	1	CMD = 20	Command
1	1	STATUS	Command status

3.10.2 ISP Service

The ISP service allows the Cortex-M3 processor to directly provide a bitstream for programming. The ISP service is unique in that all communication utilizes the COMM_BLK interface such that the system controller receives the entire bitstream as a continuous stream of bytes.

The Cortex-M3 processor must continue to operate for the duration of the programming operation. If the bitstream reprograms the code region being used, the Cortex-M3 processor must copy its code to RAM and execute it from there.

Table 93 • ISP Programming Service Request

Offset	Length (bytes)	Field	Description
0	1	CMD = 21	Command
1	1	OPTIONS	See Table 90 , page 47
2	BSLENGTH	BSDATA	Bitstream Data

Since the size of the bitstream is variable, the length of the bitstream data, BSLENGTH, can only be determined by analyzing all component headers in the bitstream. Since the component header formats

are fixed, the Cortex-M3 processor can determine how much data it needs to send for each component. Any excess data received is ignored.

Table 94 • ISP Response

Offset	Length (bytes)	Field	Description
0	1	CMD = 21	Command
1	1	STATUS	Command Status

3.11 NVM Data Integrity Check Service

The NVM data integrity check service recalculates and compares cryptographic digests of the selected NVM component(s)—fabric, ENVM0, and ENVM1—to those previously computed and saved in NVM.

The contents of all the nonvolatile configuration memory segments are digested (hashed) using the SHA-256 algorithm. The results are compared to values stored in dedicated nonvolatile memory words located in each segment. If the contents are unchanged from when the digests were computed and stored during the original programming steps—that is, if the current and stored digests match—the test will pass; otherwise a failure is flagged.

The OPTIONS field in the NVM data integrity check service request selects the NVM components (fabric configuration, eNVM0, and eNVM1) for data integrity check, as shown in the following table.

Table 95 • NVM Data Integrity Check Service Request

Offset	Length (bytes)	Field	Description
0	1	CMD = 23	Command
1	1	OPTIONS	Service options. See Table 96 , page 49

Table 96 • OPTIONS

7	6	5	4	3	2	1	0
Reserved					ENVM1	ENVM0	FABRIC

Note: Reserved bits indicate that even if the user writes these bits, it does not affect the functionality.

If the bit FABRIC is set to 1 then FPGA fabric configuration digest is performed. For FPGA fabric configuration digest, if the fabric is powered up, it is first placed in the Flash*Freeze state. The requester must, therefore, be prepared for an immediate Flash*Freeze shutdown if the fabric digest is to be checked. The Flash*Freeze shutdown follows the exact shutdown sequence for a normal Flash*Freeze request, except that the wake-up mechanism is not armed. The wake-up happens automatically after completion of the data integrity check service.

If the bits ENVM0/1 are set to 1 then the corresponding eNVM digests are checked. The eNVM digests are computed over eNVM pages that have been declared as static by the user, as if those pages were ROM. Pages that are not flagged as ROM (that is, as write-protected in the original programming bitstream) are not included in the eNVM digest calculation.

If no digest is present, the result is a digest mismatch for the requested NVM block.

Table 97 • NVM Data Integrity Check Response

Offset	Length (bytes)	Field	Description
0	1	CMD = 23	Command
1	1	DIGESTERR	Pass/fail flags. See Table 98 , page 50

If a digest mismatch occurs, DIGESTERR indicates which of the selected digests are in error.

Table 98 • DIGESTERR

Bit Number	Name	Description
[7:3]	RESERVED	Reserved
2	ENVM1ERR	0 - ENVM1 digest check passed 1 - ENVM1 digest mismatch
1	ENVM0ERR	0 - ENVM0 digest check passed 1 - ENVM0 digest mismatch
0	FABRICERR	0 - FPGA fabric configuration digest check passed 1 - FPGA fabric configuration digest mismatch

Note: NVM data integrity check can be automatically performed on power-up by setting a flash bit at programming stage using Libero Security Policy Manager in the Libero SoC design software. For more information, refer [Libero user guide](#).

3.12 Unrecognized Command Response

If an unrecognized command is received, which is not listed in [Table 3](#), page 15, the System Controller generates a response which includes the command value and status code equal to 252. The status code 252 indicates an unrecognized command.

Table 99 • Unrecognized Command Message

Offset	Length (bytes)	Field	Description
0	1	CMD	Command
1	1	STATUS = 252	Unrecognized command

3.13 Asynchronous Messages

Asynchronous messages are sent when certain events are detected, allowing the user design or System Controller to take remedial or defensive action. No response is required from the user design or System Controller. These messages may simply be discarded if desired.

3.13.1 Power-on-Reset (POR) Digest Error

The user may choose to have the device verify stored digests as part of its start-up sequence. When this feature is enabled, user-specified NVM data digests are recalculated and compared against their stored values and any inconsistency results in a POR digest error message. Refer to the following table.

If an error is detected, the system is still allowed to boot as normal, but messages are sent to the fabric indicating the failure.

Table 100 • POR Digest Error Message

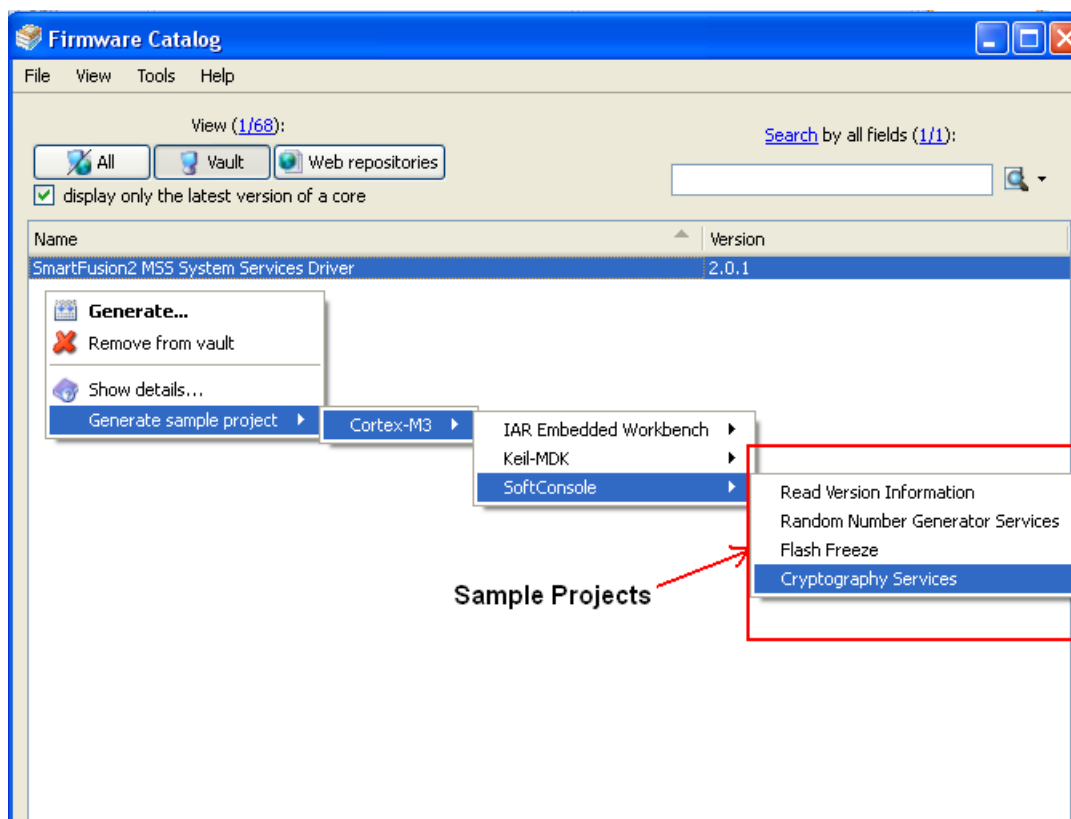
Offset	Length (bytes)	Field	Description
0	1	CMD = 241	Command
1	1	DIGESTERR	See Table 98 , page 50

3.14 How to Use System Services in SmartFusion2

Microsemi provides system services firmware drivers to access the system services implemented by the System Controller. The SmartFusion2 system services driver can be downloaded from the Firmware Catalog. The system services driver provides APIs to access the system services. The system services APIs must be called in the user application code to access system services. The system services driver provides the `MSS_SYS_init()` API to initialize the communication with the System Controller. The `MSS_SYS_init()` function can be used to register a system services event handler. Each system service API returns a service response to know the status of the service. For the list of system services APIs and their descriptions, refer [UG0837: IGLOO2 and SmartFusion2 FPGA System Services Simulation User Guide](#).

The system services driver package includes sample projects to show the usage of system services driver. The sample projects are available for three different tool chains: IAR Embedded Work, Keil-MDK, and SoftConsole. The sample project can be generated by right clicking on the system services driver and selecting the **Generate sample project**, as shown in the following figure.

Figure 11 • System Services Sample Projects



3.14.1 Use Model 1: Fetching Device and Design Information

This use model shows the usage of System Services driver to fetch the device and design information. System Services drivers provide the following APIs to execute system services for fetching the device and design information.

Table 101 • System Services APIs for Fetching Device and Design Information

System Services API	Description
MSS_SYS_get_serial_number()	Fetches the 128-bit device serial number and writes at address passed as the function argument.
MSS_SYS_get_user_code()	Fetches the 32-bit JTAG user code and writes at address passed as the function argument.
MSS_SYS_get_design_version()	Fetches the 16-bit design version and writes at address passed as the function argument.
MSS_SYS_get_device_certificate()	Fetches the 768 bytes device certificate from the eNVM and writes at address passed as the function argument.

The following example application code fetches the device serial number and displays it on the UART terminal. This application code is targeted at SmartFusion2 design which has MMUART0 enabled and connected to the host PC for serial communication. The same application code can be extended to fetch the JTAG user code, design version and device certificate by calling the corresponding APIs. Refer to the sample project provided with the system services driver for a complete project.

```
#include <stdio.h>

#include "drivers/mss_sys_services/mss_sys_services.h"
#include "drivers/mss_uart/mss_uart.h"

/*=====
=====

    Private functions.
*/

static void display_hex_values
(
    const uint8_t * in_buffer, uint32_t byte_length
);

/*=====
=====

    UART selection. MMUART0 is selected.
*/
mss_uart_instance_t * const gp_my_uart = &g_mss_uart0;

/*=====
=====

    Main function.
*/
```

```

int main()
{
    uint8_t serial_number[16];
    uint8_t status;

    /*-----
    ---
    * Initilize the system services communication with the System Controller.
    */
    MSS_SYS_init(MSS_SYS_NO_EVENT_HANDLER);

    /*-----
    ---
    * Initilize the MMUART with required configuration
    */
    MSS_UART_init(gp_my_uart,
                  MSS_UART_57600_BAUD,
    MSS_UART_DATA_8_BITS | MSS_UART_NO_PARITY | MSS_UART_ONE_STOP_BIT);

    /*-----
    ---
    * Fetch the Device Serial Number (DSN).
    */
    status = MSS_SYS_get_serial_number(serial_number);

    /*-----
    ---
    * Check the service status for SUCCESS and then display the DSN on UART
    terminal
    */
    if(MSS_SYS_SUCCESS == status)
    {
        MSS_UART_polled_tx_string(gp_my_uart,
                                (const uint8_t*)"Device serial number: ");
        display_hex_values(serial_number, sizeof(serial_number));
    }

    for(;;)
    {
        ;
    }
}

```

```

    }

    return 0;

}

/*=====
=====
    Display content of buffer passed as parameter as hex values
    */
static void display_hex_values
(
    const uint8_t * in_buffer,
    uint32_t byte_length
)
{
    uint8_t display_buffer[128];
    uint32_t inc;

    if(byte_length > 16u)
    {
        MSS_UART_polled_tx_string( gp_my_uart, (const uint8_t*)"\\r\\n" );
    }

    for(inc = 0; inc < byte_length; ++inc)
    {
        if((inc > 1u) &&(0u == (inc % 16u)))
        {
            MSS_UART_polled_tx_string( gp_my_uart, (const uint8_t*)"\\r\\n" );
        }

        snprintf((char *)display_buffer, sizeof(display_buffer), "%02x ",
            in_buffer[inc]);

        MSS_UART_polled_tx_string(gp_my_uart, display_buffer);
    }
}

```

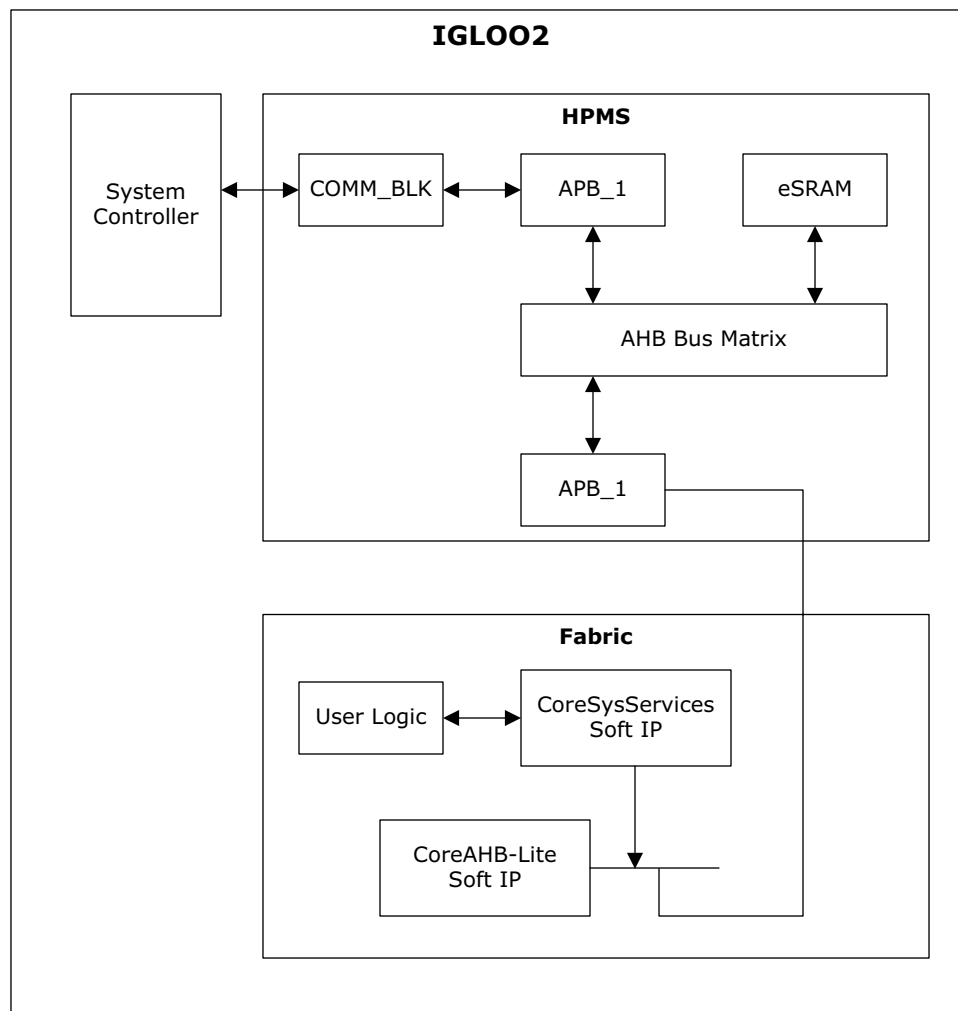
Note: For more information, refer to the [Related Applications](#), page 63.

3.15 How to Use System Services in IGLOO2

This section describes how to use system services. Microsemi provides CoreSysServices soft IP to access the system services implemented by the System Controller. The CoreSysServices soft IP provides a user interface for each of the system services and an Advanced High-performance Bus (AHB)-Lite master interface on the Fabric Interface Controller (FIC) side. The core communicates with the COMM_BLK through the FIC_0 interface.

CoreSysServices soft IP decodes the command values received from the user logic and translates the user logic transactions to the AHB-Lite master transactions. For more information about the CoreSysServices soft IP, refer [CoreSysServices Handbook](#). The following figure shows the functional block diagram for accessing system services.

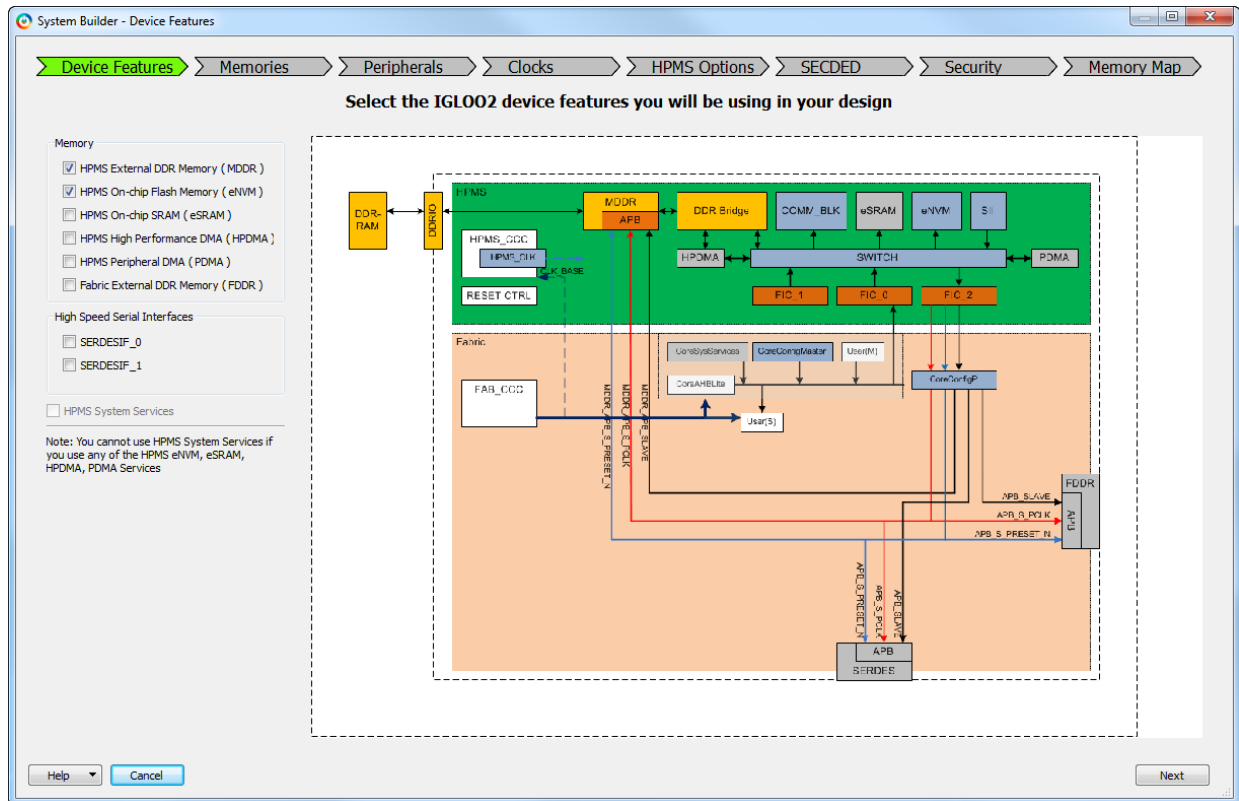
Figure 12 • Functional Block Diagram for Accessing System Services



To configure the IGLOO2 device features and to build a complete IGLOO2 system, use the **System Builder** graphical design wizard in the Libero software.

The following figure shows the initial **System Builder** window where you can select the device features that you require. For details on how to launch the **System Builder** wizard and a detailed information on how to use it, refer [IGLOO2 System Builder User Guide](#).

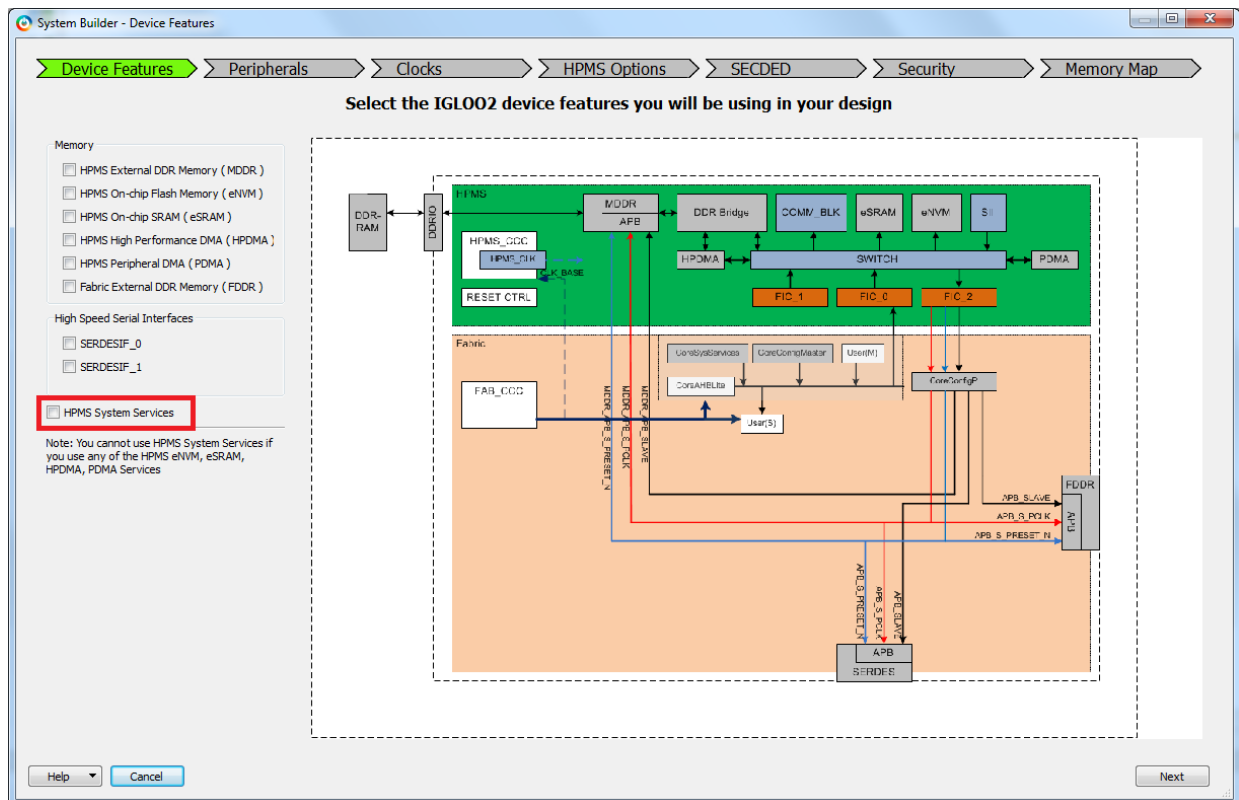
Figure 13 • System Builder Window



3.15.1 Configuring System Services

1. Check the **HPMS System Services** check box under the **Device Features** tab and leave the other check boxes unchecked. The following figure shows the **System Builder - Device Features** tab.

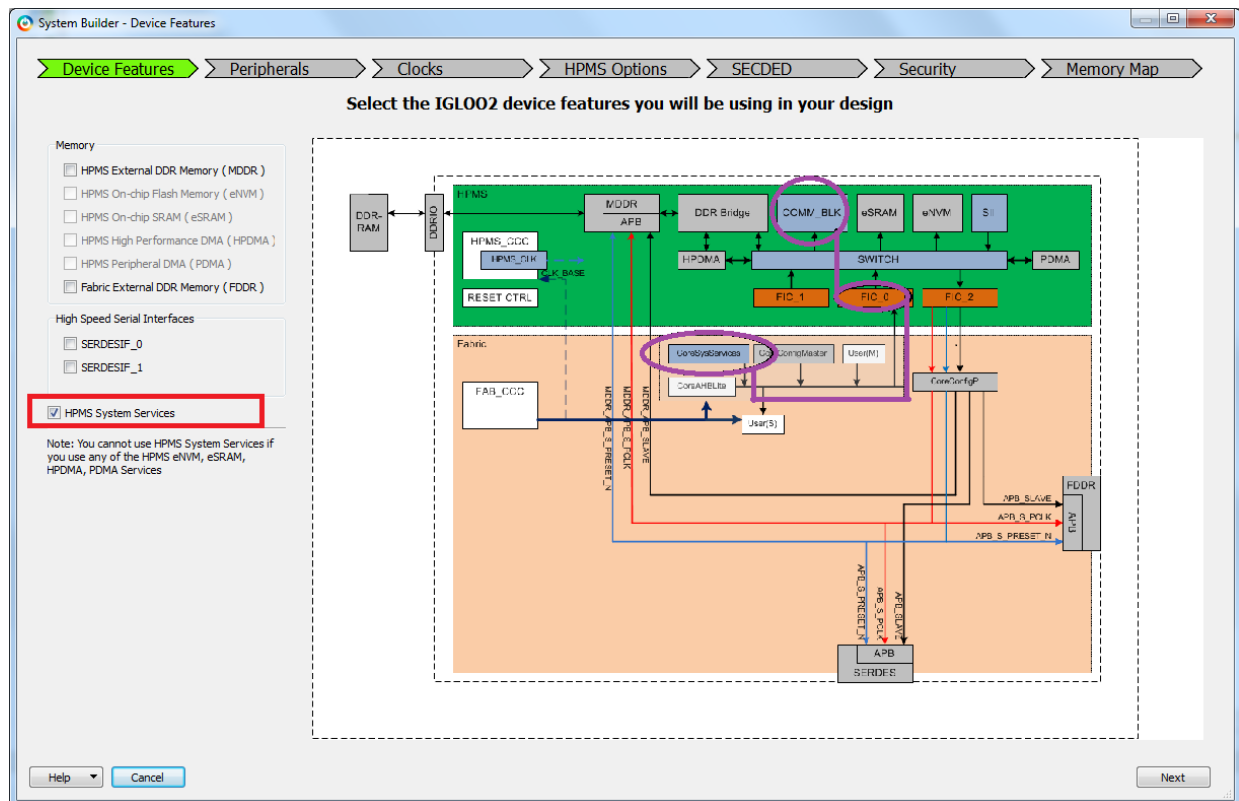
Figure 14 • System Builder - Device Features Tab



2. Checking the **HPMS System Services** check box establishes a path for connecting the CoreSysServices soft IP to the COMM_BLK through the FIC_0 interface. After selecting the CoreSysServices block, the color of the CoreSysServices block changes to a darker color indicating that the function is enabled in the System Builder. The following figure shows the path between the COMM_BLK and the CoreSysServices soft IP.

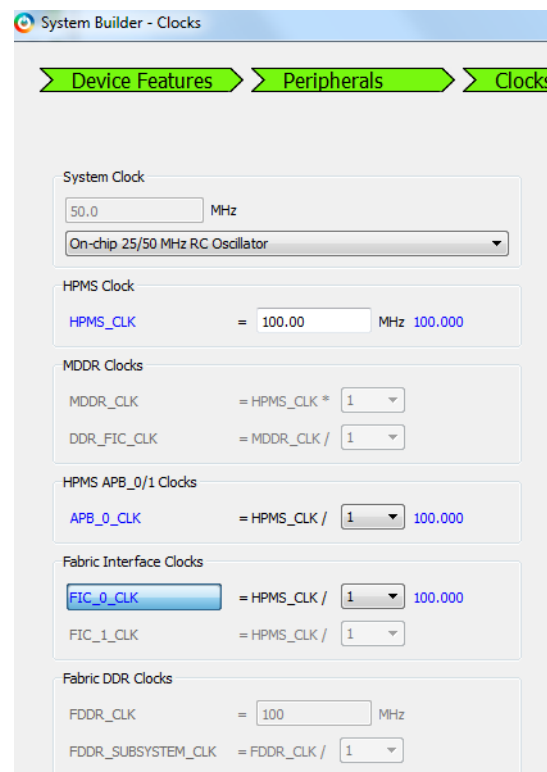
Note: The **System Builder** does not automatically instantiate the CoreSysServices soft IP but allows you to connect it with the FIC_0 master interface port.

Figure 15 • CoreSysServices IP to COMM_BLK Path



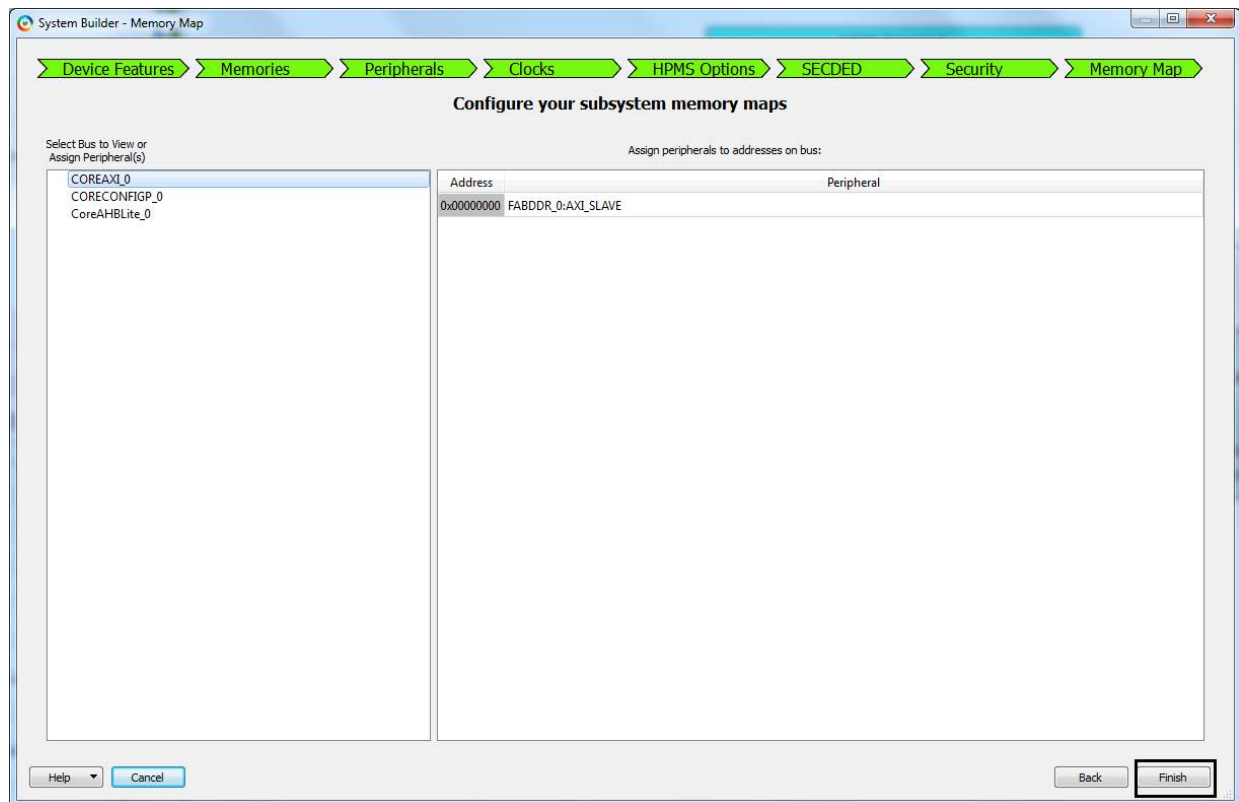
3. Go to **Clocks** tab and configure the required FIC_0 clock frequency. The following figure shows the **System Builder - Clocks** tab.

Figure 16 • System Builder - Clock Tab



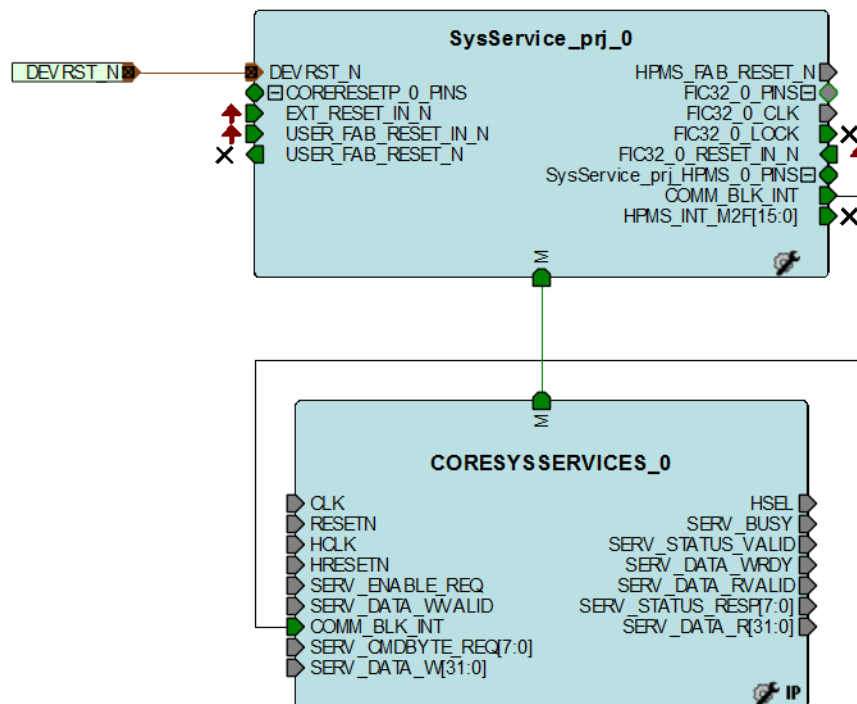
4. Navigate to the **Memory Map** tab giving the required data in the rest of the **System Builder** tabs. Click **Finish** to complete the system builder flow. The following figure shows the **System Builder - Memory Map** tab.

Figure 17 • System Builder - Memory Map Tab



5. Instantiate the CoreSysServices soft IP in SmartDesign canvas and configure it for the required System Services features.
6. Connect
 - The CoreSysServices master interface port to the system builder generated a top-level component master interface port.
 - The COMM_BLK_INT signal of CoreSysServices to the COMM_BLK_INT signal of system builder generated a top-level component.

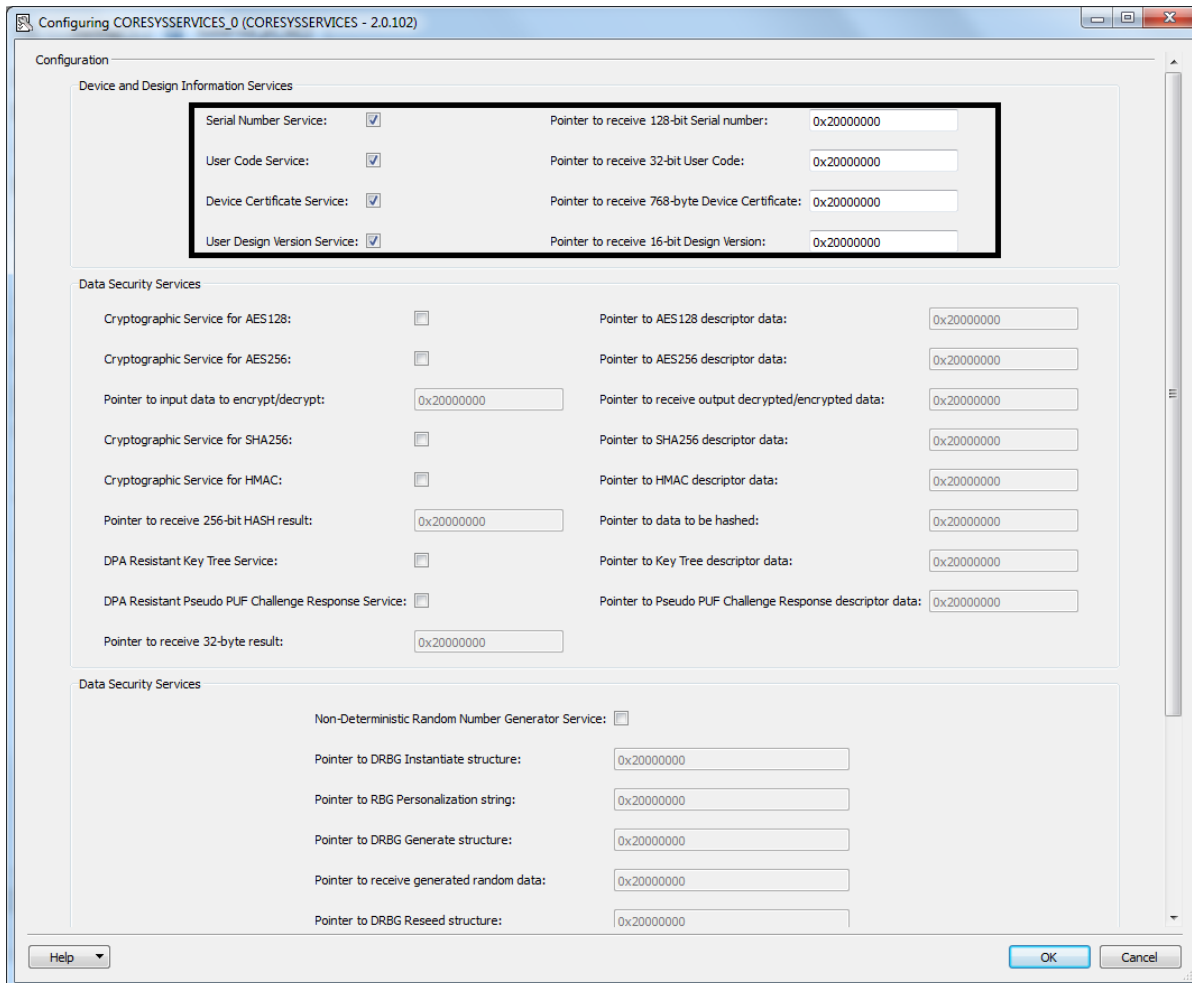
Figure 18 • SmartDesign Connections



3.15.2 Fetching Device and Design information

This section describes the usage of the CoreSysServices soft IP to fetch the device and design information.

1. Right-click on the **CORESYS SERVICES_0** instance to open the Configuring **CORESYS SERVICES_0** dialog.
2. Check all the check boxes under **Device and Design Information Services** to configure the CoreSysServices soft IP for **Device and Design Information services**. The following figure shows the **Device and Design Information Services** dialog.

Figure 19 • CORESYS SERVICES_0 Configuration


Configuring CORESYS SERVICES_0 (CORESYS SERVICES - 2.0.102)

Configuration

Device and Design Information Services

Serial Number Service: <input checked="" type="checkbox"/>	Pointer to receive 128-bit Serial number: 0x20000000
User Code Service: <input checked="" type="checkbox"/>	Pointer to receive 32-bit User Code: 0x20000000
Device Certificate Service: <input checked="" type="checkbox"/>	Pointer to receive 768-byte Device Certificate: 0x20000000
User Design Version Service: <input checked="" type="checkbox"/>	Pointer to receive 16-bit Design Version: 0x20000000

Data Security Services

Cryptographic Service for AES128: <input type="checkbox"/>	Pointer to AES128 descriptor data: 0x20000000
Cryptographic Service for AES256: <input type="checkbox"/>	Pointer to AES256 descriptor data: 0x20000000
Pointer to input data to encrypt/decrypt: 0x20000000	Pointer to receive output decrypted/encrypted data: 0x20000000
Cryptographic Service for SHA256: <input type="checkbox"/>	Pointer to SHA256 descriptor data: 0x20000000
Cryptographic Service for HMAC: <input type="checkbox"/>	Pointer to HMAC descriptor data: 0x20000000
Pointer to receive 256-bit HASH result: 0x20000000	Pointer to data to be hashed: 0x20000000
DPA Resistant Key Tree Service: <input type="checkbox"/>	Pointer to Key Tree descriptor data: 0x20000000
DPA Resistant Pseudo PUF Challenge Response Service: <input type="checkbox"/>	Pointer to Pseudo PUF Challenge Response descriptor data: 0x20000000
Pointer to receive 32-byte result: 0x20000000	

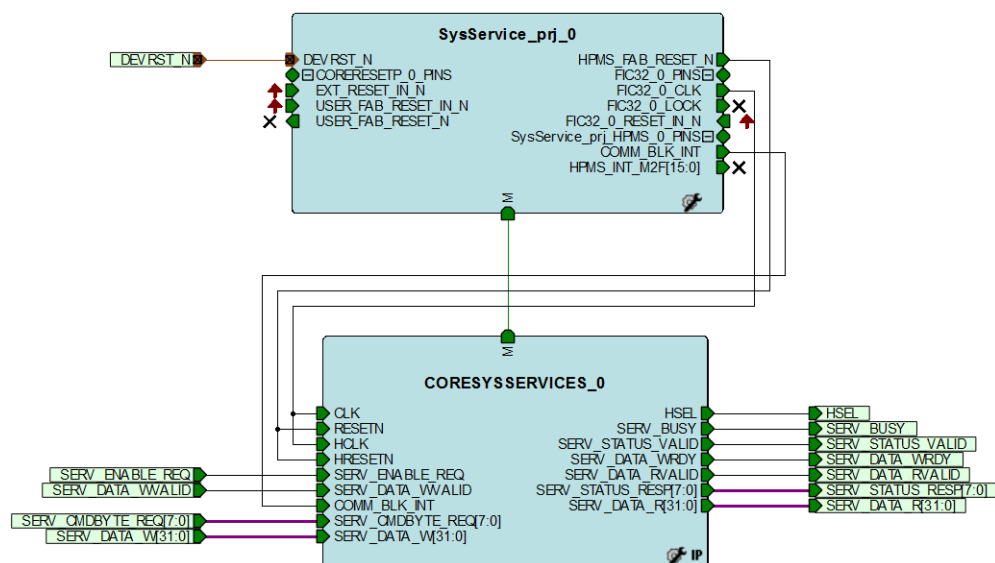
Data Security Services

Non-Deterministic Random Number Generator Service: ☐

Pointer to DRBG Instantiate structure: 0x20000000
Pointer to RBG Personalization string: 0x20000000
Pointer to DRBG Generate structure: 0x20000000
Pointer to receive generated random data: 0x20000000
Pointer to DRBG Reseed structure: 0x20000000

Help OK Cancel

3. Make the SmartDesign connections, as shown in the following figure.

Figure 20 • HPMS Subsystem and CoreSysServices IP Connections

4. The CoreSysServices soft IP provides the following user interface signals and Command codes to execute the system services and to fetch the device and design information.

The following table lists the required user interface signals that the user is responsible for while executing the system services for the **Device and Design Information Services**.

Table 102 • User Interface Signals

Port	Type	Description
Handshaking Signals		
SERV_BUSY	Out	Service busy When de-asserted, indicates that no service has been requested. When asserted, indicates that the current service is in progress.
SERV_DATA_RVALID	Out	Data Valid for User Read Data
SERV_DATA_R[31:0]	Out	User Read Data
User Interface Signals: Request Signals		
SERV_ENABLE_REQ	In	Active high request to start the service
SERV_CMDBYTE_REQ[7:0]	In	Command byte to indicate the type of system service. Command byte for each of the requested
User Interface Signals: Response Signals		
SERV_STATUS_RESP[7:0]	Out	Response status of the requested service
SERV_STATUS_VALID	Out	Response Status valid

For more information about port description and timing diagrams, refer [CoreSysServices Handbook](#).

The following table lists the Command Codes for Device and Design Information Services.

Table 103 • Command Codes for Device and Design Information Services

System Service Name	Command Value
Serial Number Service	1
USERCODE Service	4
Device Certificate Service	0
User Design Version Service	5

For other services Command values, refer [Table 3](#), page 15.

5. The user logic must adhere to the following:
- Monitor the SERV_BUSY signal. When this signal is Low, a service can be requested.
 - Drive the SERV_ENABLE_REQ signal HIGH and write corresponding command values to the SERV_CMDBYTE_REQ[7:0] signal.
 - Capture the requested service data SERV_DATA_R[31:0] when SERV_DATA_RVALID signal is HIGH.
 - Capture the SERV_STATUS_RESP[7:0] response signal when SERV_STATUS_VALID signal is HIGH. Check the response and verify the status.

3.15.3 Related Applications

For more information, refer to the following application notes:

- [*AC436: Using Device Certificate System Service in SmartFusion2*](#)
- [*AC433: Using Zeroization in SmartFusion2 and IGLOO2 Devices*](#)
- [*AC432: Using SHA-256 System Services in the SmartFusion2 and IGLOO2*](#)
- [*AC434: Using SRAM PUF System Service in SmartFusion2*](#)
- [*AC435: Using ECC System Service in SmartFusion2*](#)
- [*AC410: Using AES System Services in SmartFusion2 and IGLOO2 Devices*](#)
- [*AC407: Using NRBG Services in SmartFusion2 SoC and IGLOO2 FPGA Devices*](#)