SmartFusion 2 SoC and IGLOO 2 FPGA System Controller User Guide

Introduction (Ask a Question)

The SmartFusion® 2 family includes the Arm® Cortex™ -M3 processor and a rich set of analog peripherals, while the IGLOO® 2 family focuses on low-power, cost-efficient designs without the embedded processor, targeting applications where programmable logic and mixed-signal functionality are paramount. Both families feature advanced power management capabilities and offer high levels of integration, making them ideal for modern applications that require real-time performance and energy efficiency.

Table of Contents

1. System Controller (Ask a Question)

This chapter describes various system services implemented by the SmartFusion® 2 and IGLOO® 2 System Controller.

1.1 Overview (Ask a Question)

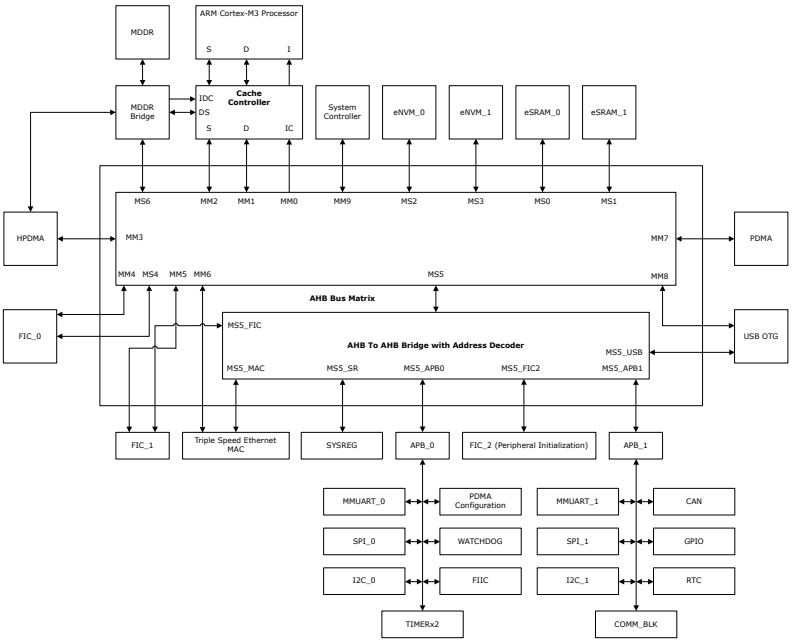
The System Controller manages the programming of the device and handles the system service requests. The System Controller serves as the base on which the following system services are made available with SmartFusion 2 and IGLOO 2 devices as listed:

- 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

For more information, see System Services.

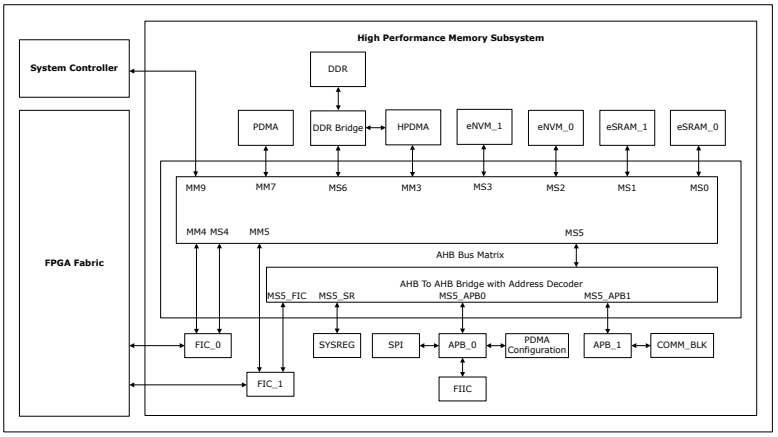
The following figure shows the connectivity of the SmartFusion 2 System Controller to the Advanced High-performance Bus (AHB) matrix.

Figure 1-1. SmartFusion 2—System Controller Interfacing with AHB Bus Matrix



The following figure shows the connectivity of the IGLOO 2 System Controller to the AHB bus matrix.

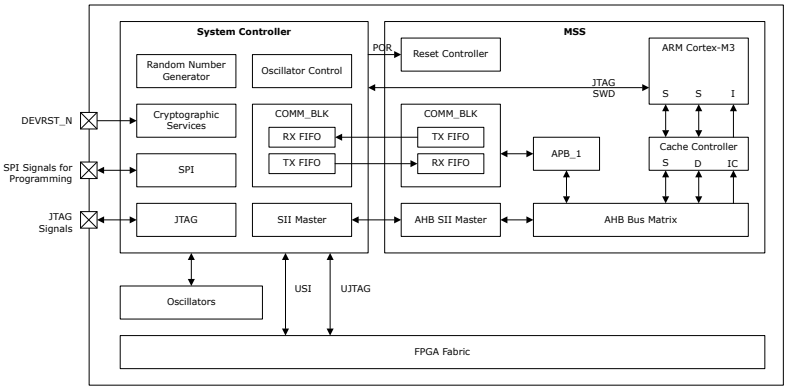
Figure 1-2. IGLOO 2—System Controller Interfacing with AHB Bus Matrix



1.2 Functional Description (Ask a Question)

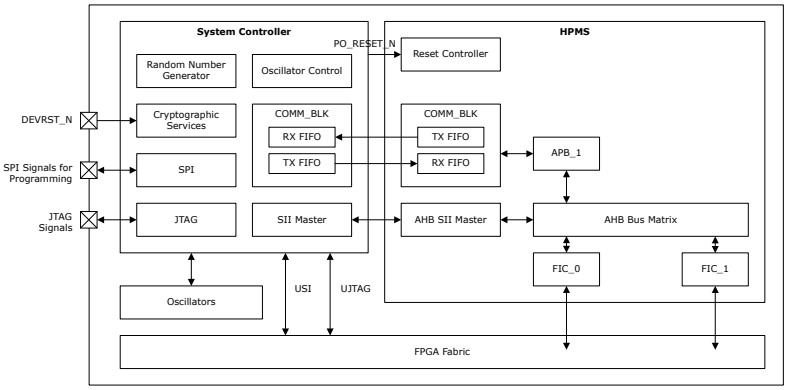
The following figure shows the interfacing of the SmartFusion® 2 System Controller with Microcontroller Sub-System (MSS) and the FPGA fabric.

Figure 1-3. SmartFusion 2— Interfacing of the System Controller with MSS and FPGA Fabric



The following figure shows the interfacing of the IGLOO® 2 System Controller with the High-Performance Mixed Signal (HPMS) and the FPGA fabric.

Figure 1-4. IGLOO 2—Interfacing of the System Controller with HPMS and FPGA Fabric



The SmartFusion 2 and IGLOO 2 System Controllers consist of the following subsystems and interfaces as shown in the preceding figures.

1.2.1 Subsystems (Ask a Question)

This section discusses the following subsystems:

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

1.2.2 Interfaces (Ask a Question)

This section discusses the following interfaces:

- DEVRST_N
- SPI signals for programming
- JTAG signals
- User JTAG
- User Services Interface (USI)

1.2.3 System IP Interface (SII) Master (Ask a Question)

The System IP Interface (SII) master connects the system controller with all the internal elements. It transfers data to and from the MSS or High Performance Memory Subsystem (HPMS) memory space by the system controller for system services. It is also used for factory tests but not available for customer.

1.2.4 Communication Block (COMM_BLK) (Ask a Question)

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, see the "Communication Block" chapter in the UG0448: IGLOO2 High Performance Memory Subsystem User Guide and UG0331: SmartFusion 2 Microcontroller Subsystem User Guide. The `COMM_BLK` calls the system services. System services are implemented using API functions. For more information, see System Services.

1.2.5 Oscillator Control (Ask a Question)

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. If the subsystem is not properly configured, the oscillator turns 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, see the "Oscillator Configuration" section in the UG0449 User Guide SmartFusion2 and IGLOO2 Clocking Resources.

1.2.6 Random Number Generator (Ask a Question)

The system controller contains a random number generator block which is available for user cryptographicservices. The Deterministic Random Bit Generator (DRBG) is implemented as defined

in "*National Institute of Standards and Technology (NIST) Special Publication 800-90*". It has 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, see the "Non-Deterministic Random Bit Generator (NRBG)" section in the UG0443:UG0443 User Guide SmartFusion2 and IGLOO2 FPGA Security and Best Practices .

1.2.7 Cyptographic Services (Ask a Question)

The system controller contains an Advanced Encryption Standard (AES) encryption and decryption engine which can be dynamically configured for key lengths of 128 or 256 bits. An NIST-approved SHA-256 authentication algorithm is also provided. For more information, see the "Other Cryptographic Services" section in the UG0443 User Guide SmartFusion2 and IGLOO2 FPGA Security and Best Practices .

1.2.8 JTAG (Ask a Question)

The system controller implements the functionality of a JTAG client, 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, see UG0451 User Guide SmartFusion2 and IGLOO2 Programming.
- Serial Wire JTAG Debug Port (SWJ-DP)/Serial Wire Debug (SWD) for the Cortex-M3 processor
- For more information, see the "Debug Port" section in the UG0331:SmartFusion2Microcontroller Subsystem User Guide.
- UJTAG
- Boundary scan

1.2.8.1 Boundary Scan (Ask a Question)

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.

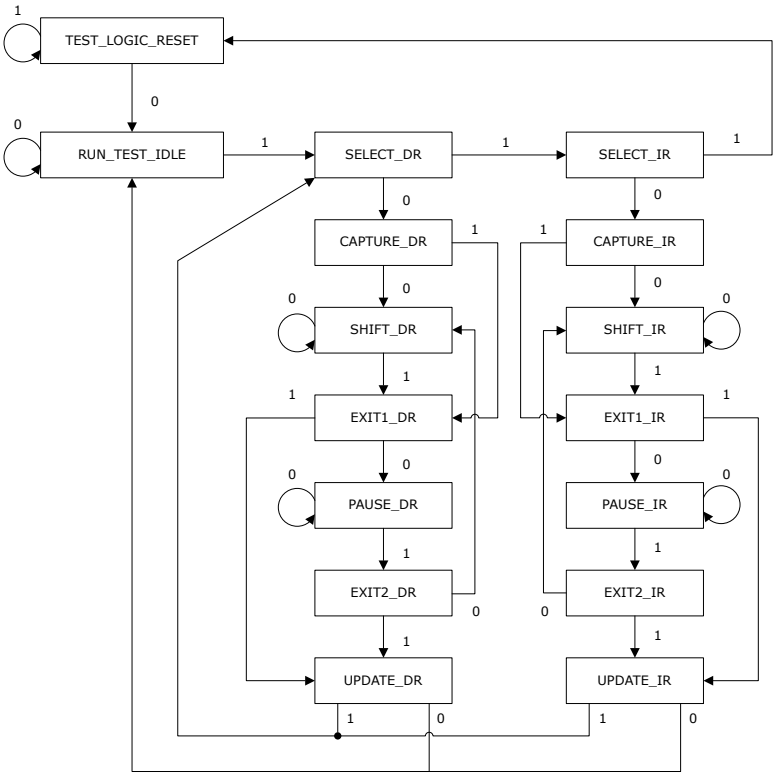
SmartFusion 2 and IGLOO 2 devices support three types of test data registers: bypass, device identification, and boundary scan. The BYPASSregister is selected when no other register needs to be accessed in a device. This speeds up the test data transfer to the other devices in a test data path. The 32-bit 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.

1.2.8.1.1 TAP Controller State Machine (Ask a Question)

The TAP controller is a 4-bit state machine (16 states) that operates, see Figure5. The 1s and 0s represent the values that must be present on Test Mode Select (TMS) at a rising edge of Test Clock (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 (TCK and TMS) 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 1-5. TAP Controller State Machine



1.2.8.2 Boundary Scan Opcodes (Ask a Question)

SmartFusion 2 and IGLOO 2 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 1-1. Opcodes Description

Instruction	HexOpcode
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, see the "JTAG I/O" section in the UG0445: User Guide SmartFusion2 SoC FPGA and IGLOO2 FPGA Fabric.

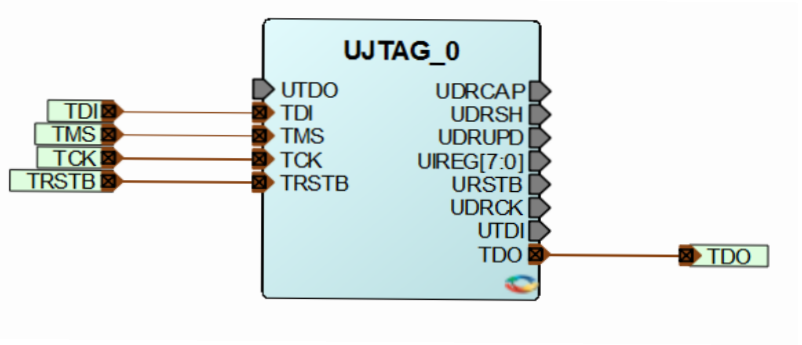
1.2.9 User JTAG (Ask a Question)

The user JTAG (UJTAG) interface is an extension to the external JTAG port of SmartFusion 2 and IGLOO 2 devices, controlled by the TAP controller when it is not performing JTAG programming. It shifts 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, see Flash UJTAG Application Note.

1.2.9.1 UJTAG Macro (Ask a Question)

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 1-6. UJTAG Macro



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

Table 1-2. 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 T Contoller 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.

1.2.9.2 UJTAG Operation (Ask a Question)

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, see TAP Controller State Machine.

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 provides 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 indicates that UTDI contains the data bitstream, and UDRUPD is a candidate for the end-of-data-stream flag.

1.2.9.3 Typical UJTAG Applications (Ask a Question)

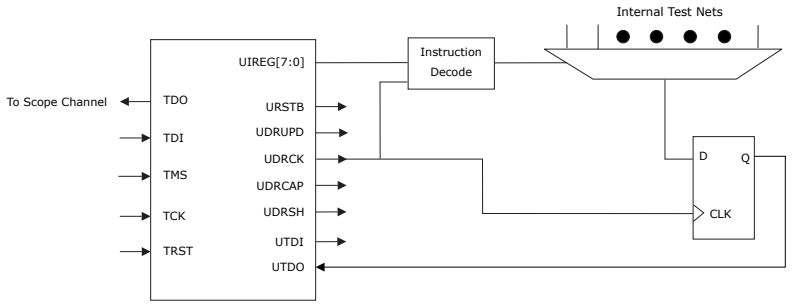
Bidirectional access to the JTAG port from the FPGA fabric—without putting the device into test mode—creates flexibility for implementing a variety of applications. This section describes one of Such aaplication. This is based on importing/exporting data through the UJTAG macro. However, the possible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

1.2.9.3.1 Silicon Testing and Debugging (Ask a Question)

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 1-7. UJTAG Usage Example in Test and Debug Applications



Multiple test nets are brought into an internal Multiplexer (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.

1.2.9.4 How to Use UJTAG (Ask a Question)

For more information on how to use UJTAG, see How To Use UJTAG application note.

1.2.10 Dedicated Programming SPI Peripheral (Ask a Question)

The system controller contains an SPI block that is dedicated for programming. For more information, see "SPI-slave Programming" chapter in the UG0451 User Guide SmartFusion2 and IGLOO2 Programming.

1.2.11 Device Reset (Ask a Question)

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

For more information, see "Reset Controller" chapter in the UG0448:IGLOO2 FPGA High Performance Memory Subsystem User Guide.

1.2.12 USI Interface (Ask a Question)

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.

1.2.12.1 SYSRESET (Ask a Question)

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 DEVRST_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, see "Reset Controller" chapter in the UG0448:IGLOO2 FPGA High Performance Memory Subsystem User Guide.

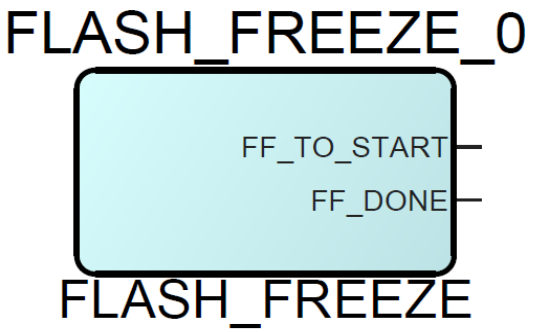
1.2.12.2 Flash*Freeze Signals (Ask a Question)

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

- FF_TO_START is asserted by the system controller to indicate that the Flash*Freeze service is about to start. Only 10 µs are available to do housekeeping before the core is turned off. Microchip 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 1-8. FLASH_FREEZE Macro



1.2.13 Clock Requirements (Ask a Question)

The system controller is clocked by the on-chip 50 MHz RC oscillator.

Itis 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.

1.2.14 System Controller Suspend Mode (Ask a Question)

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.

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

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

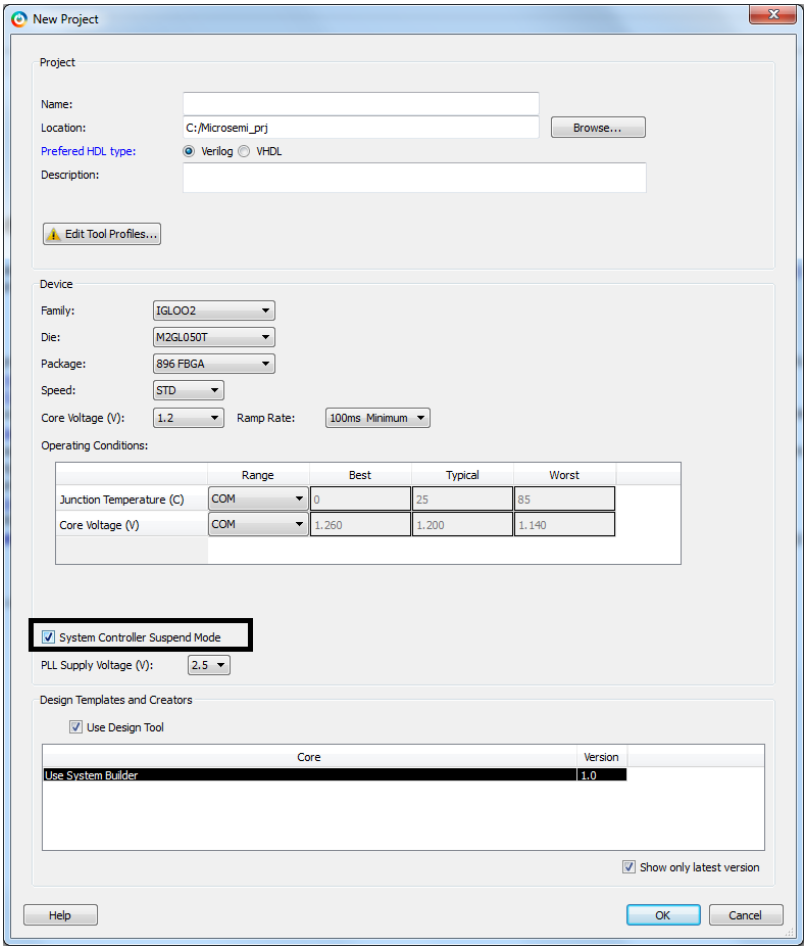
In the system controller suspend mode, the device can be reprogrammed or debugged using the JTAG port if the TRSTB pin is high. If the TRSTB pin is low, all the other JTAG input signals are blocked from activating the system controller. For prototyping or debugging, the device can be forced out of suspend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

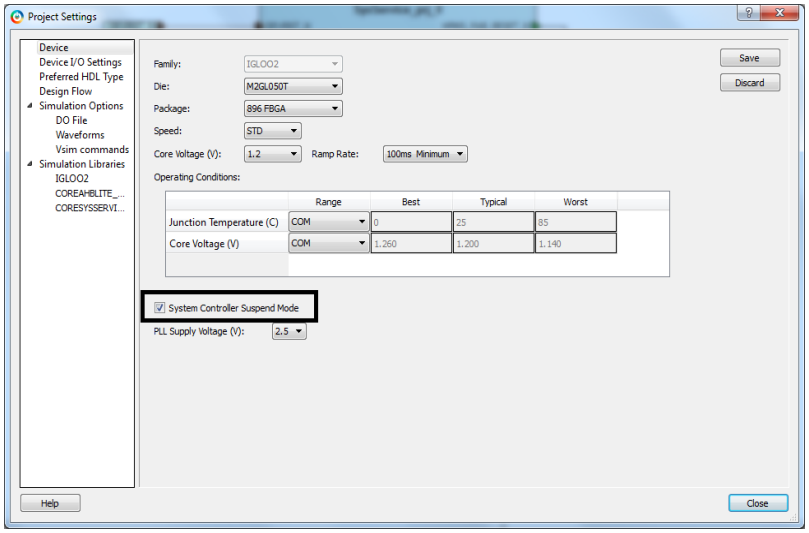
- To enable the System Controller Suspended mode in the **New Project** window, select the device family as SmartFusion 2 or IGLOO 2 and then select the **System Controller Suspended Mode** check box.

Figure 1-9. Enabling System Controller Suspended Mode in New Project Window



- To enable the System Services Suspend mode in **Project Settings** window, select the **System Controller Suspended Mode** check box.

Figure 1-10. 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 turned on. This effectively suspends all the system services from being performed. For a list of system services, see SmartFusion 2 and IGLOO 2 System Services table.

2. System Services (Ask a Question)

This chapter describes the various system services implemented by the SmartFusion® 2 and IGLOO® 2 system controller.

2.1 Overview (Ask a Question)

This chapter describes the various system services implemented by the SmartFusion 2 and IGLOO 2 system controller. The SmartFusion 2 and IGLOO 2 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). For more information on COMM_BLK, see the "Communication Block" chapter in the "IGLOO 2 High Performance Memory Subsystem User Guide" and "SmartFusion 2 MSS User Guide".

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 through 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 an 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 must be some form of arbitration or co-operation between the masters to ensure that the services are used successfully. All pointers and multibyte 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
- Asynchronous Messages
- NVM Data Integrity Check Service

The following table lists all the SmartFusion 2 and IGLOO 2 system services with their command values. Microchip 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 2-1. SmartFusion 2 and IGLOO 2 System Services[1]

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 254: Service disabled by factory security 255: Service disabled by user security
	128-bit AES Cryptographic Service	0x60x3	
	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 254: Service disabled by factory security 255: Service disabled by user security
	Challenge-Response Cryptographic Service	0xE	
Elliptic Curve Cryptography Services	ECC Point Multiplication Service	0x10	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
	ECC Point Addition Service	0x11	
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

.....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
Non-Deterministic Random Bit Generator (NRBG) Services	Self Test Service	0x28	0: Successful 1: Fatal error 2: Maximum instantiations exceeded 3: Invalid handle 4: Generate request too big 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
	Instantiate Service	0x29	
	Generate Service	0x2A	
	Reseed Service	0x2B	
	Uninstantiate Service	0x2C	
	Reset Service	0x2D	

.....continued			
Category	**System Service Name**	**Command Value**	**Response Status**
Zeroization Service	Zeroization Service	0xF0	No response status is sent.
Programming Service	IAP Service	0x14	See Factory Security Keys and Configuration Settings section.
—	ISP Service	0x15	See Table 2-122 section.
—	The ISP Service is applicable only for the SmartFusion 2 devices.		
NVM Data Integrity Check Service	NVM Data Integrity Check Service	0x17	Digest error byte Bits [7:3] – Reserved Bits [2] – eNVM1 Error 0: ENVM1 digest check passed 1: ENVM1 digest check mismatch Bits [1] – eNVM0 Error 0: ENVM0 digest check passed 1: ENVM0 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: ENVM1 digest check passed 1: ENVM1 digest check mismatch Bits [1] – eNVM0 Error 0: ENVM0 digest check passed 1: ENVM0 digest check mismatch Bits [0] – Fabric Error 0: Fabric FPGA configuration digest check passed 1: Fabric FPGA configuration digest check mismatch

Note:

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.

2.2 Device and Design Information Services (Ask a Question)

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, see the following table.

Table 2-2. Device and Design Information Services Status

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

2.2.1 Serial Number Service (Ask a Question)

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 2-3. 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 2-4. Serial Number Service Response

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

2.2.2 USERCODE Service (Ask a Question)

This service fetches the 32-bit JTAG USERCODE programmed by the user. The following tables list the service request and response for USERCODE service.

Table 2-5. 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 2-6. USERCODE Service Response

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

2.2.3 Device Certificate Service (Ask a Question)

This service fetches the device certificate from eNVM. The device certificate is a digitally-signed X-509 certificate, signed by Microchip Corp, programmed during the manufacturing process. The certificate ensures the authenticity of a device and its characteristics. The certificate is cryptographically validated before being delivered. For more information about the certificate and format, see AC436: Using Device Certificate System Service in SmartFusion2 Application Note.

Table 2-7. 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 2-8. Device Certificate Service Response

Offset	Length (bytes)	Field	Description
0	1	CMD = 0	Command
1	1	STATUS	Command status
2	4	DEVICECERTPTR	Pointer to original buffer from request

2.2.4 User Design Version Service (Ask a Question)

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

Table 2-9. Design Version Service Request

Offset	Length (bytes)	Field	Description
0	1	CMD = 5	Command

.....continued			
Offset	**Length (bytes)**	**Field**	**Description**
1	4	DESIGNVERPTR	Pointer to 2-byte buffer to receive the 16-bit design version

Table 2-10. Design Version Service Response

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

2.3 Flash*Freeze Service (Ask a Question)

This service requests the System Controller to execute the Flash*Freeze entry sequence. For more information about the Flash*Freeze entry sequence, see "Flash*Freeze" chapter in the UG0444: SmartFusion2 and IGLOO2 Low-Power Design User Guide.

Important: 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.

The following table lists the Flash*Freeze request entry sequence.

Table 2-11. Flash*Freeze Request

Offset	Length (bytes)	Field	Description
0	1	CMD = 2	Command
1	1	FFOPTIONS	Flash*Freeze options. See Table 2-12.

Table 2-12. FFOPTIONS

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

Important: 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 2-12:

- If ENVM0PD is '1' then eNVM module 0 is placed in its deep-power-down state. See 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). See the following note.
- If MPLLPD is '1' then the MPLL is powered down for the duration of the Flash*Freeze period.

Important: 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 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 2-13. Flash*Freeze Service Response

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

Table 2-14. Flash*Freeze Service Status

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

2.4 Cryptographic Services (Ask a Question)

Selected devices of the SmartFusion 2 or IGLOO 2 family can access the built-in AES and Secure Hash Standard (SHA-256) engines through cryptographic services. These cryptographic services are 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, see the “Highest Security Devices” section in the IGLOO2 FPGA Product Brief and SmartFusion2 SoC FPGA Product Brief. The possible service status values for the cryptographic services are listed in the following table.

Table 2-15. 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

2.4.1 AES Services (Ask a Question)

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 through the SII Master.

The referenced key data and plain text/cipher text is then also retrieved through the SII Master. The resultant cipher text/plain text is copied back to the MSS or HPMS memory space using the SII Master. The SmartFusion 2 and IGLOO 2 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.

2.4.1.1 128-bit AES Cryptographic Service (Ask a Question)

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

Table 2-16. 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 2-17. 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. See Table 2-18.
35	1	RESERVED	Reserved
36	4	DSTADDRPTR	Pointer to return data buffer
40	4	SRCADDRPTR	Pointer to data to encrypt/decrypt

Table 2-18. MODE Parameter

7	6	5	4	3	2	1	0
DECRYPT	Reserved	—	—	—	—	OPMODE	—

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 2-19. 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 2-20. 128-bit AES Service Response

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

.....continued			
Offset	**Length (bytes)**	**Field**	**Description**
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. The description of the MODE parameter is the same as for the 128-bit AES service.

Table 2-21. 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 65535)
50	1	MODE	Cipher Operating mode. See Table 2-18.
51	1	RESERVED	Reserved
52	4	DSTADDRPTR	Pointer to return data buffer
56	4	SRCADDRPTR	Pointer to data to encrypt/decrypt

2.4.1.2 256-bit AES Cryptographic Service (Ask a Question)

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

Table 2-22. 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

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 2-23. AES256DATA Descriptor

Offset	Length (bytes)	Field	Description
0	32	KEY	Encryption key is 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. See Table 2-23
51	1	RESERVED	Reserved
52	4	DSTADDRPTR	Pointer to return data buffer
56	4	SRCADDRPTR	Pointer to data to encrypt/decrypt

Table 2-24. 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

2.4.2 SHA-256 Services (Ask a Question)

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.

2.4.2.1 SHA-256 Cryptographic Service (Ask a Question)

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 2-25. 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 2-26. 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 2-27. 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

2.4.2.2 HMAC Cryptographic Service (Ask a Question)

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 must 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 2-28. HMAC Service Request

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

Table 2-29. 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
40	4	RESULTPTR	Pointer to 32-byte buffer to receive 256-bit HMAC result.

Table 2-30. 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

2.4.3 SHA-256 Cryptographic Service (Ask a Question)

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 2-31. 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 2-32. 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 2-33. 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

2.4.4 HMAC Cryptographic Service (Ask a Question)

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 must 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 2-34. HMAC Service Request

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

Table 2-35. 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
40	4	RESULTPTR	Pointer to 32-byte buffer to receive 256-bit HMAC result.

Table 2-36. 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

2.5 DPA-Resistant Key-Tree Services (Ask a Question)

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, see 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 SmartFusion 2 and IGLOO 2 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 2-37. 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

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 separates up to 128 possible uses of the key-tree
- A 128-bit path input which mixes 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 2-38. 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 2-39. 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 2-40. 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

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 SmartFusion 2 or IGLOO 2 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 SmartFusion 2 or IGLOO 2 larger devices (M2S060S or M2GL060S M2S090/M2GL090, and M2S150/M2GL150), the challenge-response service is based on a unique SRAM-PUF ECC private key. Microchip adds a completely random unique-per-device 384-bit ECC private key during device manufacturing. This key is generated in the Microchip FIP140-2 level 3 HSM during the key provisioning steps of the device manufacturing process, and is not recorded or re-constructible by Microchip. 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 2-41. 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 2-42. 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 2-43. 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

2.5.1 Key-Tree Cryptographic Service (Ask a Question)

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 separates up to 128 possible uses of the key-tree
- A 128-bit path input which mixes 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 2-44. 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 2-45. 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 2-46. 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

2.5.2 Challenge-Response Cryptographic Service (Ask a Question)

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 SmartFusion 2 or IGLOO 2 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 IGLOO 2 larger devices (M2S060S or M2GL060S M2S090/M2GL090, and M2S150/M2GL150), the challenge-response service is based on a unique SRAM-PUF ECC private key. Microchip adds a completely random unique-per-device 384-bit ECC private key during device manufacturing. This key is generated in the Microchip FIP140-2 level 3 HSM during the key provisioning steps of the device manufacturing process, and is not recorded or re-constructible by Microchip. 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 2-47. 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 2-48. 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 2-49. Challenge-Response Service Response

Offset	Length (bytes)	Field	Description
0	1	CMD = 14	Command
1	1	STATUS	Command status

.....continued			
Offset	Length (bytes)	Field	Description
2	4	CHRESPPTR	Pointer to CHRESP structure

2.6 Elliptic Curve Cryptography Services (Ask a Question)

The premium S grade SmartFusion 2 or IGLOO 2 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 2-50. 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

2.6.1 ECC Point Multiplication Service (Ask a Question)

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:

$Q = d \times P$

The ECC point multiplication computes 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 discussed, 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 2-51. Elliptic Curve Point Multiplication Service Request

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

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

Table 2-52. 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:

$Q = d \times G$

Where G is the generator point or base point for the NIST P-384 curve. This form of the service creates 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 2-53. ECCPMULT Structure

Offset	Length (bytes)	Field	Description
0	1	CMD = 16	Command
1	1	STATUS	Command status, see Table 2-50
2	4	ECCPMULTPTR	Pointer to ECCPMULT structure

2.6.2 ECC Point Addition Service (Ask a Question)

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 2-54. 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 2-55. 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 2-56. Elliptic Curve Point Addition Service Response

Offset	Length (bytes)	Field	Description
0	1	CMD = 17	Command
1	1	STATUS	Command status, see Table 2-50
2	4	ECCPADDPTR	Pointer to ECCPADD structure

2.7 SRAM-PUF Services (Ask a Question)

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 SmartFusion 2 or IGLOO 2 devices (M2S060 or M2GL060, M2S090 or M2GL090, and M2S150 or M2GL150) for design security applications. On premium S grade larger SmartFusion 2 or IGLOO 2 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
- 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 2-57. 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 2-58. 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 2-59. Elliptic Curve Point Addition Service Response

Offset	Length (Bytes)	Field	Description
0	1	CMD = 25	Command
1	1	STATUS	Command status, (see Table 2-60)

Table 2-60. PUFUSERAC Status(1)

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
255	Service disabled by user security

Note:

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

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 SmartFusion 2 and IGLOO 2 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 plain text 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.

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 2-61. 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 2-62. 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

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.

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.

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 2-63. 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.

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 2-63.

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 2-64. 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

.....continued			
Offset	**Length (bytes)**	**Field**	**Description**
...	...	...	...
220	4	Key#57 address	MSS/HPMS address where user Key #57 resides

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 2-65. PUFUSERKC Service Response

Offset	Length (bytes)	Field	Description
0	1	CMD = 26	Command
0	1	STATUS	Command status, (see Table 2-66)
1	4	PUFUSERKCPTR	Pointer to original buffer from request

Table 2-66. 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
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

Notes:

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.

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 EXPPORT operation is performed or the key is deleted. In both cases, a status code 3 is returned.

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 2-67. 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 filed defines the key to be reconstructed.

Table 2-68. 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 2-69. PUFUSERKC Service Response

Offset	Length (Bytes)	Field	Description
0	1	CMD = 27	Command
1	1	STATUS	Command status, (see Table 2-71)
2	4	PUFUSERKEYPTR	Pointer to original buffer from request

Table 2-70. 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

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.

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 2-71. 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 2-72. 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 2-73. PUFPUBLICKEY Structure

Offset	Length (Bytes)	Field	Description
0	1	CMD = 28	Command
1	1	STATUS	Command status (see Table 2-74)
2	4	PUFPUBLICKEYPTR	Pointer to original buffer from request

Table 2-74. 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

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.

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 2-75. 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 2-76. PUFPUBLICKEY Structure

Offset	Length (Bytes)	Field	Description
0	1	CMD = 28	Command
1	1	STATUS	Command status (see Table 2-74)
2	4	PUFPUBLICKEYPTR	Pointer to original buffer from request

Table 2-77. 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 2-78. 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

2.7.1 Create User Activation Code (Ask a Question)

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 eliminates 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 SmartFusion 2 or IGLOO 2 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 SmartFusion 2 or IGLOO 2 device is decommissioned or re-purposed.

2.7.2 Create or Delete User Activation Code Service (Ask a Question)

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 2-79. 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 2-80. 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 2-81. Elliptic Curve Point Addition Service Response

Offset	Length (bytes)	Field	Description
0	1	CMD = 25	Command
1	1	STATUS	Command status (see Table 2-82)

The following table lists the PUFUSERAC Status. For more information, see Note

Table 2-82. PUFUSERAC Status

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
255	Service disabled by user security

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

2.7.3 Key Generation and Enrollment Services (Ask a Question)

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 SmartFusion 2 and IGLOO 2 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.

2.7.4 Create or Delete User Key Code and Export or Import All Service (Ask a Question)

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

2.7.5 Get Number of Enrolled Keys (Ask a Question)

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.

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.

2.7.6 User Key (Extrinsic or Intrinsic) Enrollment (Ask a Question)

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.

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.

2.7.7 Exporting User Activation Code and All Key Codes (Ask a Question)

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 2-83. 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.

2.7.8 Importing User Activation Code and All Key Codes (Ask a Question)

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 2-83.

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 2-84. 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

.....continued			
Offset	**Length (bytes)**	**Field**	**Description**
...	...	...	...
220	4	Key#57 address	MSS/HPMS address where user Key #57 resides

2.7.9 Delete a User Key from the Enrolled Keys (Ask a Question)

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 2-85. PUFUSERKC Service Response

Offset	Length (bytes)	Field	Description
0	1	CMD = 26	Command
0	1	STATUS	Command status
1	4	PUFUSERKCPTR	Pointer to original buffer from request

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

Table 2-86. PUFUSERKC Service Response

Offset	Length (bytes)	Field	Description
0	1	CMD = 26	Command
0	1	STATUS	Command status
1	4	PUFUSERKCPTR	Pointer to original buffer from request

Table 2-87. 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
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

Important:

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

2.7.10 Key Reconstruction (Ask a Question)

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 EXPPORT operation is performed or the key is deleted. In both cases, a status code 3 is returned.

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 2-88. PUFUSERKEY Service Request

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

The following table shows how to layout the data descriptor (PUFUSERKEY) for the service request. The KEYNUM field in the data descriptor defines which key to reconstruct.

Table 2-89. 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 fetching 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 2-90. PUFUSERKC Service Response

Offset	Length (bytes)	Field	Description
0	1	CMD = 27	Command
1	1	STATUS	Command status, see Table 2-92
2	4	PUFUSERKEYPTR	Pointer to original buffer from request

Table 2-91. USERPUFKEY Status

STATUS	Description
0	Success completion
2	PUF error, when creating
3	Invalid keynum or argument or exported or invalid key

.....continued	
STATUS	Description
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

2.7.11 Fetch a User PUF Key Service (Ask a Question)

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 2-92. 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) is passed in the service request. In the data descriptor, the KEYNUM filed defines the key to be reconstructed.

Table 2-93. 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 2-94. PUFUSERKC Service Response

Offset	Length (bytes)	Field	Description
0	1	CMD = 27	Command
1	1	STATUS	Command status, see Table 2-92
2	4	PUFUSERKEYPTR	Pointer to original buffer from request

Table 2-95. 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

2.7.12 User-Enrolled SRAM-PUF Keys for Design Security (Ask a Question)

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.

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 2-96. PUFPUBLICKEY Service Request

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

The following table shows how the layout of the data descriptor (PUFPUBLICKEY) is 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 2-97. 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 2-98. PUFPUBLICKEY Structure

Offset	Length (bytes)	Field	Description
0	1	CMD = 28	Command
1	1	STATUS	Command status (see Table 2-99)
2	4	PUFPUBLICKEYPTR	Pointer to original buffer from request

Table 2-99. 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

2.7.13 SRAM-PUF based True Random Number Seed Generation (Ask a Question)

From one turn-on to the next, the SRAM generates a large number of repeatable bits-enough 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.

2.7.14 Get a PUF Seed Service (Ask a Question)

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 2-100. 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 2-101. PUFPUBLICKEY Structure

Offset	Length (bytes)	Field	Description
0	1	CMD = 28	Command
1	1	STATUS	Command status, see Table 2-95
2	4	PUFPUBLICKEYPTR	Pointer to original buffer from request

Table 2-102. 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 2-103. 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

2.8 Non-Deterministic Random Bit Generator (NRBG) Services (Ask a Question)

The Non-Deterministic Random Bit Generator (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 SmartFusion 2 and IGLOO 2 true random entropy source and other possible user and device supplied seed inputs. In the SmartFusion 2 and IGLOO 2 devices where the SRAM-PUF is available, the SRAM-PUF hardware generates 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 strengthens the NRBG by providing it as additional entropy when seeding the DRBG. In the SmartFusion 2/IGLOO 2 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 1024 random bits can be returned per call to an instantiation. The SmartFusion 2 and IGLOO 2 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, see 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 following table lists the common set of service response status codes that are shared by the NRBG services.

Table 2-104. 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

2.8.1 Self Test Service (Ask a Question)

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.

The following tables lists the layout of the DRBG Self Test request and response.

Table 2-105. DRBG Self Test Request

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

Table 2-106. DRBG Self Test Response

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

2.8.2 Instantiate Service (Ask a Question)

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. Microchip 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 must 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 must 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.

The following table lists the layout of the DRBG Instantiate request.

Table 2-107. DRBG Instantiate Request

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

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

Table 2-108. DRBGINSTANTIATE 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

The following table lists the layout of the DRBG Instantiate service response.

Table 2-109. DRBG Instantiate Service Response

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

2.8.3 Generate Service (Ask a Question)

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 2-110. 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 2-111. 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.
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.

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

Table 2-112. 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

2.8.3.1 Reseed Service (Ask a Question)

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 reseeds the random bit generator identified by the DRBG handle that is provided in the service request. The reseed service forces a reseed operation.

The following tables lists the layout of the DRBG Reseed request.

Table 2-113. DRBG Reseed Request

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

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

Table 2-114. 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 2-115. 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

2.8.4 Uninstantiate Service (Ask a Question)

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.

The following tables lists the layout of the DRBG Uninstantiate request and response.

Table 2-116. DRBG Uninstantiate Request

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

Table 2-117. DRBG Uninstantiate Response

Offset	Length (bytes)	Field	Description
0	1	CMD = 44	Command
1	1	STATUS	Command status

2.8.5 Reseed Service (Ask a Question)

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.

The following tables lists the layout of the DRBG Reset request and response.

Table 2-118. DRBG Reset Request

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

Table 2-119. DRBG Reset Response

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

2.9 Zeroization Service (Ask a Question)

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, see UG0443: SmartFusion2 and IGLOO2 FPGA Security Best Practices User Guide.

Table 2-120. Zeroization Request

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

The state of SmartFusion 2 and IGLOO 2 internal memories affected by zeroization services are described in this section. 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.

2.9.1 FPGA Fabric Configuration NVM (Ask a Question)

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

2.9.2 User Security Keys and Settings (Ask a Question)

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

2.9.3 Factory Security Keys and Configuration Settings (Ask a Question)

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

Table 2-121. 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
Unrecoverable	Nothing stays	Device is permanently disabled (“bricked”) and cannot be recovered.

2.9.4 System Controller Memory (Ask a Question)

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

2.9.5 Digital Data Path (Ask a Question)

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

2.9.6 Fabric Registers (Ask a Question)

Fabric Register content is destroyed.

2.9.7 Fabric SRAM (Ask a Question)

Fabric SRMAs are cleared.

2.9.8 HPMS SRAM (Ask a Question)

eSRAM_0 and eSRAM_1 are cleared to zero.

2.9.9 eNVM Memory Array and eNVM Registers (Ask a Question)

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

2.10 Programming Service (Ask a Question)

Programming services include In-Application Programming (IAP) and in-system programming services. For more information about IAP and ISP, see SmartFusion2 and IGLOO2 Programming User Guide. The following table lists the programming service status codes.

Table 2-122. Programming Service Status Codes

Status							Description
7	6	5	4	3	2	1	
0							Success
0				AUTHERRCODE			Authentication error, see Table 2-123

.....continued			
Status			**Description**
1	0	ERRORCODE	Programming error, see Table 2-124
255			Service disabled

The following tables list the Autherrcode and Errorcode with their description.

Table 2-123. Autherrcode

Autherrcode	Description
0	Noerror
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 2-124. 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

2.10.1 IAP Service (Ask a Question)

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 through the SII Master and passed along to the programming engine.

In SmartFusion 2, the Cortex-M3 processor remains active and continues to operate during IAP service execution, but it requires to execute the code from eSRAM if the eNVMs are being reprogrammed.

Table 2-125. IAP Programming Service Request

Offset	Length (bytes)	Field	Description
0	1	CMD = 20	Command
1	1	OPTIONS	—
2	4	SPIADDR	Base address of bitstream in MSS SPI0 or HPMS SPI (MS byte ignored)

Table 2-126. OPTIONS

OPTIONS							
7	6	5	4	3	2	1	0
Reserved	—	—	—	—	—	MODE, see Table 2-126	

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

Table 2-127. 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 2-128. IAP Response

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

2.10.2 ISP Service (Ask a Question)

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.

The following table lists the offsets of ISP programming service request.

Table 2-129. ISP Programming Service Request

Offset	Length (bytes)	Field	Description
0	1	CMD = 21	Command
1	1	OPTIONS	See Table 2-126
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.

The following table lists the offsets of ISP responses.

Table 2-130. ISP Response

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

2.11 NVM Data Integrity Check Service (Ask a Question)

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 2-131. NVM Data Integrity Check Service Request

Offset	Length (bytes)	Field	Description
0	1	CMD = 23	Command
1	1	OPTIONS	Service options, seeTable 2-132

Table 2-132. OPTIONS

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

Important: 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 occurs 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 2-133. NVM Data Integrity Check Response

Offset	Length (bytes)	Field	Description
0	1	CMD = 23	Command
1	1	DIGESTERR	Pass/fail flags, see Table 2-134

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

Table 2-134. 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

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, see Libero SoC Design Flow User Guide.

2.12 Unrecognized Command Response (Ask a Question)

If an unrecognized command is received, which is not listed in Table 2-1, 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 2-135. Unrecognized Command Message

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

2.13 Asynchronous Messages (Ask a Question)

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.

Table 2-136. POR Digest Error Message

Offset	Length (bytes)	Field	Description
0	1	CMD = 241	Command
1	1	DIGESTERR	See Table 2-134

2.13.1 Power-on-Reset (POR) Digest Error (Ask a Question)

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. See the following table.

Offset	Length (Bytes)	Field	Description
0	1	CMD = 241	Command
1	1	DIGESTERR	See Table 2-134

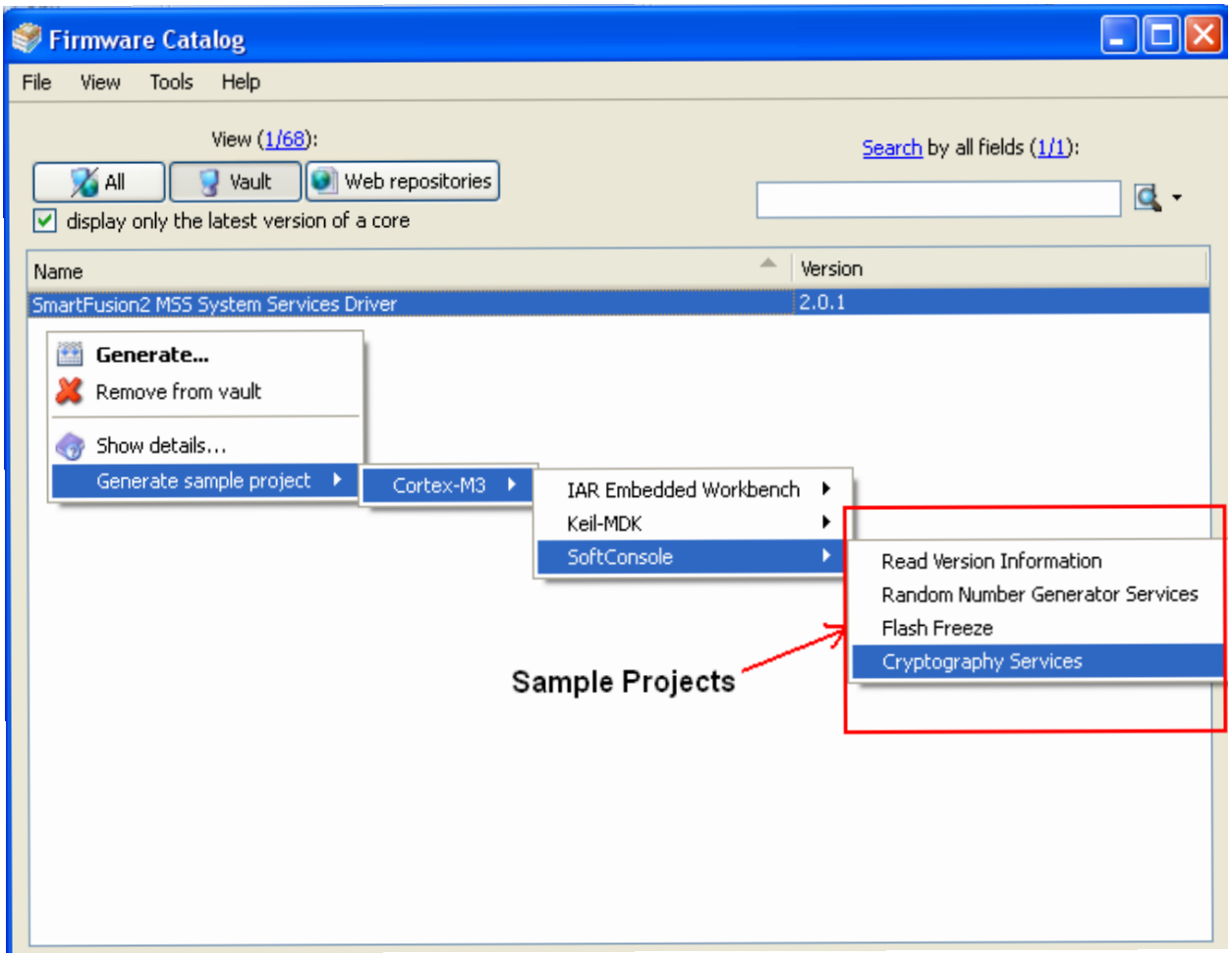
If an error is detected, the system is still allowed to boot as normal, but messages are sent to the fabric indicating the failure.

2.14 How to Use System Services in SmartFusion 2 (Ask a Question)

Microchip 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 registers 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, see 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 is generated by right clicking on the system services driver and selecting the Generate sample project, as shown in the following figure.

Figure 2-1. System Services Sample Projects



2.14.1 Use Model 1: Fetching Device and Design Information (Ask a Question)

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 2-137. 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. See 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);
```

```
}
}
```

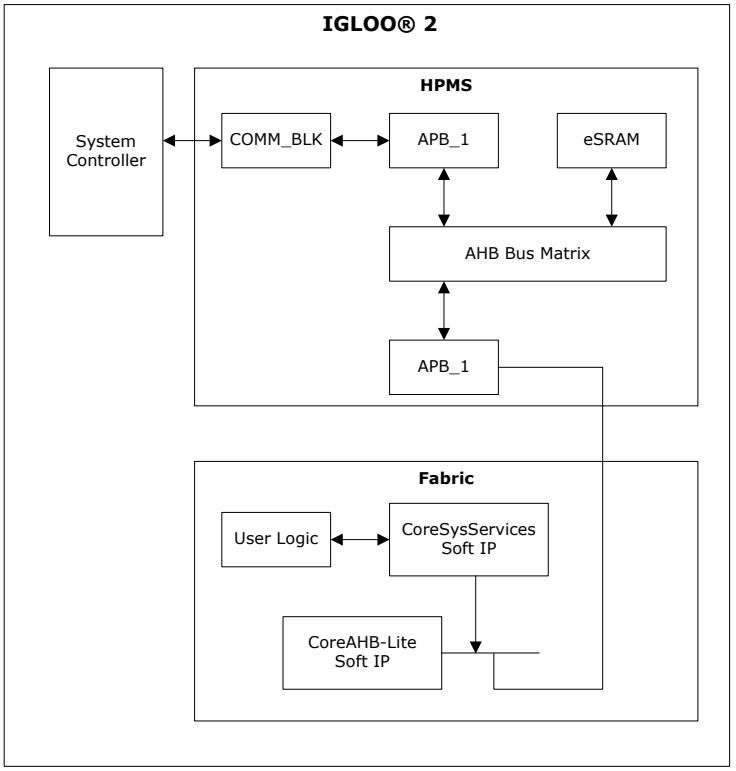
For more information, see How to Use System Services in IGLOO 2.

2.15 How to Use System Services in IGLOO 2 (Ask a Question)

This section describes how to use system services. Microchip 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 AHB-Lite master interface on the Fabric Interface Controller (FIC) side. The core communicates with the COMM_BLK through the FIC_0 interface.

Core Sys Services 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, see CoreSysServices Handbook. The following figure shows the functional block diagram for accessing system services.

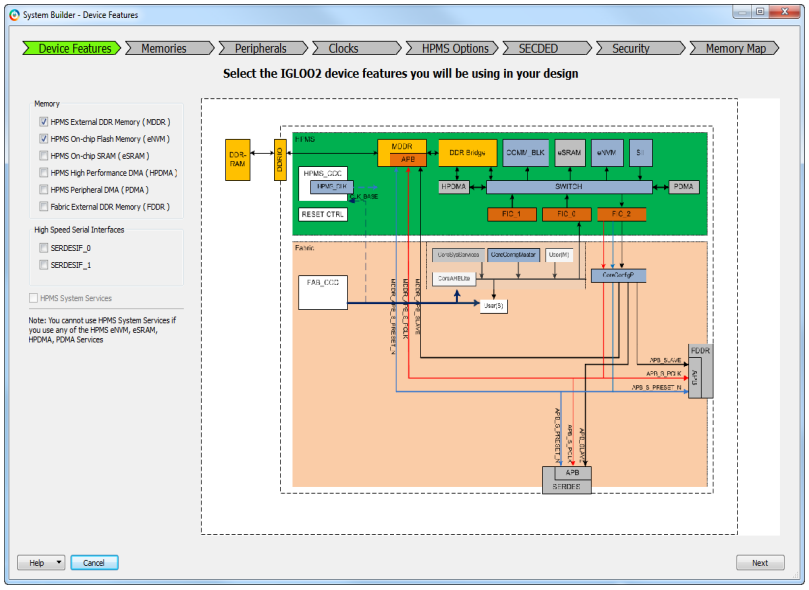
Figure 2-2. Functional Block Diagram for Accessing System



To configure the IGLOO 2 device features and to build a complete IGLOO 2 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, see IGLOO 2 System Builder User's Guide.

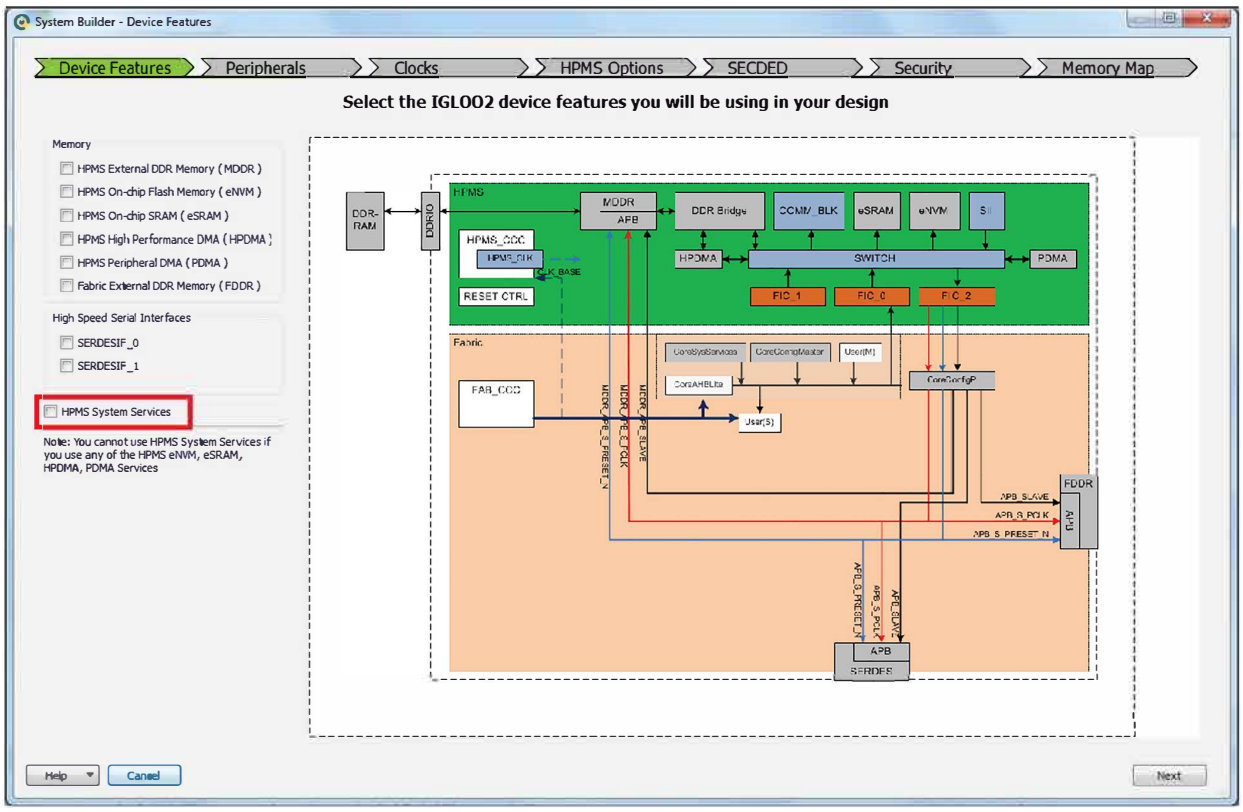
Figure 2-3. System Builder Window



2.15.1 Configuring System Services (Ask a Question)

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 2-4. System Builder—Device Features Tab

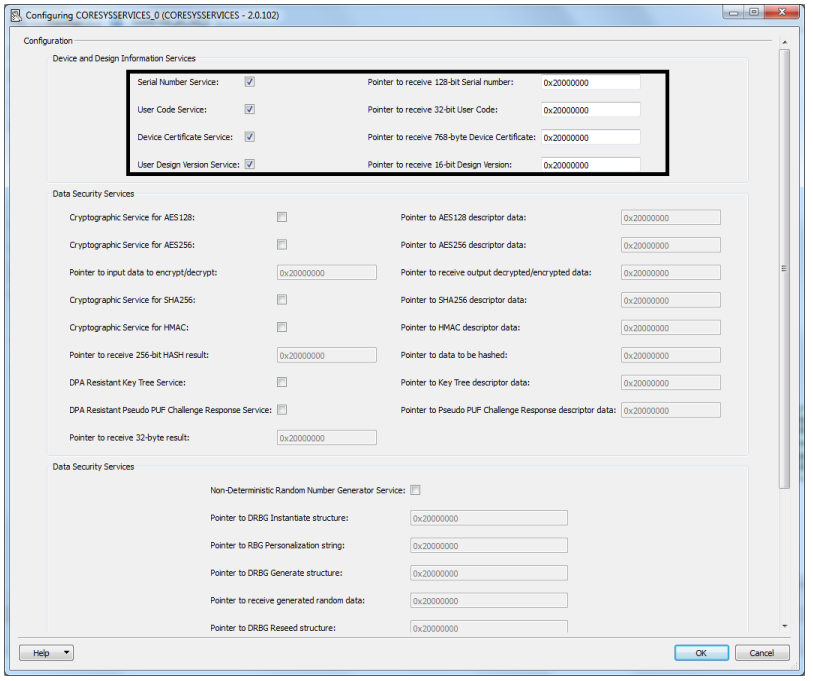


2.15.2 Fetching Device and Design Information (Ask a Question)

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

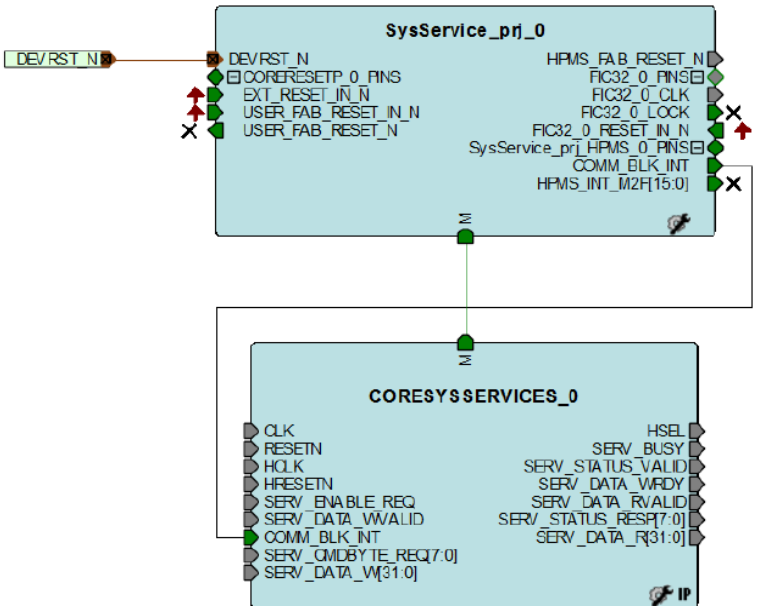
1. To open the **Configuring CORESYSSERVICES_0** dialogue box, right-click on the **CORESYSSERVICES_0** instance.
2. To configure the CoreSysServices soft IP for **Device and Design Information Services**, under **Device and Design Information Services**, select all the check boxes.

Figure 2-5. CORESYSSERVICES_0 Configuration



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

Figure 2-6. SmartDesign 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 2-138. User Interface Signals

Port	Type	Description
Handshaking Signals		
SERV_BUSY	Output	Service busy When deasserted, indicates that no service has been requested. When asserted, indicates that the current service is in progress.
SERV_DATA_RVALID	Output	Data Valid for User Read Data
SERV_DATA_R[31:0]	Output	User Read Data
User Interface Signals: Request Signals		
SERV_ENABLE _REQ	Input	Active-High request to start the service
SERV_CMDBYTE_REQ[7:0]	Input	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]	Output	Response status of the requested service
SERV_STATUS_VALID	Output	Response Status valid

   For more information about port description and timing diagrams, see CoreSysServices Handbook.

   The following table lists the command codes for device and design information services.

Table 2-139. 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, see Table 2-1.

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.

2.15.3 Related Applications (Ask a Question)

For more information, see 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
- AC410: Using AES System Services in SmartFusion2 and IGLOO2 Devices

- AC435: Using ECC System Service in SmartFusion2
- AC407: Using NRBG Services in SmartFusion2 SoC and IGLOO2 FPGA Devices

3. Revision History (Ask a Question)

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

Table 3-1. Revision History

Revision	Date	Description
A	12/2024	The following is a summary of the changes made in this revision: • The document was migrated to Microchip template. • The document number was updated to “DS50003787” from “UG0450”. • Removed the “Recoverable” Zeriozation option related information from Table 2-121.
6.0	—	The following is a summary of the changes in this revision. • Updated System Controller Suspend Mode. • Updated Figure 1-5.
5.0	—	Updated Dedicated Programming SPI Peripheral and System Controller Suspend Mode.
4.0	—	The following is a summary of the changes in this revision. • Added the Related Applications. • Updated the Key-Tree Cryptographic Service. • Updated the Challenge-Response Cryptographic Service. • Added the Elliptic Curve Cryptography Services. • Added the SRAM-PUF Services. • Updated the Non-Deterministic Random Bit Generator (NRBG) Services. • Added a note in Flash*Freeze Service about deep-power-down action. • Updated Table 2-135 for command change. • Updated the references as per the standard.
3.0	—	The following is a summary of the changes in this revision. • Updated value for OPMODE 3 in Table 2-19. • Updated Reseed Service. • Updated System Controller Suspend Mode.
2.0	—	The following is a summary of the changes in this revision. • Updated Table 2-1. • Updated DPA-Resistant Key-Tree Services. • Updated Non-Deterministic Random Bit Generator (NRBG) Services. • Updated the Instantiate Service. • Updated the Generate Service. • Updated the Reseed Service. • Added Table 2-123 and Table 2-124. • Updated Figure 1-2 and Figure 1-4. • Updated SYSRESET. • Updated System Controller Suspend Mode. • Added Clock Requirements.

.....continued

Revision	Date	Description
1.0	—	The following is a summary of the changes in this revision. • Restructured the user guide. • Updated Table 2-19, Table 2-67, Table 2-121. • Added How to Use System Services in IGLOO 2, page 54 section. • Updated Figure 1-2. • Added System Controller Suspend Mode.

Microchip FPGA Support

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

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

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

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

Microchip Information

Trademarks

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

ISBN: 979-8-3371-0014-2

Legal Notice

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

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

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

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

Microchip Devices Code Protection Feature

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

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