PolarFire Family FPGA Security User Guide

Introduction (Ask a Question)

This user guide describes the security architecture and its components available in the PolarFire® family. The FPGA fabric is common to the PolarFire family, which consists of the following FPGA devices.

PolarFire FPGAs	Microchip's PolarFire® FPGAs are the fifth-generation family of non-volatile FPGA devices, built on state-of-the-art 28 nm non-volatile process technology. PolarFire FPGAs deliver the lowest power at mid-range densities. PolarFire FPGAs lower the cost of mid-range FPGAs by integrating the industry's lowest power FPGA fabric, lowest power 12.7 Gbps transceiver lane, built-in low power dual PCI Express® Gen2 (EP/RP), and, on select data security (S) devices, an integrated low-power crypto co-processor.
PolarFire SoC FPGAs	Microchip's PolarFire SoC FPGAs are the fifth-generation family of non-volatile SoC FPGA devices, built on state-of-the-art 28 nm non-volatile process technology. The PolarFire SoC family offers industry's first RISC-V based SoC FPGAs capable of running Linux®. It combines a powerful 64-bit 5x core RISC-V Microprocessor Subsystem (MSS), based on SiFive's U54-MC family, with the PolarFire FPGA fabric in a single device.
RT PolarFire FPGAs	Microchip's RT PolarFire FPGAs combine our 60 years of space flight heritage with the industry's lowest-power PolarFire FPGA family to enable new capabilities for space and mission-critical applications. RT PolarFire FPGA family includes RTPF500T, RTPF500TL, RTPF500TS, RTPF500TLS, RTPF500ZT, RTPF500ZTL, RTPF500ZTS, and RTPF500ZTLS devices.

Today's applications are expected to meet demanding functional requirements, and must do so securely by protecting both application design and information. Protecting design and information calls for secure hardware, and design and data security. Microchip's PolarFire FPGAs and PolarFire SoC FPGAs provide a solid foundation for all application security needs.

Design security protects the design intellectual property (IP) and other sensitive information such as cryptographic keys that are used for the FPGA configuration. Design IP includes designer's logic design, firmware code, and security settings loaded in the device. Design security assures that the user design programmed onto a device is secure, and operates as intended, for the life of the product. Data security protects application data—stored, communicated, or computed at run-time—from being copied, altered, or corrupted. PolarFire FPGA and PolarFire SoC FPGA devices have a dedicated crypto processor, referred as User Cryptoprocessor, for data security applications.

The following table summarizes the important attributes for a strong security.

Table 1. Attributes of Security

Component	PolarFire® FPGA (MPF)	PolarFire SoC FPGA (MPFS)	RT PolarFire FPGA
Design Security Keys and Key Management	✓	✓	✓
Bitstream Security	✓	✓	✓
Hardware Access Control	✓	✓	✓
Device-level Anti-tamper Features	✓	✓	✓
Supply Chain Assurance	✓	✓	✓
Secure Boot	—	✓	—
Physical Memory Protection (PMP)	—	✓	—

Table 1. Attributes of Security (continued)			
Component	PolarFire® FPGA (MPF)	PolarFire SoC FPGA (MPFS)	RT PolarFire FPGA
Memory Protection Unit (MPU)	—	✓	—
Data Security	✓	✓	✓

The rest of the chapters describe how these security attributes are implemented in the device families. These devices include features that provide enhanced security during all stages of the device life cycle from silicon manufacturing, user key injection and bitstream programming, to field updates, and finally to device decommissioning.

References

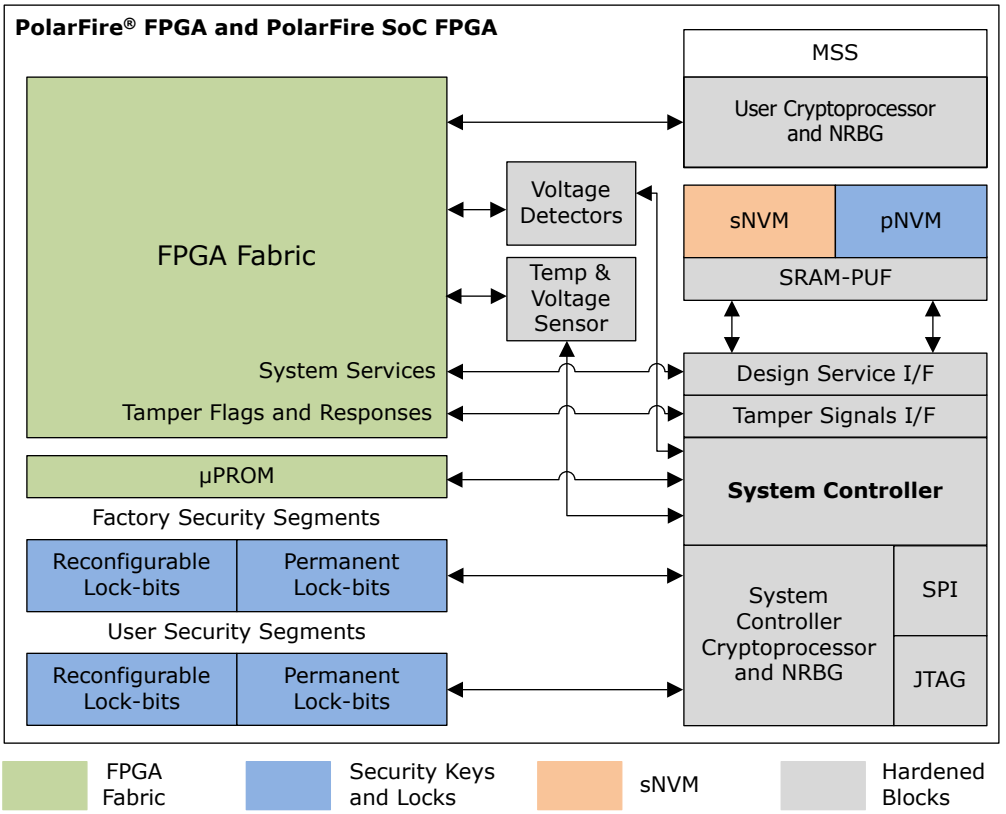
- For information about system services, see PolarFire FPGA and PolarFire SoC FPGA System Services User Guide.
- For information about secure production programming solution, see Secure Production Programming Solution (SPPS) User Guide.
- For information about MSS configurator, see PolarFire SoC Standalone MSS Configurator User Guide.
- For information about MSS, see PolarFire SoC FPGA MSS Technical Reference Manual.

Table of Contents

1. Security Architecture (Ask a Question)

The following figure illustrates the device architecture from a security perspective. The rest of the chapter describes the components of the security architecture.

Figure 1-1. Simplified Security Model



Note: User Cryptoprocessor is part of MSS in PolarFire SoC FPGAs and there is no MSS in PolarFire FPGAs and RT PolarFire FPGAs. User Cryptoprocessor is a standalone block in both PolarFire FPGAs and RT PolarFire FPGAs.

1.1 System Controller (Ask a Question)

The System Controller manages device programming, design security, key-management, and related operations. During the programming process, the System Controller authenticates and decrypts the incoming bitstream, erases and writes the target flash memory segments, and responds to other external programming related protocols, such as key verification. The system controller has both a JTAG interface and a SPI interface.

The System Controller contains a dedicated cryptoprocessor, the Athena TeraFire® F5200ASR, for accelerating device-specific cryptographic functions. All cryptographic algorithms are implemented using patented DPA-resistant techniques to minimize the probability of secret key extraction by an adversary using simple or differential power analysis (SPA or DPA), simple or differential electromagnetic analysis (SEMA or DEMA), or timing analysis (TA). This protection extends to the messages digested using any of the secure hash algorithms (SHA), even though they do not directly use a secret key, because they are used in the hash-based message authentication algorithm (HMAC) that does.

The System Controller also incorporates a non-deterministic random bit generator (NRBG), known as a true random number generator (TRNG). The integrated TRNG enables modern FPGA cryptographic protocols that provide protection against attacks such as replay attacks. It is also used for high-quality key, nonce, and initialization vector generation.

The system controller also provides system services such as reporting the device serial number, the JTAG USERCODE value, exporting the device certificate, and so on. For more information, see PolarFire FPGA and PolarFire SoC FPGA System Services User Guide.

1.2 SRAM Physically Unclonable Function (SRAM-PUF) (Ask a Question)

The devices integrate Quiddikey-Flex IP licensed from Intrinsic-ID, a secure key management solution based on SRAM Physically Unclonable Function (SRAM-PUF) technology. The SRAM-PUF can be used for secure key generation and storage as well as for a source of randomness.

Keys that are derived from the SRAM-PUF are not stored 'on the chip' but they are extracted 'from the chip', only when they are needed. In that way, they are only present in the chip during a very short time window. When the SRAM is not powered on, there is no key present on the chip making the solution very secure.

The system controller can extract two PUF master secret keys from the device using Quiddikey-Flex IP. One PUF master secret key is used for wrapping design security keys, and the second is used for the secure non-volatile memory (sNVM) encryption and authentication features. The SRAM-PUF functionality is provided to the user through the sNVM encryption/authentication mechanism. The sNVM is intended to provide a highly secure storage for the user to store application keys and other sensitive data in authenticated plaintext or authenticated ciphertext form using SRAM-PUF technology. For more information, see Secure Non-Volatile Memory (sNVM) section.

1.3 Secure Non-Volatile Memory (sNVM) (Ask a Question)

The sNVM block is a user non-volatile flash memory that can be programmed independently. Each device has 56 Kbytes of sNVM. The sNVM is organized into 224 pages, each page is of 256 bytes in size. Three pages are reserved for administrative purposes, leaving 221 pages available for user data. Individual pages in the sNVM can be designated as write-protected (ROM) when its programming bitstream is generated, to make it easy to control sensitive data and prevent overwriting of those pages at run-time. sNVM pages marked as ROM can only be modified by device reprogramming. The sNVM content is accessible to the user logic through the system service calls. For more information, see PolarFire FPGA and PolarFire SoC FPGA System Services User Guide.

The sNVM can be written with data along with device programming or using system service at run time. The data written to the sNVM can be protected by a device unique intrinsic PUF secret key (SMK) using AES-256 in the synthetic initialization vector (SIV) mode.

The data may be stored in any of the following formats (listed in the ascending order of access time) in sNVM:

- Non-authenticated plaintext
- Authenticated plaintext (Available in all PolarFire family “S” version devices)
- Authenticated ciphertext (Available in all PolarFire family “S” version devices)

Non-authenticated plaintext provides the fastest access time and authenticated ciphertext is the slowest but provides the highest level of security. For authenticated plaintext or ciphertext, a user provided user sNVM key (USK) is used for authentication during read. When the user data is stored in non-authenticated format, 252 bytes of storage per page is available for user data. When the user data is stored in authenticated format, 236 bytes of storage per page is available for user data. If the data is programmed using authentication, the USK key used at the time of programming must be provided while retrieving the data using system service call. You must configure security policies of the Configure Security tool when authentication is used.

The data stored in plaintext format using device programming can be used to initialize LSRAM and µSRAM blocks in the FPGA fabric during device initialization.

sNVM configurator is available in **Configure Design Initialization Data and Memories** under Libero® Design Flow. Click **Add** to add data storage client in sNVM. Add USK Client when authentication is used.

Important: In Libero, the added USK client is stored in the user specified sNVM page and this USK is used for all the authenticated plaintext or authenticated ciphertext clients created in the Libero project. User application in the fabric may use a different USK and overwrite any of the sNVM data clients (not marked as ROM) using sNVM write system service during runtime. However, it causes design verification failure using bitstream, even if the data is same.

Figure 1-2. sNVM Configurator

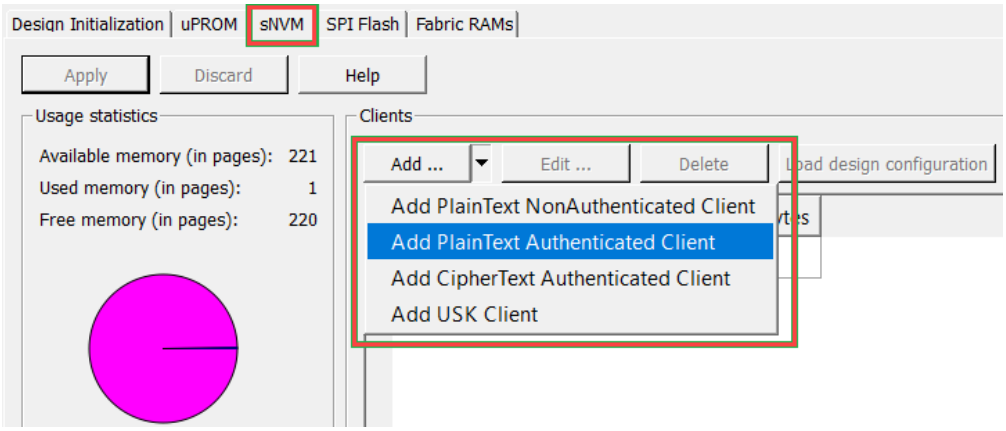


Figure 1-3. sNVM Data Client Configuration

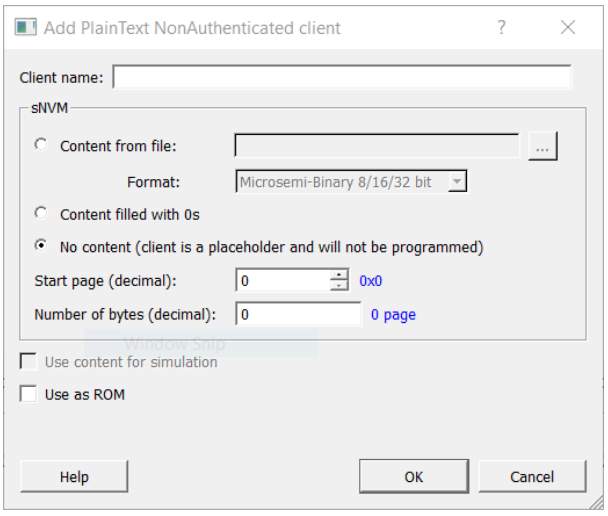
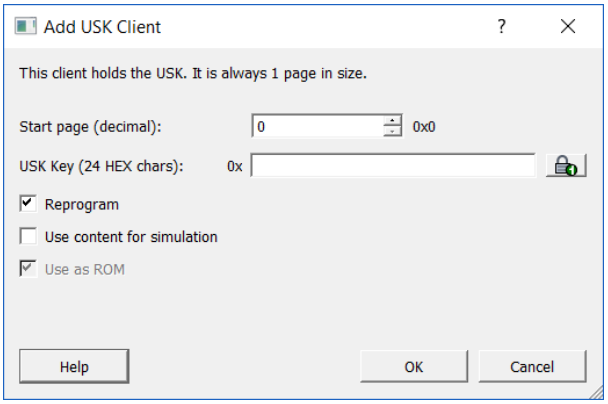


Figure 1-4. USK Client



Notes:

- For PolarFire FPGA and RT PolarFire FPGA, if the data is programmed using authentication, then the USK key used at the time of programming must be provided while retrieving the data using the system service call. You must configure security policies of the Configure Security tool when authentication is used.
- For PolarFire SoC FPGA, only one USK client in the sNVM is allowed in Libero SoC. The sNVM system services can be used to use per-page USK. Here the per-page USK is not stored on the device but must be presented to the sNVM Read system service to correctly retrieve the data for each protected page. See PolarFire FPGA and PolarFire SoC FPGA System Services User Guide for more information.
- The authenticated writes to the sNVM using system services pass only after the SMK (sNVM Master Key) is successfully generated by the device. To generate the SMK, program the device with an authenticated client in sNVM using Libero SoC. When the SMK is generated, it can be used for performing authenticated writes to the sNVM through System Services firmware.
- If VDD, VDD18, or VDD25 power is lost or experiences a brownout while an sNVM page write is in process, the page may corrupt where it is no longer capable of being written. This occurs because the page is first erased during the write process, which sets the page 'Read Only' bit. If the page is not restored during the subsequent data storage step of the write, the page becomes read-only. This can be detected by a subsequent page write after power is restored, which results in a write return status = 0x4, indicating ‘write not permitted’. Alternatively, a failed digest check of sNVM also indicates this condition. Reprogramming the device with a bitstream containing the affected sNVM page content is required to restore the page to allow writes.

1.4 Private Non-Volatile Memory (pNVM) (Ask a Question)

The pNVM provides protected non-volatile storage for both factory and user keys using a device unique intrinsic PUF secret key. The pNVM also stores the device's X.509-compliant certificate. For more information about factory and user keys, see Design Security Keys and Key Management.

1.5 Security Segments (Ask a Question)

The factory and user security segments store the factory and user security locks respectively. For more information about the security locks, see Hardware Access Controls.

1.6 Secure Boot (For PolarFire SoC FPGA Only) (Ask a Question)

PolarFire SoC comes with two secure boot options to securely boot the application processors. For the default PolarFire SoC secure boot method, the system controller copies the Microchip secure boot loader from its private, secure memory area and load it into the 8 KB DTIM of the E51 monitor core. After that, the reset is released to the application CPUs and then the secure boot code starts execution. The default secure boot loader performs a signature check on the 128 KB eNVM, then run a hash on the eNVM image. If no errors are reported, the code jumps to the user application

stored in the eNVM. If errors are reported, the system controller activates a tamper alarm that asserts a signal to the FPGA fabric. Users can then decide on a plan of action.

The second secure boot method allows users to place their own boot code in the secure non-volatile memory (sNVM) area of the chip. The sNVM is a 56 KB nonvolatile memory that can be protected by the built-in Physically Unclonable Function (PUF), that is, the unique PUF ID can serve as an initialization vector for an AES encrypt/decrypt operation performed by the side-channel resistant system controller co-processor. On power-up, the system controller copies the user code to the target address within the on-chip RAMs at the top of the UBL Image.

Important: Secure Boot is available on all PolarFire SoC devices including "S" and "non-S" versions.

1.7 Physical Memory Protection (For PolarFire SoC FPGA Only) (Ask a Question)

Each CPU in PolarFire SoC includes a physical memory protection (PMP) unit compliant with the RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10. The PMP unit can be used to set memory access privileges (read, write, execute) for specified memory regions. Each PMP supports 16 regions with a minimum region size of 4 bytes. It is permitted to have overlapping regions. PMP unit can be used to restrict access to memory and isolate processes from each other. The PMP allows for region locking whereby once a region is locked, further writes to the configuration and address registers are ignored. Locked PMP entries may only be unlocked with a system reset. For more information, see PolarFire SoC FPGA MSS Technical Reference Manual.

1.8 Memory Protection Unit (For PolarFire SoC FPGA Only) (Ask a Question)

Random access to memory regions by any non-CPU master can corrupt the memory and the overall system. To avoid random access to memory, the PolarFire SoC MSS includes a built-in Memory Protection Unit (MPU) for each non-CPU master. The GEM0, GEM1, eMMC, USB, SCB, Crypto Processor, Trace, FIC0, FIC1, and FIC2 master blocks interface with an MPU. The MPU can be used to create access regions in memories for a particular master and define privileges to those access regions. The access regions are created by setting the PMP registers inside an MPU. The privileges are also defined by setting particular bits of the PMP registers. At reset, access to the MSS is not provided until the access regions of the required MPUs are created.

MPUs monitor transactions on the AXI read and write channels and only legal accesses pass through. Illegal transactions are not allowed to pass from MPU to the AXI switch, and the MPU initiates AXI response transaction. For more information, see PolarFire SoC FPGA MSS Technical Reference Manual.

1.9 Device-Level Anti-Tamper Features (Ask a Question)

The PolarFire family include a number of built-in tamper detection and response capabilities that can be used to enhance the security of the device. These countermeasures are intended to address various types of attacks that include non-invasive, semi-invasive, and invasive attacks. The System Controller can detect a number of conditions that may indicate attempted tampering with the device.

The anti-tamper system present in device includes voltage, frequency, and temperature monitors. When a tamper condition is detected, a notification event is sent to the fabric through one of many dedicated control lines. Voltages and temperature can be digitally monitored through another fabric interface. For more information, see Temperature and Voltage Sensor.

User logic, in response to the detected tamper event, can request the system controller to disable all IO pins, lock-down, reset, or zeroize the device through the tamper response interface. Ignoring the event does not impact user design operation. The tamper responses may also be initiated on-demand by the user (for example, if the user has their own system-level tamper detection connected to FPGA IOs).

The devices also incorporate DPA countermeasures for all built-in design security protocols to protect the secret keys from discovery using side-channel analysis.

For more inforamtion about anti-tamper features, see Device-Level Anti-Tamper Features.

1.10 User Cryptoprocessor and NRBG (Ask a Question)

The "S" grade devices include a dedicated cryptoprocessor (referred to as the User Cryptoprocessor) for data security applications. In PolarFire SoC FPGA, the user cryptoprocessor is integrated within the microcontroller subsystem (MSS). The user cryptoprocessor can be accessed from MSS or Fabric. The User Cryptoprocessor is an Athena TeraFire EXP-F5200B cryptography microprocessor. It provides complete support for the Commercial National Security Algorithm (CNSA) suite and beyond, and also includes side-channel analysis (SCA) resistant cryptographic countermeasures. These countermeasures provide strong resistance against SCA attacks such as SPA and DPA.

The User Cryptoprocessor also incorporates an NRBG. The User Cryptoprocessor specifically supports an NRBG combined with an AES counter mode-based DRBG, compliant with NIST SP800-90A.

Many of the commonly used cryptographic operations available are certified by an independent third-party NIST-accredited security laboratory under the NIST cryptographic algorithm validation program (CAVP) scheme. This includes the AES, SHA, HMAC, ECDSA, RSA, DSA, and DRBG implementations, providing a high level of assurance that they are implemented correctly. The following table lists the CAVP validation numbers, see the NIST CAVP website for details on the specific algorithms and modes that are certified.

Table 1-1. NIST CAVP Validation Numbers

Algorithm	CAVP Number
AES	3950
SHA1/2	3258
HMAC	2573
DSA	1077
RSA	2018
ECDSA	867
DRBG	1153

For more information about User Cryptoprocessor, see Data Security .

Note: The User Cryptoprocessor and NRBG block is disabled using an SEU immune flash bit in the non 'S' grade devices.

1.11 Pass-through License for CRI Patented DPA Protection (Ask a Question)

Microchip has obtained a license from Cryptography Research, Inc. (CRI, now a division of Rambus) for the DPA patent portfolio, consisting of more than fifty patents. The pass-through licensing enables additional DPA-resistant high-speed cryptographic implementations in the FPGA fabric, if higher performance (or a different algorithm) is needed. Customers do not need to negotiate a separate license with Rambus if they need to incorporate DPA resistant functions in the PolarFire SoC FPGA MSS application or PolarFire FPGA/RT PolarFire FPGA fabric when purchasing “S” devices. For example, ordering code for a PolarFire MPF300T “S” device is MPF300TS-FSCG536I and for a non “S” device is MPF300T-FSCG536I. Ordering code for a PolarFire SoC FPGA MPFS250T “S” device is MPFS250TS-FCG1152 and for a non “S” device is MPFS250T-FCG1152.

2. Design Security Keys and Key Management (Ask a Question)

All devices are provisioned with a set of device unique factory keys. In addition, the end users can also enroll their own security keys, thus providing complete independence from using Microchip provided factory keys.

2.1 Key Management (Ask a Question)

Key management is often the critical link in a secure system. Key management includes securely generating, distributing, and storing keys. The devices contain factory provisioned key material and X.509-complaint certificate that can be used to authenticate a device and provide a starting point for enrolling user keys.

Factory keys and passcodes are generated by a Microchip NIST-certified HSM and injected into the virgin devices in encrypted form after proving that the silicon wafers have been fabricated using the genuine Microchip design. Only the genuine Microchip device can decrypt the bitstream used to load the Factory keys and passcodes. During the design of each device, Microchip inserts a number of highly obfuscated secrets. When the devices are fabricated according to this design, the secrets are “baked” into the devices, and can be used by Microchip to provide assurance that they are legitimate devices made according to the correct design.

The devices include the following non-volatile memory blocks for storing the security keys and passcodes:

- Private NVM (pNVM)
- Secure NVM (sNVM)

To improve the security of the non-volatile storage used, all passcodes are hashed and all keys are enciphered as key codes by the SRAM-PUF before being stored. The SRAM-PUF then reconstructs the decryption keys from the key codes before being used. Thus, an attacker who manages to somehow measure the states of the non-volatile cells or monitor a data bus to/from the non-volatile storage cannot directly learn the actual passcode or encryption key.

Multiple user-selectable key modes are available and a secure mechanism (SPPS) is provided to update encryption keys and passcodes.

2.2 Factory Keys (Ask a Question)

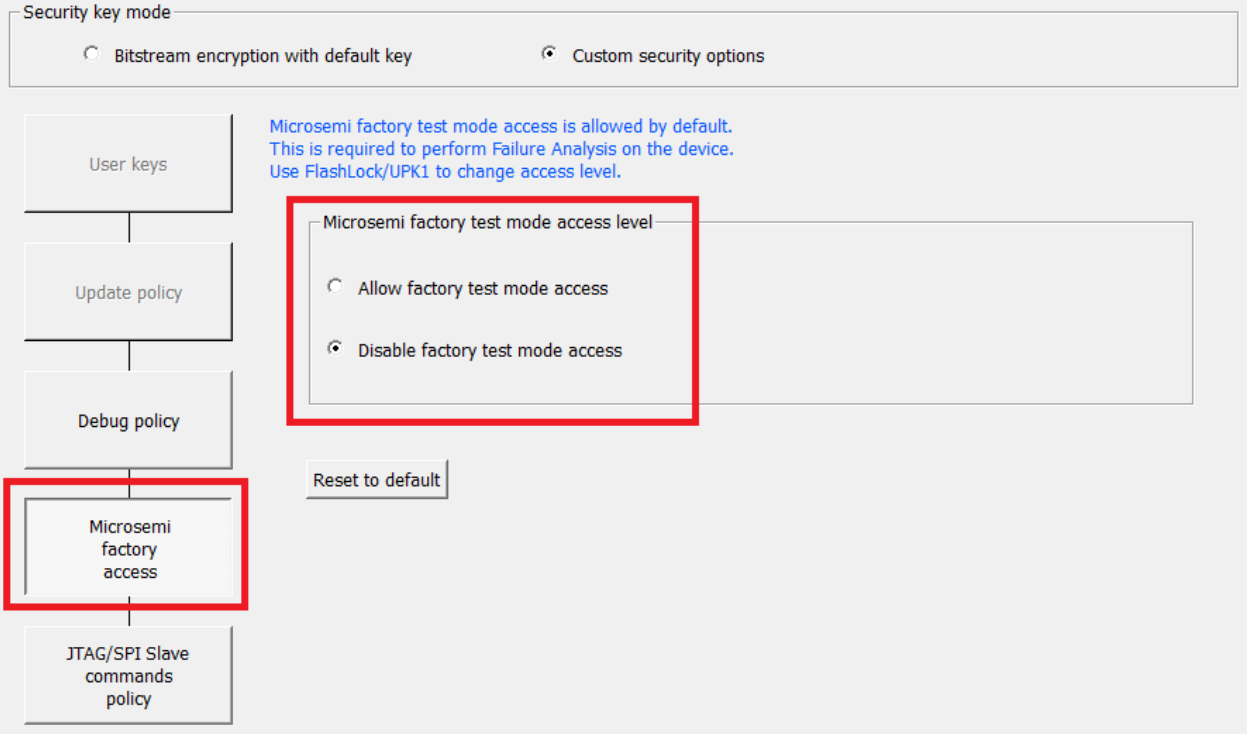
The section describes the factory keys and parameters present in the devices.

2.2.1 Factory Passcode Key (FPK) (Ask a Question)

The factory passcode Key (FPK) is a 256-bit unique passkey used to set the device in factory test mode. FPK can override all non-permanent factory-defined locks. The factory test mode allows in-depth testing of the device by Microchip for failure analysis. When a passcode is entered for attempted validation, the entered passcode is hashed, and compared with the hashed version present in the device.

User must enable (as shown in the following figure) the factory test mode access if the user wishes Microchip to perform failure analysis on a given device. The user can permanently disable factory test mode access. When that is done, no future failure analysis is possible.

Figure 2-1. Factory Test Mode Access



2.2.2 Device Serial Number (Ask a Question)

The device serial number (DSN) is a 128-bit unique device ID. It comprises of two parts—the factory serial number (FSN) and the serial number modifier (SNM). The first part of the device serial number is the 64-bit FSN that uniquely identifies a device. The DSN is zeroized if “unrecoverable” zeroization action is performed on the device. For more information about zeroization, see Zeroization.

The device serial number can be read by the user logic using a system service call. For more information, see PolarFire FPGA and PolarFire SoC FPGA System Services User Guide.

2.2.3 Microsemi Certificate Public Key (MCPK) (Ask a Question)

The Microsemi certificate public key (MCPK) is a trusted immutable NIST P-384 768-bit elliptic curve public key used for checking the Microsemi signature of the device's X.509-compliant certificate—also known as the supply chain assurance certificate. MCPK is stored in the pNVM. The system controller uses this key to verify the signature on its own certificate every time it is requested to export it. Although MCPK is a public key used in multiple devices of the same type and vintage, it is encoded by the PUF to provide confidentiality and to check for authenticity, in order to prevent tampering.

2.2.4 Key Loading Key (KLK) (Ask a Question)

The key-loading key (KLK) is the default 256-bit symmetric key used to encrypt any of the flash configuration components present in a bitstream. It is used to load user keys and security settings in situations where high levels of security are not required. One such situation could be where programming is done in a completely trusted secure facility with cleared personnel and stringent data handling and protection processes in place. Another is where the design IP is not very valuable and security is not a primary concern for the user. In this case, KLK can be selected as the root key for encryption and authentication of the bitstream component used to load the user keys.

The KLK is common to a relatively large number of devices of the same type and version, and resides within the programming tool software. This makes it the easiest key to use, but is not as secure as the other options, having a “software” rather than a “hardware” level of protection.

When the user keys are loaded, the KLK is automatically disabled by a user lock bit reserved for this purpose, without any action required by the user. After this point, any programming update requires using the user keys.

Microchip offers HSM based secure production programming solution for loading keys in untrusted environments. For more information, see Secure Production Programming Solution (SPPS) User Guide.

2.2.5 Factory ECC Key (Ask a Question)

The Factory ECC Key (KFP) is the device unique 384-bit private NIST P-384 elliptic curve key. The corresponding public key (KFPK), unique for each device, is certified in the device's X.509 supply chain assurance certificate.

The primary use model is to support initial loading of user keys, wherein an ECDH operation is executed to derive a shared secret key to encrypt a bitstream containing the user keys. Since the public key is certified by Microchip in the supply chain assurance certificate, the user can be assured that the communication transpires with an authentic device and not a clone or a man-in-the-middle. KFP can also be used as a signing key for device-generated certificates via Digital Signature system service. For more information, see PolarFire FPGA and PolarFire SoC FPGA System Services User Guide. Therefore, the authenticity of any such certificate can be checked using the public key from the supply chain assurance certificate, providing a strong cryptographic chain to Microchip and the device PUF.

To utilize KFP and the associated public-key method to provision user keys into a device requires use of the optional Secure Production Programming Solution (SPPS) available from Microchip.

There are two available key modes based on factory ECC key:

- One key mode is KFP, in which the device uses the certified key pair and the HSM uses a randomly generated ephemeral key pair. They follow the ECDH protocol to derive the shared secret key.
- The other key mode is KFPE, in which the device uses the certified key pair along with a second randomly generated ephemeral key pair, and the HSM uses two randomly generated ephemeral key pairs. In this case, the ECDH protocol is run twice, which results in two shared secret keys that are used in another round of key derivation to generate a single shared secret key. This key mode is preferred over KFP key mode, since it uses randomly generated key pairs and therefore is more secure. However, this key mode takes longer, because there are two ECDH operations and key generations.

2.2.6 Factory Key (FK) (Ask a Question)

The factory key (FK) is a 256-bit symmetric AES key unique to each device. It is a secure, quantum-safe alternative to KFP that can be used to load the user keys if it is selected as the root key for encryption and authentication of the bitstream component containing them. After the user's security settings are loaded, the factory key is automatically disabled for encryption purposes by a user lock bit without any action required by the user.

Since the factory key is a symmetric key, the programmer must know the related key (Diversified Factory Key for every device) in order to prepare bitstreams that can be decrypted by the devices, or to verify that the device is familiar with the factory key. This key mode requires the Microchip SPPS. Microchip customers who use the SPPS solution are given a database of the Diversified Factory Keys (DFKs) upon registering their U-HSM via the Microchip Portal. Upon registration, a UUID is assigned to the U-HSM. The Customer UUID is used to diversify the Diversified Factory Key Database for the customer. This prevents anyone else with a key database and HSM from decrypting another user's bitstream files.

FK is destroyed by the unrecoverable zeroization mode actions. See Zeroization for more information.

2.2.7 PUF Emulation Key (PEK) (Ask a Question)

The PUF emulation key (PEK) is a 256-bit factory-defined key unique to each device that is randomly generated by the device during initialization of the factory keys. This key is used only in the PUF emulation protocol.

Note: The PEK is not generated by the PUF circuit every time the user recalls it, but is instead a static programmed value that is read from Flash bits.

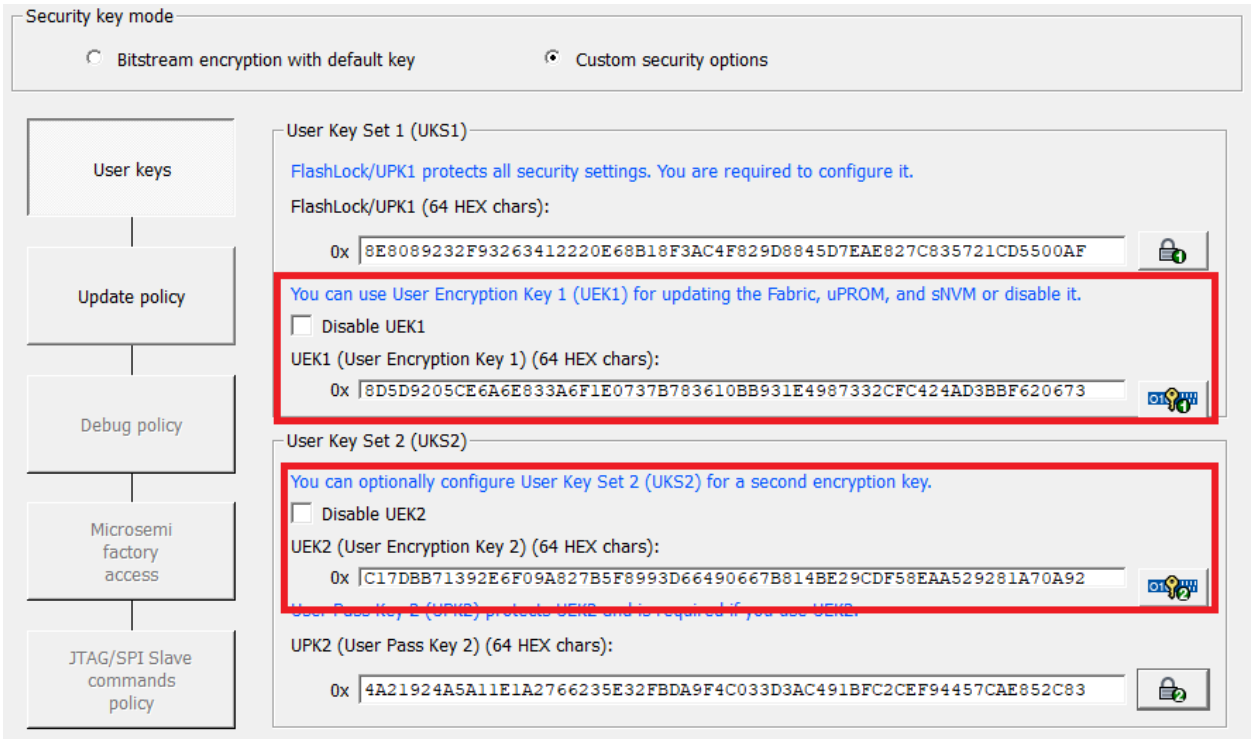
2.3 User Keys and Passcodes (Ask a Question)

The following sections describe the user keys and passcodes. The Configure Security Wizard in the Libero software must be used to set the user keys and passcodes.

2.3.1 User Encryption Keys (UEK1 and UEK2) (Ask a Question)

The user encryption key1 (UEK1) and user encryption key2 (UEK2) are user-defined 256-bit symmetric keys. These keys are wrapped by the PUF as key codes and then stored in the pNVM. Either of these keys can be used as the root key for encrypting and decrypting bitstreams, and to authenticate them.

Figure 2-2. User Encryption Keys—UEK1 and UEK2



Use of UEK2 is strictly optional. Having a second user key can facilitate use models that would be difficult to implement with just one user key. The UEK2 can be used to update a subset or class of products in the field. For example, UEK1 could be common across all the devices in a project, and UEK2 could be unique in every device. Thus, when preparing an update that is only intended to go into one device, the UEK2 from that single device could be used as the root key, thus preventing the bitstream from being copied and loaded into other devices in the project accidentally or maliciously.

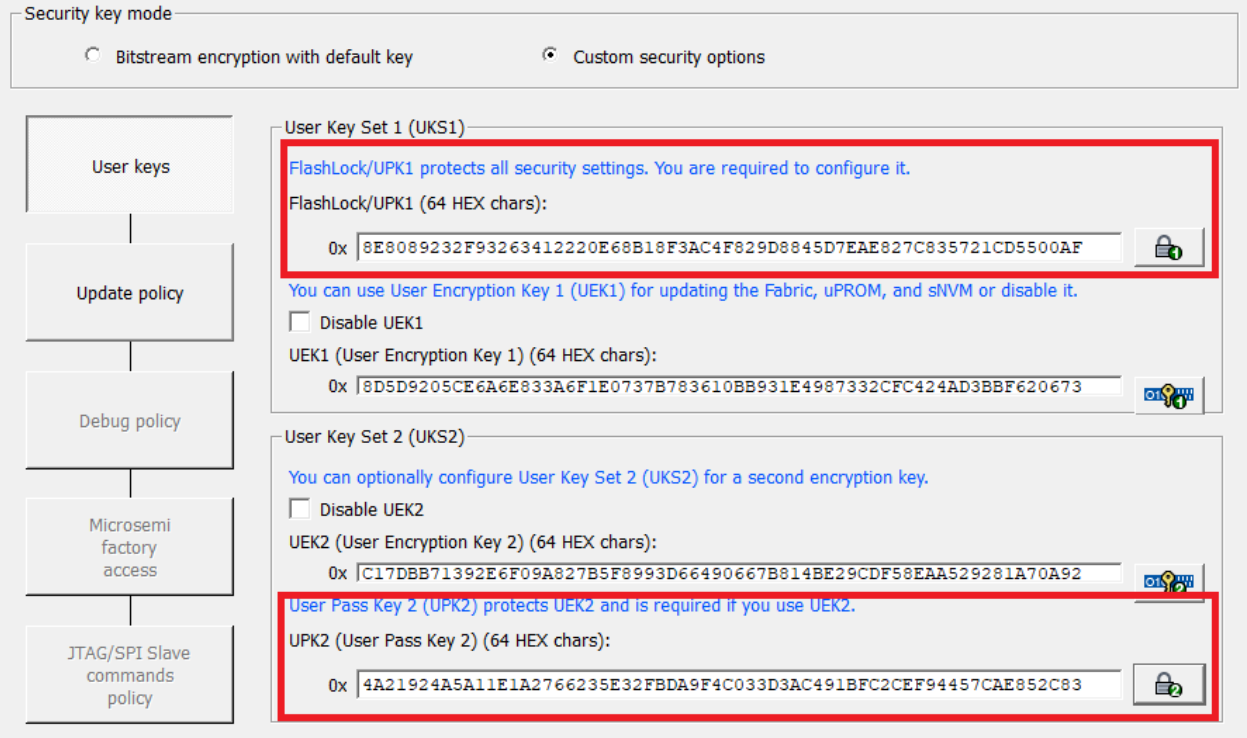
2.3.2 User Passcode Keys (UPK1 and UPK2) (Ask a Question)

The user passcode key 1 (UPK1), also known as the FlashLock passcode, is the primary user passcode that unlocks the majority of non-permanent user-defined locks when matched by the user. The user passcode key 2 (UPK2) is the secondary user passcode protecting the secondary user

key segment, which contains UPK2 and UEK2. When the UPK2 is matched, it allows itself or UEK2 to be overwritten. UPK1 and UPK2 can be matched using either the plaintext, the one-time-use passcode protocol, or the one-way passcode protocol (for PolarFire SoC FPGA only).

These passcodes are loaded along with the other user keys using an encrypted bitstream, and are stored in the user key segments of pNVM. Passcodes are never used for encryption. They are used only for escalating privileges during the session when the passcode is matched successfully. The privilege escalation provided by UPK1 or UPK2 stays in effect only until the device is reset or power cycled.

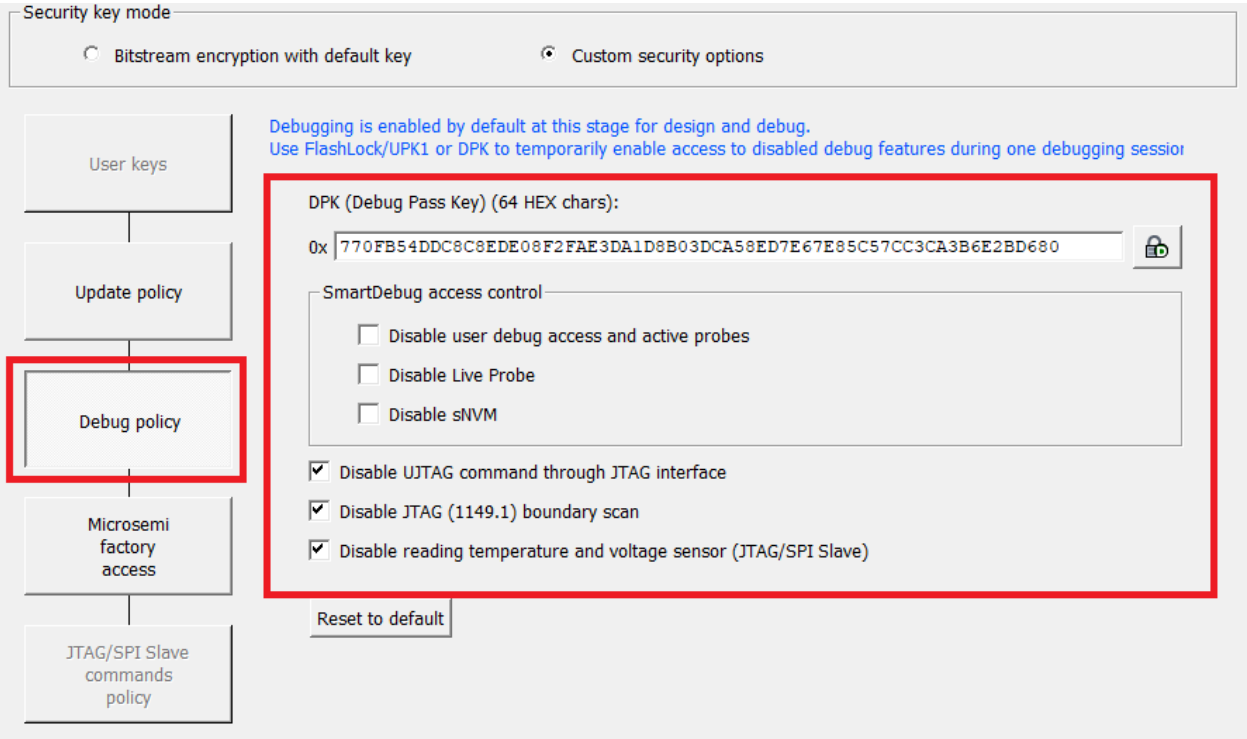
Figure 2-3. User Passcode Keys—UPK1 and UPK2



2.3.3 Debug Pass Key (DPK) (Ask a Question)

The debug pass key (DPK) is a debug passcode that overrides all debug-related locks. DPK may be unlocked using the plaintext or the one-time-use passcode protocol. It stays in effect only until the device is reset or power cycled.

Figure 2-4. Debug Pass Key



2.3.4 User ECC Keys (KUP and KUPE) (Ask a Question)

There are two key modes, which are based on user ECC key. These key modes are defined using the device EC capabilities, which allow secure bitstream programming without requiring the user to have first programmed any secret keys into the device.

The ECC key modes use ECDH to derive a shared secret between device and Microchip HSM, which can then be used as root key either for encrypting/authenticating a bitstream (such as for injecting User symmetric keys and security settings in a new device), or for authenticating the device using the key confirmation challenge-response protocol. After the user ECC key is used to load the User's symmetric keys, it is automatically disabled for loading any further bitstreams without any action required for the User. However, it can still be used for device authentication purposes using the key confirmation protocol.

The devices support JTAG and SPI instruction that can create and/or retrieve the user ECC public key associated with the user EC private key (KUP). When the public key is exported, it is signed by the private key (KFP) of the device's factory-certified ECC key pair. This provides a verifiable method of creating a NIST ECC P-384 key pair and securely exporting the public key in a way that can avoid man-in-the-middle attacks on it. Thus, the device can be enrolled in a user public key infrastructure (PKI) by having a certificate authority (CA) sign (certify) the exported public key. After it has been verified, the device provides proof-of-possession (PoP) of the private key of the key pair, and any other steps deemed necessary by the CA or local registration authority (LRA).

2.3.4.1 KUP Key Mode (Ask a Question)

KUP is the 384-bit NIST P-384 user private ECC key. It is protected by SRAM-PUF, neither leaves the device nor is it ever exported to the user of the FPGA internally. The key is randomly generated by the device during the initial key loading process using SPPS. This key can be used for secure initial loading of User keys (such as UEK1 and UEK2) using SPPS. The corresponding public ECC key can be exported. When the public key is exported, it is signed by the device's factory certified ECC private key. This provides a verifiable method of creating a NIST ECC P-384 key pair and securely exporting the public key in a way that can avoid man-in-the-middle attacks on it.

2.3.4.2 KUPE Key Mode (Ask a Question)

This key mode uses KUP and a device ephemeral key. Two ECDH shared secrets are generated from these keys and a user/HSM supplied EC public keys. The shared secrets are then used to derive a shared symmetric key, which is used for secure loading of user keys using SPPS. This is more secure than KUP key mode because of randomly generated ephemeral key. However, this key mode takes longer because there are two ECDH operations and key generations.

2.3.5 sNVM Master Key (SMK) (Ask a Question)

The sNVM master key (SMK) is a 512-bit symmetric key for securing the content of the sNVM. It is the concatenation of a 256-bit key used for authentication and another 256-bit key used for encryption. It is randomly self-generated on each device, so it is unique for each device. The key is stored as an encrypted and authenticated key code in the pNVM using the PUF key-wrapping mechanism. It is the primary key used for the optional user-specified authentication or authenticated encryption of sNVM pages using AES and the synthetic initialization vector (SIV) cipher mode. SIV mode effectively incorporates a tweak that allows each page to be encrypted and authenticated with different resulting ciphertext and authentication tags, even if the plaintext contents are the same. For more information, see Secure Non-Volatile Memory (sNVM).

Note: The authenticated writes to the sNVM using the system services pass only after the SMK (sNVM Master Key) is successfully generated by the device. To generate the SMK, program the device with an authenticated client in sNVM using Libero SoC. When the SMK is generated, it can be used for performing authenticated writes to the sNVM through system services firmware.

Note: When the SMK is generated, it can be modified, but not deleted. Every time authenticated/ciphertext sNVM clients are programmed through Libero SoC/FP Express. SMK key is re-generated or initialized on new blank device. SMK cannot be generated using system services.

3. Bitstream Security (Ask a Question)

The PolarFire family has a layered protection to ensure that the user's intent is met. These protection layers include the use of encryption to protect the confidentiality of the design IP and prevent reverse engineering, and authentication to ensure that only legitimate bitstream files are loaded by devices.

All programming operations including erase, program, verify, and security key management are managed by the system controller. In programming mode, the system controller authenticates and decrypts the incoming bitstream, erases and writes the target programmable sub-blocks. The system controller and associated security hardware includes hardware-based security countermeasures to protect the device against a broad range of threats.

3.1 Bitstream Content (Ask a Question)

The devices are configured using a Microchip proprietary and confidential format bitstream file. The device bitstreams are divided into four main components—FPGA fabric, sNVM, eNVM, and security —that can be targeted during the configuration process.

- **FPGA fabric**—the FPGA fabric configuration component holds the configuration bits that configure the routing switches and look-up tables of the logic elements that define the user's design, as well as the I/O cells, embedded memories, math blocks, transceiver blocks, uPROM, and so on.
- **sNVM**—the sNVM configuration component contains programming data for one or more sNVM pages.
- **eNVM** (for PolarFire SoC FPGA Only)—the eNVM configuration component contains programming data for one or more eNVM pages.
- **Security**—the security component holds cryptographic keys, passcodes, and lock bits required for design security. All the keys are stored in encrypted form and all the passcodes are stored only after cryptographically hashing them.

3.2 Bitstream Encryption and Authentication (Ask a Question)

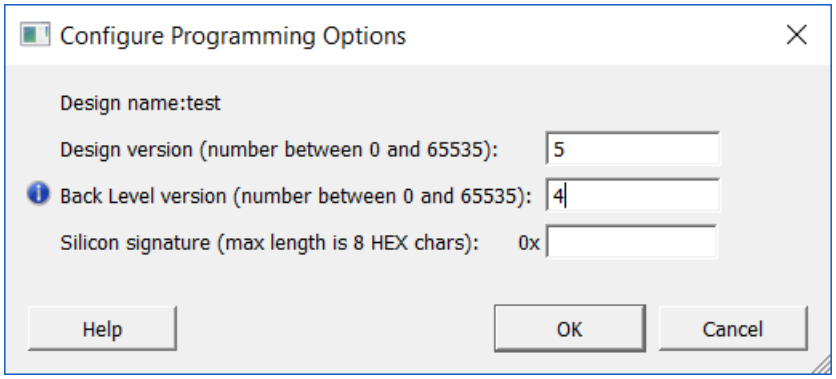
The device bitstream is always encrypted and authenticated to be DPA-resistant. A factory-defined default key (KLK) is used when the user does not specify a user encryption key (UEK1/UEK2), thus disallowing the use of a plaintext bitstream.

The bitstream is encrypted with the AES-CTR mode using 256-bit secret key, and then an authentication tag is added using a symmetric message authentication code (MAC) based on SHA-256. The encryption key is permuted after every ciphertext block to protect the AES encryption from DPA attack. The bitstream authentication is provided by a licensed protocol from CRI. This protocol uses SHA-256 and proprietary algorithms for checking authenticity in a way that resists differential power analysis and other side-channel attacks. Under this protocol, data is never decrypted unless it has first been authenticated. Additionally, decryption keys are hashed frequently to improve DPA resistance. DPA resistance is also included in the AES and SHA algorithms implemented in the system controller's cryptographic processor.

3.3 Back-Level Protection (Bitstream Versioning) (Ask a Question)

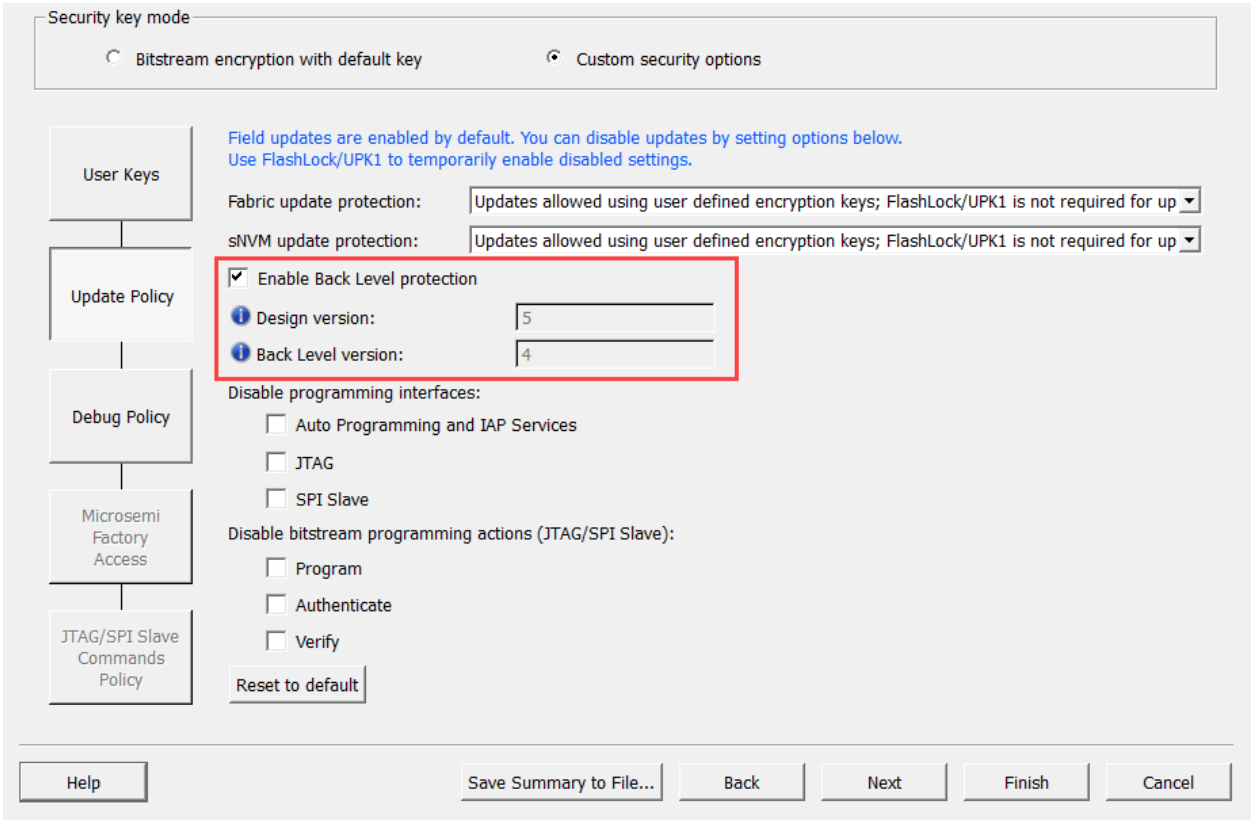
The devices allow protection against a replay attack, where an earlier form of a bitstream (one perhaps with security vulnerabilities) may be reintroduced to gain information about a system. The user can assign a version number to each configuration bitstream, and add a back-level version using the **Configure Programming Options**, as shown in the following figure.

Figure 3-1. Setting Bitstream Version and Back Level Version



The back-level version value restricts the design version that the device accepts as an update. The **Back Level version** must be smaller than or equal to **Design version**. The back-level version number must be set higher than all of the (old) versions that the user wishes the device to reject. So, only (new) programming bitstreams with a **Design version** strictly greater than the current **Back Level version** stored on the device are allowed for programming. The new back-level version programmed along with an accepted bitstream affects any future bitstreams, and can be higher or lower than the back-level it overwrites, at the user's discretion. Back-level protection is secured by FlashLock/UPK1, which can be used to bypass it. The back-level protection must be enabled using the **Configure Security Wizard > Update Policy** as shown in the following figure. The **Back Level version** number is not programmed into the device if Back Level protection is disabled.

Figure 3-2. Back Level Protection



3.4 Initial Key Loading (Ask a Question)

When a device is blank, there are no user secrets on the device that can be used to encrypt the bitstream to load user keys. On the devices, there are three approaches for handling this:

- A factory-defined default key (KLK) may be used to load user keys. The KLK is common to a relatively large number of devices of the same type and version, and resides within the programming tool software.
- A key, DFK, derived from a device-unique factory key FK, may be used to load user keys. This allows a bitstream to be encrypted for a specific device. SPPS is required for this scheme.
- ECDH based scheme whereby the device and the programmer dynamically derive an ephemeral shared secret key unique to each FPGA and each session. This allows to truly secure the initial user key loading since the shared key is never exposed. It is ephemeral and only exists for a short time during the initial user key provisioning, only the Microchip certified target device is capable of decrypting the bitstream. SPPS is required for using this scheme.

The KLK based approach provides customer not concerned with bitstream security a simple method for programming devices that does not require the SPPS. Because all the devices support the ECDH based scheme, DFK mode is mainly reserved as an alternative quantum-safe mode, in case, quantum computing makes elliptic curve cryptography obsolete. Until then, in most cases, users equipped with the SPPS may want to use the ECDH-based key scheme.

3.5 Certificate of Compliance (C-of-C) (Ask a Question)

As new devices can be programmed by those possessing them, the devices must either be initialized using user keys in a trusted facility with vetted personnel, or another method should be used to ensure that the correct keys, security settings, and user-supplied design are programmed into the devices. The best practice is to bring the fully-assembled and programmed systems to a trusted facility where the programming can be verified, before they are put into any sensitive applications. Either of these approaches—using a trusted facility to preload keys, or afterwards, to verify programming—requires extra time and expense. This may include the cost of maintaining such facilities and staff, and the inconvenience of not being able to put otherwise finished systems into service until additional steps have been performed.

The PolarFire family offers an alternative approach that uses cryptographic techniques to provide assurance that they are programmed correctly, and not with some malicious entity's keys instead of the user's keys, or with (intentionally) wrong security settings, or with a different design than intended (perhaps containing a Trojan Horse).

During programming, the device can generate a short message called a certificate of conformance (C-of-C). This includes the keyed digests (message authentication code tags) for each bitstream component programmed. The input data for digest calculation includes the data programmed by that bitstream component, and the device serial number. This ensures that the C-of-C tag from each device is unique, even if the programmed data is the same. The key makes the tag impossible to forge.

The SPPS software can validate the returned C-of-C messages from each device and report the status in secure log files. This is one aspect of keeping tight accounting control over the number and identity of the parts programmed, the scrapped parts, and so on. The C-of-C proves that each component is programmed with the expected data.

The advantage of the C-of-C approach is that it provides this assurance minus the expense of shipping parts or systems around the world between less-trusted assembly facilities and more-trusted facilities where additional programming or verification steps must be performed, and it may even eliminate the need for the more expensive facilities. Programming can be performed in a less expensive facility without the risk of undetected tampering of the programming data. Generating and confirming the C-of-C is very efficient, and adds almost nothing to the programming time. It is completely automated by the Microchip SPPS tools.

3.6 Digests (Ask a Question)

Digests are used for protecting data integrity. In PolarFire®, digests are used to protect the integrity of the user design programmed into the device as well as the programming files used for device programming. Digests are the result of the SHA-256 hash executed over the programmed device content. The resulting digest value is highly dependent on the device programming information and can be used with high confidence to determine a change in the content programmed into the device.

To assure the integrity of the user design programmed into the device, the device system controller generates digests, known as **Component Digests**, for each component in the device. These component digests are calculated and stored within the device during device programing. Multiple digests are calculated including factory and security segment digests, FPGA fabric component digest, sNVM and eNVM (for PolarFire SoC FPGA only) digests for pages marked as ROM. These digests can be verified on-demand by the user, either internally using a system service, or externally using a programming instruction. In addition, the user can automatically run digest checks on each power-up. These checks assure no device configuration changes occurred, either maliciously or naturally, since the component was last programmed. Any mismatch in the digests checks is an indication that the programmed content, currently residing in the device, does not match the content previously programmed into the device during device programming. The device must be reprogrammed to correct this mismatch.

The following table lists the **Component Digests** generated and stored in the device.

Table 3-1. Component Digests

Digest Name	Description
Fabric digest (CHECK FABRIC digest)	Digest of fabric configuration
Fabric configuration data digest (CC Digest)	Digest of fabric configuration parameters such as cycle count, design version, and back level protection value
sNVM ROM pages digest	Digest of sNVM pages marked as ROM
User security segment digest (UL Digest)	Digest of user security settings
UKDIGEST0	Digest of user key segment containing SRAM-PUF data
UKDIGEST1	Digest of user key segment containing KUP (User EC key)
UKDIGEST2	Digest of user key segment containing UPK1
UKDIGEST3	Digest of user key segment containing UEK1
UKDIGEST4	Digest of user key segment containing DPK
UKDIGEST5	Digest of user key segment containing UPK2
UKDIGEST6	Digest of user key segment containing UEK2
UPDIGEST (UPERM Digest)	Digest of permanent lock security segments
FDIGEST (SYS Digest)	Digest of factory lock segment, factory key segment in pNVM and System Controller ROM
UKDIGEST7	Digest of One-Way Passcode HWM (For PolarFire® SoC FPGA only)
ENVMDIGEST	Digest of eNVM (For PolarFire SoC FPGA only)
UKDIGEST8	Digest of MSS Boot mode Information (For PolarFire SoC FPGA only)
UKDIGEST9	Digest of SNVM_RW_ACCESS_MAP (For PolarFire SoC FPGA only)
UKDIGEST10	Digest of Secure Boot Image Certificate (SBIC) (For PolarFire SoC FPGA only)

The integrity of device programming bitstream files is also protected by digests. When the Libero® SoC design tool generates a programing bitstream file, a set of **Bitstream Payload Digests** are generated. These digests differ from the Component Digests. While the Bitstream Payload Digests are calculated over the component programing data and the meta data required to program the device, **Component Digests** include only the programmed data. When the programming bitstream

file is generated by the Libero SoC design tool, the data for each component is hashed (SHA-256) and the resulting digest value is retained in an accompanying file along with the programming bitstream file.

During device programming the FPGA, flash cells are loaded with the programming file data and a Bitstream Payload Digest is calculated on the fly. This digest is compared to the Bitstream Payload Digest stored in the bitstream programming file to assure the device correctly received the bitstream programming file during the programming operation.

The following table lists the Bitstream Payload Digests that are generated during programming file generation:

Table 3-2. Bitstream Payload Digests

Digest Name	Description
Security component bitstream digest	Digest of all configured device security settings
Fabric component bitstream digest	Digest of the FPGA fabric configuration
eNVM component bitstream digest	Digest of all ROM content of eNVM (PolarFire® SoC only)
sNVM component bitstream digest	Digest of all ROM content of sNVM
Entire bitstream digest	Combined digest of all components

The following sections describe various options available to check digest. For more information about Digest Check System Services, see PolarFire FPGA and PolarFire SoC FPGA System Services User Guide.

3.6.1 Power-On Reset Digest Check (Ask a Question)

The device may be configured to perform automatic component digest checks while powering up the user design (after power-on reset) to check the integrity of the selected components. The user can specify which digest to check. If any of the selected digest checks fails, a tamper event is generated to fabric for user action. A digest check failure is an indication that the programmed content currently in the device is not what was last programmed into the device and cannot be trusted and that the device must be reprogrammed. The power-on digest check can be enabled and monitored using PF_TAMPER macro for PolarFire FPGA and PFSoC_TAMPER for PolarFire SoC FPGA.

Figure 3-3. Power-On Reset Digest Check Controls for PolarFire® FPGA

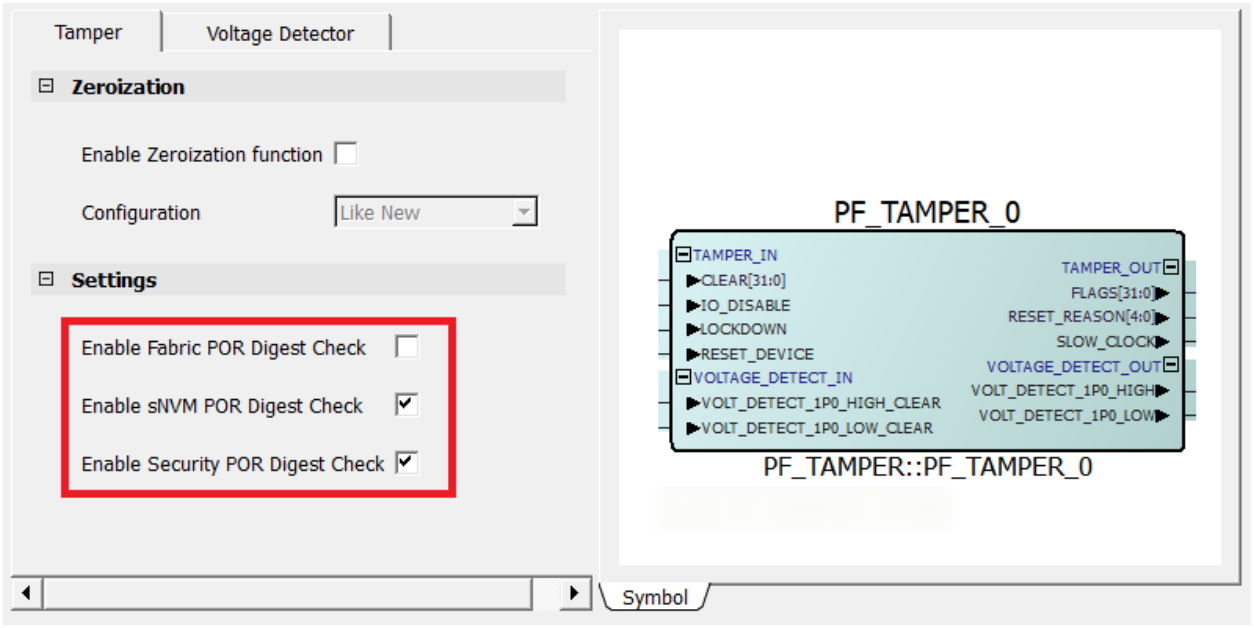
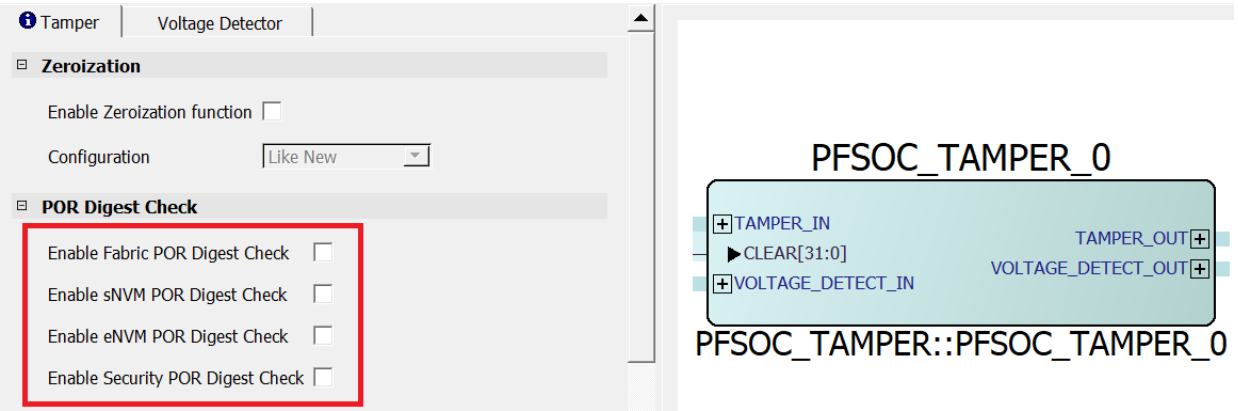


Figure 3-4. Power-On Reset Digest Check Controls for PolarFire® SoC FPGA



For example, if the first-stage boot code for a soft CPU is stored in the sNVM or eNVM (for PolarFire SoC FPGA only), then the power-on reset digest check could be used to automatically provide a high level of assurance that the code had not been changed, either through a natural or malicious event, since the digest was stored.

The following table lists the digests of the POR Digest Check to the actual digests that are checked.

Table 3-3. POR Digest Check

Digest Name	POR Digest Check	Comment
Fabric digest (CHECK FABRIC digest)	Fabric	—
Fabric configuration data digest (CC Digest)	Fabric	—
sNVM ROM pages digest	sNVM	—
User security segment digest (UL Digest)	Security	—
UKDIGEST0	—	Not currently supported by Libero SoC
UKDIGEST1	—	Not currently supported by Libero SoC
UKDIGEST2	Security	—
UKDIGEST3	Security	—
UKDIGEST4	Security	—
UKDIGEST5	Security	—
UKDIGEST6	Security	—
UPDIGEST (UPERM Digest)	Security	—
FDIGEST (SYS Digest)	Any	Check when any POR Digest Enabled
UKDIGEST7	Security	—
ENVMDIGEST	eNVM	—
UKDIGEST8	eNVM	—
UKDIGEST9	eNVM	—
UKDIGEST10	eNVM	—

A read-endurance limit specifies how many times a digest of the FPGA fabric can be run before the long-term reliability of the FPGA configuration data could be affected. For more information about the FPGA configuration memory endurance limits, see respective PolarFire FPGA Datasheet or PolarFire SoC Datasheet. Therefore, depending upon how the system is deployed and used (for example, how often it is powered-up), the on-demand digest check may be more appropriate for testing the integrity of the FPGA fabric.

3.6.2 On-Demand Digest Check (Ask a Question)

On-demand digest check recalculates and compares digests of selected components with the stored digests. A failure of any digest results in the tamper event being triggered. A digest check failure is an indication that the programmed content currently in the device is not what was last programmed into the device and cannot be trusted and that the device must be reprogrammed. The on-demand digest check is invoked by calling digest check design system service. For more information about running system services, see PolarFire FPGA and PolarFire SoC FPGA System Services User Guide.

Digest check can also be executed over JTAG using the VERIFY_DIGEST action. This action verifies all component digests, returning PASS/FAIL for each component digest along with the digest value.

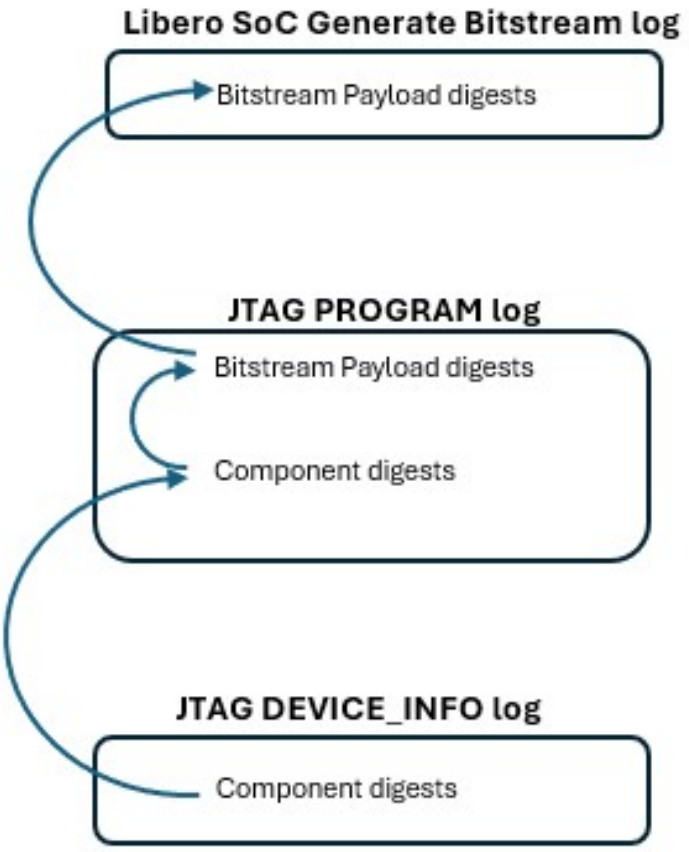
A read-endurance limit specifies how many times a digest of the FPGA fabric can be run before the long-term reliability of the FPGA configuration data could be affected. For more information about the FPGA configuration memory endurance limits, see respective PolarFire FPGA Datasheet or PolarFire SoC Datasheet.

3.6.3 Exporting Digests (Ask a Question)

The stored component digests can be exported through a design system service or the JTAG or SPI-slave interface. This can be useful to determine the programming file used to program the device. The Read Digests system service returns the stored digests. For more information about running system services, see PolarFire FPGA and PolarFire SoC FPGA System Services User Guide. The stored Component Digests can also be exported with the DEVICE_INFO JTAG instruction. These Component Digests can be used to verify the programmed content of the device.

Recall that during programming file generation each bitstream component (for example, FPGA, eNVM and so on) is hashed (SHA-256) and the resulting digest value is retained along with the bitstream file. The Libero SoC design tool stores these Bitstream Payload Digests in bitstream log files. When the device is programmed, the device generates Component Digests and stores them in the device. A programming log file, that contains both the Bitstream Payload Digests and the Component Digests, is generated after the JTAG programming completes. After exporting the Component Digests, compare these to the values in the programming log file. If the exported Component Digests match those in the programming log file then the Bitstream Payload Digests in the programming log file are the digests of the file used to program the device. Compare these Bitstream Payload Digests to the bitstream log files to determine which bitstream was used to program the device. The following figure shows the digest traceability process.

Figure 3-5. Digest Traceability



The process is only valid, if the device was JTAG programmed because this process generates a programming log file. SPI initiator (IAP/Auto-Update) device programming does not generate any log files that can be used to trace device programming content.

4. Hardware Access Controls (Ask a Question)

The PolarFire family implements the following configurable hardware access controls to prevent overwriting device configuration:

- **Passcode security**—protects against unauthorized changes to security policies.
- **Security locks**—enforces more granular write-protections when compared to passcode security for fabric, sNVM, and security segments.

4.1 Passcodes (Ask a Question)

Security locks protect sensitive device functions. Passcodes protect these security locks. Passcodes are used only for escalating privileges during the session when the passcode is matched successfully. All passcodes are 256 bits long, making a brute-force attack infeasible. Passcodes are salted and hashed when stored. The salt is unique to each device.

The PolarFire family includes the following passcodes:

- FlashLock™ or User Passcode Key 1 (UPK1)
- User Passcode Key 2 (UPK2)
- Factory Passcode Key (FPK)
- Debug Passcode Key (DPK)

The configure security wizard in the Libero SoC software is used to set these passcodes for the user design.

4.1.1 FlashLock or User Passcode Key 1 (UPK1) (Ask a Question)

The FlashLock, also known as UPK1, is the primary user passcode that unlocks a majority of the non-permanent user-defined locks when matched by the user. This passcode is loaded along with the other user keys and passcodes using an encrypted bitstream, and is stored (after being hashed) in the primary user key security segment in the pNVM.

The FlashLock passcode may be matched using either the plaintext, or the one-time-use passcode protocol, or one-way passcode protocol (for PolarFire SoC FPGA only). When the FlashLock passcode is entered for attempted validation, it is hashed and compared with the stored hashed passcode. If the hashed value of the passcode is successfully matched, it temporarily allows changes to the data items normally protected by the affected user-defined locks. The device returns to normal locked state (as defined by the non-volatile lock bit settings) on the next device reset or JTAG reset, or power cycle.

A plaintext passcode does not provide selective unlocking of individual locks in its associated class. Furthermore, a plaintext passcode is subject to possible monitoring attacks. Once known, it can be used to escalate privileges associated with that passcode forever (unless plaintext passcodes are prohibited by the device's security settings). The use of plaintext passcodes is therefore discouraged. The plaintext passcodes are supported to maintain legacy functionality.

The one-time-use passcode protocol is a challenge-response type Online protocol in which the User (or Factory) proves to the device that they know the selected passcode and key. The one-time-use passcode protocol permits selective unlocking of individual locks. When a one-time-use passcode is used, a subset of locks to override may be specified in the one-time-use passcode protocol. The one-time-use passcode protocol uses a nonce, and generates an encrypted one-time passcode using the user selected root key. The HSM used in SPPS flow supports the one-time-use passcode protocol. The one-time-use passcode protocol can be used to match the FlashLock passcode (without revealing its value outside the SPPS HSM). One possible use of the FlashLock passcode is to have it allow bitstream updates again, even if overwriting the FPGA fabric or sNVM was disabled by locks in the security policy stored on the device.

The One-Way Passcode (OWP) protocol (for PolarFire SoC FPGA only) is used for overriding locks via a bitstream without requiring any interaction with an external intelligence. It is similar in concept to the One-Time Passcode protocol in that it can only be used once, has selective unlocking capability but does not require a FPGA nonce and is bounded to a specific bitstream. It pertains to locks associated with UPK1 and UPK2. The One-Way Passcode protocol allows one-time passcodes to be created offline and it is a one-way communication protocol unlike one-time-use passcode protocol. The primary use-model is to support one-time bitstream updates where the one-way passcode is bound to a given bitstream. The bitstream created with OWP temporarily undo locks that would otherwise prevent the bitstream from being used. When the bitstream is complete, or terminates for any other reason, locks are restored to their original non-volatile states. The bitstream created with OWP is protected by the standard bitstream encryption and authentication mechanisms supported by PolarFire SoC FPGA.

The Configure Security Wizard in the Libero SoC software is used to apply the FlashLock passcode for the user design. It is recommended to allow the SPPS with a hardware security module (HSM), when available, to select the FlashLock passcode (and all other user keys and passcodes) rather than importing it manually or generating it on a less-secure general-purpose workstation. The HSM generates a high-quality 256-bit true random bit string, and stores it securely within its certified hardware security boundary. The random bit string if exported for storage on the host workstation, injection into the device (as a bitstream), or for unlocking the device (using the onetime passcode protocol), is strongly encrypted. The HSM does not export secret keys or passcodes in plaintext form, even to the host workstation.

4.1.2 User Passcode Key 2 (UPK2) (Ask a Question)

User passcode key 2 (UPK2) is the secondary user passcode protecting the secondary user key segment of pNVM, which contains UEK2. Like all passcodes, UPK2 may be matched using the plaintext or the one-time-use passcode protocol. When matched, it allows itself or UEK2 to be overwritten.

4.2 FPGA Security Locks (Ask a Question)

The factory and user security segments holds various lock bits. These lock bits acts as access control bits to the security segment to which they are applied. The factory lock bits are set and locked in the factory security segments before shipping the parts. The user lock bits are set and locked in the user security segments. Some factory lock bits prohibit the same function as a user lock bit. In this case, if either one is set, the function is disabled.

The user lock bits can be temporarily unlocked using the appropriate passcode assigned to that bit. Some lock bits can only be modified by erasing or overwriting the security segment to which they belong using an encrypted and authenticated bitstream. If lock bits are unlocked using a passcode, it is just temporary until the next device reset, JTAG reset, or power-down. Any permanent change to the user security segments must come from a bitstream and take place after the reset, or at the next power-up cycle.

Although a lock bit may be referred to in the singular in this document, that is just a reference to its logical existence. All lock bits are stored with physical redundancy. The most important lock bits, from an anti-tamper perspective, also use parity bits to detect any loss of integrity. These bits are monitored continuously during run-time, and generate a tamper detection flag immediately if a tamper event is detected. This process is independent of whether there is any programming or security-related operation going on in the FPGA. All the lock bits are monitored at the time they are consumed, by re-computing and comparing a digest value before using the stored data.

The Configure Security Wizard in the Libero SoC software is used to apply these lock bits.

4.2.1 User Security Locks (Ask a Question)

This group of locks prevents erasing and overwriting UEK1, UEK2, and the user security segments. These locks are automatically set when user sets UEK1 and UEK2. The lock for UEK1 and user

security segments can be temporarily overridden in the event of a FlashLock (UPK1) passcode match. The lock for UEK2 can be temporarily overridden by UPK2.

4.2.2 Key Mode Locks (Ask a Question)

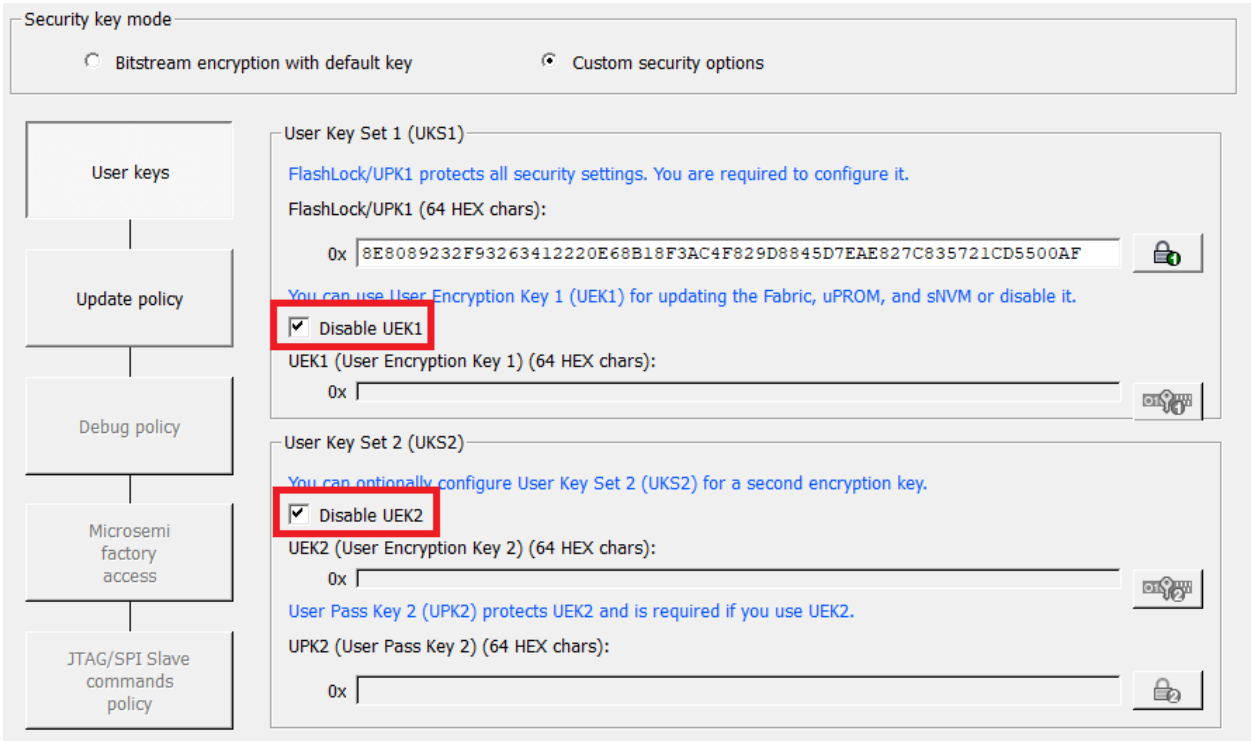
Key modes are used to select the root key and algorithm to encrypt and/or authenticate data in a device protocol, for example, the bitstream loading protocol. Not all key modes are applicable to every protocol, and some combinations are only supported if the optional secure production programming solution (SPPS) is used. Key modes can be disabled using lock bits.

In a new device, any one of the supported factory key modes may be used to load the initial user keys in encrypted form. After the user keys are loaded, all the factory key modes are automatically disabled, leaving only the user key modes in operation. Thus, any subsequent bitstream update must be done using the user keys.

Key modes associated with keys that are not loaded are also automatically disabled. It is required to match the FlashLock passcode to allow the key mode lock bits to be erased, after which they can be reprogrammed by a new bitstream.

In the **Configure Security** Wizard, click **User keys** and select **Disable UEK1** and **Disable UEK2** to disable UEK1 and/or UEK2 key mode, as shown in the following figure.

Figure 4-1. Disable Key Mode using Configure Security Wizard



4.2.3 Fabric and sNVM Update Protection Locks (Ask a Question)

These locks prevent FPGA and sNVM from being erased and written with a new bitstream. These locks can be temporarily unlocked by matching the FlashLock (UPK1) passcode through JTAG or SPI interface.

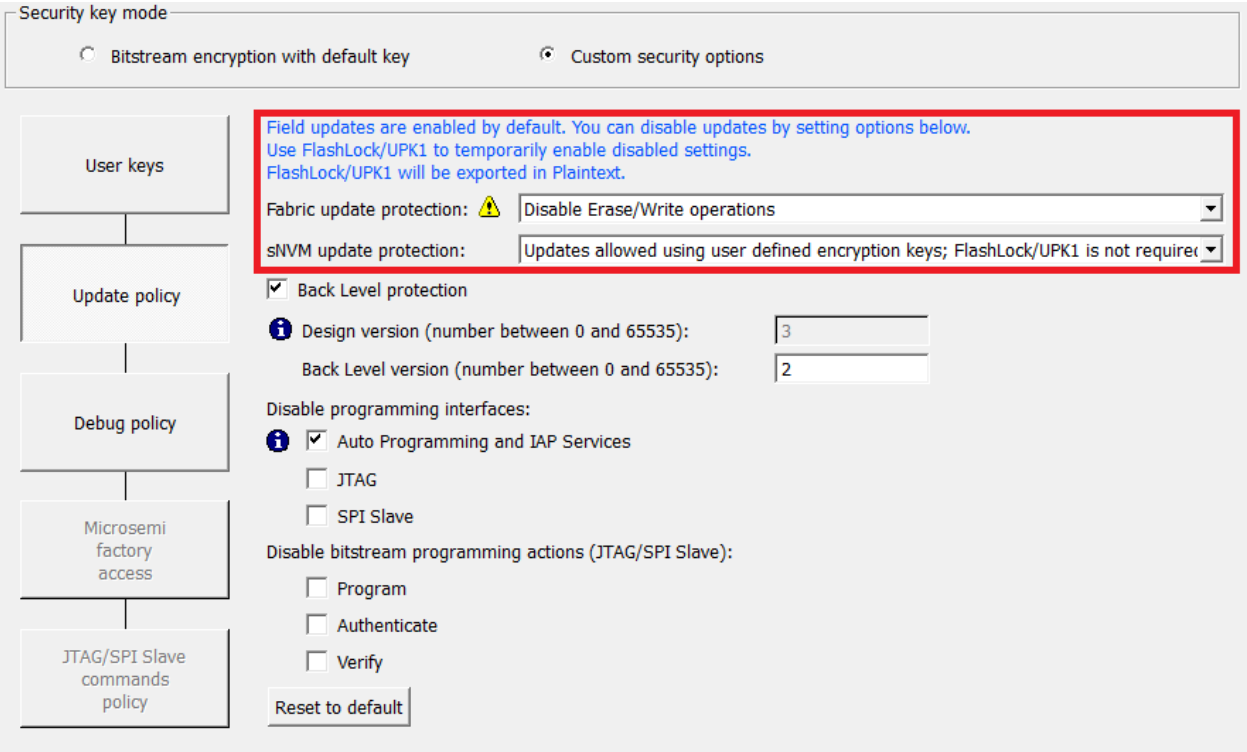
When the lock is cleared (either because it was never set, or was temporarily unlocked by a passcode match), and you have provisioned a user encryption key (UEK1 or UEK2), you need to use the bitstream encrypted with one of the user encryption keys to update either the fabric or the sNVM.

In the **Configure Security** Wizard, the **Update Policy** is used to apply the FPGA fabric and the sNVM update protection lock bits.

- To set the FPGA fabric update protection lock bit, select **Disable Erase/Write operations** under **Fabric update protection**.
- To set the sNVM update protection lock bit, select **Disable Write operations** under **sNVM update protection**.

Auto Programming, Auto Update, IAP Services, and Programming Recovery are disabled for update when the preceding lock bits are set. FlashLock/UPK1 unlocking is only available for JTAG and SPI Slave programming.

Figure 4-2. Update Policy Page in Configure Security Wizard



4.2.4 Programming Port Locks (Ask a Question)

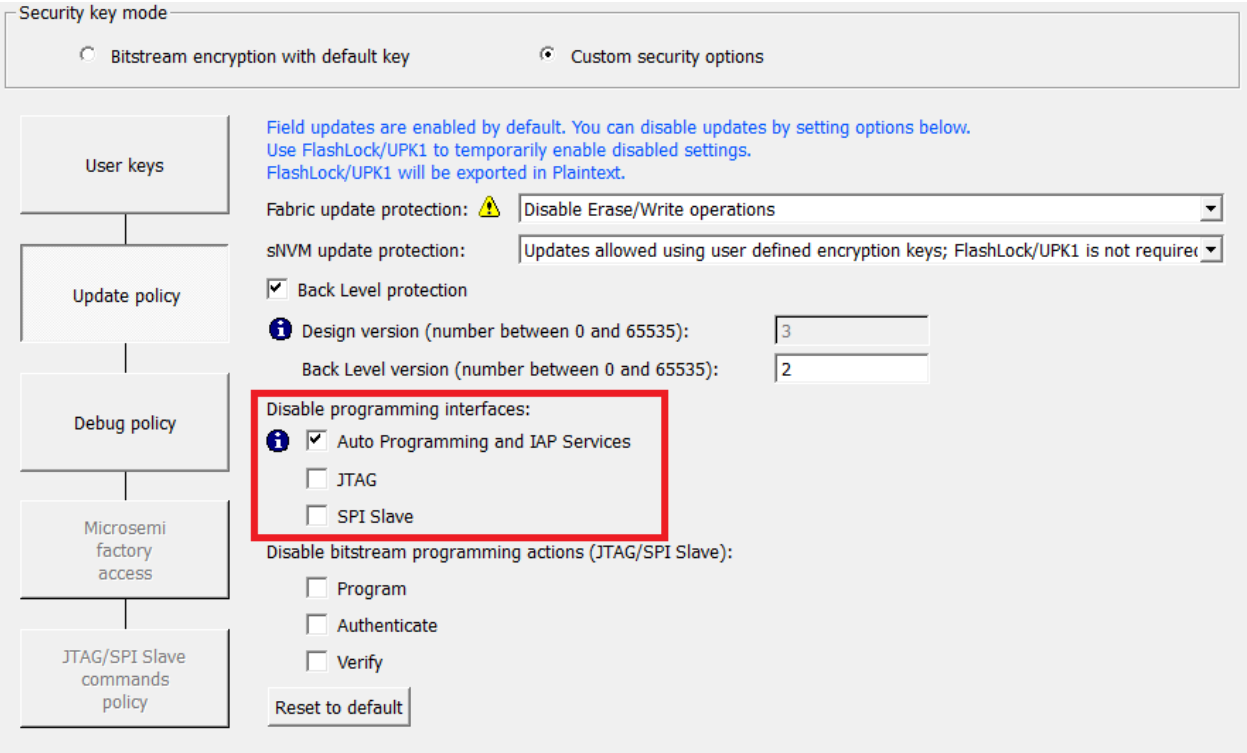
Several user lock bits are available to block access to programming through specific programming ports. These lock bits include:

Table 4-1. Locks to Disable Programming Interfaces

Lock Function when Active	Description
Disable Auto Programming and IAP Services	This lock disables Auto Programming, Auto Update, IAP Services, and Programming Recovery. SPI initialization functionality is not affected. FlashLock/UPK1 unlocking is only available for JTAG and SPI Slave interfaces.
Disable JTAG	This lock completely disables the JTAG interface. If the device is part of a serial JTAG chain, the chain is broken. The JTAG pins themselves remain active but the TAP controller does not respond to activity on these pins.
Disable SPI Slave	This lock completely disables the SPI slave interface. Any activity on the SPI pins is ignored.

In the **Configure Security** Wizard, click **Update policy** to apply the programming port locks, as shown in the following figure.

Figure 4-3. Protect Programming Interfaces in the Libero® SoC Software



4.2.5 Programming Action Protection Locks (JTAG/SPI Slave) (Ask a Question)

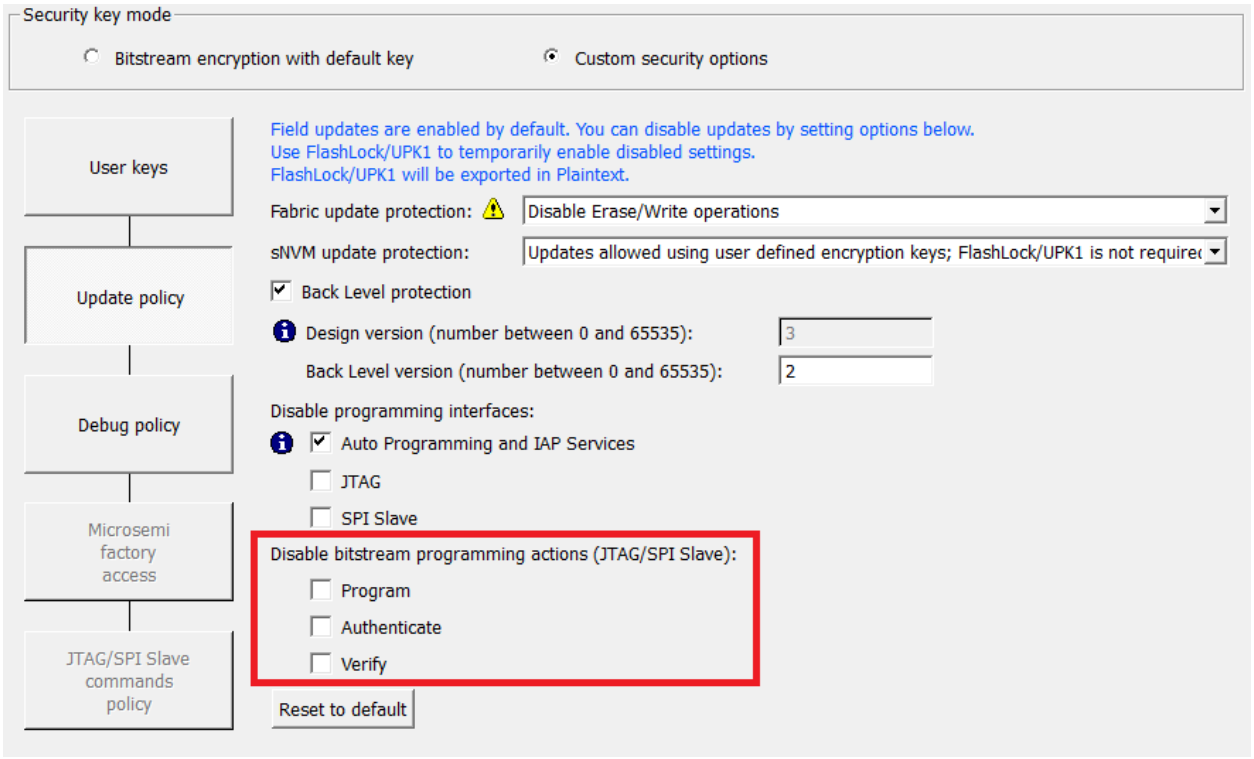
There are three user lock bits to block the various programming actions on the bitstream. These locks can be temporarily unlocked by matching the FlashLock (UPK1) passcode. Bitstream can be used in any of three programming actions—authenticate, program, and verify. Each programming action is protected by one of the following dedicated lock bits listed in the table:

Table 4-2. Locks to Disable Bitstream Programming Actions

Lock Function when Active	Description
Disable Bitstream Program	Program action is disabled for JTAG and SPI Slave interfaces. Auto Programming and IAP Services are not affected.
Disable Bitstream Authentication	Standalone authenticate action is disabled for JTAG and SPI Slave interfaces. Auto Programming and IAP Services are not affected. Note that this does not affect required authentication checks used in the program and verify programming actions.
Disable Bitstream Verify	Standalone verify action is disabled for JTAG and SPI Slave interfaces. Auto Programming and IAP Services are not affected. Note that this lock does not affect verification executed during programming operations.

In the **Configure Security** Wizard, click **Update policy** to apply the programming action protection locks, as shown in the following figure.

Figure 4-4. Locks to Protect Bitstream Programming Actions



4.2.6 User Debug Security Locks (Ask a Question)

The debugging features can be deactivated using the user debug lock bits. These lock bits enable the following:

Table 4-3. Locks to Control Debug Access

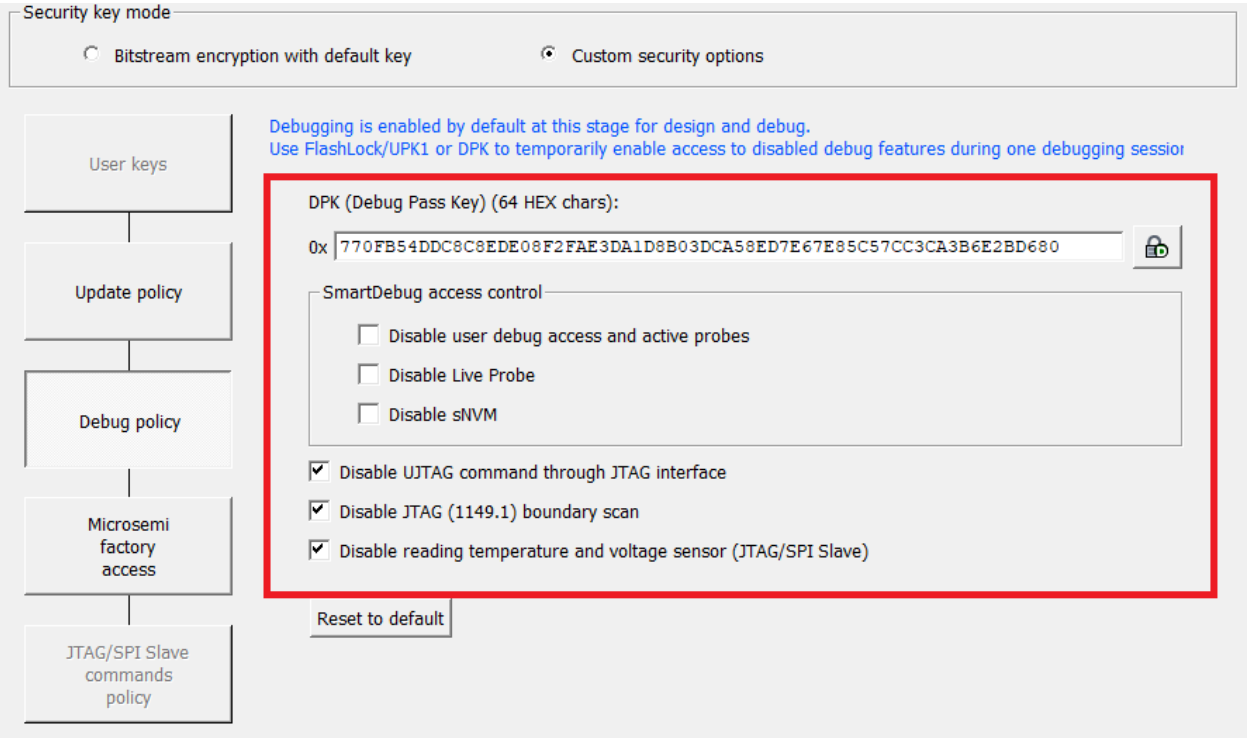
Lock Function when Active	Description
Disable user debug access and active probes	SmartDebug user debug access and active probes are disabled on both JTAG and SPI slave interfaces.
Disable live probes	SmartDebug Live probe debug access is disabled on both JTAG and SPI slave interfaces.
Disable sNVM	SmartDebug sNVM debug access is disabled on both JTAG and SPI slave interfaces.
Disable UJTAG command through JTAG interface	Disables the UJTAG interface to the FPGA fabric by asserting the URSTB fabric input, holding the associated user logic in reset. All other signals on the UJTAG interface continue to operate as normal allowing the interface to continue to be used for monitoring functions. Libero® catalog includes a UJTAG macro to access UJTAG interface.
Disable JTAG (1149.1) boundary scan	JTAG (1149.1) boundary scan is disabled. The following JTAG instructions will be disabled: EXTEST, INTEST, CLAMP, SAMPLE, and PRELOAD. I/Os will be tristated when in JTAG programming mode and BSR control during programming is disabled. BYPASS, IDCODE, and USERCODE instructions will remain functional.
Disable reading temperature and voltage sensor (JTAG/SPI Slave)	Reading of temperature and voltage sensor is disabled on both JTAG and SPI slave interfaces.

Use FlashLock/UPK1 or DPK to temporarily enable access to the disabled debug features during one debug session. It stays in effect only until the device is reset or power cycled. DPK unlocks just certain lock bits related to FPGA debugging, but does not unlock as many lock bits as the FlashLock

passcode does. For example, it does not allow the user to overwrite any keys, passcodes, or security settings.

In the **Configure Security** Wizard, click **Debug policy** to lock the debugging features, as shown in the following figure.

Figure 4-5. Debug Policy Page in Configure Security Wizard

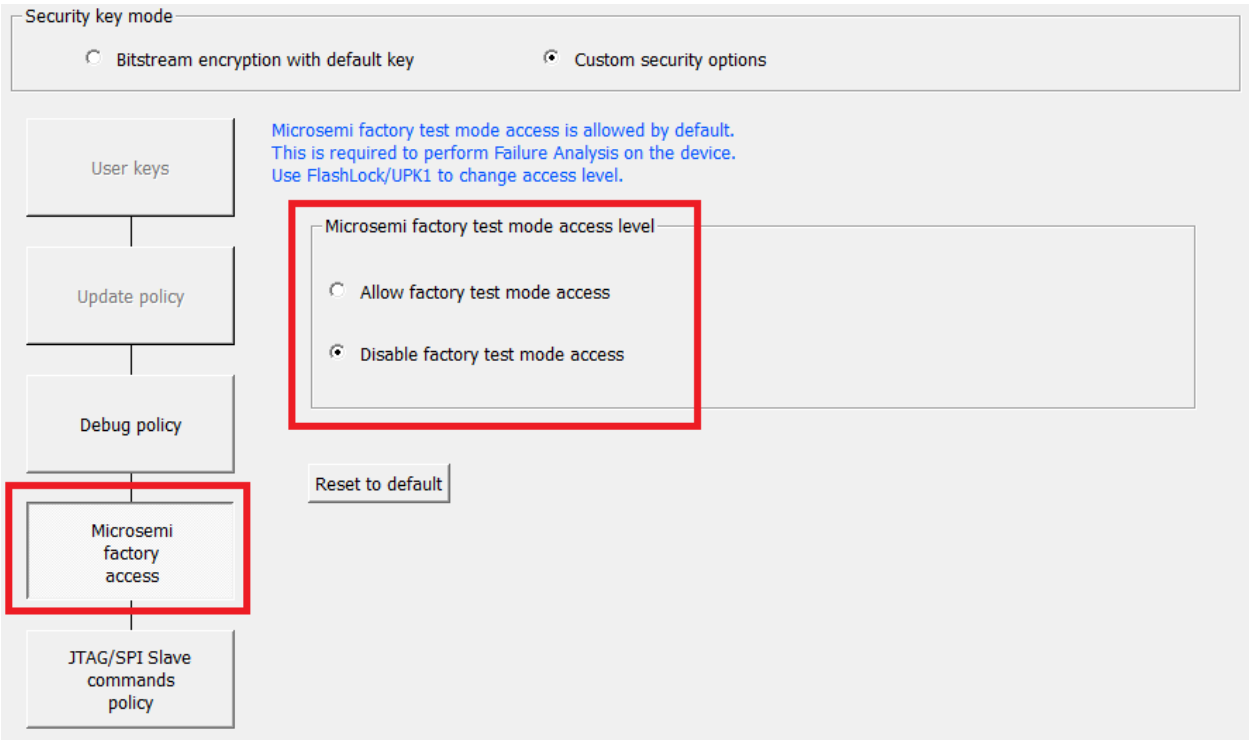


4.2.7 Factory Test Mode Access Lock (Ask a Question)

The factory test mode allows in-depth testing of the device by Microchip for failure analysis. The factory test mode access can be disabled using a lock bit protected by UPK1. If you wish Microchip to perform failure analysis on a given device, enable the factory test mode access with UPK1. The factory test mode access can also be permanently disabled by the user. When that is done, no future failure analysis is possible on that device.

In the **Configure Security** Wizard, click **Microsemi factory access** to lock the factory test mode access, as shown in the following figure. To permanently disable factory test mode access, use the **Configure OTP Security** tool.

Figure 4-6. Microsemi Factory Access Page in Configure Security Wizard



4.2.8 JTAG/SPI Slave Commands Locks (Ask a Question)

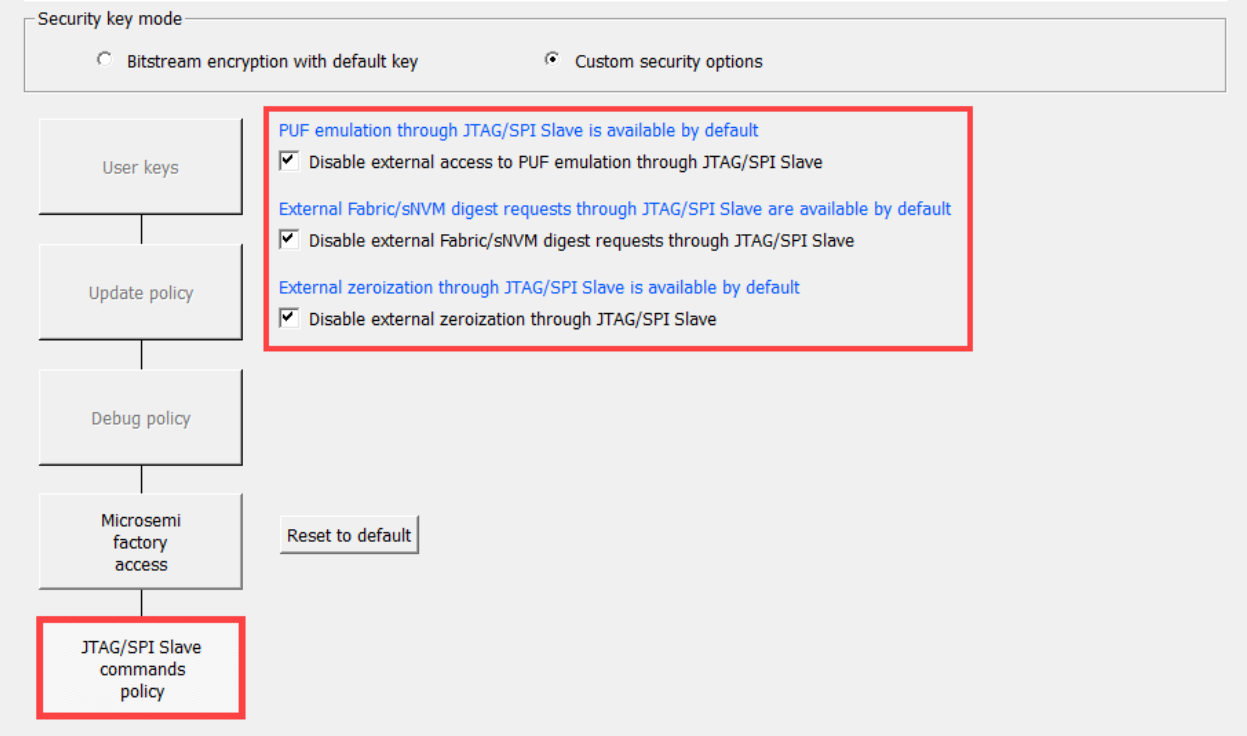
The JTAG/SPI slave command locks include the following:

Table 4-4. Locks to Disable JTAG/SPI Slave Commands

Lock Function when active	Description
Disable external access to PUF emulation	External access to PUF emulation through JTAG and SPI Slave interfaces are disabled.
Disable external fabric/sNVM/eNVM (for PolarFire® SoC FPGA only) digest requests	External Fabric/sNVM/eNVM (for PolarFire SoC FPGA only) digest check requests through JTAG and SPI Slave interfaces are disabled. Export of stored digests is not affected.
Disable external zeroization requests	External zeroization requests through JTAG/SPI Slave interfaces are disabled.

These locks can be temporarily overridden in the event of a FlashLock (UPK1) passcode match. Use the **Configure Security Wizard** and select the **JTAG/SPI Slave commands policy** page to apply the JTAG/SPI slave command policy lock bits, as shown in the following figure. These locks only apply when bitstreams are being loaded through the JTAG or SPI slave interfaces, and they do not apply to IAP functions.

Figure 4-7. JTAG/SPI Slave Commands Policy Page



4.2.9 Permanent Locks (Ask a Question)

The devices' ability to be reconfigured allows the FPGAs to be updated in the lab or in the field with encrypted and authenticated bitstreams They also have the capability to be one-time programmable to provide higher assurance that overwriting the design by unauthorized entities is impossible. This is beneficial for designs where single function ASICs are traditionally used, but the design and development flow requires the ability to be reprogrammed through development.

The devices offer the following permanent lock bits. You can enable any of the following locks permanently:

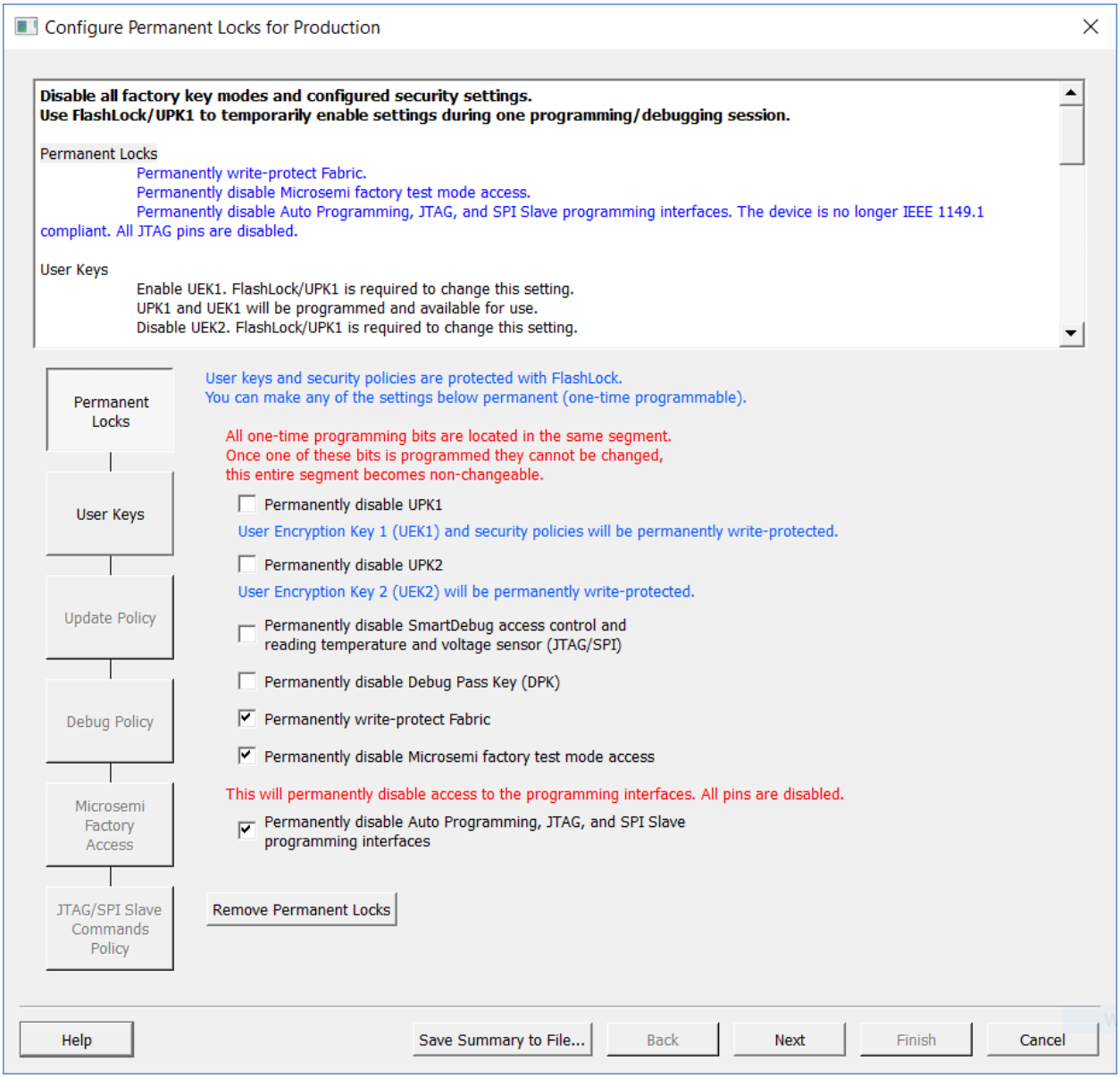
- **Permanently disable UPK1**—This will permanently disable FlashLock/UPK1 from being able to be matched by the device. Any feature that is disabled will be permanently disabled. Any feature that is available will be permanently available.
- **Permanently disable UPK2**—This will permanently disable UPK2 from being able to be matched by the device. If UEK2 is enabled and selected for programming, then it cannot be changed.
- **Permanently disable SmartDebug access and reading TVS**—This will permanently disable SmartDebug access for user debug along with the ability to read the temperature and voltage sensor (TVS).
- **Permanently disable Debug Pass Key (DPK)**—This will permanently disable the DPK from being able to be matched by the device. If DPK was programmed, then it can no longer be used for SmartDebug access.
- **Permanently write-protect Fabric**—This will make the Fabric One-Time Programmable. Verify of the Fabric will still be possible. Erase/Program of the Fabric is permanently disabled.
- **Permanently disable Microsemi factory test mode access**—This will permanently disable Microsemi factory test mode access. Microchip will not be able to perform a Failure Analysis on this device.
- **Permanently disable Auto Programming, JTAG and SPI Slave programming interfaces**—This will permanently disable all programming interfaces. The actual JTAG and SPI Slave ports are

disabled and you cannot access the device for any operations including reading the IDCODE of the device. The device will become a One-Time Programmable and there will be no way to Erase/Program/Verify the device.

The permanent locks cannot be unlocked by passcodes. All the permanent lock bits are located in the same segment. The permanent locks segment can be written only once and is immune to zeroization operations. Once one or more of the permanent lock bits are programmed then they cannot be changed and the entire segment becomes unchangeable.

The Configure Permanent Locks for Production in the Libero Flow allows to configure Permanent Locks for Production programming. Permanent Locks must be configured after the Design/Debug phase is completed. The Permanent Lock settings are not applied when the device programming is done using **Program Design > Run PROGRAM** Action. They are only applied to the Export tools used for Production programming. Once the Permanent Locks are programmed, they cannot be changed. Configuring the Permanent Locks affect the settings on the subsequent pages and should be reviewed carefully. The settings cannot be changed once they are programmed.

Figure 4-8. Configure Permanent Locks for Production



5. Supply Chain Assurance (Ask a Question)

5.1 Supply Chain Assurance Certificate (Ask a Question)

Counterfeiting in electronic parts takes various forms, including black-topping and re-marking the devices to misrepresent the used devices as new, change the date codes, improve the speed grade or temperature grade. To prevent counterfeiting, the PolarFire family incorporates an X.509-compliant device unique supply chain assurance certificate.

The supply chain assurance certificate cryptographically binds the DSN, date code, enabled features, and a public key with a digital signature in a way that can be validated internally by the device and externally by the user. Any mismatch between how the device is represented by its shipping paperwork, the label printed on its surface and the supply chain assurance certificate indicates the possibility of counterfeiting fraud.

The supply chain assurance certificate can be fetched by running device certificate system service. For more information about system services usage, see PolarFire FPGA and PolarFire SoC FPGA System Services User Guide. When the supply chain assurance certificate is exported, the DSN and public key are checked for consistency against the actual values encoded in the device. Internally regenerating the public key from the private key adds an additional layer of protection against cloning, since the encrypted value of the private key and its authentication tag depend on the SRAM-PUF, thus deeply binding the private key to that particular device. The signature on the certificate is also checked using the immutable trusted Microsemi public key (MCPK) stored in the device.

5.2 Anti-Cloning Protection (Ask a Question)

The supply chain assurance certificate provides protection against re-marking of devices. An actual clone, however, would not be detected since the certificate, which is public information, could be copied from a genuine device onto the clone. To make cloning more difficult, the user requires the device to provide proof-of-possession (PoP) of the private key of the ECC key pair, which is certified by the supply chain assurance certificate. This is done either by employing a challenge-response protocol or by having the device digitally sign a nonce and then verifying the signature using the public key. This would then require the clone to have knowledge of the device's private key, which is protected by SRAM-PUF encryption and stored in pNVM. A digital copy of the public certificate alone would no longer be sufficient to prove the device's identity. This protocol thus improves confidence in the authenticity of a device.

5.3 Device Integrity Protection (Ask a Question)

It should be possible to distinguish new devices from a previously used or tampered device. The used device has obvious implications for device quality and endurance. Attempts may be made to extract the device's unique factory keys with the intention of later intercepting or forging communications with the device. To mitigate this class of attacks, devices employ a mechanism to mark used devices. This requires reading the device integrity bits using JTAG/SPI instruction. It returns the signed certificate with device integrity bits and device serial number. The returned device integrity bits can be matched with an expected value for a new device. The data integrity bits are initialized to the following 256-bit big-endian value at the factory:

4BE48DC078655D410FCDCE9BF440E55E2FAB9525A27EB8F1E4B1DB5C9D0CAFF6

When you receive a new device, examine the device integrity bits and check that they are still intact. The device integrity bits are invalidated in the following events:

- Loading a bitstream in Program mode
- Attempted execution of a user passcode protocol
- Failed execution of the factory passcode protocol

Device integrity bits cannot be modified by any user operation. Zeroization changes the state of the device integrity bits, but cannot restore them to their pristine state. Device integrity bits can be read from the device using Libero SoC or FlashPro Express by running Device Info programming action.

It is also possible to check the validity of the device integrity bits before commencing the device programming. If the device integrity bits are invalid, the programming action fails and the error is logged in FRAME_ERRORCODE as 14. This feature will be supported in Secure Production Programming Solution (SPPS). This debug information can be retrieved from the device using Read Debug Information Service. For more information, see PolarFire FPGA and PolarFire SoC FPGA System Services User Guide.

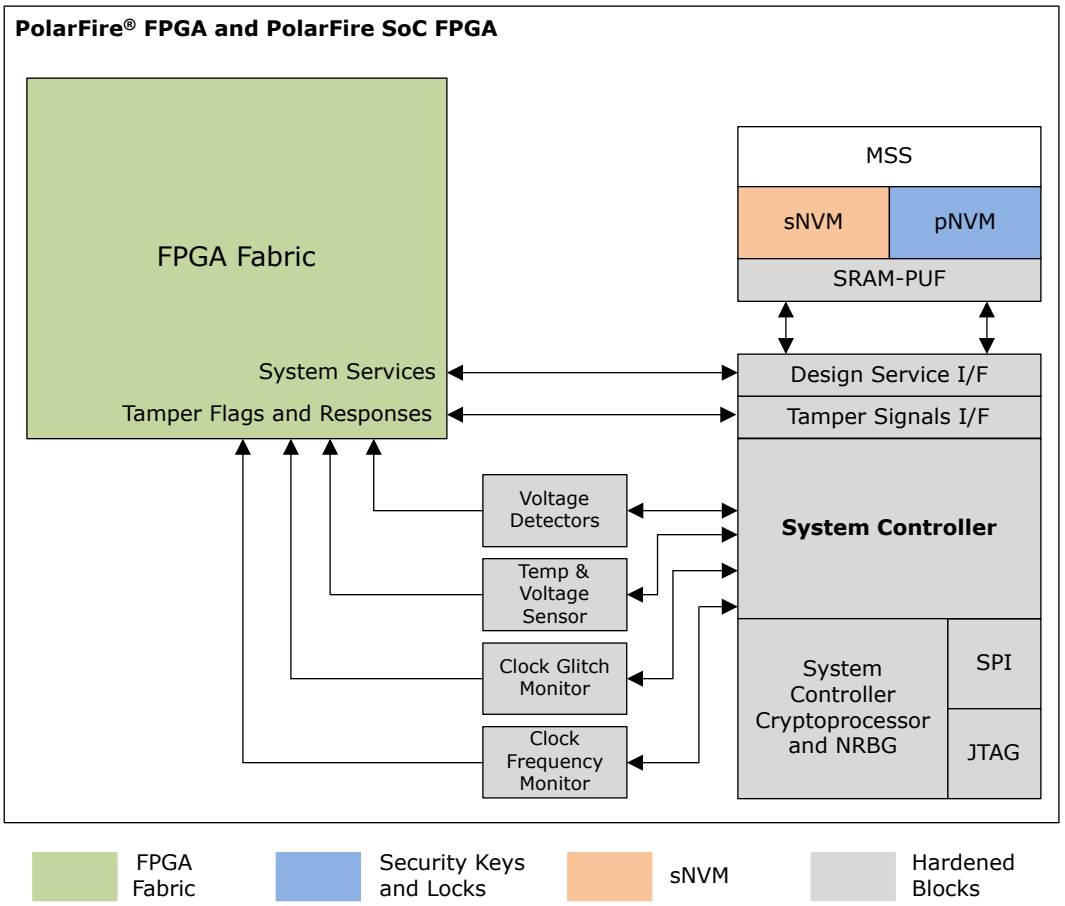
6. Device-Level Anti-Tamper Features (Ask a Question)

The PolarFire family includes a number of built-in tamper detection and response capabilities that can be used to enhance the security of the device. These countermeasures are intended to address various types of attacks that include non-invasive, semi-invasive, and invasive attacks. The devices can detect a number of conditions that may indicate an attempt to tamper.

When a tamper condition is detected, a notification is sent to the fabric via one of many dedicated lines. On receiving a tamper event, the fabric design may either choose to ignore the event or take defensive action using built-in tamper responses. Tamper events can only be cleared by assertion of the associated fabric clear signal or a device reset.

In addition to tamper detection and tamper response, all built-in design security protocols have protection against DPA and related side-channel monitoring attacks. All cryptographic algorithms implemented in a device are DPA-resistant. The PolarFire family have integrity check mechanisms that can optionally be used to check the reliability and security of a device upon power-up or on-demand.

Figure 6-1. Tamper Detection and Response Interfaces to the FPGA Fabric



Note: User Cryptoprocessor is part of MSS in PolarFire SoC FPGAs and there is no MSS in PolarFire FPGAs and RT PolarFire FPGAs. User Cryptoprocessor is a standalone block in PolarFire FPGAs and RT PolarFire FPGAs.

6.1 JTAG Security Monitor (Ask a Question)

The UJTAG_SEC macro, available in the Libero Catalog, allows a user-defined security monitor implemented in the FPGA fabric to observe all the JTAG signal activity. It also enables control of the System Controller TDI and the TRSTB inputs.

Figure 6-2. UJTAG_SEC

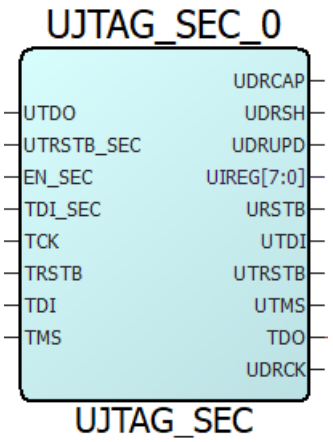


Table 6-1. UJTAG_SEC Ports and Descriptions

Port	Direction	Polarity	Description
UIREG[7:0]	Output	—	This 8-bit bus carries the contents of the JTAG instruction register of each device. The instruction values, from 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 the Test-Logic-Reset mode. URSTB is asserted at power-up, and a power-on reset signal resets the TAP controller state.
UTDI	Output	—	Directly connected to the TAP's TDI signal.
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	—	Directly connected to the TAP's TCK signal. **Note:** UDRCK must be connected to a global macro such as CLKINT. If this is not done, Synthesis/Compile will add it to the netlist to legalize it.
UDRUPD	Output	High	Active-high signal enabled in the Update_DR_TAP state.
TDO	Output	—	Test Data Out. Serial output for JTAG boundary scan. The TDO pin does not have an internal pull-up/pull-down resistor.
TDO	Output	—	Test Data Out. Serial output for JTAG boundary scan. The TDO pin does not have an internal pull-up/pull-down resistor.
UTDO	Input	—	User TDO output. Inputs to the UTDO port are sent to the TAP TDO output MUX when the IR address is in user range.
TCK	Input	—	Test Clock. Serial input for JTAG boundary scan, ISP, and UJTAG. The TCK pin does not have an internal pull-up/pull- down resistor. Connect TCK to GND or +3.3V through a resistor (500 - 1 KΩ) placed closed to the FPGA pin to prevent totem-pole current on the input buffer and TMS from entering into an undesired state. If JTAG is not used, connect it to GND.
TDI	Input	—	Test Data In. Serial input for JTAG boundary scan. There is an internal weak pull-up resistor on the TDI pin.
TMS	Input	—	Test mode select. The TMS pin controls the use of the IEEE® 1149.1 boundary scan pins (TCK, TDI, TDO, and TRST). There is an internal weak pull-up resistor on the TMS pin.

Table 6-1. UJTAG_SEC Ports and Descriptions (continued)

Port	Direction	Polarity	Description
TRSTB	Input	Low	Test reset. The TRSTB pin is an active low input. It synchronously initializes (or resets) the boundary scan circuitry. There is an internal weak pull-up resistor on the TRSTB pin. To hold the JTAG in reset mode and prevent it from entering into undesired states in critical applications, connect TRSTB to GND through a 1 KΩ resistor (placed close to the FPGA pin).
EN_SEC	Input	High	Enable Security. Enables the user design to override the external TDI and TRSTB input to the TAP. Tie LOW in the design when not used.
TDI_SEC	Input	—	TDI Security override. Overrides the external TDI input to the TAP when SEC_EN is HIGH.
TRSTB_SEC	Input	Low	TRSTB Security override. Overrides the external TRSTB input to the TAP when SEC_EN is HIGH.

Note: When the JTAG Security Monitor interface is enabled, the maximum TCK frequency for the device may be limited by delays through the user logic. TCK frequency is not monitored by the device as no security sensitive logic operates in this clock domain. When the FPGA fabric is powered down, the Security Monitor interface is disabled.

6.2 User Voltage Detectors (Ask a Question)

Each device supply rail—VDD (1.0V), VDD18 (1.8V), and VDD25 (2.5V)—is equipped with a voltage detector, which triggers an alarm when the supply voltage falls below a minimum level and/or raises above a maximum level. For proper operation of the VDD voltage detector, VDD must be set to 1.0V. The alarm condition is passed directly to the fabric. The alarm condition is cleared by asserting the event clear signal from the fabric. The threshold levels on the detectors are set in the factory. For more information about voltage detectors threshold values, see respective PolarFire FPGA Datasheet or PolarFire SoC Datasheet. Use Tamper macro to access the voltage detector inputs and outputs. See Figure 6-10.

6.3 Temperature and Voltage Sensor (Ask a Question)

Each device is equipped with a Temperature and Voltage Sensor (TVS). It reports die temperature and voltages of—VDD (1.0V), VDD18 (1.8V), and VDD25 (2.5V)—device supply rails in digital form to the FPGA fabric. TVS is implemented using a 4-channel ADC and the channel information is given as follows:

- Channel 0 - VDD (1V) voltage supply
- Channel 1 - VDD18 (1.8V) voltage supply
- Channel 2 - VDD25 (2.5V) voltage supply
- Channel 3 - die temperature

The TVS outputs a 16-bit encoded value that represents voltage or temperature, and corresponding channel number. The temperature and voltage information is translated into standard temperature and voltage values. The temperature channel output is also given to a comparator residing within the system controller that raises an alarm if the temperature is not between the maximum and minimum threshold levels specified by the user.

The voltage channel's 16-bit output value is represented in millivolts (mV) and can be decoded as listed in the following table. For example, the voltage channel's output value of 0x385E implies 1803.75 mV.

Table 6-2. Voltage Channel Value Decoding

Bit Number	Description
15	Signed bit
[14:3]	Integer value of voltage

Table 6-2. Voltage Channel Value Decoding (continued)	
Bit Number	**Description**
[2:0]	Fractional value of voltage

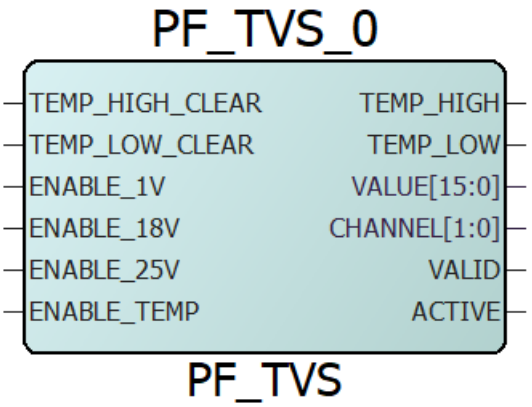
The temperature channel's 16-bit output value is represented in Kelvin and can be decoded as listed in the following table. For example, the temperature channel's output value of 0x133B implies 307.6875 Kelvin.

Table 6-3. Temperature Channel Value Decoding

Bit Number	Description
15	Reserved
[14:4]	Integer value of temperature
[3:0]	Fractional value of temperature

The TVS is accessible by instantiating PF_TVS macro in the design, see the following figure.

Figure 6-3. PF_TVS Macro



The following table lists the PF_TVS macro ports and their description.

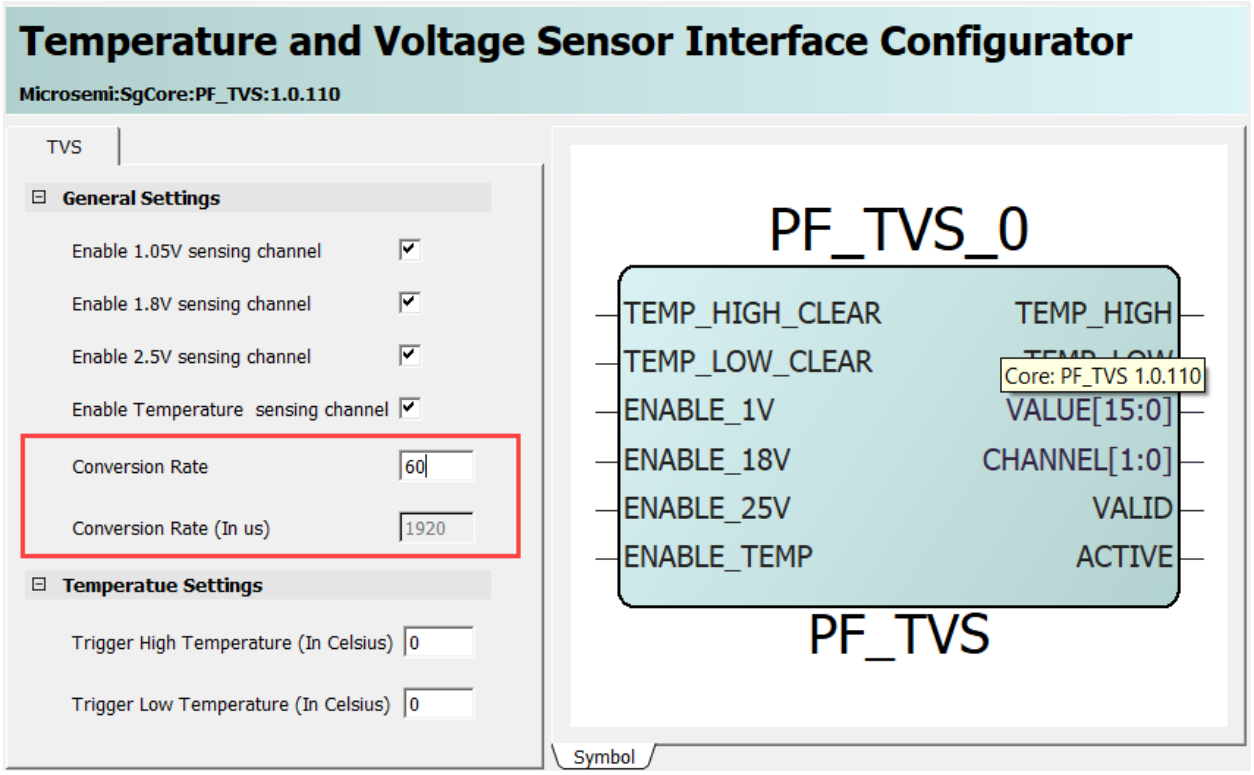
Table 6-4. PF_TVS Macro and Port Description

Port Name	Direction	Description
ENABLE_1V	Input	VDD (1V) channel enable signal
ENABLE_18V	Input	VDD18 (1.8V) channel enable signal
ENABLE_25V	Input	VDD25 (2.5V) channel enable signal
ENABLE_TEMP	Input	Temperature channel enable signal
TEMP_HIGH_CLEAR	Input	Control input to clear TEMP_HIGH flag
TEMP_LOW_CLEAR	Input	Control input to clear TEMP_LOW flag
VALID	Output	Asserted after channel/value changes
CHANNEL[1:0]	Output	Indicates available channel data on VALUE[15:0], held until the next conversion completes
VALUE[15:0]	Output	Channel data, held until the next conversion completes
TEMP_HIGH	Output	Set when temperature is above the high level threshold
TEMP_LOW	Output	Set when temperature is below the low level threshold
ACTIVE	Output	Indicates that the TVS is active

The data present on the VALUE and CHANNEL outputs is valid only when the VALID output is asserted. When a channel is disabled by deasserting the corresponding channel enable input, then the channel data present on the outputs is not valid even if the VALID output is asserted.

The following figure shows the TVS configurator. In General Settings, users has the option to choose channels for conversion and conversion rate options. In Temperature Settings, user can specify the threshold values for Trigger High Temperature and Trigger Low Temperature alarm flags generation. The temperature ranges from -55 °C to 125 °C. This is enabled only when the temperature channel sensing is enabled. The temperature high and low alarm flags (TEMP_HIGH and TEMP_LOW) can only be cleared by the user once the measured temperature returns to normal.

Figure 6-4. TVS Configurator

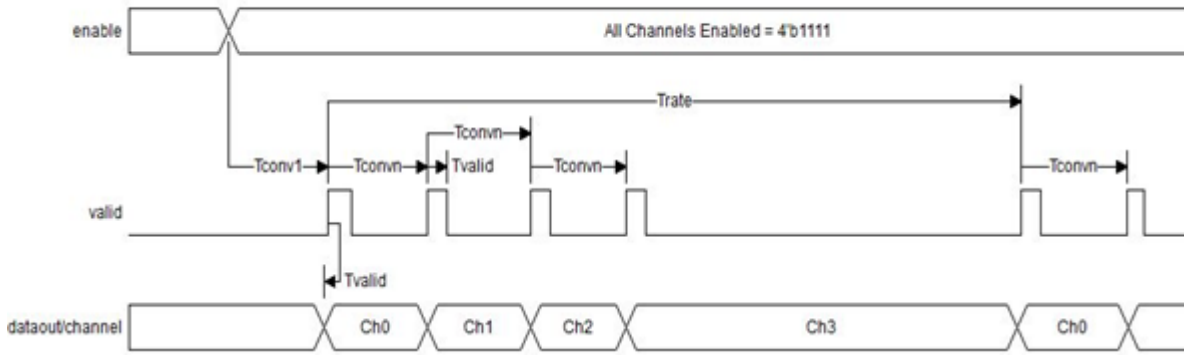


The following figure shows the TVS conversion sequence when all the channels are enabled. The time between subsequent channel conversions is called conversion delay (Tconvn). The time between the start of the first set of conversions to the start of the next set is called conversion rate (Trate).

- Conversion rate = conversion rate parameter × 32 µs.
- Conversion rate parameter ≥ Number of channels enabled × 15

For example, if one channel is enabled then the minimum conversion rate parameter is equal to 15, which means the conversion rate is 480 µs. If all four channels are enabled then the minimum conversion rate parameter is equal to 60, which mean the conversion rate is 1920 µs.

Figure 6-5. TVS Conversion Sequence



See respective PolarFire FPGA Datasheet or PolarFire SoC Datasheet for accuracy and electrical characteristics of TVS.

Important: To receive correct output data values from TVS, it is necessary to synchronize the TVS output data to the user clock domains and then sample it, before decoding it. To achieve this, synchronize the PF_TVS IP "VALID" output signal to the incoming clock domain and then sample the output data "VALUE[15:0]" and "CHANNEL[1:0]" at the active edge of the "VALID" signal.

6.4 Clock Glitch Monitor (Ask a Question)

The system controller includes a clock glitch tamper detector that detects when the master clock input to the system controller is tampered in such a way to cause rapid change in frequency.

The monitor continuously monitors the clock period. Should if the period be significantly shorter or longer than the previous period, an error is reported. Small changes in period are ignored as the circuit delays vary due to PVT changes.

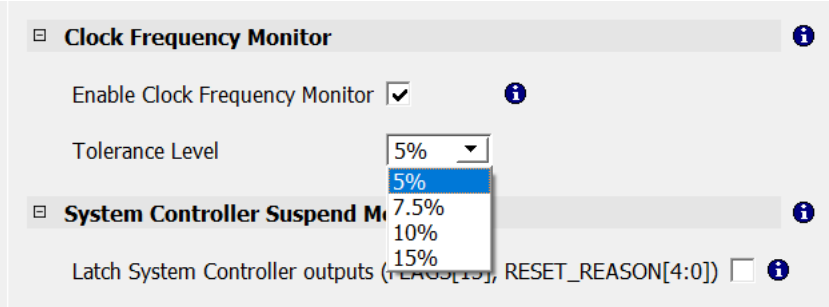
When an error occurs, the error is reported to the fabric through the tamper macro.

6.5 Clock Frequency Monitor (Ask a Question)

This monitor verifies that there is an 80:1 relationship between the 160 MHz and 2 MHz oscillator clocks. An error is reported to the fabric through the tamper macro when a frequency mismatch is detected between the two oscillators.

The following figure shows the **Clock Frequency Monitor** feature in the **Tamper Configurator**. For the full list of features in the Tamper Configurator, see Figure 6-9 .

Figure 6-6. Clock Frequency Monitor



6.6 Anti-tamper Mesh (Ask a Question)

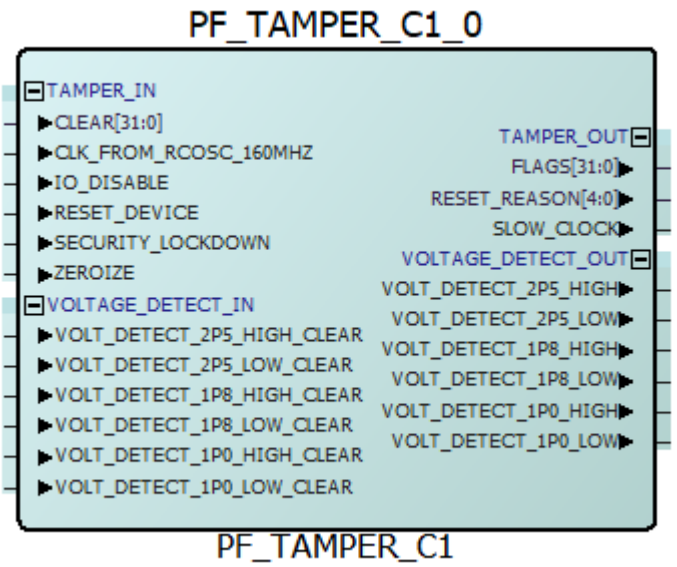
The system controller implementation includes an anti-tamper mesh. This covers the complete system controller and user cryptoprocessor silicon area with an active mesh that detects if wires are cut or shorted. To obtain access to the active circuitry under the mesh, a “hacker” would need to break the mesh; this will trigger a tamper alarm to the system controller and the fabric to take appropriate action.

6.7 Tamper Detection and Tamper Responses (Ask a Question)

A tamper macro is provided in the Libero software catalog to access tamper flags and response inputs from fabric. There are two sets of input/output ports in a tamper macro. One set corresponds to the tamper detection flags and the tamper responses. Other set corresponds to the user voltage detectors, which triggers an alarm when the supply voltage falls below a minimum level or raises above a maximum level.

The following figure shows the PF_TAMPER macro and its configurable options for the PolarFire FPGA.

Figure 6-7. PolarFire® FPGA Tamper Macro



The following figure shows the PFSOC_TAMPER macro and its configurable options for the PolarFire SoC FPGA.

Figure 6-8. PolarFire® SoC FPGA TAMPER Macro

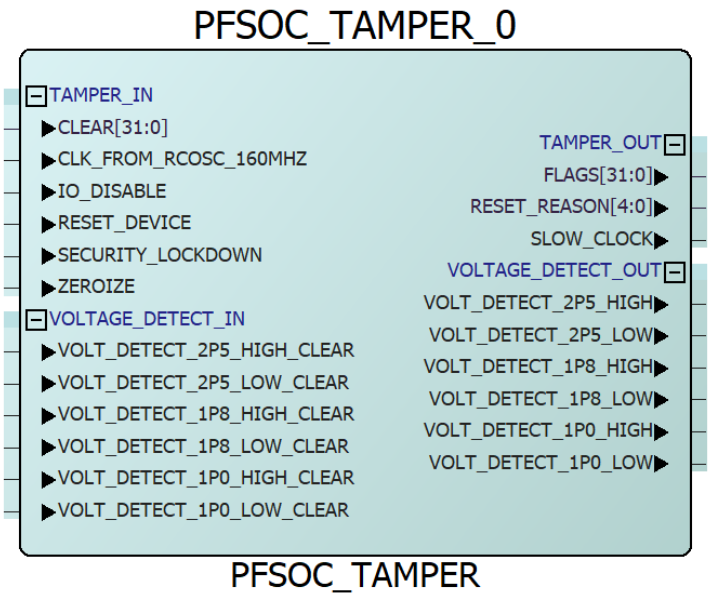
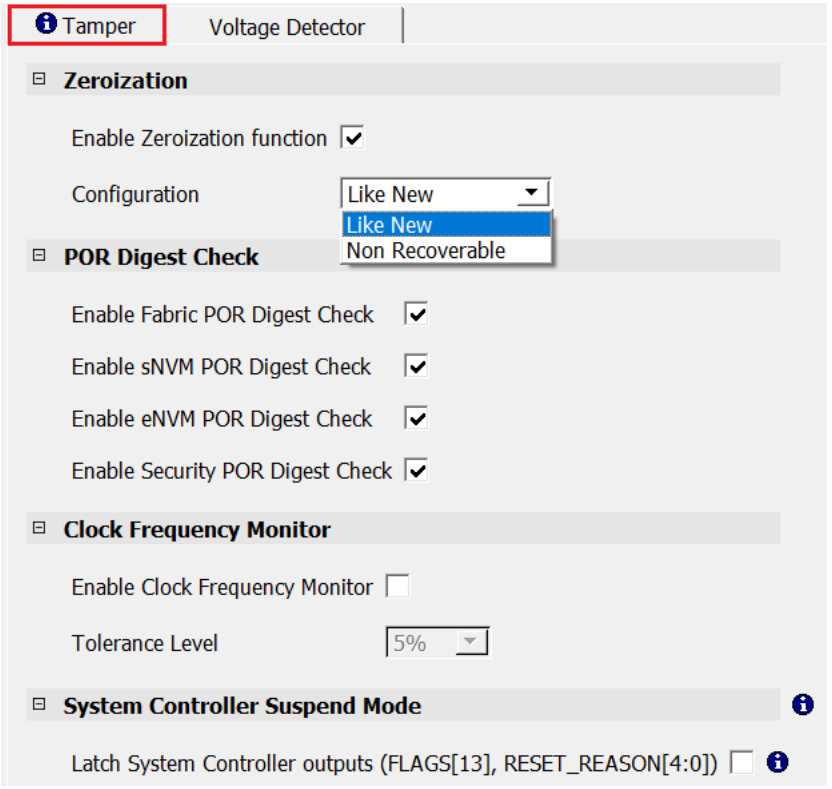
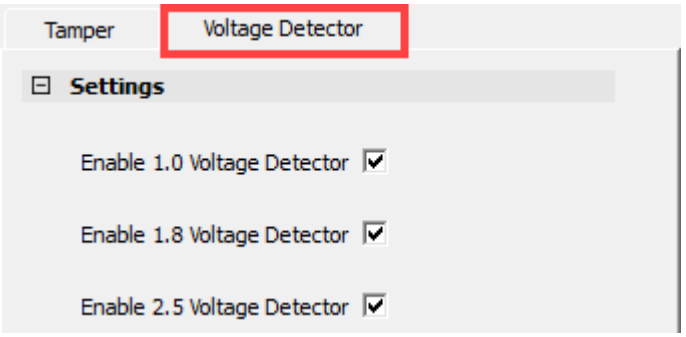


Figure 6-9. Tamper Configurator—Tamper



Important: Enable eNVM POR Digest Check is only applicable for the PolarFire SoC FPGA.

Figure 6-10. Tamper Configurator—Voltage Detector



The following table lists the tamper macro ports.

Table 6-5. Tamper Macro Port Description

Port	Direction	Description
CLEAR[31:0]	Input	Assertion of these signals clear the associated tamper flags (FLAGS[31:0]).
IO_DISABLE	Input	Disables all the user I/O (or non-dedicated) pins when this signal is asserted. All the output buffers are tri-stated (any configured pull-up or pull-down is still honored) and the input buffers are disabled.
SECURITY_LOCKDOWN	Input	When asserted, it forces all the access control locks active and clears all the security unlocks that might be set. It puts the device in lockdown state.
RESET_DEVICE	Input	This is equivalent to asserting the device reset pin. The user design is immediately powered down and the device re-executes its initialization sequence. This pin should not be connected directly to I/O pad, the PLL lock output or the designs system reset as it puts the system controller in reset. See PolarFire FPGA: How to Perform On-Demand Digest Check Application Note for usage of this port.
ZEROIZE	Input	Assertion of this signal initiates zeroization process. See Zeroization.
VOLT_DETECT_2P5_HIGH_CLEAR	Input	Assertion of this signal clears the VOLT_DETECT_2P5_HIGH tamper flag.
VOLT_DETECT_2P5_LOW_CLEAR	Input	Assertion of this signal clears the VOLT_DETECT_2P5_LOW tamper flag.
VOLT_DETECT_1P8_HIGH_CLEAR	Input	Assertion of this signal clears the VOLT_DETECT_1P8_HIGH tamper flag.
VOLT_DETECT_1P8_LOW_CLEAR	Input	Assertion of this signal clears the VOLT_DETECT_1P8_LOW tamper flag.
VOLT_DETECT_1P0_HIGH_CLEAR	Input	Assertion of this signal clears the VOLT_DETECT_1P0_HIGH tamper flag.
VOLT_DETECT_1P0_LOW_CLEAR	Input	Assertion of this signal clears the VOLT_DETECT_1P0_LOW tamper flag.
CLK_FROM_RCOSC_160MHz	Input	Clock for latches when **Latch System Controller outputs** option is enabled. Must be connected to internal 160 MHz RC oscillator.
FLAGS[31:0]	Output	Tamper detection flags to the fabric. These are cleared by asserting CLEAR[31:0] signals or the device reset. See Tamper Detection Flags.
RESET_REASON[4:0]	Output	Indicates the source of last system reset, see Reset Reason.

Table 6-5. Tamper Macro Port Description (continued)

Port	Direction	Description
SLOW_CLOCK	Output	The flag is asserted when the system controller clock is slowed down to 20 MHz. When the VDD brownout is detected, the system controller's clock automatically reduces from 80 MHz to 20 MHz. There is no automatic shutdown of the device due to detection of brownout in operational mode, but an alarm signal (slow_clock) is asserted to the fabric so that the user may take action if desired. User may observe few pulses on this signal before it settles to high. The system controller's clock switches to 80 MHz if voltage recovers without going below the reset threshold.
VOLT_DETECT_2P5_HIGH	Output	When asserted, it indicates that the VDD25 (2.5V) supply is above the maximum threshold voltage level.
VOLT_DETECT_2P5_LOW	Output	When asserted, it indicates that the VDD25 (2.5V) supply is above the minimum threshold voltage level. In other words, when it indicates low, the supply is below the minimum threshold voltage level.
VOLT_DETECT_1P8_HIGH	Output	When asserted, it indicates that the VDD18 (1.8V) supply is above the maximum threshold voltage level.
VOLT_DETECT_1P8_LOW	Output	When asserted, it indicates that the VDD18 (1.8V) supply is above the minimum threshold voltage level. In other words, when it indicates low, the supply is below the minimum threshold voltage level.
VOLT_DETECT_1P0_HIGH	Output	When asserted, it indicates that the VDD (1.0V) supply is above the maximum threshold voltage level.
VOLT_DETECT_1P0_LOW	Output	When asserted, it indicates that the VDD (1.0V) supply is above the minimum threshold voltage level. In other words, when it indicates low, the supply is below the minimum threshold voltage level.

6.7.1 Reset Reason (Ask a Question)

RESET_REASON[4:0] is a 5-bit field compressed from a 14-bit reset reason register, see Table 6-6 for bit definitions. This field is intended to provide the FPGA fabric a general cause of device reset. To determine the exact reset reason, the user must read the RESET_REASON[13:0] register using the Read Debug System Service. For information about this register and the service, see PolarFire FPGA and PolarFire SoC FPGA System Services User Guide. When the System Controller Suspend Mode feature is enabled, RESET_REASON[4:0] is updated only at FPGA initialization. To prevent loss of the reset reason when using this feature, these outputs should be latched by enabling the **Latch System Controller outputs** option in the Tamper macro.

Table 6-6. Reset Reason

Reset Reason	Description
RESET_REASON[4:0]	This field indicates to the user the reason for the most recent reset of the System Controller. The bits are allocated as follows: [0]: Device is reset through the DEVRSTN pin [1]: Device reset through tamper response input [2]: System Controller's Watchdog had triggered the reset [3]: Reset due to security locks system detected a security issue [4]: Any other reset

6.7.2 Tamper Detection Flags (Ask a Question)

Tamper-detection flags (FLAGS[31:0]) inform the user about tampering activity. Each tamper event is signaled over a dedicated wire to the fabric. On receiving the tamper detection flags, the user can choose to use the appropriate tamper response (see Tamper Response) or ignore/clear the flags. Tamper events can only be cleared by asserting the associated fabric clear signal or a system reset.

In the event of a fatal tamper event, the user design is powered down and an automatic POR is executed. The shutdown sequence guarantees a minimum of 10 µs after the tamper alarm fires before the shutdown begins, allowing the user design to perform internal clean up tasks. The system services may not be used during this time.

The following table lists and describes the tamper detection flags.

Important: When System Controller Suspend Mode is enabled, only FLAG[13] is available and other FLAGS [31:14] and [12:0] are not available.

Table 6-7. Tamper Detection Flags

Flags[31:0]	Flag	Description
0	JTAG_ACTIVE	This flag is asserted whenever the JTAG port is active, that is, the JTAG TAP controller enters the Run-Test-Idle state.
1	MESH_ERROR	This flag is asserted whenever the active security mesh observes a mismatch between the actual metal mesh output and the expected output. This allows protection against invasive attacks, such as cutting and probing of traces using focused ion beam (FIB) technology with an active metal mesh on one of the higher metal layers.
2	CLOCK_MONITOR_GLITCH	This flag is asserted whenever the clock glitch monitor detects a pulse width violation.
3	CLOCK_MONITOR_FREQUENCY	This flag is asserted whenever the clock frequency monitor observes a frequency mismatch between the 160 MHz and 2 MHz RC oscillators.
4	LOW_1P05	This flag is asserted when the 1.05V supply (VDD) is below the low threshold of the System Controller 1.05V detector. The tamper event is continuously generated until the supply returns to a level above the low threshold. This condition is also used during device programming to initiate shutdown procedures to protect the device programming circuits and integrity of the device NVM.
5	HIGH_1P8	This flag is asserted when the 1.8V supply (VDD18) is above the high threshold of the System Controller 1.8V detector. The tamper event is continuously generated until the supply returns to a level below the high threshold.
6	HIGH_2P5	This flag is asserted when the 2.5V supply (VDD25) is above the high threshold of the System Controller 2.5V detector. The tamper event is continuously generated until the supply returns to a level below the high threshold.
7	Reserved	Reserved
8	SECDED	This flag is asserted when a 2-bit error occurs in the System Controller's internal memory. This is a fatal condition which results in a POR.
9	SCB_BUS_ERROR	This flag is asserted when an error has been detected on System Controller bus.
10	WATCHDOG	This flag is asserted when the System Controller's watchdog reset is about to fire. This is a fatal condition that results in a POR.
11	LOCK_ERROR	This flag is asserted when a single- or double-bit error is detected in the continuously-monitored security lock segments.
12	Reserved	Reserved
13	DIGEST	This flag is asserted when a requested POR digest check is failed. To prevent loss of the DIGEST flag status when using the System Controller Suspend Mode feature, this flag output must be latched by enabling the **Latch System Controller outputs** option in the Tamper macro.

Table 6-7. Tamper Detection Flags (continued)

Flags[31:0]	Flag	Description
14	INST_BUFFER_ACCESS	The flag is asserted when read/write access is performed to system controller's shared buffer using JTAG/SPI interface. The shared buffer holds the data requested by JTAG/SPI instructions.
15	INST_DEBUG	This flag is asserted when debug instruction executed.
16	INST_CHECK_DIGESTS	This flag is asserted when an external digest check has been requested.
17	INST_EC_SETUP	This flag is asserted when elliptic curve slave instructions have been used.
18	INST_FACTORY_PRIVATE	This flag is asserted when factory JTAG/SPI instruction is executed.
19	INST_KEY_VALIDATION	This flag is asserted when key validation protocol is requested.
20	INST_MISC	This flag is asserted when uncategorized SPI slave instruction executed.
21	INST_PASSCODE_MATCH	This flag is asserted when an attempt has made to match a passcode.
22	INST_PASSCODE_SETUP	This flag is asserted when the one-time-passcode protocol is initiated.
23	INST_PROGRAMMING	This flag is asserted when an external programming instruction has been used.
24	INST_PUBLIC_INFO	This flag is asserted when a request for device public information is issued.
25	Reserved	Reserved
26	INST_PASSCODE_FAIL	This flag is asserted when the passcode match fails.
27	INST_KEY_VALIDATION_FAIL	This flag is asserted when the key validation fails.
28	INST_UNUSED	This flag is asserted when the unused instruction opcode is executed.
29	BITSTREAM_AUTHENTICATION_FAIL	This flag is asserted when the bitstream authentication fails.
30	IAP_AUTO_UPDATE	This flag is set if an IAP update occurs (either by IAP system service or auto-update at device boot).
31	IAP_AUTO_RECOVERY	This flag is set if the IAP recovery procedure occurs.

6.7.3 Tamper Response (Ask a Question)

The devices have four built-in tamper responses that can be triggered from the FPGA Fabric. The Tamper macro, exposes these tamper response inputs to the FPGA fabric. The tamper responses are initiated by the System Controller after receiving a tamper event described in the Tamper Detection Flags section, but they may also be initiated on-demand by the user (for example, if the users have their own system level tamper detection connected to FPGA I/Os).

Tamper responses are initiated through dedicated control wires from the fabric to the System Controller.

6.7.3.1 I/O Disable (Ask a Question)

The I/O disable response allows the user design to immediately disable user I/Os (non-dedicated) to prevent any further communication. For FPGA I/Os, I/O disable is implemented on a per-IO basis, according to user pre-programmed per-IO configuration flash bits. The I/O disable persists for as long as the user holds IO_DISABLE asserted. During I/O disable, outputs of FPGA I/O cells are tri-stated and inputs from I/O cells to the internals of the device are specified as low. All output buffers are tri-stated (any configured pull-up or pull-down is still honored) and input buffers are disabled. Configuring FPGA I/Os to support the per-IO disable feature is controlled in the Libero I/O editor column **User I/O Lock Down**.

For PolarFire SoC FPGA MSS-related I/Os, the I/Os are disabled on a per-bank basis, according to user pre-programmed per-IO configuration flash bits. The IO disable persists for as long as the user holds IO_DISABLE asserted. During I/O disable, outputs of MSS related I/O cells of any bank for which lockdown is enabled, are tri-stated with weak-pull up active and inputs from I/O cells to the internals of the device are specified as low. All output buffers are tri-stated (any configured

pull-up or pull-down is still honored) and input buffers are disabled. Configuring PolarFire SoC FPGA MSS I/Os to support the per-IO or per-bank disable feature is controlled in the PolarFireSoC MSS Configurator tool.

Note: The I/O disable feature does not disable transceiver I/Os. Hence, you must put the transceiver blocks in reset before asserting the IO_Disable signal to disable transceiver I/Os.

6.7.3.2 Security Lockdown (Ask a Question)

The security lockdown response activates all the user lock bits to behave as if they are locked. This is irrespective of the underlying user-defined lock state. Any passcodes that were unlocked become immediately locked.

As long as this signal is asserted, the device does not respond to any programming command. The security lockdown response persists until the associated fabric control signal is negated. When the control signal negation happens, lock states return to the user-defined state, but passcodes are locked.

6.7.3.3 Reset (Ask a Question)

This reset response signal sends a reset request to the system controller. The system controller immediately powers down the device and re-executes its normal power-up sequence. This is equivalent to asserting the device reset pin. This tamper response is not available when the System Controller suspend mode is enabled.

6.7.3.4 Zeroization (Ask a Question)

The devices have a built-in tamper response capability that can zeroize (clear and verify) any or all configuration storage elements as per the user setting. Internal volatile memories such as LSRAMs, uSRAMs, and System Controller RAMs are cleared and verified. Once the zeroization is complete, a zeroization certificate can be retrieved using a JTAG/SPI slave instruction to confirm that the zeroization process is successful. This tamper response is not available when the System Controller Suspend mode is enabled.

When zeroization is initiated, it always runs to completion, even if interrupted by a device reset or loss of power. To achieve this, a Zeroization-In-Progress (ZIP) flag is programmed at the start of zeroization. The ZIP flag is checked during device boot and if set, the zeroization procedure is restarted or resumed. Upon completion of zeroization, the device generates a certificate proving that all the requested data has been omitted. The certificate contains the device serial number, a digest of the zeroized memory and a user nonce. The ZIP flag is cleared after generation of the certificate.

The user can monitor the built-in tamper detection flags or other system events and then decide to trigger one of the two types of built-in zeroization requests and zeroize the device. Zeroization is immune to the security lockdown response, which essentially means that asserting a security lockdown does not prevent zeroization from initiating or completing. Factory locks and user permanent locks are not affected by zeroization.

Zeroization is enabled and configured using the Tamper macro configurator as shown in Figure 6-9. During device operation, the zeroization action is initiated by asserting the Zeroize input on the Tamper macro HIGH. Zeroization can also be triggered through a JTAG or SPI slave instruction.

The PolarFire family have the following two zeroization modes (ZMODE):

- **1 = Like New**—All user data and keys are destroyed. The device is effectively returned to its original factory state, allowing it to be programmed like a new device.
- **3 = Non Recoverable (Default)**—All user data, user keys, factory keys, device certificate, and factory data are destroyed. Upon completion of zeroization in the Non Recoverable mode, the only allowed access to the device is retrieval of the zeroization certificate. The device may not otherwise be used again.

The following table lists the status of the various FPGA components during the two zeroization modes.

Table 6-8. Status of Various FPGA Components During the Two Zeroization Modes[1]

			Factory and User Re-configurable Lock Bit Segment			pNVM			sNVM	eNVM[2]
Zeroization Modes	Description	FPGA	Factory Lock segment	User Lock segment	User Permanent Lock Segment	Factory Parameter Segment	User Key	Factory Key		
Like New	Zeroize user data and keys	✓	X	✓	X	X	✓	X	✓	✓
Non Recoverable	Zeroize everything	✓	X	✓	X	✓	✓	✓	✓	✓

Notes:

1. ✓ – part of zeroization process and X – not part of zeroization process.
2. For PolarFire SoC FPGA only.

Regardless of the security settings enabled in the Libero project, default or custom, even without the Tamper macro included, the ZMODE is set to 3.

Libero Default Security: If the Tamper macro is added to a design using default security, then the ZMODE specified within the macro is applied, overwriting the default value.

Custom Security: ZMODE can only be set in the master programming file. If the master programming file does not contain the Tamper macro, the ZMODE is set to 3. Update images can be created with the Tamper macro, however, the ZMODE setting is ignored and remains at the default value set in the Master file. The only method to update the ZMODE settings is with a new master programming file, which includes the Tamper macro.

Notes:

- The tamper macro ZEROIZE mode is controlled by security settings. The default ZEROIZE mode in the tamper macro is **Like New**.
- If the ZEROIZE mode is set as **Like New** in the Tamper macro and programmed with custom security, the user can create an Update Image. If the ZEROIZE mode is set to non recoverable, the setting created with the Update Image is ignored. To change the ZEROIZE mode, the Tamper macro must be modified and the device must be reprogrammed with a new master bitstream with custom security.

6.7.3.4.1 Zeroization of Volatile Memories (Ask a Question)

All volatile user memories (SRAM or registers) are zeroized in all zeroization modes.

- **System Controller Memories**—memories containing sensitive information are written zeros.
- **Fabric Memories**—all fabric LSRAMs and µRAMs are cleared by writing zero to all the memory locations.
- **MSS RAMs (For PolarFire SoC Only)**—All the MSS RAMs are zeroized
- **Fabric Registers**—fabric registers content is destroyed.

6.7.3.4.2 Zeroization of Fabric NVM (Ask a Question)

In all zeroization modes, the user fabric NVM blocks (FPGA fabric configuration memory and uPROM) are zeroized to a verifiable state by applying cycles of erase and program pulses. The content is destroyed and the NVM cells “scrubbed” to ensure there are no usable remnants left in the memory that could be used to identify the prior content.

6.7.3.4.3 Zeroization of pNVM (Ask a Question)

For the applicable modes (Table 6-8), Like the fabric NVM, the pNVM factory parameter segments are zeroized to a verifiable state by applying cycles of erase and program pulses.

6.7.3.4.4 Zeroization of eNVM (Ask a Question)

In all zeroization modes, like the fabric NVM, the eNVM is zeroized to a verifiable state by applying cycles of erase and program pulses.

6.7.3.4.5 Zeroization of System Controller and User Cryptoprocessors Memories (Ask a Question)

For both cryptoprocessors, an on-demand purge function is executed to zeroize all the internal memories. Both cryptoprocessors support a purge function to clear all internal memories.

Important: In PolarFire® FPGA devices, the PF_DRI macro can be used from the fabric to exercise the PURGE functionality of the user cryptoprocessor. Once the data is purged, it cannot be recovered, so use this functionality with caution.

6.7.3.4.6 Zeroization of SRAM-PUF (Ask a Question)

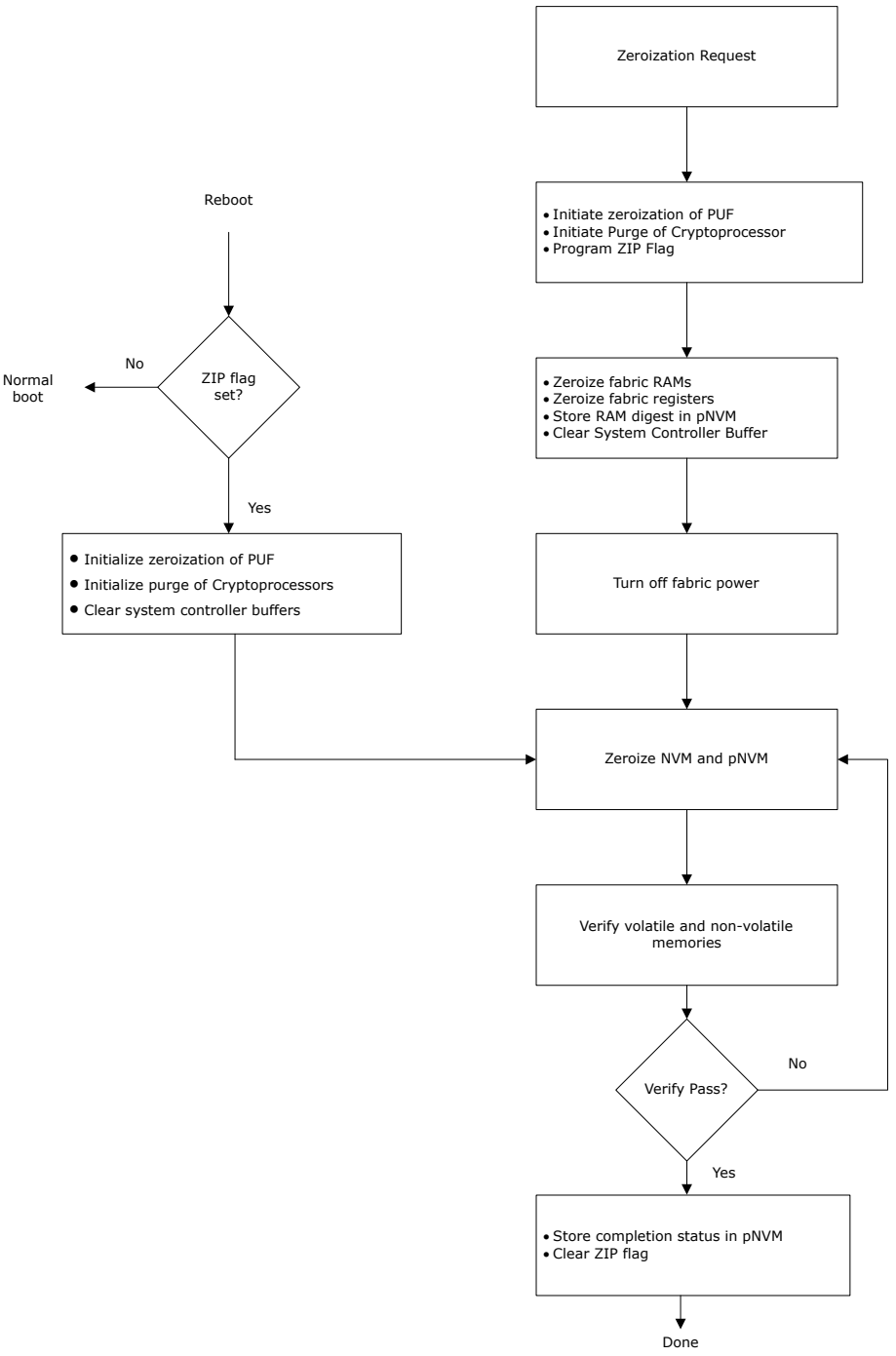
The PUF's built-in zeroization command is executed and the PUF SRAM is turned off.

6.7.3.5 Zeroization Flow (Ask a Question)

The zeroization procedure includes several erase and programming operations to reduce any data remnants in the flash array to undetectable levels (a process known as “scrubbing”). When zeroization is initiated, it always runs to completion, even if interrupted by a device reset or loss of power.

After the activation of zeroization request from the fabric, JTAG, or SPI Slave, the system controller programs a Zeroization-In-Progress (ZIP) flag that act as status flags during the zeroization process. The ZIP flag is checked during device boot and, if set, the zeroization procedure is restarted or resumed. The ZIP flag is only cleared after successful completion of the zeroization procedure, which involves both scrubbing of non-volatile memories and verification thereof. The ZIP flag is only cleared if verification is successful. If verification fails, the zeroization procedure is re-executed until verification passes. The zeroization flow is shown in the following figure. Once zeroization is complete, the zeroization certificate (proof of zeroization) can be read from the device through the JTAG or SPI slave interfaces in response to a challenge from the user, proving the response was fresh and not just replayed from another device or time.

Figure 6-11. Zeroization Flow



7. Data Security (Ask a Question)

The PolarFire family represents the industry's most advanced security programmable RISC-V processor-based devices. Data security protects application data—stored, communicated, or computed at run-time—from being copied, altered, or corrupted. PolarFire FPGA and PolarFire SoC FPGA "S" devices have a dedicated Cryptoprocessor, referred as User Cryptoprocessor, for data security applications.

In PolarFire FPGA and RT PolarFire FPGA, the User Cryptoprocessor is a standalone block, which is accessible to a soft processor in the FPGA fabric.

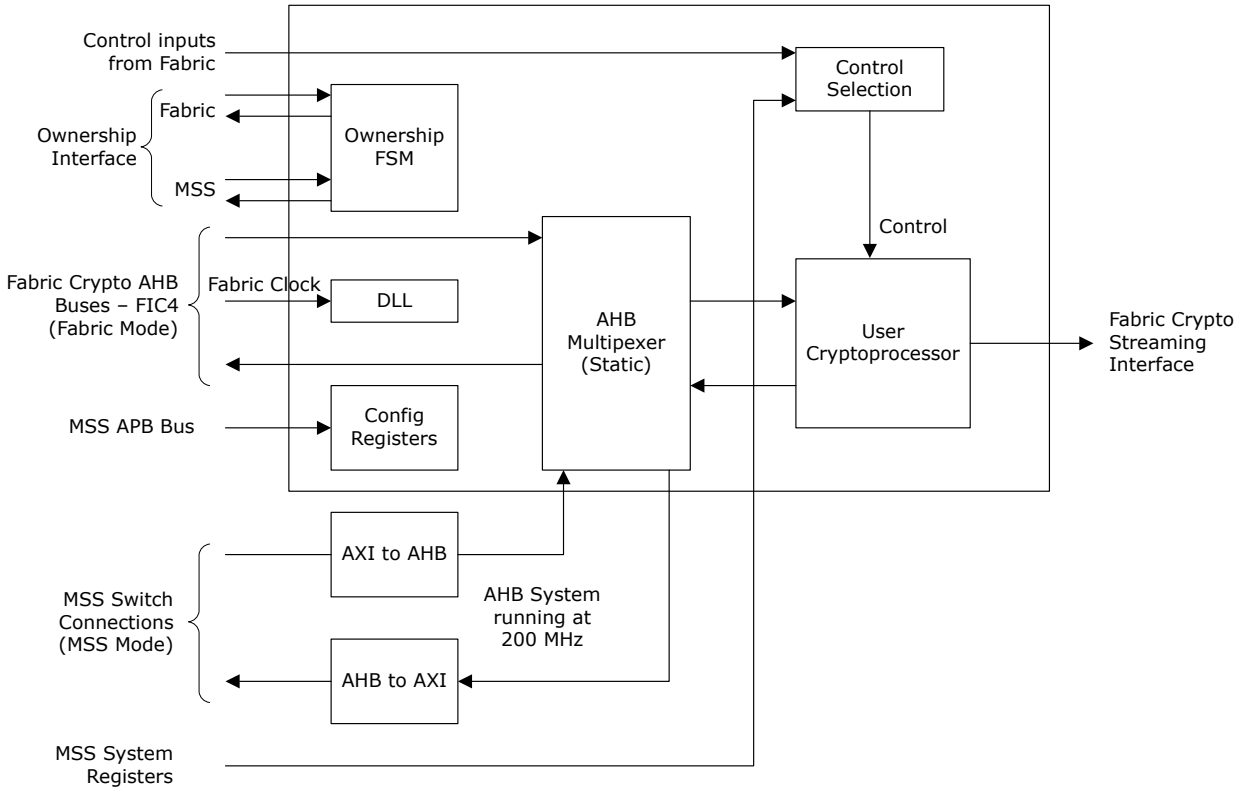
In PolarFire SoC FPGA, the User Cryptoprocessor is integrated within the MSS. For information about MSS, see PolarFire SoC FPGA MSS Technical Reference Manual. The User Cryptoprocessor can be accessed from MSS or Fabric. The default configuration after power-up is defined by the Libero configuration. In PolarFire SoC FPGAs, the User Cryptoprocessor can be configured to operate in following modes using the flash bits set by MSS configurator in the Libero:

Table 7-1. PolarFire® SoC FPGA User Cryptoprocessor Modes

Mode	Crypto Ownership Mode Flash Bits	Description
Reset	0xx	The Cryptoprocessor is not available to the MSS or Fabric and is held in reset
MSS	100	The Cryptoprocessor is only available to the MSS
Fabric	101	The Cryptoprocessor is only available to the Fabric
Shared-MSS	110	The Cryptoprocessor is initially connected to the MSS, and may be requested by the Fabric
Shared-Fabric	111	The Cryptoprocessor is initially connected to the Fabric, and may be requested by the MSS

For more information about MSS configurator, see PolarFire SoC Standalone MSS Configurator User Guide.

Figure 7-1. PolarFire® SoC FPGA User Cryptoprocessor Interfaces Block Diagram



7.1 User Cryptoprocessor Features (Ask a Question)

The User Cryptoprocessor is an Athena TeraFire® EXP-F5200B cryptography microprocessor. It provides complete support for the Commercial National Security Algorithm (CNSA) Suite and beyond and includes Side-Channel Analysis (SCA) resistant cryptography using patented leakage reduction countermeasures. These countermeasures provide strong resistance against SCA attacks such as Differential Power Analysis (DPA) and Simple Power Analysis (SPA). The User Cryptoprocessor is available in the PolarFire family “S” devices.

Table 7-2. User Cryptoprocessor Algorithm Support

Algorithm	Mode	Key Size (bits)
AES	ECB/CBC/CFB/OFB/CTR/GCM	128, 192, and 256
Hash	SHA1	NA
	SHA-224	
	SHA-256	
	SHA-384	
	SHA-512	
	SHA-512/224	
	SHA-512/256	
MAC	HMAC SHA1	NA
	HMAC SHA-224	
	HMAC SHA-256	
	HMAC SHA-384	
	HMAC SHA-512	
	AES-CMAC	128, 192, and 256

Table 7-2. User Cryptoprocessor Algorithm Support (continued)

Algorithm	Mode	Key Size (bits)
KeyWrap	AES	128, 192, and 256
ECC	ECC Point Multiplication	NIST P-Curves – P-192, P-224, P-256, P-384, and P-521. Brainpool Curves – P-256, P-384, and P-521. Supports twisted elliptic curve
	ECDSA Sign/Verify	
	ECC Point Addition	NIST P-Curves – P-192, P-224, P-256, P-384, and P-521. Brainpool Curves - P-256, P-384, and P-521.
	ECC Key Pair Generation	
	ECDH	
RSA	RSA Decryption	1024, 2048, and 3072
	RSA Sign/Verify	1024, 2048, and 3072
DSA	DSA Sign/Verify	1024, 2048, and 3072
Modular Exponentiation	DH/Modular multiplication	1024, 2048, and 3072
True Random Number Generation (TRNG)	SP800-90A CTR_DRBG-256; SP800-90B (draft) NRBG	NA
Key Derivation Function	Key-Tree	256

The User Cryptoprocessor is a hard block in the PolarFire family. The maximum operating frequency is 189 MHz in PolarFire FPGAs/RT PolarFire FPGAs and 200 MHz in PolarFire SoC FPGAs. When the cryptoprocessor is accessed from Fabric, if the frequency of the crypto block is greater than or equal to 125 MHz, select the Use embedded DLL in the fabric interface option for removing clock insertion delay. If the embedded DLL is not enabled, the maximum frequency is limited to 70 MHz.

The User Cryptoprocessor is accessible to MSS (PolarFire SoC FPGA only) or a soft processor in the fabric through the AHB-Lite slave interface for control and primary data input and output. The User Cryptoprocessor has built-in DMA to offload the main processor from doing data transfers between the User Cryptoprocessor and the user memory. The DMA functionality is accessible from fabric through an AMBA AHB-Lite master interface.

Microchip provides an Athena TeraFire Cryptographic Applications Library (CAL) to access the User Cryptoprocessor functions. TeraFire CAL is a C language library that provide functions to access symmetric key, elliptic curve, public key, hash, random number generation, and message authentication code algorithms. The user application running on the main processor must include CAL APIs to perform the cryptographic operations on the User Cryptoprocessor.

For Athena TeraFire CAL and their CAL API descriptions, email FPGA_marketing@microchip.com.

7.2 Port List (Ask a Question)

The following tables list the User Cryptoprocessor port list for the PolarFire family.

Table 7-3. PolarFire® FPGA and RT PolarFire FPGA CRYPTO Port List

Port Name	Direction	Description
AHB_SLAVE	Bus	AHB-Lite slave interface, which is used for control, and primary data input and output.
AHB_MASTER	Bus	AHB-Lite master DMA interface, which may optionally be used for data input and output.
DRI_SLAVE	Bus	Control and status signals are accessible through the DRI.
HCLK	Input	AHB bus clock.
HRESETN	Input	The reset signal, CRYPTO_HRESETN, is active low, synchronous, and is sampled on the rising edge of the clock. Asserts the functional reset of the User Cryptoprocessor block and zeroizes all the internal RAM and registers as PURGE signal. It is necessary to assert this signal for a minimum of two clock cycles to reset the core.
START	Input	External execution initiation input when the User Cryptoprocessor operates in the standalone configuration without a host processor connected to the bus interface. Asserting the START signal causes the User Cryptoprocessor to initiate execution. During execution, the status of the User Cryptoprocessor is reflected by the BUSY and DLL_LOCK ports. This signal must be tied low when the User Cryptoprocessor is used as a co-processor.
PURGE	Input	When the signal is set to '1', it initializes the Zeroization of User Cryptoprocessor internal RAM and registers. For normal operation, this signal must be tied low. The PURGE input is level sensitive, and if the PURGE pin is still asserted when a purge operation completes, another purge operation is initiated. **Note:** In PolarFire devices, the PF_DRI macro can be used from the fabric to exercise the PURGE functionality of the User Cryptoprocessor. It is important to note that once the data is purged, it cannot be recovered, so it is essential to use this functionality with caution.
STALL	Input	Stalls the User Cryptoprocessor for a clock cycle, to introduce variance in the external signatures. The STALL input is expected to be generated by a LFSR circuit in the fabric and asserted randomly for a single cycle to achieve the required stall rates. The STALL input must not be asserted until at least three clock cycles after the HRESETN is de-asserted and the DLL has indicated LOCK for three cycles.
ALARM	Output	Asserted to indicate an uncorrectable memory error condition. An uncorrectable memory error causes the Crypto core to perform a reset and purge. This reset terminates any in-progress operation. For most CAL operations, the CALPKTrfRes() function is used to complete the operation and generates a hardware fault code in the event of an alarm.
BUS_ERROR	Output	Asserted when a HRESP response error is detected by the User Cryptoprocessor AHB master. When set, a reset is required to clear.
BUSY	Output	Execution status signal
COMPLETE	Output	Active high signal, asserted on raising edge of CRYPTO_HCLK to indicate that the User Cryptoprocessor has completed an operation. This signal can be connected to the host microprocessor as an interrupt request signal, enabling the User Cryptoprocessor to interrupt the processor when it completes an operation.
DLL_LOCK	Output	DLL lock status

Important: In PolarFire FPGA and RT PolarFire FPGA devices, the User Crypto processor's control and status registers are accessible to the user logic through DRI interface. See PolarFire Device Register Map for more information about the User Crypto registers.

Table 7-4. PolarFire SoC FPGA CRYPTO Port List [1]

Port Name	Direction	Description
CRYPTO_AHB_SLAVE	Bus	AHB-Lite slave interface, which is used for control, and primary data input and output.
CRYPTO_AHB_MASTER	Bus	AHB-Lite master DMA interface, which may optionally be used for data input and output.
CRYPTO_HCLK	Input	AHB bus clock.
CRYPTO_HRESETN	Input	The reset signal, CRYPTO_HRESETN, is active low, synchronous, and is sampled on the rising edge of the clock. Asserts the functional reset of the User Cryptoprocessor block and zeroizes all the internal RAM and registers as PURGE signal. It is necessary to assert this signal for a minimum of two clock cycles to reset the core.
CRYPTO_GO_F2M	Input	External execution initiation input when the User Cryptoprocessor operates in the standalone configuration without a host processor connected to the bus interface. Asserting the GO signal causes the User Cryptoprocessor to initiate execution. During execution, the status of the User Cryptoprocessor is reflected by the BUSY and DLL_LOCK ports. This signal must be tied low when the User Cryptoprocessor is used as a co-processor.
CRYPTO_PURGE_F2M	Input	When the signal is set to '1', it initializes the Zeroization of User Cryptoprocessor internal RAM and registers. For normal operation, this signal must be tied low. The PURGE input is level sensitive, and if the PURGE pin is still asserted when a purge operation completes, another purge operation is initiated.
CRYPTO_STALL_F2M	Input	Stalls the User Cryptoprocessor for a clock cycle, to introduce variance in the external signatures. The STALL input is expected to be generated by a LFSR circuit in the fabric and asserted randomly for a single cycle to achieve the required stall rates. The STALL input must not be asserted until at least three clock cycles after the HRESETN is de-asserted and the DLL has indicated LOCK for three cycles.
CRYPTO_ALARM_M2F	Output	Asserted to indicate an uncorrectable memory error condition. An uncorrectable memory error causes the Crypto core to perform a reset and purge. This reset terminates any in-progress operation. For most CAL operations, the CALPKTrfRes() function is used to complete the operation and generates a hardware fault code in the event of an alarm.
CRYPTO_BUSERROR_M2F	Output	Asserted when a HRESP response error is detected by the User Cryptoprocessor AHB master. When set, a reset is required to clear.
CRYPTO_BUSY_M2F	Output	Execution status signal
CRYPTO_COMPLETE_M2F	Output	Active high signal, asserted on raising edge of CRYPTO_HCLK to indicate that the User Cryptoprocessor has completed an operation. This signal can be connected to the host microprocessor as an interrupt request signal, enabling the User Cryptoprocessor to interrupt the processor when it completes an operation.
CRYPTO_DLL_LOCK_M2F	Output	DLL lock status
CRYPTO_MESH_CLEAR_F2M	Input	Crypto Mesh error clear should be asserted for at least 5 ns
CRYPTO_MESH_ERROR_M2F	Output	Indicates that the security mesh detected an error because of wires are cut or shorted. When set, stays set until cleared.
Cryptoprocessor Ownership Handshake Interface		
CRYPTO_REQUEST_F2M	Input	Fabric request or is using the Cryptoprocessor
CRYPTO_MSS_REQUEST_M2F	Output	MSS request or is using the Cryptoprocessor
CRYPTO_RELEASE_F2M	Input	Fabric released the Cryptoprocessor

Table 7-4. PolarFire SoC FPGA CRYPTO Port List [1] (continued)

Port Name	Direction	Description
CRYPTO_MSS_RELEASE_M2F	Output	MSS released the Cryptoprocessor
CRYPTO_OWNER_M2F	Output	Indicates that the Fabric owns the Cryptoprocessor and the fabric interface is enabled
CRYPTO_MSS_OWNER_M2F	Output	Indicates that the MSS owns the Cryptoprocessor and the fabric interface is disabled
CRYPTO_REQUEST_F2M	Input	Fabric request or is using the Cryptoprocessor
CRYPTO_MSS_REQUEST_M2F	Output	MSS request or is using the Cryptoprocessor
Cryptoprocessor Streaming Interface		
CRYPTO_XWDATA_F2M	Input	Transfer in data
CRYPTO_XWADDR_M2F	Output	Transfer in data address output
CRYPTO_XENABLE_F2M	Input	Transfer in data request
CRYPTO_XINACCEPT_M2F	Output	Transfer in data accept
CRYPTO_XRDATA_M2F	Output	Transfer out data
CRYPTO_XRADDR_M2F	Output	Transfer out data address output
CRYPTO_XVALIDOUT_M2F	Output	Transfer out data valid output
CRYPTO_XOUTACK_F2M	Input	Transfer out data acknowledgment

Note: 1. Input refers to an input port to MSS from Fabric and output refers to an output port from MSS to Fabric.

7.3 Crypto MSS Mode (For PolarFire SoC FPGA Only) (Ask a Question)

When configured in MSS mode, the User Crypto block is totally disconnected from the fabric interface and connected directly to the MSS switch. In this mode:

1. The Cryptoprocessor is clocked by the MSS PLL at a frequency of 200 MHz.
2. The Cryptoprocessor is connected to MSS AXI switch through asynchronous AXI to AHB and AHB to AXI bridges. The Cryptoprocessor can be used by the MSS processors using CAL driver.
3. The Cryptoprocessor control inputs (GO, PURGE, and so on) are directly controlled by MSS system register bits in normal operation.

7.3.1 Cryptoprocessor Address Map in MSS Mode (Ask a Question)

The Cryptoprocessor uses two address regions in the MSS, the first gives access to the internal Athena Crypto block address space and the second to configuration register uses to control the Crypto functions directly by the internal processors. See the MSS Cryptoprocessor Configuration Registers for register bit definitions.

Table 7-5. Cryptoprocessor Address Map in MSS Mode

Address Range	Description
0x22000000-0x2201FFFF	Cryptoprocessor internal address region
0x20127000-0x201270FF	MSS Cryptoprocessor configuration registers

7.3.1.1 MSS Cryptoprocessor Configuration Registers (Ask a Question)

The Cryptoprocessor includes three APB mapped registers that are used to configure the Cryptoprocessor and control the ownership from the MSS side.

Table 7-6. MSS Crypto Registers

Offset from 0x20127000	Register Name
0x00	MSS Crypto Control register
0x04	MSS Crypto stall seed register

Table 7-6. MSS Crypto Registers (continued)

Offset from 0x20127000	Register Name
0x08	MSS Crypto address upper register

Table 7-7. MSS Crypto Control Register

Bits	Type	Field	Reset	Description
0	RW	RESET	1	Asserts the internal Crypto core reset signal
1	RW	PURGE	0	Asserts the Crypto core purge command input
2	RW	GO	0	Asserts the Crypto core go input
3	RW	RING_OSC_ON	0	Turns on the Crypto core ring oscillators, note turned off at reset
4	RW	STREAM_ENABLE	0	Enables the streaming interface to the fabric
5	RW	STALL_ENABLE	0	Enables the stall system on the Crypto core 0: Operates in Fabric mode using fabric stall signal 1: Internal mode enabled
7:6	RW	STALL_RATE	0	Sets the average stall rate used in internal mode 00: 1 in 8 01: 1 in 16 10: 1 in 32 11: 1 in 64
8	RW	COMPLETE	0	Status signal from Crypto core indicating complete
9	RO	ALARM	0	Status signal from Crypto core indicating alarm condition
10	RO	BUSERROR	0	Status signal from Crypto core indicating it received an AHB bus error response
11	RO	STREAM_ENABLED	0	Indicates that the streaming interface is enabled
12	RO	BUSY	0	Status signal from Crypto core indicating busy
15:13	RO	RESERVED	0	Reads as zero
16	RW	USE_FAB_CLK	0	Forces the block to use the fabric sourced clock when in MSS mode, allowing the streaming interface to operate concurrently with MSS access the AHB buses.
23:17	RO	RESERVED	0	Reads as zero
24	RW	MSS_REQUEST	0	MSS requests Crypto core use
25	RW	MSS_RELEASE	0	MSS releases Crypto core
26	RO	FAB_REQUEST	0	Fabric is requesting Crypto core use
27	RO	FAB_RELEASE	0	Fabric is requesting Crypto core use
28	ROR	MSS_OWNER	0	MSS controls the Crypto core
29	RO	FAB_OWNER	0	Fabric controls the Crypto core
31:30	RO	RESERVED	0	Reads as zero

Table 7-8. MSS Crypto Stall Seed Register

Bits	Type	Field	Reset	Description
31:0	RW	SEED	0	Sets the 32-bit seed value used by the Crypto core stall logic. Any 32-bit value should be used, ideally a random value at each device boot.

Table 7-9. MSS Crypto Address Upper Register

Bits	Type	Field	Reset	Description
5:0	RW	UPPER_ADDR	0	Sets the upper six bits [37:32] of the Address used by the Crypto AHB master, allows the 32-bit Crypto core to interface to the full 38-bit MSS system

7.4 Crypto Fabric Mode (For PolarFire SoC FPGA Only) (Ask a Question)

When in fabric mode, the Cryptoprocessor is dedicated to Fabric and in this mode:

1. The Cryptoprocessor must be clocked from the Fabric and the maximum supported frequency is
   200 MHz. The Cryptoprocessor includes a DLL, which needs to be enabled if the Cryptoprocessor clock frequency is greater than or equal to 125 MHz. If the embedded DLL is not enabled, the maximum frequency is limited to 70 MHz.
2. The Cryptoprocessor is accessible from Fabric through AHB master and slave interfaces using FIC4 (Fabric interface controller). A soft processor is needed in the FPGA fabric to access the Cryptoprocessor using CAL driver..
3. The Cryptoprocessor control inputs (GO, PURGE, and so on) are directly controllable from Fabric ports.
4. No master HBURST connectivity to the fabric. If user logic uses HBURST, it should be tied to 3'b001 in the Fabric design.
5. No master HPROT support is provided. If user logic uses HPROT, it must be tied off to a suitable value.

7.5 Shared-MSS and Shared-Fabric Modes (For PolarFire SoC FPGA Only) (Ask a Question)

During device operation, the ownership of the User Cryptoprocessor can be switched between MSS and Fabric through a handshake interface. The handshake interface is asynchronous with synchronizers inside the MSS as required. The switching is not dynamic, and the handover requires co-operation between the MSS and Fabric design to ensure a secure handover. Assuming the fabric is initially the master, the ownership switching happens as follows:

1. The MSS requests a handover using Table 7-7.
2. The fabric design purges the Crypto core, and release it by asserting CRYPTO_RELEASE_F2M signal.
3. The Crypto core is put into reset, and the clock switched to the MSS by the Crypto ownership FSM.
4. The Crypto core is released from reset by the Crypto ownership FSM and then, it is available to the MSS

The same occurs in the opposite direction.

In the Shared-MSS mode, the Cryptoprocessor is initially connected to the MSS, and may be requested by the Fabric. In the Shared-Fabric mode, the Cryptoprocessor is initially connected to the Fabric, and may be requested by the MSS. The following table lists the handshake interface ports:

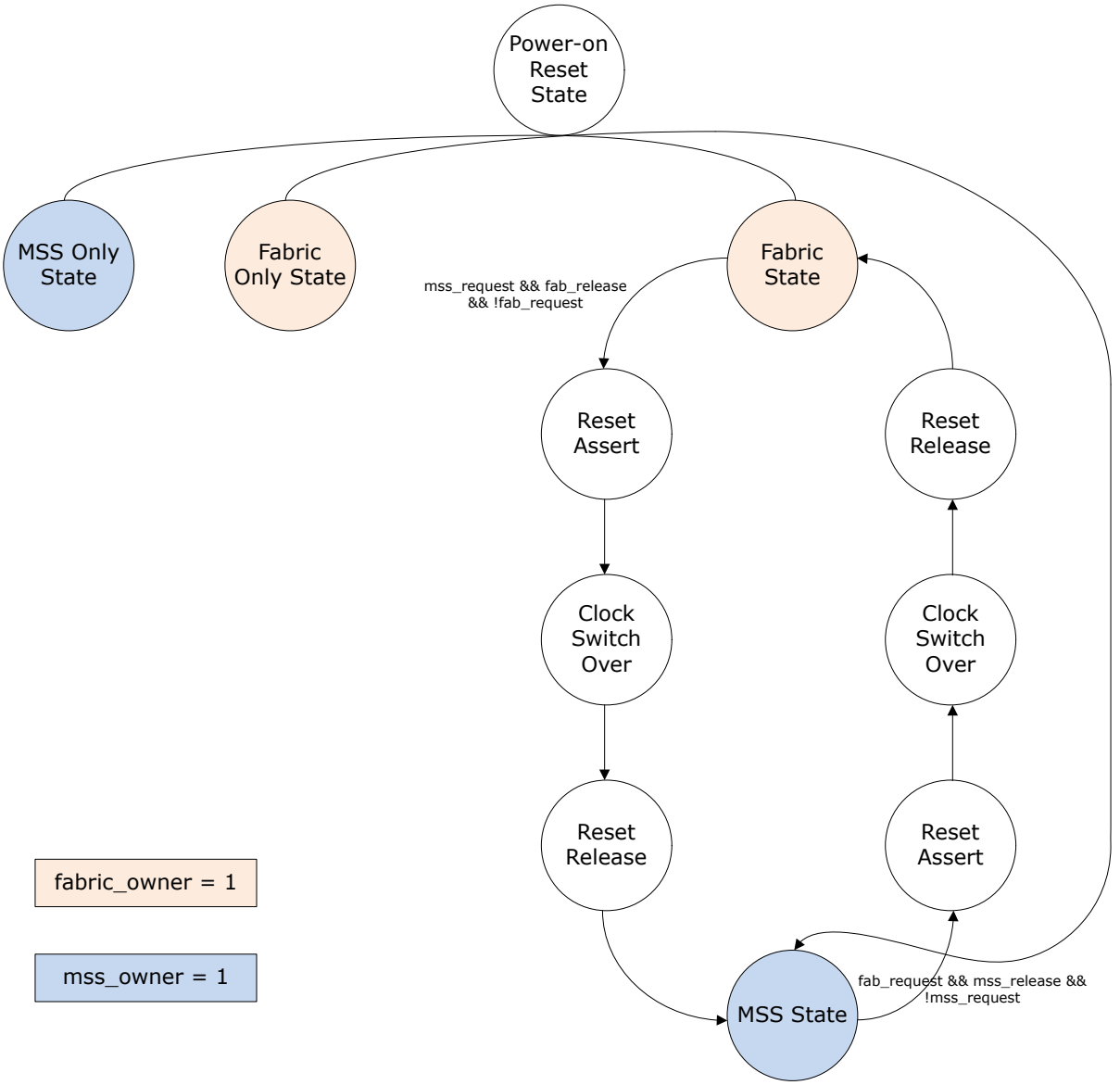
Table 7-10. Crypto Ownership Signals

Port Name	Direction	Description
CRYPTO_REQUEST_F2M	Fabric to MSS	Fabric request or is using the Cryptoprocessor
CRYPTO_MSS_REQUEST_M2F	MSS to Fabric	MSS request or is using the Cryptoprocessor
CRYPTO_RELEASE_F2M	Fabric to MSS	Fabric released the Cryptoprocessor
CRYPTO_MSS_RELEASE_M2F	MSS to Fabric	MSS released the Cryptoprocessor
CRYPTO_OWNER_M2F	MSS to Fabric	Indicates that the Fabric owns the Cryptoprocessor and the fabric interface is enabled
CRYPTO_MSS_OWNER_M2F	MSS to Fabric	Indicates that the MSS owns the Cryptoprocessor and the fabric interface is disabled

All the preceding signals should be considered as asynchronous to the fabric design and appropriate synchronization is used in the fabric design. Within the MSS, the FSM controlling this

interface runs of the System Controller clock (80 MHz) and all inputs are synchronized. The following figure shows the Cryptoprocessor ownership FSM.

Figure 7-2. Cryptoprocessor Ownership FSM



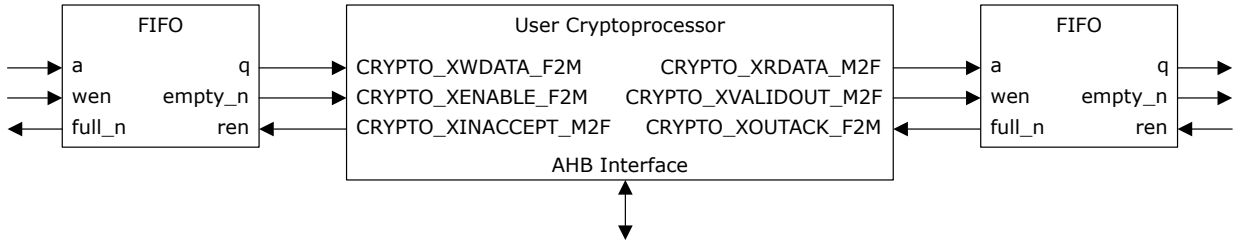
When the Cryptoprocessor is disabled, then the ownership FSM stays in the reset state. Before handing over ownership, that is, asserting the release signals, it is recommended that the current owner purges the Cryptoprocessor to prevent sensitive data being accidentally released to the other system.

The MSS has no notification that the Fabric is requesting the use of the Cryptoprocessor, the fabric design should also connect its request signal to one of the general purpose F2M (fabric to MSS) interrupt signals so the MSS can be informed about the request and take the required actions to release the Cryptoprocessor to fabric.

7.6 Cryptoprocessor Streaming Interface (For PolarFire SoC FPGA Only) (Ask a Question)

The Cryptoprocessor streaming interface comprises of unidirectional data input and output ports and associated handshakes for direct data transfer operations between Cryptoprocessor and fabric logic. The streaming interface can be used to load operands and/or store results as shown in the following figure. The streaming interface can be enabled in MSS, Shared-MSS and Shared-Fabric modes.

Figure 7-3. Cryptoprocessor Streaming Interface Use Case



The streaming interface is synchronous to the CRYPTO_HCLK. The Crypto streaming interface signals are listed in the following table.

Table 7-11. Cryptoprocessor Streaming Interface

Port Name	Direction	Description
CRYPTO_XWDATA_F2M	Input	Transfer in data
CRYPTO_XWADDR_M2F	Output	Transfer in data address output
CRYPTO_XENABLE_F2M	Input	Transfer in data request
CRYPTO_XINACCEPT_M2F	Output	Transfer in data accept
CRYPTO_XRDATA_M2F	Output	Transfer out data
CRYPTO_XRADDR_M2F	Output	Transfer out data address output
CRYPTO_XVALIDOUT_M2F	Output	Transfer out data valid output
CRYPTO_XOUTACK_F2M	Input	Transfer out data acknowledgment

The Cryptoprocessor must be clocked from the fabric for using Crypto streaming interface. The direct transfers are performed when commanded by the direct transfer instructions—DXI: Direct Transfer Block In, and DXO: Direct Transfer Block Out—through AHB slave interface.

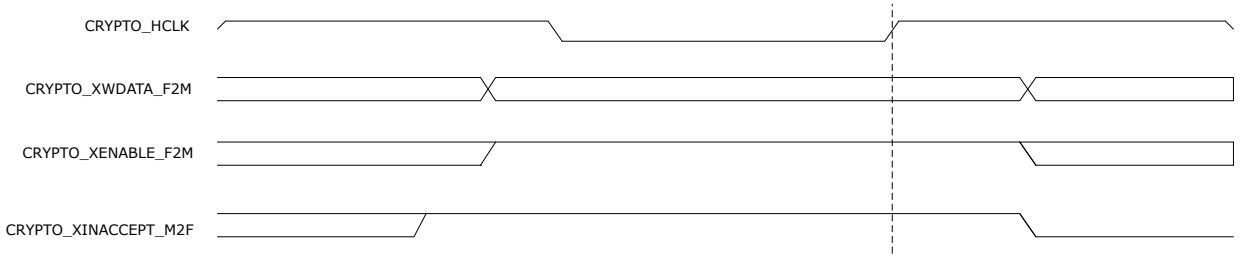
7.6.1 Direct Transfer Input (Ask a Question)

The DXI instruction allows to copy a block of data from the streaming interface input port, CRYPTO_XWDATA_F2M, to a destination register. The destination register address and length of the transfer is specified in the DXI instruction. Transfers are controlled by the CRYPTO_XENABLE_F2M/ CRYPTO_XINACCEPT_M2F handshake, and the DXI instruction will run until the specified number of words have been transferred. The DXI instruction will not complete until the specified number of words have been transferred. If the specified number of words of data never arrive, then the instruction will never complete.

If the direct transfer input port is not used, the CRYPTO_XENABLE_F2M signal should be tied high and the CRYPTO_XWDATA_F2M signal should be tied to a known value.

The transmitting party drives CRYPTO_XWDATA_F2M and asserts CRYPTO_XENABLE_F2M. The Cryptoprocessor indicates that it will accept the data on the next rising edge of CRYPTO_HCLK by asserting the CRYPTO_XINACCEPT_M2F signal. If the CRYPTO_XINACCEPT_M2F signal is negated, then the CRYPTO_XENABLE_F2M and CRYPTO_XWDATA_F2M inputs will be ignored by the Cryptoprocessor. The waveform in the following figure shows an example of a case where data is presented for input to the Cryptoprocessor and accepted in the same clock cycle.

Figure 7-4. Direct Transfer Input Signal Waveforms



If the CRYPTO_XENABLE_F2M signal is tied low, any use of the DXI instruction can cause the Cryptoprocessor to halt, since the DXI instruction blocks until the transfer is complete. Recovery in this case requires a reset.

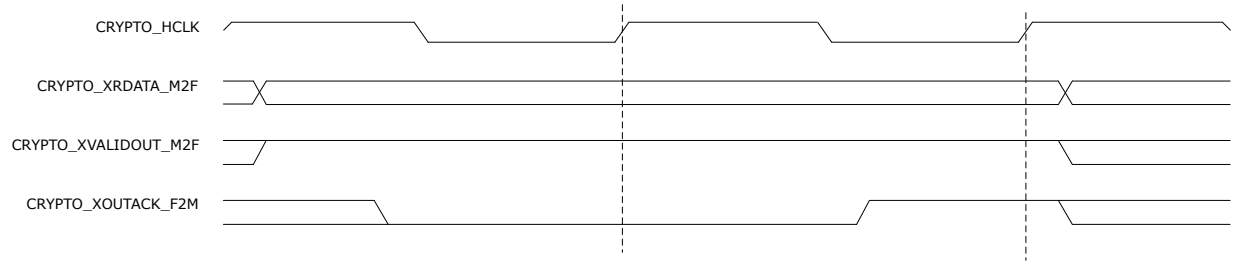
7.6.2 Direct Transfer Output (Ask a Question)

The DXO instruction copies a block of data to the direct transfer output port, CRYPTO_XRDATA_M2F, from the source register. The source register address and length of the transfer is specified in the DXO instruction. Transfers are controlled by the CRYPTO_XVALIDOUT_M2F/CRYPTO_XOUTACK_F2M handshake, and the DXO instruction runs until the specified number of words are transferred.

The DXO instruction will not complete until the specified number of words have been transferred. If the specified number of words of data are never accepted, then the instruction will never complete.

On the rising edge of CRYPTO_HCLK, data is presented on CRYPTO_XRDATA_M2F, and the CRYPTO_XVALIDOUT_M2F signal is asserted. The receiving party indicates receipt of the data by asserting the CRYPTO_XOUTACK_F2M signal, which is sampled on the rising edge of CRYPTO_HCLK by the Cryptoprocessor. The CRYPTO_XOUTACK_F2M signal may be asserted on the same clock cycle that CRYPTO_XVALIDOUT_M2F is asserted or any subsequent clock cycle. The waveform in the following figure shows an example where the CRYPTO_XOUTACK_F2M is asserted one cycle after CRYPTO_XVALIDOUT_M2F is asserted. If CRYPTO_XVALIDOUT_M2F is negated, the CRYPTO_XOUTACK_F2M signal is ignored.

Figure 7-5. Direct Transfer Output Signal Waveforms



If the direct transfer output port is not used, the CRYPTO_XOUTACK_F2M signal must be tied high. If the CRYPTO_XOUTACK_F2M signal is tied low, any use of the DXO instruction causes the Cryptoprocessor to halt, since the DXO instruction blocks until the transfer is complete. Recovery in this case requires a reset.

7.7 Cryptoprocessor Stall System (Ask a Question)

The Cryptoprocessor incorporates a stall system that allows the clock to the Cryptoprocessor core to have clock pulses suppressed in a pseudo random way. This makes the internal operations of the Crypto core harder to infer from external observations. The stall cycles cannot occur during an active AHB cycle and occurs as soon as the AHB buses are idle.

User Cryptoprocessor has a STALL input in PolarFire FPGAs and RT PolarFire FPGAs by default. This input is available in Fabric, Shared-MSS and Shared-Fabric modes of PolarFire SoC FPGAs. The STALL input from fabric is expected to be generated by a LFSR circuit in the fabric and asserted randomly

for a single cycle to achieve the required stall rates. The STALL input must be synchronous to the Cryptoprocessor clock sourced from the fabric. The STALL input must not be asserted until at least three clock cycles after the HRESETN is de-asserted and the DLL has indicated LOCK for three cycles.

In PolarFire SoC FPGA, there is an internal stall generator, which can be enabled or disabled through MSS Crypto Control Register. A 32-bit seed value is provided for initializing the random generator through MSS Crypto Stall Seed Register. It is recommended to set this to a random value on each device reset. When the configuration stall enable bit is set, stall operation is also enabled on the MSS AHB interface using a pseudo-random generator to insert a stall cycle on average of every 8, 16, 32 or 64 clock cycles depending on the set rate in MSS Crypto Control Register.

The internal stall generator can also be used in Fabric mode. In this case, stall cycles are inserted by the internal generator and the stall signal from the fabric.

8. Security Glossary (Ask a Question)

A

Advanced Encryption Standard (AES)

Advanced Encryption Standard (AES) is a 128-bit block cipher with a choice of a 128-bit, 192-bit, or 256-bit key.

ANSI

American National Standards Institute (ANSI) is one of the main organizations responsible for furthering technology standards within the USA. ANSI is also a key player with the International Standards Organization (ISO).

Authentication

Authentication refers to the verification of the authenticity of either a person or of data. An example is a message authenticated as originating from its claimed source. Authentication techniques usually form the basis for all forms of access control to systems and data.

Authorization

Authorization is the process whereby a person approves a specific event or action. In companies with access rights hierarchies, it is important that audit trails identify both the creator and the authorizer of new or amended data. It is often an unacceptably high-risk situation for one to have the power to create new entries and then to authorize those same entries oneself.

B

Block Cipher

A block cipher is a type of cipher that works on a block of data. For example, the DES block cipher works on a block size of 64 bits and the AES block cipher works on a block size of 128 bits.

Most block ciphers operate by alternately performing a reversible (“affine”) non-linear transformation on groups of bits in the block (often using a small carefully designed look-up table), then permuting bits or small groups of bits and then mixing in key information all in a series of “rounds” that are repeated a number of times with different parts of the key or with sub-keys derived from the key.

Block Cipher Modes of Operation

Since block ciphers only work on relatively small blocks of data such as 64 or 128 bits, some form of unambiguous padding is required for messages that are not exact multiples of the block size, and a scheme for handling multiple blocks is needed.

One way to pad is to add a one to the end of the message, and then fill with zeroes until the next block boundary.

The simplest mode for handling multiple blocks of data is just to encrypt each block individually using the same secret key. This is called Electronic Codebook (ECB) mode, since it is equivalent to using a hypothetical (albeit humongous) code book with 2^{128} input-output pairs recorded in it (for the case of a 128-bit block cipher like AES). Though this efficiently scrambles the contents of each block, it is unsuitable for use in most cases because repeated message blocks are encrypted exactly the same way; a situation that is all too common in real messages.

Popular modes of operation that overcome this problem include:

- Cipher Block Chaining (CBC) mode—the output ciphertext of each block is used to randomize the input to the next block using a bit-wise XOR operation.
- Counter (CTR) mode—increments and then encrypts an ever increasing count value, and then uses the result as keying material that is XORed with the plaintext, as in a stream cipher.

The NIST recommended block cipher modes are documented in Special Publication (SP) 800-38 parts A, B, C, D, and E:

- SP 800-38A-Electronic Codebook (ECB), Cipher Block Chaining (CBC), Cipher Feedback (CFB), Output Feedback (OFB), and Counter (CTR) modes
- SP 800-38B-A block cipher-based Message Authentication Code (CMAC)
- SP 800-38C-Counter with Cipher Block Chaining Message Authentication Code (CCM) mode
- SP 800-38D-Galois/Counter Mode (GCM) and Galois Message Authentication Code (GMAC)
- SP 800-38E-XEX Tweakable Block Cipher with ciphertext Stealing (XTS) mode, for use with storage devices

C

CERT

The Computer Emergency Response Team (CERT) is recognized as the Internet's official emergency team. It was established in the USA by the Defense Advanced Research Projects Agency (DARPA) in 1988, following the Morris computer Worm incident, which crippled approximately 10% of all computers connected to the Internet.

CERT is located at the Software Engineering Institute (SEI), a US government funded research and development center operated by Carnegie Mellon University, and focuses on security breaches, denial-of-service incidents, providing alerts, and establishing incident-handling and avoidance guidelines. CERT also covers hardware and component security deficiencies that may compromise existing systems.

CERT is the publisher of Information Security alerts, training, and awareness campaigns. The CERT website is www.cert.org.

Checksum

Checksum is a technique whereby the individual binary values of a string of storage locations on your computer are totaled, and the total retained for future reference. On subsequent accesses, the summing procedure is repeated, and the total compared to the one derived previously. A difference indicates that an element of the data has changed during the intervening period. Agreement provides a high degree of assurance (but not total assurance) that the data has not changed during the intervening period.

A checksum is also used to verify that a network transmission has been successful. If the counts agree it is assumed that the transmission was completed correctly.

A checksum refers to the unique number that results from adding up every element of a pattern in a programmable logic design. Typically either a four- or eight-digit hex number, it is a quick way to identify a pattern, since it is very unlikely any two randomly selected patterns ever have the same checksum. Because they are linear functions, checksums are virtually useless in the face of a malicious adversary who can easily find two messages with the same checksum.

See also, Cyclic Redundancy Check (CRC), Hash Function, and Message Digest.

Cipher

A cipher is the generic term used to describe a means of encrypting data. In addition, the term cipher can refer to the encrypted text itself (ciphertext, as opposed to the unencrypted plaintext). Encryption ciphers use an algorithm, which is a complex mathematical calculation required to scramble the text and a key. Knowledge of the key allows the encrypted data to be decrypted.

Ciphers scramble bits or digits or characters or blocks of bits, whereas codes replace natural language words or phrases with another word or symbol. Modern block ciphers like AES use alternating non-linear substitutions and permutations repeated for a number of “rounds” to encrypt the data. AES, for example, does byte-wide operations on the contents of a 16-byte data block for 10, 12, or 14 rounds, depending upon the key size chosen. Modern ciphers such as AES can be

very resistant to mathematical cryptanalysis, requiring an infeasible number of messages encrypted under the same key and a practically infinite amount of computing power to break them.

Code

Codes are a technique for encrypting data, usually in a natural language such as English, by substituting each word or phrase with a secret word or symbol. Because codes require the cumbersome distribution of large code books (essentially a dictionary-like look-up table) to all the participants they are seldom used today. Ciphers are used instead; they work at the alphabet or binary level and require only a relatively short (256-bit) key to be shared by the users.

Codes can be broken through the use of word frequency analysis, and by correctly guessing plaintext words from the message. For example, it may be known that a weather report is sent at a certain time each day, and by examining several of these messages from known locations the code for “rain” can be guessed. Codes were traditionally used both for confidentiality, and to make telegraph messages, which were charged by length, shorter. Sometimes codes are cascaded with a cipher, a weak form of double-encryption.

Cloning

In FPGAs, cloning is the act of copying a design without making any changes. No understanding of the design or the ability to modify the design is required.

Configuration

The act of programming an FPGA. For SRAM-based FPGAs this must be done at each system power-up to make it functional. Configuration of SRAM FPGAs require the use of an external configuration device, which is typically a PROM (see the entry for PROM) or other type of nonvolatile memory which must be present in the system.

Since they are nonvolatile, flash- and antifuse-based FPGAs only require configuring once, usually during the system assembly process. Flash FPGAs have the option of being reconfigured, but antifuse FPGAs are intrinsically one-time programmable.

Corrupt Data

Corrupt data is data that has been received, stored, or changed, such that it cannot be read or used by the program that originally created the data.

CPLD

A complex programmable logic device is usually a simple low density programmable logic solution. It typically contains macrocells that are interconnected through a central global routing pool. This type of architecture provides moderate speed and predictable performance. CPLDs are traditionally targeted towards low end consumer products.

CRC

See Cyclic Redundancy Check (CRC).

Cryptography

Cryptography is primarily concerned with maintaining the privacy of communications and modern methods use a number of techniques to achieve this. Encryption is the transformation of data into another usually unrecognizable form. The only means to read the data is to decrypt the data using the secret key. Other common cryptographic services include ensuring data integrity, authentication of data sources, and digital signatures.

Cyclic Redundancy Check (CRC)

A class of algorithms for computing a short digest value from an arbitrarily long message, similar to a checksum or hash. CRC may also refer to the resulting digest value itself. The “cyclic” in CRC refers to the underlying cyclic codes describing the mathematics of the algorithm. More precisely, CRC algorithms use linear operations in a Galois Field (usually a binary extension field) which are similar to polynomial division using a generator polynomial.

Common CRC algorithms and their generator polynomials have been standardized for many uses, such as detection of bit errors in data transmission. CRC codes are efficient in detecting large bursts of errors, which suits some types of storage media or transmission channels. Examples of some standardized CRC algorithms are CRC-16-CCITT, which is used by Bluetooth (personal area wireless network), CRC-32-IEEE, which is used in 802.3 (wired Ethernet), and MPEG-2 (video).

Because they are linear operations, they are unsuitable for use in the presence of malicious attacks. An attacker can easily create messages with arbitrary CRC digest values. Cryptographic hash functions must be used instead of a CRC in applications such as digital signatures, data integrity, and authentication where there might be non-random errors (malicious attacks).

See also, Hash Function.

D

Data Encryption

Data encryption is a means of scrambling data so it can be read only by the person(s) holding the key—a password of some sort. Without the key, the cipher (hopefully) cannot be broken and the data remains secure. Using the key, the cipher is decrypted and the data is returned to its original value or state.

For example, using the DES cipher, a key from approximately 72,000,000,000,000,000 possible key variations is randomly generated and used to encrypt the data. The same key must be made known to the receiver so the data can be decrypted at the receiving end. DES can be broken in a matter of hours using a brute-force search because the number of possible keys is low by today's standards.

See also, Public Key Cryptography.

Data Encryption Standard (DES)

An unclassified cryptographic algorithm adopted by the U.S. National Bureau of Standards (NBS, now called the National Institute of Standards and Technology, NIST) for public and government use as Federal Information Processing Standard (FIPS) 46. It is a 64-bit block cipher with a 56-bit effective key length.

DES is a data encryption standard for the scrambling of data to protect its confidentiality. It was developed by IBM in cooperation with the United States National Security Agency (NSA) and published in 1974 by NIST. It is extremely popular and, because at the time it was thought to be so difficult to break, with approximately 72,000,000,000,000,000 possible key variations, was banned from export from the USA. However, restrictions by the US Government on the export of encryption technology to the countries of Europe and a number of other countries were lifted in 2000.

DES was cracked by researchers in 96 days in 1997 by the DESSHALL project and again in 41 days by distributed.net, both projects using thousands of distributed personal computers, where they showed that DES was susceptible to brute force attacks. One of the final blows to the short 56-bit key length of DES was in 1998 when the Electronic Frontier Foundation (EFF) and Cryptography Research, Inc. (CRI) discovered several DES keys, first in 56 hours and then later in only 22 hours, using a custom-designed computer called DES Cracker. The industry then turned to Triple DES, which uses DES three times, as a short term standard to secure transactions. Generally sluggish performance caused an outcry that resulted in a new standard. The NIST has since standardized the Advanced Encryption Standard (AES), based on the Rijndael algorithm, as recommended for all new block cipher applications, although Triple DES is still used extensively in the finance industry for legacy reasons.

Decryption

The process by which encrypted data is restored to its original form in order to be understood/usable by another computer or person.

Denial of Service

Denial of service (DoS) attacks deny service to valid users trying to access a site. Consistently ranked as the single greatest security problem for IT professionals, DoS attack is an Internet attack against a website whereby a client is denied the level of service expected. In a mild case, the impact can be unexpectedly poor performance. In the worst case, the server can become so overloaded that it crashes the system.

DoS attacks are not primarily intended for theft or corruption of data, and are often executed by persons who nurse a grudge against the target organization. The following are the main types of DoS attacks:

- Buffer Overflow Attacks whereby data is sent to the server at a rate and volume that exceeds the capacity of the system, causing errors. This could be just a single long message that exceeds the size of the receiving buffer.
- SYN Attack. This takes places when connection requests to the server are not properly responded to, causing a delay in connection. Although these failed connections eventually time out, they can result in denial of access to other legitimate requests for access should they occur in large volumes.
- Teardrop Attack. The exploitation of TCP/IP protocol features whereby large packets of data are split into bite-sized chunks, with each fragment being identified to the next by an offset marker. Later the fragments are supposed to be reassembled by the receiving system. In the teardrop attack, the attacker enters a confusing offset value in the second (or later) fragment, which can crash the recipient's system.
- Ping Attack. This is where an illegitimate attention request or 'ping' is sent to a system, with the return address being that of the target host (to be attacked). The intermediate system responds to the ping request but responds to the unsuspecting victim system. If the receipt of such responses becomes excessive, the target system is unable to distinguish between legitimate and illegitimate traffic.
- Viruses. Viruses are not usually targeted but where the host server becomes infected, it can cause a DoS.
- Physical Attacks. A physical attack may be little more than cutting the power supply, or perhaps the removal of a network cable.

Differential Power Analysis (DPA)

An analysis technique that relies on multiple measurements of a security device's instantaneous power consumption in order to recreate a secret being manipulated inside the device. Simple and Differential Power Analysis were first reported by Paul Kocher et al in 1989. Generally this class of techniques uses statistical methods to amplify the effects of small unintentional leakages of the secret information in power consumption measurements, buried in large amounts of noise.

For example, if the same secret key is used to process multiple independent blocks of data, a DPA attack might be mounted to determine the secret key using anywhere from a handful of power consumption traces to over a million, depending on the magnitude of the leak, the amount of noise that may be obscuring the secret data, and what countermeasures are used. Systems that handle large amounts of data using the same key, or which can be repeatedly be given random or chosen input data which is then processed using the secret key, are especially vulnerable to DPA.

Diffie-Hellman Key Exchange

The Diffie-Hellman key exchange algorithm, named after Whitfield Diffie and Martin Hellman, was the first public key algorithm ever published, in 1976. The third inventor was Ralph Merkle. With it, they revolutionized the field of cryptology, and made secure communication over the Internet feasible.

It is based upon the difference in difficulty of a particular function and its inverse, namely the ease of exponentiation and the difficulty of computing the discrete logarithm (both) in a finite field.

When the numbers involved are large (that is, over one thousand bits) the difference in difficulty is approximately 30 orders of magnitude, and grows with the size of the numbers.

The Diffie-Hellman protocol allows two entities (computers or people) who do not have nor have ever had a secure channel between them to compute a common secret using public information they send to each other. Anyone eavesdropping on the conversation would find it computationally infeasible to learn the shared secret, even though they see all the messages. This is because each of the parties to the computation holds one secret they do not transmit, but use in the exponentiation formula to compute a value that is practically impossible to reverse; and this is the value that is sent over the insecure channel.

Prior to this invention, secret communications always involved having a shared secret key. This shared key had to be transmitted securely between the parties by a trusted courier or some similar means before encrypted communication over an insecure medium such as radio or telegraph could be done using the shared secret key. Since each possible pair of entities might need a unique shared key, the system did not scale well to large groups where the number of combinations can be exceedingly large. It can thus be claimed that public key cryptography made practical encrypted communications between large numbers of parties, such as shopping with credit cards on the Internet, that was not feasible before.

Digital Signatures

With the advent of public key cryptography a number of new cryptographic services were born, with digital signatures perhaps being the most important.

The concept of digital signatures is that the signer performs a computation using a secret key that only the signer knows, but which can be confirmed by anyone having the matching public key.

Using the RSA cryptosystem, this is done mainly by interchanging the usual role of the private and public keys: In “normal” encryption, any sender encrypts the message using the recipient's public key and the recipient decrypts it using the private key that only the recipient knows. In the RSA digital signature algorithm, the signer “encrypts” the message using the private key that only the signer knows, and any verifier can “decrypt” the signature and verify it is the same as the message using the freely available public key.

Since only the signer has a copy of the private key, it is difficult for the signer to repudiate any valid signatures. This is different from symmetric (shared key) systems where at least two parties must be in possession of a key for it to have any use.

In practice, the whole message is not signed. Because of computational efficiency, and to reduce the size of the signature that has to be transmitted along with the message, a hybrid scheme is used. The message is first hashed; that is, a short digest is computed from the message, and it is this digest that is signed using the private key. The verifier also hashes the received message, and verifies the signature matches the hash using the public key. Standards such as PKCS#1 specify other details important to the security of the system such as how padding is done, and the use of random nonces.

There are variations of this hybrid signature scheme using the ElGamal and elliptic curve cryptosystems.

Disable

Disabling is the process by which hardware or software is deliberately prevented from functioning in some way. For hardware, it may be as simple as switching off a piece of equipment, or disconnecting a cable. It is more commonly associated with software, particularly shareware or promotional software, which has been supplied to a user at little or no cost, to try before paying the full purchase or registration fee. Such software may be described as “crippled”, in that certain functions, such as saving or printing files, are not permitted. Some in-house development staff may well disable parts of a new program, so that the user can try out the parts that have been developed, while work continues on the disabled functions.

Disabling is also often used as a security measure. For example, the risk of virus infection through the use of infected floppy diskettes or USB thumb drives can be greatly reduced by disconnecting a cable within the PC, thereby disabling the drive. Even greater protection is achieved by removing the drive altogether.

E

Electromagnetic Analysis (EMA)

A form of side-channel analysis where the unintentional information leakage from the cryptographic system is via electromagnetic (EM) emissions. Electromagnetic emissions have been a well-known source of leakage, prompting the US government to specify EM requirements for secure applications in what are called TEMPEST requirements. In one example of EM leakage, the van Eck radiation of a display terminal is read from a distance of hundreds of meters using simple equipment.

Many power analysis (PA) classifications have an EMA analog where a similar attack can be performed using essentially the same method for EMA as for PA. For instance, differential electromagnetic analysis (DEMA) is the analog of differential power analysis (DPA), and can be used to extract the AES key, for example, from an unprotected device using an RF antenna and amplifier instead of a current monitor. One important difference is that in EMA the usable signal is often more strongly modulated on harmonics of the fundamental frequencies due to the better propagation properties of higher frequencies; therefore demodulation is often used to bring these harmonic-related signals back to baseband before completing the analysis. Often, it is possible to focus the area of the attack by the placement of the antenna, resulting in an improved signal-to-noise ratio (from the analyst's perspective).

Elliptic Curve Cryptography (ECC)

Elliptic curve cryptography is a public key cryptographic system defined using elliptic curve polynomials in finite fields. The important principle is related to the Diffie-Hellman problem of finding discrete logarithms in finite fields, but instead of exponentiation the group operator is scalar point multiplication. Since some of the most efficient (non-quantum) algorithms available for finding discrete logarithms do not work on elliptic curves, the key sizes required for elliptic curves can be much shorter than for the Diffie-Hellman (or RSA) cryptosystems for a roughly equivalent security strength.

As an example, for a security strength of around 128 bits, i.e., requiring an attack with approximately 2^{128} operations to brute-force attack on AES-128. ECC requires a key size of 256 bits, whilst RSA requires around 3072 bits. As a result, ECC is generally substantially more computationally efficient than RSA. ECC's “hard problem” is susceptible to Shor's attack using a quantum computer. When suitable quantum computers are available, ECC will become ineffective at providing security.

Encryption

The process by which data is temporarily rearranged into an unreadable or unintelligible form for confidentiality, transmission, or other security purposes.

Entropy

In information theory, entropy is a measure of the uncertainty of a system. For example, if all the bits of an n-bit binary number are unbiased (equal probability of a one or zero) and independent (not correlated with any other bits) and are unknown, then the number “contains” n bits of entropy and is said to have full entropy.

In this case, there would be no better method of guessing the number than a brute force search attempting every possible value (2n values), with an expected match after about half of the values have been tried. However, if the bits were known to be biased (for example: 1/3 were randomly selected as zero, and 2/3 as one), then the entropy would be less than n bits and a more efficient search could be performed that started by guessing more ones than zeroes, with an expected match much earlier than in the unbiased case.

In cryptographic applications it is usually critically important that random numbers, such as those used for secret keys, have full entropy.

There is a beautiful and unexpected relationship between entropy as used in information theory and entropy as used in the physical sciences (such as thermodynamics), but in most practical applications the two uses are distinct.

F

Fault Analysis

Fault attacks attempt to break the security of a cryptographic implementation by injecting energy in the form of voltage glitches on the power supply, or light or concentrated electromagnetic energy to generate a fault in the device. Other ways to induce faults can be to operate the temperature or voltage outside the normal ranges. Faults can be used by an adversary in different ways, depending on the precise fault and the design of the system. For example, if a microcontroller can always be made to take a certain branch, whether or not a passcode is matched properly, the passcode protection may be made ineffective. Another broad class of fault attacks are called differential fault analysis (DFA) where if the correct and one or more faulted outputs of a cryptographic calculation using the same secret key can be obtained, the key may be extracted. Fault analysis a subset of the general class of active side channel analysis.

Firmware

Firmware is a sort of halfway house between hardware and software. Firmware often takes the form of a device that is attached to or built into a computer–such as a ROM chip–which performs some software function but is not a program in the sense of being installed and run from the computer's storage media.

Flash FPGA

A flash-based FPGA uses flash memory technology to control the switching of the interconnect and the operation of the logic elements. Flash-based FPGAs are nonvolatile, live on power-up, and reprogrammable They are and relatively secure from reverse engineering or cloning since the programming bitstream is only required to be loaded once, during the initial configuration. This can be performed either in a trusted location, or using strong cryptographic techniques in less trusted locations.

Most flash FPGAs also allow for secure field upgrades using encrypted bitstream files and a decryption key which was loaded in the nonvolatile memory during the initial configuration process. The discovery of the possibly millions of configuration bit values stored in the internal nonvolatile flash memory cells is considered a very difficult problem, thus contributing to the security of flash FPGAs.

FPGA

A field programmable gate array is a very complex programmable logic device (PLD). The FPGA usually has an architecture that comprises a large number of simple logic blocks, a number of input/output pads, and a method to make the desired connections between the elements. The largest programmable logic devices have gate counts running into the millions, and modern devices often have many ancillary hardware blocks such as microprocessor units (MPUs), phase-locked loops (PLLs), static random access memory (SRAM), specialized digital signal processing (DSP) elements, embedded nonvolatile memory (eNVM).

These devices are user customizable and programmable on an individual device basis. They are valued by designers for their flexibility.

H

Hacker

A hacker is an individual whose primary aim in life is to penetrate the security defenses of large, sophisticated, computer systems. A truly skilled hacker can penetrate a system right to the core

and withdraw without leaving a trace of the activity. Hackers are a threat to all computer systems that allow access from outside the organization's premises, and the fact that most hacking is just an intellectual challenge should not allow it to be dismissed as a prank. Clumsy hacking can do extensive damage to systems even when such damage was not intentional.

In 2015 the US government issued a report from the Defense Cybersecurity Culture and Compliance Initiative (DC3I) that indicated there were around 100,000 attempted malicious network attacks on Department of Defense assets every day.

Hash Function

A cryptographic hash, also called a message digest, is a publicly-known function that takes as its input a message of (almost) any length and compresses it into a random-like short message called a digest or fingerprint. “Hash” may refer to either the function or the output digest value itself.

Commonly used digest output lengths are from 160 to 512 bits. Hash functions are important components of integrity, authentication, and digital signature schemes, amongst other uses.

A good cryptographic hash must have several properties:

1. **Pre-image resistance**—it must be infeasible to determine any part of the input message from the output digest.
2. **Second pre-image resistance**—it must be infeasible to generate any input message with a given output digest.
3. **Collision resistance**—it must be infeasible to find any two input messages with the same output digest.

These imply a strong one-way-ness property for cryptographic hash functions. For a good hash function, if even one bit of the input message is changed, roughly one-half of the output bits changes pseudo-randomly.

Commonly-used hash functions are MD5, SHA-1, and the SHA-2 family of hashes, including SHA-256, SHA-384, and SHA-512. Though still in widespread use, MD5 is considered broken, and SHA-1 has some serious weaknesses. The US government agency NIST recently completed a competition for a new family of hash functions called SHA-3 that must have better security than the current standard hash functions. An algorithm called Keccak was selected as the winner. It uses different principles than most prior hash functions, and is very efficient in hardware implementations.

Cryptographic hashes are related to, but not the same as hashes used in computer science for creating tables for looking up data by value. Those hash functions do not have the three security properties (above) required for a cryptographic hash and as a result must never be used in a cryptographic (adversarial) setting.

See also, Cyclic Redundancy Check (CRC) and Security Strength.

HEX / Hexadecimal

Hexadecimal, or hex, is a base 16 numbering system (as opposed to the usual decimal base 10). Hex is a useful way to express binary computer numbers. A byte is normally expressed as having 8 binary bits. Two hex characters of four bits each, called nibbles, represent eight binary digits, also known as a byte. Nibbles are sometimes represented using the sixteen 8-bit ASCII character set symbols 0-9 and a-f (or A-F) for human consumption such as when displayed or printed.

I

In-Application Programming (IAP)

IAP is the ability of a microcontroller to run an application that reconfigures (reprograms) its own nonvolatile program code storage. Some flash FPGAs having a built-in microcontroller natively support both IAP and ISP.

See also the entries for “In-System Programming (ISP)”.

In-System Programming (ISP)

ISP is the ability to program and reprogram an FPGA that is mounted on a circuit as part of a functional system. Flash and SRAM-based FPGA technologies support ISP.

Intellectual Property (IP)

Intellectual property is defined as creative, technical, and intellectual products, often associated with custom circuit designs implemented in ASIC or programmable logic architectures.

Invasive Attack

Invasive attack is an attack on a semiconductor to determine its functionality and requires physical entry to the part. Typical methods include probing, etching, and FIB (focused ion beam) intrusion.

See also the entries for “Noninvasive Attack” and “Semi-Invasive Attack”.

M

Malicious Code

Malicious code includes all and any programs (including macros and scripts) that are deliberately coded in order to cause an unexpected (and usually unwanted) event on a PC or other system. However, whereas antivirus definitions (vaccines) are released weekly or monthly, they operate retrospectively. In other words, someone's PC has to become infected with the virus before the antivirus definition can be developed. In May 2000, when the Love Bug was discovered, although the antivirus vendors worked around the clock, the virus had already infected tens of thousands of organizations around the world, before the vaccine became available.

Message Authentication Code

A Message Authentication Code (MAC) is similar to a hash function in that it computes a random-like output digest from any size input message, but unlike a hash, which is a public function that anyone can compute, a MAC uses a secret key so that only those in possession of the secret can correctly create or verify it.

Message Digest

See Hash Function.

Modes of Operation

See Block Cipher Modes of Operation.

N

National Institute of Standards and Technology (NIST)

NIST was formerly the National Bureau of Standards (NBS). NIST is the government agency that sets weights and measures for the United States. It is an agency of the Commerce Department. In security and cryptography, NIST works closely with the National Security Agency (NSA), a part of the Defense Department, to set government standards and make recommendations for private sector use.

Nonce

A number used only once. Nonces are an important element of many protocols because they help protect against replay attacks. By incorporating a unique nonce in the protocol the attacker cannot replay data from an earlier run of the protocol that, by definition, used a different nonce. Nonces are also often required for initialization vectors such as those used with some block cipher modes of operation, or stream ciphers. If the same initialization vector is used with the same key on more than one message, the security of the cipher mode can be very seriously compromised. Nonces are also used in some types of digital signatures.

Common ways of generating nonces are by counting, using a time stamp, or using a sufficiently large random number whose chance of repeating is vanishingly small. The best choice depends

upon the circumstances, because each of these has its own difficulties and advantages. For instance, in many systems it is very difficult to be sure of a secure time source. With a counter, the issue is to make sure that it is never reset or a count value used twice, even if the power supply is tampered with. In other systems there may not be a good source of entropy with which to create sufficiently large random numbers.

Noninvasive Attack

A noninvasive attack is an attack on a semiconductor to determine its functionality that does not require physical entry to the part. Types of attacks include actively varying voltage levels to gain access, and passive side-channel analysis.

See also the entries for “Invasive Attack”, “Semi-Invasive Attack” and “Side-Channel Analysis”.

Nonvolatile

A device is nonvolatile if it does not lose its contents when its power is removed. Nonvolatile memory is useful in microcomputer circuits because it can provide instructions for a CPU as soon as the power is applied, before secondary devices, such as disk, can be accessed. Nonvolatile memories include metal-mask read-only memory (ROM), fusible-link programmable ROM (PROM), ultra-voilet- erasable electrically-programmable ROM (UV-EPROM), and electrically-erasable PROM (EEPROM) including “flash” memory, a special type of EEPROM where the memory is erased in large blocks rather than by individual bytes or words, making it much faster and also less expensive.

O

Overbuilding

Unscrupulous contract manufacturers (CM) overbuild on a program or contract and sell the excess on the gray market.

P

Power Analysis

See Side-Channel Analysis (a super-set of power analysis), Simple Power Analysis, and Differential Power Analysis (DPA) (both sub-types).

Public Key Cryptography

Public key cryptography is based upon the revolutionary principle that instead of using a shared secret key for two or more parties to communicate privately, as in all ciphers and codes before 1976, a key can have two parts: a public part and a secret part. The public part may be communicated to anyone and does not have to be kept secret. It can be used for encryption, thus allowing anyone in the world to encrypt a message intended for a given recipient. Only the recipient, namely the holder of the secret part of the key, can perform the decryption.

The first public key scheme, called the Diffie-Hellman key exchange algorithm, was published by Whitfield Diffie, Martin Hellman, and Ralph Merkle, in which they used mathematics based upon the difficulty of the discrete logarithm problem to generate a shared secret key between two parties that had no prior secret communication. This was later expanded into the ElGamal encryption system for enciphering messages. Shortly after, Ron Rivest, Adi Shamir, and Len Adleman published the now well known RSA encryption scheme named after them, based upon the difficulty of factoring large primes.

Besides greatly simplifying key distribution between anonymous parties, public key cryptography also introduced a new cryptographic service called digital signatures. The holder of the secret key “signs” a message with a message-dependent code only they can generate, and anyone in possession of the public key can verify the integrity of the data and the correctness of the signature. Since only one person holds the private key (unlike in symmetric key systems where at least two people have the key), it makes it much more difficult for the signer to later repudiate their signature.

Though attributed to the inventors mentioned above who were the first to publish their results, it is now known that public key cryptography had been invented a few years earlier by James Ellis, Clifford Cocks and Malcolm Williamson, employees of the General Communications Headquarters (GCHQ), a British government agency, which kept their results secret and largely failed to recognize the importance of the discoveries.

R

Random Numbers

Random numbers are used extensively in cryptography, for generating secret keys and nonces, for example. In most implementations, they are binary numbers. The random numbers must be unknown and unpredictable to an adversary. An n-bit binary number which is completely unpredictable and unknown to an adversary is said to contain n bits of entropy; if the adversary has a better than 50%- 50% chance of guessing some of the bits, the entropy is reduced.

True random numbers are derived from an unpredictable physical source, most often some form of electrical noise although radiation decay and some other physical processes are also sufficiently random though less practical. If each bit generated by the physical process is unbiased and uncorrelated with all the other bits then it has one bit of entropy. By gathering many such bits, one can accumulate a large amount of entropy.

Pseudo-random numbers are derived from a deterministic computational process. With good algorithms pseudo-random bits can be computationally indistinguishable from true random bits. However, no matter how many such bits are generated, the entropy content is limited by the lesser of the initial true random seed used to initialize the computation process and the number of bits of internal state storage. If an adversary were able to learn the internal state of a pseudo-random generator (by guessing or other means) he could predict all future values, and may even learn something about past values.

Important standards related to random numbers include:

- SP 800-90A-(NIST) Recommendation for Random Number Generation Using Deterministic Random Bit Generators. Part A covers deterministic random bit generators. Parts B and C (still in draft form) cover non-deterministic entropy sources and how to combine them to create hybrid random bit generators
- FIPS 140-2 Annex C-(NIST) Approved Random Number Generators for FIPS PUB 140-2
- SP 800-22-(NIST) A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications
- Test Suite-(BSI) Random number Test Suite
- AIS-31-(BSI) Functionality classes and evaluation methodology for physical random number generators

Reverse Engineering

Reverse engineering is the act of examining a design to understand exactly how it works, perhaps with the intent to copy the design. The design is then altered to differentiate it from the original design for the purpose of improving upon it or to prevent legal action because of the theft, or to insert a “Trojan Horse”. Also, reverse engineering is sometime used to determine if any patents are being violated. Some applications of reverse engineering are legal depending on the subject and the legal jurisdiction, while other cases may be considered theft.

S

Security Strength

Security strength is a rough measure of the work effort, log base 2, required to attack a given cryptographic problem. For a well-designed block cipher, the best approach an attacker has is a brute force search over all the possible keys. In this case the security strength, measured in bits, is

the same as the length of the key (in bits). For example, AES-128 has an estimated security strength of 128 bits since the best known attack is a brute force search of all 2^{128} keys.

For a well-designed hash function, the security strength varies depending upon which of the security properties is being depended upon in its usage (see the entry for Hash Function). For pre-image resistance and second-pre-image resistance, the security strength is the same as the digest output size (in bits). For collisions, the security strength is very nearly half the number of bits in the output. The reduced strength is due to the Birthday Attack, which is applicable in this situation.

For public key algorithms, the security strength is a complicated function of the key size but also depends upon the most efficient attack algorithm known. Since the most efficient attacks on RSA or Elgamal do not work on elliptic curve algorithms, shorter keys can be used with elliptic curve cryptography for a given security strength. For elliptic curve algorithms, the keys must be roughly twice as long as for symmetric algorithms such as AES. RSA, Diffie-Hellman, and Elgamal all require comparable (to each other) but much longer keys. For example, a one-thousand bit RSA key is roughly equivalent in security strength to an 80-bit symmetric key and a 160-bit elliptic curve key.

Not all block ciphers and hash functions have the ideal security strength shown above. If some attacks are known that reduce the work factor to find the key (or pre-image, or collision, etc.) caused by a weakness in the algorithm, then the security strength is correspondingly downgraded. For instance, the MD5 hash algorithm design in 1994, which has a digest size of 128 bits, must have a collision resistance security factor of 64 bits (which in itself is marginal), but attacks had been found by 2006 that reduced the work factor to less than 224, (one trillion times easier) making it unsuitable for cryptographic applications since the latest/best attack algorithm known can find an MD5 collision in less than one minute on a standard notebook computer.

Security strength is often equated with the length of time the algorithm or secret data is used. For short term (ephemeral) use, 80 bits may be enough for strong security, but for data that has to last a few years 100 bits or more is recommended, and for data that may have to keep secret for several decades, 128 bits is recommended. This is because attacks only get better, and computing equipment has been getting faster and cheaper due to Moore's Law.

Grover's algorithm, applicable to quantum computing, is expected to reduce the security strength of most cryptographic algorithms by a square-root factor, i.e., by about halving the security strength measured in bits. For example, to maintain a security strength of 128 bits in a post-quantum world, one should use AES-256. Shor's algorithm, which is applicable to many public key algorithms such as ECC, RSA, and DH (but not most block ciphers or hashes) has a more devastating effect on the security strength, making these algorithms next to worthless.

Semi-Invasive Attack

A semi-invasive attack is an attack on a cryptographic device such as an integrated circuit which may involve removing all or part of the package, but does not require internal probing or cutting of circuit lines. Instead, the attack is carried out using optical or electron microscope observations or by injecting (temporary) faults optically or electromagnetically, which do not require the active device to be touched. This family of attacks is generally less expensive to conduct than invasive attacks but more expensive than other types of active fault attack or passive side-channel analysis.

See also, Invasive Attack, Differential Power Analysis (DPA), and Side-Channel Analysis.

Side-Channel Analysis

Passive side-channel analysis is a noninvasive (or occasionally a semi-invasive) analysis technique which attempts to break the security of a cryptographic system by monitoring information unintentionally leaked via side-channels. These side-channels could be power consumption, electromagnetic emissions, optical emissions, thermal signatures, or timing of response times, for example. As all “real world” implementations of cryptographic systems have unintended side-channels, they represent a serious threat to the security provided by these systems.

Active side-channel analysis attempts to break the security by using light, electrons, electromagnetic energy, or other active sources of energy to probe or disrupt the target system. Sometimes active and passive techniques are combined.

See also, Simple Power Analysis, Differential Power Analysis (DPA), Electromagnetic Analysis (EMA), and Fault Analysis.

Simple Power Analysis

Simple power analysis is a side-channel analysis technique based upon one or just a few measurements of a security device's power consumption. Information about secrets being manipulated inside the device are unintentionally leaked out via the instantaneous power consumption of the device. In some cases, a secret key can be read more-or-less directly from simple observations of a single oscilloscope trace.

SRAM FPGA

An SRAM FPGA is an FPGA that utilizes SRAM (Static Random Access Memory) technology to configure the interconnect and to define the logic. SRAM FPGAs are reprogrammable, volatile, and require a boot-up process to initialize. SRAM FPGAs are generally considered less secure than flash or antifuse technology based FPGAs because the design configuration bitstream has to be loaded from an external component at each power-up cycle.

See also, Differential Power Analysis (DPA).

T

Tamper Detection

Tamper detection is an alarm set off when any of a number of possible tamper detection sources is triggered. Common tamper detectors for high-end security integrated circuits include voltage, clock and temperature alarms, internal redundancy violations, physical tampering alarms such as a failure of a mesh covering important circuits, etc.

See also, Zeroization, which is one possible response to a tamper detection alarm.

Tamper Resistant Packaging

Often used in smart card systems, tamper resistant packaging is designed to render electronics inoperable if the product is physically (invasively) attacked. Tamper evident packaging can also be used to deter tampering attacks.

See also, Zeroization, and Tamper Detection.

Timing Analysis

Timing analysis uses detectable data dependent variations in the time to perform calculations to determine secrets contained in the data. The time may be detected by monitoring external signals, unintended side channel leakage, network response time, cache hits, or other means. To prevent timing analysis constant time forms of common cryptographic algorithms are used.

V

Volatile

As applied to memory technology, volatile memory loses its data when power is removed. SRAM and DRAM technologies are volatile, while flash, EEPROM, and fuse-type memories are nonvolatile. The inability of an SRAM-based FPGA to maintain its configuration when power is removed is a function of the volatile memory technology upon which it is based. Thus, SRAM-based FPGAs require additional external nonvolatile memory components, and the sensitive data must be securely transported from the external device to the FPGA at each power-up cycle.

Z

Zeroization

Active zeroization is used to erase critical information, followed by verification that the erase operation was successful. It can be used as one of many possible responses to a tamper detection alarm. See also, Tamper Detection.

Passive zeroization is erasure of volatile memory by removal of the power source. Verification may be infeasible in this case.

9. 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 9-1. Revision History

Revision	Date	Description
H	03/2025	Updated information about digests and digest checks in the Digests, Power-On Reset Digest Check, On-Demand Digest Check and Exporting Digests sections.
G	09/2024	Updated a sentence regarding the user code in Secure Boot (For PolarFire SoC FPGA Only).
F	04/2024	Updated the information about Zeriozation in Security Glossary.
E	05/2023	The following is a summary of the changes made in this revision: • Updated the document title and added RT PolarFire® support. • Updated Secure Non-Volatile Memory (sNVM) to mention that Authenticated plaintext and Authenticated ciphertext options are available in all PolarFire family “S” version devices. • Added a note in Secure Boot (For PolarFire SoC FPGA Only) to mention that Secure Boot is available on all PolarFire SoC devices including “S” and “non-S” versions. • Updated information about the DIGEST flags in Tamper Detection Flags. • Updated information about Key Size support for RSA and DSA. See Table 7-2. • Updated information about Port List.
D	11/2022	The following is a summary of the changes made in this revision: • Added a note about sNVM wrtie condition when there is a power cut or brownout during sNVM write. See Secure Non-Volatile Memory (sNVM) • Added information about PFSOC_TAMPER macro. See Tamper Detection and Tamper Responses and Power-On Reset Digest Check • Added information about Clock Frequency Monitor
C	09/2022	Added TVS synchronization recommendation, see Temperature and Voltage Sensor
B	08/2022	The following is a summary of the changes made in this revision. • Information about Tamper Detection and Tamper Responses was updated • Information about Reset Reason was added • Information about Zeroization was updated
A	08/2021	The first publication of the document. This user guide was created by merging the following documents: • UG0753: PolarFire FPGA Security User Guide • UG0918: PolarFire SoC FPGA Security User Guide

The following revision history table describes the changes that were implemented in the UG0753: PolarFire FPGA Security User Guide document. The changes are listed by revision.

Note: UG0753: PolarFire FPGA Security User Guide document is now obsolete and the information in the document has been migrated to PolarFire® FPGA and PolarFire SoC FPGA Security User Guide.

Table 9-2. Revision History of UG0753: PolarFire FPGA Security User Guide

Revision	Date	Description
Revision 7.0	5/21	The following is a summary of the changes made in this revision. • Information about sNVM Master Key (SMK) was updated. • Information about Secure Non-Volatile Memory (sNVM) was updated. • Information about sNVM Write Service was updated. • Information about SYSCTRL_STATUS register was added. See Table 26.
Revision 6.0	9/20	The following is a summary of the changes made in this revision. • Information about Secure Non-Volatile Memory (sNVM) was updated. • Information about key modes based on factory ECC key were added. See Factory ECC Key. • Information about User ECC Keys (KUP and KUPE) was updated. • Information about RESET_DEVICE port was updated. See Table 10. • Information about Tamper Detection Flags was updated. • Information about IO Disable was updated. • Information about CYCLECOUNT parameter was updated. See Table 25. • Information about sNVM Write Service was updated. • Information about Digest Check Service was updated.
Revision 5.0	8/19	The following is a summary of the changes made in this revision. • Temperature channel's output value for the given example was corrected. See Temperature and Voltage Sensor. • Information about TEMP_HIGH port was updated. See Table 9. • Information about Returned Digests Format was updated. See Table 21.
Revision 4.0	10/18	The following is a summary of the changes made in this revision. • This document was updated for Libero SoC PolarFire v2.3 release. • Information about TVS was updated. See Temperature and Voltage Sensor. • Information about Glitch Detector was removed.
Revision 3.0	3/18	Structural changes were made throughout the document.
Revision 2.0	2/18	The following is a summary of the changes made in this revision. • Updated security overview information. For more information, see PolarFire FPGA Security Architecture. • Added tamper macro and configurator images. For more information, see Figure 25, Figure 26, and Figure 27. • Updated Table 12 by adding tamper detection flags. For more information, see Table 12. • Added a note about disabling Transceiver I/Os. For more information, see IO Disable. • Added voltage detector information. For more information, see User Voltage Detectors. • Updated tamper and voltage sensor information. See Temperature and Voltage Sensor.
Revision 1.0	7/17	The first publication of UG0753: PolarFire FPGA Security User Guide.

The following revision history table describes the changes that were implemented in the UG0918: PolarFire SoC FPGA Security User Guide document. The changes are listed by revision.

Note: UG0918: PolarFire FPGA Security User Guide document is now obsolete and the information in the document has been migrated to PolarFire® FPGA and PolarFire SoC FPGA Security User Guide.

Table 9-3. Revision History of UG0918: PolarFire SoC FPGA Security User Guide

Revision	Date	Description
Revision 3.0	5/21	The following is a summary of the changes made in this revision • Information about sNVM Master Key (SMK) was updated. • Information about Secure Non-Volatile Memory (sNVM) was updated.
Revision 2.0	9/20	The following is a summary of the changes made in this revision. • Information about Secure Non-Volatile Memory (sNVM) was updated. • Information about key modes based on factory ECC key were added. See Factory ECC Key. • Information about User ECC Keys (KUP and KUPE) was updated. • Information about RESET_DEVICE port was updated. See Table 10. • Information about Tamper Detection Flags was updated. • Information about IO Disable was updated.
Revision 1.0	6/20	The first publication of UG0918: PolarFire SoC FPGA Security User Guide.

Microchip FPGA Support

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

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

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

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

Microchip Information

Trademarks

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

ISBN: 979-8-3371-0895-7

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.