Lattice Sentry 4.0 PFR/DC-SCM Single Chip Platform RoT User Guide

Reference Design

FPGA-RD-02327-1.0

December 2025

Disclaimers

Lattice makes no warranty, representation, or guarantee regarding the accuracy of information contained in this document or the suitability of its products for any particular purpose. All information herein is provided AS IS, with all faults, and all associated risk is the responsibility entirely of the Buyer. The information provided herein is for informational purposes only and may contain technical inaccuracies or omissions, and may be otherwise rendered inaccurate for many reasons, and Lattice assumes no obligation to update or otherwise correct or revise this information. Products sold by Lattice have been subject to limited testing and it is the Buyer's responsibility to independently determine the suitability of any products and to test and verify the same. LATTICE PRODUCTS AND SERVICES ARE NOT DESIGNED, MANUFACTURED, OR TESTED FOR USE IN LIFE OR SAFETY CRITICAL SYSTEMS, HAZARDOUS ENVIRONMENTS, OR ANY OTHER ENVIRONMENTS REQUIRING FAIL-SAFE PERFORMANCE, INCLUDING ANY APPLICATION IN WHICH THE FAILURE OF THE PRODUCT OR SERVICE COULD LEAD TO DEATH, PERSONAL INJURY, SEVERE PROPERTY DAMAGE OR ENVIRONMENTAL HARM (COLLECTIVELY, "HIGH-RISK USES"). FURTHER, BUYER MUST TAKE PRUDENT STEPS TO PROTECT AGAINST PRODUCT AND SERVICE FAILURES, INCLUDING PROVIDING APPROPRIATE REDUNDANCIES, FAIL-SAFE FEATURES, AND/OR SHUT-DOWN MECHANISMS. LATTICE EXPRESSLY DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY OF FITNESS OF THE PRODUCTS OR SERVICES FOR HIGH-RISK USES. The information provided in this document is proprietary to Lattice Semiconductor, and Lattice reserves the right to make any changes to the information in this document or to any products at any time without notice.

Inclusive Language

This document was created consistent with Lattice Semiconductor’s inclusive language policy. In some cases, the language in underlying tools and other items may not yet have been updated. Please refer to Lattice’s inclusive language FAQ 6878 for a cross reference of terms. Note in some cases such as register names and state names it has been necessary to continue to utilize older terminology for compatibility.

Contents

Figures

Tables

Abbreviations in This Document

A list of abbreviations used in this document.

Abbreviation	Definition
AES	Advanced Encryption Standard
AES-CBC	Advanced Encryption Standard – Cipher Block Chaining
AES-GCM	Advanced Encryption Standard – Galois/Counter Mode
AHBL	Advanced High-performance Bus Lite
AMBA	Advanced Microcontroller Bus Architecture. An architecture used by the RISC-V to communicate with the peripherals.
APB	Advanced Peripheral Bus Interface
API	Application Programming Interface
BMC	Baseboard Management Controller
BSP	Board Support Package. The layer of software containing hardware-specific drivers and libraries to function in a particular hardware environment.
CDI	Compound Device Identifier
CoT	Chain of Trust
CPLD	Complex Programmable Logic Device
CPU	Central Processing Unit
CRE	Cryptographic Engine
CUA	Customer User Application
CUT	Customer User Test
DC-SCM	Data Center – Secure Control Module
DICE	Device Identifier Composition Engine
DMA	Direct Memory Access
DSPI	Dual Serial Peripheral Interface
ECC	Elliptic Curve Cryptography
ECDSA	Elliptic Curve Digital Signature Algorithm
ECIES	Elliptic Curve Integrated Encryption Scheme
ESFB	Embedded Security Function Block
eSPI	Enhanced Serial Peripheral Interface
FCH	Firmware Controller Hub
FIFO	First In, First Out
FPGA	Field Programmable Gate Array
FSD	Firmware Security Descriptor
FTDI	Future Technology Devices International
FW	Firmware
GPIO	General Purpose Input/Output
GUI	Graphic User Interface
HAL	Hardware Abstraction Layer. A software interface to hide the detail of the hardware design and provide general services to the upper layer.
HMAC	Hash-based Message Authentication Code
HPM	Host Processor Module
HSE	Hardware Security Engine
HW	Hardware
I2C	Inter-Integrated Circuit
I3C	Improved Inter-Integrated Circuit
I/O	Input/Output
IBI	In-Band Interrupt
ID	Identification

Abbreviation	Definition
IP	Intellectual Property
ISK	Image Signing Key
IV	Initial Vector
JTAG	Joint Test Action Group
KAK	Key Authentication Key
LISP	Last Inch Security Protocol
LTPI	LVDS Tunneling Protocol & Interface
LVDS	Low Voltage Differential Signaling
MCTP	Management Component Transport Protocol
MIPI	Mobile Industry Processor Interface
MPESTI	Modular-Peripheral Sideband Tunneling Interface
MRK	Master Root Key
OCP	Open Compute Project
OEM	Original Equipment Manufacturer
OOB	Out of Band
PC	Personal Computer
PCH	Platform Controller Hub
PFR	Platform Firmware Resiliency
PLIC	Platform Level Interrupt Controller
PLD	Programmable Logic Device
PRoT	Platform Root-of-Trust
QSPI	Quad Serial Peripheral Interface
RAM	Random Access Memory
RBP	Rollback Protection
RISC-V	Reduced Instruction Set Computer – Five
RoT	Root-of-Trust
RTD	Root-of-Trust for Detection
RTL	Register Transfer Level
RTRec	Root-of-Trust for Recovery
RTU	Root-of-Trust for Update
Rx	Receiver
SCFC	Secure Configure and Flash Controller
SCM	Secure Control Module
SCPU	Secure Central Processing Unit
SDK	Software Development Kit. A set of software development tools that allows the creation of applications for software package on the Lattice embedded platform.
SGPIO	Serial General Purpose Input/Output
SHA	Secure Hash Algorithm
SMBus	System Management Bus
SoC	System on Chip
SPI	Serial Peripheral Interface
SRAM	Static Random Access Memory
SSPI	Slave Serial Peripheral Interface
TCB	Trusted Computing Base
TCI	Trusted Computing Base Component Identity
TCM	Tightly-Coupled Memory
Tx	Transmitter
UART	Universal Asynchronous Receiver – Transmitter

Abbreviation	Definition
UDS	Unique Device Secret
UFM	User Flash Memory

1. Introduction

1.1. Purpose

The Lattice LFMXO5-55TD device is from the Lattice MachXO5™-NX device family which supports the Lattice Sentry™ solution stacks. The LFMXO5-55TD device is a low-density FPGA device with enhanced security features and on-chip boot flash. The enhanced bitstream security and user-mode security functions enable the LFMXO5-55TD device to be used as a Root-of-Trust (RoT) hardware solution in a complex system. With the LFMXO5-55TD device, you can implement a Platform Firmware Resiliency (PFR) solution in your system, as described in the NIST SP 800-193: Platform Firmware Resiliency Guidelines.

The purpose of this document is to introduce the design methodology for implementing the Lattice Sentry PFR solution on the LFMXO5-55TD device. This solution integrates DC-SCM CPLD functionality into a single chip performing both Platform Root of Trust (PRoT) and DC-SCM CPLD functions, using the Lattice Propel toolsets. This approach can significantly reduce the design complexity.

1.2. Audience

The intended audience for this document includes embedded system designers and embedded software developers. This document assumes that you have expertise in embedded system design and FPGA technologies. You are recommended to read the NIST SP 800-193: Platform Firmware Resiliency Guidelines before reading this document.

2. Platform Firmware Resiliency System Root-of-Trust Introduction

2.1. Platform Firmware Resiliency

The NIST SP 800-193: Platform Firmware Resiliency Guidelines describes the principles of supporting platform resiliency. Based on the NIST SP 800-193 guidelines, the security guidelines are based on the following three principles:

- Protection mechanisms for ensuring that Platform Firmware code and critical data remain in a state of integrity and are protected from corruption, such as the process for ensuring the authenticity and integrity of firmware updates.
- Detection mechanisms for detecting when Platform Firmware code and critical data have been corrupted, or otherwise changed from an authorized state.
- Recovery mechanisms for restoring Platform Firmware code and critical data to a state of integrity when any such firmware code or critical data are detected to have been corrupted, or when forced to recover through an authorized mechanism. Recovery is limited to the ability to recover firmware code and critical data.

2.2. Root-of-Trust

The security mechanisms are founded in RoT. A RoT is an element that forms the basis of providing one or more security-specific functions, such as measurement, storage, reporting, recovery, verification, and update. A RoT device must be designed to always behave in the expected manner. Proper function of the device is essential to providing security-specific functions. If this device is unchecked, faulty behavior cannot be detected. A RoT is typically the first element in a Chain of Trust (CoT) and can serve as an anchor for the chain to deliver more complex functionality.

The foundational guidelines on the RoT support the subsequent guidelines for Protection, Detection, and Recovery. These guidelines are organized based on the logical component responsible for each of the security properties.

- The Root-of-Trust for Update (RTU) is responsible for authenticating firmware updates and critical data changes to support platform protection.
- The Root-of-Trust for Detection (RTD) is responsible for firmware and critical data corruption detection.
- The Root-of-Trust for Recovery (RTRec) is responsible for recovery of firmware and critical data when corruption is detected.

2.3. Lattice Root-of-Trust Mechanism

The Lattice LFMXO5-55TD device can serve as the RoT device and can provide the following services:

- Image Authentication: On system power-up or reset, the LFMXO5-55TD device holds the protected devices in reset while it authenticates their boot images in SPI flash. After each signature authentication passes, the LFMXO5-55TD device releases each reset, and those devices can boot from their authenticated SPI flash image. Image authentication can also be requested at any time through the Out of Band (OOB) communication path.
- Image Recovery: If a flash image becomes corrupted for any reason, it fails to be authenticated. The LFMXO5-55TD device can restore it to a known good state by copying from an authenticated backup image.
- SPI Flash Monitoring and Protection: The LFMXO5-55TD device can monitor multiple SPI/QSPI buses for unauthorized activity and block unauthorized accesses using external quick switches. The monitors can be configured to check for specific SPI flash commands and address ranges defined by the system designer and designate them as allowed or non-allowed transactions.
- Event Logging: The LFMXO5-55TD device logs security events, such as unauthorized flash accesses and notifies the Baseboard Management Controller (BMC).
- SMBus Filtering: The LFMXO5-55TD device can monitor a SMBus for unauthorized activity and filter the unauthorized transactions. The monitor can be configured with multiple allow or disallow filters to watch for specific commands defined by the system designer and designate them as allowed or non-allowed SMBus transactions.

2.4. System Architecture

Figure 2.1 shows the architecture of the LFMXO5-55TD device working as a RoT device. The system design consists of the Embedded Security Function Block (ESFB) module, which includes a User JTAG interface. The ESFB contains the Security CPU, Crypto services, and Secure Flash Controller. The RoT functionality adds the User RISC-V processor, UART, and SMBus Mailboxes. The Sentry PFR adds the required system level interfaces and control functionality. The PFR IP blocks available are QSPI Streamer, QSPI Monitor, I3C interface, I2C filter, LTPI, SGPIO, and GPIO functions. The user RISC-V processor is connected to a set of peripherals through the AMBA bus. Software running on the processor controls the general and PFR solution peripherals and handles all the events at runtime to perform the system functionalities.

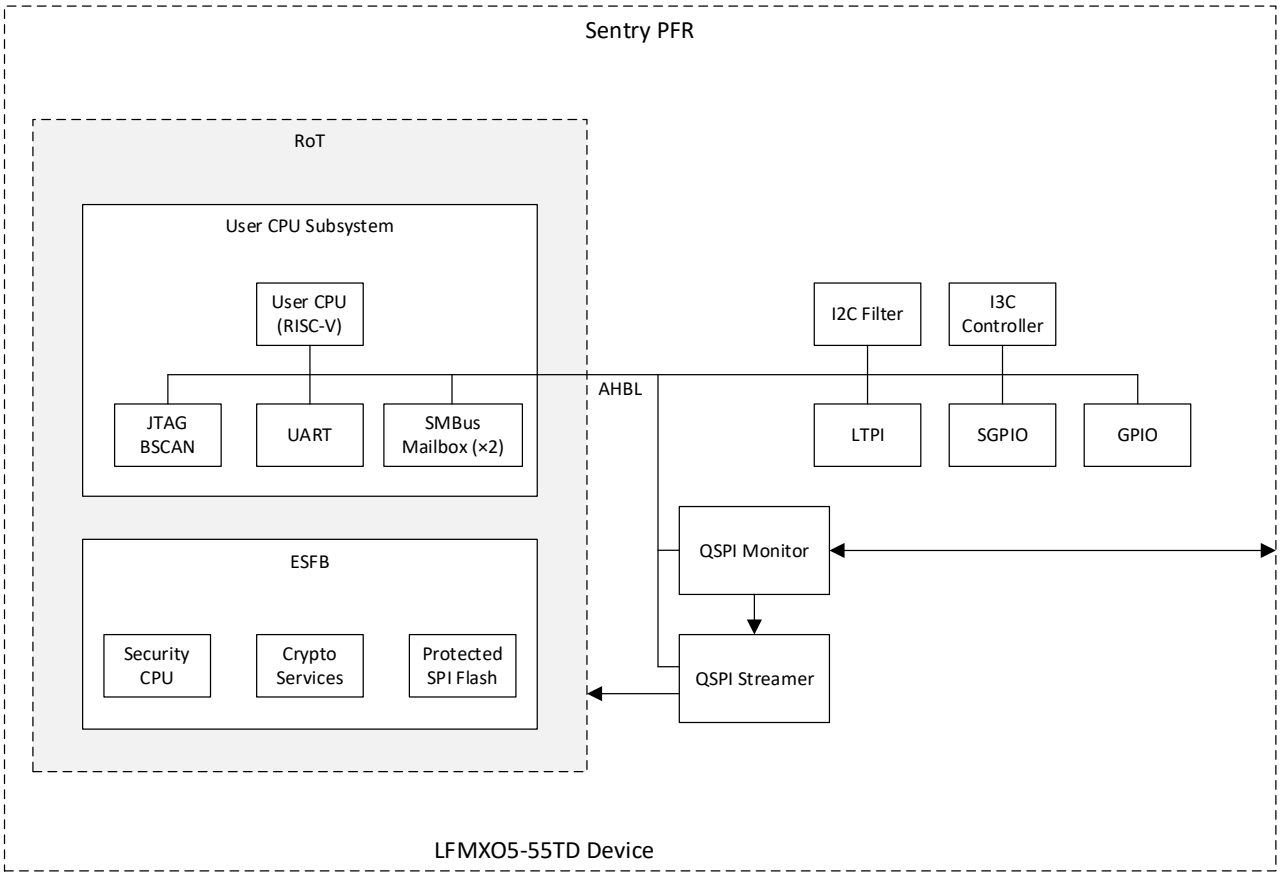

Figure 2.1. Lattice LFMXO5-55TD Device System Architecture

2.5. Functionality Overview

2.5.1. LFMXO5-55TD Device Embedded Security Function Block

The Embedded Security Function Block (ESFB) which is a hard module in the LFMXO5-55TD device is the base building block for Lattice RoT and Sentry PFR solution. It contains Security CPU, Crypto services, Secure Flash Controller, AHBL interface, FIFO DMA blocks and User JTAG interface.

2.5.1.1. Security CPU

The Security CPU (SCPU) executes code of the immutable image and provides Application Programming Interface (API) services to the User RISC-V processor. The SCPU is only accessible to user requests through API calls. The SCPU governs all access to available crypto services, access to internal SPI Flash, and internal configuration engine.

The SCPU supports the following security services for the LFMXO5-55TD device: ECC256, ECC384, ECC521, RSA-3K, RSA-4K, SHA256, SHA384, SHA512, HMAC256, HMAC384, HMAC512, ECIES, AES-CBC, and AES-GCM crypto functions.

The module has two interfaces for sending and receiving data: a register interface and a high-speed FIFO DMA interface.

Besides the security services, the SCPU has a boot loader function which performs secure boot for the immutable block.

The SCPU can also securely access the Unique Device Secret (UDS) of the LFMXO5-55TD device to generate the L0 Device Identifier Composition Engine (DICE) Certificate for DICE Attestation. DICE is an optional functionality that can be made available for the solution.

2.5.1.2. User JTAG Interface

The User JTAG interface is implemented to allow the SCPU to control the external JTAG access to the LFMXO5-55TD device. The interface elevates the security of the JTAG port to perform updates, provision or perform other JTAG accesses to the internal SPI Flash asset when user RISC-V CPU requests access through JTAG. Only a valid firmware image can request access to the JTAG interface and is allowed by the SCPU.

Refer to the Lattice Sentry 4.0 PFR IP API Reference (FPGA-TN-02377) for more details on the API reference.

2.6. LFMXO5-55TD Device Root-of-Trust Function

2.6.1. User CPU

The User RISC-V Processor provides the main RoT/PFR orchestration control function in the LFMXO5-55TD device. This processor requests services from the ESFB module. The processor integrates JTAG debugger, Platform Level Interrupt Controller (PLIC), and Timer. The RISC-V core supports RV32I instruction set and 5-stage pipelines to fulfill the performance requirement for the PFR system. JTAG debugger, PLIC, and Timer can be enabled or disabled based on the system requirement.

The User CPU also conducts the boot loader function which performs the secure boot for the whole system.

2.6.2. Tightly-Coupled Memory

The Tightly-Coupled Memory (TCM) is the memory for the User CPU instruction and data space. The image(s) for the orchestration firmware is stored in UFM (User Flash Memory) internal to the LFMXO5-55TD device. These orchestration firmware images are validated prior to being ingested into the TCM memory upon boot up.

2.6.3. SMBus Mailbox

The SMBus Mailbox provides a communication link to the System Management/BMC CPU to interface with the User CPU system. This SMBus Mailbox IP has both Controller and Target capability for Management Component Transport Protocol (MCTP). This IP also performs Mailbox functionality as per PFR specification. This block can also be used as OOB communication channel.

2.6.4. Sentry PFR Function

2.6.4.1. Lattice Sentry QSPI Streamer

Lattice Sentry QSPI Streamer is a configurable SPI controller that supports single, dual, and quad modes. It contains FIFOs for Tx and Rx data, which supports long SPI transactions. It also provides an external 8-bit Rx FIFO interface that can be connected to the ESFB for image authentication.

The QSPI Streamer incorporates a SPI FIFO Controller that provides significant performance improvement by supporting data read and write transactions of programmable length, allowing an entire SPI flash device to be read in one SPI transaction. The external Rx FIFO interface enables direct transmission of input data from the SPI target to another block, such as the ESFB, which frees up the CPU or system bus.

Refer to the Lattice Sentry 4.0 PFR IP API Reference (FPGA-TN-02377) for more details on the API reference.

2.6.4.2. Lattice Sentry QSPI Monitor

The QSPI Monitor is a configurable security module which can monitor up to three SPI, DSPI, or QSPI buses for unauthorized activity and block transactions by controlling the chip select signal and external quick switch devices. In addition to monitoring, it can connect external SPI/DSPI/QSPI buses to the QSPI Streamer through a programmable mux/demux block.

The QSPI Monitor checks the external buses for allowed flash commands and flash addresses. It provides fine-grain control over the set of allowed commands. It supports 4 kB address space size that can monitor erase, program, and read commands up to the full flash size. Address spaces are either allowed for erase or program commands or be denied for read commands. Flash addresses of 24 bits and 32 bits are supported. This block is used to monitor the SPI transactions and block unauthorized SPI transactions in real time.

Refer to the Lattice Sentry 4.0 PFR IP API Reference (FPGA-TN-02377) for more details on the API reference.

2.6.4.3. Lattice Sentry System Management Bus (I2C/SMBus) Filter

The SMBus filter is a configurable security module which can monitor traffic on the SMBus to identify unauthorized activity, based on the set of up to 256 programmable filters. If unauthorized activity is detected, the SMBus is disabled and the PFR firmware is notified so that an event can be logged.

Refer to the Lattice Sentry 4.0 PFR IP API Reference (FPGA-TN-02377) for more details on the API reference.

2.6.4.4. I3C Controller

An I3C controller can be used to communicate with the BMC (Management CPU) or PCH/FCH (Host CPU). The Lattice Sentry I3C Controller is designed to comply with the MIPI I3C specification v1.1.1. It is a configurable IP that supports the primary controller, hot-join, and IBI transactions up to 12.5 MHz using Push-Pull. It also provides 8b × 256 size receive and transmit FIFOs.

2.6.4.5. LTPI

The Lattice DC-SCM LTPI IP is an Open Computer Project (OCP) DC-SCM Standards compatible solution which is introduced in the DC-SCM 2.0 Specification. LTPI is a protocol and interface designed for tunneling various low-speed signals between Host Processor Module (HPM) and Secure Control Module (SCM). The LTPI protocol utilizes the LVDS electrical interfaces supported by majority of the CPLD and FPGA devices. This is the next generation protocol for DC-SCM 2.0 that serves as a replacement for two SGPIO interfaces. The LVDS interface provides higher bandwidth and better scalability than the SGPIO interface. It allows for tunneling of not only GPIOs but also low-speed serial interfaces such as I2C and UART. It is also extensible with additional proprietary OEM interfaces and provides support for raw Data tunneling between HPM and SCM CPLD devices. It also provides a solution for minimal wire connection between two FPGA devices.

The template design supports 16 low-latency GPIO inputs, 16 low-latency GPIO outputs, 16 normal-latency GPIO inputs, 16 normal-latency GPIO outputs, 1 UART Rx, 1 UART Tx, and 3 I2C channels through the LTPI interface.

2.6.4.6. SGPIO

The SGPIO is a peripheral IP designed to control SGPIOs through the Advanced Peripheral Bus Interface (APB). The number of SGPIO bits and SGPIO bus frequency are configurable.

2.6.4.7. GPIO

GPIO signals are also available. Typically, GPIO signals are used by the User CPU to hold or release reset signals to downstream devices, like BMC, PCH, and CPU, while the authentication process is running to validate the images that will be used by the devices to boot up.

2.6.4.8. UART

The UART IP is designed for use in serial communication, supporting RS-232. It is used to print debug messages from the Orchestration Firmware running in the User CPU.

2.7. Boot Sequence

The immutable ESFB block starts first. This ESFB block contains hardened RoT functionality. After the immutable ESFB block runs, if the Test Mode pins are set to Normal Mode, the configuration image looks for a valid Customer User Application (CUA) image. If the configuration image finds a valid CUA image, this valid CUA image completes the booting process.

The test mode pins are set through GPIO. On the Lattice Sentry 4.0 Demo Board for MachXO5D™-NX, the pins are connected to SW4.1, SW4.2, and SW4.3. The available modes are as follows:

- Mode ×11 – Normal Mode. Boots the valid CUA image.
- Mode ×10 – Customer Test Mode. Boots the valid Customer User Test (CUT) image.
- Mode 001 (SSPI) or 101 (JTAG) – Provision Flow Test Mode. Forces the provisioning flow to run and can be used for reprovisioning during development.

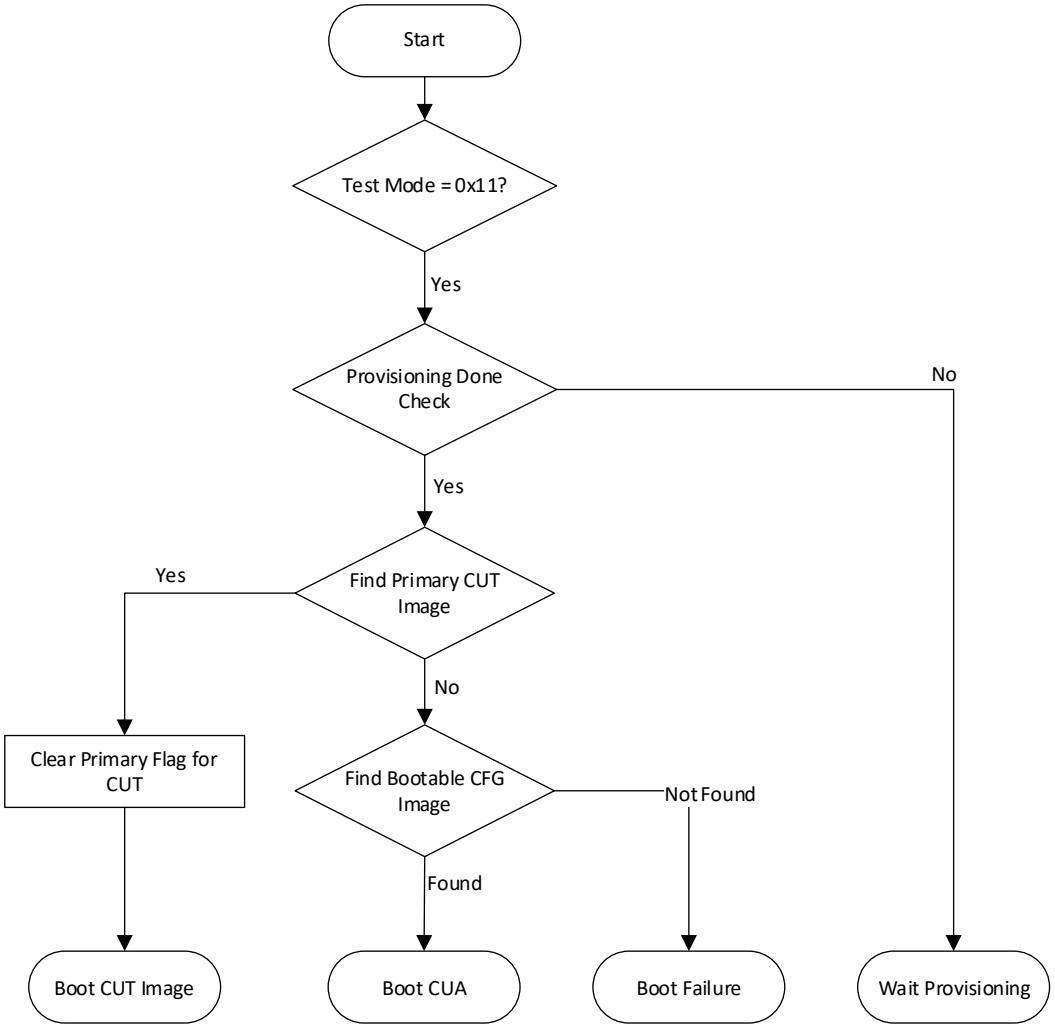

Figure 2.2. Boot Sequence Flow

2.8. Provision and Update Flow

2.8.1. Provision Flow

Device provisioning is a process to program customer security policies, keys and lock policies, initial CUA image(s), and optional UFM sectors. Provisioning is usually performed only once and the exceptions are as follows:

- For zeroization, which resets the device to factory settings. Reprovisioning step is required to make the device functional.
- For test mode 001 (SSPI Provisioning) or 101 (JTAG Provisioning), which forces the provisioning flow to run, and is used for development.

Both exceptions can be disabled in the customer security policy, so that provisioning cannot be rerun.

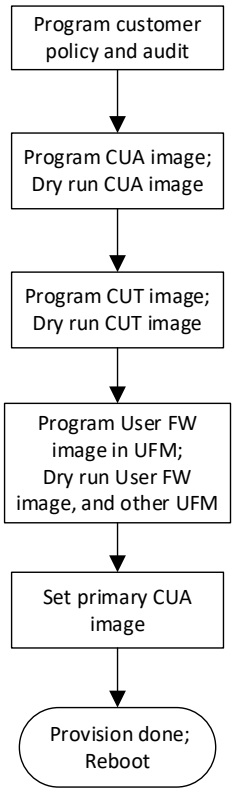

Figure 2.3. Provisioning Flow

Table 2.1 describes the image status flags.

Table 2.1. Image Status Flags

Flag	Description
Customer_policy_done	Indicates Customer security policy, keys, and locks complete
CUA_image_done	Indicates Customer User Application image complete
CUT_image_done	Indicates Customer User Test image complete
Orchestration_FW_image_done	Indicates Customer User Firmware image complete

2.8.2. Customer User Application Image Update Flow

Only the backup CUA image can be updated. The current active CUA image cannot be updated. The CUT image does not support the CUA image update flow.

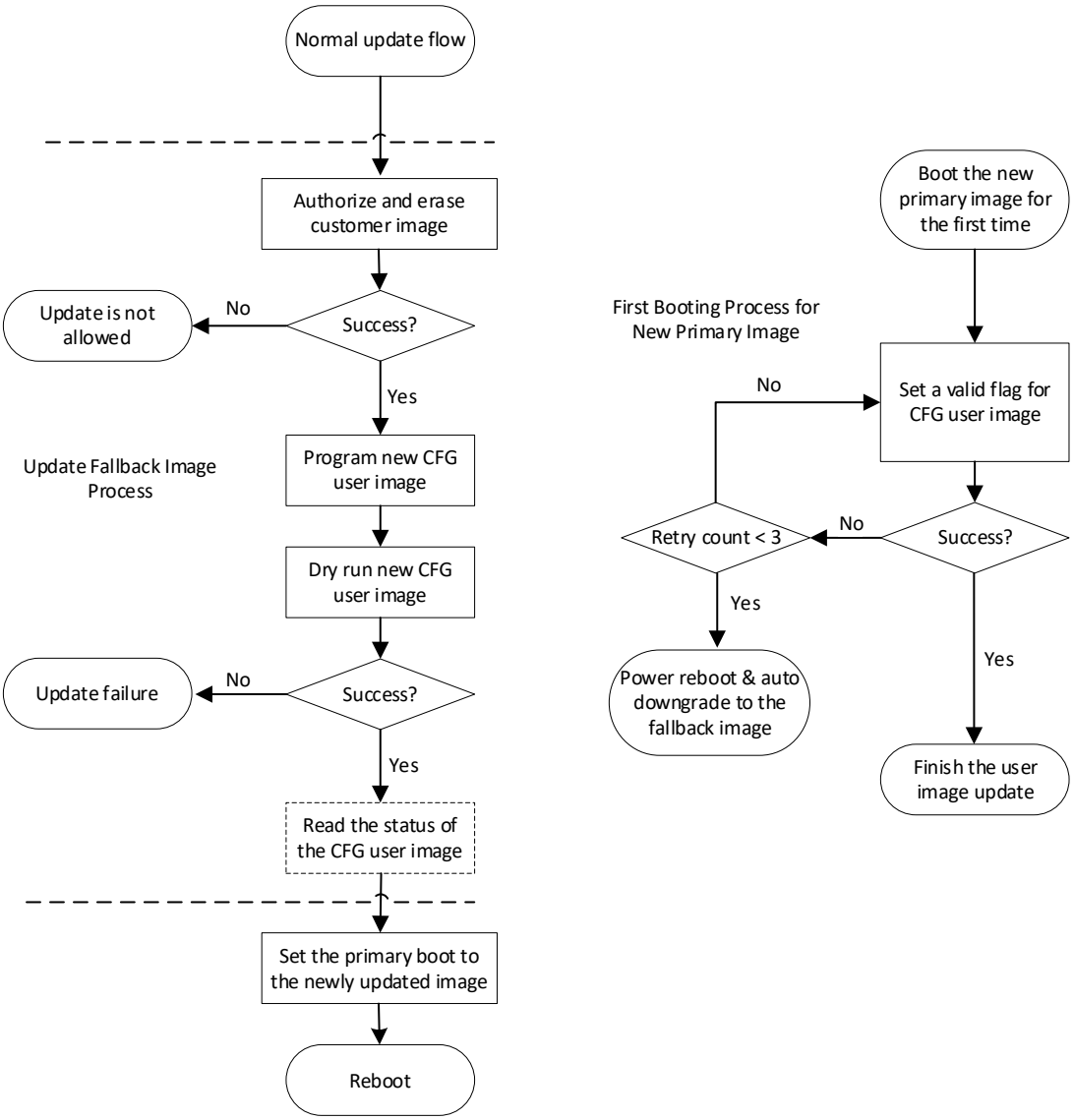

Figure 2.4. CUA Image Update Flow

3. Secure Firmware

3.1. Firmware Architecture

Secure firmware handles the requests from AHBL/JTAG register interfaces and target SPI interface for crypto services and flash assets. From the top level, the service handling module processes all the data from the AHBL/JTAG register interface and target SPI interface, then feeds into the ESFB layer accordingly. The ESFB layer is based on the driver layer and processes all the crypto service requests from the application layer. The driver layer hides all the hardware implementation detail and provides the chip-level functions.

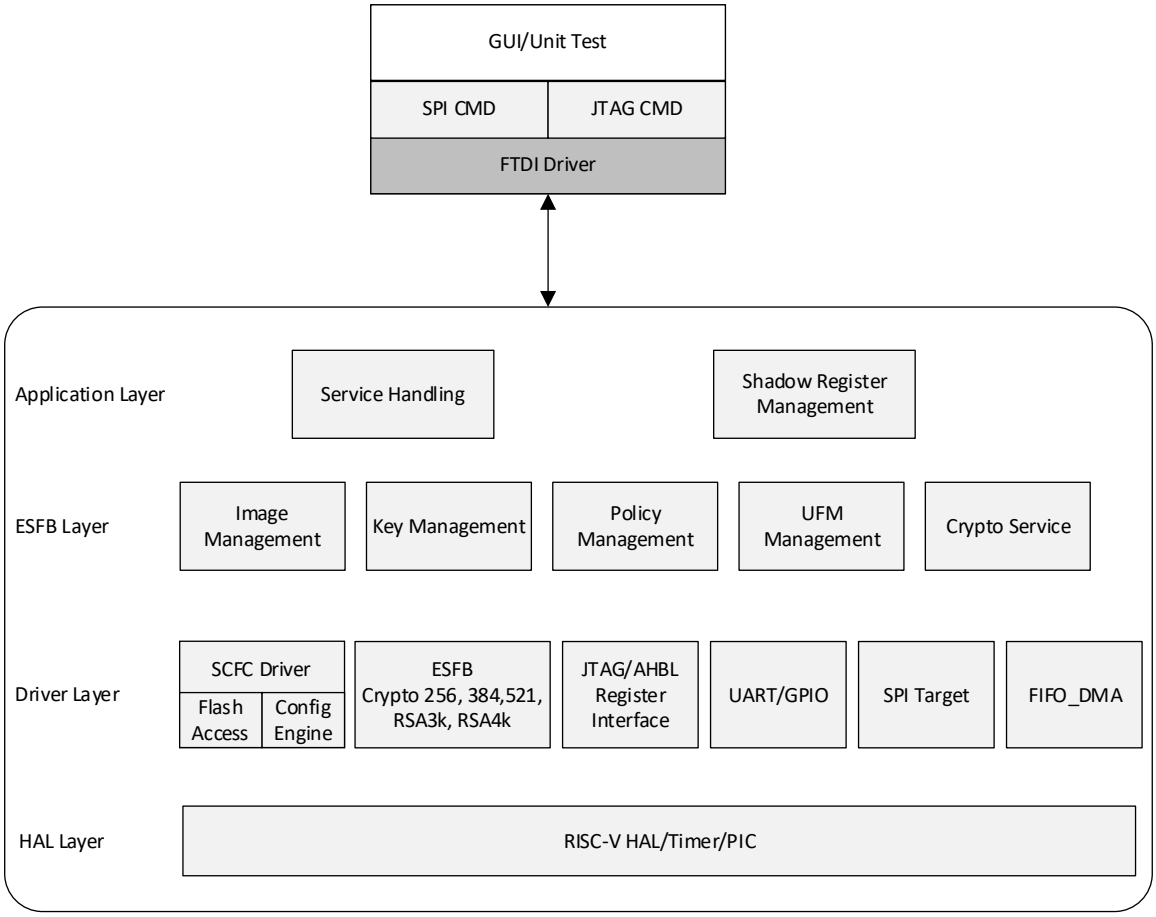

Figure 3.1. Firmware Architecture Layers

3.2. CUA Image Secure Firmware

3.2.1. CUA Image Firmware Flow

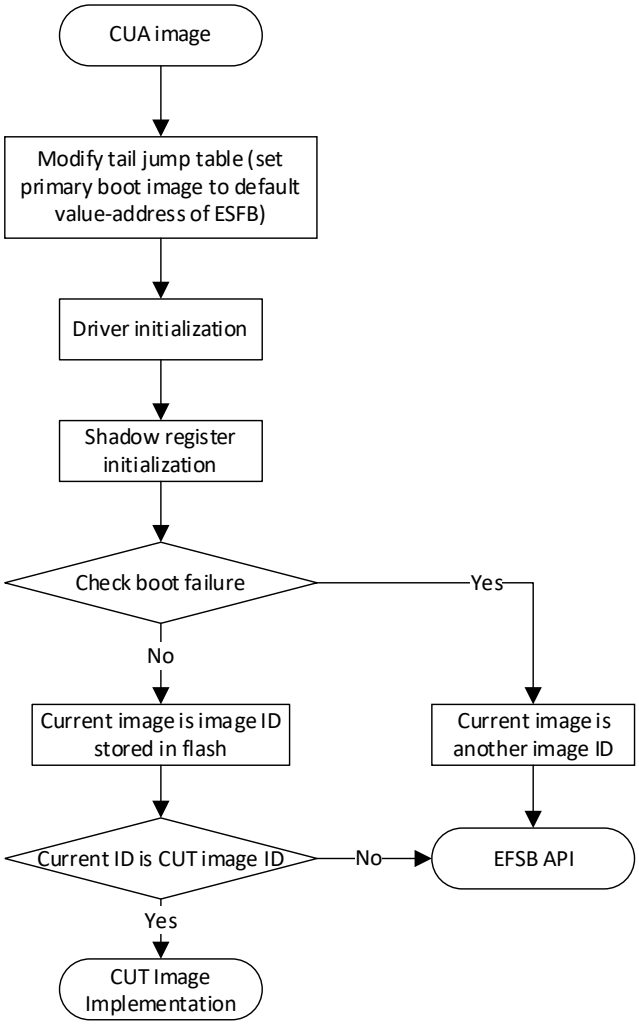

Figure 3.2. CUA Image Firmware Flow

3.2.2. Shadow Register

To minimize flash wear due to frequent flash access, some critical data is downloaded from the flash to their shadow registers during system initialization.

3.2.2.1. Shadow Register Definition

All frequently accessed flash data are implemented as part of the shadow register. The frequently access flash data includes:

- Customer policy info, including UFM partition, lock policy, and keys
- Key allow list
- Current image ID, boot image ID
- Customer image status (including CUA0, CUA1, and CUT status)
- Internal lock status (imaging, policy, UFM, JTAG, and SPI target)
- use_main_policy
- cut_primary_flag
- provision done

3.2.3. API List

User orchestration firmware calls all the ESFB APIs to get the requested services. See the Embedded Security and Function Block with Advanced Key Management for MachXO5-NX (55TD) Devices Technical Note (FPGA-TN-02353) for customer accessible APIs. The ESFB APIs implement all available services.

3.3. Policy Management

3.3.1. Overview

The LFMXO5-55TD device maintains many user assets on the internal flash. For each of the assets, a separate policy control strategy is provided to manage and protect critical data. There are two sectors for policy:

- Customer Policy Sector
- Customer Central Lock Sector

3.3.2. Customer Policy Sector

Customer policy sector stores all the *set once* settings from users. The ESFB block implements all the features and control based on these settings, including audit methods of each sector, boot policy, image 1 (CUA0) and image 2 (CUA1) information, and UFM information. The customer policy is described in Table 3.1.

Table 3.1. Security Sector Format

Block Name	Size (Byte)	Offset	Default Value	Byte Ordering	Description
Reserved	4	0	0x0000009001	Little endian	Reserved.
re-provisiontest_mode_0n1_permit	4	4	0x00000001	Little endian	Permission to perform reprovisioning through MODE10 pin.
test_cutmode_10_permit	4	8	0x00000001	Little endian	Permission to run CUT image through MODE01 pin.
image_auth_method	4	12	0x00000001	Little endian	User Image Authentication mode: 0: ecdsa256 1: ecdsa384 2: ecdsa521 3: rsa3k 4: rsa4k
KAK_num	4	16	0x00000001	Little endian	Number of Key Authentication Keys (valid range: 1–8).
Reserved	4	20	0x00000001		Reserved.
ufm aes type	48	2024	0x000000100000 00010x00000002	Little endian	AES configuration: 2: Device-generated AES + customer salt hash. 3: No encryption. Reserved.
ufm_aes_key_salt	32	28	All 0s	Big endian	User AES key salt.
zero_permit	4	60	0x00000001	—	Permission to zeroize: 0: Forbid 1: Allow
null_permit	4	64	0x00000001	—	Permission to nullify: 0: Forbid 1: Allow
images_min_version	4	68	0x00000000	Little endian	Initial minimal version for customer image.
ufm_number	4	72	0x00000008	—	Number of UFM partitions (valid range: 1–8).
ufm_size1	ufm_size[0] / 4 bytes	76	0x0000001E	Little endian	ufm0 page count.

Block Name	Size (Byte)	Offset	Default Value	Byte Ordering	Description
	ufm_size[1] / 4 bytes	80	0x0000001E	Little endian	ufm1 page count, if the ufm number enables this sector.
	ufm_size[2] / 4 bytes	84	0x0000001E	Little endian	ufm2 page count, if the ufm number enables this sector.
	ufm_size[3] / 4 bytes	88	0x0000001E	Little endian	ufm3 page count, if the ufm number enables this sector.
	ufm_size[4] / 4 bytes	92	0x0000001E	Little endian	ufm4 page count, if the ufm number enables this sector.
	ufm_size[5] / 4 bytes	96	0x0000001E	Little endian	ufm5 page count, if the ufm number enables this sector.
	ufm_size[6] / 4 bytes	100	0x0000001E	Little endian	ufm6 page count, if the ufm number enables this sector.
	ufm_size[7] / 4 bytes	104	0x0000001E	Little endian	ufm7 page count, if the ufm number enables this sector.
ufm_start_addr2	ufm_start_addr[0] / 4 bytes	108	0x00f00000	Little endian	ufm0 start address (in bytes, 256-byte alignment) must be equal or greater than 0x2E0000.
	ufm_start_addr[1] / 4 bytes	112	0x00f1e000	Little endian	ufm1 start address, if the ufm number enables this sector.
	ufm_start_addr[2] / 4 bytes	116	0x00f3c000	Little endian	ufm2 start address, if the ufm number enables this sector.
	ufm_start_addr[3] / 4 bytes	120	0x00f5a000	Little endian	ufm3 start address, if the ufm number enables this sector.
	ufm_start_addr[4] / 4 bytes	124	0x00f78000	Little endian	ufm4 start address, if the ufm number enables this sector.
	ufm_start_addr[5] / 4 bytes	128	0x00f96000	Little endian	ufm5 start address, if the ufm number enables this sector.
	ufm_start_addr[6] / 4 bytes	132	0x00fb4000	Little endian	ufm6 start address, if the ufm number enables this sector.
	ufm_start_addr[7] / 4 bytes	136	0x00fd2000	Little endian	ufm7 start address, if the ufm number enables this sector.
ufm_lock	ufm_lock[0] / 4 bytes	140	0x00000000	Little endian	Lock status: 0: No lock 1: Soft lock 2: Hard lock
	ufm_lock[1] / 4 bytes	144	0x00000000	Little endian	Lock status: 0: No lock 1: Soft lock 2: Hard lock
	ufm_lock[2] / 4 bytes	148	0x00000000	Little endian	Lock status: 0: No lock 1: Soft lock 2: Hard lock
	ufm_lock[3] / 4 bytes	152	0x00000000	Little endian	Lock status: 0: No lock 1: Soft lock 2: Hard lock
	ufm_lock[4] / 4 bytes	156	0x00000000	Little endian	Lock status: 0: No lock 1: Soft lock 2: Hard lock
	ufm_lock[5] / 4 bytes	160	0x00000000	Little endian	Lock status: 0: No lock

Block Name	Size (Byte)	Offset	Default Value	Byte Ordering	Description
					1: Soft lock 2: Hard lock
	ufm_lock[6] / 4 bytes	164	0x00000000	Little endian	Lock status: 0: No lock 1: Soft lock 2: Hard lock
	ufm_lock[7] / 4 bytes	168	0x00000000	Little endian	Lock status: 0: No lock 1: Soft lock 2: Hard lock
jtag_port_lock	4	172	0x00000000	Little endian	Lock status: 0: No lock 1: Soft lock 2: Hard lock
CFG0_hard_lock	4	176	0x00000000	Little endian	Lock status: 0: No lock 1: Soft lock 2: Hard lock
customer_policy_lock	4	180	0x00000000	Little endian	Lock status: 0: No lock 1: Soft lock 2: Hard lock
Reserved	4	184	0x00000000	Little endian	Reserved.
mrk_key_hash	64	188	All 0s	Big endian	SHA-512 of the MRK key.
kak_key_hash	kak_key_hash[0] / 64 bytes	252	All 0s	Big endian	SHA-512 of the KAK key.
	kak_key_hash[1] / 64 bytes	316	All 0s	Big endian	
	kak_key_hash[2] / 64 bytes	380	All 0s	Big endian	
	kak_key_hash[3] / 64 bytes	444	All 0s	Big endian	
	kak_key_hash[4] / 64 bytes	508	All 0s	Big endian	
	kak_key_hash[5] / 64 bytes	572	All 0s	Big endian	
	kak_key_hash[6] / 64 bytes	636	All 0s	Big endian	
	kak_key_hash[7] / 64 bytes	700	All 0s	Big endian	
cu_image_aes	32	764	All 0s	Big endian	User AES key for bitstream encryption.
Reserved	64	796	Varies	—	Reserved.
encrypt_image_only	4	860	0x00000000	Little endian	Turning on this option forces the device to accept encrypted bitstream only.
sentry_flag	4	864	0x00000000	Little endian	Enables storage for user firmware.
sentry_fw_min_version	4	868	0x00000000	Little endian	Sets minimum user firmware version.
Sha512_hash	64	872	—	Big endian	SHA-512 hash of security sector.

3.4. Key Management

3.4.1. Overview

The LFMXO5-55TD device supports 1 MRK + 8 KAK + 2,048 ISK. More information about these keys can be found in the ESFB and Key Management document for the LFMXO5-55TD device. MRK is used to perform nullification and zeroization. The Hash 512 values of MRK and Key Authentication Key (KAK) are stored in customer policy. MRK cannot be canceled. KAK and ISK keys can be canceled by key revoke API. The key status is recorded in the customer key allow list. The customer key allow list has redundant A/B copies.

3.4.2. Flow of Keyblob

There are four types of keyblob: Normal, ISK Revoke, KAK Revoke, and MRK. The first three keyblobs are signed by KAK private key. KAK public key is used to verify keyblob. The customer images are signed by ISK private key and verified by ISK public key. The Normal keyblob is used to update customer image or customer policy.

MRK keyblob is used for the following items:

- Policies update
- Nullification
- Zeroization

Keyblobs can be generated using the Lattice Radiant Bitstream Security Settings GUI. You can also use your own tools to generate and manage keyblobs, as long as the keyblobs follow the format specified in Table 3.2. Different types of keyblob flows are described in the following sections.

Table 3.2. Keyblob Types and Contents

Type	Field	Size (Bytes)	Description
Normal	Auth Method	4	0: ECDSA256 1: ECDSA384 2: ECDSA521 3: RSA3K 4: RSA4K
	Type = 1 Normal	4	Normal CUA image update Keyblob
	SALT	64	Random
	KAK Pb Key	512	KAK Public Key value
	ISK ID	4	Image Signing Key ID (0–255)
	ISK Pb Key	512	ISK Public Key value
	KAK signature	512	Signature of Keyblob uses KAK private key
ISK Revoke	Auth Method	4	0: ECDSA256 1: ECDSA384 2: ECDSA521 3: RSA3K 4: RSA4K
	Type = 2 Revoke ISK	4	ISK Revocation Keyblob
	SALT	64	Random
	KAK pb Key	512	KAK Public Key value
	Revoke ISK ID	4	Image Signing Key ID (0–255) being revoked
	ISK ID	4	New Image Signing Key (0–255) that was used for signing the current bitstream
	ISK Pb Key	512	New ISK Public Key value
	KAK signature	512	Signature of Keyblob uses KAK private key
KAK Revoke	Auth Method	4	0: ECDSA256 1: ECDSA384 2: ECDSA521 3: RSA3K 4: RSA4K

Type	Field	Size (Bytes)	Description
	Type = 3 KAK Revoke	4	KAK Revocation Keyblob
	SALT	64	Random
	Revoke KAK Pb Key	512	KAK value that is being revoked
	KAK Pb Key	512	New KAK Public Key value
	KAK signature	512	Signature of Keyblob uses new KAK Private Key
MRK Operation	Auth Method	4	0: ECDSA256 1: ECDSA384 2: ECDSA521 3: RSA3K 4: RSA4K
	Type = 4 MRK	4	Used for nullification/zeroization/policy re-write
	SALT	64	Random
	MRK Pb Key	512	MRK Public Key value
	MRK signature	512	Signature of Keyblob uses MRK private key

3.4.3. Normal Keyblob Flow

During image configuration or flash programming process, the keyblob that resides in the bitstream header is audited first by the ESFB before image booting or flash programming is allowed to be executed. Figure 3.3 shows the Normal keyblob auditing process by the ESFB.

The process involves the following three stages:

1. Authenticate the hash of the KAK public key in the keyblob with the hash of the KAK public key that is pre-provisioned in the key sector.
2. Authenticate the keyblob using the KAK public key and signature.
3. Verify whether the ISK in the keyblob is put on the allow list.

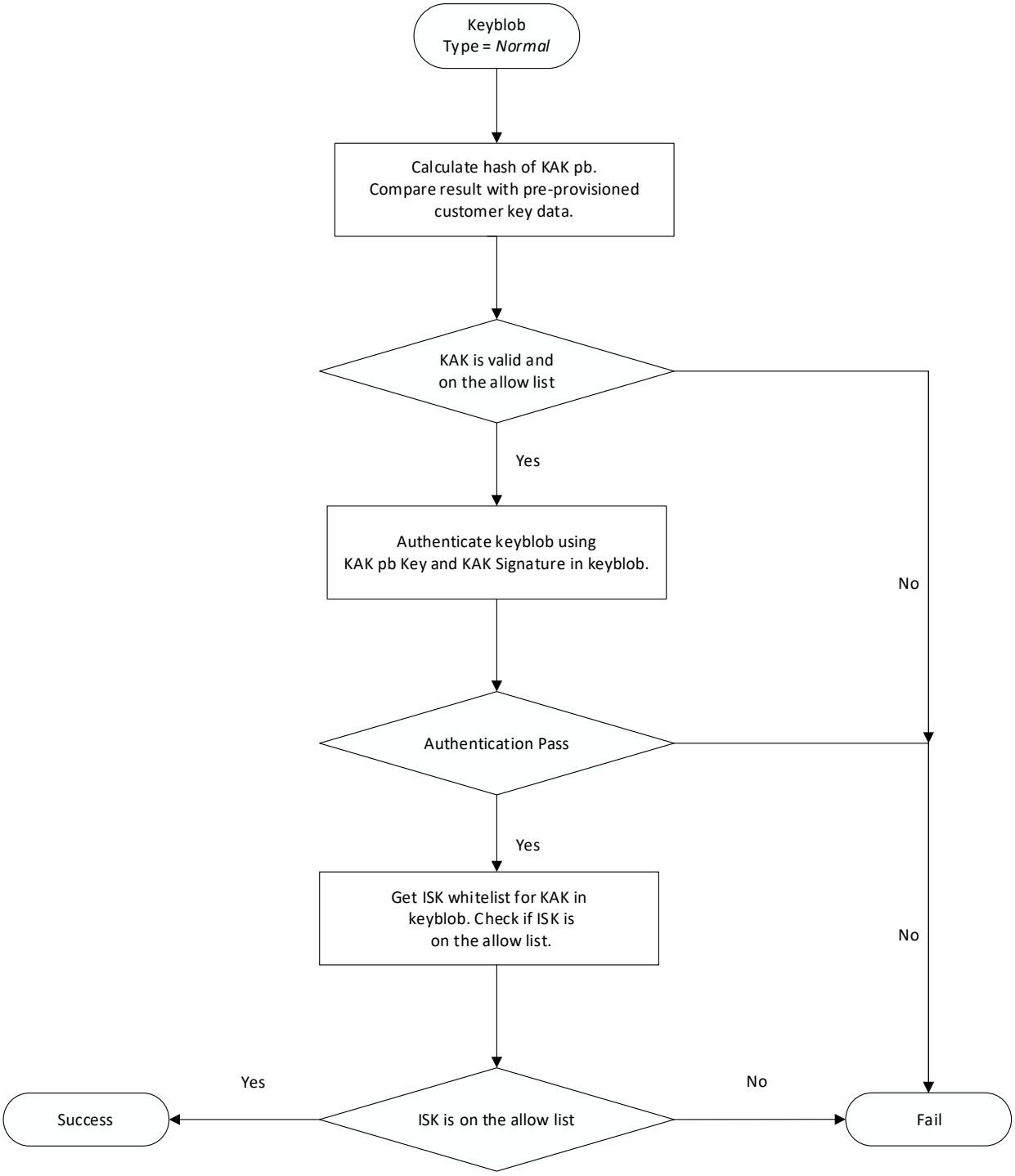

Figure 3.3. Normal Type Keyblob Flow

3.4.4. ISK Revoke Flow

When performing revocation, ensure the *make before break* concept. This means you must ensure that the new design with the new ISK ID works correctly before revoking the old ISK ID. Hence, there are two types of keyblob involved, which are Normal keyblob and ISK Revoke keyblob.

The purpose of the Normal keyblob is to perform initial assessment of the new image with the new ISK ID. Once the design and new ISK ID work correctly, the ISK Revoke keyblob is used to revoke the old ISK ID and update it to the new one.

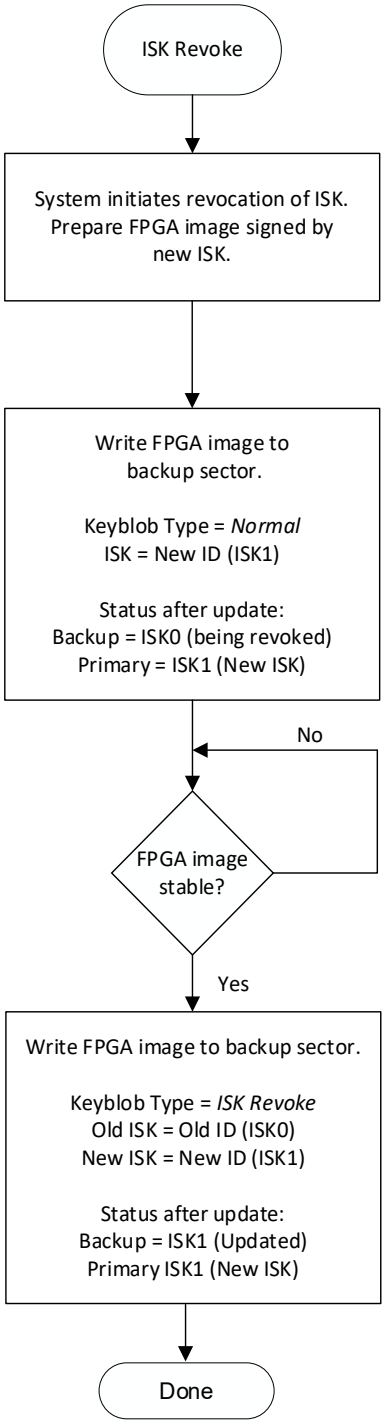

Figure 3.4. ISK Revoke Keyblob Flow

3.4.5. KAK Revoke Flow

In addition to ISK revoke, KAK can also be revoked up to seven times. Unlike performing revoke through the bitstream header like ISK Revoke, KAK revocation is done through a data packet using the KAK Revoke keyblob type. In other words, KAK can be revoked by calling the KAK revoke API through the ESFB interface.

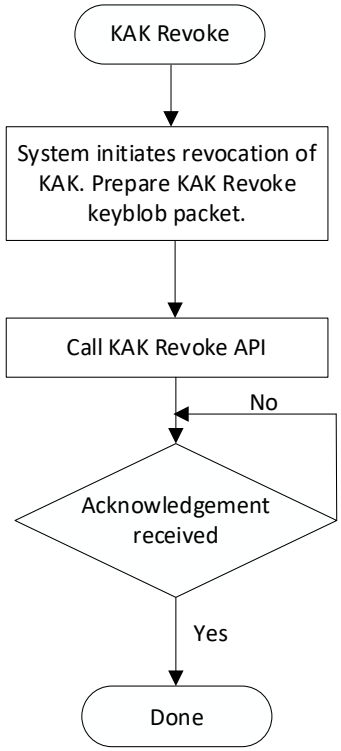

Figure 3.5. KAK Revocation Flow

3.4.6. Zeroization Flow

Zeroization allows you to set the device to a zero state so that the device can be recommissioned.

Zeroization is a security feature used to erase sensitive data stored within the device to prevent unauthorized access or data recovery. This process involves altering or deleting the contents of the data storage, effectively rendering the data irretrievable. Zeroization is crucial in applications where high-value assets are at risk, such as in military, financial, and secure communication systems.

To perform zeroization, the procedure must satisfy the following conditions:

- The MRK type of keyblob is used with the authorize_and_perform_zeroization API. Refer to Table 3.2 for the keyblob type and format.
- Zeroization is permitted in the device policy.

After the zeroization operation, the flash user data, device policy, keys, and locks are all cleared. You must perform provisioning again to set up the device with the new policy and keys.

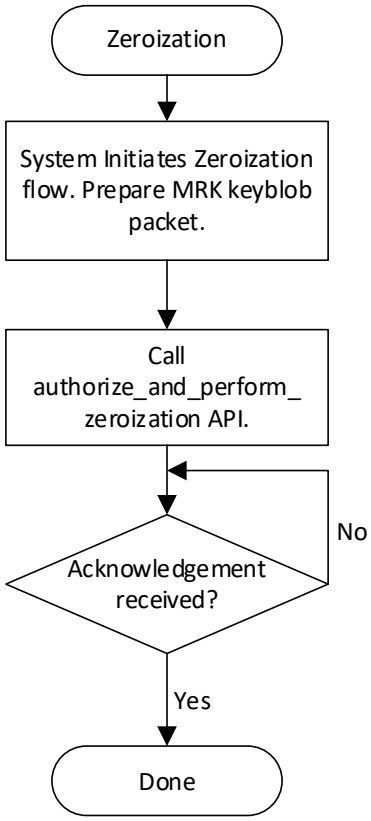

Figure 3.6. Zeroization Flow

3.4.7. Nullification Flow

Nullification allows you to erase all policies, keys, locks, images, or data and mark a device as invalid for future deployment.

The goal of nullification serves two main usage scenarios:

- Anti-tampering: Depending on your control logic and the nature of tampering events, you may perform nullification to destroy all data and render the device inoperable.
- Failure return to Lattice: You may perform nullification if you want the particular device to stay out of service.

To perform nullification, the procedure must satisfy the following conditions:

- MRK type of keyblob is used with the authorize_and_perform_nullification API. Refer to Table 3.2 for the keyblob type and format.
- Nullification is permitted in the device policy.

After the nullification operation, all flash user data is erased. The device policy, keys, and locks are all marked as *all 1's* to indicate that the device is *nullified*. Devices that complete the nullification procedure successfully are not allowed to return to service.

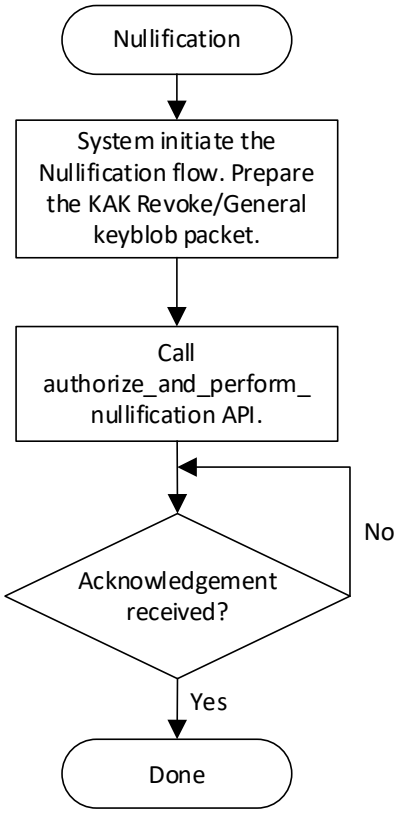

Figure 3.7. Nullification Flow

3.4.8. API List

Three categories of APIs are provided in the key management component, Key Map processing, and Keyblob processing. Refer to the Lattice Sentry 4.0 PFR IP API Reference (FPGA-TN-02377) for more information on the key-related APIs.

3.5. Image Management

The factory image and customer image update and audit flow are implemented in the Image Management component.

3.5.1. Customer Design Image Format

Customer user image, for both CUA and CUT, have the same format as shown in Table 3.3.

Table 3.3. Customer Image Format

Section	Name	Offset	Size (Bytes)	Value and Description	From
0	Magic Number for CU IMG	0x0000	4	0x683BA9FC – Initial release with following format, no XMSS/LMS support. Once CU format needs change, a new Magic number is needed.	Fixed value
1	Customer Signature	0x0004	512	Signature of FPGA Bitstream signed by customer ISK described in Keyblob Section (due to the length limit).	Customer sign using ISK (in Keyblob), depends on Sections 3–7.
2	Reserved	—	—	Reserved (write as 0)	—
3	Customer IMG Version Number	0x0400	4	Version for rollback protection, use *date stamp* as initial version number.	Customer may change this sector and re-sign the image.
4	Customer Key Blob	0x0404	Up to 1,616	Refer to the Flow of Keyblob section.	Customer
5	Reserved	—	—	Reserved (write as 0)	—
6	Lattice ESFB Manifest	0x1000	8,192	See Manifest Format	Lattice BitGen
7	FPGA Bitstream	0x3000	Varies	ECDSA + AES-CBC (Customer AES key)	Lattice BitGen

3.5.1.1. Customer Image Dry Run

Table 3.4 shows the image internal status. Clear the status when image is erased.

Table 3.4. Customer Image Status Flag Description

Status	Description
Done status	1: Dry run success 2: Dry-run fail
Cancel status	1: image is cancelled by ISK revoke/second dry-run fail
Corrupt count	Increase by one when detecting corrupt info
First boot	Set this flag when image is erased; clear this flag before booting this image
Valid status	Set this flag by calling the *set_customer_image_valid()* API

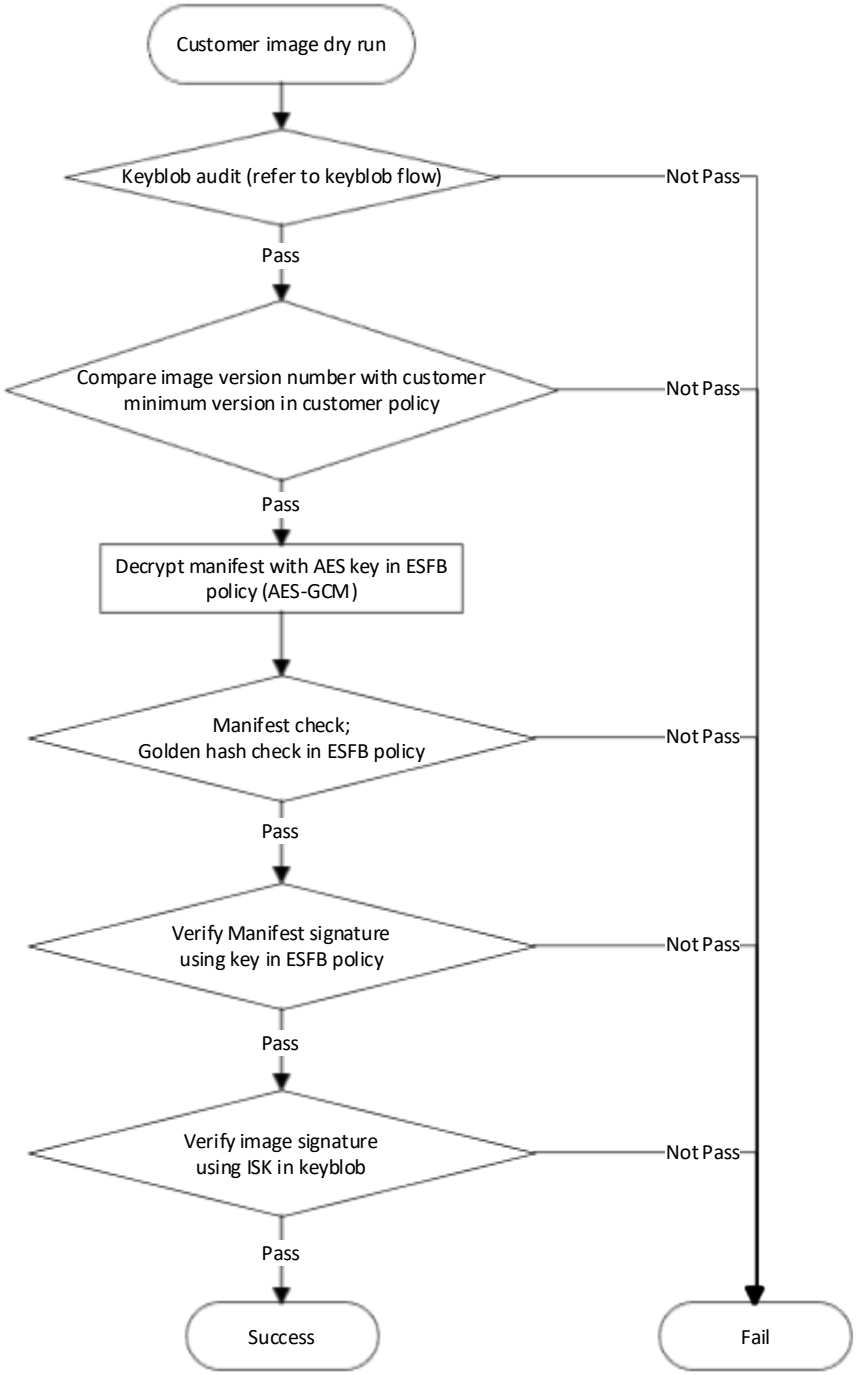

Figure 3.8. Customer Image Dry Run Flow

3.5.1.2. Customer Image Erase

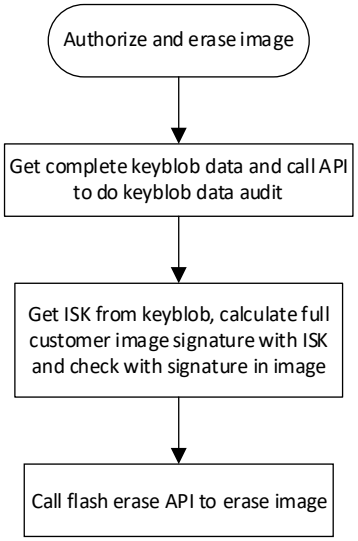

Figure 3.9. Customer Image Erase Flow

3.6. UFM Management

UFM management provides all the UFM access functions and the UFM configuration. UFM data is encrypted when writing to flash and is decrypted when reading from flash.

The UFM is encrypted with AES-GCM. Each sector is 4k bytes and uses an initial vector (IV). The first page of a sector is reserved to store IV and tag. Table 3.5 shows the AES-GCM input value sizes.

Table 3.5. AES-GCM Input Value Size

Input	Size (Bytes)
Initial Vector	16
Tag	16
Data Length	4

3.7. Other APIs

3.7.1. User Message Audit

Table 3.6. Update Image Format

Section	Type	Size (Bytes)	Contents
1	Signature	96	Signature, which includes Sections 2–4 and uses ISK public key corresponding private key to sign.
2	Keyblob	360	Normal keyblob. See Figure 3.3 for more details.
3	Payload length	4	Payload data length.
4	Payload	0–1,300	Payload data.

4. User Orchestration Firmware

4.1. Overview

User orchestration firmware runs on the user CPU. It can access the LFMXO5-55TD device through the register interface. User orchestration firmware is built based on the ESFB APIs and other IPs. The ESFB APIs access the Lattice Secure Subsystem for service through the AHBL register interfaces. User CPU can create OOB connection (I2C/JTAG) with Demo GUI, receive OOB commands and execute ESFB APIs for image update, key management, UFM management, and crypto services.

4.2. Firmware Architecture

The LFMXO5-55TD device firmware includes two parts:

- Lattice ESFB, which functions as a boot loader and configuration engine;
- User orchestration firmware, which builds based on the ESFB block and other soft IPs to implement the customer specific functions.

For each firmware, a hardware abstraction layer (HAL) at the bottom provides the platform dependent services. Based on the HAL, the driver layer provides a set of APIs to encapsulate function of each IP. The component layer provides all the higher-level services, and the application layer sits on the top.

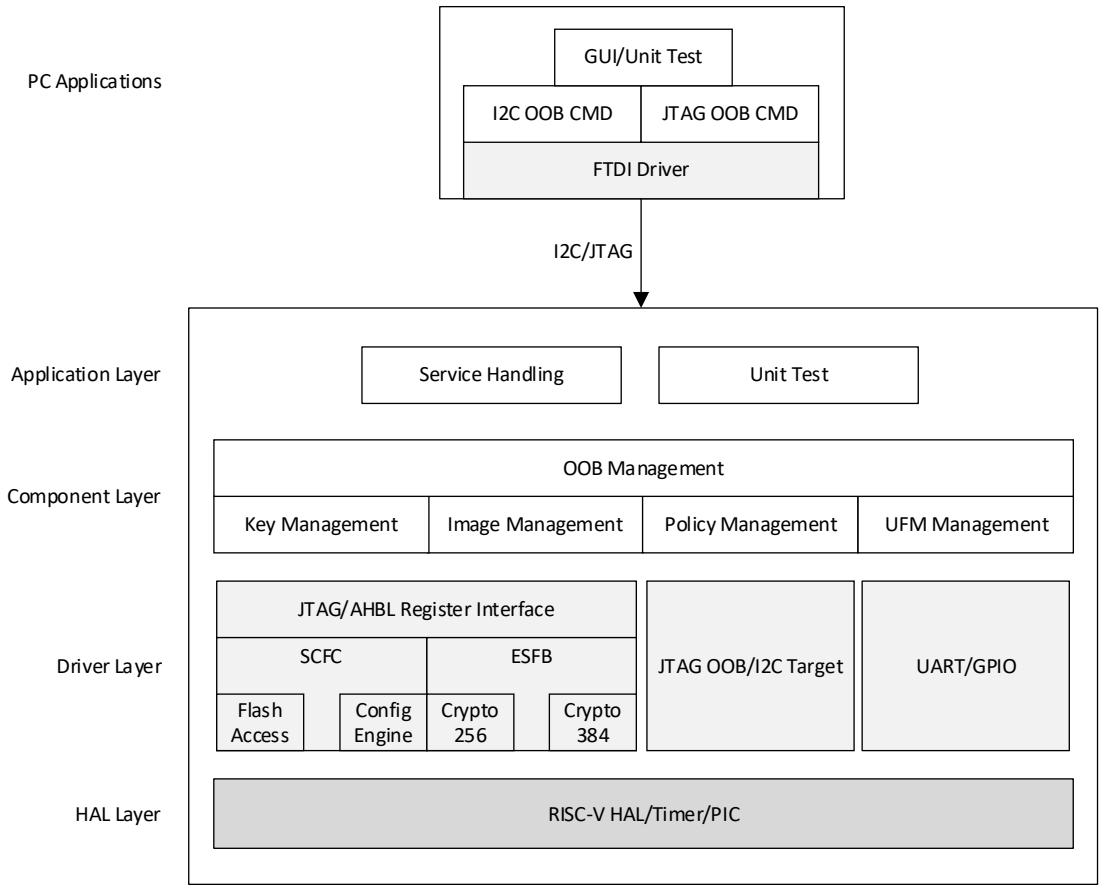

Figure 4.1. User Orchestration Firmware Architecture

4.3. Orchestration Firmware Image

An image signing tool must be used to sign the orchestration firmware image in which a header is attached to the original firmware binary. The firmware header definition is shown in Table 4.1.

Table 4.1. Orchestration Firmware Image Format

Field	Size (Bytes)	Secured	Non-Secured
Magic Number 1	4	*LSCC*. 0x4C, 0x53, 0x43, 0x43	
Firmware size	4	Firmware image size in bytes	
ECDSA signature	132	ECDSA signature (96, 384-96bytes, 521-132 bytes) (big endian)(calculate from RBP version)	
Magic Number 2	4	*HSIW*. 0x48, 0x53, 0x49, 0x57	
hash value	64	Hash 384/512, only calculate firmware, filled when signing, only calculate firmware.	
reserved	48	Zero	
RBP Version	4	From 1 to 255	
Keyblob	1,612	Normal ISK keyblob	
Firmware binary			

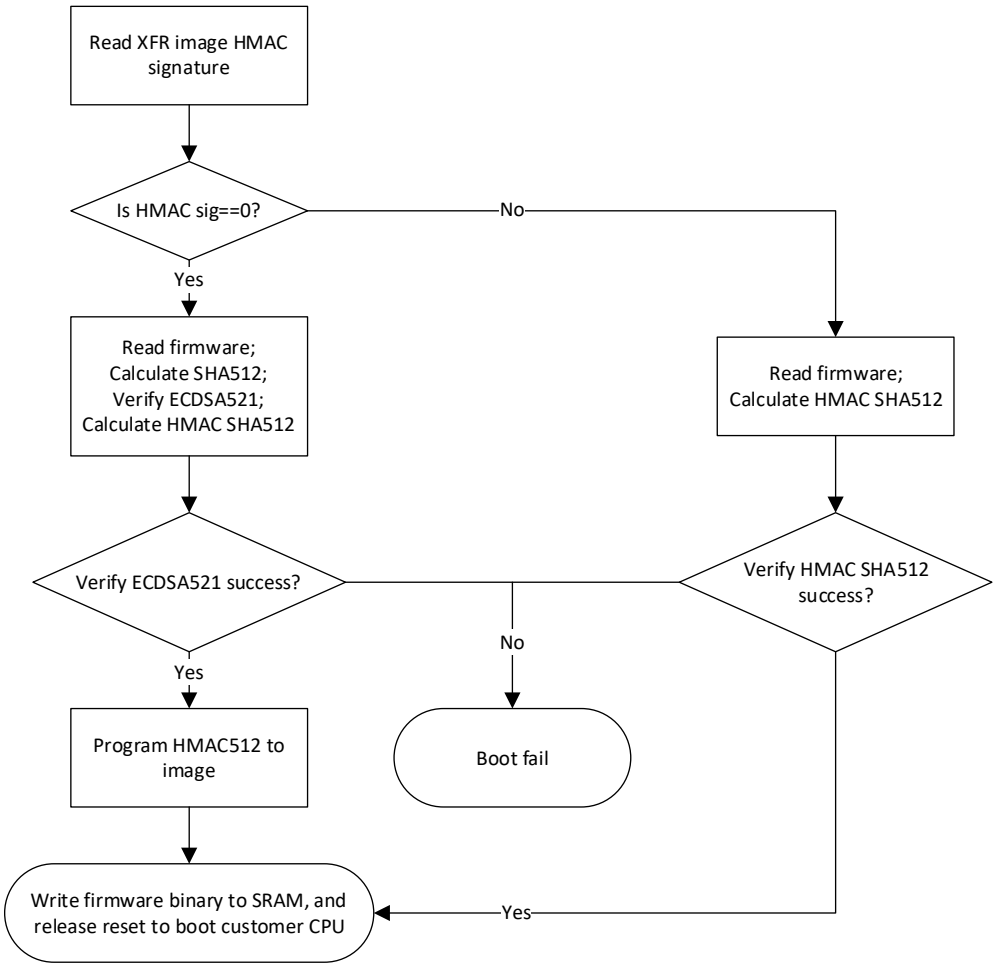

Figure 4.2. Orchestration Firmware Image Boot Flow

4.4. User Orchestration Firmware Flow

Once the user orchestration firmware receives data from the OOB channel, it parses the data packet and calls the appropriate ESFB API. Most of the OOB commands are designed as multi-packet, so the same ESFB API would be called several times to feed the entire data block.

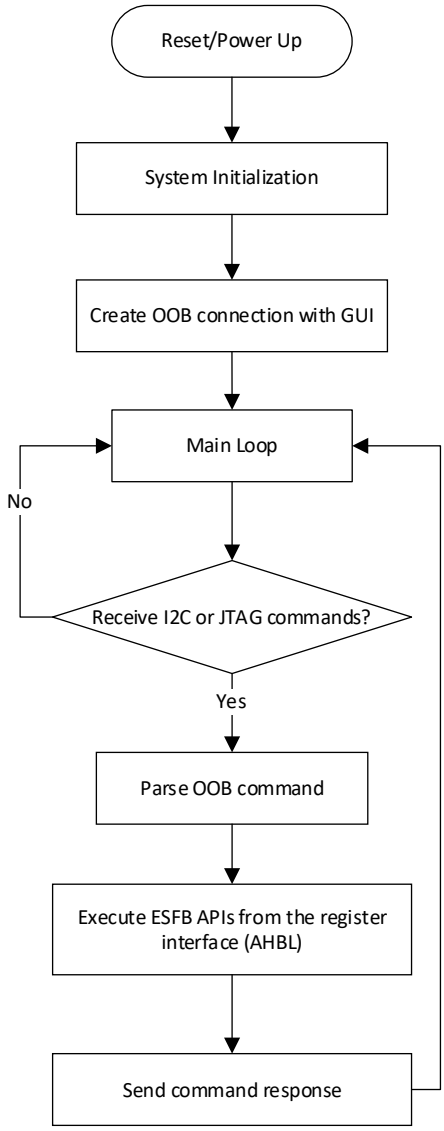

Figure 4.3. User Orchestration Firmware Flow

4.5. ESFB API

Refer to the Embedded Security and Function Block with Advanced Key Management for MachXO5-NX (55TD) Devices Technical Note (FPGA-TN-02353) for a list of the ESFB APIs.

Refer to the Lattice Sentry 4.0 PFR IP API Reference (FPGA-TN-02377) for a list of other Sentry 4.0 APIs.

4.6. OOB Management

4.6.1. OOB Protocol

The OOB channel is used to communicate between System CPUs and user CPU. Two kinds of OOB channels are supported by the LFMXO5-55TD device solution:

- I2C OOB channel
- JTAG OOB channel

These communication interfaces are used to monitor and manage the device, such as image update, key revocation, and get image status. Both I2C and JTAG interfaces communicate to user CPU.

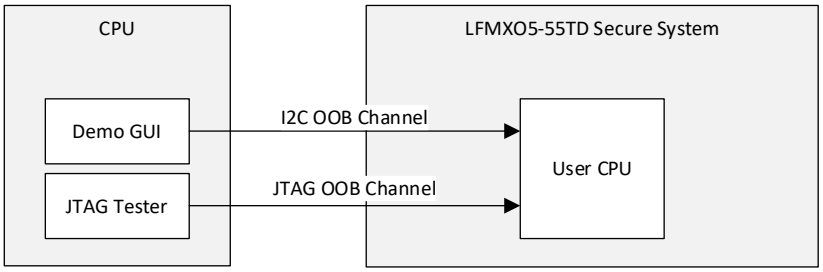

Figure 4.4. OOB Communication to User CPU

Header(0xBC)	CMD ID (1 byte)	Fragment Flag (2 bytes)			Length (1 byte, whole packet size)	Data (Max length 122 bytes)	Checksum
		F	L	Offset			

Figure 4.5. OOB Command Request Packet Format

Header(0xBC)	CMD ID (1 byte)	Fragment Flag (2 bytes)			Length (1 byte)	Type (Status/Data) (1 byte)	Response Data Size (2 bytes)	Checksum
		F	L	Offset				

Figure 4.6. OOB Command Response Status Packet Format

Header(0xBC)	CMD ID (1 byte)	Fragment Flag (2 bytes)			Length (1 byte)	Type (Status/Data) (1 byte)	Detail Data (Max length 121 bytes)	Checksum
		F	L	Offset				

Figure 4.7. OOB Command Response Data Packet Format

4.6.2. Supported Command List

The OOB commands can be extended according to the request. For Lattice demo system, the typical commands are listed in Table 4.2.

Table 4.2. Command List Description

Command name	ID	Description	Command Data Item
Erase customer image	0x01	Authorize keyblob, calculate whole image ISK signature, and compare with signature in customer image.	Keyblob data
Customer Image program	0x02	Program whole image information to flash.	Image data – check customer image format

Command name	ID	Description	Command Data Item
Dry Run Customer Image	0x03	Dry run the specified customer image 1 or 2.	0: Auto select 1: Image 1 (CUA0) 2: Image 2 (CUA1)
Get image status	0x04	Read selected image status bit.	Image IDs: 0: Current 1: Image 1 (CUA0) 2: Image 2 (CUA1)
Revoke KAK	0x05	Keyblob audit, set old KAK invalid, and set new KAK valid.	—
Lock/unlock jtag port	0x06	For JTAG register interface	—
Authorize and Perform zeroization	0x20	Keyblob authenticate (KAK signature)	—
Authorize and perform nullification	0x21	Keyblob authenticate (KAK signature)	—
Read custom policy	0x40	Read the customer policy	—
Read ufm info	0x41	Read the specified UFM	—
Program ufm	—	Program the specified UFM	—

4.6.3. API List

The API is listed in Table 4.3.

Table 4.3. List of APIs

API	Description
Oob message handle	Get message from I2C/JTAG OOB, and parse message.
Oob receive msg	Receive message from I2C or JTAG OOB.
Oob send msg	Send response message to I2C or JTAG OOB.
Oob status	Return OOB status (connection type, availability).

4.6.4. Service Flow

The OOB component should be a thin layer mainly for data exchange, pack/unpack, or encrypt/decrypt.

The flow charts in the subsequent sections describe the basic process for OOB command handling and customer image handling.

4.6.4.1. OOB Message Handling

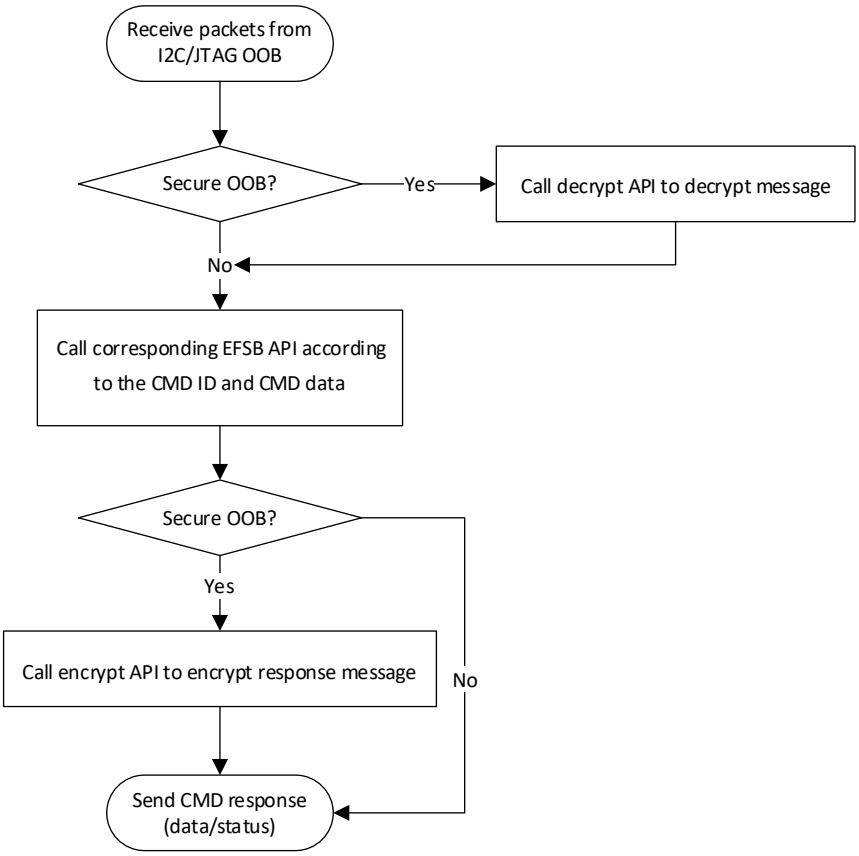

Figure 4.8. OOB Message Flow

4.6.4.2. Image Update

For the customer image to be encrypted, the payload of OOB packet for image update is in raw data. After getting the data from the OOB channel, the payload is fed to the ESFB API and the content is updated to the flash directly. The dry-run function and the key block authentication keep the data integrity and authorization.

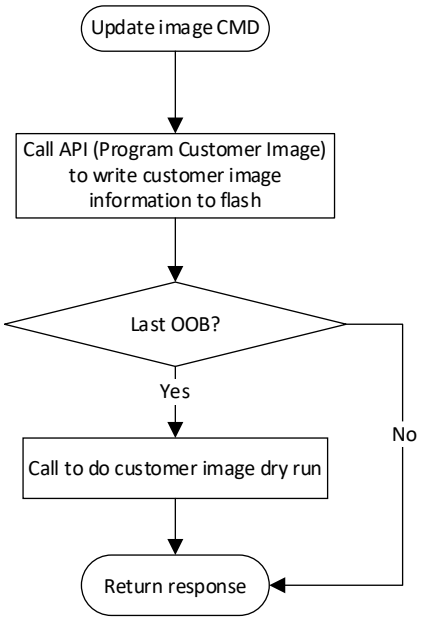

Figure 4.9. Update Customer Image

5. DICE

5.1. DICE Architecture

The ESFB in CUA follows the architecture below to generate layer 0 and 1 CDI keys and certificates:

4. Generate UDS derived key pair:
   a. The certificate that contains the UDS derived public key is generated in Lattice factory in manufacturing phase (through DICE enrollment through LISP) and programmed to the Certificate Sector of internal flash.
   b. ESFB provides API to read the UDS certificate from the Certificate Sector.
5. Generate layer 0 CDI based on UDS and CUA image to derive key pair:
   a. Keep the derived layer 0 private key in secure RAM. Provide API to use this key to sign the input hash digest.
   b. Generate certificate for the derived layer 0 public key. Provide API to read the Layer 0 certificate.
6. Generate layer 1 CDI based on layer 1 CDI and PFR firmware image to derive key pair:
   c. Keep the derived layer 1 private/public key in secure RAM. Provide API to read the key pair out.

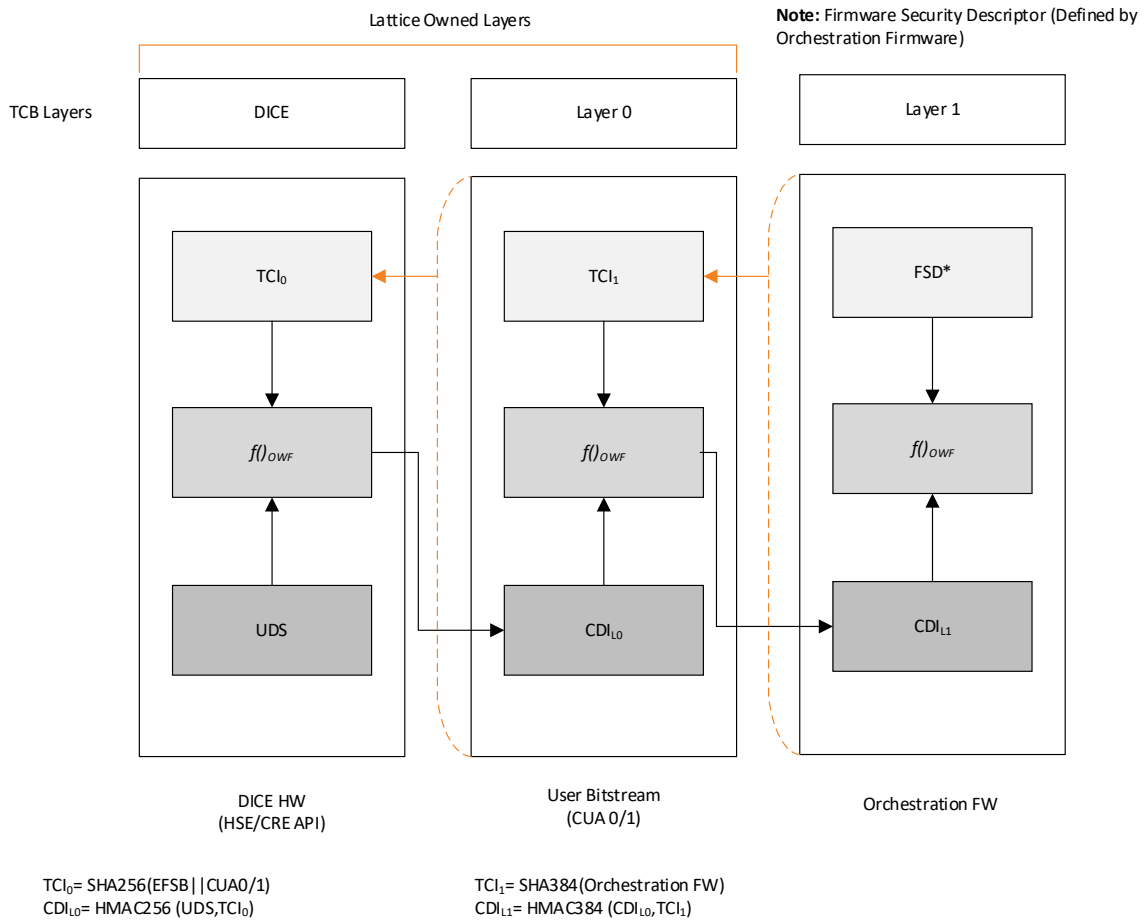

Figure 5.1. DICE Architecture

5.1.1. UDS Layer

- Tmp1 = SFB_UDS_HMAC256(Device_Salt)
  - During the DICE Enrollment stage in the manufacturing, a random number is generated as the device salt to be used for calculation for the DICE engine.
  - The Device_Salt is stored into the flash memory in the Certificate locality. ESFB uses the device salt to regenerate the keys.
- $PrK384_{uds}$ = HKDF (Tmp1)
  - Use HKDF function to derive the key.
- $PbK384_{uds}$ = ECC384_PubKey_Generate($Prk384_{uds}$)
  - Use P-384 ECC to derive public key from given private key.

5.1.2. Layer 0

- CDI_{L0} = HMAC-SHA384(Tmp1, { CUA, firmware signature or hash})
  - CUA and firmware signature is inside the bit-stream images which is on internal flash. There are two CUA images in XO55TD, and ESFB firmware gets the address of CUA image which is running and reads the signature from internal SPI Flash.
- $PrK384_{L0}$ = HKDF (CDI_{L0})
  - Use HKDF key derive function.
  - This private key is kept in the ESFB.
- $PbK384_{L0}$ = ECC384_PubKey_Generate($Prk384_{L0}$)
  - Use P-384 ECC to derive public key from given private key.
  - Generate layer 0 certificate for this derived public key.

5.1.3. Layer 1

- CDI_{L1} = HMAC_SHA384 (CDI_{L0}, {XFR Provides TCI_{L1}})
  - Orchestration PFR firmware. Its length and version are in the header of the signed image on external flash.
- $PrK384_{L1}$ = HKDF (CDI_{L1})
  - This layer 1 private key is kept in the secure RAM.
- PbK384L1 = ECC384_PubKey_Generate(PrK384L1)
  - This layer 1 public key is kept in the secure RAM.

5.2. Certificate format

- UDS certificate: Enrolled in Lattice factory, so the certificate format is not defined here.
- Layer 0 certificate:

```
    Signature Algorithm: ecdsa-with-SHA384
    Serial Number:
        d0:c1:2a:c0:60:06:63:c9
    Signature Algorithm: ecdsa-with-SHA384
    Issuer:  serialNumber=[Hash of PbK384uds], CN = [TraceID]
    Validity
        Not Before:  Oct  5 01:07:46 2022 GMT
        Not After :  Dec 31 23:59:59 9999 GMT
    Subject:  serialNumber =[Hash of PbK384L0]
    Subject Public Key Info:
        Public Key Algorithm: id-ecPublicKey
            Public-Key: (384 bit)
            pub:
                04:72:12:9e:b7:ce:71:c7:c3:68:b3:3a:34:dc:a0:
                a8:8c:4c:0b:2e:a7:40:7a:a1:90:8b:ef:b5:9c:ea:
```

```
                8e:e8:ef:61:a0:65:6e:2e:bc:04:5d:17:62:c5:23:
                e0:c6:e4:ec:5a:d4:d6:7d:fd:c5:7b:c3:be:1e:0f:
                af:3d:f5:15:a6:75:85:83:49:94:9d:ed:3a:d1:95:
                89:fc:97:e5:36:76:ee:9e:cb:a0:50:d7:10:96:16:
                aa:0d:1f:b4:83:66:64
                ASN1 OID: secp384r1
            NIST CURVE: P-384
    X509v3 extensions:
        X509v3 Extended Key Usage:
            ecdsa-with-Specified, 2.23.133.8.1.6
Signature Algorithm: ecdsa-with-SHA384
     30:65:02:31:00:9f:33:43:5a:08:6f:a5:dc:96:5c:90:44:fa:
     8d:2b:af:ac:03:f5:44:1c:58:ad:5b:6c:70:c5:0d:d2:39:45:
     6d:17:37:53:1d:02:d4:e9:8b:01:a3:23:a1:04:80:68:ff:02:
     30:13:bf:8c:a2:b5:30:63:c1:ae:38:30:72:1a:cd:d1:b5:c2:
     d5:0b:0c:1e:33:18:a5:66:fb:12:ed:be:c0:f6:fe:9a:15:31:
     e3:0b:12:81:bc:8b:82:33:f4:dc:b5:93:66
```

- Layer 1 certificate: Generated by customer firmware, so the certificate format is not defined here.

5.3. DICE APIs Exposed to Customer Firmware

These are the APIs that are used to generate associated CDI and Certificates for the DICE attestation function:

```
int esfb_dice_cert_get(esfb_ctx_t *esfb_inst_p, uint8_t *p_cert, uint32_t *cert_length);
int esfb_l0_cert_get(esfb_ctx_t *esfb_inst_p, uint8_t *p_l0_cert, uint32_t *cert_length);
int esfb_cdi_keypair_derive(esfb_ctx_t *esfb_inst_p, struct ecc_point * p_publickey,
uint8_t p_privateKey[NUM_ECC_DIGITS_384]);
int esfb_cdi_ecdsa_sign(esfb_ctx_t *esfb_inst_p, uint32_t digest[], uint32_t pub_key_l1[],
uint32_t signature[]);
int esfb_dev_trn_get(esfb_ctx_t *esfb_inst_p, uint8_t *p_trn);
```

6. PFR System Design

Lattice Propel is a platform for embedded system design, development, and validation. Lattice Propel provides a PFR Solution Template to simplify customer PFR solution design.

For more information, refer to the Lattice Sentry 4.0 MachXO5-NX LFMXO5-55TD Walkthrough Guide (FPGA-UG-02217).

6.1. PFR Solution Template

The Lattice Sentry 4.0 RoT Project template provides a baseline PFR implementation with all required features enabled. You can follow the Lattice Propel tool flow to create or modify a standard PFR design.

Choose the **Lattice Sentry 4.0 RoT Project Template** during the **Select Template** step. After that, follow the steps described in the Lattice Sentry 4.0 MachXO5-NX LFMXO5-55TD Walkthrough Guide (FPGA-UG-02217).

For more information on the Propel tools workflow, refer to the Lattice Propel User Guide.

6.2. PFR System Design Customization

You can customize your hardware and software designs on top of the Lattice Sentry 4.0 RoT Project Template to meet your specific requirements.

To customize your design when creating a new PFR system design, see the following recommendations:

- After creating the SoC project, customize the SoC design in Lattice Propel Builder.
- After creating a project in the Lattice Radiant software:
  - Add/edit RTL source files to bring in customer logic;
  - Edit the PDC file for I/O mapping and constraint settings.
- After the software project is created, edit the source files in Propel SDK.

Further changes can be made to the existing Sentry 4.0 template design which is created through the Propel tool sets. Note that when an SoC design is changed in the System Builder, it is necessary to regenerate the hardware project to update the system environment file (sys_env.xml). After that, the software project needs to be updated with the updated sys_env.xml file, in order to regenerate the board support package (BSP).

References

- Lattice Sentry Solution Stack web page
- MachXO5-NX web page
- Lattice Propel Design Environment web page
- Lattice Radiant Software web page
- NIST SP 800-193: Platform Firmware Resiliency Guidelines
- Lattice Sentry SMBus Mailbox IP User Guide (FPGA-IPUG-02165)
- DC-SCM LTPI IP User Guide (FPGA-IPUG-02200)
- Lattice Sentry I2C Filter IP User Guide (FPGA-IPUG-02166)
- Lattice Sentry 4.0 MachXO5-NX LFMXO5-55TD Walkthrough Guide (FPGA-UG-02217)
- Lattice Sentry 4.0 Demo Board for MachXO5-NX (FPGA-EB-02071)
- Lattice Sentry 4.0 PFR IP API Reference (FPGA-TN-02377)
- Embedded Security and Function Block with Advanced Key Management for MachXO5-NX (55TD) Devices Technical Note (FPGA-TN-02353)
- Lattice - Sentry 4.0 BKC for XO5-55TD – Hardware Connections Technical Note (FPGA-TN-02374)
- Lattice Insights web page for Lattice Semiconductor training courses and learning plans

Technical Support Assistance

Submit a technical support case through www.latticesemi.com/techsupport.

For frequently asked questions, refer to the Lattice Answer Database at www.latticesemi.com/Support/AnswerDatabase.

Revision History

Revision 1.0, December 2025

Section	Change Summary
All	Initial release.

www.latticesemi.com