MachXO3D Cyber Resilience Reference Design on EXOR uSOM10

Reference Design

FPGA-RD-02344-1.0

September 2026

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.

Abbreviations	Definition
AHB	Advanced High-performance Bus
APB	Advanced Peripheral Bus
BFM	Bus Functional Model
CFG	Configuration
CS3	Control and Status for System Simulation
DUT	Design Under Test
ECC	Elliptic Curve Cryptography
ECDSA	Elliptic Curve Digital Signature Algorithm
EEPROM	Electrically Erasable Programmable Read-Only Memory
EFB	Embedded Functional Block
ESB	Embedded Security Block
FPGA	Field-Programmable Gate Array
GPIO	General Purpose Input/Output
I/O	Input/Output
I2C	Inter-Integrated Circuit
MC	Microcontroller
OOB	Out-of-Band
PFR	Platform Firmware Resiliency
PKI	Public Key Infrastructure
PLD	Programmable Logic Device
QSPI	Quad Serial Peripheral Interface
QSW	QSPI Switch
RISC-V	Reduced Instruction Set Computer Five
RoT	Root of Trust
RTC	Real Time Clock
SDK	Software Development Kit
SHA256	Secure Hash Algorithm 256-bit
SoM	System-On-Module
SPI	Serial Peripheral Interface
SSL	Secure Sockets Layer
TLS	Transport Layer Security
TRNG	True Random Number Generator
UART	Universal Asynchronous Receiver/Transmitter
UFM	User Flash Memory

1. Introduction

The Cyber Resilience Reference Kit is a collaborative solution from Lattice Semiconductor, EXOR International, and TrustiPhi, designed to help industrial and edge device manufacturers implement comprehensive security measures without increasing design complexity. Built on the Lattice MachXO3D™ FPGA, EXOR's uSOM10 platform, and TrustiPhi's ProtoPilot, it delivers hardware-rooted trust, secure lifecycle management, centralized fleet security configuration, and authenticated industrial communication through a unified, plug-and-play workflow that accelerates the path to a secure system design.

This reference design also incorporates Platform Firmware Resiliency (PFR), a hardware-based security approach that prevents unauthorized firmware modifications, detects tampering, and automatically restores the system to a trusted state when needed. The solution is built on the Lattice MachXO3D Root of Trust (RoT) FPGA, integrated with an ARM-based System-On-Module (SoM) from EXOR and TrustiPhi's security management software, facilitating seamless integration into existing industrial system architectures. Extending the foundational MachXO3D Lattice Sentry™ Root of Trust design, this reference implementation adds an ECDSA-based runtime security service that supports firmware authentication, digital signature generation, and TLS/SSL certificate creation. The design targets EXOR's SOM, which pairs an NXP i.MX8M Plus processor with the MachXO3D RoT device and the Lattice ECP5™ FPGA — where the processor manages connectivity and hosts the OPC UA environment. The MachXO3D governs the secure boot process of both the ECP5 FPGA and the i.MX8 processor, demonstrating its ability to control multiple host devices within the system. Together, these components showcase the MachXO3D core security capabilities, including secure boot, PKI-based security, secure key storage, on-chip flash memory, and dual-boot resilience.

1.1. Quick Facts

Download the reference design files from the Lattice reference design page on the Lattice website.

Table 1.1. Summary of the Reference Design

General	Target Devices	MachXO3D
	Source Code Format	Verilog, C
Regression Test	Functional Automated Test	Performed
	System Level Integration Test	Performed
	Testbench	Performed
Software Requirements	Software Tool and Version	• Lattice Propel™ SDK 2026.1 • Lattice Propel Builder 2026.1 • Lattice Diamond® 3.13
	IP Version (if applicable)	• AHB-Lite Interconnect v1.6.0 • AHB-Lite to APB Bridge v1.2.0 • APB Interconnect v1.5.0 • EFB v1.0.0 • GPIO v1.8.1 • Lattice Sentry™ Embedded Security Block Mux v1.0.0 • Lattice Sentry I2C Filter v1.4.1 • Lattice Sentry QSPI Monitor v1.0.0 • Lattice Sentry QSPI Master Streamer 1.0.0 • RISC-V MC v2.9.0 • System Memory v2.6.0 • UART v1.3.0
Hardware Requirements	Board	EXOR uS10 board
	Cable	• Lattice Programming Cable • USB to UART converter • Micro USB cable

1.2. Features

The MachXO3D Cyber Resilience reference design includes the following key features:

- Secure boot with MachXO3D
- Host firmware authentication
  - 256-bit ECDSA algorithm – NIST certified
- ECDSA runtime security service
  - ECDSA public/private key pair generation and management
  - I2C Out-of-Band (OOB) communication with the host, including reading the public key from UFM3
  - Data signing
- Secure keys storage in User Flash Memory 3 (UFM3)
- Manifest and Log Storage in User Flash Memory 2 (UFM2)
- I2C Filter – filtering I2C data packets
- QSPI Monitor — monitors allowed, blocked, and undefined address regions within SPI flash
- QSPI Master Streamer — handles host image authentication and firmware recovery

1.3. Limitations

In the current hardware revision, the MachXO3D is unable to hold the i.MX8 Plus in reset. This limitation will be addressed in a future hardware revision.

2. Functional Description

In addition to the RoT PFR capabilities described in the Lattice Sentry Root of Trust Demo Setup for MachXO3D (FPGA-UG-02114), this design provides a runtime ECDSA service. This service supports host firmware authentication, digital signature generation for secure inter-system communication, and TLS/SSL certificate creation for server registration and authentication. These features are accessed through extended I2C OOB messages.

On the board, the MachXO3D device connects to four external SPI flash devices: Flash A, Flash B, Flash C, and Flash D. These flashes store the boot images for the i.MX8 Plus processor and the ECP5 FPGA. Flash A and Flash B serve the i.MX8 Plus, while Flash C and Flash D serve the ECP5. The MachXO3D controls access to these flashes through the QSPI flash switch (QSW) and manages the boot sequence for both the i.MX8 Plus and ECP5. Upon power-up of the uS10 gateway, the MachXO3D asserts reset on both chips while authenticating their respective bitstreams. After successful authentication, the MachXO3D device releases the reset signal to allow the chips to proceed with the boot sequence. For ongoing security monitoring, a QSPI Monitor tracks all SPI transactions to the flash devices, and an I2C Filter acts as a transparent intermediary between the I2C master and slaves to detect and block malicious I2C activity.

The MachXO3D device also supports image recovery. If a corrupted SoC image is detected, the MachXO3D authenticates the golden (backup) image, overwrites the corrupted image with the verified backup, and releases the SoC to boot from the restored image. Note that this capability is not demonstrated in this reference design.

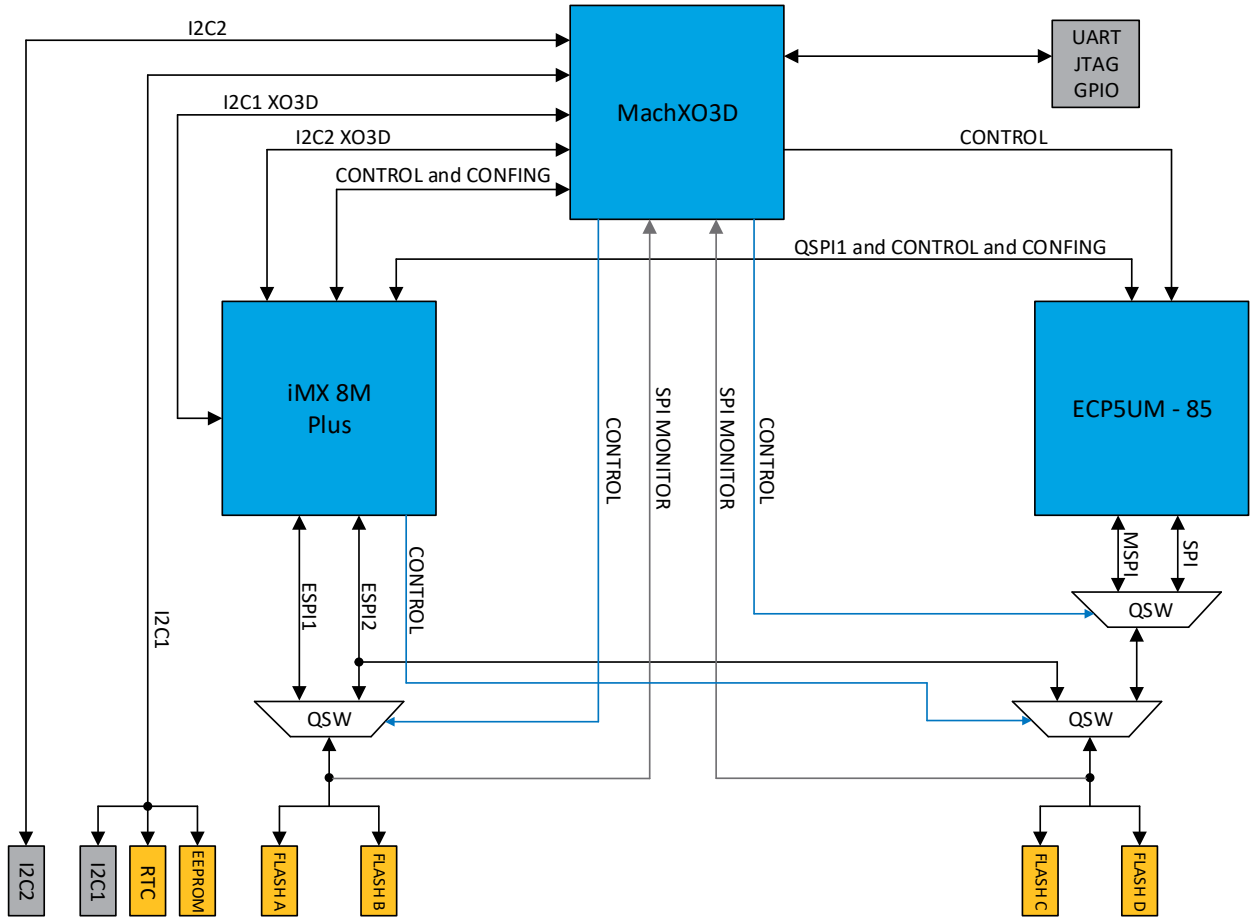

Figure 2.1. EXOR uS10 System Block Diagram

2.1. Design Components

The MachXO3D Cyber Resilience reference design is built around a RISC-V soft SoC that uses standard AHB and APB buses for interconnection. The design includes the following blocks:

Table 2.1. Design Components

Component	Description
Embedded Functional Block (EFB)	A hard IP core that provides I2C interface support, Configuration Blocks (CFG), and User Flash Memory (UFM).
Embedded Security Block Mux (ESB Mux)	An interface through which the MachXO3D accesses the Embedded Security Block (ESB) to perform cryptographic operations.
GPIO	General-purpose I/O pins used to control LEDs and reset signals.
I2C Filter	An IP core that monitors I2C traffic and filters out malicious transactions.
QSPI Master Streamer	An IP core that handles communication with flash devices over both SPI and QSPI protocols.
QSPI Monitor	An IP core that monitors SPI transactions to the connected SPI flash devices.
UART	Provides asynchronous serial communication to the host for console output.
RISC-V MC	Main control unit in the system.
System Memory	Local system memory used to store and execute the PFR firmware.

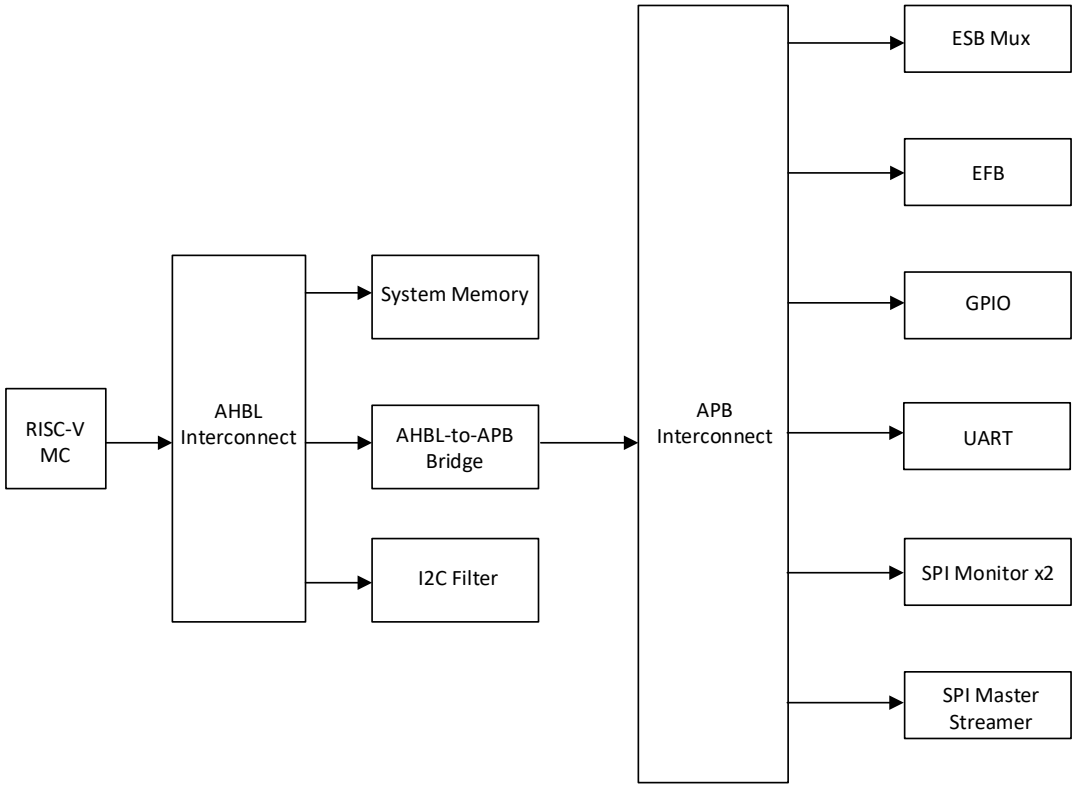

Figure 2.2. MachXO3D Cyber Resilience Reference Design Block Diagram

2.2. Clocking Scheme

All design blocks derive their clock from the MachXO3D internal oscillator, which operates at 44.33 MHz.

2.3. Reset Scheme

The rstn_i input serves as the single active-low reset source for all blocks in the design.

3. Signal Descriptions

Table 3.1 lists the primary I/O interface signals for the MachXO3D Cyber Resilience reference design. The corresponding interface groups are shown in Figure 2.1.

Table 3.1. Primary I/O for MachXO3D Cyber Resilience Reference Design

Port Name	I/O	Interface Group	Description
ESPI1_CS0n_QS	In	CSn from i.MX8	Chip Select signal from the i.MX8 to the MachXO3D for Flash A. This signal activates SPI monitoring on Flash A.
ESPI2_CS0n_QS	In	CSn from i.MX8	Chip Select signal from the i.MX8 to the MachXO3D for Flash B. This signal activates SPI monitoring on Flash B.
IMX8_FSPI_CSn	Out	MachXO3D QSPI	SPI Streamer Chip Select output from the MachXO3D.
IMX8_FSPI_CLK	Out	MachXO3D QSPI	SPI Streamer CLK signal for i.MX8 flash
IMX8_FSPI_MOSI	Out	MachXO3D QSPI	SPI Streamer MOSI signal for i.MX8 flash
IMX8_FSPI_MISO	In	MachXO3D QSPI	SPI Streamer MISO signal for i.MX8 flash
QS_IMX8_EN	Out	MachXO3D QSW Controls	QSPI switch enable signal for i.MX8 flash
SEL_OR_FB	Out	MachXO3D QSW Controls	Select output from the MachXO3D to switch to SPI Flash B.
SEL_OR_FA	Out	MachXO3D QSW Controls	Select output from the MachXO3D to switch to SPI Flash A.
I2C2_SCL_XO3D	InOut	MachXO3D I2C OOB	I2C2 SCL signal used for OOB messaging
I2C2_SDA_XO3D	InOut	MachXO3D I2C OOB	I2C2 SDA signal used for OOB messaging
XO3D_UART_RX	In	MachXO3D UART	UART RX signal
XO3D_UART_TX	Out	MachXO3D UART	UART TX signal
IMX8_RSTn	Out	i.MX8 Reset	i.MX8 reset signal
ECP5_RSTn	Out	ECP Reset	ECP5 reset signal
FA_RSTn	Out	MachXO3D Flash Reset	Flash A reset signal
FB_RSTn	Out	MachXO3D Flash Reset	Flash B reset signal
I2C1_SCL_XO3D	InOut	I2C Filter Master	Main I2C SCL signal from the i.MX8 to the MachXO3D.
I2C1_SDA_XO3D	InOut	I2C Filter Master	Main I2C SDA signal from the i.MX8 to the MachXO3D.
I2C1_SCL	InOut	I2C Filter Slave	I2C SCL signal from the MachXO3D to downstream I2C devices on the board, such as RTC and EEPROM.
I2C1_SDA	InOut	I2C Filter Slave	I2C SDA signal from the MachXO3D to downstream I2C devices on the board, such as RTC and EEPROM.

4. Cyber Resilience Reference Design Architecture

4.1. MachXO3D PFR Firmware

4.1.1. Firmware Flow

Figure 4.1 illustrates the complete RoT and PFR firmware execution flow for the MachXO3D.

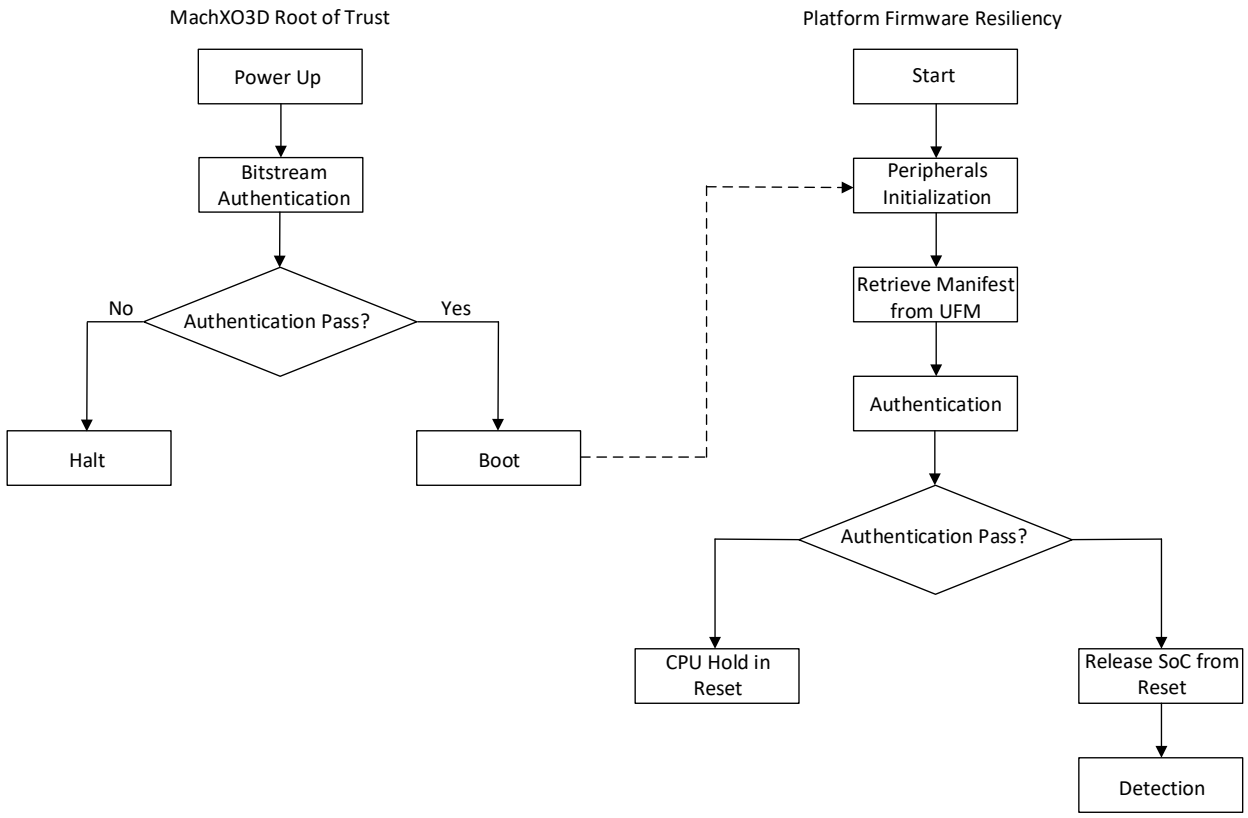

Figure 4.1. MachXO3D Cyber Resilience Reference Design Flow

4.1.2. Authentication Phase

Figure 4.2 shows the authentication phase, detailing how the MachXO3D uses ECDSA to authenticate the i.MX8 firmware.

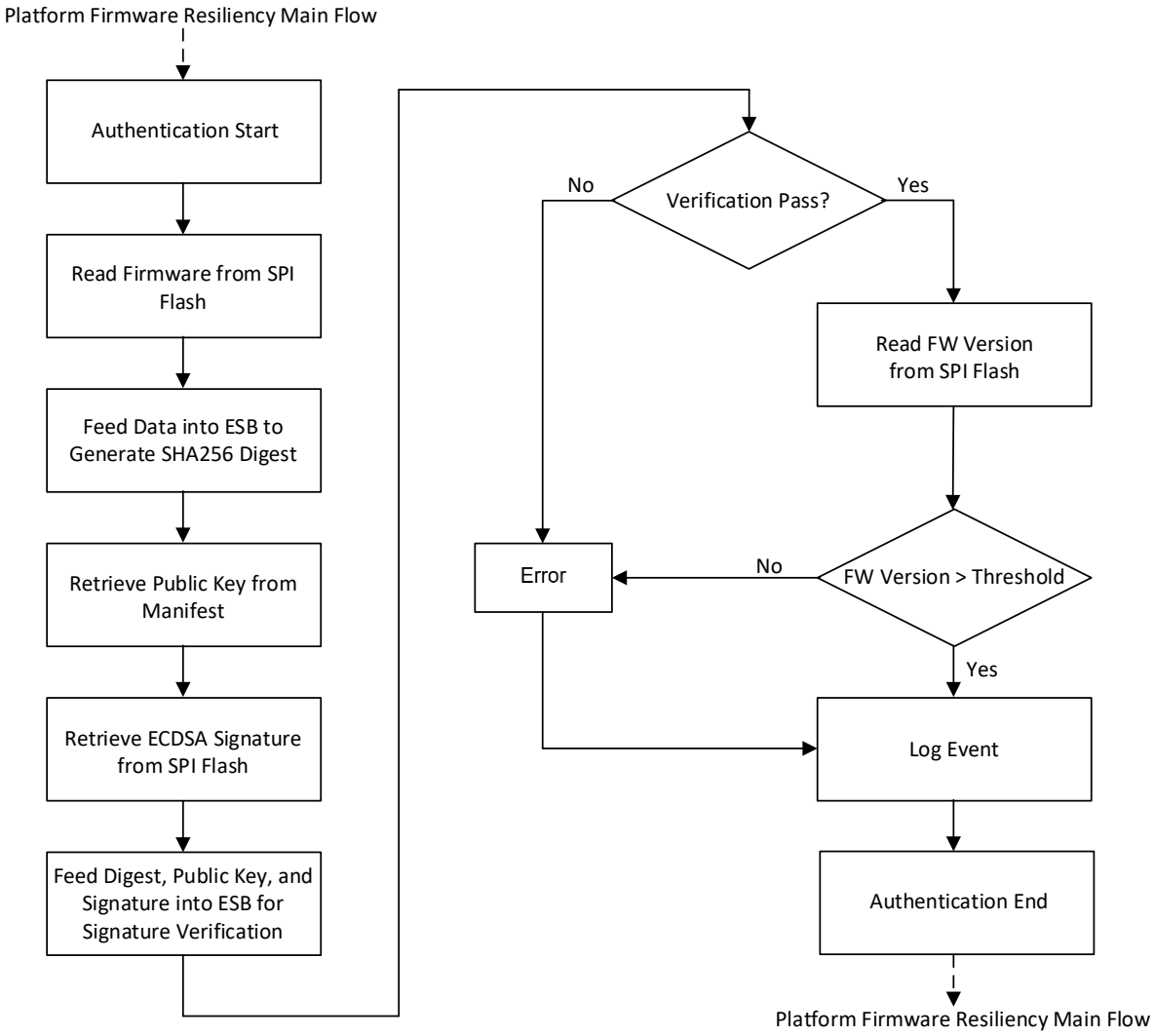

Figure 4.2. Authentication Flow

4.1.3. Detection Phase – I2C Filter/SPI Monitor

Figure 4.3 shows the detection phase, which covers real-time monitoring of I2C and SPI events.

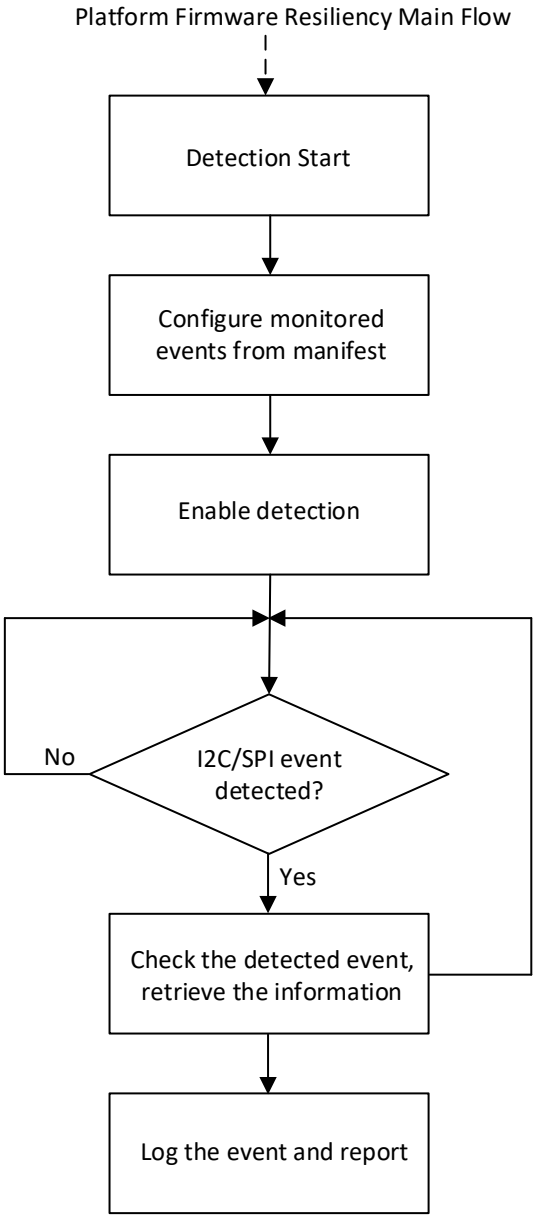

Figure 4.3. Detection Flow

5. ECDSA Security Service Runtime

5.1. Data Structures

The ECDSA security service requires two new data structures in esb.h. The ecc_sig structure stores an ECDSA signature, while the ecc_keypair structure stores an ECDSA private/public key pair.

```
struct ecc_sig {
      unsigned char r[NUM_ECC_DIGITS];
      unsigned char s[NUM_ECC_DIGITS];
} ;

struct ecc_keypair {
      unsigned char priv_key[NUM_ECC_DIGITS];
      unsigned char pub_key[NUM_ECC_DIGITS * 2];
} ;
```

5.2. Application Programming Interfaces (APIs)

The following sections describe the new APIs introduced to support the ECDSA security service.

5.2.1. ESB_Mux

Table 5.1. esb_ecc_keypair_gen Function

esb_ecc_keypair_gen	
struct ecc_keypair esb_ecc_keypair_gen(struct esb_instance *this_esb)	
Parameter	**Description**
this_esb	The pointer to the ESB instance
Returns	**Description**
struct ecc_keypair	A key pair struct containing a 32-byte private key and a 64-byte public key.
Description	
Generates an ECC key pair. The private key is produced by a 256-bit random number generator, and the public key is derived from it.	

Table 5.2. esb_ecc_sig Function

esb_ecc_sig	
struct ecc_sig esb_ecc_sig(struct esb_instance *this_esb, unsigned int *private_key, unsigned int *sha256_digest)	
Parameter	**Description**
this_esb	The pointer to the ESB instance
private_key	The pointer to the private key instance
sha256_digest	The pointer to the SHA256 digest instance
Returns	**Description**
struct ecc_sig	An ECC signature struct containing the R and S components.
Description	
Generates an ECC signature consisting of a 32-byte R component and a 32-byte S component.	

Table 5.3. esb_ecc_verify Function

esb_ecc_verify	
unsigned char esb_ecc_verify(struct esb_instance *this_esb, EccPoint *pub_key, struct ecc_sig *ecc_sig, unsigned int *digest, unsigned char *auth_pass)	
Parameter	**Description**
this_esb	The pointer to the ESB instance
pub_key	The pointer to the public key instance
ecc_sig	The pointer to the ECC signature instance
digest	The pointer to the SHA256 digest instance
auth_pass	The pointer to the pass/fail flag
Returns	**Description**
unsigned char	0: Verification failed 1: Verification passed
Description	
Verifies an ECC signature against the provided public key, ECC signature, and SHA256 digest.	

5.2.2. ECC

Table 5.4. ecc_gen_keypair Function

ecc_gen_keypair	
unsigned char ecc_gen_keypair(struct esb_instance *esb, struct efb_instance *efb, unsigned int key_idx)	
Parameter	**Description**
esb	The pointer to the ESB instance
efb	The pointer to the EFB instance
key_idx	32-bit key index
Returns	**Description**
unsigned char	0: No error in key pair generation 1: Error in key pair generation
Description	
An ECC component function that interfaces with the main application and invokes the ESB driver to generate an ECC key pair.	

Table 5.5. ecc_get_pubkey Function

ecc_get_pubkey	
struct EccPoint ecc_get_pubkey(struct esb_instance *esb, struct efb_instance *efb, unsigned int key_idx)	
Parameter	**Description**
esb	The pointer to the ESB instance
efb	The pointer to the EFB instance
key_idx	32-bit key index
Returns	**Description**
struct EccPoint	An ECC public key struct containing a 32-byte X component and a 32-byte Y component.
Description	
Retrieves the ECC public key stored in User Flash Memory 3 (UFM3).	

Table 5.6. ecc_sign_msg Function

ecc_sign_msg	
struct ecc_sig ecc_sign_msg(struct esb_instance *esb, struct efb_instance *efb, unsigned int *sha256_digest, unsigned int key_idx)	
Parameter	**Description**
esb	The pointer to the ESB instance
efb	The pointer to the EFB instance
sha256_digest	The pointer to the SHA256 digest instance
key_idx	32 bits key index
Returns	**Description**
struct ecc_sig	Struct of the ECC signature consisting of 32 bytes of R component and 32 bytes of S component
Description	
Generates an ECC signature consisting of a 32-byte R component and a 32-byte S component.	

6. I2C Out-of-Band Message

For detailed information regarding I2C Out-of-Band messages, contact the Lattice Semiconductor Support Team.

7. Compiling and Running the Reference Design

This section provides step-by-step instructions for building and running the MachXO3D Cyber Resilience reference design using Lattice Propel and Lattice Diamond software tools. For detailed information on these tools, refer to the Lattice Propel software user guide and the Lattice Diamond software user guide.

7.1. Software Tools

- Propel SDK 2026.1 — used to build the MachXO3D PFR firmware (C application).
- Propel Builder 2026.1 — used to assemble the SoC design for the MachXO3D PFR reference design, including all required IP blocks.
- Lattice Diamond 3.13 — the FPGA design tool used to synthesize and generate the MachXO3D bitstream.

7.2. Compiling the C Project

To compile the C project, perform the following:

1. Launch the Propel software.
2. Set the downloaded reference design firmware folder as the workspace.
3. In the Project Explorer, right-click the project and select **Build Project** to initiate firmware compilation. The .mem file is created.

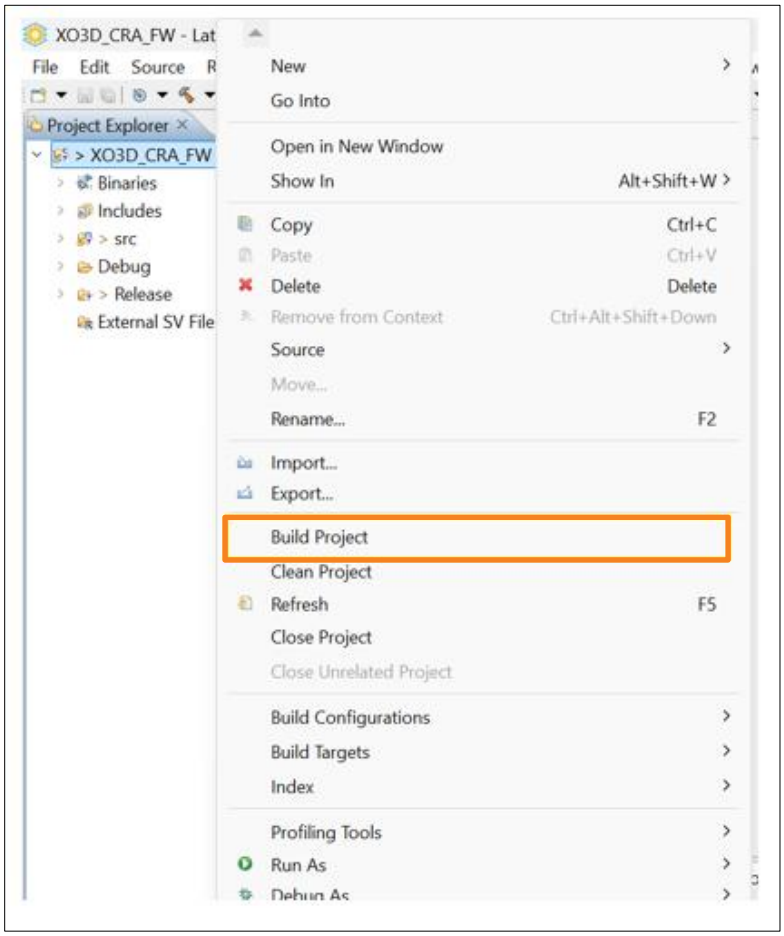

Figure 7.1. Compile the C Project

7.3. Loading the Reference Design

To load the reference design, perform the following:

1. Launch the Propel Builder software.
2. Go to **File > Open Design**.
3. Select the reference design Propel project file (.sbx) to open it.

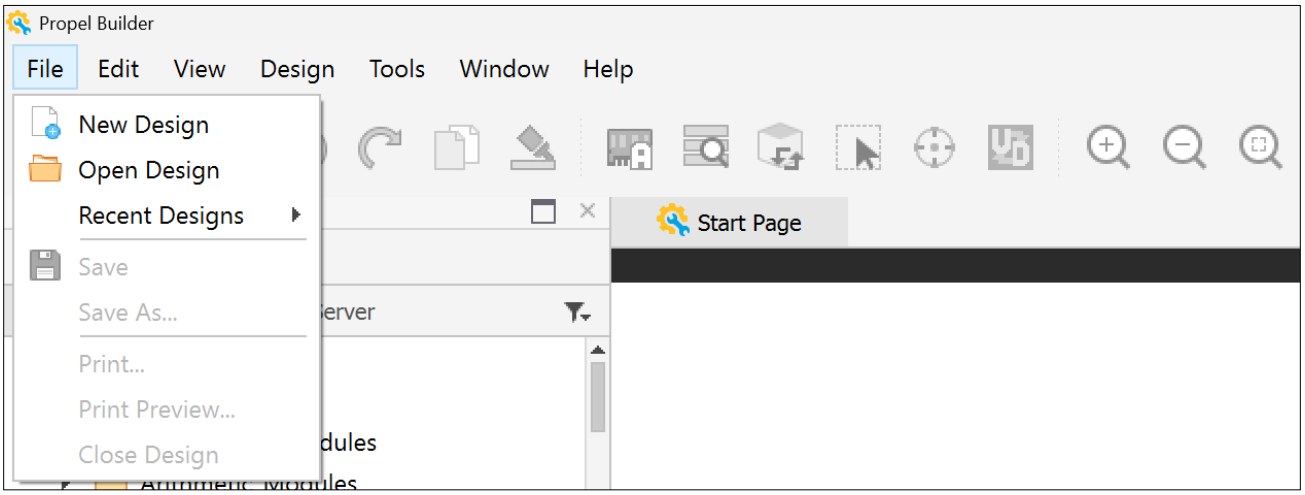

Figure 7.2. Open Design

4. Double-click the **System Memory IP** block.
5. Update the Initialization File field to point to the .mem file generated in the previous step.

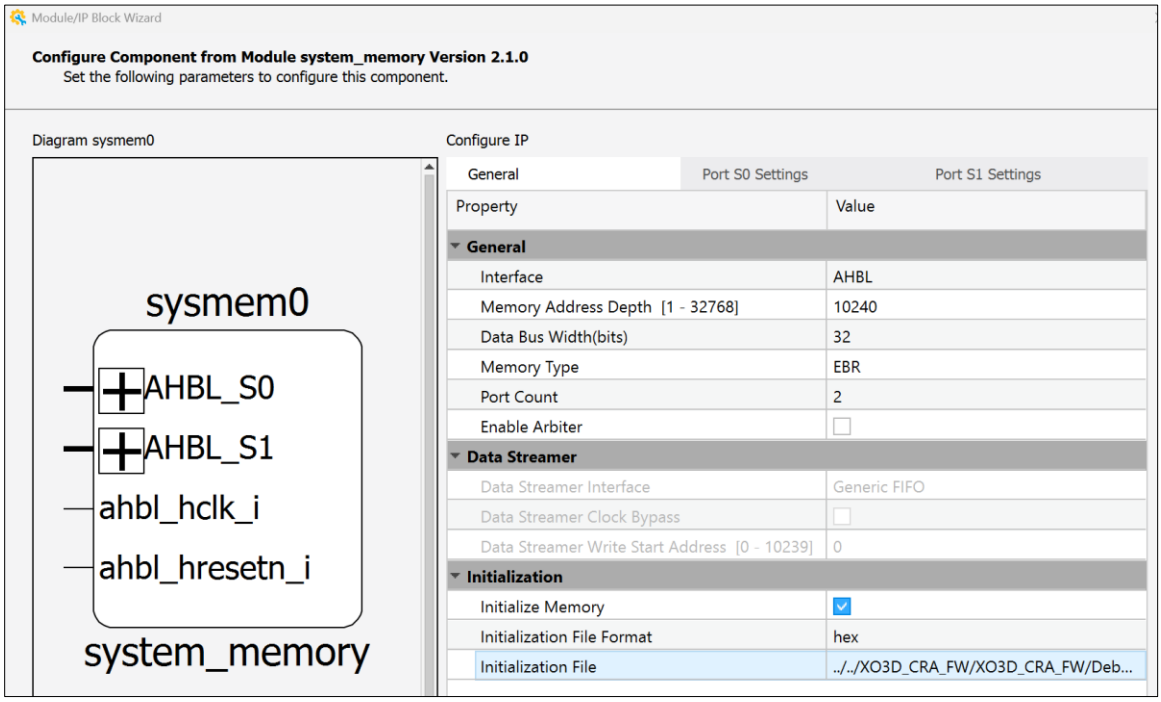

Figure 7.3. Update System Memory

6. Click **Generate**.

7.4. Compiling the Reference Design

To compile the reference design, perform the following:

1. Launch the Lattice Diamond software.
2. Right-click **Export File** and select **Rerun** or **Rerun All** to trigger the compilation.

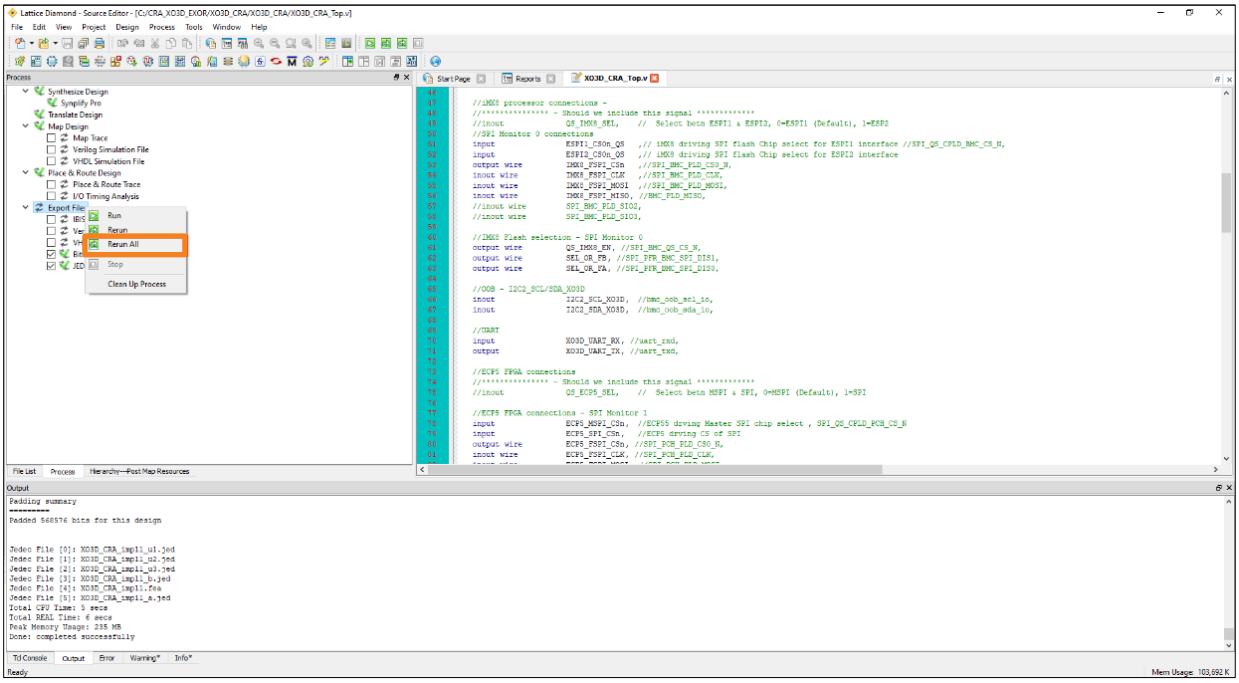

Figure 7.4. Reference Design Compilation with Lattice Diamond 3.13

7.5. Generating the ECDSA Signature for a Binary Image

To generate the ECDSA signature, perform the following:

1. Open the Diamond Security Settings tool by navigating to **Tools > Security Setting**. When prompted, authenticate using the default password *LATTICESEMI*.

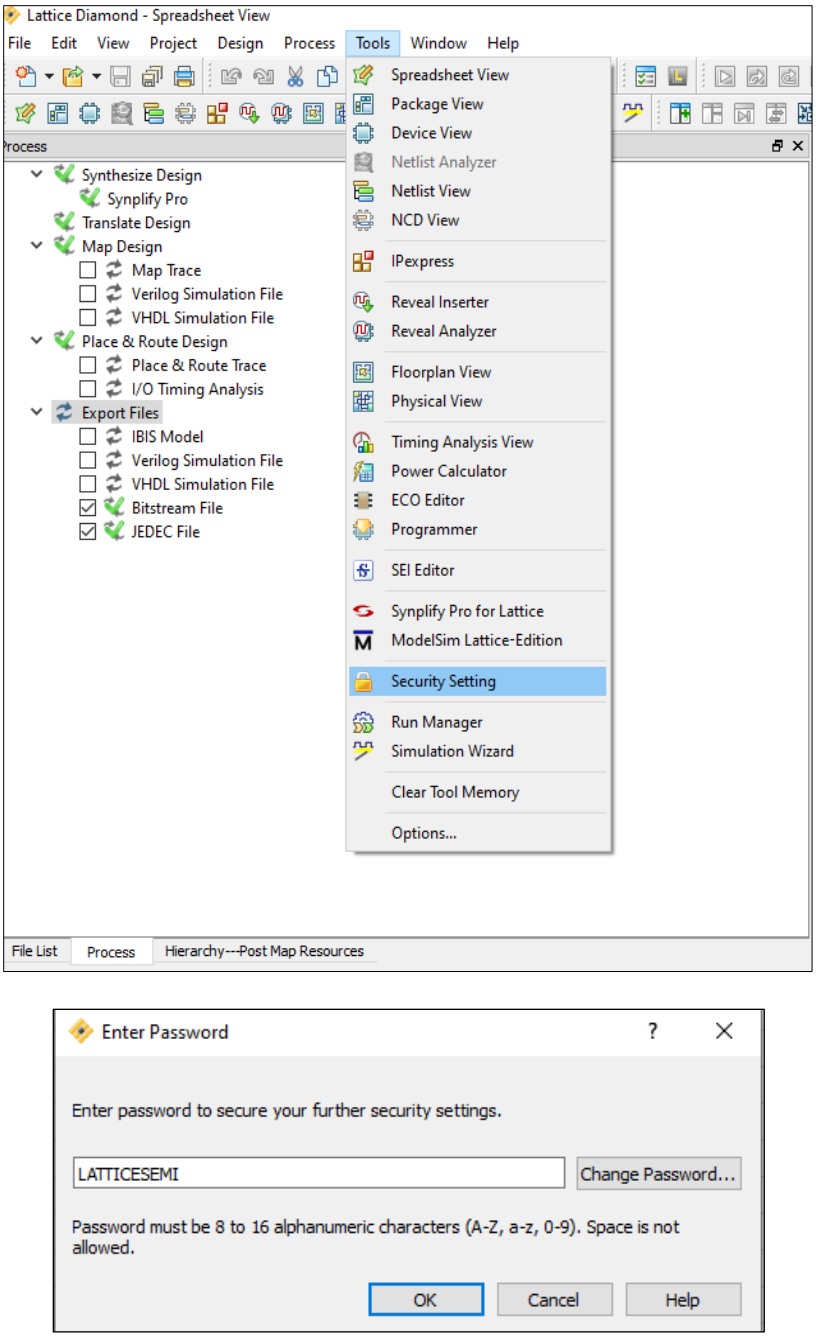

Figure 7.5. Diamond Security Setting Tool

2. Navigate to the **Signature Authentication** tab and select **Signature Generation**. Click **Auto Generated Key Pair** to generate an ECDSA private/public key pair automatically.
3. Click **Load from File** and select *flash.bin* as the input.
4. Click **Generate** to produce the ECDSA signature. The tool produces three output files: a digest file (flash.bin.digest), a signature in hex format (flash.bin.sig.hex), and a signature in plain text (flash.bin.sig).

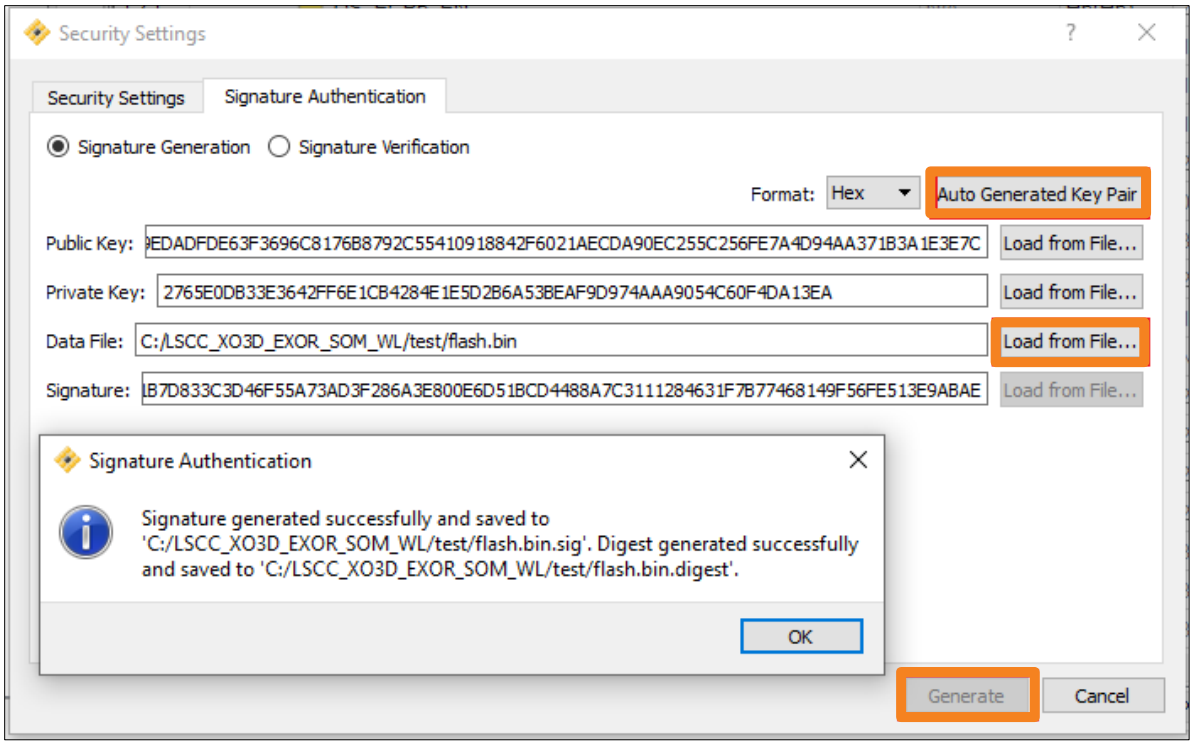

Figure 7.6. Generating ECDSA Signature

5. Use the Linux *xxd* command to convert the plain text signature file (.bin.sig) to a binary signature file (.bin).

7.6. Creating and Modifying the MachXO3D Manifest

Lattice Propel provides a Manifest Manager tool to manage the manifest for your own system. The manifest is stored in UFM2 of the MachXO3D device.

To modify the manifest, perform the following:

1. Open Lattice Propel SDK and go to **LatticeTools > Lattice Sentry Manifest Manager** to run the Manifest Manager.

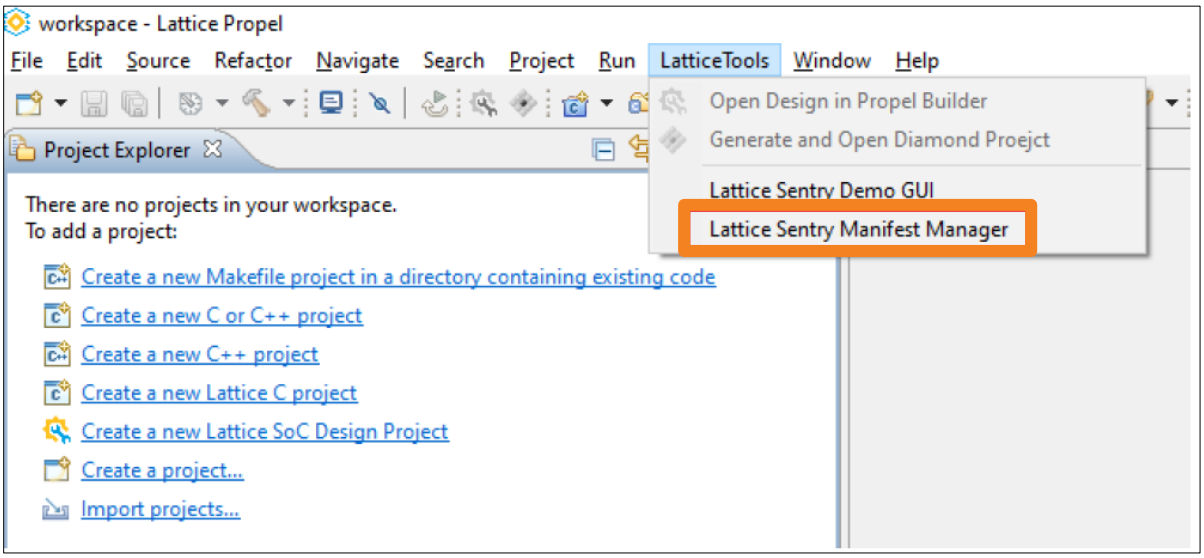

Figure 7.7. Launch Manifest Manager in Lattice Propel SDK

2. Click **Open** and choose the .mem file. The Manifest Manager loads the .mem file and parses its manifest information as shown in the three tabs: Image Data, Flash Data, and I2C Filter Data.
3. Click **Generate** to create the .mem file for UFM2 initialization.

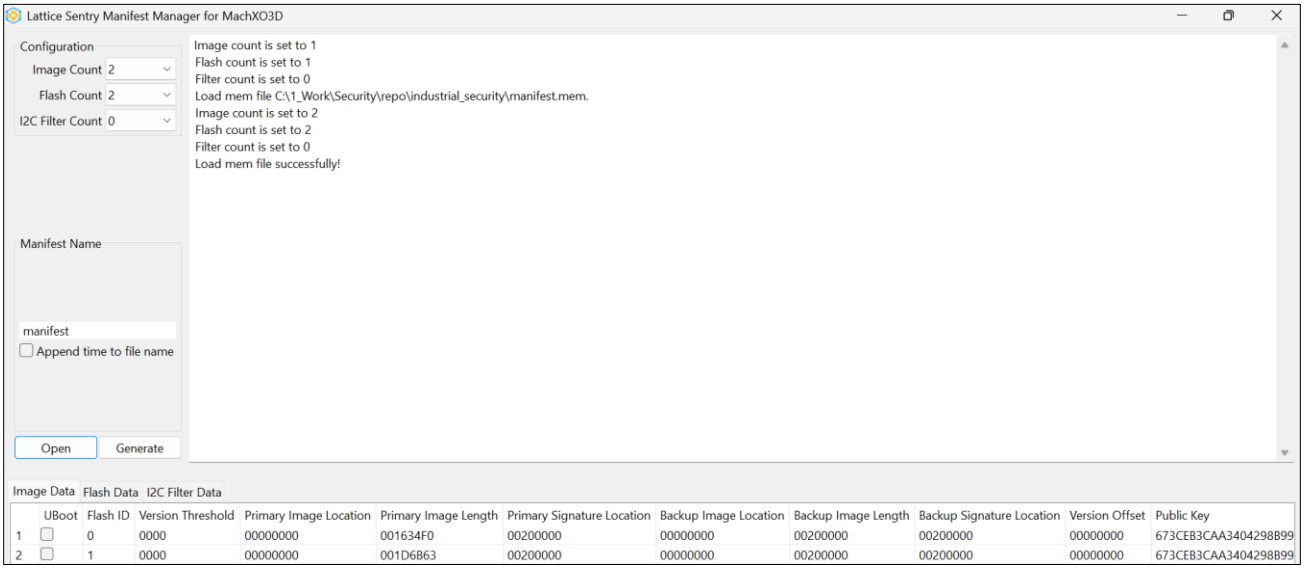

Figure 7.8. Manifest Manager Window

7.7. Programming MachXO3D Firmware and Manifest

To program the MachXO3D firmware, perform the following:

1. In the Device Properties window, go to the **Operation** tab and select **FLASH CFG Erase, Program, Verify**.
2. Ensure the **CFG0 Programming Options** are checked.
3. Choose the bitfile with the .jed extension.

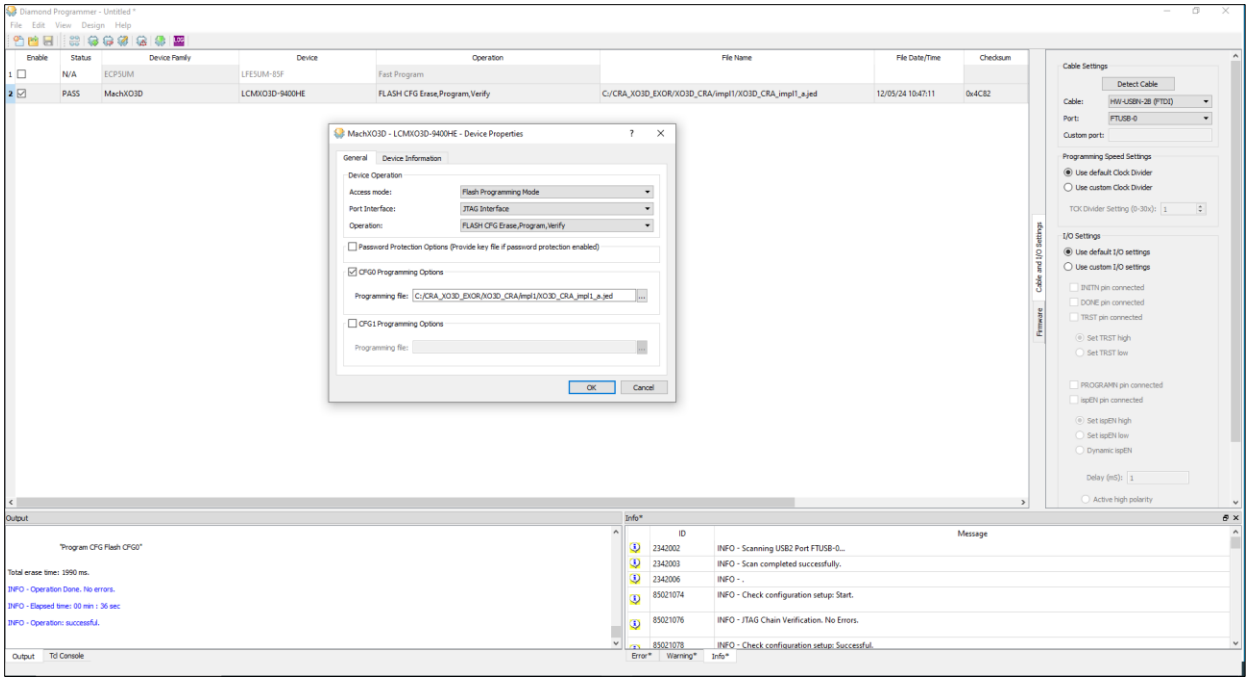

Figure 7.9. Programming MachXO3D Firmware

To program the MachXO3D manifest, perform the following:

1. In the Device Properties window, go to the **Operation** tab and select **FLASH UFM Erase, Program, Verify**.
2. Ensure the **UFM2/UFM3 Programming Options** and **UFM2 Programming** file are checked.
3. Choose the manifest file with the .jed extension.

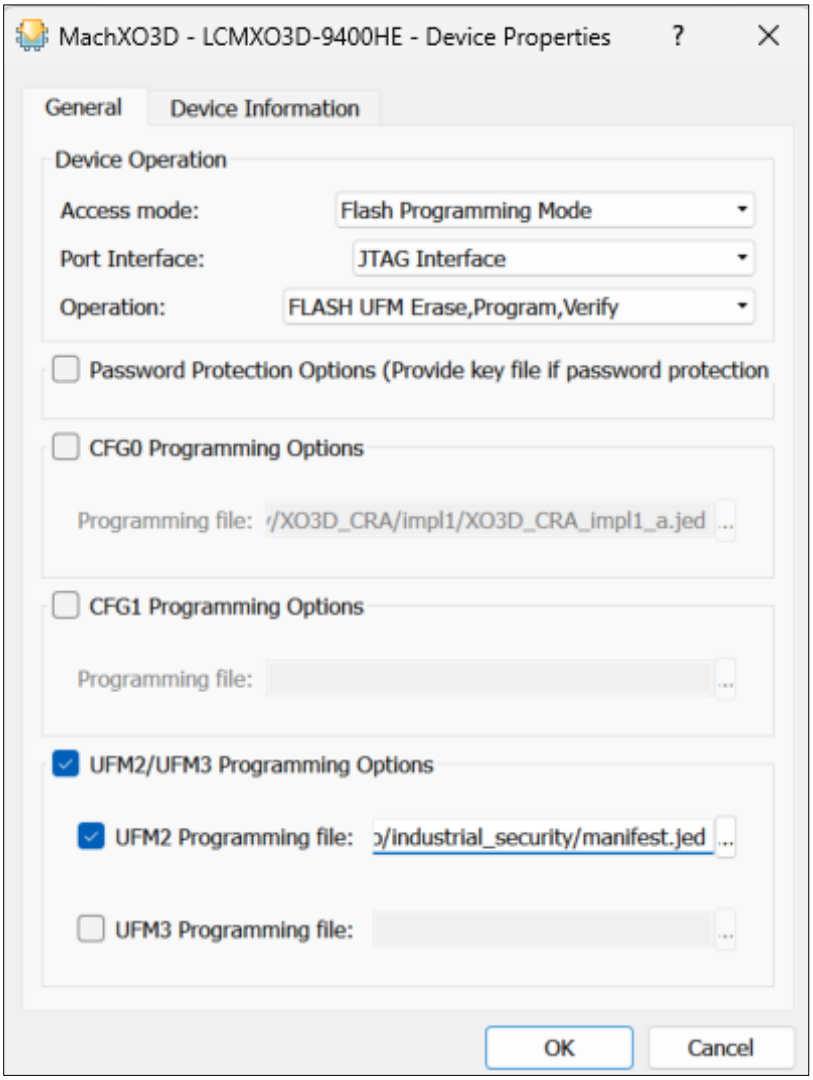

Figure 7.10. Programming MachXO3D Manifest

8. Industrial Security Simulation

8.1. Industrial Security Simulation Testbench Overview

The Industrial Security simulation testbench, shown in Figure 8.1, offers a flexible environment for validating and debugging the system-level design. The testbench also supports the addition of custom test cases to the Industrial Security system.

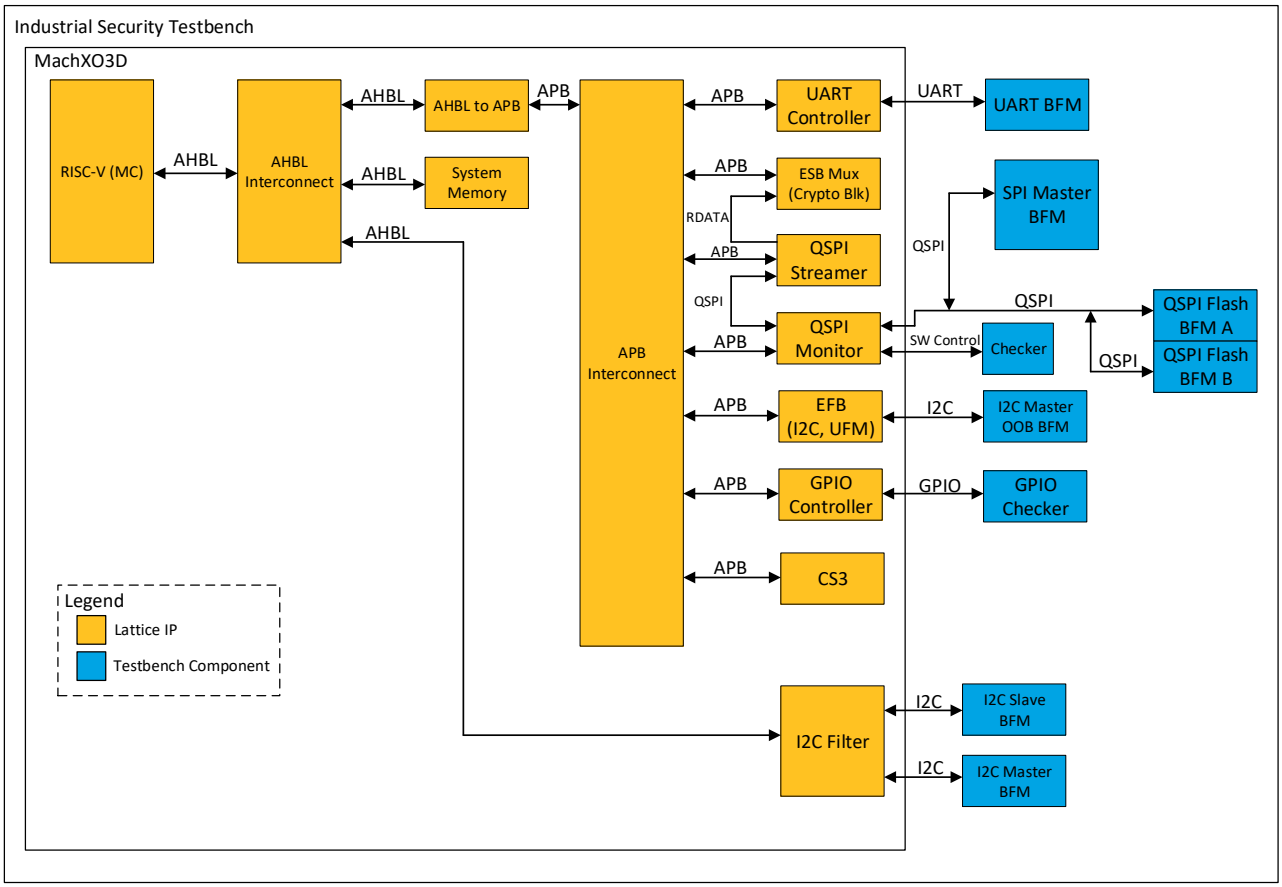

Figure 8.1. Industrial Security Simulation Testbench

Within the testbench, the Design Under Test (DUT) is controlled by a RISC-V MC core executing the dedicated test firmware. The Bus Functional Models (BFM) and checkers are included to simulate and verify interactions with the DUT's peripheral interfaces.

8.2. Testbench Configuration

The verification directory contains all testbench files. It contains two subdirectories: socsim, which holds simulation components, and bfm, which contains the Bus Functional Models.

The socsim folder contains XO3D_CRA_v.sv, the top-level testbench file, as well as flist_socsim.f, which lists all BFM paths and file references. The bfm folder holds all BFM directories and source files. When adding a new BFM, update flist_socsim.f to include the new directory and file path.

Figure 8.2 shows an example of how to add the SPI Master BFM entry to flist_socsim.f.

XO3D_CRA > verification > socsim > flist_socsim.f

```
+incdir+../bfm/apb_slave_register/1.0.0/rtl
../bfm/apb_slave_register/1.0.0/rtl/apb_slave_register.sv
+incdir+../bfm/qspi_slave_bfm/1.0.0/rtl
../bfm/qspi_slave_bfm/1.0.0/rtl/qspi_slave_bfm.sv
+incdir+../bfm/qspi_slave_bfm/1.0.0/rtl/MX25L51245G
../bfm/qspi_slave_bfm/1.0.0/rtl/MX25L51245G/MX25L51245G.v
+incdir+../bfm/qspi_slave_bfm/1.0.0/rtl/W25Q512JVxIQ
../bfm/qspi_slave_bfm/1.0.0/rtl/W25Q512JVxIQ/W25Q512JVxIQ.v
+incdir+../bfm/i2c_peripheral/1.0.0/rtl/
../bfm/i2c_peripheral/1.0.0/rtl/i2c_peripheral.v
../bfm/i2c_peripheral/1.0.0/rtl/i2c_peripheral_define.v
../bfm/i2c_peripheral/1.0.0/rtl/registerInterface.v
../bfm/i2c_peripheral/1.0.0/rtl/serialInterface.v
+incdir+../bfm/sysclk_rst_gen1/1.0.0/rtl/
../bfm/sysclk_rst_gen1/1.0.0/rtl/sysclk_rst_gen1.v
../bfm/sysclk_rst_gen1/1.0.0/rtl/sysclk_rst_gen1_globals.v
../bfm/sysclk_rst_gen1/1.0.0/rtl/SysClk_Rst_GenMon.sv
+incdir+../bfm/uart_model1/1.0.0/rtl/
../bfm/uart_model1/1.0.0/rtl/uart_model.v
../bfm/uart_model1/1.0.0/rtl/uart_model1.v
../bfm/uart_model1/1.0.0/rtl/uart_model1_globals.v
../bfm/uart_model1/1.0.0/rtl/uart_rx.v
../bfm/uart_model1/1.0.0/rtl/uart_tx.v
+incdir+../bfm/i2c_bfm/
../bfm/i2c_bfm/i2c_master_bfm.sv
../bfm/i2c_bfm/i2c_slave_bfm.sv
+incdir+../bfm/i2c_oob_bfm/
../bfm/i2c_oob_bfm/i2c_oob_master_bfm.sv
+incdir+../bfm/spi_master_bfm/
../bfm/spi_master_bfm/spi_master_bfm.sv
```

Figure 8.2. Industrial Security Simulation Testbench Configuration

8.2.1. Test Suites

The TestSuites folder contains SystemVerilog (.sv) files, each targeting a specific component of the Industrial Security hardware design. Table 8.1 lists all available test suites and their descriptions. To test a new component, add a corresponding .sv test suite file to this folder.

Table 8.1. Available Test Suites in the Simulation Environment

Test Suite	Description
TestEcdsa.sv	Validates the cryptography block, including the True Random Number Generator (TRNG), ECDSA public key derivation, signing, and signature verification.
TestGpio.sv	Validates GPIO input and output behavior.
TestI2Cfilter.sv	Validates I2C filter behavior based on whitelist and blacklist configurations.
TestI2COOB.sv	Validates I2C OOB write, read, and unsupported command handling.
TestQSPImonitor.sv	Validates QSPI Monitor behavior against the configuration set in Propel.

Test Suite	Description
TestSPIstreamer.sv	Validates SPI Streamer flash erase, program, and read operations.
TestSysmem.sv	Validates system memory read/write operations and basic code execution.
TestUart.sv	Validates UART console output.
TestUfm.sv	Validates UFM read and write operations.

8.2.2. Control and Status for System Simulation (CS3)

CS3 is a simulation-only IP module that provides testbench control and status monitoring. It allows test cases to specify which test to execute, with test logic running on the RISC-V MC core. This module is not used in production hardware.

The CS3 IP core contains 32 registers: 16 writable control registers accessible by the APB master, and 16 read-only status registers. Each register is 32 bits wide, addressed via a 10-bit address bus. Table 8.2 and Table 8.3 list the control and status registers used in the simulation. The remaining registers are available for general purpose use.

Table 8.2. Control Registers Used in the Simulation

Control Register	Signal
Control Register 0	Bit 0 – done Bit 1 – start check
Control Register 1	current test suite ID
Control Register 2	current test case ID
Control Register 3	sw msg ID
Control Register 4	sw msg register 0
Control Register 5	sw msg register 1
Control Register 6	read data register
Control Register 7	sw msg counter
Control Register 8	I2C OOB transmit start
Control Register 9	I2C OOB receive ready
Control Register 10	I2C master transmit start
Control Register 11	SPI master transmit start

Table 8.3. Status Registers Used in the Simulation

Status Register	Signal
Status Register 0	testbench status
Status Register 1	testsuite ID

8.2.3. Bus Functional Model

The Bus Functional Model is a software model that replicates the communication behavior of a bus or protocol interface, without necessarily implementing full hardware functionality. In this testbench, BFMs drive and monitor signals at the DUT's peripheral interfaces. The available BFMs are listed below. Additional BFMs can be added to the bfm directory, with corresponding entries added to flist_socsim.f.

Table 8.4. BFMs Available in the Industrial Security Simulation

BFM	Description
UART BFM	Simulates the behavior of a UART device.
I2C Master BFM	Simulates the behavior of an I2C master.
I2C Slave BFM	Simulates the behavior of an I2C slave.
I2C OOB Master BFM	Simulates the behavior of an I2C OOB master.
QSPI Flash BFM A	Simulates the behavior of a QSPI flash device.

BFM	Description
QSPI Flash BFM B	Simulates the behavior of a QSPI flash device.
SPI Master BFM	Simulates the behavior of a SPI master.

8.2.4. Test Firmware

In the simulation framework, the DUT is controlled by a RISC-V microcontroller core running dedicated test firmware. This firmware is in the XO3D_CRA_TEST_FW directory and is primarily implemented in main.c, using the driver APIs from the IP drivers. It is compiled with the same RISC-V compiler, framework, and drivers used for the Propel firmware. At runtime, the TestSuite ID determines which test the firmware executes. New tests can be added to main.c as additional test functions.

8.3. Running the Simulation Test

Simulations are run using ModelSim Lattice FPGA Edition 2025.2. To launch the simulation, perform the following steps.

1. Navigate to the socsim folder using the cd command. For example:
   ```
   cd C:\\CRA_XO3D_EXOR\\XO3D_CRA\\verification\\socsim
   ```
2. Run a test using the do socsim.do <test_name> command. For example:
   ```
   do socsim.do TestUart
   ```

Note: Before running the simulation, ensure that the System Memory initialization file is updated to reference the simulation firmware.

Table 8.5 lists all simulation tests available in the framework.

Table 8.5. Tests Available in the Industrial Security Simulation

Test	Description
TestUart	UART tests
TestSysmem	System memory tests
TestGpio	GPIO tests
TestUfm	UFM tests
TestEcdsa	ECDSA tests
TestI2Cfilter	I2C filter tests
TestI2COOB	I2C Out of Band (OOB) tests
TestSPImonitor	SPI Monitor tests
TestSPIstreamer	SPI Streamer tests

Figure 8.3 shows the ModelSim interface after a successful simulation run.

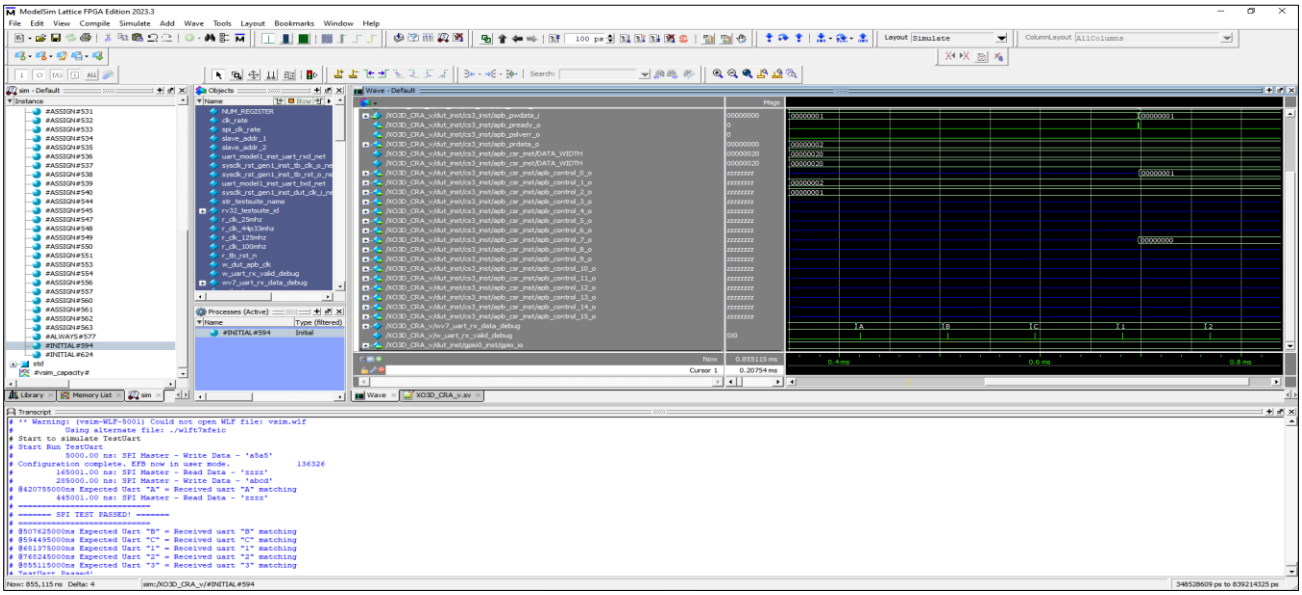

Figure 8.3. ModelSim Interface Displayed During Simulation Execution

9. License

The Lattice Sentry IP blocks used in this reference design require a valid license. Without a valid license, the design can still be compiled and a bitstream generated. However, a built-in timeout mechanism causes the design to cease operation after a predetermined interval. The precompiled bitstream included with this package was built with a valid license. License details for each IP block are available in the corresponding IP user guide.

10. Resource Utilization

```
Design Information

Command line:   map -a MachXO3D -p LCMXO3D-9400HE -t WLCSP69 -s 5 -oc Commercial
     XO3D_CRA_impl1.ngd -o XO3D_CRA_impl1_map.ncd -pr XO3D_CRA_impl1.prf -mp
     XO3D_CRA_impl1.mrp -lpf C:/1_Work/Security/repo/industrial_security/XO3D_CR
     A/impl1/XO3D_CRA_impl1_synplify.lpf -lpf
     C:/1_Work/Security/repo/industrial_security/XO3D_CRA/XO3D_CRA.lpf -c 0 -gui
     -msgset C:/1_Work/Security/repo/industrial_security/XO3D_CRA/promote.xml
Target Vendor:  LATTICE
Target Device:  LCMXO3D-9400HEWLCSP69
Target Performance:   5
Mapper:  se5c00,  version:  Diamond (64-bit) 3.13.0.56.2
Mapped on:  08/10/26  14:15:45


Design Summary
   Number of registers:   4694 out of  9577 (49%)
      PFU registers:          4691 out of  9400 (50%)
      PIO registers:             3 out of   177 (2%)
   Number of SLICEs:      4624 out of  4700 (98%)
      SLICEs as Logic/ROM:    4525 out of  4700 (96%)
      SLICEs as RAM:            99 out of  3525 (3%)
      SLICEs as Carry:         870 out of  4700 (19%)
   Number of LUT4s:        9103 out of  9400 (97%)
      Number used as logic LUTs:         7165
      Number used as distributed RAM:    198
      Number used as ripple logic:       1740
      Number used as shift registers:       0
   Number of PIO sites used: 32 + 4(JTAG) out of 59 (61%)
   Number of block RAMs:  48 out of 48 (100%)
   Number of GSRs:        1 out of 1 (100%)
   EFB used :          Yes
   JTAG used :         No
   Readback used :     No
   Oscillator used :   Yes
   Security used :     Yes
   Startup used :      No
   POR :               On
   Bandgap :           On
   Number of Power Controller:  0 out of 1 (0%)
   Number of Dynamic Bank Controller (BCINRD):  0 out of 6 (0%)
   Number of Dynamic Bank Controller (BCLVDSO):  0 out of 1 (0%)
   Number of Dynamic Bank Controller (BCSLEWRATEA):  0 out of 6 (0%)
   Number of DCCA:  0 out of 8 (0%)
   Number of DCMA:  0 out of 2 (0%)
   Number of PLLs:  0 out of 2 (0%)
   Number of DQSDLLs:  0 out of 2 (0%)
   Number of CLKDIVC:  0 out of 4 (0%)
   Number of ECLKSYNCA:  0 out of 4 (0%)
   Number of ECLKBRIDGECS:  0 out of 2 (0%)
```

Figure 10.1. MachXO3D Cyber Resilience Reference Design Full Device Utilization

Hierarchy---Post Map Resources

Unit	File	LUT4	PFU Registers	IO Registers	EBR	Distributed RAM	Carry Cells	SLICE
PFR_top	...p.v	7165(2)	4691(8)	3(3)	48(0)	99(1)	870(0)	4624(3.50)
XO3D_CRA_uniq_1(PFR_inst)	...A.v	7163(1)	4683(0)	0(0)	48(0)	98(0)	870(0)	4620.50(0.50)
uart0_uniq_1(uart0_inst)	...t0.v	184(0)	147(0)	0(0)	0(0)	0(0)	23(0)	120.20(0)
sysmem0_uniq_1(sysmem0_inst)	...0.v	954(0)	276(0)	0(0)	40(0)	98(0)	23(0)	643.27(0)
qspi_mst_streamer0_uniq_1(qspi_mst_streamer0_inst)	...r0.v	883(883)	940(940)	0(0)	2(2)	0(0)	240(240)	678.58(678....
qspi_monitor0_uniq_1(qspi_monitor0_inst)	...r0.v	1345.50(1345.50)	1080(1080)	0(0)	0(0)	0(0)	362(362)	996.78(996....
I2C_filter0_uniq_1(I2C_filter0_inst)	...r0.v	216.50(216.50)	156(156)	0(0)	2(2)	0(0)	15(15)	132.58(132....
gpio0_uniq_1(gpio0_inst)	...0.v	94(0)	72(0)	0(0)	0(0)	0(0)	0(0)	47.90(0)
esb_mux0_uniq_1(esb_mux0_inst)	...x0.v	90(90)	90(90)	0(0)	0(0)	0(0)	0(0)	46.33(46.33)
efb0_uniq_1(efb0_inst)	...0.v	9(0)	29(0)	0(0)	0(0)	0(0)	0(0)	12.42(0)
cs3_uniq_1(cs3_inst)	...s3.v	392(0)	547(0)	0(0)	0(0)	0(0)	0(0)	237.07(0)
cpu0_uniq_1(cpu0_inst)	...0.sv	2538.50(2538.50)	1183(1183)	0(0)	4(4)	0(0)	207(207)	1480.75(14...
apb1x8_uniq_1(apb1x8_inst)	...x8.v	141.50(141.50)	8(8)	0(0)	0(0)	0(0)	0(0)	69.67(69.67)
ahbl2apb0_uniq_1(ahbl2apb0_inst)	...0.sv	215(215)	148(148)	0(0)	0(0)	0(0)	0(0)	113.83(113....
ahbl1x2_uniq_1(ahbl1x2_inst)	...x2.v	99(99)	7(7)	0(0)	0(0)	0(0)	0(0)	40.62(40.62)

Figure 10.2. MachXO3D Cyber Resilience Reference Design Device Utilization Per Component

References

- Lattice Sentry Root of Trust Reference Design for MachXO3D User's Guide (FPGA-UG-02114)
- Lattice Sentry Demo Board for MachXO3D User's Guide (FPGA-UG-02097)
- Lattice Sentry Root of Trust Demo for MachXO3D User's Guide (FPGA-RD-02203)
- MachXO3D Platform Firmware Resiliency Manifest and Log Management (FPGA-UG-02025)
- Lattice I2C Filter IP Core web page
- Lattice PLD Interface IP Core web page
- Lattice SMBus Mailbox IP Core web page
- Lattice Sentry QSPI Streamer IP Core web page
- Lattice Sentry QSPI Monitor IP Core web page
- Lattice Insights 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 https://www.latticesemi.com/Support/AnswerDatabase.

Revision History

Revision 1.0, September 2026

Section	Change Summary
All	Initial release.

www.latticesemi.com