



Lattice Sentry 4.0 SCM and HPM Design Template for MachXO3D Devices

User Guide

FPGA-UG-02222-1.0

September 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

Contents	3
Abbreviations in This Document.....	6
1. Introduction	8
2. Block Diagram	9
3. Pin-out for SCM and HPM Design	10
3.1. HPM (Sentry™ 4.0 Board).....	10
3.2. SCM (LTPI Demonstration Board)	11
4. Functional Description	13
4.1. SoC Design Template.....	13
4.2. Firmware Template	14
4.2.1. Firmware Flow	14
4.3. Timing Constraints	18
4.3.1. Blocked Paths.....	18
4.3.2. Clock Definition	19
5. Hardware Bring-up	20
5.1. SCM – MachXO3D Breakout Board and LTPI Board	20
5.1.1. Setting Up the Hardware	20
5.1.2. Initial Connection	21
5.1.3. Uploading the Configuration File	21
5.2. HPM – Sentry 4.0 Board	23
5.2.1. Rework for LTPI	23
5.2.2. Initial Connections	24
5.2.3. Uploading the Configuration File (MachXO3D)	25
5.2.4. Uploading the Configuration File (MachXO5-NX)	27
5.3. Connecting the Two Boards	29
5.4. eSPI Controller Using Total Phase Promira Serial Platform (Optional for eSPI Demonstration).....	30
5.5. Total Phase Aardvark for I2C (Optional for LTPI I2C Aggregation)	32
6. IP Demonstration.....	35
6.1.1. LTPI Demonstration	35
6.1.2. eSPI Demonstration	38
6.1.3. M-PESTI Demonstration.....	39
6.1.4. SGPIO Demonstration for HPM.....	43
6.1.5. SGPIO Demonstration for SCM	45
7. IP User Guide Reference Documents	48
7.1. LTPI	48
7.1.1. Features	48
7.1.2. Reference Guide	48
7.2. eSPI Target	48
7.2.1. Features	48
7.2.2. Reference Guide	49
7.3. M-PESTI Initiator	49
7.3.1. Features	49
7.3.2. Reference Guide	49
7.4. SGPIO Controller and Target	49
7.5. RISC-V SM	49
7.5.1. Features	49
7.5.2. Reference Guide	49
8. Resource Utilization	50
9. Design Guidelines, Known Issues, and Limitation	51
9.1. LTPI	51
9.1.1. SoC Template (IP).....	51
9.1.2. Diamond Project	51

9.1.3. Firmware (Propel C Project).....	51
9.2. eSPI.....	52
9.3. M-PESTI	52
10. IP Versions	53
11. Design Template Package	54
References.....	55
Technical Support Assistance	56
Revision History	57

Figures

Figure 2.1. HPM Design Template Block Diagram	9
Figure 2.2. SCM Design Template Block Diagram	9
Figure 4.1. PLL and EFB RTL Instance.....	13
Figure 4.2. CSR Module for EFB and LTPI Control Signals	14
Figure 4.3. Firmware Flow Chart	15
Figure 4.4. Main Function Code.....	16
Figure 4.5. Instance Initialization Code.....	16
Figure 4.6. IP Initialization Code	16
Figure 4.7. CSR Initialization Code	17
Figure 4.8. M-PESTI Initialization Code.....	17
Figure 4.9. PLL Initialization Code.....	17
Figure 4.10. LTPI Status Checking Code	18
Figure 4.11. UART IP Demonstration Selection	18
Figure 4.12. Clock Definition.....	19
Figure 5.1. MachXO3D Breakout Board.....	20
Figure 5.2. LTPI Demonstration Board.....	20
Figure 5.3. MachXO3D Breakout Board (Soldered)	21
Figure 5.4. Assembled LTPI Demonstration Board	21
Figure 5.5. Open SCM_soc in Lattice Diamond Software	22
Figure 5.6. JED File Upload Using Flash Programming Mode (SCM).....	22
Figure 5.7. BIT File Upload Using Static RAM Cell Mode (SCM).....	23
Figure 5.8. LTPI Board Rework	24
Figure 5.9. Initial Connections	25
Figure 5.10. MachXO3D Configuration Jumpers.....	25
Figure 5.11. Open HPM_soc in the Lattice Diamond Software	26
Figure 5.12. JED File Upload Using Flash Programming Mode (HPM)	27
Figure 5.13. BIT File Upload Using Static RAM Cell Mode (HPM)	27
Figure 5.14. MachXO5-NX Configuration Jumpers	28
Figure 5.15. MachXO5-NX Configuration.....	29
Figure 5.16. Demonstration Setup.....	29
Figure 5.17. eSPI Pin Connection	30
Figure 5.18. API for eSPI Transactions	30
Figure 5.19. Promira Port IP Address Detection	31
Figure 5.20. Running in IDLE Shell	31
Figure 5.21. IDLE Shell Output	32
Figure 5.22. Aardvark I2C/SPI Host Adapter	32
Figure 5.23. SCM Project Design Spreadsheet.....	33
Figure 5.24. MachXO3D Breakout Board – Debug Pinout and Aardvark Pinouts	33
Figure 5.25. HPM Project Design Spreadsheet	33
Figure 5.26. Sentry 4.0 board – J11 RPi Header and Aardvark Pinouts	34
Figure 5.27. Aardvark device ID	34
Figure 5.28. Control Center Adapter Configuration	34
Figure 6.1. DC-SCM 2.0 LTPI board and Sentry 4.0 Demonstration Board	35

Figure 6.2. LED Indicator for LTPI Connection	35
Figure 6.3. UART Terminal upon HPM Power On	36
Figure 6.4. LTPI Demonstration Function	36
Figure 6.5. Aardvark-Controller Setting.....	36
Figure 6.6. Aardvark-Target Setting.....	36
Figure 6.7. Aardvark Target Response	37
Figure 6.8. Aardvark Controller	37
Figure 6.9. Aardvark Target	38
Figure 6.10. eSPI Demonstration Diagram	38
Figure 6.11. VW Channel Set Configuration Pseudo Code	39
Figure 6.12. Terminal Print Output.....	39
Figure 6.13. M-PESTI Demonstration Block Diagram	40
Figure 6.14. M-PESTI Demonstration Jumpers	40
Figure 6.15. Power and Data Connections	41
Figure 6.16. Demonstration Menu Screen	41
Figure 6.17. D24 LED Blink	42
Figure 6.18. M-PESTI Demonstration Terminal	42
Figure 6.19. HPM SGPIO Demonstration Diagram.....	43
Figure 6.20. RPi Header Jumper Settings.....	43
Figure 6.21. UART SGPIO Demonstration Selection	44
Figure 6.22. SGPIO Demonstration Code.....	44
Figure 6.23. SGPIO Interrupt Initialization.....	44
Figure 6.24. SGPIO Interrupt Callback Function	44
Figure 6.25. SCM SGPIO Demonstration Diagram	45
Figure 6.26. RPi Header Jumper Settings.....	45
Figure 6.27. UART SGPIO Demonstration Selection	46
Figure 6.28. SGPIO Demonstration Code.....	46
Figure 6.29. SGPIO Interrupt Initialization.....	46
Figure 6.30. SGPIO Interrupt Callback Function	47
Figure 9.1. Capabilities Match	51
Figure 9.2. Enable Automatically Move to Configuration State	51
Figure 9.3. File Name Extension Manual Edit	52
Figure 9.4. LFMX05-55TD Programmer Settings.....	52
Figure 11.1. Sentry 4.0 Design Package	54

Tables

Table 3.1. HPM Pin-out for Sentry 4.0 Board (MachXO3D)	10
Table 3.2. SCM Pin-out for LTPI Demonstration Board (MachXO3D)	11
Table 5.1. Jumper Settings for MachXO3D Configuration	26
Table 5.2. Jumper Settings for MachXO5-NX Configuration.....	28
Table 8.1. Resource Utilization	50
Table 10.1. IP Versions.....	53

Abbreviations in This Document

A list of Abbreviations used in this document.

Abbreviation	Definition
ACK	Acknowledge bit sent from RX side to TX side to indicate the received data parity check is OK
AHB-Lite	Advanced High-performance Bus-Lite
APB	Advanced Peripheral Bus
API	Application Programming Interface
CPLD	Complex Programmable Logic Device
CSR	Control Status Register
DAC	Digital to Analog Converter
DC-MHS	Data Center-Modular Hardware System
DC-SCM	Data Center-Secure Control Module
DDR	Double Data Rate
EFB	Embedded Function Block
eSPI	Enhanced Serial Peripheral Interface
FIFO	First In First Out
FTDI	Future Technology Devices International
FW	Firmware
GPIO	General Purpose Inputs and Outputs
GUI	Graphical User Interface
HBA	Host Bus Adapter
HPM	Host Processing Module
I2C	Inter-Integrated Circuit
I2S	Inter-IC Sound
IP	Intellectual Property
LCIP	Lock Control IP
LED	Light Emitting Diode
LTPI	LVDS Tunneling Protocol and Interface
M-PESTI	Modular-Peripheral Sideband Tunneling Interface
OCP	Open Compute Project
OEM	Original Equipment Manufacturer
OOB	Out of Band
PIC	Programmable Interrupt Controller
PID	Payload ID
PLL	Phase Locked Loop
PT	Payload Type
RX	Receiver
RISC-V	Reduced Instruction Set Computer – V (Five)
RTL	Register Transfer Level
SCM	Secure Control Module
SDR	Single Data Rate
SGPIO	Serial General Purpose Input/Output
SM	State Machine
SoC	System on Chip
SubLVDS	(Reduced Voltage) Low Voltage Differential Signaling
TDM	Time Domain Multiplexing
TX	Transmitter
UART	Universal Asynchronous Receiver/Transmitter

Abbreviation	Definition
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit

1. Introduction

This document introduces a System on Chip (SoC)-based solution template that uses Lattice connectivity and peripheral IPs for Secure Control Module (SCM) and Host Processing Module (HPM) CPLD design applications. It aims to help you design an SCM/HPM CPLD design with limited knowledge and experience in Verilog/VHDL. This package template includes the following collaterals.

- The SoC design project on which you can open, view, and modify the design through the Lattice Propel™ Builder software. The bitstream can be generated using the Lattice Diamond™ software.
- The SoC workspace, which includes necessary drivers for the connectivity and peripheral IPs and firmware for the demonstration.
- The CPLD demonstration bitstream

The current template is designed and verified using the Lattice MachXO3D™ device (LCMXO3D-9400-5BG484C). This solution is OCP Ready™.

2. Block Diagram

Figure 2.1 and Figure 2.2 show the functional block diagram of SCM and HPM Design Template. For demonstration and debug purposes, miscellaneous IPs, such as GPIO0-2 and UART, are instantiated on the system design template. Two PLL modules are also included to support the LVDS Tunneling Protocol and Interface (LTPI) dynamic PLL clock feature.

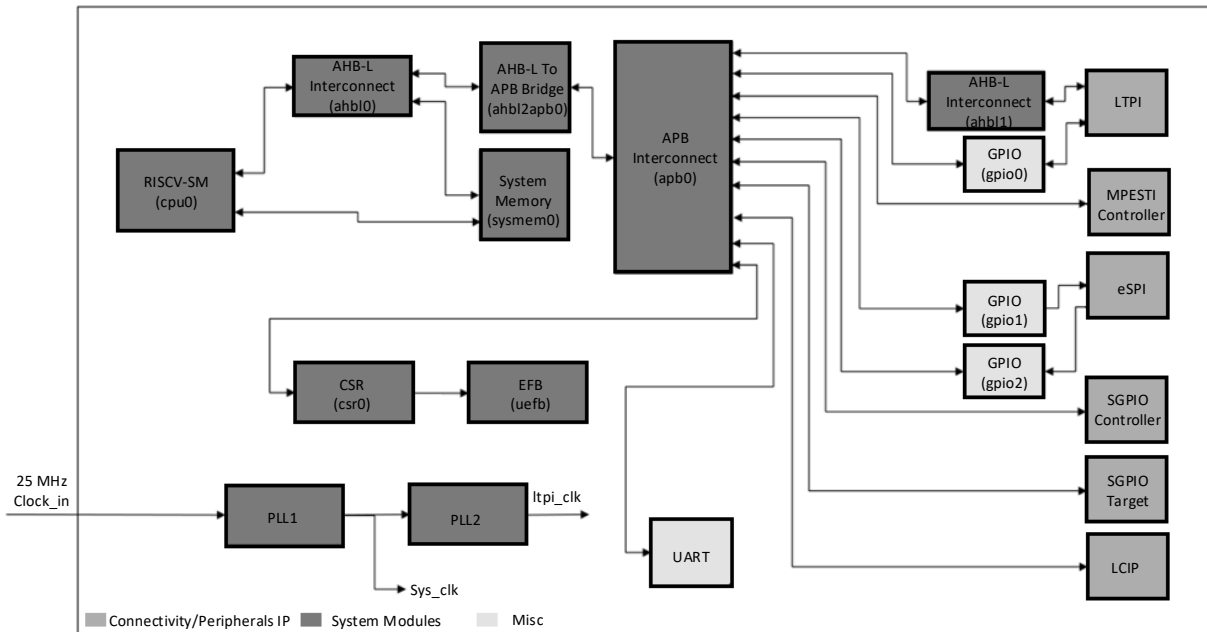


Figure 2.1. HPM Design Template Block Diagram

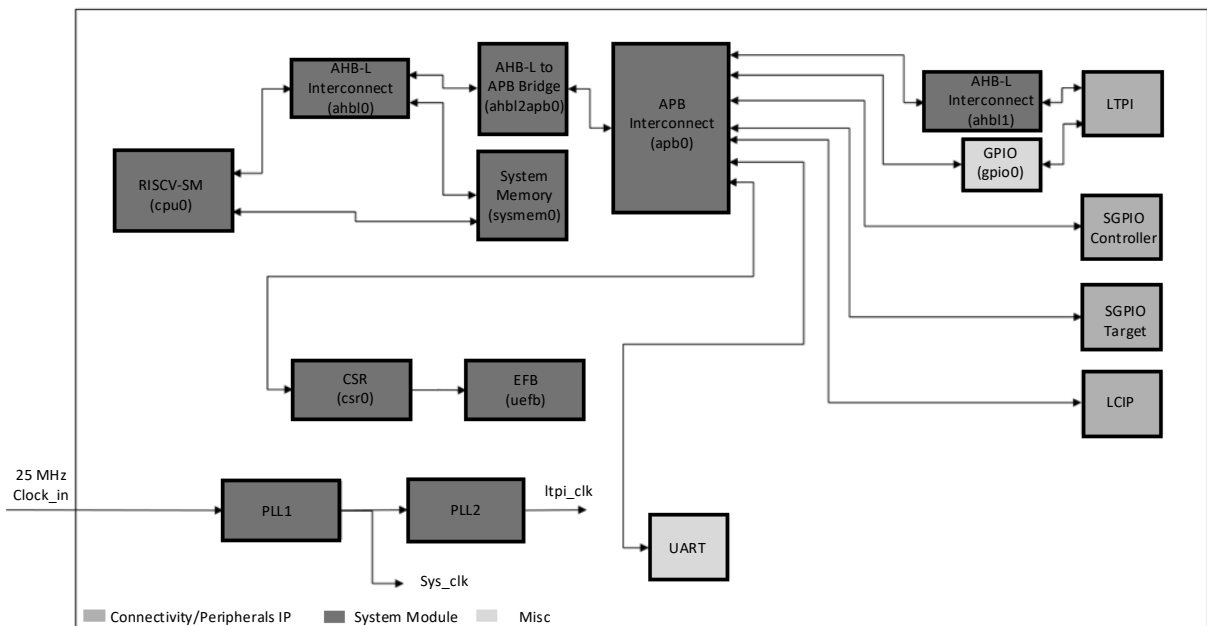


Figure 2.2. SCM Design Template Block Diagram

3. Pin-out for SCM and HPM Design

The [HPM \(Sentry™ 4.0 Board\)](#) and [SCM \(LTPI Demonstration Board\)](#) sections describe the pin-out for the MachXO3D device used for the demonstration.

3.1. HPM (Sentry™ 4.0 Board)

Table 3.1. HPM Pin-out for Sentry 4.0 Board (MachXO3D)

Port Name	Port Map	Sentry 4.0 Map	Description	Note
Terminal Access				
uart0_inst_rxd_o_port	M1	JP1 (1)	Terminal UART RX	Short JP1 for UART access via FTDI.
uart0_inst_txd_o_port	N1	JP2 (2)	Terminal UART TX	Short JP2 for UART access via FTDI.
LTPI				
ltpi0_inst_uart_rx_o_portbus[0]	D6	J11 (35)	CH0 UART RX	—
ltpi0_inst_uart_tx_i_portbus[0]	B4	J11 (33)	CH0 UART TX	—
ltpi0_inst_lvds_tx_data_o_port	F10 & E10	J23 (9 & 5)	LVDS TX Data	Requires rework. See the HPM – Sentry 4.0 Board section for details.
ltpi0_inst_lvds_tx_clk_o_port	B11 & A11	J16-1 (A24 & A23)	LVDS TX Clock	—
ltpi0_inst_lvds_rx_data_i_port	AA11 & AB11	J16-1 (B21 & B20)	LVDS RX Data	—
ltpi0_inst_lvds_rx_clk_i_port	AA10 & AB10	J16-1 (B23 & B24)	LVDS RX Clock	—
ltpi0_inst_ll_gpio_o_portbus[0]	T1	X3_LED_0 (D55)	Low Latency Output 0	—
ltpi0_inst_ll_gpio_o_portbus[1]	U1	X3_LED_1 (D56)	Low Latency Output 1	—
ltpi0_inst_ll_gpio_o_portbus[2]	R2	X3_LED_2 (D57)	Low Latency Output 2	—
ltpi0_inst_ll_gpio_o_portbus[3]	T2	X3_LED_3 (D58)	Low Latency Output 3	—
ltpi0_inst_ll_gpio_o_portbus[4]	R3	X3_LED_4 (D59)	Low Latency Output 4	—
ltpi0_inst_ll_gpio_o_portbus[5]	T3	X3_LED_5 (D60)	Low Latency Output 5	—
ltpi0_inst_ll_gpio_o_portbus[6]	R4	X3_LED_6 (D61)	Low Latency Output 6	—
ltpi0_inst_ll_gpio_o_portbus[7]	T4	X3_LED_7 (D62)	Low Latency Output 7	—
ltpi0_inst_i2c_scl_io_portbus[0]	C6	J11 (16)	CH0 I2C SCL	—
ltpi0_inst_i2c_sda_io_portbus[0]	D7	J11 (29)	CH0 I2C SDA	—
ltpi0_inst_i2c_scl_io_portbus[1]	A3	J11 (38)	CH1 I2C SCL	—
ltpi0_inst_i2c_sda_io_portbus[1]	C3	J11 (18)	CH1 I2C SDA	—
ltpi0_inst_i2c_scl_io_portbus[2]	E7	J11 (31)	CH2 I2C SCL	—
ltpi0_inst_i2c_sda_io_portbus[2]	B6	J11 (40)	CH2 I2C SDA	—
eSPI				
espi0_inst_espi_alert_n_o_port	A7	J11 (15)	Alert Signal	Active Low
espi0_inst_espi_data_io_portbus[0]	A4	J11 (19)	Data 0	—
espi0_inst_espi_data_io_portbus[1]	F8	J11 (21)	Data 1	—
espi0_inst_espi_data_io_portbus[2]	B1	J11 (23)	Data 2	—
espi0_inst_espi_data_io_portbus[3]	G8	J11 (27)	Data 3	—
espi0_inst_espi_cs_n_i_port	B3	J11 (24)	Chip Select	Active Low
espi0_inst_espi_clk_i_port	B10	J11 (32)	Clock	—

Port Name	Port Map	Sentry 4.0 Map	Description	Note
M-PESTI				
mpesti0_inst_mpesti_portbus[0]	K2	JP27 (2)	M-PESTI Bus 0	Short JP27 for the demonstration.
mpesti0_inst_mpesti_portbus[1]	L6	JP29 (2)	M-PESTI Bus 1	Short JP29 for the demonstration
gate_o[0]	K3	—	Bus Gate 0	For Bus 0 control
gate_o[1]	K1	—	Bus Gate 1	For Bus 1 control
SGPIO				
sgpios0_inst_iGSXDataIn_port	E6	J11 (3)	SGPIO Target Data In	Connect with Controller Data Out for the demonstration.
sgpios0_inst_oGSXDataOut_port	B5	J11 (11)	SGPIO Target Data Out	Connect with Controller Data In for the demonstration.
sgpios0_inst_iGSXClk_port	B2	J11 (7)	SGPIO Target Clock	Connect with Controller Clock for the demonstration.
sgpios0_inst_inGSXLoad_port	D5	J11 (10)	SGPIO Target Load	Connect with Controller Load for the demonstration.
sgpiom0_inst_sdatain_i_port	B8	J11 (13)	SGPIO Controller Data In	Connect with Target Data Out for the demonstration.
sgpiom0_inst_sdataout_o_port	F7	J11 (5)	SGPIO Controller Data Out	Connect with Target Data In for the demonstration.
sgpiom0_inst_sclock_o_port	A5	J11 (8)	SGPIO Controller Clock	Connect with Target Clock for the demonstration.
sgpiom0_inst_sload_o_port	A6	J11 (12)	SGPIO Controller Load	Connect with Target Load for the demonstration.

3.2. SCM (LTPI Demonstration Board)

Table 3.2. SCM Pin-out for LTPI Demonstration Board (MachXO3D)

Port Name	Port Map	Sentry 4.0 Map	Description	Note
Terminal Access				
uart0_inst_rxd_o_port	F8	JP1 (1)	Terminal UART RX	—
uart0_inst_txd_o_port	L14	JP2 (2)	Terminal UART TX	—
LTPI				
ltpi0_inst_uart_rx_o_portbus[0]	E7	DEBUG_0 (3)	CH0 UART RX	—
ltpi0_inst_uart_tx_i_portbus[0]	M16	DEBUG_0 (4)	CH0 UART TX	—
ltpi0_inst_lvds_tx_data_o_port	A5 & B6	J1 (A20 & A22)	LVDS TX Data	—
ltpi0_inst_lvds_tx_clk_o_port	D10 & E10	J1 (A24 & A23)	LVDS TX Clock	—
ltpi0_inst_lvds_rx_data_i_port	T11 & P11	J1 (B21 & B20)	LVDS RX Data	—
ltpi0_inst_lvds_rx_clk_i_port	T9 & P9	J1 (B23 & B24)	LVDS RX Clock	—
ltpi0_inst_ll_gpio_o_portbus[0]	H11	LED0 (D9)	Low Latency Output 0	—
ltpi0_inst_ll_gpio_o_portbus[1]	J13	LED1 (D8)	Low Latency Output 1	—
ltpi0_inst_ll_gpio_o_portbus[2]	J11	LED2 (D7)	Low Latency Output 2	—

Port Name	Port Map	Sentry 4.0 Map	Description	Note
ltpi0_inst_ll_gpio_o_portbus[3]	L12	LED3 (D6)	Low Latency Output 3	—
ltpi0_inst_ll_gpio_o_portbus[4]	K11	LED4 (D5)	Low Latency Output 4	—
ltpi0_inst_ll_gpio_o_portbus[5]	L13	LED5 (D4)	Low Latency Output 5	—
ltpi0_inst_ll_gpio_o_portbus[6]	N15	LED6 (D3)	Low Latency Output 6	—
ltpi0_inst_ll_gpio_o_portbus[7]	P16	LED7 (D2)	Low Latency Output 7	—
ltpi0_inst_i2c_scl_io_portbus[0]	D8	DEBUG_0 (13)	CH0 I2C SCL	—
ltpi0_inst_i2c_sda_io_portbus[0]	L16	DEBUG_0 (14)	CH0 I2C SDA	—
ltpi0_inst_i2c_scl_io_portbus[1]	E9	DEBUG_0 (15)	CH1 I2C SCL	—
ltpi0_inst_i2c_sda_io_portbus[1]	M15	DEBUG_0 (16)	CH1 I2C SDA	—
ltpi0_inst_i2c_scl_io_portbus[2]	D6	DEBUG_0 (17)	CH2 I2C SCL	—
ltpi0_inst_i2c_sda_io_portbus[2]	N14	DEBUG_0 (18)	CH2 I2C SDA	—
SGPIO				
sgpios0_inst_iGSXDataIn_port	P7	DEBUG_0 (34)	SGPIO Target Data In	Connect with Controller Data Out for the demonstration.
sgpios0_inst_oGSXDataOut_port	M11	DEBUG_0 (38)	SGPIO Target Data Out	Connect with Controller Data In for the demonstration.
sgpios0_inst_iGSXClk_port	R7	DEBUG_0 (32)	SGPIO Target Clock	Connect with Controller Clock for the demonstration.
sgpios0_inst_inGSXLoad_port	N10	DEBUG_0 (36)	SGPIO Target Load	Connect with Controller Load for the demonstration.
sgpiom0_inst_sdatain_i_port	J14	DEBUG_0 (33)	SGPIO Controller Data In	Connect with Target Data Out for the demonstration.
sgpiom0_inst_sdataout_o_port	K15	DEBUG_0 (37)	SGPIO Controller Data Out	Connect with Target Data In for the demonstration.
sgpiom0_inst_sclock_o_port	H15	DEBUG_0 (31)	SGPIO Controller Clock	Connect with Target Clock for the demonstration.
sgpiom0_inst_sload_o_port	K16	DEBUG_0 (35)	SGPIO Controller Load	Connect with Target Load for the demonstration.

4. Functional Description

This section walks you through the SoC design template, the firmware template, and the timing constraints.

4.1. SoC Design Template

The template uses a RISC-V SM core to facilitate control over a system consisting of various connectivity and peripheral IPs. The template is mainly patterned to currently known use cases of DC-SCM and HPM systems.

The template package consists of an SCM and HPM template using a RISC-V SM core. The following connectivity and peripheral IPs are enabled on each template.

- SCM
 - SGPIO Controller
 - SGPIO Target
 - LTPI (SCM)
 - LCIP¹
- HPM
 - SGPIO Controller
 - SGPIO Target
 - LTPI (HPM)
 - LCIP¹
 - eSPI Target
 - M-PESTI Initiator

Note:

1. LCIP is to be included in the next release.

The design also uses RTL-based instances for the PLLs and EFB since these are not available on Lattice Propel by default. The RTL used is a placeholder for the IPX files that are added to the Lattice Diamond project. This allows you to modify the settings of the PLL and EFB using the Diamond software (Figure 4.1).

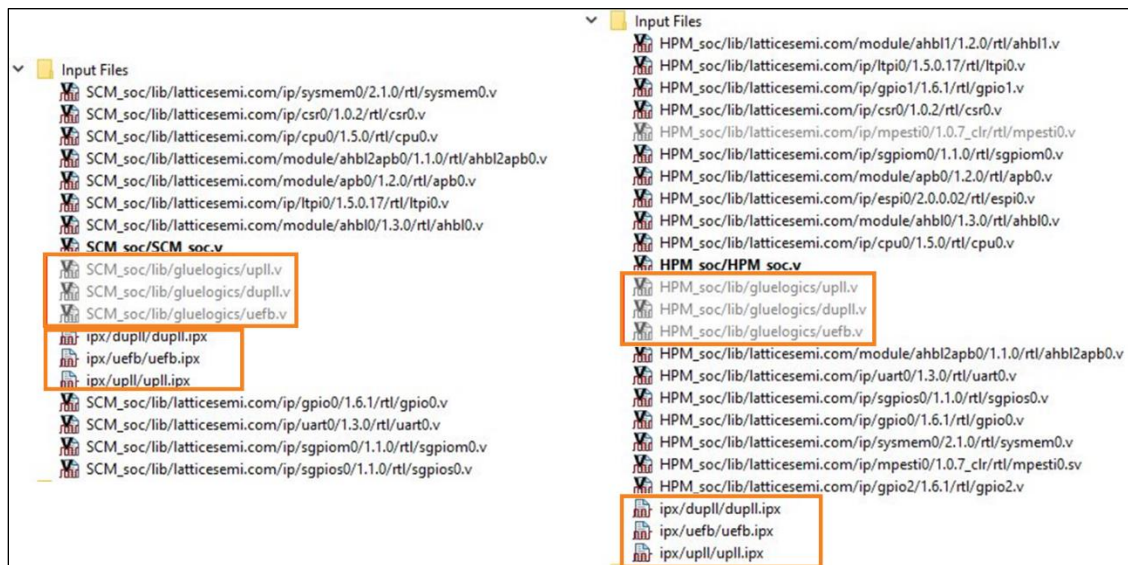


Figure 4.1. PLL and EFB RTL Instance

A register file module (csr0.v) is also used to facilitate translation of discrete EFB and LTPI control signals to a register access implementation through APB (Figure 4.2).

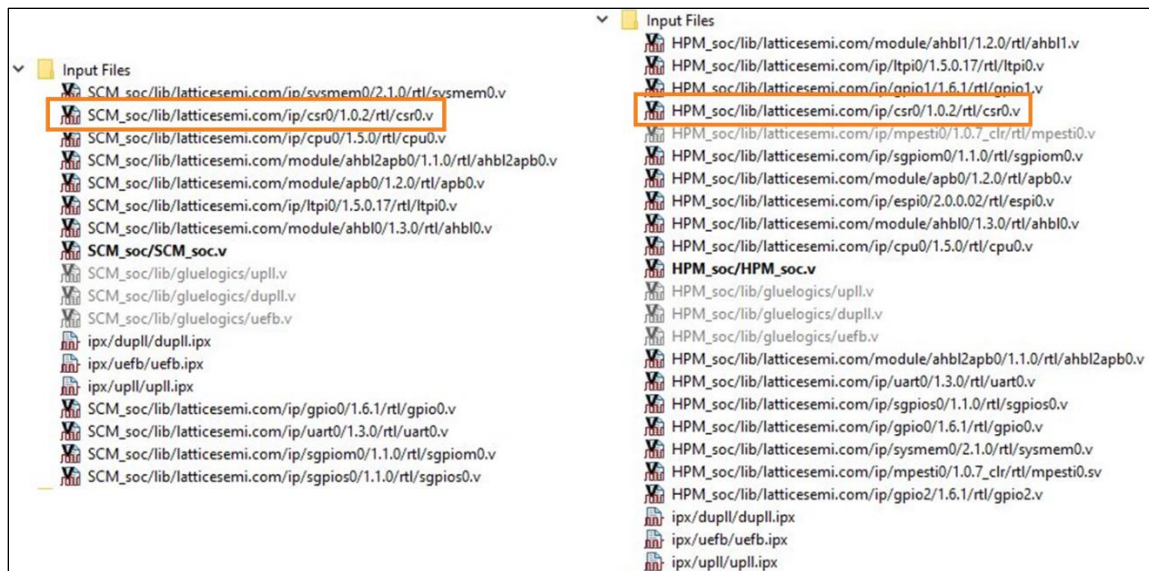


Figure 4.2. CSR Module for EFB and LTPI Control Signals

Modifying the design template can be done through Lattice Propel Builder and the Lattice Diamond software. Refer to the following web pages for more information.

- Lattice Propel Design Environment:
<https://www.latticesemi.com/products/designsoftwareandip/fpgaandids/latticepropel>
- Lattice Diamond Software:
<https://www.latticesemi.com/products/designsoftwareandip/fpgaandids/latticediamond>

4.2. Firmware Template

Sentry 4.0 HPM template firmware includes demonstration of different IPs for server segments. LTPI, SGPIO, M-PESTI Initiator, and eSPI Target are the IPs to be demonstrated based on your selection through the UART terminal.

4.2.1. Firmware Flow

4.2.1.1. Main Function

The flowchart (Figure 4.3) and the code (Figure 4.4) below show the flow of the firmware and c workspace for HPM SoC design.

1. All the instances are initialized to make all the instances global and can be used in the whole workspace.
2. Each IP is initialized together with its corresponding interrupt with a callback function.
3. After the initialization of the IPs, PLL initialization is required for LTPI.
4. Upon PLL initialization, LTPI statuses are checked to confirm the linking of the IP.
5. You can choose the demonstration you want to perform after all the initializations. Each IP demonstration has its corresponding routine to perform.

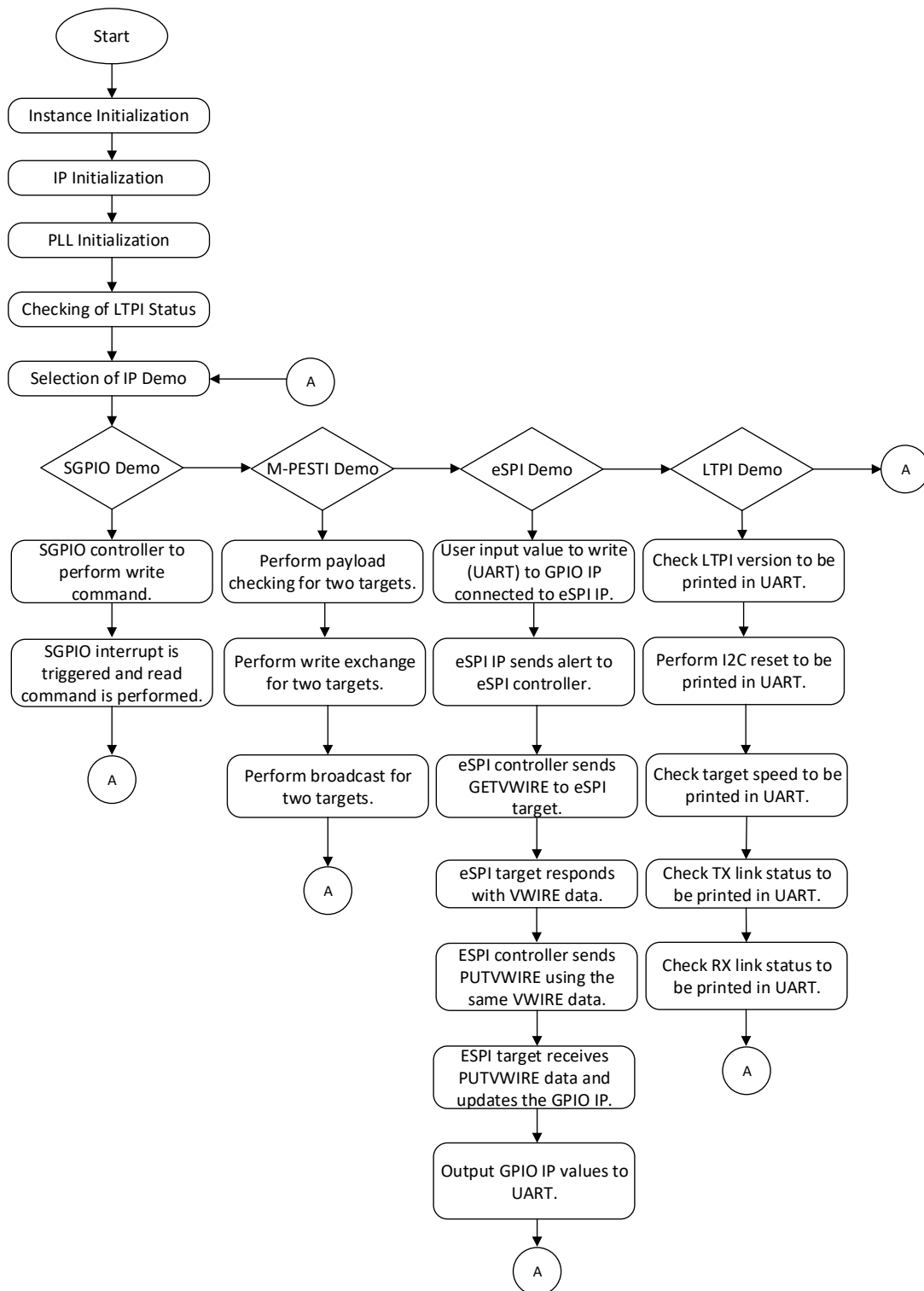


Figure 4.3. Firmware Flow Chart

```

513 int main(void)
514 {
515     soc_init(soc0_inst_p);
516     system_init();
517     pll_init(soc0_inst_p->csr_inst, soc0_inst_p->ltpi_inst->op_freq);
518     while (soc0_inst_p->print_init == 1) {
519         printf(" ");
520     }
521     ltpi_VERSION(soc0_inst_p->ltpi_inst); //LTPI IP specs
522     ltpi_TGT_SPEED(soc0_inst_p->ltpi_inst); //Cap0/Cap1 included
523     ltpi_TX_STATE(soc0_inst_p->ltpi_inst); //Link status
524     ltpi_RX_STATE(soc0_inst_p->ltpi_inst); //Link status
525
526     demo_selection();
527     while (true) {
528     }
529     return 0;
530 }
531

```

Figure 4.4. Main Function Code

4.2.1.2. Instance Initialization (soc_init)

This function initializes the instances to be used in the c workspace or firmware (Figure 4.5).

```

287 void soc_init(struct soc_instance *soc_inst) {
288     soc_inst->csr_inst = &csr0_inst;
289     soc_inst->ltpi_inst = &ltpi0_inst;
290     soc_inst->timer_inst = &timer0_inst;
291     soc_inst->delay = 1;
292     soc_inst->print_init = 1;
293     soc_inst->counter = 0;
294 }
295

```

Figure 4.5. Instance Initialization Code

4.2.1.3. IPs Initialization (system_init)

This function initializes the IPs in the HPM design. The initialization of each IP interrupt is also performed through this function (Figure 4.6 and Figure 4.7).

```

476 void system_init() {
477     //Initialization of different IPs for HPM (GPIO, LTPI UART, MPESTI, SGPIO and RISCv)
478     pic_init(CPU0_INST_BASE_ADDR);
479     ltpi_init(soc0_inst_p->ltpi_inst, LTPI0_INST_BASE_ADDR);
480     csr_init(soc0_inst_p->csr_inst, CSR0_INST_BASE_ADDR);
481     pic_isr_register(LTPI0_INST_IRQ, ltpi_isr, (void *) soc0_inst_p);
482     pic_isr_register(CSR0_INST_IRQ, csr_isr, (void *) soc0_inst_p);
483     //Initialize UART
484     uart_init(&uart_core_uart, UART_INST_BASE_ADDR, UART_INST_SYS_CLK * 1000000, UART_INST_BAUD_RATE, 1, 8);
485     iob_init(lsc_uart_putc, lsc_uart_getc, lsc_uart_flush);
486     //Initialize GPIO
487     gpio_inst.instance_name = GPIO1_INST_NAME;
488     gpio_init(&gpio_inst, GPIO1_INST_BASE_ADDR, GPIO1_INST_LINES_NUM, GPIO1_INST_GPIO_DIRS);
489     //Initialize GPIO
490     gpio_inst0.instance_name = GPIO0_INST_NAME;
491     gpio_init(&gpio_inst0, GPIO0_INST_BASE_ADDR, GPIO0_INST_LINES_NUM, GPIO0_INST_GPIO_DIRS);
492     //Initialize GPIO
493     gpio_inst2.instance_name = GPIO2_INST_NAME;
494     gpio_init(&gpio_inst2, GPIO2_INST_BASE_ADDR, GPIO2_INST_LINES_NUM, GPIO2_INST_GPIO_DIRS);
495     //Initialization of MPESTI initiator and targets
496     mpesti_init();
497     //Initializing sgpio master with base address 0x00133000
498     sgpio_init(&sgpio_inst, SGPIOM0_INST_BASE_ADDR, 0, 0);
499     //To register the callback function for sgpio
500     sgpio_register_callback(&sgpio_inst, sgpio_callback, NULL);
501     //Register th ISR for SGPIO
502     pic_isr_register(SGPIOM0_INST_IRQ, sgpio_callback, (void *)&sgpio_inst);
503     //Enable the interrupt for SGPIO
504     sgpio_enable_interrupt(&sgpio_inst, 1);
505     //Initialiation of timer
506     timer_init(soc0_inst_p->timer_inst, CPU0_INST_BASE_ADDR, 25000);
507     timer_start(soc0_inst_p->timer_inst, delay_isr, (void *) soc0_inst_p, true, 100000);
508 }
509
510 }
511

```

Figure 4.6. IP Initialization Code

```

342=void csr_isr(void *context) {
343     unsigned int data;
344     struct soc_instance *ctx = (struct soc_instance *) context;
345     reg_32b_read(ctx->csr_inst->base_address | CSR_INT_STATUS, &data);
346     reg_32b_write(ctx->csr_inst->base_address | CSR_INT_STATUS, data);
347     switch (data) {
348         case CSR_INT_SYNC_RDY_IDX:
349             if (ctx->ltpi_inst->resync == 1)
350             {
351                 ctx->ltpi_inst->resync = 0;
352                 reg_32b_modify(ctx->csr_inst->base_address | CSR_CONTROL, CSR_CONTROL_START_RX_IDX | CSR_CONTROL_START_TX_IDX | CSR_CONTROL_START_RX_IDX | CSR_CONTROL_START_TX_IDX);
353             }
354             break;
355         case CSR_INT_PLL_LOCK_IDX:
356             reg_32b_modify(ctx->csr_inst->base_address | CSR_CONTROL, CSR_CONTROL_SYNC_START_IDX | CSR_CONTROL_SYNC_RST_IDX, CSR_CONTROL_SYNC_START_IDX | CSR_CONTROL_SYNC_RST_IDX);
357             break;
358         default:
359             break;
360     }
361 }
362

```

Figure 4.7. CSR Initialization Code

M-PESTI initialization has an additional process since the M-PESTI targets are placed in an external component, the MachXO5™-NX device. The timing of reset is time-sensitive and is triggered by the GPIO connected from the MachXO3D device to the MachXO5-NX device (Figure 4.8).

```

194=void mpesti_init() {
195     static uint8_t idx = 0;
196     static uint8_t pin_state = 0xFF;
197     // Performing reset to X05 M-PESTI targets
198     gpio_output_write_2(&gpio_inst, 0x00);
199     gpio_output_write_2(&gpio_inst, 0x01);
200
201     uint8_t mpesti_init_result;
202
203     // Initialization MPESTI Initiator
204     mpesti_init_result = mpesti_initiator_init(&mpesti_inst, MPESTI0_INST_BASE_ADDR); //AISLA: MPESTI0_INST_BASE_ADDR
205     break;
206 }
207

```

Figure 4.8. M-PESTI Initialization Code

4.2.1.4. PLL Initialization (pll_init)

Figure 4.9 shows the PLL initialization code.

```

295=void pll_init(struct csr_instance *this_csr, enum operating_frequency freq) {
296
297     //Initialization of PLL for LTPI
298     reg_32b_modify(this_csr->base_address | CSR_CONTROL, CSR_CONTROL_SYNC_START_IDX | CSR_CONTROL_SYNC_RST_IDX | CSR_CONTROL_PLL_RESET_IDX, 0x00);
299     switch (freq) {
300         case FREQ_255:
301             reg_32b_write(this_csr->base_address | CSR_PLL_02, PLL_02_255);
302             reg_32b_write(this_csr->base_address | CSR_PLL_03, PLL_03_255);
303             reg_32b_write(this_csr->base_address | CSR_PLL_04, PLL_04_255);
304             reg_32b_write(this_csr->base_address | CSR_PLL_05, PLL_05_255);
305             reg_32b_write(this_csr->base_address | CSR_PLL_06, PLL_06_255);
306             reg_32b_write(this_csr->base_address | CSR_PLL_07, PLL_07_255);
307             reg_32b_write(this_csr->base_address | CSR_PLL_08, PLL_08_255);
308             reg_32b_write(this_csr->base_address | CSR_PLL_09, PLL_09_255);
309             reg_32b_write(this_csr->base_address | CSR_PLL_0A, PLL_0A_255);
310             reg_32b_write(this_csr->base_address | CSR_PLL_0E, PLL_0E_255);
311             break;
312
313         case FREQ_1000:
314             reg_32b_write(this_csr->base_address | CSR_PLL_02, PLL_02_1000);
315             reg_32b_write(this_csr->base_address | CSR_PLL_03, PLL_03_1000);
316             reg_32b_write(this_csr->base_address | CSR_PLL_04, PLL_04_1000);
317             reg_32b_write(this_csr->base_address | CSR_PLL_05, PLL_05_1000);
318             reg_32b_write(this_csr->base_address | CSR_PLL_06, PLL_06_1000);
319             reg_32b_write(this_csr->base_address | CSR_PLL_07, PLL_07_1000);
320             reg_32b_write(this_csr->base_address | CSR_PLL_08, PLL_08_1000);
321             reg_32b_write(this_csr->base_address | CSR_PLL_09, PLL_09_1000);
322             reg_32b_write(this_csr->base_address | CSR_PLL_0A, PLL_0A_1000);
323             reg_32b_write(this_csr->base_address | CSR_PLL_0E, PLL_0E_1000);
324             break;
325
326         default:
327             reg_32b_write(this_csr->base_address | CSR_PLL_02, PLL_02_255);
328             reg_32b_write(this_csr->base_address | CSR_PLL_03, PLL_03_255);
329             reg_32b_write(this_csr->base_address | CSR_PLL_04, PLL_04_255);
330             reg_32b_write(this_csr->base_address | CSR_PLL_05, PLL_05_255);
331             reg_32b_write(this_csr->base_address | CSR_PLL_06, PLL_06_255);
332             reg_32b_write(this_csr->base_address | CSR_PLL_07, PLL_07_255);
333             reg_32b_write(this_csr->base_address | CSR_PLL_08, PLL_08_255);
334             reg_32b_write(this_csr->base_address | CSR_PLL_09, PLL_09_255);
335             reg_32b_write(this_csr->base_address | CSR_PLL_0A, PLL_0A_255);
336             reg_32b_write(this_csr->base_address | CSR_PLL_0E, PLL_0E_255);
337             break;
338     }
339     reg_32b_modify(this_csr->base_address | CSR_CONTROL, CSR_CONTROL_PLL_RESET_IDX, CSR_CONTROL_PLL_RESET_IDX);
340 }

```

Figure 4.9. PLL Initialization Code

4.2.1.5. LTPI Status

Figure 4.10 shows the LTPI status checking code.

```

469=int main(void)
470 {
471     soc_init(soc_inst_p);
472     system_init();
473     pll_init(soc_inst_p->csr_inst,soc_inst_p->ltpi_inst->op_freq);
474     while (soc_inst_p->print_init == 1) {
475         printf(" ");
476     }
477     ltpi_VERSION(soc_inst_p->ltpi_inst); //LTPI IP specs
478     ltpi_TGT_SPEED(soc_inst_p->ltpi_inst); //Cap0/Cap1 included
479     ltpi_TX_STATE(soc_inst_p->ltpi_inst); //Link status
480     ltpi_RX_STATE(soc_inst_p->ltpi_inst); //Link status
481
482     demo_selection();
483     while (true) {
484     }
485     return 0;
486 }

```

Figure 4.10. LTPI Status Checking Code

4.2.1.6. IP Demonstration Selection (demo_selection)

You can choose a particular demonstration to run (Figure 4.11). Each IP demonstration has its corresponding routine and functionalities.

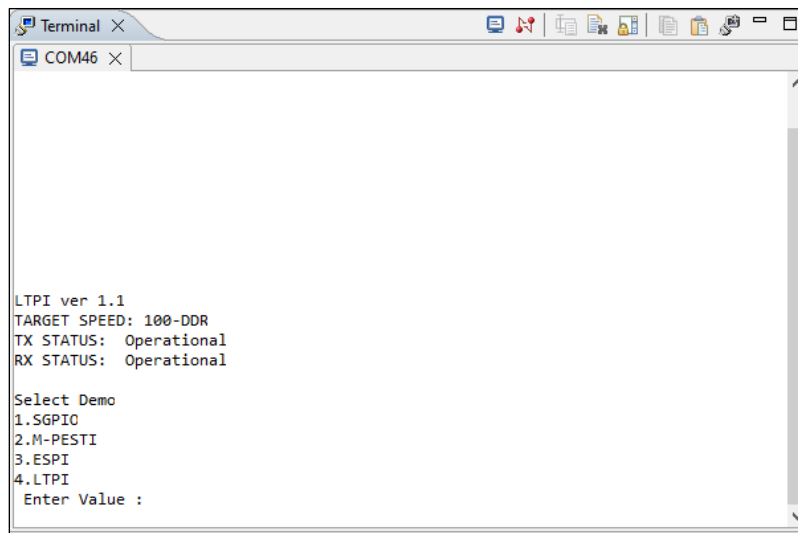


Figure 4.11. UART IP Demonstration Selection

4.3. Timing Constraints

The following constraints are already defined on the default SoC template package.

4.3.1. Blocked Paths

The following are clock sources that are used in blocked path constraints:

- System Clock: upll_inst_CLKOS_net
- LTPI Sync Clock: upll_inst_CLKOS2_net
- LTPI Clock: dupll_inst_CLKOS2_net
- LTPI TX Divider Clock:
ltpi0_inst/lscm_dcscm_ltpi_inst/lscm_tx_path_inst/xo3_tx_PHY.lscm_phy_tx_xo3_inst/lscm_xo3_ddrx4_tx_inst/ecldk
- LTPI RX PHY Clock: ltpi0_inst/lscm_dcscm_ltpi_inst/lscm_rx_path_inst/xo3_rx_PHY.lscm_phy_rx_xo3_inst/rx_sclk_o
- LTPI TX PHY Clock: ltpi0_inst/lscm_dcscm_ltpi_inst/lscm_tx_path_inst/xo3_tx_PHY.lscm_phy_tx_xo3_inst/tx_sclk_o

The following paths must be set both ways (from A to B and from B to A):

- LTPI Clock – System Clock

- LTPI Sync Clock – System Clock
- LTPI TX Divider Clock – System Clock
- LTPI Sync Clock – LTPI RX PHY Clock
- LTPI Clock – LTPI RX PHY Clock
- LTPI Clock – LTPI TX PHY Clock
- System Clock – LTPI RX PHY Clock
- System Clock – LTPI TX PHY Clock

4.3.2. Clock Definition

Only the LTPI RX clock (ltpi0_inst_lvds_rx_clk_i_port) needs to be manually set. It can be set up to 400 MHz. Every other clock is inferred by the Lattice Diamond software. [Figure 4.12](#) shows the clock definition.

Preference Name	Preference Value	Preference Unit
▼ PORT "ltpi0_inst_lvds_rx_clk_i_port"		
Frequency	100.000000	MHz
Hold Margin	0.000000	ns
PAR_ADJ	0.000000	
Clock Jitter(p-p)	0.000000	ns
▼ NET "upll_inst_CLKOP"		
Frequency	24.000000	MHz
Hold Margin	Auto	ns
PAR_ADJ	0.000000	
▼ NET "upll_inst_CLKOS3"		
Frequency	100.000000	MHz
Hold Margin	Auto	ns
PAR_ADJ	0.000000	
▼ NET "dupll_inst_CLKOS3"		
Frequency	25.000000	MHz
Hold Margin	Auto	ns
PAR_ADJ	0.000000	
▼ NET "upll_inst_CLKOS_net"		
Frequency	25.000000	MHz
Hold Margin	Auto	ns
PAR_ADJ	0.000000	
▼ NET "dupll_inst_CLKOP_net"		
Frequency	400.000000	MHz
Hold Margin	Auto	ns
PAR_ADJ	0.000000	
▼ NET "dupll_inst_CLKOS_net"		
Frequency	400.000000	MHz
Hold Margin	Auto	ns
PAR_ADJ	0.000000	
▼ NET "dupll_inst_CLKOS2_net"		
Frequency	80.000000	MHz
Hold Margin	Auto	ns
PAR_ADJ	0.000000	
▼ NET "upll_inst_CLKOS2_net"		
Frequency	2.500000	MHz
Hold Margin	Auto	ns
PAR_ADJ	0.000000	
▼ NET "clk_i_c"		
Frequency	12.000000	MHz
Hold Margin	Auto	ns
PAR_ADJ	0.000000	

Figure 4.12. Clock Definition

5. Hardware Bring-up

5.1. SCM – MachXO3D Breakout Board and LTPI Board

5.1.1. Setting Up the Hardware

The SCM setup requires two boards, a MachXO3D Breakout Board (Figure 5.1) and a custom LTPI Demonstration Board that is shipped by request (Figure 5.2). The OPN of the MachXO3D Breakout Board is LCMXO3D-9400HC-B-EVN. Refer to the [MachXO3D Breakout Board](#) web page for more information.



Figure 5.1. MachXO3D Breakout Board

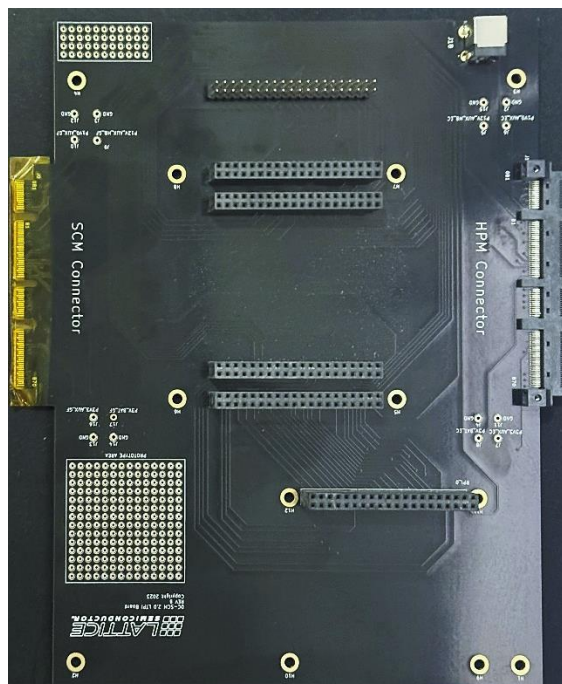


Figure 5.2. LTPI Demonstration Board

To set up the MachXO3D Breakout Board, it requires soldering plugs on the bottom side. [Figure 5.3](#) shows the soldered MachXO3D Breakout Board and [Figure 5.4](#) shows the assembled LTPI Demonstration Board.

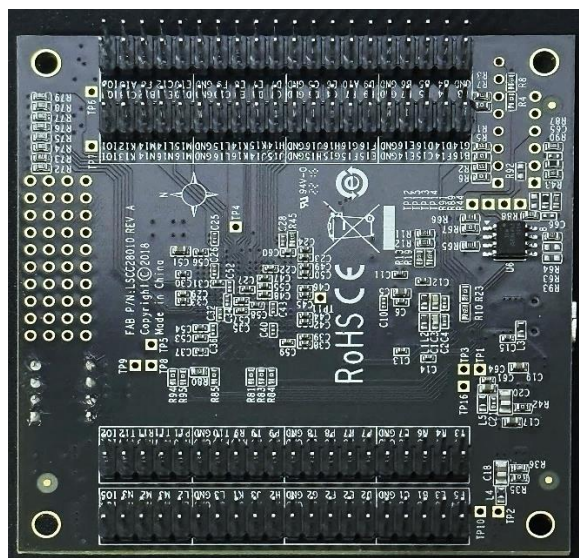


Figure 5.3. MachXO3D Breakout Board (Soldered)



Figure 5.4. Assembled LTPI Demonstration Board

5.1.2. Initial Connection

Connect a USB Mini B to J2 of the MachXO3D Breakout Board. Connect the other end to a PC. This serves as the connection to the on-board FTDI chip, which is used for configuration as well as the power supply for the board.

5.1.3. Uploading the Configuration File

1. Open the SCM_soc project file (SCM_soc.lbf) in the Lattice Diamond software ([Figure 5.5](#)).

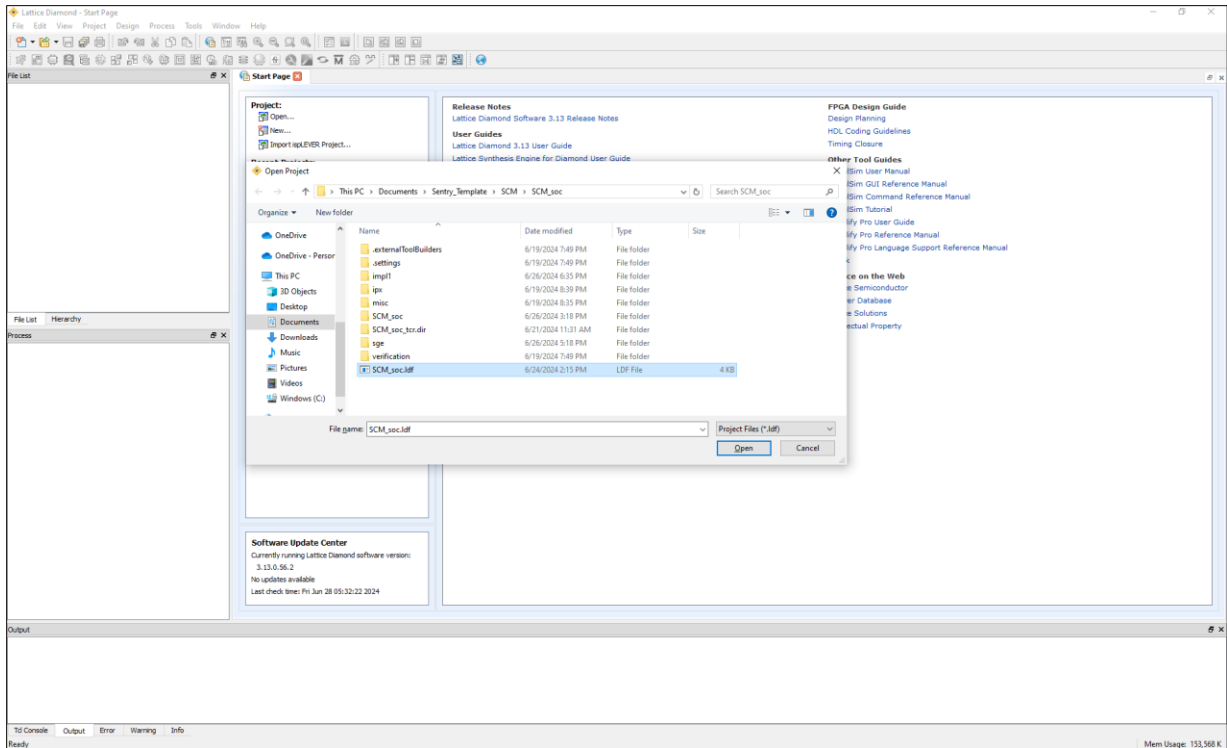


Figure 5.5. Open SCM_soc in Lattice Diamond Software

- Using Diamond Programmer, upload either the JED or BIT file using either Flash Programming Mode or Static RAM Cell Mode, as shown in Figure 5.6 and Figure 5.7 respectively. Note that the template comes with a pre-exported JED and BIT files. If any changes are made in the design, it is recommended that you export a new JED or BIT file.

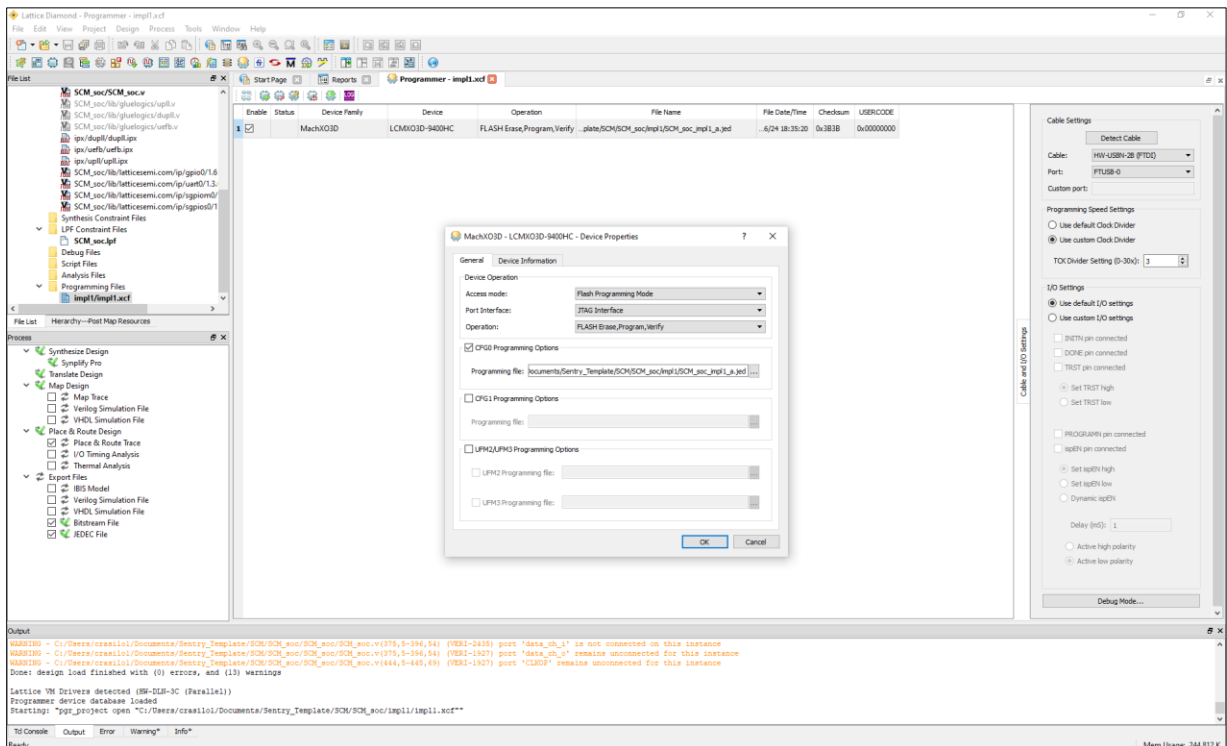


Figure 5.6. JED File Upload Using Flash Programming Mode (SCM)

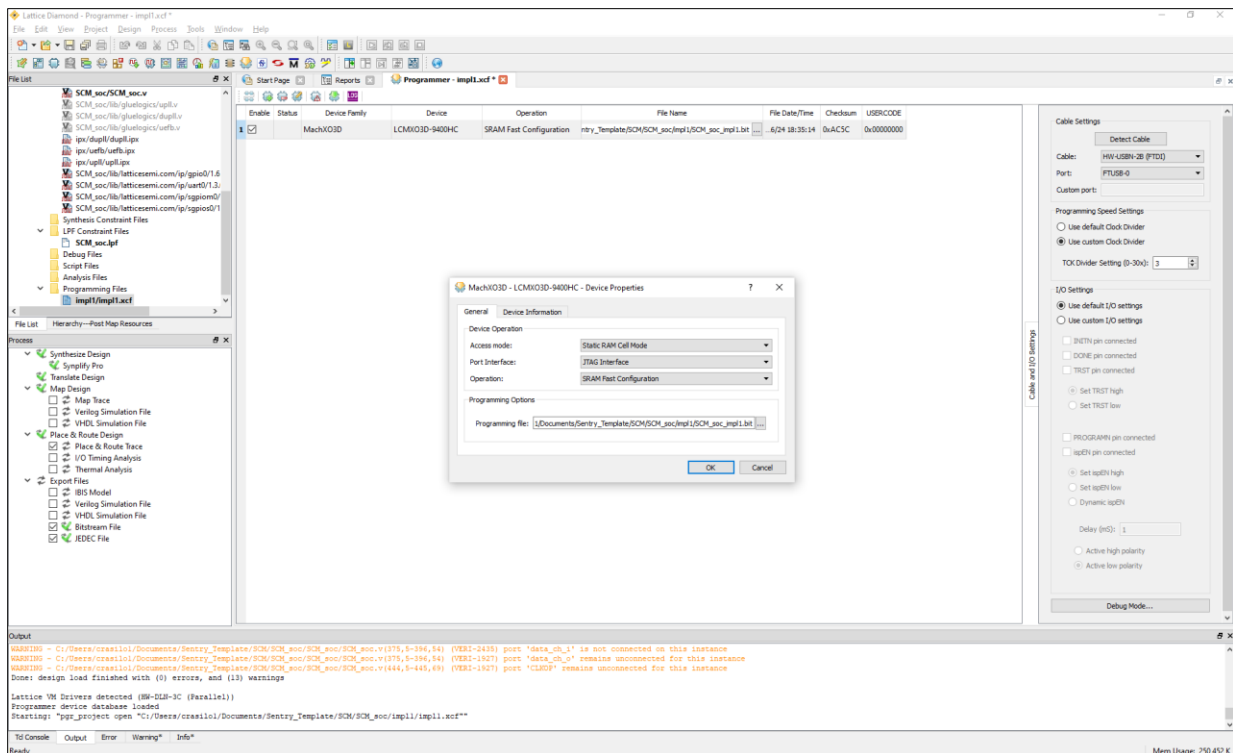


Figure 5.7. BIT File Upload Using Static RAM Cell Mode (SCM)

5.2. HPM – Sentry 4.0 Board

The Sentry 4.0 board has three device sections: the MachXO3D device, MachXO5-NX device, and CertusPro™-NX device. For this demonstration, the MachXO3D device is used as the HPM, and the MachXO5-NX device is used as the M-PESTI target, and the CertusPro-NX device is unutilized. Due to some device limitations, some level of board rework needs to be applied. Jumper and wire connections for board programming and debugging are also described below.

5.2.1. Rework for LTPI

The capacitors C297 and C298 need to be removed before soldering the wire connections. It is recommended to use at least AWG30 size or thinner. The wires should be as short as possible and have equal lengths. See [Figure 5.8](#) for details.



1. Connect a 12 V DC Power Supply to J1.
2. Connect a USB Mini B to J2. Connect the other end to a PC. This serves as the connection to the on-board FTDI chip, which is used for configuration or UART access.
 - a. To use the FTDI for configuration, refer to the [Uploading the Configuration File \(MachXO3D\)](#) section and the [Uploading the Configuration File \(MachXO5-NX\)](#) section. Disconnect JP1 and JP2.
 - b. To use the FTDI for UART terminal access, short JP1 and JP2 and disconnect all configuration jumpers mentioned in the [Uploading the Configuration File \(MachXO3D\)](#) section and the [Uploading the Configuration File \(MachXO5-NX\)](#) section. To establish terminal access using an external UART device, use pin 1 of both JP1 and JP2 as TXD and RXD, respectively.
3. Toggle SW15 towards J2 to turn on the board.

[Figure 5.9](#) shows the initial connections.

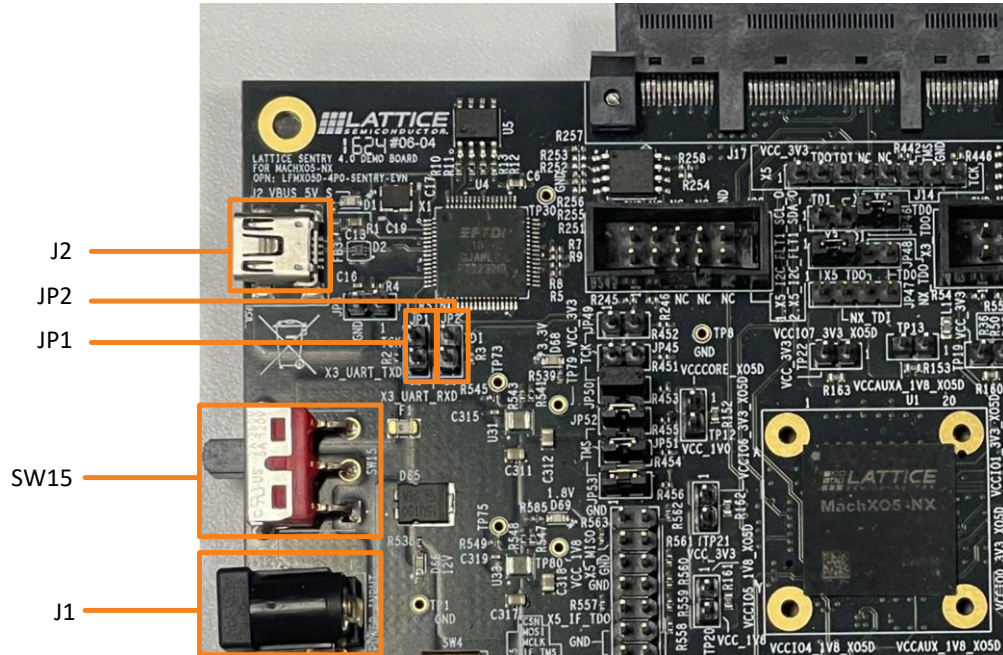


Figure 5.9. Initial Connections

5.2.3. Uploading the Configuration File (MachXO3D)

1. Connect the board jumpers, as shown in Table 5.1 and Figure 5.10.

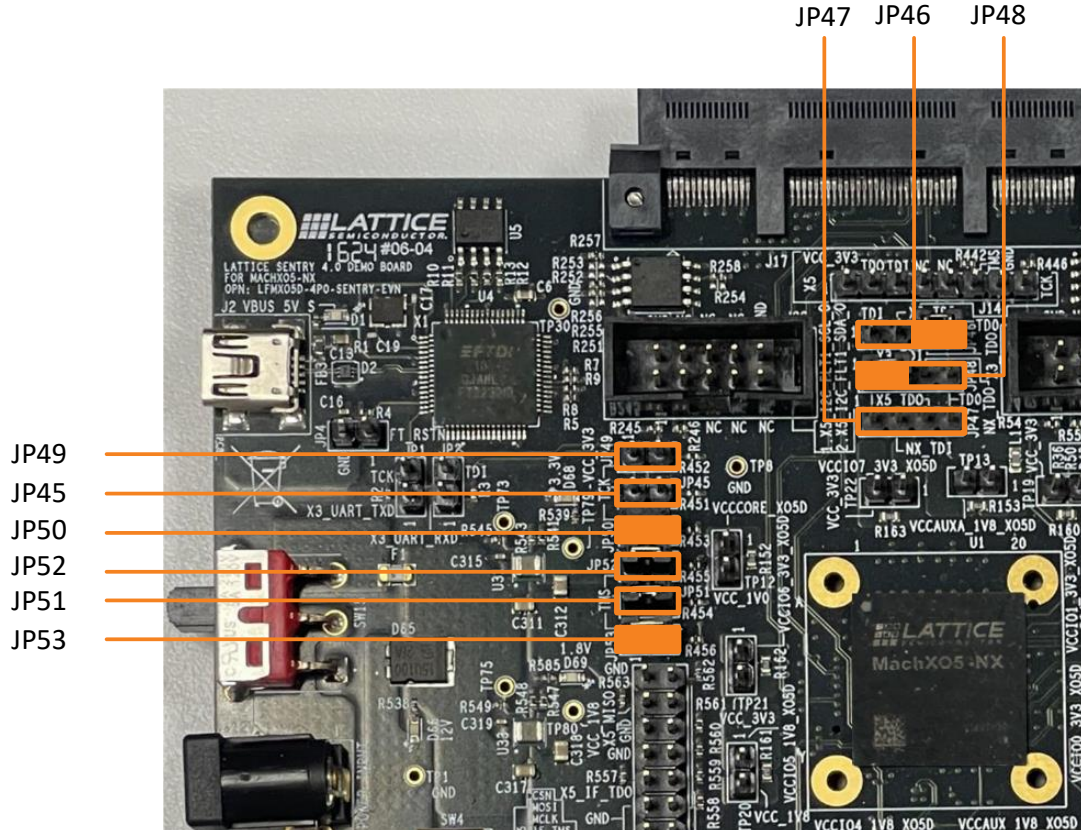


Figure 5.10. MachXO3D Configuration Jumpers

Table 5.1. Jumper Settings for MachXO3D Configuration

Jumper Name	Connection
JP45	Open
JP46	3-4
JP47	Open
JP48	1-2
JP49	Open
JP50	Short
JP51	Open
JP52	Open
JP53	Short

- Open the HPM_soc project file (HPM_soc.ldf) in the Lattice Diamond software (Figure 5.11).

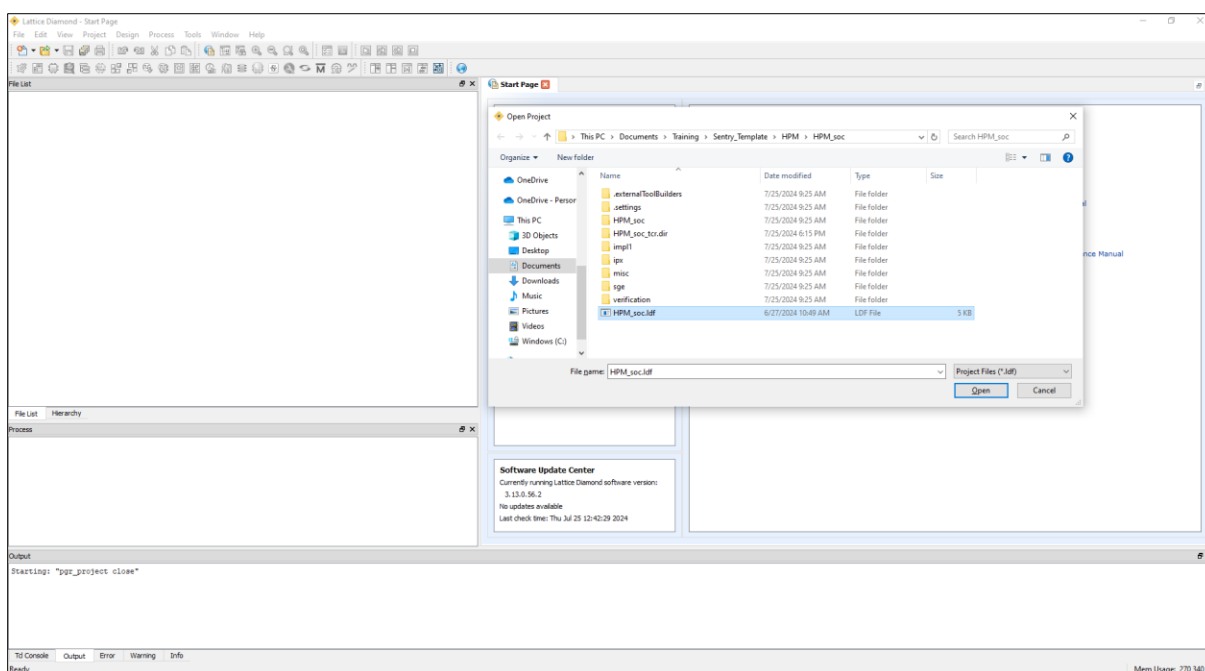


Figure 5.11. Open HPM_soc in the Lattice Diamond Software

- Using Diamond Programmer, upload either the JED or BIT file using either Flash Programming Mode or Static RAM Cell Mode, as shown in Figure 5.12 and Figure 5.13 respectively. Note that the template comes with a pre-exported JED and BIT files. If any changes are made in the design, it is recommended that you export a new JED or BIT file.

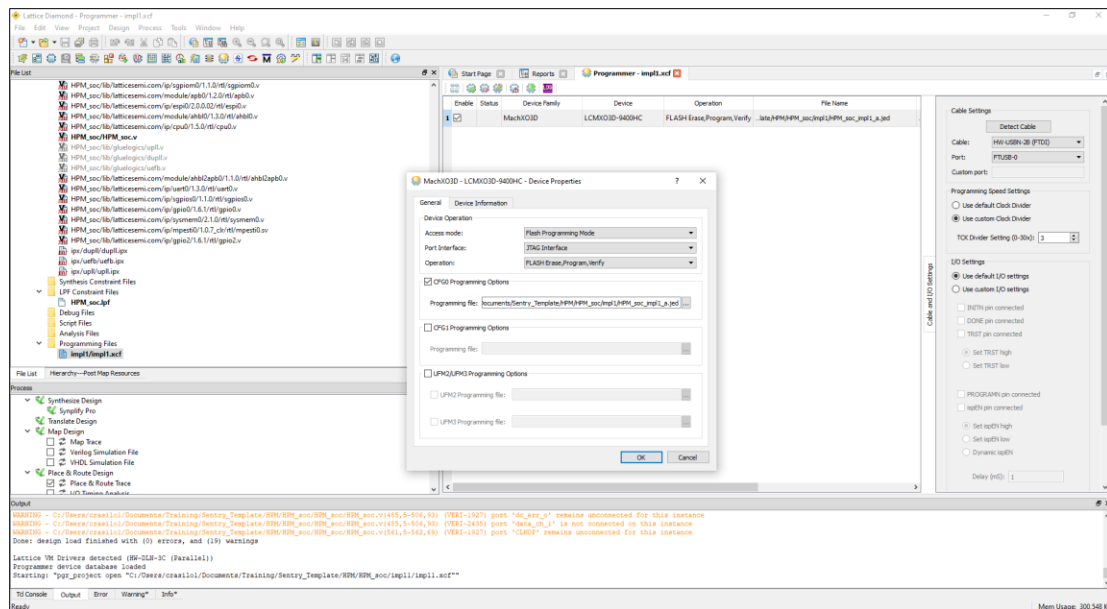


Figure 5.12. JED File Upload Using Flash Programming Mode (HPM)

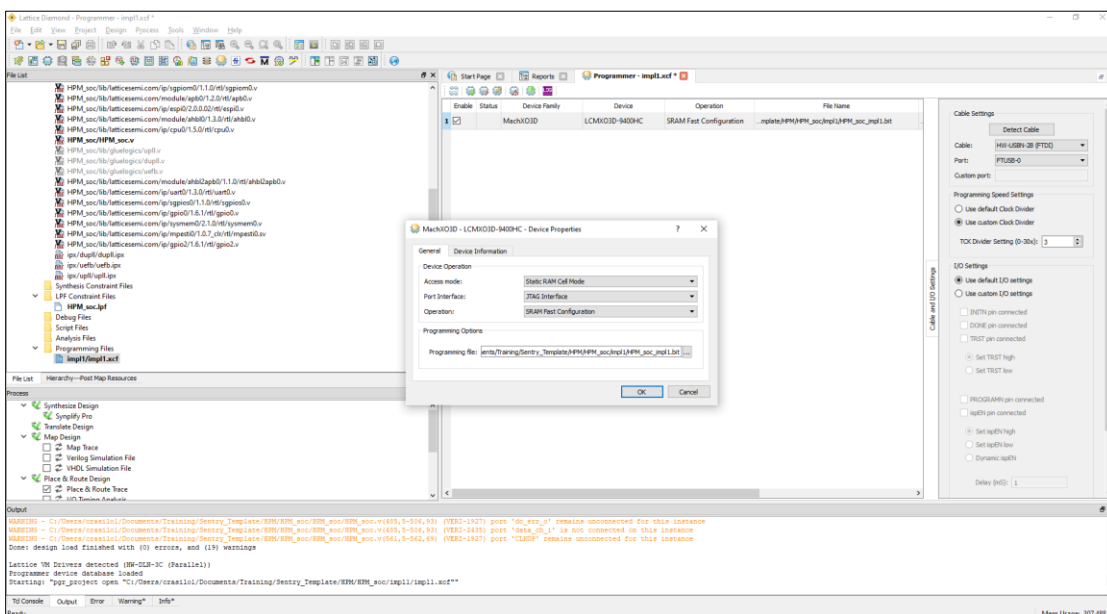


Figure 5.13. BIT File Upload Using Static RAM Cell Mode (HPM)

5.2.4. Uploading the Configuration File (MachXO5-NX)

For this demonstration setup, the MachXO5-NX device is used as multiple pseudo-M-PESTI targets. The design template package comes with a pre-built JED file for this purpose.

1. Connect the jumpers, as shown in [Table 5.2](#) and [Figure 5.14](#).

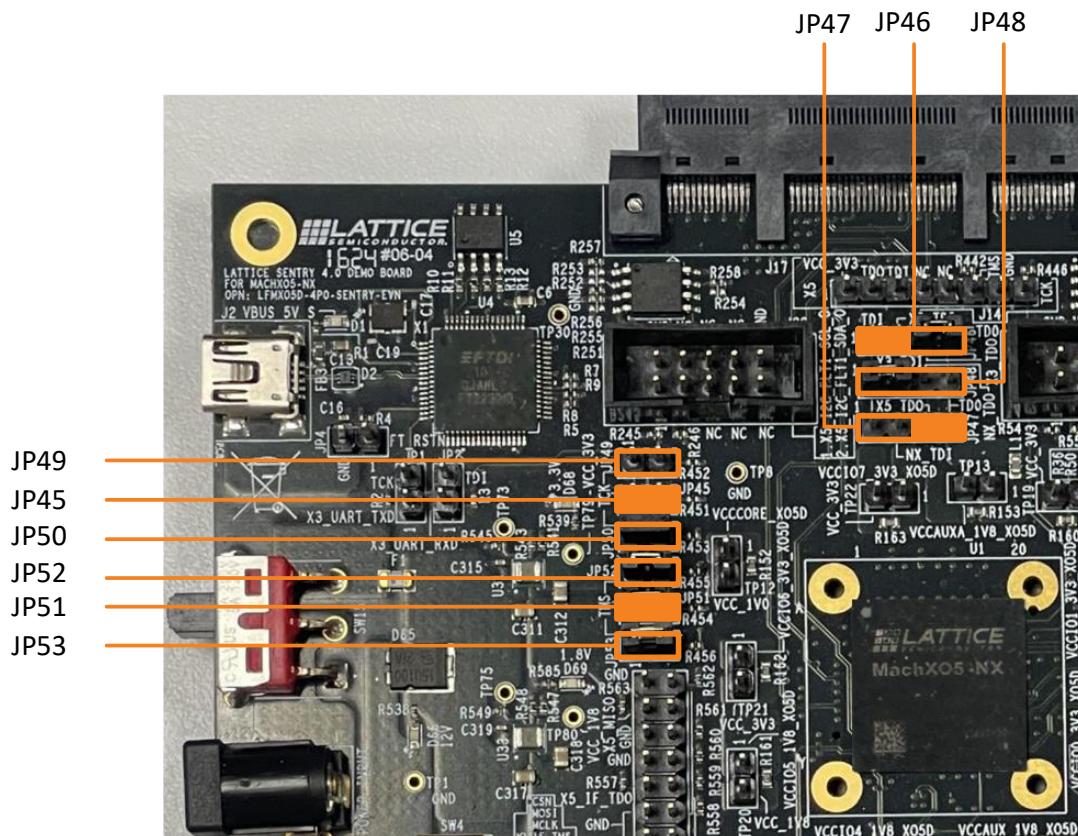


Figure 5.14. MachXO5-NX Configuration Jumpers

Table 5.2. Jumper Settings for MachXO5-NX Configuration

Jumper Name	Connection
JP45	Short
JP46	1-2
JP47	3-4
JP48	Open
JP49	Open
JP50	Open
JP51	Short
JP52	Open
JP53	Open

- In Radiant Programmer, upload the JED file using Flash Configuration Memory (Figure 5.15).

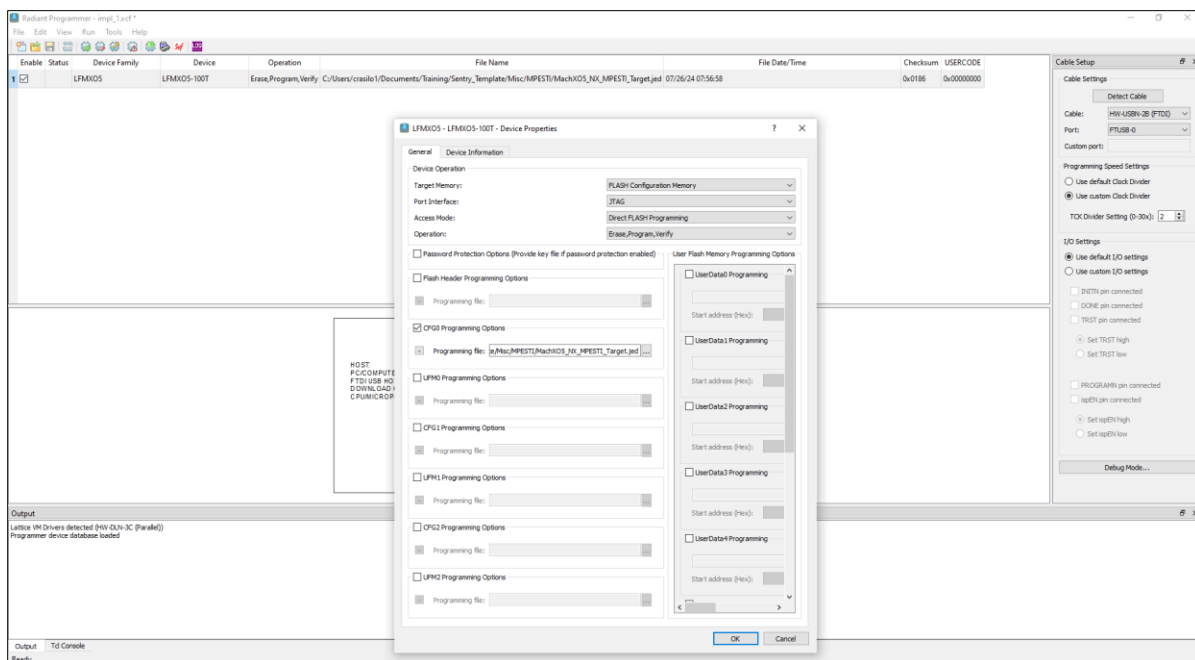


Figure 5.15. MachXO5-NX Configuration

5.3. Connecting the Two Boards

Connect J1 (HPM Connector) of the LTPI demonstration board to J16 of the Sentry 4.0 board, as shown in Figure 5.16.

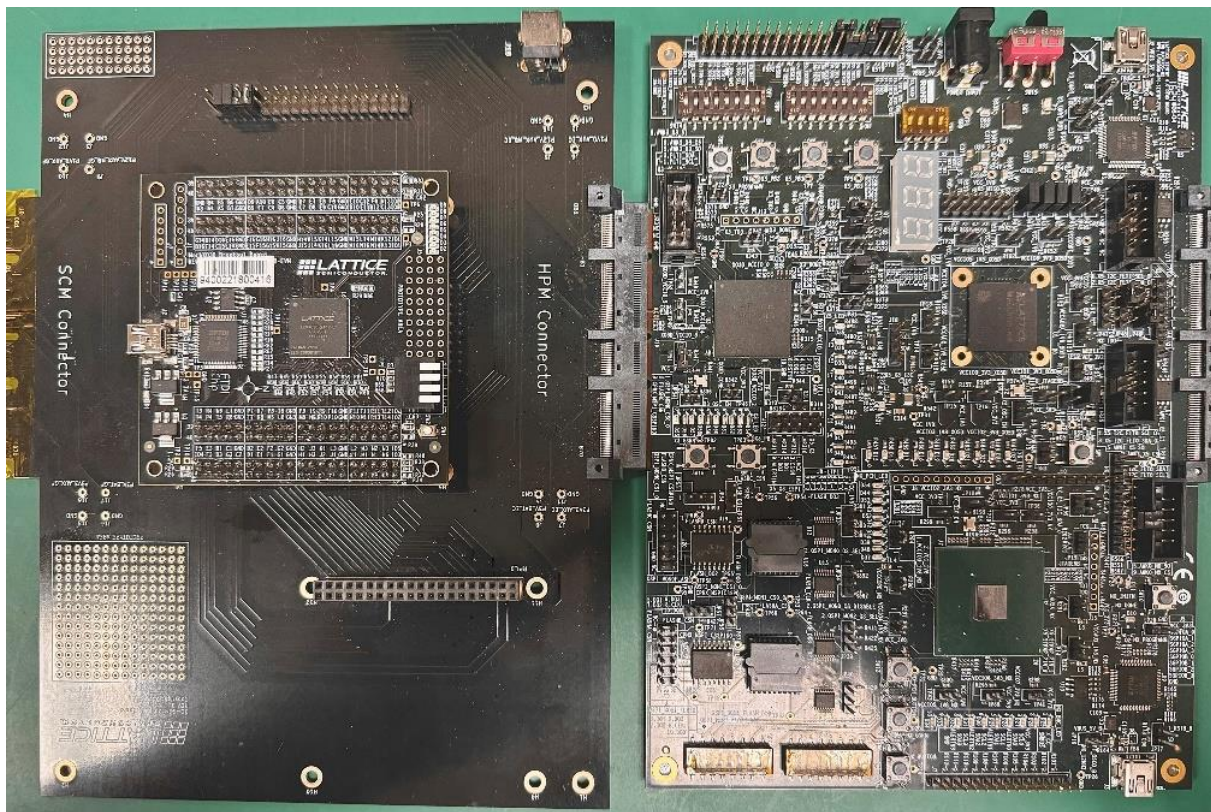


Figure 5.16. Demonstration Setup

5.4. eSPI Controller Using Total Phase Promira Serial Platform (Optional for eSPI Demonstration)

This demonstration uses the Total Phase Promira Serial platform with I2C/SPI active applications. It uses an API code that can generate and support up to 66 MHz single, dual, and quad I/O eSPI transactions. You may opt to use a different eSPI controller since the eSPI Target IP included in the HPM Template follows the Intel eSPI Base Specification 1.0 and should work with any other devices.

Below are the steps for setting up the Promira Serial Platform.

1. Connect eSPI Controller to HPM hardware through RPI header (J11) as highlighted in Figure 5.17.

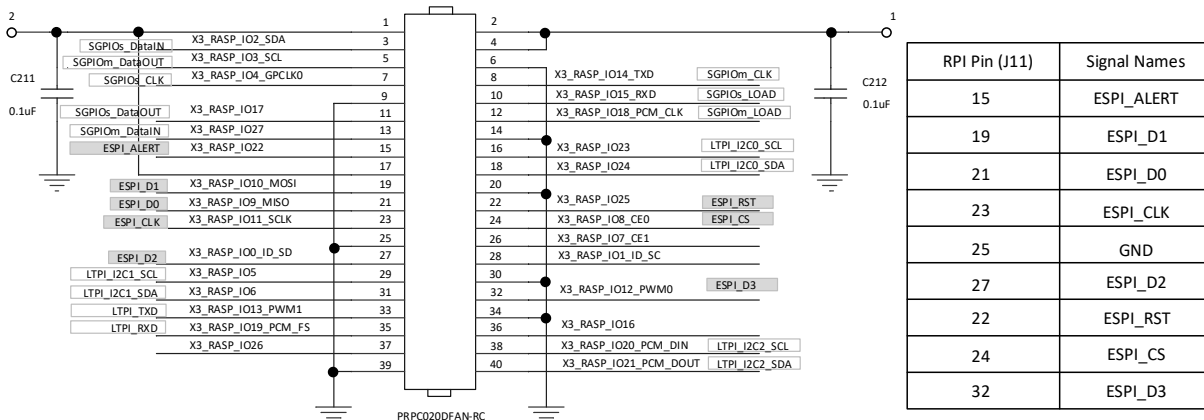


Figure 5.17. eSPI Pin Connection

2. Download a copy of the API code included in the Sentry 4.0 source package.
3. Install Python 3.12 32-bit version.
4. Open IDLE Shell. Then, open the espi_generator.py file. This is the main file for running the eSPI transactions.

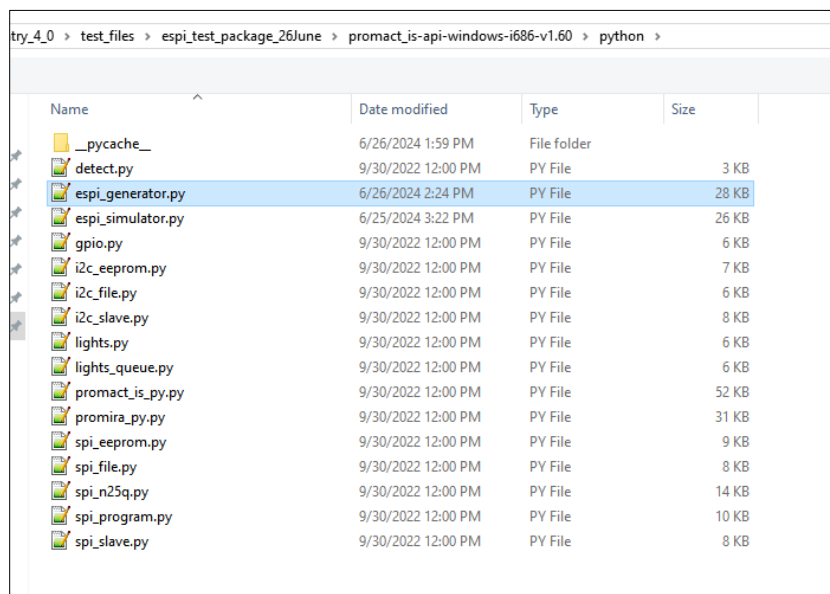


Figure 5.18. API for eSPI Transactions

Note: The API code includes the Promira devices detection, as shown in Figure 5.19. So, there is no need to manually input the IP address.

```

541 #Detect Promira Devices
542 print("Detecting the Promira platforms...")
543
544 # Find all the attached devices
545 (num, ips, unique_ids, statuses) = pm_find_devices_ext(16, 16, 16)
546
547 if num > 0:
548     print("%d device(s) found:" % num)
549
550     # Print the information on each device
551     for i in range(num):
552         ip = ips[i]
553         unique_id = unique_ids[i]
554         status = statuses[i]
555
556         # Determine if the device is in-use
557         if status & PM_DEVICE_NOT_FREE:
558             inuse = "(in-use)"
559         else:
560             inuse = "(avail)"
561
562         # Display device ip address, in-use status, and serial number
563         ipstr = "%u.%u.%u.%u" % (ip & 0xff, ip >> 8 & 0xff, ip >> 16 & 0xff, ip >> 24 & 0xff)
564         print("ip = %s %s (%04d-%06d)" % (ipstr, inuse, unique_id // 1000000, unique_id % 1000000))
565
566 else:
567     print("No devices found.")
568     exit(1)
569
570
571
572
573
574
575 #Global Variables
576 my_promira_port = ipstr #IP address example "10.1.66.85"
577
578
579 # Open the device
580 simulator = EspiSimulator(my_promira_port)
581 simulator.espi_config_mode(1)
582
583

```

Figure 5.19. Promira Port IP Address Detection

- Click **Run** on IDLE Shell and select **Run Module** (Figure 5.20).

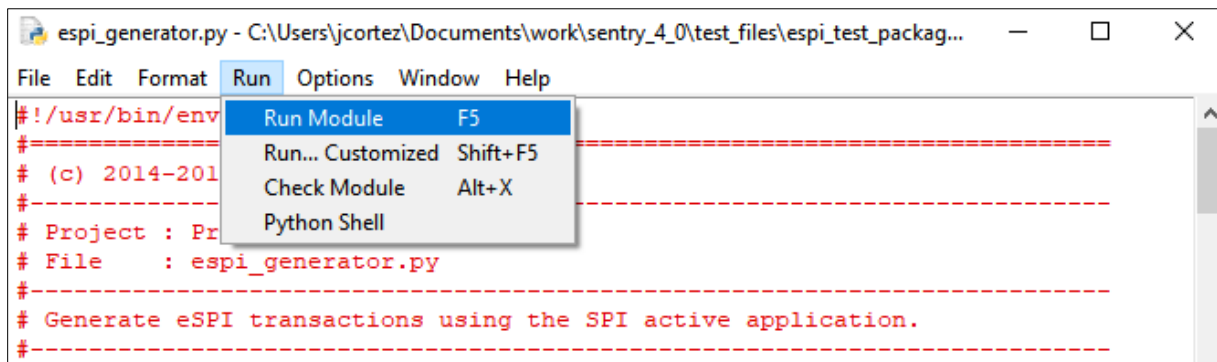


Figure 5.20. Running in IDLE Shell

- The IDLE Shell console should show prints (Figure 5.21).



```

>>> = RESTART: C:\Users\jcortez\Documents\work\sentry_4_0\test_files\espi_test_packa
ge_26June\promact_is-api-windows-i686-v1.60\python\espi_generator.py
Detecting the Promira platforms...
1 device(s) found:
    ip = 10.1.66.85    (avail)    (2416-713085)

Get Config Command Initiated
Valid Response: ['0x8', '0x2', '0x0', '0x0', '0x93', '0x4', '0x1', '0xff']
Passed

Set Config Command Initiated
Valid Response: ['0x8', '0x4', '0x1', '0x2']
Passed
espi_config_bus: bitrate=20000, iospi=0

Get Config Command Initiated
Valid Response: ['0x8', '0x2', '0x0', '0x0', '0x93', '0x4', '0x1', '0xff']
Passed

Set Channe 1 Config Command Initiated
Valid Response: ['0x8', '0x4', '0x1', '0x2']
Passed
Check Alert
  
```

Ln: 27 Col: 0

Figure 5.21. IDLE Shell Output

5.5. Total Phase Aardvark for I2C (Optional for LTPI I2C Aggregation)

Device product name: Aardvark I2C/SPI Host Adapter (Figure 5.22)
 Target Bus Interface: I2C Controller/Target
 Bit Rate: I2C Controller: 1 kHz–800 kHz



Figure 5.22. Aardvark I2C/SPI Host Adapter

On this demonstration, two Aardvark devices are used for Single Controller-Single Target setup, connected to each side (SCM–HPM) of the DC-SCM LTPI to the Sentry 4.0 board for the validation of I2C communication.

1. I2C pin connections from Aardvark to the SCM board and the Sentry 4.0 board:
 - a. Connect GND and I2C pins of one Aardvark device to the SCM board, that is, the MachXO3D breakout board. I2C bus 0, 1, 2 are available for connection on this project design. For this single controller-single target demonstration, I2C bus 0 (pins D8, L16) is used in the SCM side (Figure 5.23 and Figure 5.24).

	Name	Group By	Pin	BANK
1.2.3	Itpi0_inst_i2c_o_portbus[2]	N/A	J11(J11)	1(1)
1.2.4	Itpi0_inst_i2c_o_portbus[3]	N/A	L12(L12)	1(1)
1.2.5	Itpi0_inst_i2c_o_portbus[4]	N/A	K11(K11)	1(1)
1.2.6	Itpi0_inst_i2c_o_portbus[5]	N/A	L13(L13)	1(1)
1.2.7	Itpi0_inst_i2c_o_portbus[6]	N/A	N15(N15)	1(1)
1.2.8	Itpi0_inst_i2c_o_portbus[7]	N/A	P16(P16)	1(1)
1.2.9	Itpi0_inst_lvds_b_clk_o_port	N/A	D10(D10)	0(0)
1.2.10	Itpi0_inst_lvds_b_data_o_port	N/A	A5(A5)	0(0)
1.2.11	Itpi0_inst_uart_rx_o_portbus[0]	N/A	M16(M16)	1(1)
1.2.12	sgpiom0_inst_sclck_o_port	N/A	H15(H15)	1(1)
1.2.13	sgpiom0_inst_sdataout_o_port	N/A	K15(K15)	1(1)
1.2.14	sgpiom0_inst_sload_o_port	N/A	K16(K16)	1(1)
1.2.15	sgpiom0_inst_oGSXDataOut_port	N/A	M11(M11)	2(2)
1.2.16	uart0_inst_brd_o_port	N/A	L14(L14)	1(1)
1.3	Bidir	N/A	N/A	N/A
1.3.1	Itpi0_inst_i2c_scl_io_portbus[0]	N/A	D8(D8)	0(0)
1.3.2	Itpi0_inst_i2c_scl_io_portbus[1]	N/A	E9(E9)	0(0)
1.3.3	Itpi0_inst_i2c_scl_io_portbus[2]	N/A	D6(D6)	0(0)
1.3.4	Itpi0_inst_i2c_sda_io_portbus[0]	N/A	L16(L16)	1(1)
1.3.5	Itpi0_inst_i2c_sda_io_portbus[1]	N/A	M15(M15)	1(1)
1.3.6	Itpi0_inst_i2c_sda_io_portbus[2]	N/A	N14(N14)	1(1)

Figure 5.23. SCM Project Design Spreadsheet

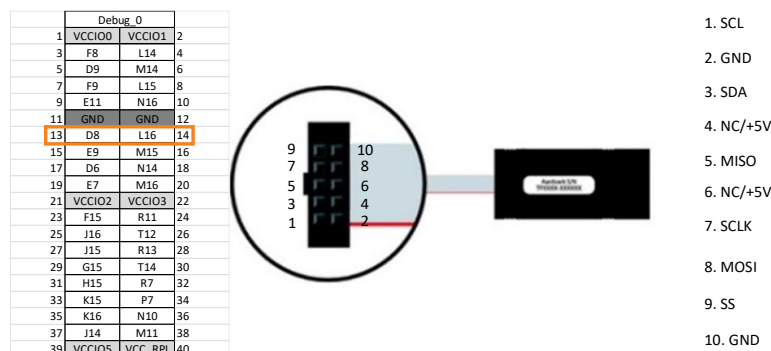


Figure 5.24. MachXO3D Breakout Board – Debug Pinout and Aardvark Pinouts

- Connect GND and I2C pins of the other Aardvark device to the HPM board, that is, the Sentry 4.0 board.
- Use I2C bus 0 (pins C6, D7) for the HPM side (Figure 5.25).

	Name	Group By	Pin	BANK
1.2.11	Itpi0_inst_i2c_o_portbus[5]	N/A	T3(T3)	3(3)
1.2.12	Itpi0_inst_i2c_o_portbus[6]	N/A	R4(R4)	3(3)
1.2.13	Itpi0_inst_i2c_o_portbus[7]	N/A	T4(T4)	3(3)
1.2.14	Itpi0_inst_lvds_b_clk_o_port	N/A	B11(B11)	0(0)
1.2.15	Itpi0_inst_lvds_b_data_o_port	N/A	F10(F10)	0(0)
1.2.16	Itpi0_inst_uart_rx_o_portbus[0]	N/A	D6(D6)	0(0)
1.2.17	sgpiom0_inst_sclck_o_port	N/A	A5(A5)	0(0)
1.2.18	sgpiom0_inst_sdataout_o_port	N/A	F7(F7)	0(0)
1.2.19	sgpiom0_inst_sload_o_port	N/A	A6(A6)	0(0)
1.2.20	sgpiom0_inst_oGSXDataOut_port	N/A	B5(B5)	0(0)
1.2.21	uart0_inst_brd_o_port	N/A	N1(N1)	4(4)
1.3	Bidir	N/A	N/A	N/A
1.3.1	espi0_inst_espi_data_io_portbus[0]	N/A	A4(A4)	0(0)
1.3.2	espi0_inst_espi_data_io_portbus[1]	N/A	F8(F8)	0(0)
1.3.3	espi0_inst_espi_data_io_portbus[2]	N/A	B1(B1)	0(0)
1.3.4	espi0_inst_espi_data_io_portbus[3]	N/A	O8(O8)	0(0)
1.3.5	Itpi0_inst_i2c_scl_io_portbus[0]	N/A	C6(C6)	0(0)
1.3.6	Itpi0_inst_i2c_scl_io_portbus[1]	N/A	A3(A3)	0(0)
1.3.7	Itpi0_inst_i2c_scl_io_portbus[2]	N/A	E7(E7)	0(0)
1.3.8	Itpi0_inst_i2c_sda_io_portbus[0]	N/A	D7(D7)	0(0)
1.3.9	Itpi0_inst_i2c_sda_io_portbus[1]	N/A	C3(C3)	0(0)
1.3.10	Itpi0_inst_i2c_sda_io_portbus[2]	N/A	B6(B6)	0(0)
1.3.11	mpest0_inst_mpest_portbus[0]	N/A	L7(L7)	4(4)
1.3.12	mpest0_inst_mpest_portbus[1]	N/A	K5(K5)	4(4)
1.3.13	mpest0_inst_mpest_portbus[2]	N/A	K2(K2)	4(4)
1.3.14	mpest0_inst_mpest_portbus[3]	N/A	L6(L6)	4(4)

Figure 5.25. HPM Project Design Spreadsheet

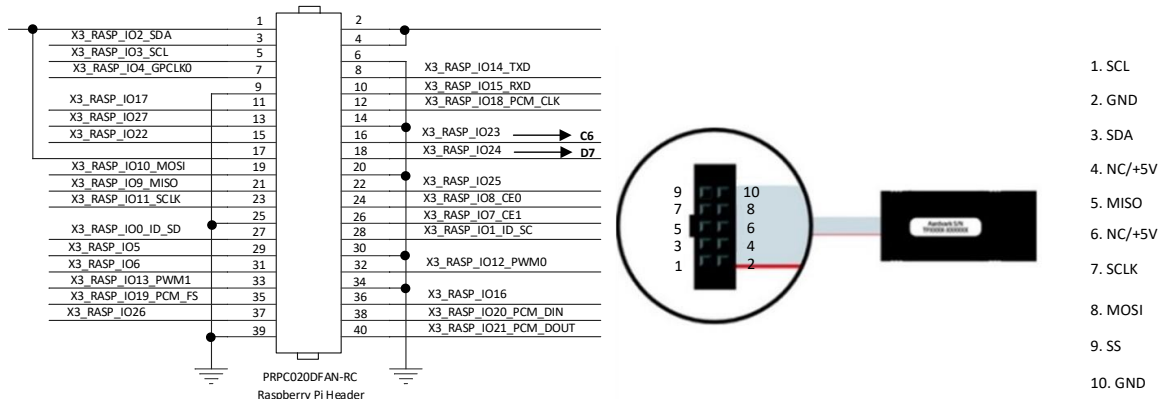


Figure 5.26. Sentry 4.0 board – J11 RPi Header and Aardvark Pinouts

3. Install Total Phase Control Center Serial Software. Refer to [Control Center Serial Software – Total Phase](#) for more information.
4. Open two Control Center applications. Set one as Controller and set the other as Target.
5. Check device ID at the back of the aardvark device (Figure 5.27). Then, connect the corresponding aardvark device by selecting **Adapter > Connect > Configure Adapter**.



Figure 5.27. Aardvark device ID

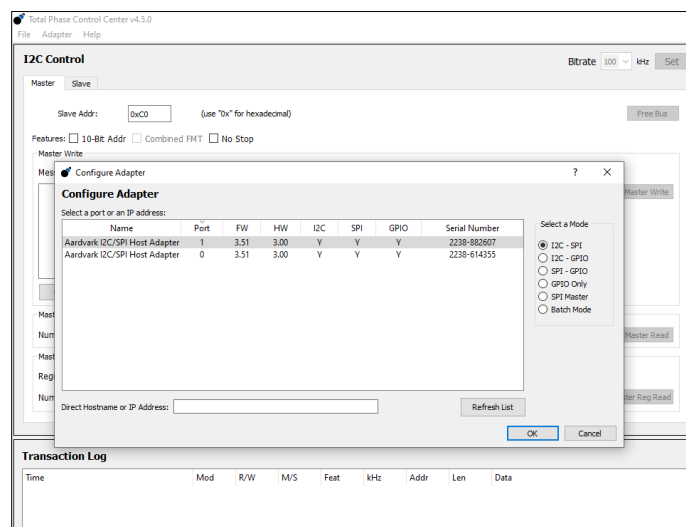


Figure 5.28. Control Center Adapter Configuration

6. After configuring the devices, the setup is now ready for I2C communication. Refer to the [I2C Communication on LTPI Using Aardvark](#) section for further discussion on I2C communication.

6. IP Demonstration

6.1.1. LTPI Demonstration

In this HPM-SCM LTPI demonstration, Sentry 4.0 Board (HPM) is connected to a DC-SCM 2.0 LTPI board (SCM). Both sides are connected through a USB cable, a power source on the HPM board, and a connection with UART terminal (Figure 6.1).

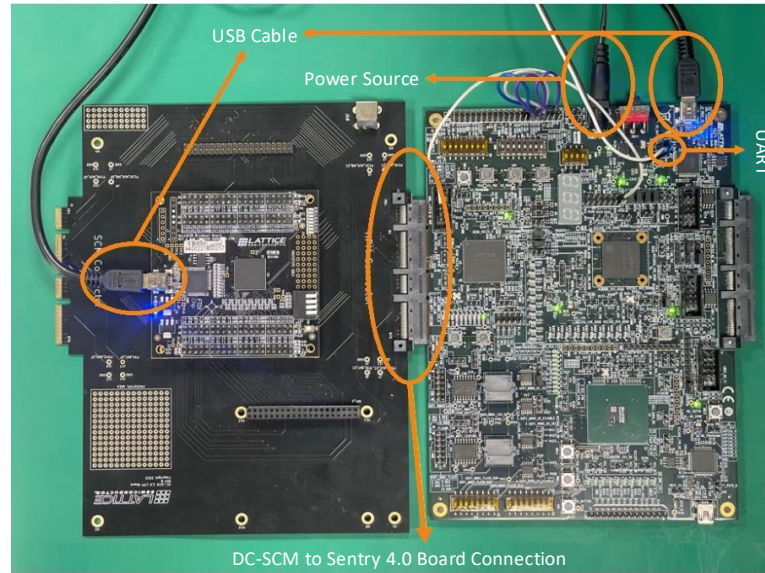


Figure 6.1. DC-SCM 2.0 LTPI board and Sentry 4.0 Demonstration Board

6.1.1.1. LTPI FW Demonstration

1. Power on the HPM and SCM to start initialization of IPs and PLL.
2. Upon power ON, LED pattern should be observed on both sides of the demonstration boards (Figure 6.2):
SCM (left side, DC-SCM Demonstration Board): increment loop pattern of LED from 0x00 to 0xFF (D2-D9)
HPM (right side, Sentry 4.0 Board): Blinking LEDs (D55-D62)

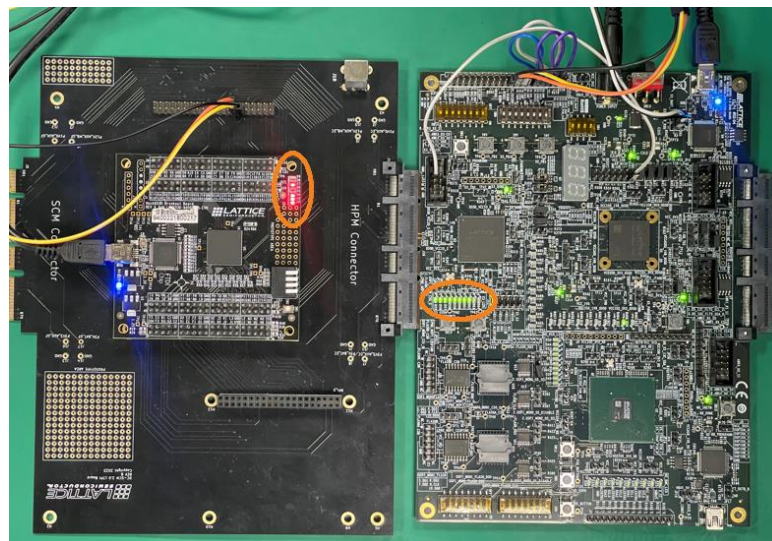


Figure 6.2. LED Indicator for LTPI Connection

- Upon power on, you can see the print output below in the UART terminal (Figure 6.3).

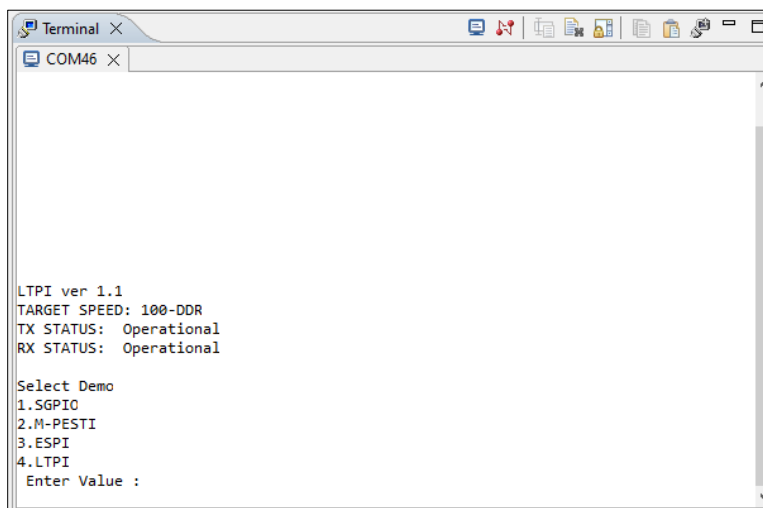


Figure 6.3. UART Terminal upon HPM Power On

- Figure 6.4 shows the function of LTPI demonstration which prints out the LTPI IP version, Capabilities (Target Speed), and Link Status (RX and TX).

```
382 void ltpi_demo() {
383
384     printf("\r\nLTPI Demo");
385     // This demo will provide access to different state and status of LTPI
386
387     ltpi_VERSION(soc0_inst_p->ltpi_inst); //LTPI IP specs
388     ltpi_I2C_RST(soc0_inst_p->ltpi_inst);
389     ltpi_TGT_SPEED(soc0_inst_p->ltpi_inst); //Cap0/Cap1 included
390     ltpi_TX_STATE(soc0_inst_p->ltpi_inst); //Link status
391     ltpi_RX_STATE(soc0_inst_p->ltpi_inst); //Link status
392 }
```

Figure 6.4. LTPI Demonstration Function

- As shown in Figure 6.3, UART prints the output indicating that RX and TX terminals are on operational mode with the target speed of 100 MHz at double data rate.

6.1.1.2. I2C Communication on LTPI Using Aardvark

- As mentioned in step 4 of the [Total Phase Aardvark for I2C \(Optional for LTPI I2C Aggregation\)](#) section, each of the two control centers is configured and used as Controller or as Target.
- Set bit rate at 100 kHz and set the same 7-bit target address on each side (Figure 6.5 and Figure 6.6).



Figure 6.5. Aardvark-Controller Setting



Figure 6.6. Aardvark-Target Setting

- Set response message on target side for read transaction (Figure 6.7).

The screenshot shows the I2C Control window with the Master tab selected. The Slave Address is 0x01. The Message field contains AA BB CC DD EE. The Master Read section shows the Number of Data Bytes set to 5. The Master Register Read section shows the Register Address as 0x00, Address Width as 2 bytes, and Number of Data Bytes as 4. The Transaction Log table below shows two transactions: a write transaction at 06:27:43.505 and a read transaction at 06:27:45.503.

Time	Mod	R/W	M/S	Feat	kHz	Addr	Len	Data
2024-07-02 06:27:43.505	I2C	W	M	---	100	0x1	5	AA BB CC DD EE
2024-07-02 06:27:45.503	I2C	R	M	---	100	0x1	5	11 22 33 44 55

Figure 6.7. Aardvark Target Response

- Do a simple I2C transaction. Input I2C data to Write message and click **Master Write** (Figure 6.8). Check I2C communication by clicking **Master Read**. Set response from the target should be visible on the Transaction Log (Figure 6.9).

The screenshot shows the I2C Control window with the Master tab selected. The Slave Address is 0x01. The Message field contains AA BB CC DD EE. The Master Write button is highlighted. The Master Read section shows the Number of Data Bytes set to 5. The Master Register Read section shows the Register Address as 0x00, Address Width as 2 bytes, and Number of Data Bytes as 4. The Transaction Log table below shows two transactions: a write transaction at 06:27:43.505 and a read transaction at 06:27:45.503.

Time	Mod	R/W	M/S	Feat	kHz	Addr	Len	Data
2024-07-02 06:27:43.505	I2C	W	M	---	100	0x1	5	AA BB CC DD EE
2024-07-02 06:27:45.503	I2C	R	M	---	100	0x1	5	11 22 33 44 55

Figure 6.8. Aardvark Controller

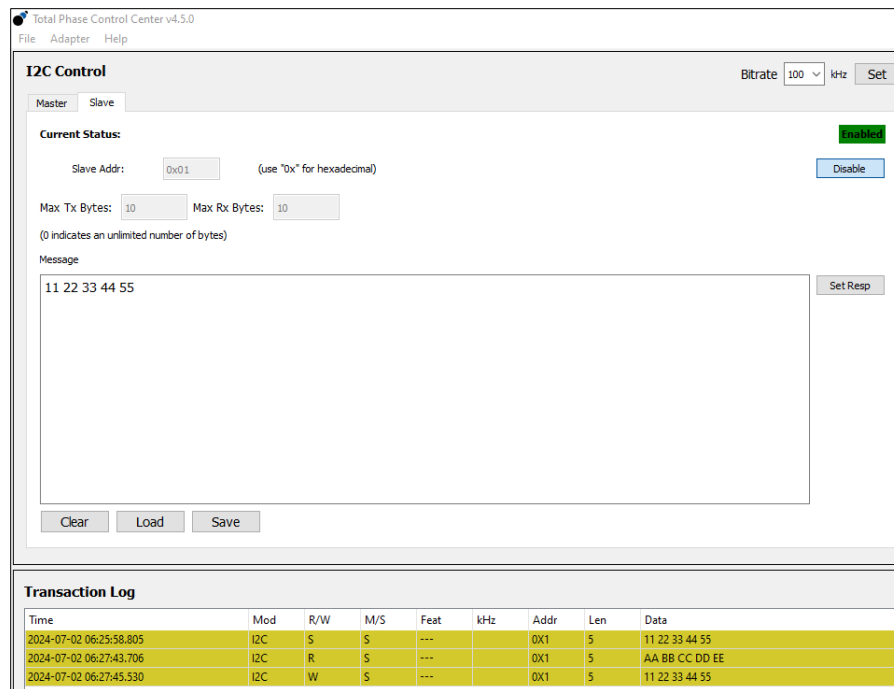


Figure 6.9. Aardvark Target

6.1.2. eSPI Demonstration

Figure 6.10 is the diagram of the eSPI demonstration. It shows the UART terminal, the HPM template with only eSPI demonstration-related components included, and the eSPI controller.

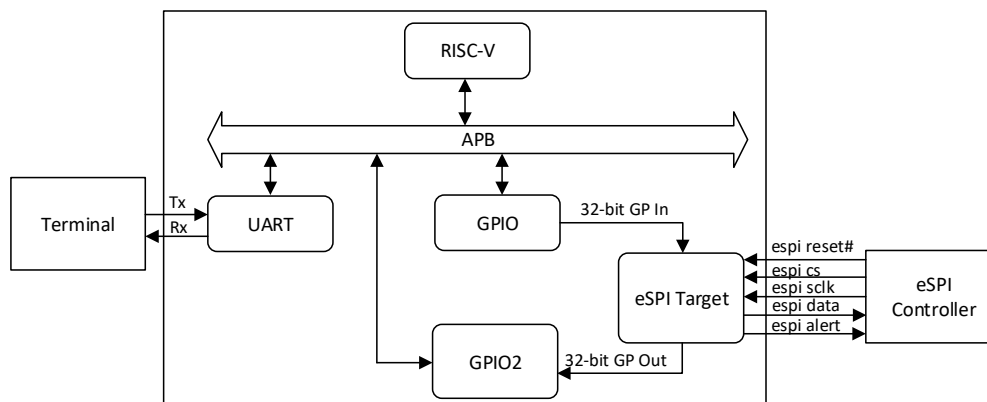


Figure 6.10. eSPI Demonstration Diagram

1. Power on the HPM hardware to turn on the eSPI target.
2. Upon power on or reset, the eSPI target is initialized to default configuration. See the Channel 1 Capabilities and Configurations section of [eSPI Target IP \(FPGA-IPUG-02260\)](#).
3. On the eSPI controller side, perform Set Configuration for General Capabilities and Configuration at address 0x008. This configures the capability of the eSPI target such as I/O mode, Operation Frequency, CRC Checking, and so on.
4. On the eSPI controller, perform Set Configuration for Channel 1 (Virtual Wire) Capabilities and Configuration at address 0x020. This configures the capability of the eSPI Target specific to Virtual Wire Channel. It is important to note that eSPI Target's Virtual Wire is by default disabled. So, performing this command to enable the bit 0 (Virtual Wire Channel Enable) of this channel is required. [Figure 6.11](#) shows pseudocode for this transaction.

```
set_configuration(address=0x020, data = [0x01,0x00,0x3F,0x00])
```

Figure 6.11. VW Channel Set Configuration Pseudo Code

5. On the eSPI controller, after performing step 4, the controller now continuously checks the Alert signal for each target-initiated transaction.
6. On the UART terminal, select 3 for eSPI demonstration and then enter any hexadecimal values to update the GPIO1 output pin values.
7. The eSPI target then captures the changes in the GPIO1 output pins and asserts the Alert pin to notify the eSPI controller of the event.
8. As programmed in the eSPI controller, once Alert is detected, it sends a Get VWIRE command to query the state from the GPIO1. Upon the Get VWIRE response, the data acquired is then used for Put VWIRE to update the GPIO2, thus, completing the loopback operation.
9. The printed output in the UART terminal should also be the same as the input ([Figure 6.12](#)).

```
Select Demo
1.SGPIO
2.M-PESTI
3.ESPI
4.LTPI
Enter Value : 3
ESPI Demo

Enter Value : 123456
GPI: 123456

GPO: 123456
```

Figure 6.12. Terminal Print Output

6.1.3. M-PESTI Demonstration

As simplified in [Figure 6.13](#), The M-PESTI demonstration involves two key components. The M-PESTI Initiator resides in HPM SoC design and is housed in the MachXO3D device, while the M-PESTI targets are located in the MachXO5-NX device. Both of these devices are part of the same Sentry 4.0 Demonstration Board.

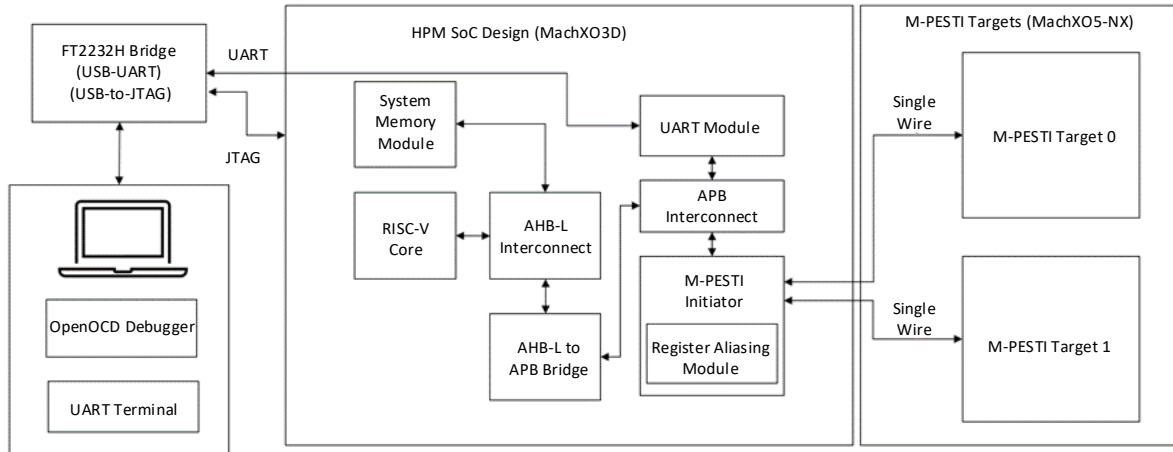


Figure 6.13. M-PESTI Demonstration Block Diagram

To run the demonstration, perform the following steps:

1. Connect JP27 and JP29, as shown in [Figure 6.14](#).

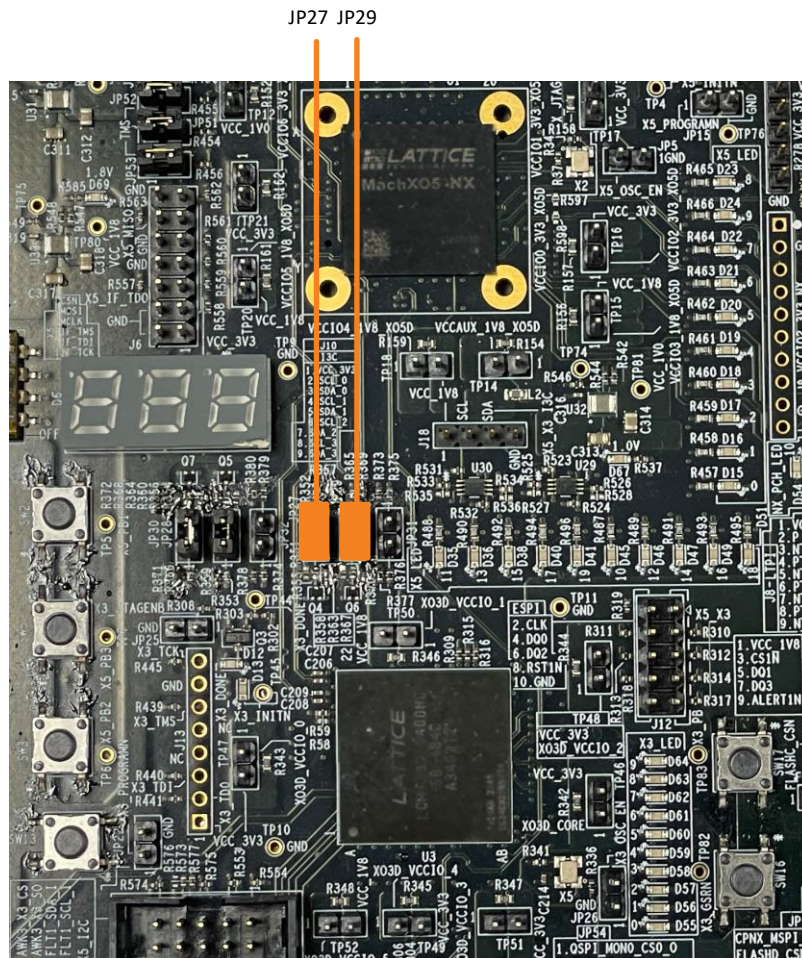


Figure 6.14. M-PESTI Demonstration Jumpers

2. Connect the 12 V power cable, USB cable, and slide the power switch to an upwards position, as shown in [Figure 6.15](#).

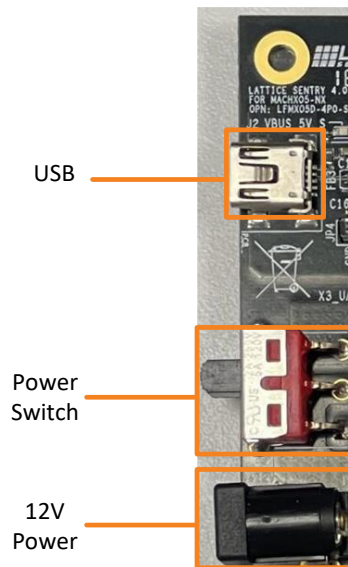


Figure 6.15. Power and Data Connections

- After power on, the demonstration menu screen is printed on the UART terminal (Figure 6.16).



Figure 6.16. Demonstration Menu Screen

- Wait until LED D24 blinks (Figure 6.17). Then, enter 2 in the Menu to select the M-PESTI demonstration and hit Enter.

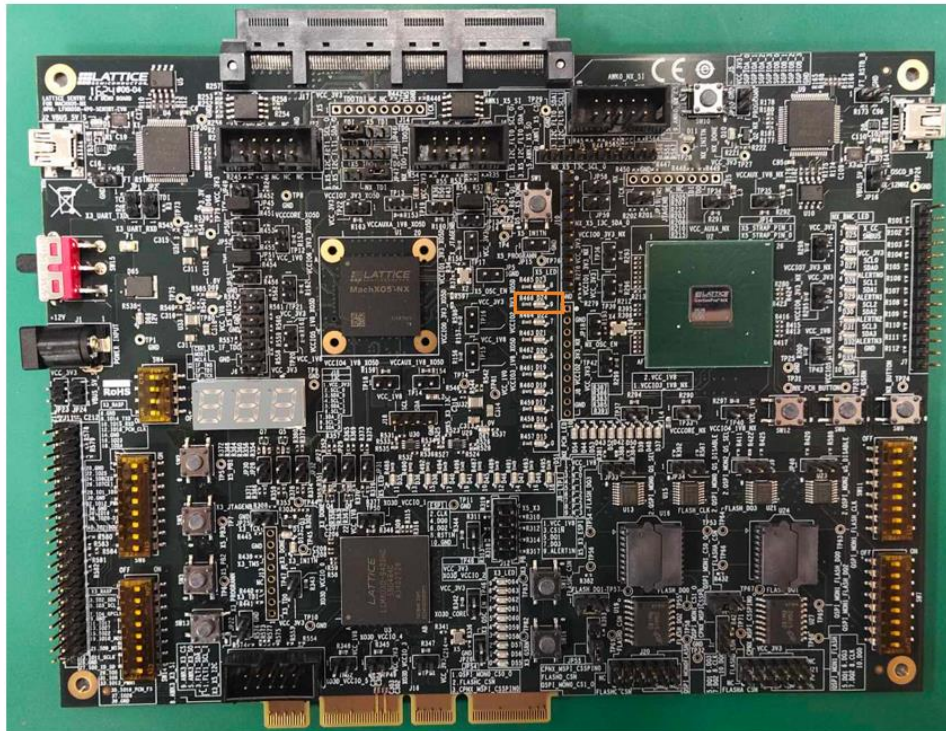


Figure 6.17. D24 LED Blink

5. After selecting the M-PESTI demonstration, the following can be observed on the UART terminal:
 - a. The target T0 and target T1 discovery payloads are printed on the UART terminal. These values originate from the MachXO5-NX FPGA, which acts as two M-PESTI target devices.
 - b. The Virtual Wire In (VW-IN) values from the target are also printed on the UART terminal. The target T0 sends a byte of data while the target T1 sends two bytes of data. In the current demonstration, these values originate from an internal counter on each M-PESTI target. In future versions of the demonstration, alternative sources can provide these values, and the number of bytes being sent can be adjusted.
 - c. The phrase Broadcast Command is also printed, signifying that a Broadcast Command 0x00 is sent by the initiator to all targets.

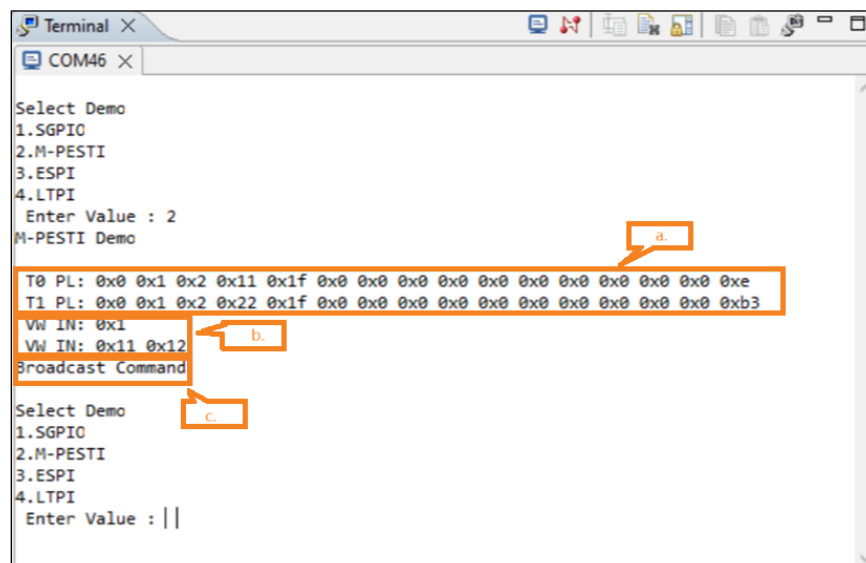


Figure 6.18. M-PESTI Demonstration Terminal

6.1.4. SGPIO Demonstration for HPM

Figure 6.19 is a diagram for the HPM SGPIO demonstration. It shows the UART terminal and the HPM template with only SGPIO demonstration-related components included.

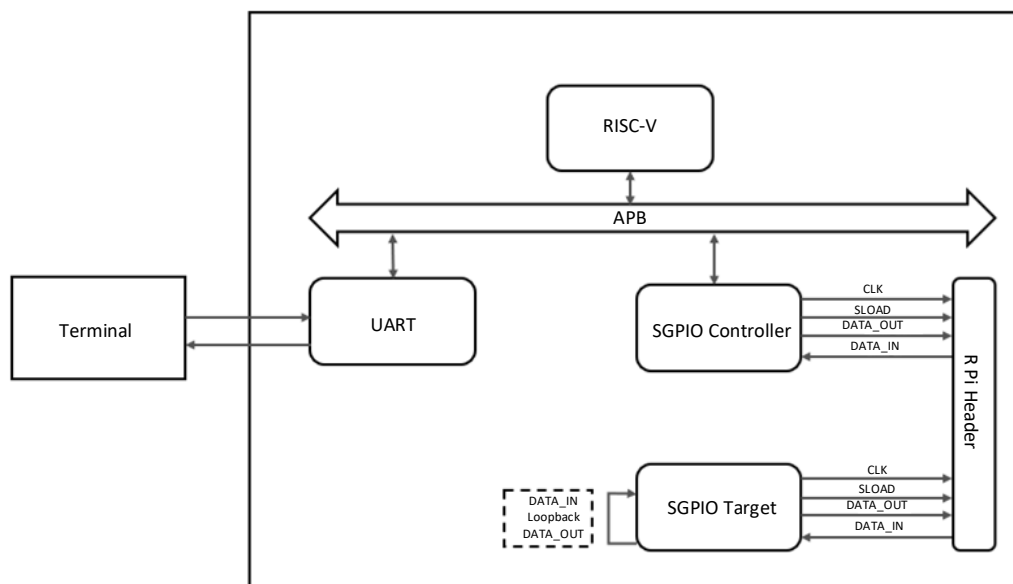


Figure 6.19. HPM SGPIO Demonstration Diagram

1. Connect the jumpers for RPi Header, as shown in Figure 6.20.

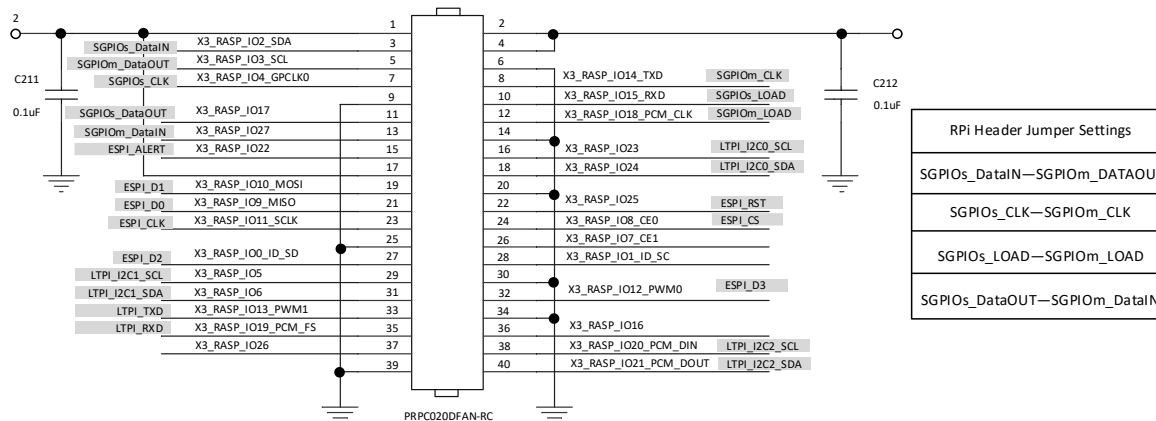


Figure 6.20. RPi Header Jumper Settings

2. Power on the HPM hardware to turn on the SGPIO controller and target. The SGPIO target has a loopback connection for its Data_IN and Data_OUT.
3. Make the following selection for the SGPIO demonstration, as shown in Figure 6.21.

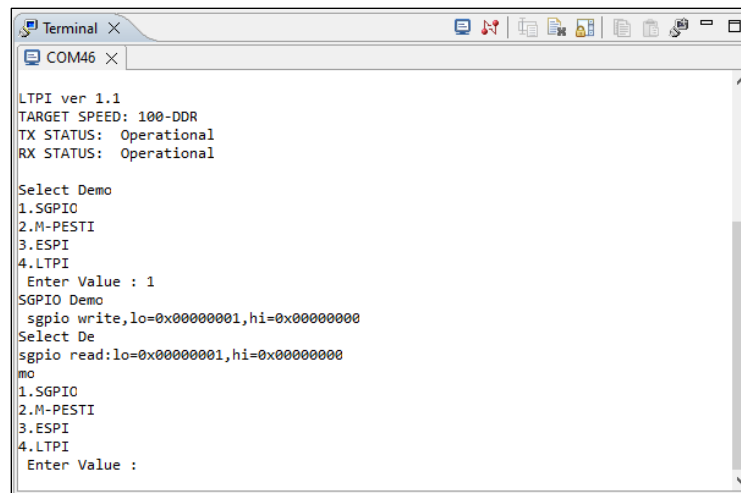


Figure 6.21. UART SGPIO Demonstration Selection

- The firmware writes 0x00000001 to in_lo and 0x00000000 to in_hi from the SGPIO controller. Below is the function for the SGPIO demonstration (Figure 6.22).

```
184 void sgpio_demo() {
185
186     printf("\nSGPIO Demo\n");
187     sgpio_inst_p->wr_cache_ext = 0; //hi = in_lo + 1;
188     sgpio_inst_p->wr_cache = sgpio_inst_p->wr_cache + 1; //low in_hi = 1;
189     printf(" sgpio write, lo=0x%08x, hi=0x%08x", sgpio_inst_p->wr_cache, sgpio_inst_p->wr_cache_ext);
190     sgpio_write_ext(&sgpio_inst, sgpio_inst_p->wr_cache, sgpio_inst_p->wr_cache_ext);
191
192
193 }
```

Figure 6.22. SGPIO Demonstration Code

- No forced reading of SGPIO is done since the code is waiting for interrupt for SGPIO to be triggered. Any changes in the values of SGPIO trigger the interrupt for SGPIO with a call back of reading it. Below are the initialization of SGPIO interrupt and the callback function (Figure 6.23 and Figure 6.24).

```
499 //Initializing sgpio master with base address 0x00133000
500 sgpio_init(&sgpio_inst, SGPIO0_INST_BASE_ADDR, 0, 0);
501 //To register the callback function for sgpio
502 sgpio_register_callback(&sgpio_inst, sgpio_callback, NULL);
503 //Register th ISR for SGPIO
504 pic_isr_register(SGPIO0_INST_IRQ, sgpio_callback, (void *)&sgpio_inst);
505 //Enable the interrupt for SGPIO
506 sgpio_enable_interrupt(&sgpio_inst, 1);
```

Figure 6.23. SGPIO Interrupt Initialization

```
173 static void sgpio_callback(void *ctx) {
174
175     sgpio_reg_t out_lo = 0;
176     sgpio_reg_t out_hi = 0;
177     sgpio_clear_interrupt(&sgpio_inst);
178     sgpio_read_ext(&sgpio_inst, &out_lo, &out_hi);
179     printf("\nsgpio read: lo=0x%08x, hi=0x%08x\n", out_lo, out_hi);
180
181
182 }
183
```

Figure 6.24. SGPIO Interrupt Callback Function

- The value for in_lo increments by 1 if the SGPIO demonstration is selected again.

6.1.5. SGPIO Demonstration for SCM

Figure 6.25 is the diagram for SCM SGPIO demonstration. It shows the UART terminal and the SCM template with only SGPIO demonstration-related components included.

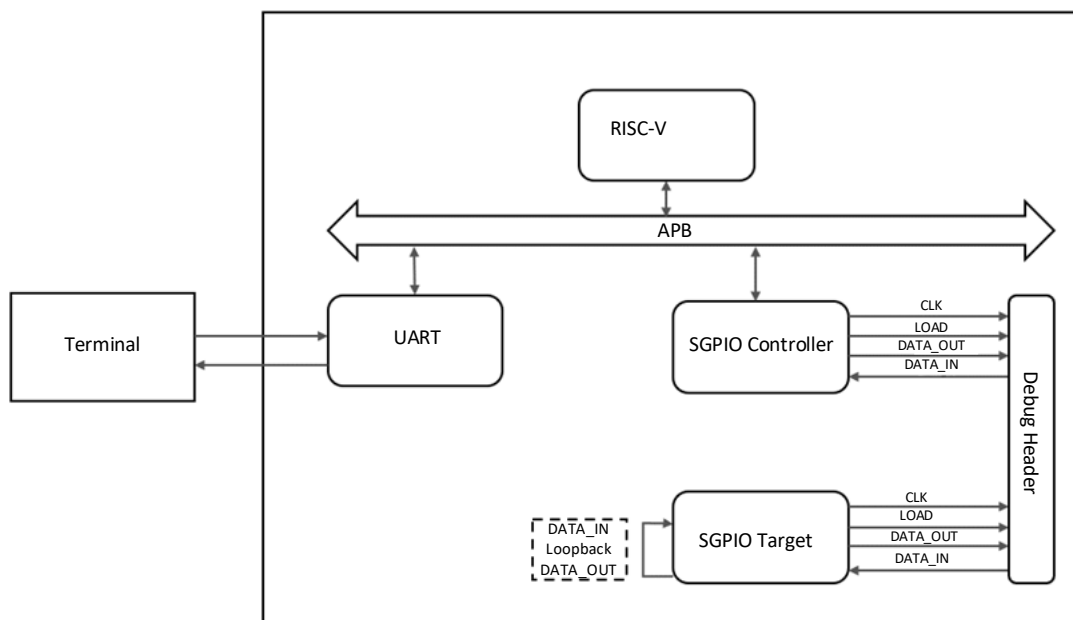


Figure 6.25. SCM SGPIO Demonstration Diagram

1. Connect the jumpers for RPi Header based on Figure 6.26.

		Debug_0			
	1	VCCIO0	VCCIO1	2	
UART0_RXD	3	F8	L14	4	UART0_TXD
	5	D9	M14	6	
	7	F9	L15	8	
	9	E11	N16	10	
	11	GND	GND	12	
LTPI_I2C0_SCL	13	D8	L16	14	LTPI_I2C0_SDA
LTPI_I2C1_SCL	15	E9	M15	16	LTPI_I2C1_SDA
LTPI_I2C2_SCL	17	D6	N14	18	LTPI_I2C2_SDA
LTPI_TXD	19	E7	M16	20	LTPI_RXD
	21	VCCIO2	VCCIO3	22	
	23	F15	R11	24	
	25	J16	T12	26	
	27	J15	R13	28	
	29	G15	T14	30	
SGPIOm_CLK	31	H15	R7	32	SGPIOs_CLK
SGPIOm_DataOUT	33	K15	P7	34	SGPIOs_DataOUT
SGPIOm_LOAD	35	K16	N10	36	SGPIOs_LOAD
SGPIOm_DataIN	37	J14	M11	38	SGPIOs_DataIN
	39	VCCIO5	VCC_RPI	40	

Figure 6.26. RPi Header Jumper Settings

2. Power on the SCM hardware to turn on the SGPIO controller and target. The SGPIO target has a loopback connection for its Data_IN and Data_OUT.
3. Make the following selection for the SGPIO demonstration, as shown in Figure 6.27.

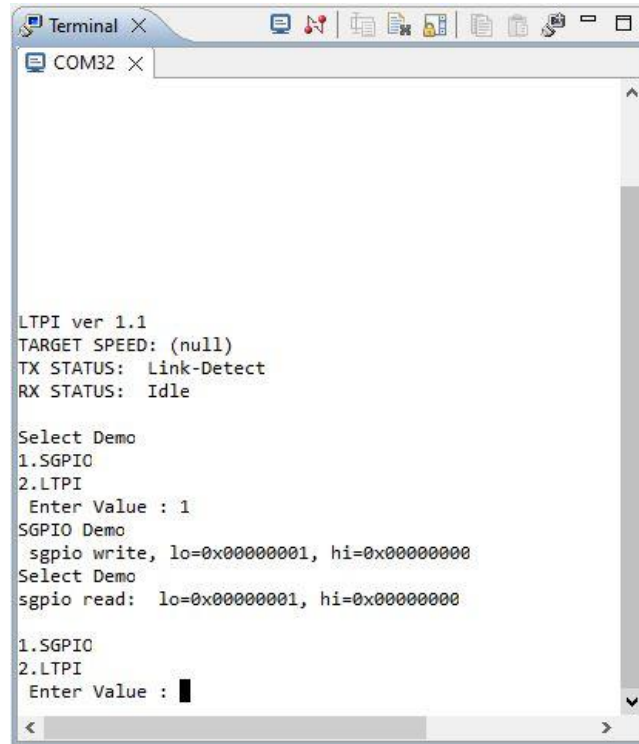


Figure 6.27. UART SGPIO Demonstration Selection

- The firmware writes 0x00000001 to in_lo and 0x00000000 to in_hi from SGPIO Controller. Figure 6.28 shows the function for the SGPIO demonstration.

```

184 void sgpio_demo() {
185     printf(" \r\nSGPIO Demo\r\n");
186     sgpio0_inst_p->wr_cache_ext = 0; //hi = in_lo + 1;
187     sgpio0_inst_p->wr_cache = sgpio0_inst_p->wr_cache + 1; //low in_hi = 1;
188     printf(" sgpio write,lo=0x%08x,hi=0x%08x", sgpio0_inst_p->wr_cache, sgpio0_inst_p->wr_cache_ext);
189     sgpio_write_ext(&sgpio_inst, sgpio0_inst_p->wr_cache, sgpio0_inst_p->wr_cache_ext);
190 }
191
192
193
194

```

Figure 6.28. SGPIO Demonstration Code

- No forced reading of SGPIO is done since the code is waiting for the interrupt for SGPIO to be triggered. Any changes in the values of SGPIO trigger the interrupt for SGPIO with a call back of reading it. Below are the initialization of SGPIO interrupt and the callback function (Figure 6.29 and Figure 6.30).

```

499 //Initializing sgpio master with base address 0x00133000
500 sgpio_init(&sgpio_inst, SGPIOM0_INST_BASE_ADDR, 0, 0);
501 //To register the callback function for sgpio
502 sgpio_register_callback(&sgpio_inst, sgpio_callback, NULL);
503 //Register th ISR for SGPIO
504 pic_isr_register(SGPIOM0_INST_IRQ, sgpio_callback, (void *)&sgpio_inst);
505 //Enable the interrupt for SGPIO
506 sgpio_enable_interrupt(&sgpio_inst, 1);

```

Figure 6.29. SGPIO Interrupt Initialization

```
173 static void sgpio_callback(void *ctx) {  
174  
175     sgpio_reg_t out_lo = 0;  
176     sgpio_reg_t out_hi = 0;  
177     sgpio_clear_interrupt(&sgpio_inst);  
178     sgpio_read_ext(&sgpio_inst, &out_lo, &out_hi);  
179     printf("\nsgpio read: lo=0x%08x, hi=0x%08x\r\n", out_lo, out_hi);  
180  
181  
182 }  
183
```

Figure 6.30. SGPIO Interrupt Callback Function

6. The value for in_lo increments by 1 if the SGPIO demonstration is selected again.

7. IP User Guide Reference Documents

7.1. LTPI

LTPI is a protocol and interface designed for tunneling various low-speed signals between the HPM and SCM. The LTPI protocol goes over the LVDS electrical interfaces. This is the next generation protocol for DC-SCM 2.0 as the replacement to two Serial GPIO (SGPIO) interfaces. 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 CPLD and SCM CPLD.

7.1.1. Features

- Compliant with Open Compute Project® (OCP) DC-SCM 2.0 LTPI 1.0 Specifications.
- Link initialization, discovery, and negotiation.
- Supports Multi-Channel Serial Interface.
- Supports LVDS and subLVDS.
- Supports up to five channels aggregation/disaggregation in total.
- Supports GPIO, I2C, UART, OEM, and data channel aggregation.
- For I2C interface, each device can be configured as Controller or Target.
- Supports up to 800 Mbps LVDS data rate for MachXO3™ and Mach™-NX family devices.
- Supports up to 1200 Mbps LVDS data rate for MachXO5-NX family devices.
- Supports AMBA 3 APB Protocol v1.0 for register access of the soft IP and data channel.
- Supports PREADY, a ready signal to indicate completion of an APB transfer.
- PSLVERR is only supported in data channel-related access, with an error signal indicating failure of a transfer.

7.1.2. Reference Guide

[DC-SCM LTPI IP \(FPGA-IPUG-02200\)](#)

7.2. eSPI Target

Enhanced Serial Peripheral Interface Target IP is compliant with the Intel eSPI specifications. It has its own virtual wire channel in the user interface while implementing peripheral channels, namely, Out of Band (OOB) Message Channel and Flash Access Channel in FIFO that are accessible by the APB or AHB-Lite interface.

7.2.1. Features

- Supports all eSPI Commands.
- Supports all required error detection in eSPI specification.
- Supports single, dual, and quad SPI mode.
- CRC check
- Supports APB/AHB-Lite user interface.
- Simple implementation interface for virtual wire channel transactions
- GPIO Expander interface for virtual wire channel transactions
- Peripheral channel transactions controlled by the host using APB/AHB-Lite interface
- OOB Message channel transactions controlled by the host using APB/AHB-Lite interface
- Flash access channel transactions controlled by the host using APB/AHB-Lite interface
- No response error detection in eSPI command
- Fatal error detection in eSPI command
- Soft resets: CSR, SPI, and FIFO

7.2.2. Reference Guide

- [Enhanced Serial Peripheral Interface \(eSPI\) Specifications](#)
- [eSPI Target IP \(FPGA-IPUG-02260\)](#)

7.3. M-PESTI Initiator

This is a DC-MHS version 1.0 spec-compliant M-PESTI Initiator IP.

7.3.1. Features

- M-PESTI Initiator IP and M-PESTI Target test component communicate through half-duplex bidirectional UART protocol at 250k BAUD rate, 8-bit data, 1-bit odd parity, 1 start bit, and 1 stop bit of M-PESTI line.
- Supports Static Discovery payload with CRC-8 payload checksum.
- Supports one-controller-to-many-target mode.
- Supports configurable number of M-PESTI line.
- Supports broadcast power break command to all connecting M-PESTI Targets.
- Supports auto-trigger Static Discovery Payload request command to all Targets on round robin manner during the Discovery phase.

7.3.2. Reference Guide

- [M-PESTI Initiator Reference Design \(FPGA-RD-02274\)](#)
- [M-PESTI Initiator IP \(FPGA-IPUG-02258\)](#)

7.4. SGPIO Controller and Target

SGPIO is a four-signal bus used for data serializer and deserializer between a host bus adapter (HBA) and a backplane. The IPUG of this IP is included in the template package.

7.5. RISC-V SM

The Lattice Semiconductor RISC-V SM CPU IP contains a 32-bit RISC-V processor core and optional submodules – Timer and Programmable Interrupt Controller (PIC). The CPU core supports the RV32I instruction set, external interrupt, and debug feature, which is JTAG – IEEE 1149.1 compliant. The Timer submodule is a 64-bit real time counter, which compares a real-time register with another register to assert the timer interrupt. The PIC submodule aggregates up to eight external interrupt inputs into one external interrupt. The submodule registers are accessed by the processor core using a 32-bit AHB-Lite interface.

7.5.1. Features

The RISC-V SM CPU IP has the following features:

- RV32I instruction set
- Five stages of pipelines
- Supports the AHB-Lite bus standard for instruction/data port.
- Optional debug through Gnu Debugger (GDB) and Open On-Chip Debugger (OpenOCD)
- Optional Timer module
- Optional PIC module
- Interrupt and exception handling with Machine mode in RISC-V privileged ISA Specification Revision 1.10
- About 0.5 DMIPS/MHz performance
- Around 40 MHz in MachXO2™ devices, 50 MHz in MachXO3D devices, and 100 MHz in CrossLink™-NX devices. Tested on Hello World templates provided by Lattice Propel.

7.5.2. Reference Guide

[RISC-V SM CPU IP \(FPGA-IPUG-02240\)](#)

8. Resource Utilization

Table 8.1. Resource Utilization

Module/IP	Utilization		Note
	LUTS	EBR	
RISC-V SM (cpu0)	1396	4	Using RISC-V-SM, debug mode enabled.
System Memory (sysmem0)	179	16	Approximately 4 KB memory size
eSPI	628	2	For 64-bit virtual wire-only configuration
LTPI	2883	0	The utilization of LTPI varies depending on the target configuration
M-PESTI	978	4	The utilization is based on four Target configurations.
SGPIO Controller	112	0	34-bit width
SGPIO Target	35	0	34-bit width
GPIO0	332	0	For debug and demonstration purposes. Used as a virtual wire for LTPI.
GPIO1	227	0	For debug and demonstration purposes. Used as a virtual wire for eSPI.
GPIO2	323	0	For debug and demonstration purposes. Used as a virtual wire for eSPI.
UART	199	0	For debug and demonstration purposes
CSR0	56	0	For LTPI CSR/Dynamic PLL access
APB	176	0	—
AHB-Lite to APB Bridge	286	0	—
AHB-Lite	100	0	—

9. Design Guidelines, Known Issues, and Limitation

9.1. LTPI

Below are a few guidelines on using the LTPI IP as part of the SoC template. Note that by default, these are already met by the design.

9.1.1. SoC Template (IP)

1. Check that the features of both SCM and HPM match (Figure 9.1).

▼ Feature Capability	
Enable Full OEM Capabilities Type	☐
Capabilities Type	8'h00
Default Feature Capability 0 in Hex (0x)	4f00080f
Default Feature Capability 1 in Hex (0x)	00002a00
Automatically move to Configuration State	☒

Figure 9.1. Capabilities Match

2. Enable the Automatically move to Configuration State option of the IP (Figure 9.2). This is only for SCM.

▼ Feature Capability	
Enable Full OEM Capabilities Type	☐
Capabilities Type	8'h00
Default Feature Capability 0 in Hex (0x)	4f00080f
Default Feature Capability 1 in Hex (0x)	00002a00
Automatically move to Configuration State	☒

Figure 9.2. Enable Automatically Move to Configuration State

9.1.2. Diamond Project

- All LTPI pins should be mapped to an A/B pin pair.
Note: Only the TRUE (A) pin needs to be assigned. The software infers the COMP (B) pin.
- For the input LTPI clock and data pins, ensure that both are mapped to a Bank 2 pin.
- For the clock port, it should be mapped to a PCLK pair.
- For the input LTPI clock and data pins, ensure that both are mapped to a Bank 0 pin.
- Ensure that the UPLL CLKIN matches the connected external clock's frequency. The LTPI demonstration board uses 12 MHz while the Sentry 4.0 board uses 25 MHz.

9.1.3. Firmware (Propel C Project)

Currently, only the 25 MHz default SDR and 100 MHz DDR option are enabled in the firmware. If the design requires another speed, uncomment on the pll_init function of main.c.

9.2. eSPI

Upon power up or reset, the eSPI Target's Virtual Wire Channel Enable is set to 0. So, it is required to perform Set Configuration for Channel 1 (Virtual Wire) Capabilities and Configuration at address 0x020, as mentioned in step 4 of the [eSPI Demonstration](#) section.

9.3. M-PESTI

When regenerating the M-PESTI Initiator IP within Lattice Propel Builder, the file name extension *.v needs to be manually changed to *.sv before synthesizing in the Lattice Diamond software ([Figure 9.3](#)). Otherwise, synthesis fails.



Figure 9.3. File Name Extension Manual Edit

The Sentry 4.0 demonstration board can either have an LFMX05-100T device or an LFMX05-55TD device. To program the board with the latter device, follow the settings shown in [Figure 9.4](#) and use the LFMX05-55TD device version of the programming file.

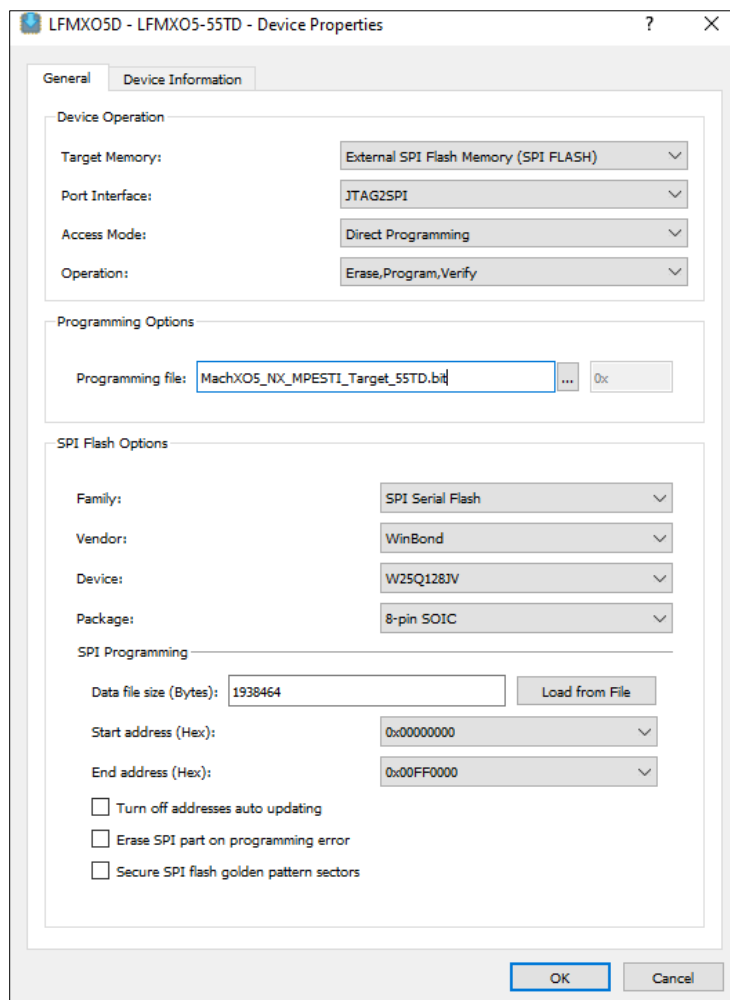


Figure 9.4. LFMX05-55TD Programmer Settings

10. IP Versions

Table 10.1. IP Versions

IP	Version
AHB-Lite Interconnect	1.3.0
AHB-Lite to APB Bridge	1.1.0
APB Interconnect	1.2.0
eSPI Target	2.0.0.02
GPIO	1.6.1
M-PESTI Initiator	1.0.8
SGPIO Controller	1.1.0
SGPIO Target	1.1.0
LTPI	1.5.1
UART	1.3.0
RISC-V SM	1.5.0

11. Design Template Package

Figure 11.1 shows the design template package.

- HPM Folder:
 - .metadata – generated by the Lattice Propel builder software. Do not remove.
 - HPM_c – contains the HPM firmware that can be modified using Lattice Propel design environment.
 - HPM_soc – contains the design template diagram that can be opened in Lattice Propel Builder as well as the Lattice Diamond project to generate the configuration file.
- Misc Folder:
 - eSPI – contains the Promira firmware that can be used for the demonstration.
 - IPKs – contains solution-specific IPs that are used in the template.
 - MPESTI – contains the pre-built JED file for the MachXO5-NX M-PESTI target design.
- SCM Folder:
 - .metadata – generated by the Lattice Propel Builder software. Do not remove.
 - SCM_c – contains the HPM firmware that can be modified using Lattice Propel design environment.
 - SCM_soc – contains the design template diagram that can be opened in Lattice Propel Builder as well as the Lattice Diamond project to generate the configuration file.

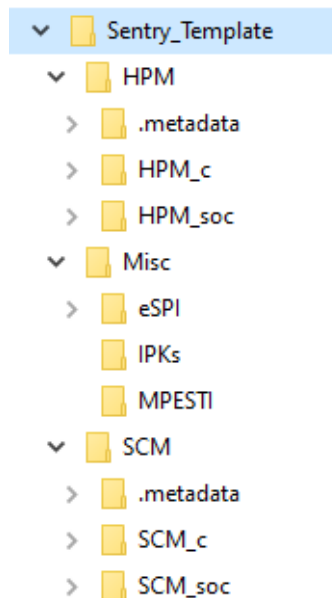


Figure 11.1. Sentry 4.0 Design Package

References

- [AHB-Lite Interconnect Module – Lattice Propel Builder \(FPGA-IPUG-02051\)](#)
- [AHB-Lite to APB Bridge Module – Lattice Propel Builder \(FPGA-IPUG-02053\)](#)
- [APB Interconnect Module – Lattice Propel Builder \(FPGA-IPUG-02054\)](#)
- [eSPI Target IP \(FPGA-IPUG-02260\)](#)
- [GPIO IP Core \(FPGA-IPUG-02076\)](#)
- [M-PESTI Initiator IP \(FPGA-IPUG-02258\)](#)
- [DC-SCM LTPI IP \(FPGA-IPUG-02200\)](#)
- [RISC-V SM CPU IP \(FPGA-IPUG-02240\)](#)
- [SGPIO Controller/Target IP \(FPGA-IPUG-02247\)](#)
- [UART IP \(FPGA-IPUG-02105\)](#)
- [Lattice Radiant Timing Constraints Methodology \(FPGA-AN-02059\)](#)
- [Lattice Radiant Software User Guide](#)
- [MachXO5-NX Family Devices](#) web page
- [MachXO3D Family Devices](#) web page
- [Lattice Sentry Solution](#) web page
- [Lattice Diamond](#) Software web page
- [Lattice Propel Design Environment](#) web page
- [Lattice Radiant](#) FPGA design software
- [Open Compute Project](#) web page
- [Lattice Insights](#) for Lattice Semiconductor training courses and learning plans

Technical Support Assistance

For assistance, submit a technical support case at www.latticesemi.com/techsupport.

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

Revision History

Revision 1.0, September 2025

Section	Change Summary
All	- Added the Open Compute Project logo to the cover page. - Updated the document title from <i>Lattice Sentry 4.0 SCM and HPM Design Template User Guide</i> to <i>Lattice Sentry 4.0 SCM and HPM Design Template for MachXO3D Devices User Guide</i> to differentiate this document from Lattice Sentry 4.0 SCM and HPM Design Template for MachXO5-NX Devices User Guide (FPGA-UG-02233).
Abbreviations in This Document	Updated this section.
Introduction	Added <i>This solution is OCP Ready™</i> .
IP User Guide Reference Documents	Updated the section title of the M-PESTI Initiator section to its current.
References	Added a reference to the Open Compute Project web page.

Revision 0.80, August 2024

Section	Change Summary
All	Preliminary release.



www.latticesemi.com