Lattice Sentry 4.0 SCM and HPM Design Template for MachXO5-NX Devices

Preliminary User Guide

FPGA-UG-02233-0.80

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

Figures

Tables

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
STI	Sideband Tunneling Interface
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) design applications on complex programmable logic devices (CPLD). It aims to help you design an SCM and HPM CPLD design with limited knowledge and experience in Verilog and 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 MachXO5-NX device, LFMXO5-25-8BBG400C. This solution is OCP Ready™.

2. Block Diagram

Figure 2.1 and Figure 2.2 show the functional block diagram of the 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.

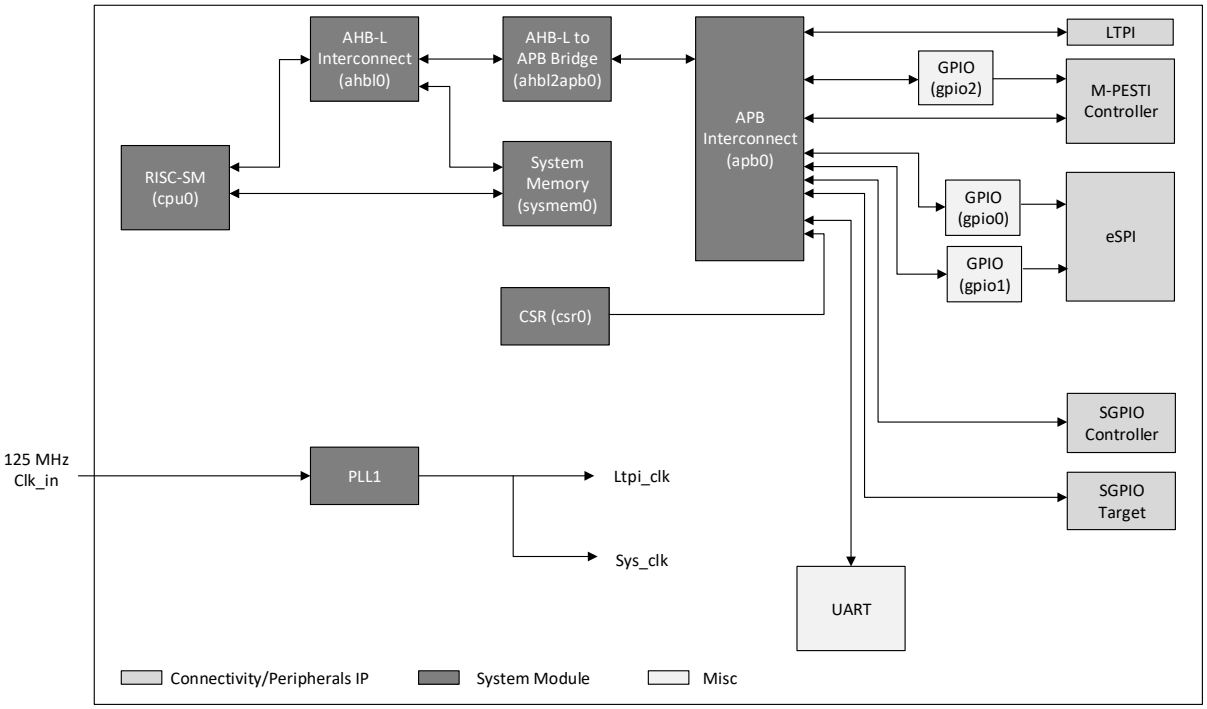


Figure 2.1. HPM Design Template Block Diagram

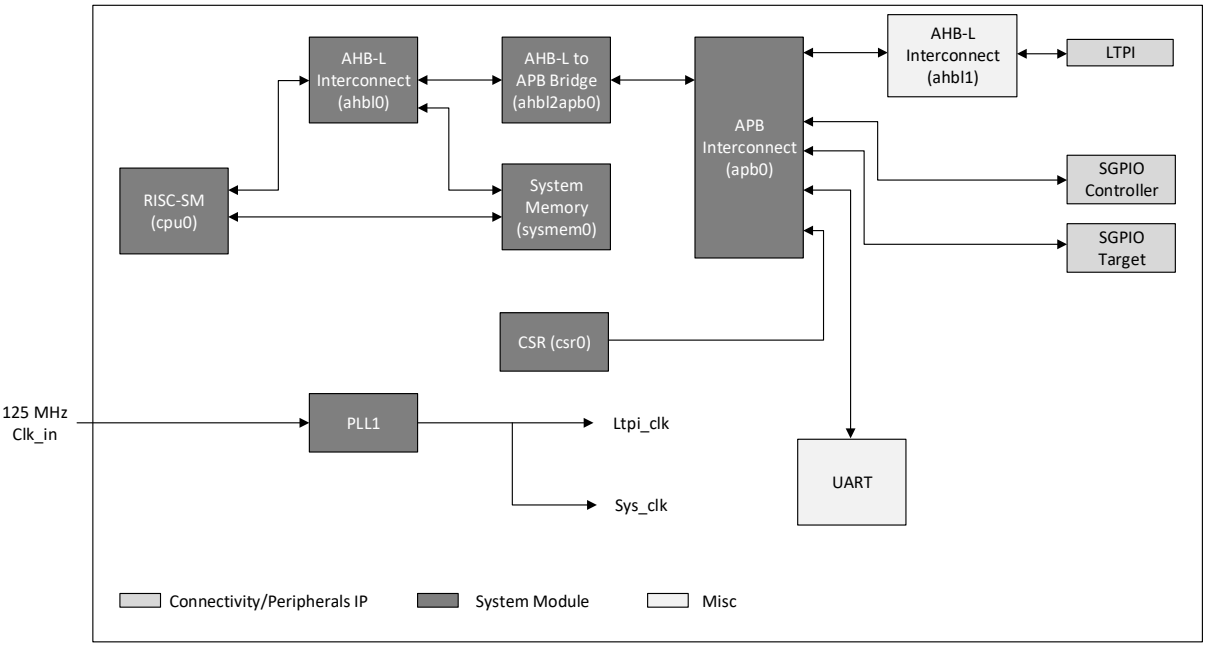


Figure 2.2. SCM Design Template Block Diagram

3. Pin-out for HPM and SCM Design

The HPM (MachXO5-NX Development Board) and SCM (MachXO5-NX Development Board) sections describe the pin-out for the MachXO5-NX device used for the demonstration.

3.1. HPM (MachXO5-NX Development Board)

Table 3.1. HPM Pin-out for MachXO5-NX Development Board

Port Name	Port Map	MachXO5-NX Map	Description	Note
Terminal Access				
uart0_inst_rxd_o_port	E7	EXPCON_IO26	Terminal UART RX	—
uart0_inst_txd_o_port	A9	EXPCON_IO23	Terminal UART TX	—
LTPI				
ltpi0_inst_uart_rx_o_portbus[0]	E8	EXPCON_IO27	CH0 UART RX	—
ltpi0_inst_uart_tx_i_portbus[0]	E6	EXPCON_IO28	CH0 UART TX	—
ltpi0_inst_lvds_tx_data_o_port	Y14	CH1_DATA1_P	LVDS TX Data	—
ltpi0_inst_lvds_tx_clk_o_port	Y12	CH3_DCK_P	LVDS TX Clock	—
ltpi0_inst_lvds_rx_data_i_port	N11	CH3_DATA1_P	LVDS RX Data	—
ltpi0_inst_lvds_rx_clk_i_port	Y11	CH1_DCK_P	LVDS RX Clock	—
ltpi0_inst_ll_gpio_i_portbus [0]	T1	DIP_SW1	Low Latency Input 0	—
ltpi0_inst_ll_gpio_i_portbus [1]	T2	DIP_SW2	Low Latency Input 1	—
ltpi0_inst_ll_gpio_i_portbus [2]	T3	DIP_SW3	Low Latency Input 2	—
ltpi0_inst_ll_gpio_i_portbus [3]	T4	DIP_SW4	Low Latency Input 3	—
ltpi0_inst_ll_gpio_o_portbus [0]	R3	XLED0	Low Latency Output 0	—
ltpi0_inst_ll_gpio_o_portbus [1]	R2	XLED1	Low Latency Output 1	—
ltpi0_inst_ll_gpio_o_portbus [2]	R1	XLED2	Low Latency Output 2	—
ltpi0_inst_ll_gpio_o_portbus [3]	P7	XLED3	Low Latency Output 3	—
ltpi0_inst_i2c_sda_io_portbus [0]	A6	EXPCON_IO16	CH0 I2C SDA	—
ltpi0_inst_i2c_scl_io_portbus [0]	B6	EXPCON_IO17	CH0 I2C SCL	—
ltpi0_inst_i2c_sda_io_portbus [1]	A7	EXPCON_IO18	CH1 I2C SDA	—
ltpi0_inst_i2c_scl_io_portbus [1]	A8	EXPCON_IO19	CH1 I2C SCL	—
ltpi0_inst_i2c_sda_io_portbus [2]	B8	EXPCON_IO21	CH2 I2C SDA	—
ltpi0_inst_i2c_scl_io_portbus [2]	B9	EXPCON_IO22	CH2 I2C SCL	—
eSPI				
espi0_inst_espi_clk_i_port	E16	EXPCON_OSC	eSPI Clock	PULLMODE: DOWN
espi0_inst_espi_cs_n_i_port	C3	EXPCON_IO2	Chip Select	PULLMODE: NONE
espi0_inst_espi_rst_n_i_port	A4	EXPCON_IO4	Reset	PULLMODE: UP
espi0_inst_espi_alert_n_o_port	F6	EXPCON_IO6	Alert Signal	PULLMODE: NONE
espi0_inst_espi_data_io_portbus[0]	B2	EXPCON_IO8	Data 0	PULLMODE: UP
espi0_inst_espi_data_io_portbus[1]	B3	EXPCON_IO10	Data 1	PULLMODE: UP
espi0_inst_espi_data_io_portbus[2]	B4	EXPCON_IO12	Data 2	PULLMODE: UP
M-PESTI				
mpesti0_inst_mpesti_io_portbus[0]	G7	PMOD0_1	—	PULLMODE: UP
mpesti0_inst_mpesti_io_portbus[1]	G9	PMOD0_2	—	PULLMODE: UP
mpesti0_inst_mpesti_io_portbus[2]	G8	PMOD0_3	—	PULLMODE: UP
mpesti0_inst_mpesti_io_portbus[3]	H8	PMOD0_4	—	PULLMODE: UP
mpesti0_inst_mpesti_io_portbus[4]	F7	PMOD0_5	—	PULLMODE: UP
mpesti0_inst_mpesti_io_portbus[5]	F9	PMOD0_6	—	PULLMODE: UP
mpesti0_inst_mpesti_io_portbus[6]	F8	PMOD0_7	—	PULLMODE: UP
mpesti0_inst_mpesti_io_portbus[7]	H9	PMOD0_8	—	PULLMODE: UP

Port Name	Port Map	MachXO5-NX Map	Description	Note
mpesti0_inst_mpesti_io_portbus[8]	E11	PMOD1_1	—	PULLMODE: UP
mpesti0_inst_mpesti_io_portbus[9]	D13	PMOD1_2	—	PULLMODE: UP
mpesti0_inst_mpesti_io_portbus[10]	D15	PMOD1_3	—	PULLMODE: UP
mpesti0_inst_mpesti_io_portbus[11]	C16	PMOD1_4	—	PULLMODE: UP
mpesti0_inst_mpesti_io_portbus[12]	D11	PMOD1_5	—	PULLMODE: UP
mpesti0_inst_mpesti_io_portbus[13]	C13	PMOD1_6	—	PULLMODE: UP
mpesti0_inst_mpesti_io_portbus[14]	C15	PMOD1_7	—	PULLMODE: UP
mpesti0_inst_mpesti_io_portbus[15]	D16	PMOD1_8	—	PULLMODE: UP
mpesti0_inst_mpesti_io_portbus[16]	H14	EXPCON_IO30	—	PULLMODE: UP
mpesti0_inst_mpesti_io_portbus[17]	G14	EXPCON_IO31	—	PULLMODE: UP
mpesti0_inst_mpesti_io_portbus[18]	H15	EXPCON_IO32	—	PULLMODE: UP
mpesti0_inst_mpesti_io_portbus[19]	G18	EXPCON_IO33	—	PULLMODE: UP
mpesti0_inst_mpesti_io_portbus[20]	H16	EXPCON_IO34	—	PULLMODE: UP
mpesti0_inst_mpesti_io_portbus[21]	G19	EXPCON_IO35	—	PULLMODE: UP
mpesti0_inst_mpesti_io_portbus[22]	H20	EXPCON_IO36	—	PULLMODE: UP
mpesti0_inst_mpesti_io_portbus[23]	H19	EXPCON_IO37	—	PULLMODE: UP
mpesti0_inst_mpesti_io_portbus[24]	E4	EXPCON_IO1	—	PULLMODE: UP
mpesti0_inst_mpesti_io_portbus[25]	C2	EXPCON_IO3	—	PULLMODE: UP
mpesti0_inst_mpesti_io_portbus[26]	E5	EXPCON_IO5	—	PULLMODE: UP
mpesti0_inst_mpesti_io_portbus[26]	C5	EXPCON_IO7	—	PULLMODE: UP
mpesti0_inst_mpesti_io_portbus[27]	A2	EXPCON_IO9	—	PULLMODE: UP
mpesti0_inst_mpesti_io_portbus[28]	A3	EXPCON_IO11	—	PULLMODE: UP
mpesti0_inst_mpesti_io_portbus[29]	D5	EXPCON_IO13	—	PULLMODE: UP
mpesti0_inst_mpesti_io_portbus[30]	B5	EXPCON_IO15	—	PULLMODE: UP
mpesti0_inst_mpesti_io_portbus[31]	E4	EXPCON_IO1	—	PULLMODE: UP
SGPIO				
sgpios0_inst_iGSXDataIn_port	J15	EXPCON_IO40	SGPIO Target Data In	Connected with Controller Data Out for the demonstration.
sgpios0_inst_oGSXDataOut_port	J12	EXPCON_IO44	SGPIO Target Data Out	Connected with Controller Data In for the demonstration.
sgpios0_inst_iGSXClk_port	E17	EXPCON_CLKIN	SGPIO Target Clock	Connected with Controller Clock for the demonstration.
sgpios0_inst_inGSXLoad_port	J13	EXPCON_IO42	SGPIO Target Load	Connected with Controller Load for the demonstration.
sgpiom0_inst_sdatain_i_port	H13	EXPCON_IO45	SGPIO Controller Data In	Connected with Target Data Out for the demonstration.
sgpiom0_inst_sdataout_o_port	J16	EXPCON_IO41	SGPIO Controller Data Out	Connected with Target Data In for the demonstration.
sgpiom0_inst_sclock_o_port	D20	EXPCON_CLKOUT	SGPIO Controller Clock	Connected with Target Clock for the demonstration.
sgpiom0_inst_sload_o_port	J14	EXPCON_IO43	SGPIO Controller Load	Connected with Target Load for the demonstration.

3.2. SCM (MachXO5-NX Development Board)

Table 3.2. SCM Pin-out for MachXO5-NX Development Board

Port Name	Port Map	MachXO5-NX Map	Description	Note
Terminal Access				
uart0_inst_rxd_o_port	E7	EXPCON_IO26	Terminal UART RX	—
uart0_inst_txd_o_port	A9	EXPCON_IO23	Terminal UART TX	—
LTPI				
ltpi0_inst_uart_rx_o_portbus[0]	E8	EXPCON_IO27	CH0 UART RX	—
ltpi0_inst_uart_tx_i_portbus[0]	E6	EXPCON_IO28	CH0 UART TX	—
ltpi0_inst_lvds_tx_data_o_port	Y14	CH1_DATA1_P	LVDS TX Data	—
ltpi0_inst_lvds_tx_clk_o_port	Y12	CH3_DCK_P	LVDS TX Clock	—
ltpi0_inst_lvds_rx_data_i_port	N11	CH3_DATA1_P	LVDS RX Data	—
ltpi0_inst_lvds_rx_clk_i_port	Y11	CH1_DCK_P	LVDS RX Clock	—
ltpi0_inst_ll_gpio_i_portbus [0]	T1	DIP_SW1	Low Latency Input 0	—
ltpi0_inst_ll_gpio_i_portbus [1]	T2	DIP_SW2	Low Latency Input 1	—
ltpi0_inst_ll_gpio_i_portbus [2]	T3	DIP_SW3	Low Latency Input 2	—
ltpi0_inst_ll_gpio_i_portbus [3]	T4	DIP_SW4	Low Latency Input 3	—
ltpi0_inst_ll_gpio_o_portbus [0]	R3	XLED0	Low Latency Output 0	—
ltpi0_inst_ll_gpio_o_portbus [1]	R2	XLED1	Low Latency Output 1	—
ltpi0_inst_ll_gpio_o_portbus [2]	R1	XLED2	Low Latency Output 2	—
ltpi0_inst_ll_gpio_o_portbus [3]	P7	XLED3	Low Latency Output 3	—
ltpi0_inst_i2c_sda_io_portbus [0]	A6	EXPCON_IO16	CH0 I2C SDA	—
ltpi0_inst_i2c_scl_io_portbus [0]	B6	EXPCON_IO17	CH0 I2C SCL	—
ltpi0_inst_i2c_sda_io_portbus [1]	A7	EXPCON_IO18	CH1 I2C SDA	—
ltpi0_inst_i2c_scl_io_portbus [1]	A8	EXPCON_IO19	CH1 I2C SCL	—
ltpi0_inst_i2c_sda_io_portbus [2]	B8	EXPCON_IO21	CH2 I2C SDA	—
ltpi0_inst_i2c_scl_io_portbus [2]	B9	EXPCON_IO22	CH2 I2C SCL	—
SGPIO				
sgpios0_inst_iGSXDataIn_port	J15	EXPCON_IO40	SGPIO Target Data In	Connect with Controller Data Out for the demonstration.
sgpios0_inst_oGSXDataOut_port	J12	EXPCON_IO44	SGPIO Target Data Out	Connect with Controller Data In for the demonstration.
sgpios0_inst_iGSXClk_port	E17	EXPCON_CLKIN	SGPIO Target Clock	Connect with Controller Clock for the demonstration.
sgpios0_inst_inGSXLoad_port	J13	EXPCON_IO42	SGPIO Target Load	Connect with Controller Load for the demonstration.
sgpiom0_inst_sdatain_i_port	H13	EXPCON_IO45	SGPIO Controller Data In	Connect with Target Data Out for the demonstration.
sgpiom0_inst_sdataout_o_port	J16	EXPCON_IO41	SGPIO Controller Data Out	Connect with Target Data In for the demonstration.
sgpiom0_inst_sclock_o_port	D20	EXPCON_CLKOUT	SGPIO Controller Clock	Connect with Target Clock for the demonstration.
sgpiom0_inst_sload_o_port	J14	EXPCON_IO43	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. Figure 4.1 and Figure 4.2 show the SCM and HPM SoC equivalent on Lattice Radiant Software.

- SCM
  - SGPIO Controller
  - SGPIO Target
  - LTPI SCM
- HPM
  - SGPIO Controller
  - SGPIO Target
  - LTPI HPM
  - eSPI Target
  - M-PESTI Initiator

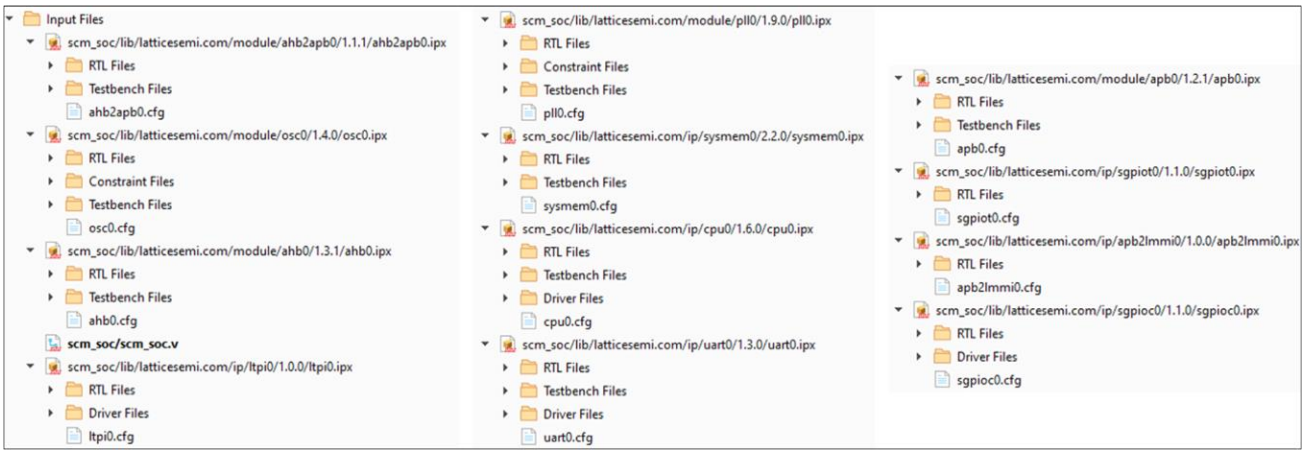


Figure 4.1. SCM SoC Equivalent on Lattice Radiant Software

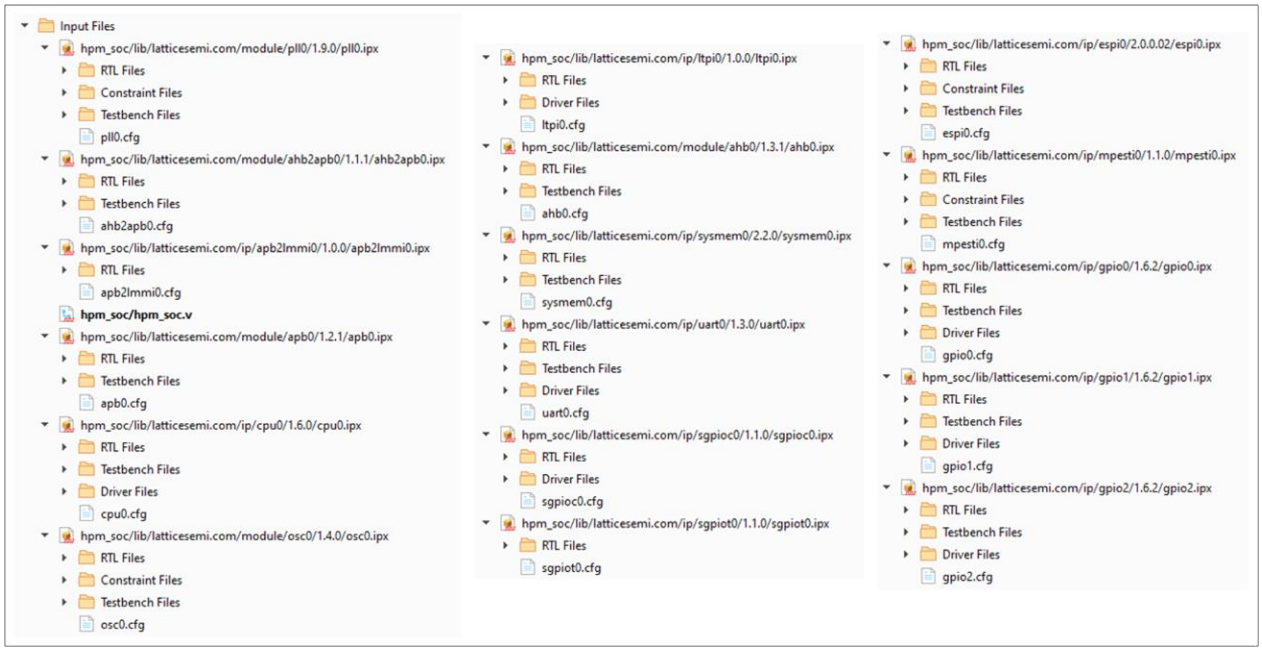


Figure 4.2. HPM SoC Equivalent on Lattice Radiant Software

Modifying the design template through Lattice Propel Builder and Lattice Radiant Software. Refer to the following web pages for more information.

- Lattice Propel Design Environment:
  https://www.latticesemi.com/products/designsoftwareandip/fpgaandlds/latticepropel
- Lattice Radiant Software:
  https://www.latticesemi.com/products/designsoftwareandip/fpgaandlds/radiant

4.2. Firmware Template

The Sentry™ 4.0 HPM template firmware includes demonstration of different IPs for server segments. LTPI, SGPIO, M-PESTI, and eSPI 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 the 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.

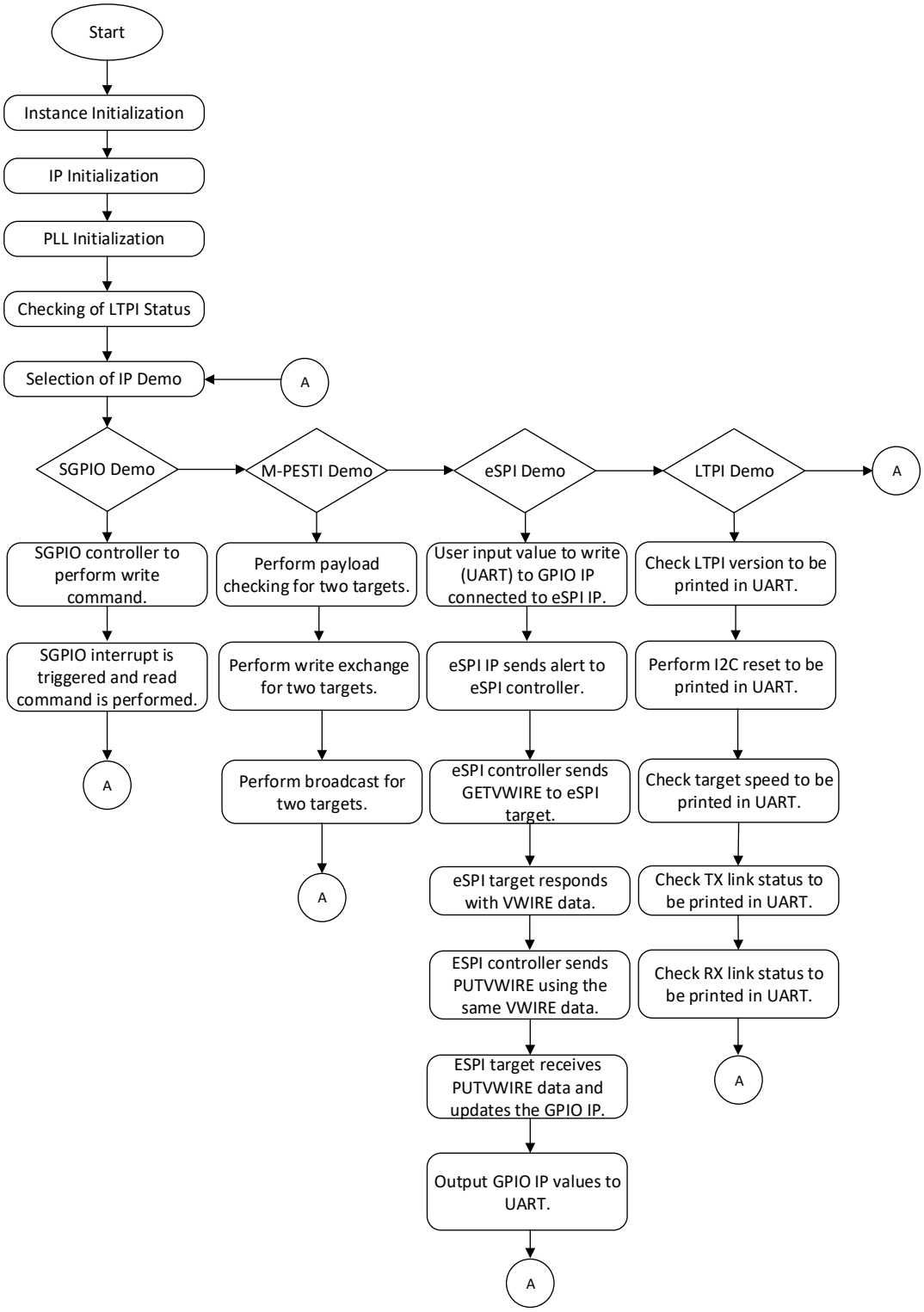


Figure 4.3. Firmware Flow Chart

```
int main(void) {
    uint8_t link_st=0;

    system_init();
    demo_selection();

    while (true) {
        while(link_st==1)
        {
            delayMS(1);
        }
    }

    return 0;
}
```

Figure 4.4. Main Function Code

4.2.1.2. IP Initialization (system_init)

This function initializes the IPs in the HPM design. The initialization of each IP interrupt is also initialized in this function (Figure 4.5).

```
static void system_init(void)
{
    pic_init(CPU0_INST_PICTIMER_START_ADDR);
    jedi_pll_init(&pll_inst0, APB2LMMI0_INST_BASE_ADDR);
    virti_inst0.intrType = 0;
    virti_inst0.intrMethod = 1;
    virti_inst0.intrMethod2 = 0;
    virti_inst0.intrEnable = 1;
    virti_inst0.intrSet =1 ;
    virti_inst0.intrLevel = GPIO1_INST_IRQ;
    virti_inst0.intrAvail = true;
    virti_inst0.instance_name = GPIO0_INST_NAME;
    gpio_init(&virti_inst0, GPIO1_INST_BASE_ADDR, GPIO1_INST_LINES_NUM, GPIO1_INST_GPIO_DIRS); //test swap virto_inst0 from virti_inst0
    gpio_init(&virto_inst0, GPIO0_INST_BASE_ADDR, GPIO0_INST_LINES_NUM, GPIO0_INST_GPIO_DIRS); //test swap virti_inst0 from virto_inst0
    pic_isr_register(GPIO1_INST_IRQ, gpio_callback, (void *)&virti_inst0);
    gpio_init(&mpesti_rst_inst0, GPIO2_INST_BASE_ADDR, GPIO2_INST_LINES_NUM, GPIO2_INST_GPIO_DIRS);
    sgpio_init(&sgpio_inst0, SGPIOC0_INST_BASE_ADDR, 0, 0);
    uart_init(&uart_inst0, UART_INST_BASE_ADDR, UART_INST_SYS_CLK * 1000000, UART_INST_BAUD_RATE, 1, 8);
    iob_init(lscc_uart_putc, lscc_uart_getc, lscc_uart_flush);
    jedi_pll_callback_register(&pll_inst0, jedi_pll_callback_demo, &pll_inst0);
    pic_isr_register(APB2LMMI0_INST_IRQ, jedi_pll_isr, (void *)&pll_inst0);
    jedi_pll_irq_en(&pll_inst0);
    pic_isr_register(SGPIOC0_INST_IRQ, sgpio_callback, (void *)&sgpio_inst0);
    pic_isr_register(MPESTI0_INST_IRQ, MPESTI_SLAVE_ISR, (void *)&mpesti_inst0);
    sgpio_enable_interrupt(&sgpio_inst0, 1);
    sgpio_register_callback(&sgpio_inst0, sgpio_callback, NULL);
    mpesti_mst_init(&mpesti_inst0, MPESTI0_INST_BASE_ADDR, MPESTI0_INST_NUM_PAYLOAD_BYTES );
    gpio_output_write_2(&mpesti_rst_inst0, 0x00000000);
    gpio_output_write_2(&mpesti_rst_inst0, 0xFFFFFFFF);
    mpesti_mst_set_int_enable(&mpesti_inst0, RCVD_GOOD_PAYLOAD | VWIN_UPDATED);
```

Figure 4.5. Instance Initialization Code

4.2.1.3. IP Demonstration Selection (demo_selection)

You can choose a particular demonstration to run (Figure 4.6). Each IP demonstration has its corresponding routine and functionalities.

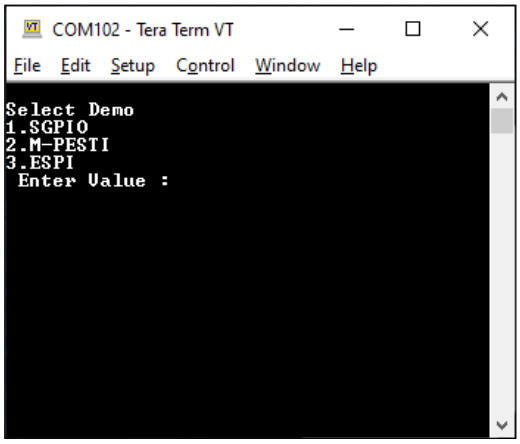


Figure 4.6. UART IP Demonstration Selection

4.3. Timing Constraints

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

4.3.1. Clock Definition

Only the internal input reference clock needs to be manually set at 125 MHz in the post-synthesis timing constraint. Every other clock is inferred by the Lattice Radiant software.
Figure 4.7 shows the clock definition.

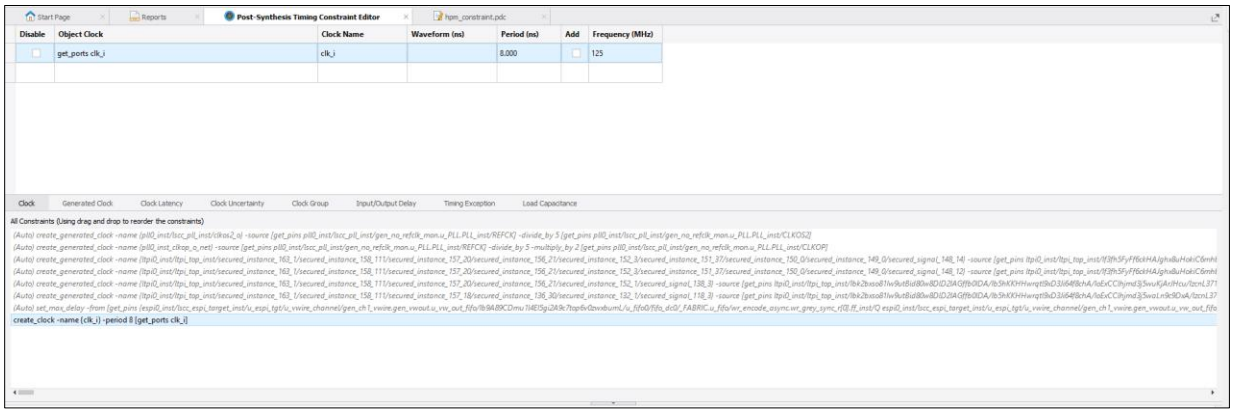


Figure 4.7. Clock Definition

5. SCM and HPM Hardware Bring-up

5.1. SCM and HPM – MachXO5-NX Development Board

5.1.1. Setting Up the Hardware

The SCM and HPM each uses a MachXO5-NX Development Board (Figure 5.1) and a DC-SCM 2.0 MachXO5-NX LTPI Board. The two DC-SCM 2.0 MachXO5-NX LTPI boards are connected from end to end (Figure 5.2). Figure 5.3 shows the complete setup of the SCM and HPM on MachXO5-NX Development Board.

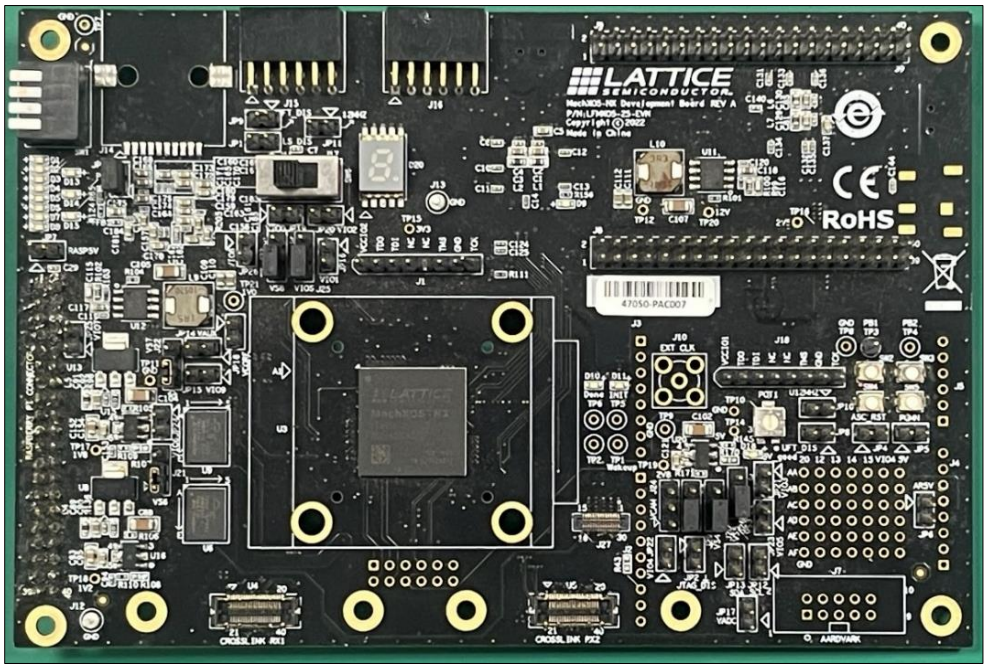


Figure 5.1. MachXO5-NX Development Board

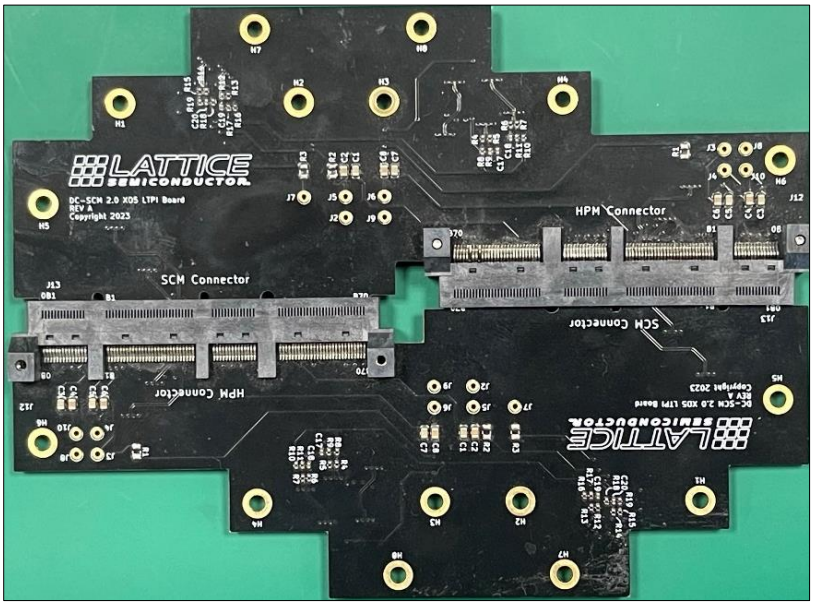


Figure 5.2. Two DC-SCM 2.0 MachXO5-NX LTPI Boards Connected from End to End

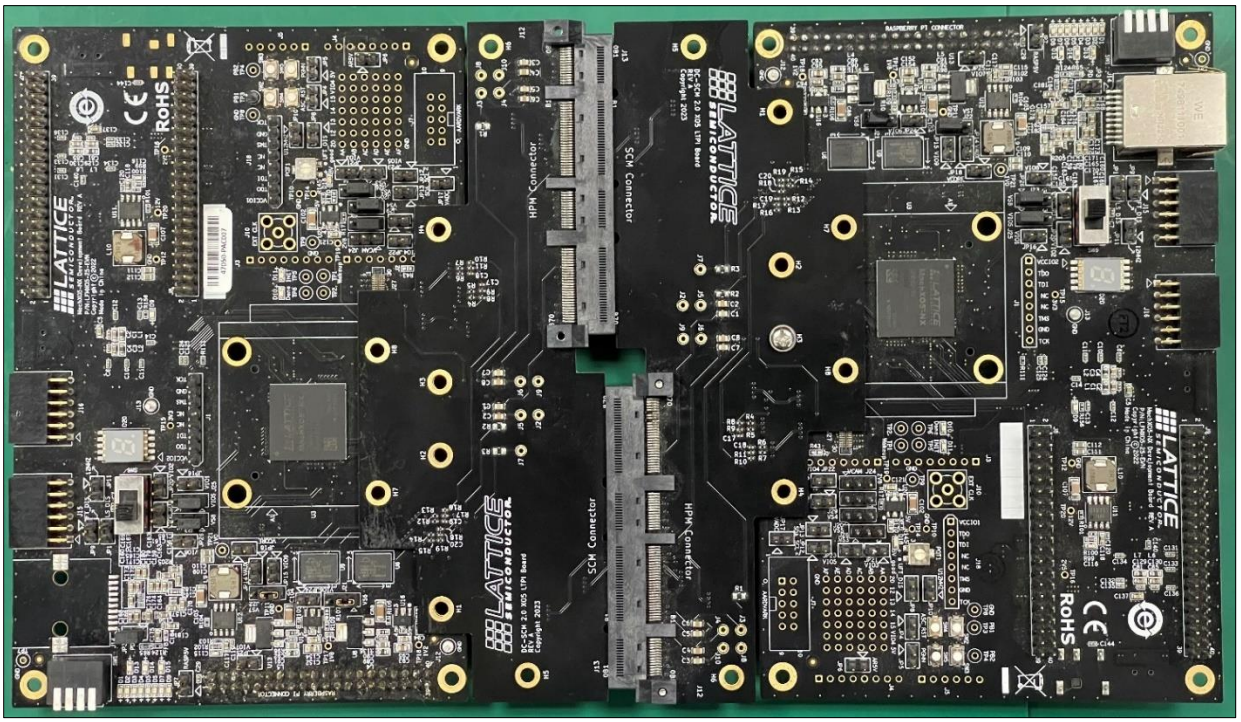


Figure 5.3. SCM-HPM on MachXO5-NX Development Board Complete Setup

5.1.2. Uploading the Configuration File

1. Open SCM_soc in Lattice Diamond Software, as shown in Figure 5.4.

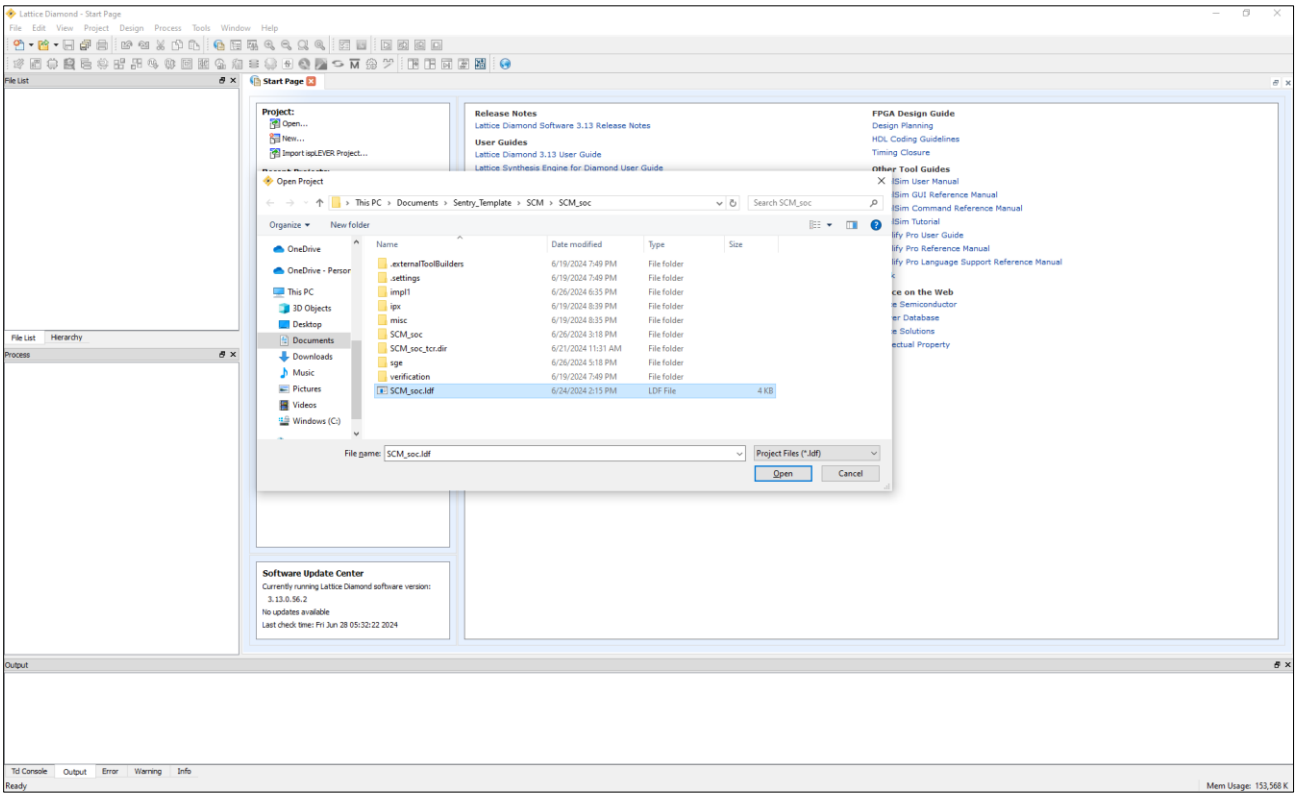


Figure 5.4. Open SCM_soc in Lattice Diamond Software

2. Using Diamond Programmer, upload either the JED or BIT file using either Flash Programming Mode or Static RAM Cell Mode, as shown in Figure 5.5 and Figure 5.6 respectively. Note that the template comes with a pre-exported JED and BIT files. If any changes are made to the design, it is recommended that you export a new JED or BIT file.

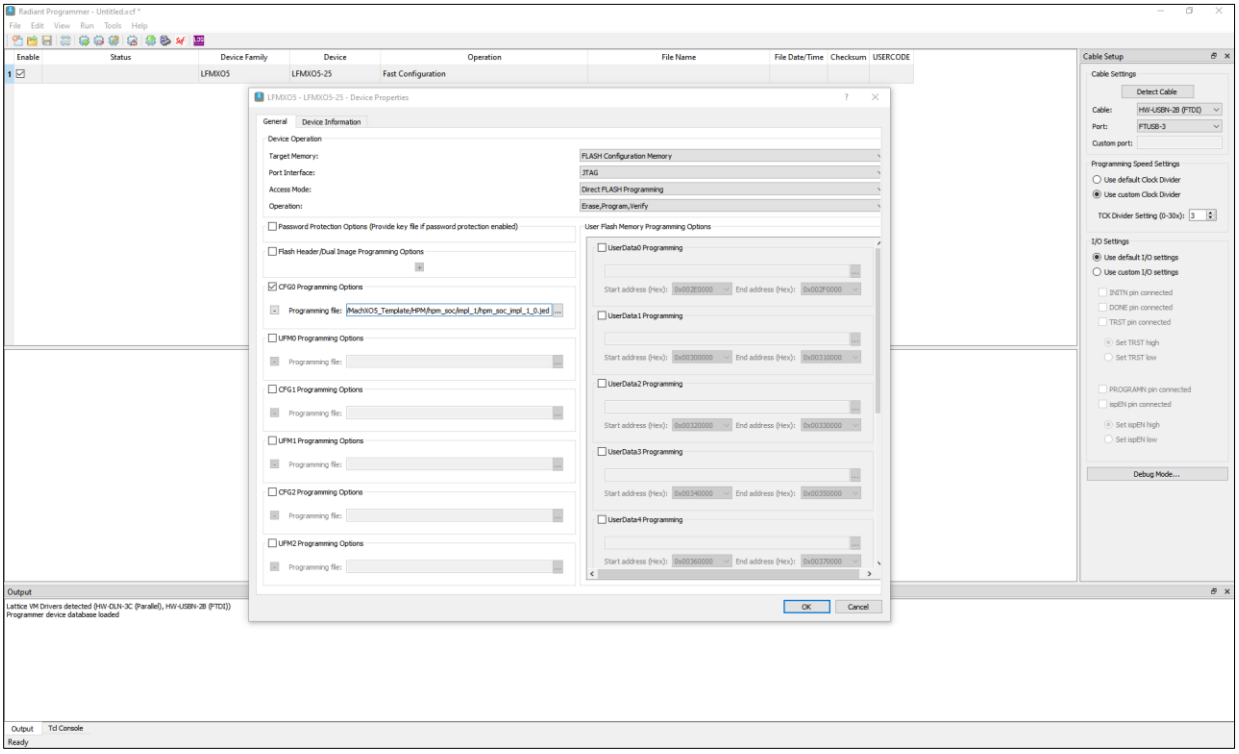


Figure 5.5. JED File Upload Using Flash Programming Mode (SCM)

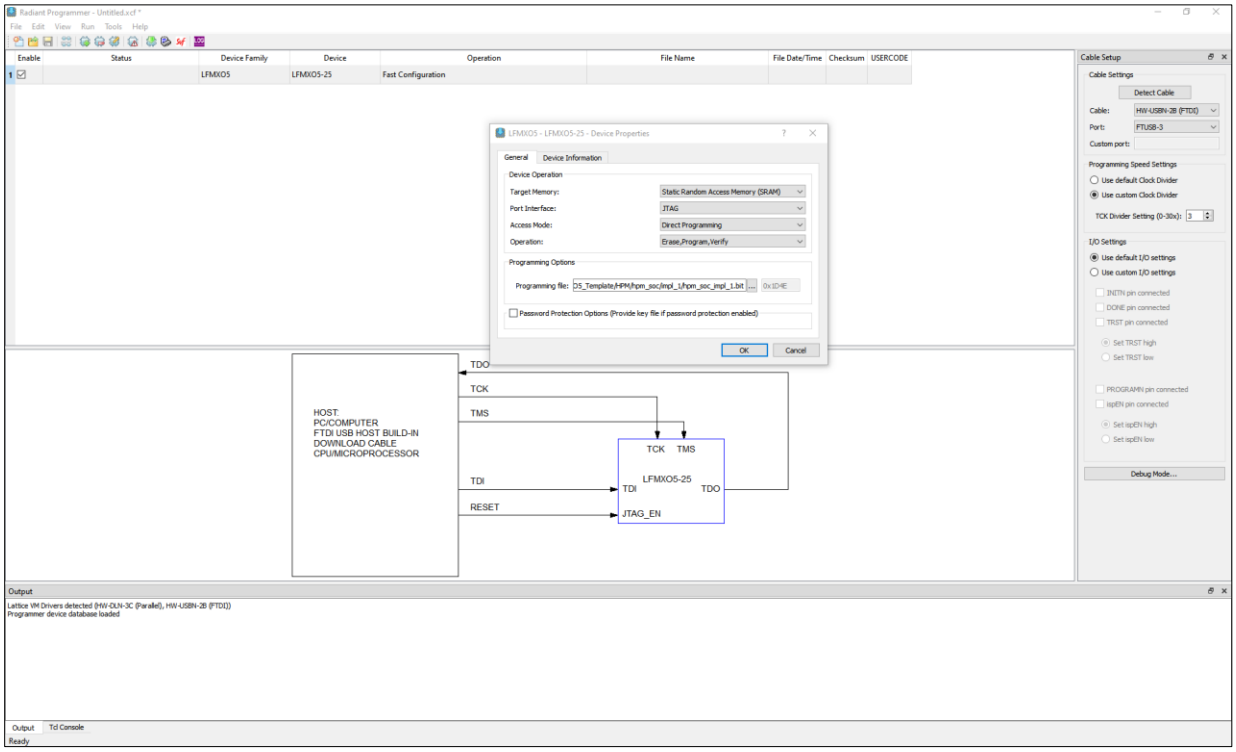


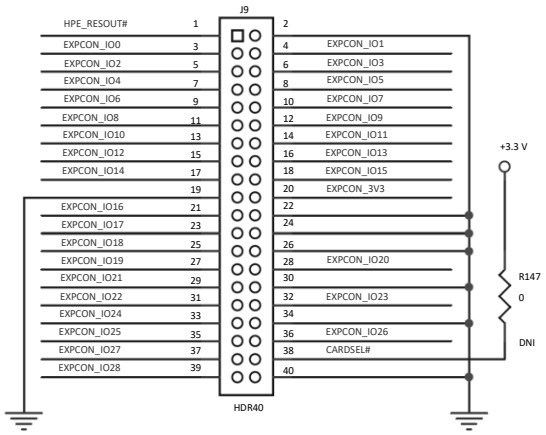
Figure 5.6. BIT File Upload Using Static RAM Cell Mode (SCM)

5.2. eSPI Controller Using Total Phase Promira Serial Platform, Optional for eSPI Demonstration

This demonstration uses the Total Phase Promira Serial platform with I2C or 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, as 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 the eSPI Controller to the HPM hardware through the Versa header (J9), as highlighted in Figure 5.7.



Pinouts	Pin#	J9		Pin#	Pinouts	
	1	HPE_RESET#	GND	2		
	D3	3	EXPCON_IO0	EXPCON_IO1	4	E4	mpesi_initiator[24]
espi0_inst_espi_cs_n_i_port	C3	5	EXPCON_IO2	EXPCON_IO3	6	C2	mpesi_initiator[25]
espi0_inst_espi_rst_n_i_port	A4	7	EXPCON_IO4	EXPCON_IO5	8	E5	mpesi_initiator[26]
espi0_inst_espi_alert_n_o_port	F6	9	EXPCON_IO6	EXPCON_IO7	10	C5	mpesi_initiator[27]
espi0_inst_espi_data_io_portbus[0]	B2	11	EXPCON_IO8	EXPCON_IO9	12	A2	mpesi_initiator[28]
espi0_inst_espi_data_io_portbus[1]	B3	13	EXPCON_IO10	EXPCON_IO11	14	A3	mpesi_initiator[29]
espi0_inst_espi_data_io_portbus[2]	B4	15	EXPCON_IO12	EXPCON_IO13	16	D5	mpesi_initiator[30]
espi0_inst_espi_data_io_portbus[3]	A5	17	EXPCON_IO14	EXPCON_IO15	18	B5	mpesi_initiator[31]
		19	GND	EXPCON_3V3	20		
ltpi0_inst_i2c_sda_io_portbus[0]	A6	21	EXPCON_IO16	GND	22		
ltpi0_inst_i2c_scl_io_portbus[0]	B6	23	EXPCON_IO17	GND	24		
ltpi0_inst_i2c_sda_io_portbus[1]	A7	25	EXPCON_IO18	GND	26		
ltpi0_inst_i2c_scl_io_portbus[1]	A8	27	EXPCON_IO19	EXPCON_IO20	28	C6	mpesti0_inst_s_mpesti_io_portbus[0]
ltpi0_inst_i2c_sda_io_portbus[2]	B8	29	EXPCON_IO21	GND	30		
ltpi0_inst_i2c_scl_io_portbus[2]	B9	31	EXPCON_IO22	EXPCON_IO23	32	A9	uart0_inst_txd_o_port
	D6	33	EXPCON_IO24	GND	34		
	C8	35	EXPCON_IO25	EXPCON_IO26	36	E7	uart0_inst_rxd_i_port
ltpi0_inst_uart_rx_i_portbus[0]	E8	37	EXPCON_IO27	CARDSEL#	38		
ltpi0_inst_uart_tx_o_portbus[0]	E6	39	EXPCON_IO28	GND	40		

Figure 5.7. 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 (Figure 5.8). This is the main file for running the eSPI transactions.

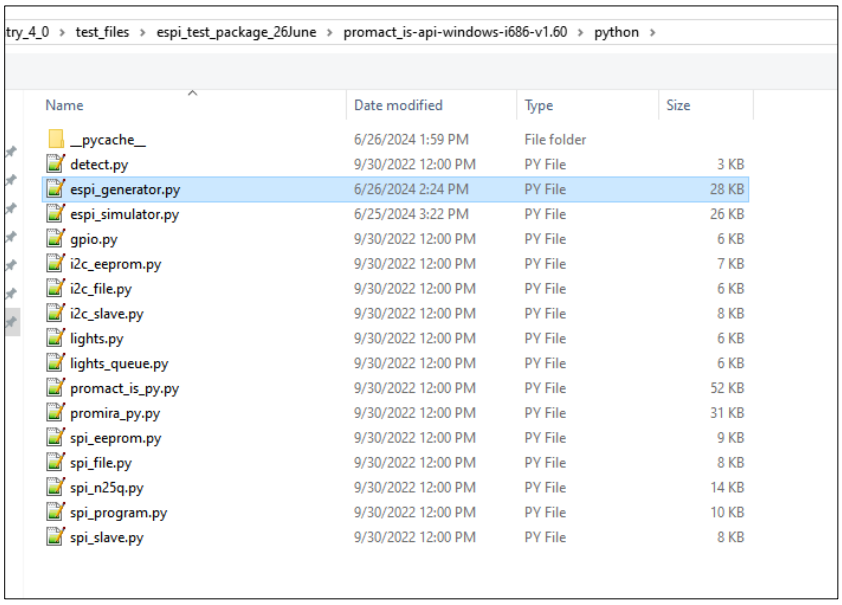


Figure 5.8. API for eSPI Transactions

Note: The API code includes the Promira devices detection, as shown in Figure 5.9. So, there is no need to manually input the IP address.

```
#Detect Promira Devices
print("Detecting the Promira platforms...")

# Find all the attached devices
(num, ips, unique_ids, statuses) = pm_find_devices_ext(16, 16, 16)

if num > 0:
    print("%d device(s) found:" % num)

    # Print the information on each device
    for i in range(num):
        ip        = ips[i]
        unique_id = unique_ids[i]
        status    = statuses[i]

        # Determine if the device is in-use
        if status & PM_DEVICE_NOT_FREE:
            inuse = "(in-use)"
        else:
            inuse = "(avail)"

        # Display device ip address, in-use status, and serial number
        ipstr = "%u.%u.%u.%u" \
            % (ip & 0xff, ip >> 8 & 0xff, ip >> 16 & 0xff, ip >> 24 & 0xff)
        print("    ip = %s   %s  (%04d-%06d)"
            % (ipstr, inuse, unique_id // 1000000, unique_id % 1000000))

else:
    print("No devices found.")
    exit(1)



#Global Variables
my_promira_port = ipstr #IP address example "10.1.66.85"


# Open the device
simulator = EspiSimulator(my_promira_port)
simulator.espi_config_mode(1)

```

Figure 5.9. Promira Port IP Address Detection

5. Click **Run** on IDLE Shell and select **Run Module** (Figure 5.10).

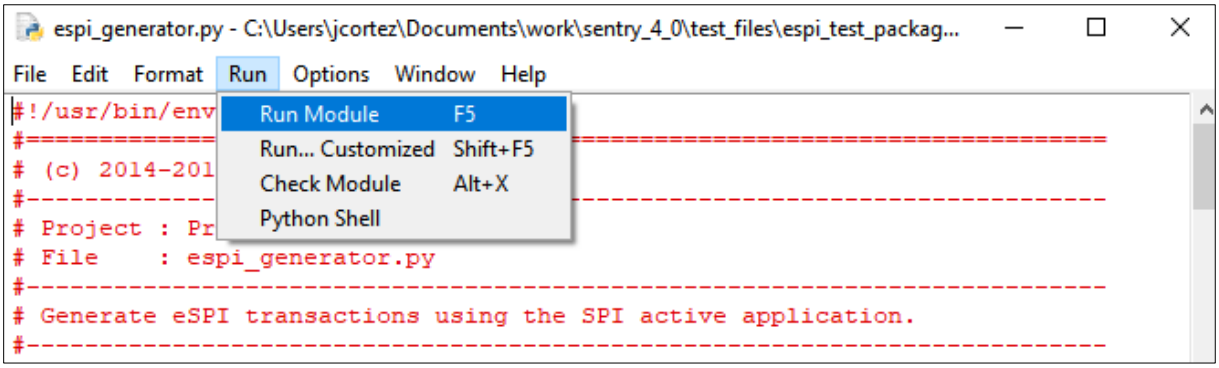


Figure 5.10. Run in IDLE Shell

6. The IDLE Shell console should show prints (Figure 5.11).

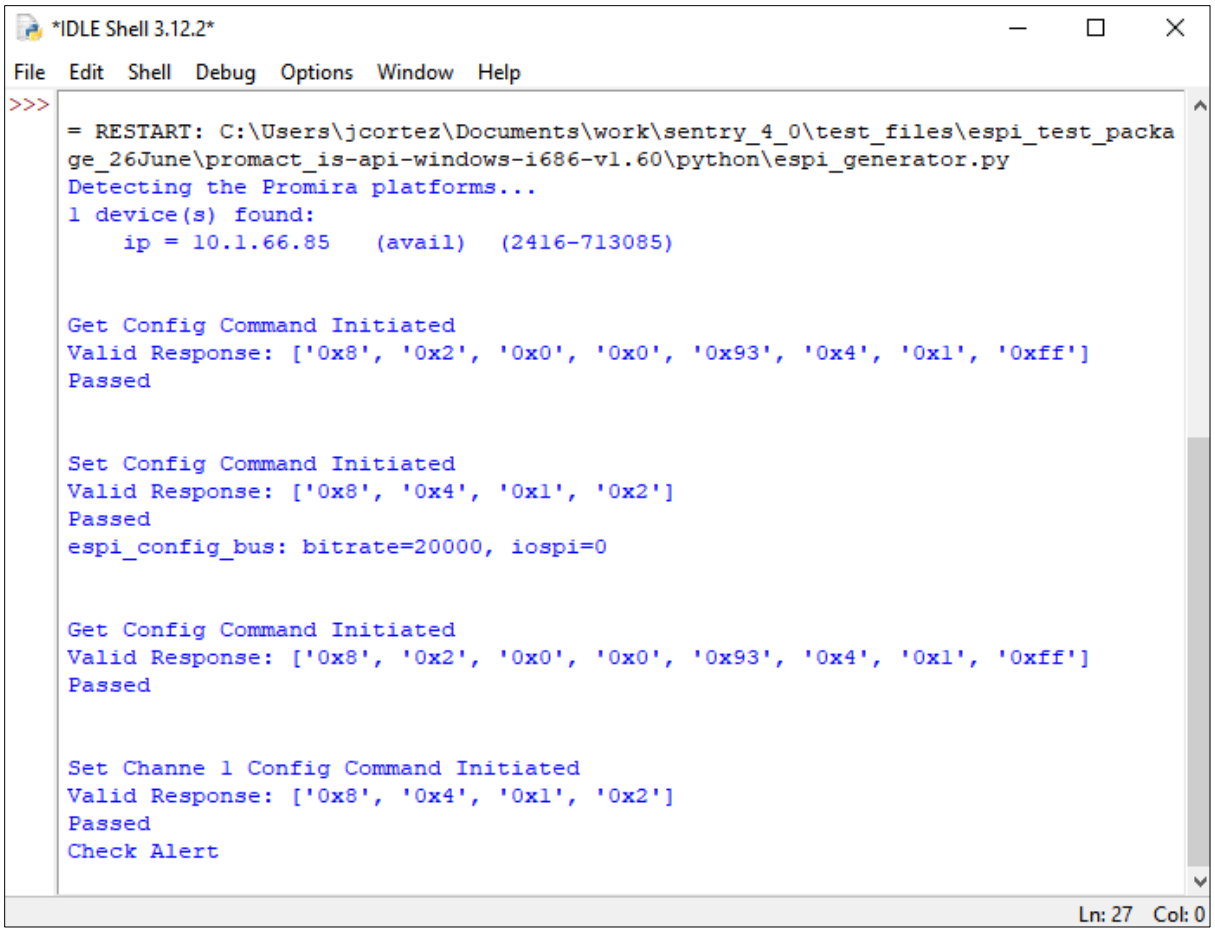


Figure 5.11. IDLE Shell Output

5.3. Total Phase Aardvark for I2C, Optional for LTPI I2C Aggregation

Device product name: Aardvark I2C/SPI Host Adapter (Figure 5.12)
Target Bus Interface: I2C Controller/Target
Bit Rate: I2C Controller: 1 kHz–800 kHz

Figure 5.12. 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) for the validation of I2C communication.

1. Follow I2C pin connections from Aardvark to the SCM board and the HPM board described below:
    a. Connect GND and I2C pins of one Aardvark device to the SCM board (MachXO5-NX development 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 B6, A6) is used in the SCM side (Figure 5.13 and Figure 5.14).

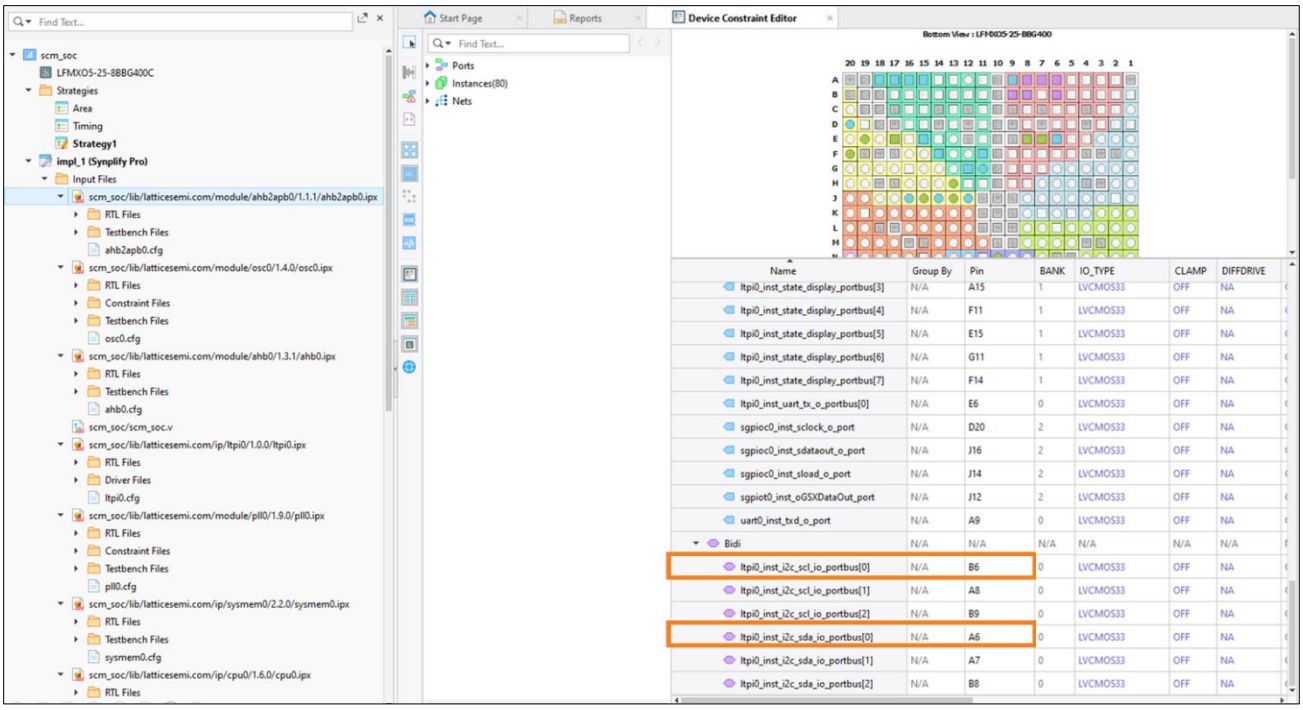


Figure 5.13. SCM Project Design Spreadsheet

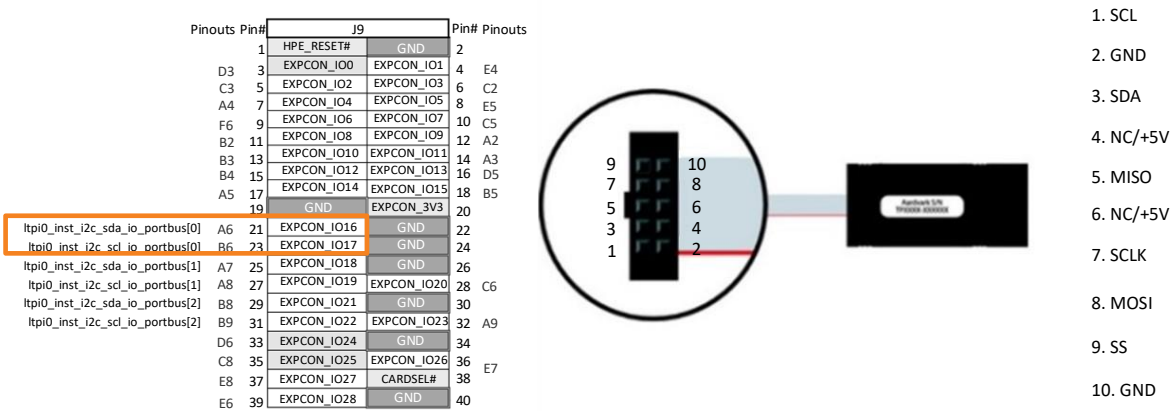


Figure 5.14. MachX05-NX Development Board – Debug Pinout and Aardvark Pinouts

b. Connect GND and I2C pins of the other Aardvark device to the HPM board.

2. Use I2C bus 0 (pins B6, A6) for the HPM side (Figure 5.15 and Figure 5.16).

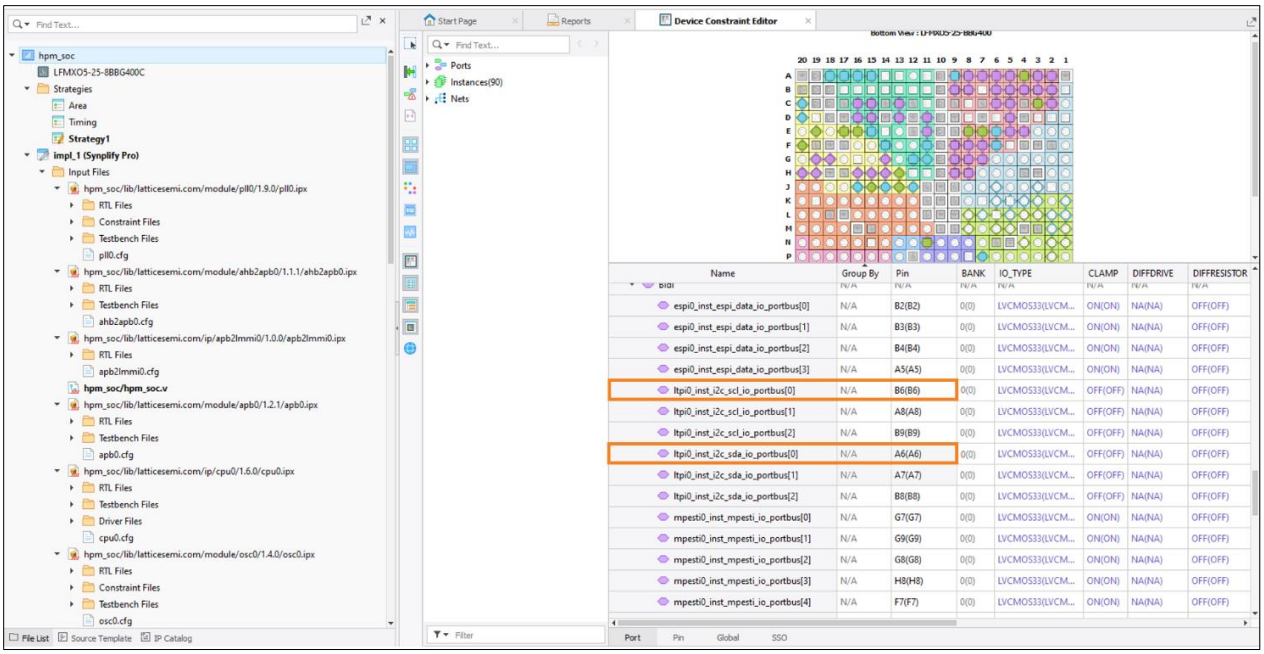
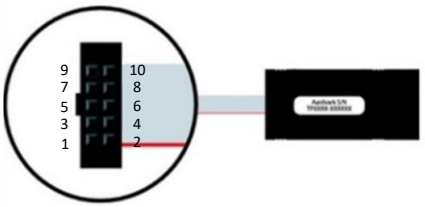


Figure 5.15. HPM Project Design Spreadsheet

Pinouts	Pin#	J9		Pin#	Pinouts
	1	HPE_RESET#	GND	2	
D3	3	EXPCON_IO0	EXPCON_IO1	4	E4
C3	5	EXPCON_IO2	EXPCON_IO3	6	C2
A4	7	EXPCON_IO4	EXPCON_IO5	8	E5
F6	9	EXPCON_IO6	EXPCON_IO7	10	C5
B2	11	EXPCON_IO8	EXPCON_IO9	12	A2
B3	13	EXPCON_IO10	EXPCON_IO11	14	A3
B4	15	EXPCON_IO12	EXPCON_IO13	16	D5
A5	17	EXPCON_IO14	EXPCON_IO15	18	B5
	19	GND	EXPCON_3V3	20	
ltpi0_inst_i2c_sda_io_portbus[0]	A6	21	EXPCON_IO16	GND	22	
ltpi0_inst_i2c_scl_io_portbus[0]	B6	23	EXPCON_IO17	GND	24	
ltpi0_inst_i2c_sda_io_portbus[1]	A7	25	EXPCON_IO18	GND	26	
ltpi0_inst_i2c_scl_io_portbus[1]	A8	27	EXPCON_IO19	EXPCON_IO20	28	C6
ltpi0_inst_i2c_sda_io_portbus[2]	B8	29	EXPCON_IO21	GND	30	
ltpi0_inst_i2c_scl_io_portbus[2]	B9	31	EXPCON_IO22	EXPCON_IO23	32	A9
	D6	33	EXPCON_IO24	GND	34	
	C8	35	EXPCON_IO25	EXPCON_IO26	36	E7
	E8	37	EXPCON_IO27	CARDSEL#	38	
	E6	39	EXPCON_IO28	GND	40	



1. SCL
2. GND
3. SDA
4. NC/+5V
5. MISO
6. NC/+5V
7. SCLK
8. MOSI
9. SS
10. GND

Figure 5.16. MachX05-NX Development Board 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 the device ID at the back of the aardvark device (Figure 5.17). Then, connect the corresponding aardvark device by selecting **Adapter** > **Connect** > **Configure Adapter**.

Figure 5.17. Aardvark device ID

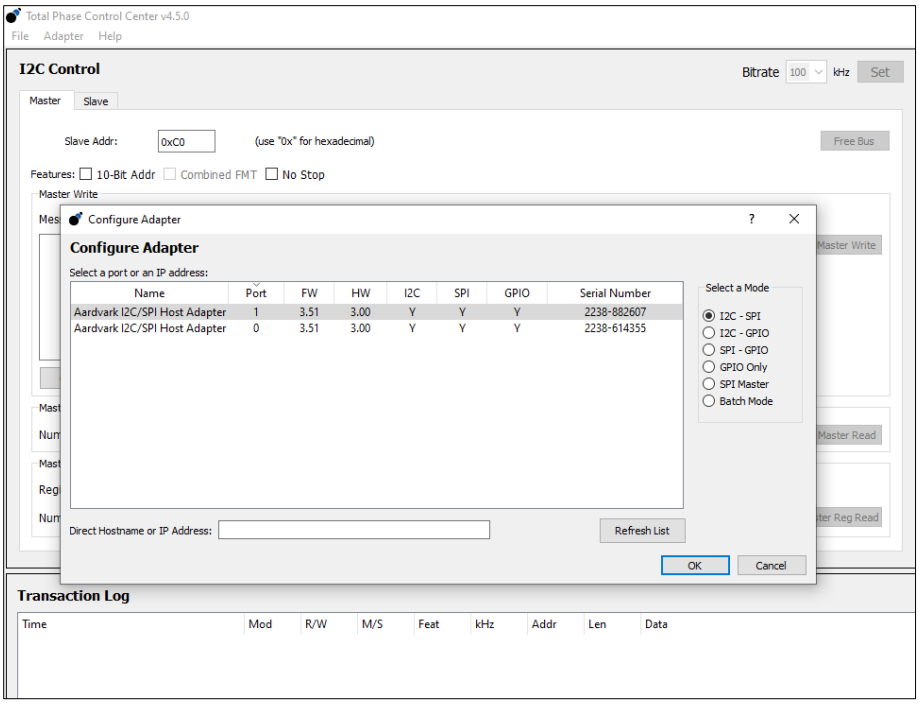


Figure 5.18. Control Center Adapter Configuration

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, two MachXO5 Development Boards are utilized. Both sides are connected with a USB cable as a power source, and a connection with UART terminal (Figure 6.1).

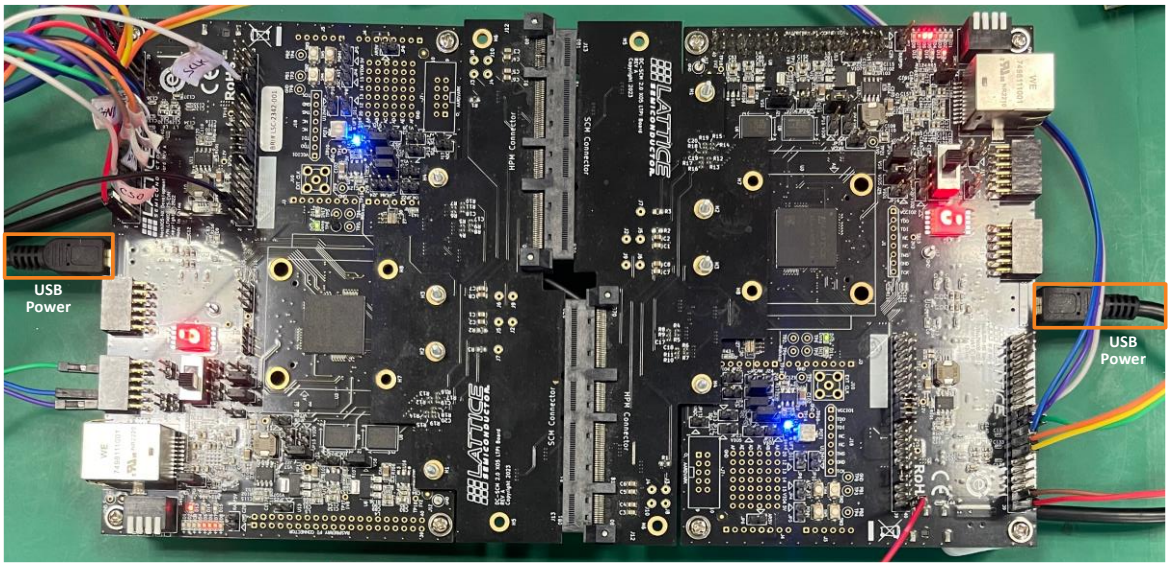


Figure 6.1. MachXO5-NX Demonstration Board

6.1.1.1. LTPI Hardware 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):
   There are two indicators you can check to see if LTPI connection is established.
   - LTPI low latency GPIO LED display — LED4(D5), LED5(D6), and LED6(D7) are lit.
   - The 7-segment LED display.

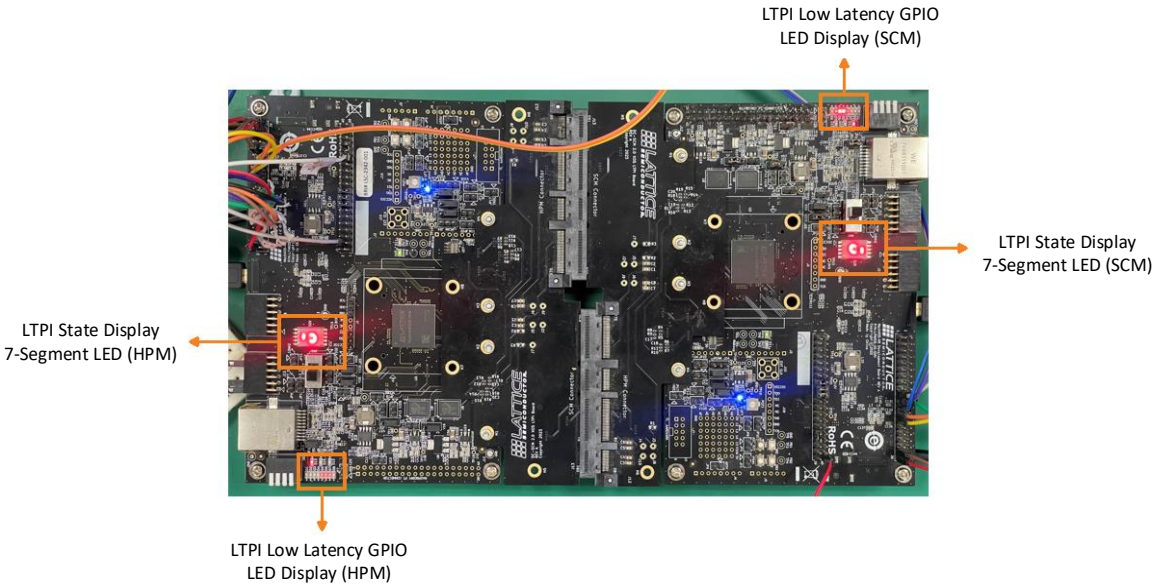


Figure 6.2. LED Indicator for LTPI Connection

Another way to check LTPI connection is through LED0(D1), LED1(D2), LED2(D3), and LED3(D4), which are connected to the DIP switches (SW1) on the opposite side. The matching LEDs on the opposite side of the board turn on and go to high when you press down the DIP switches (SW1) on each side.

6.1.1.2. I2C Communication on LTPI Using Aardvark

1. 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 Target.
2. Set bit rate at 100 kHz and set the same 7-bit target address on each side (Figure 6.3 and Figure 6.4).

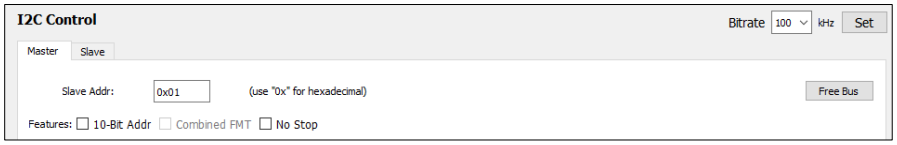


Figure 6.3. Aardvark-Controller Setting

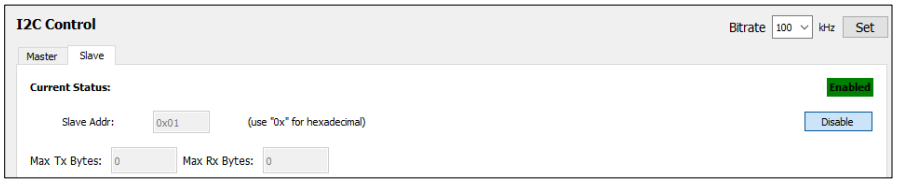


Figure 6.4. Aardvark-Target Setting

3. Set a response message on the target side for a read transaction (Figure 6.5).

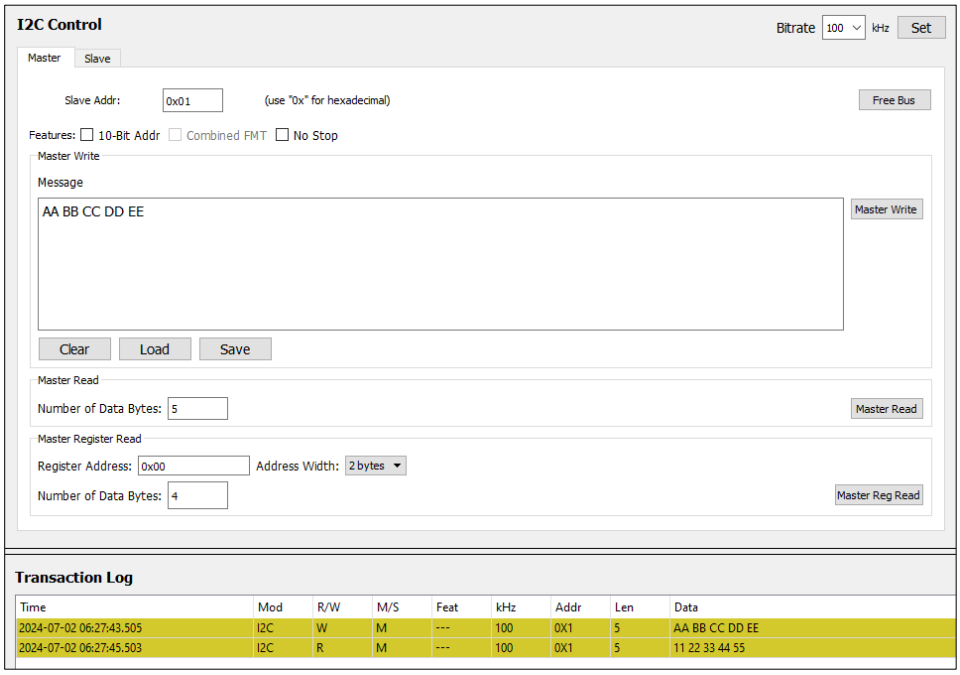


Figure 6.5. Aardvark Target Response

4. Do a simple I2C transaction. Input I2C data to the **Message** field and click **Master Write**. Check I2C communication by clicking **Master Read**. The set response from the target should be visible on the Transaction Log (Figure 6.6 and Figure 6.7).

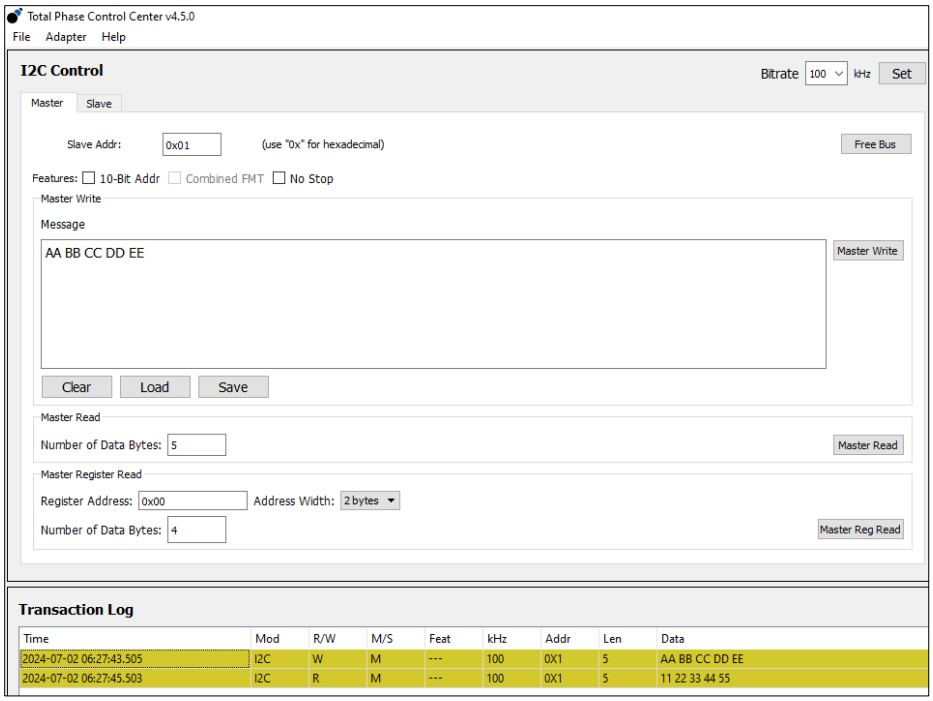


Figure 6.6. Aardvark Controller

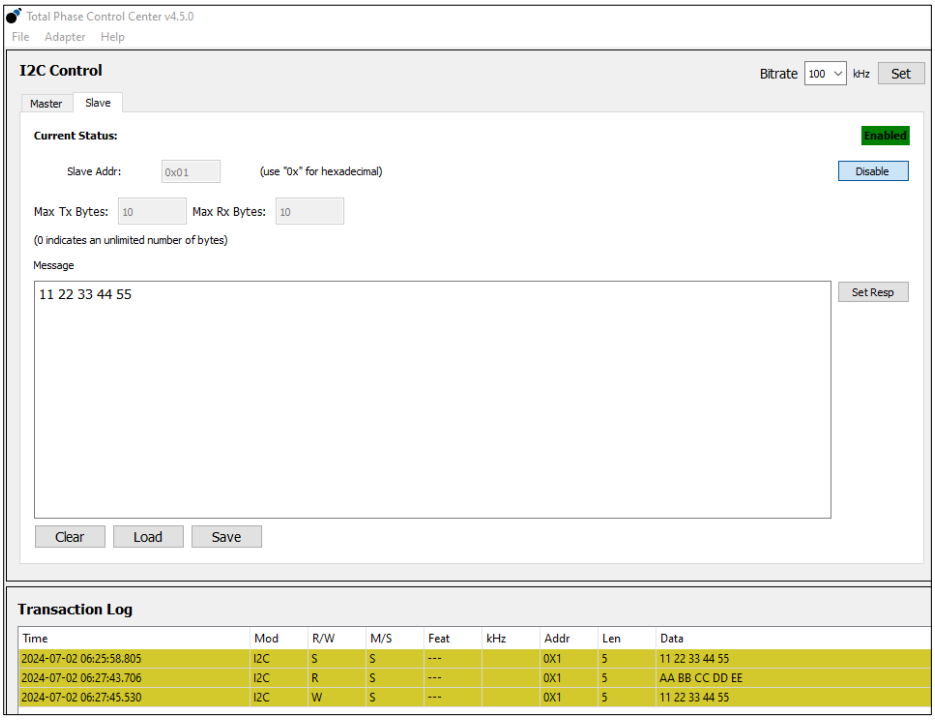


Figure 6.7. Aardvark Target

6.1.2. eSPI Demonstration

Figure 6.8 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.

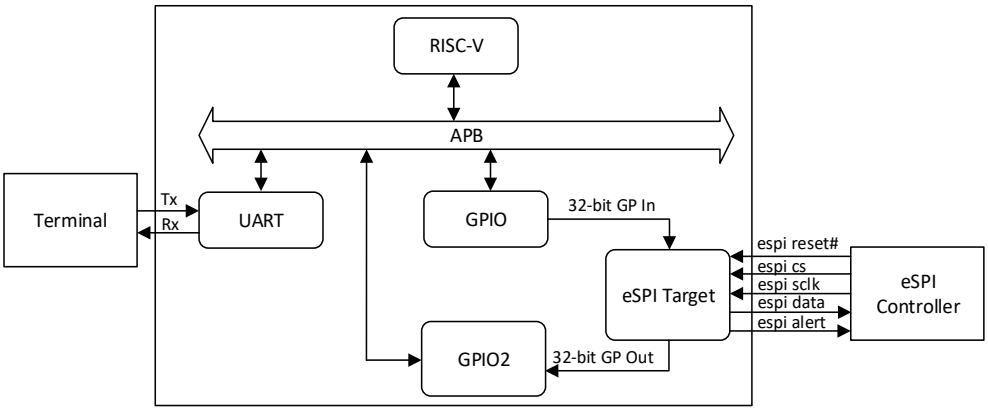


Figure 6.8. eSPI Demonstration Diagram

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 Core (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 Virtual Wire Channel Enable command to enable the bit 0 of this channel is required. Figure 6.9 shows pseudocode for this transaction.

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

Figure 6.9. 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 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 6.10).

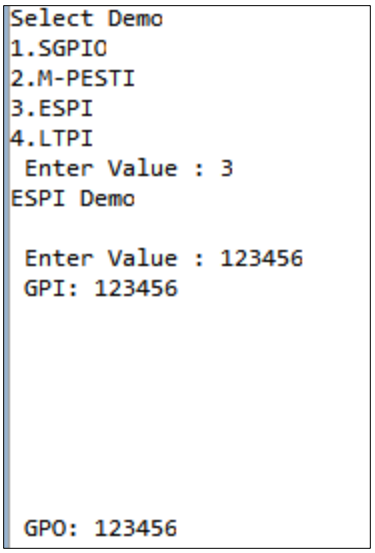


Figure 6.10. Terminal Print Output

6.1.3. M-PESTI Demonstration

As simplified in Figure 6.11, The M-PESTI demonstration involves two key components. The M-PESTI Initiator resides in HPM SoC design and is housed in the MachXO5-NX Development Board, while the M-PESTI targets are located on the MachXO3D device on the Sentry 4.0 board.

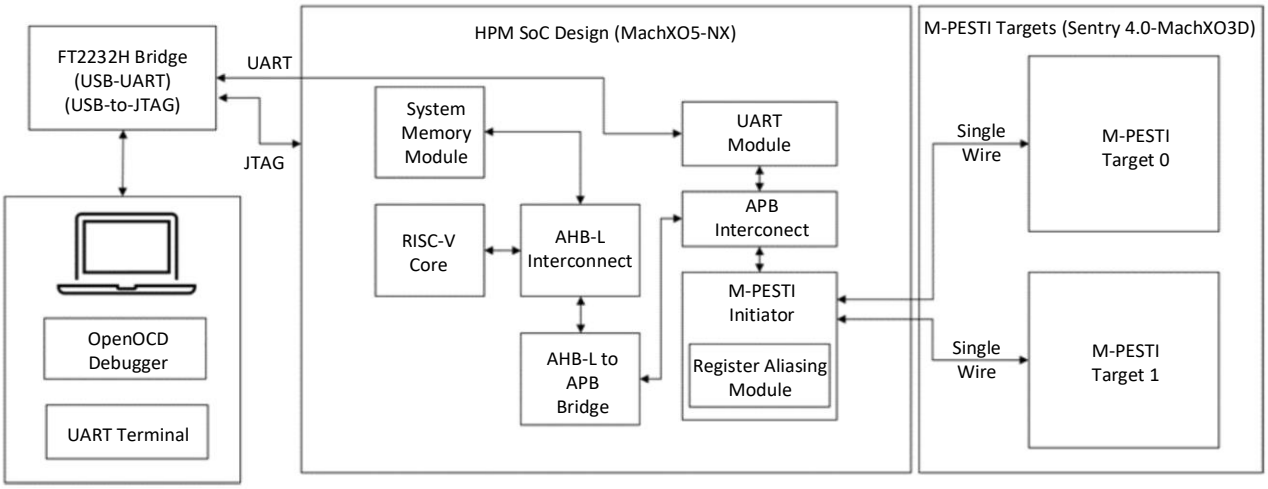


Figure 6.11. M-PESTI Demonstration Block Diagram

To run the demonstration, perform the following steps:

1. Using jumper wire, connect pin E6 of RPI connector of the Sentry 4.0 board to pin C20 of J9 versa header connector of the MachXO5-NX Development Board, as shown in Figure 6.12. This pin serves as the M-PESTI reset signal. Also, connect common ground for the MachXO5-NX Board and the Sentry 4.0 board.

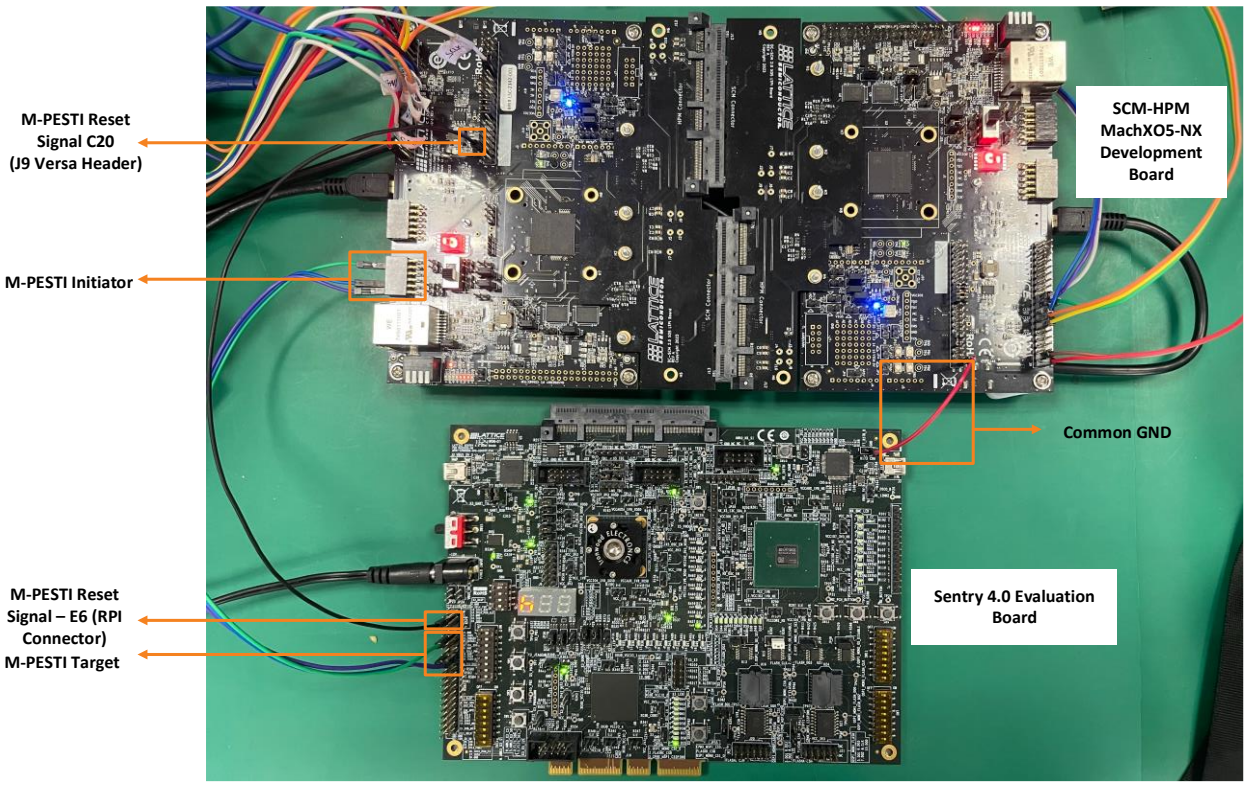


Figure 6.12. M-PESTI Demonstration Jumpers

2. Connect the M-PESTI Initiator pins on the MachXO5-NX device to the M-PESTI target of MachXO3D device on the Sentry 4.0 board, as shown in Figure 6.12. See Figure 6.13 for pin mapping information.

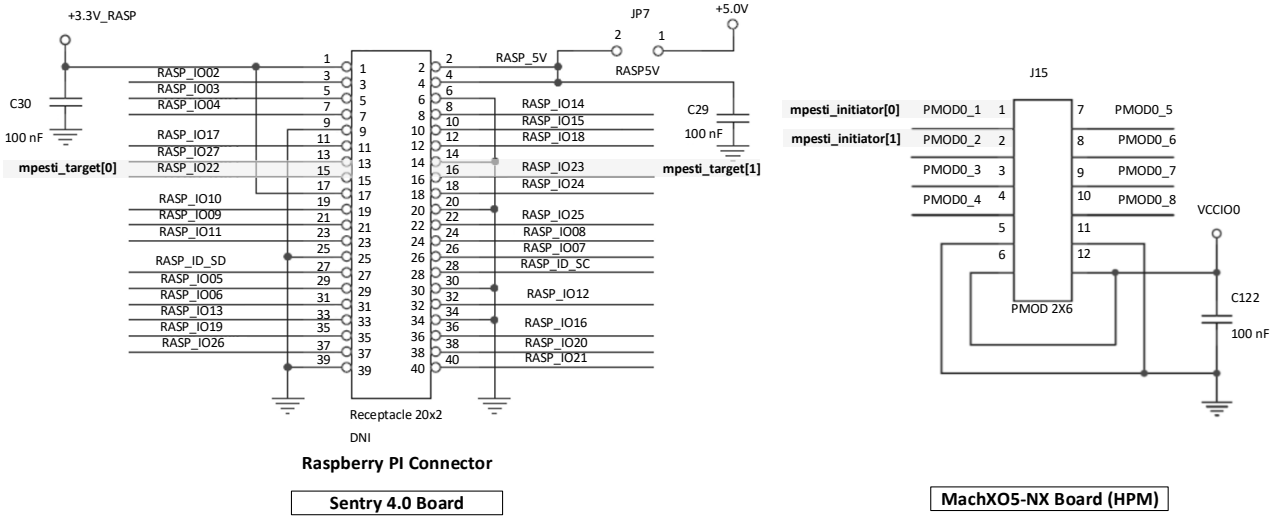


Figure 6.13. M-PESTI Initiator to M-PESTI Target connection

3. After power on, the demonstration menu screen is printed on the UART terminal (Figure 6.14).

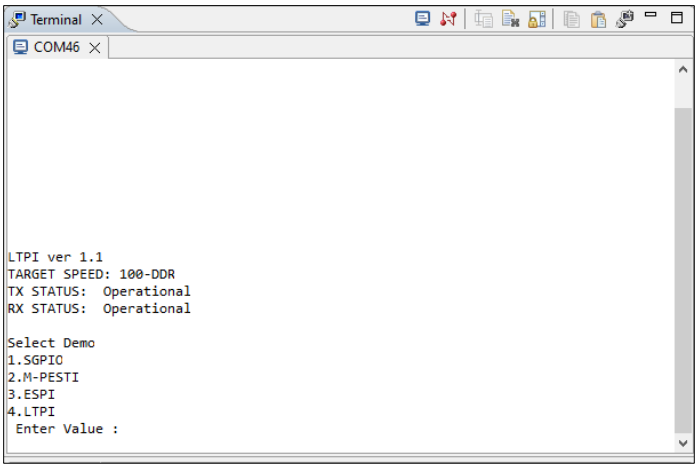


Figure 6.14. Demonstration Menu Screen

4. Wait until LED D55 blinks (Figure 6.15). Then, enter 2 in the Menu to select the M-PESTI demonstration and hit Enter.

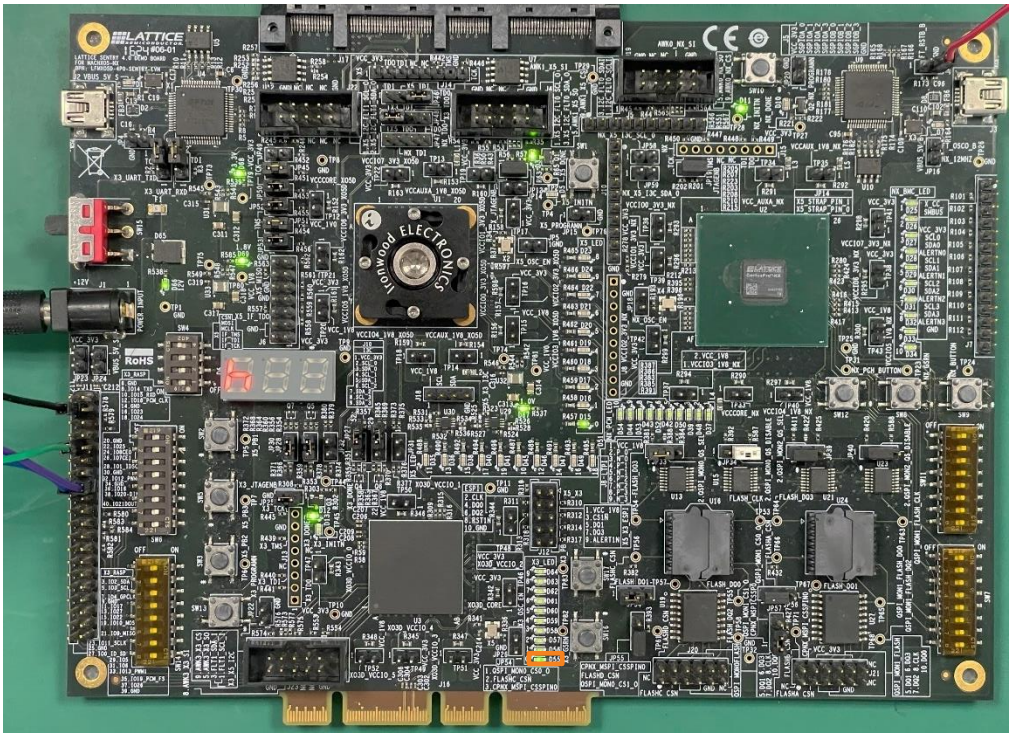


Figure 6.15. D55 LED Blink

5. After selecting the M-PESTI demonstration, the following can be observed on the UART terminal (Figure 6.16):
   a. The target T0 and target T1 discovery payloads are printed on the UART terminal. These values originate from the MachXO3D 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.

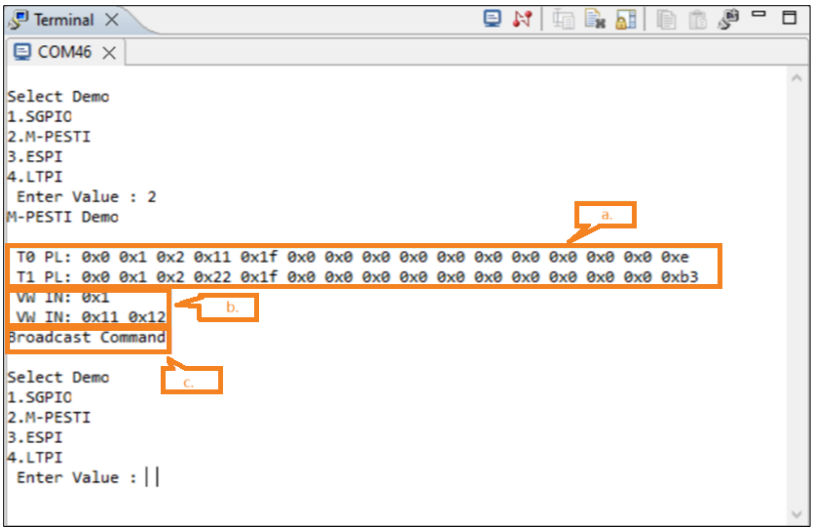


Figure 6.16. M-PESTI Demonstration Terminal

6.1.4. SGPIO Demonstration for HPM

Figure 6.17 is a diagram for the HPM SGPIO demonstration. It shows the UART terminal and the HPM template with only SGPIO demonstration-related components.

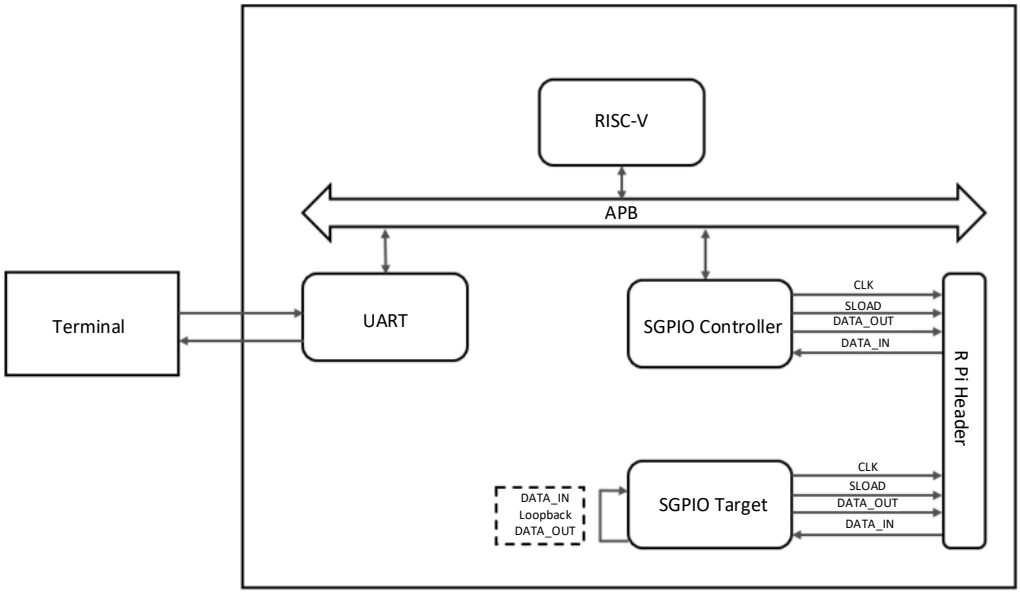


Figure 6.17. HPM SGPIO Demonstration Diagram

1. Connect the jumpers for RPi Header, as shown in Figure 6.18.

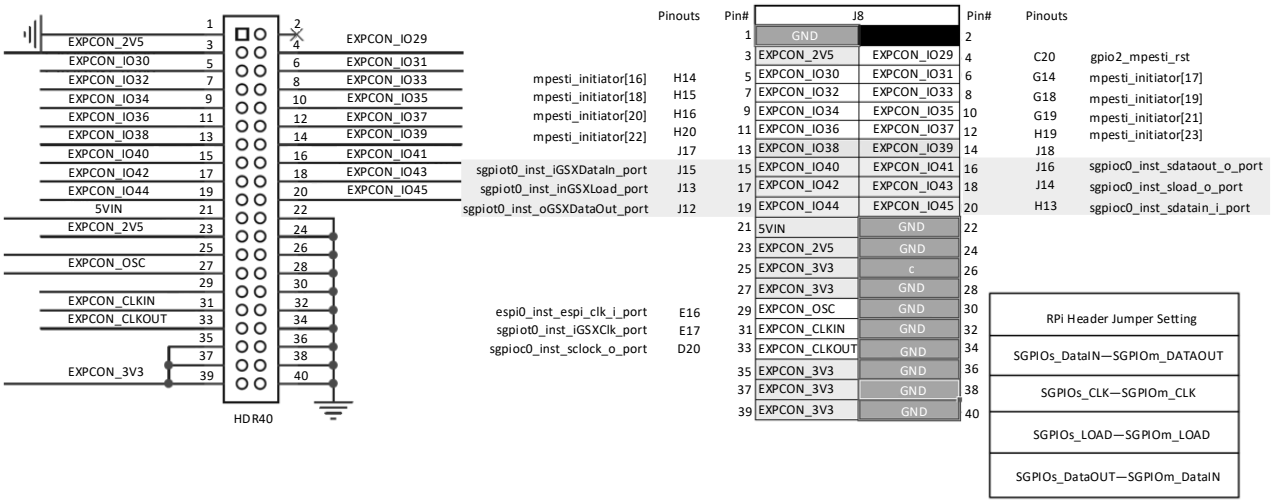


Figure 6.18. Versa Header(J8) Jumper Settings

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 6.19.

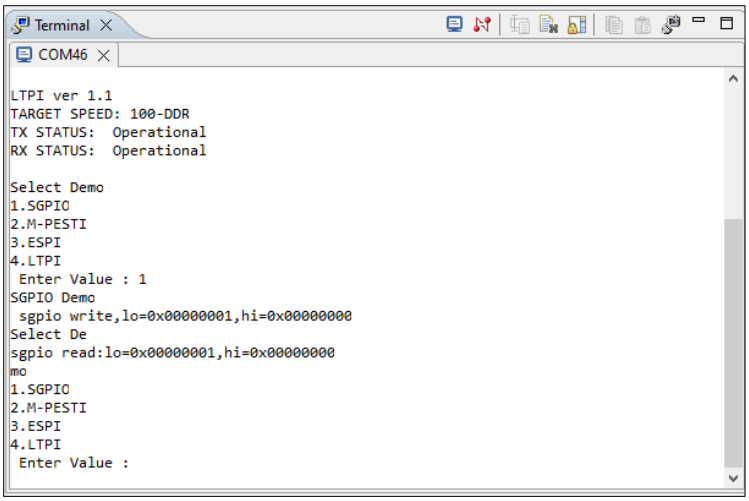


Figure 6.19. UART SGPIO Demonstration Selection

4. 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 6.20).

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

Figure 6.20. SGPIO Demonstration Code

5. 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 6.21 and Figure 6.22).

```
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 6.21. 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 6.22. SGPIO Interrupt Callback Function

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

6.1.5. SGPIO Demonstration for SCM

Figure 6.23 shows the diagram for SCM SGPIO demonstration. It shows the UART terminal and the SCM template with only SGPIO demonstration-related components.

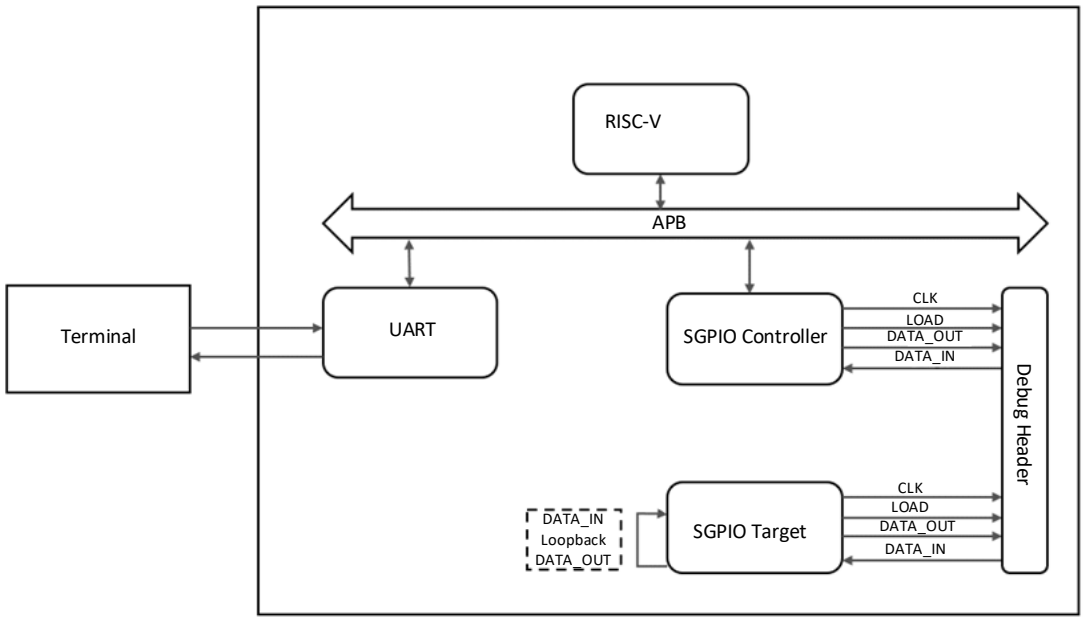


Figure 6.23. SCM SGPIO Demonstration Diagram

1. Connect the jumpers for RPi Header based on Figure 6.24.

		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			DEBUG_0 Header Jumper Settings
LTPI_I2C0_SCL	13	D8	L16	14	LTPI_I2C0_SDA		SGPIOs_DataIN---SGPIOm_DataOUT
LTPI_I2C1_SCL	15	E9	M15	16	LTPI_I2C1_SDA		SGPIOs_CLK---SGPIOm_CLK
LTPI_I2C2_SCL	17	D6	N14	18	LTPI_I2C2_SDA		SGPIOs_LOAD---SGPIOm_LOAD
LTPI_TXD	19	E7	M16	20	LTPI_RXD		SGPIOs_DataOUT---SGPIOm_DataIN
	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 6.24. 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 6.25.

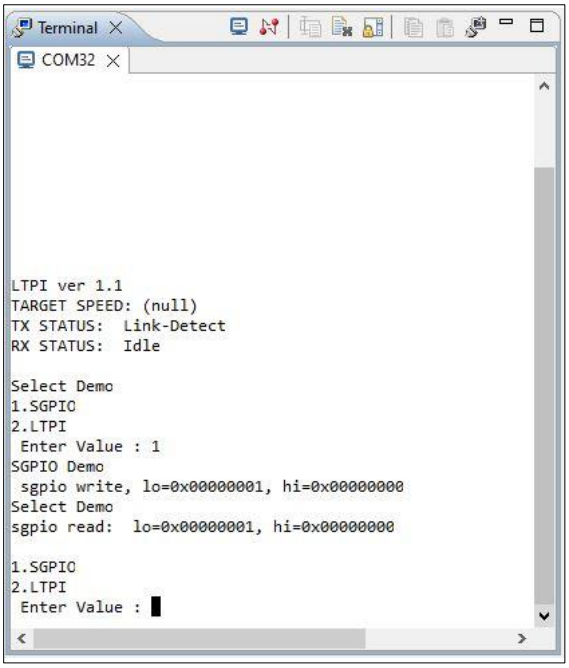


Figure 6.25. UART SGPIO Demonstration Selection

4. The firmware writes 0x00000001 to in_lo and 0x00000000 to in_hi from SGPIO Controller. Figure 6.26 shows the function for the SGPIO demonstration.

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

Figure 6.26. SGPIO Demonstration Code

5. 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 6.27 and Figure 6.28).

```
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 6.27. 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 }
```

Figure 6.28. 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 or 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 the 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 and 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 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 port and 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)	1556	2	Using RISCV-SM, debug mode enabled.
System Memory (sysmem0)	53	16	Approximately 8 KB memory size
eSPI	756	2	For 64-bit virtual wire-only configuration
LTPI	3437	2	The utilization of LTPI varies based on the target configuration
M-PESTI	2956	1	The utilization is based on 4 Target configurations.
SGPIO Controller	131	0	34-bit width
SGPIO Target	35	0	34-bit width
GPIO0	285	0	For debug and demo purposes. Used as a virtual wire for LTPI.
GPIO1	352	0	For debug and demo purposes. Used as a virtual wire for eSPI.
GPIO2	286	0	For debug and demo purposes. Used as a virtual wire for eSPI.
UART	196	0	For Debug and demo purposes
CSR0	—	—	For LTPI CSR/Dynamic PLL Access
APB2LMMI	64	0	For Server LTPI
APB	122	0	—
AHBL2 APB Bridge	263	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

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

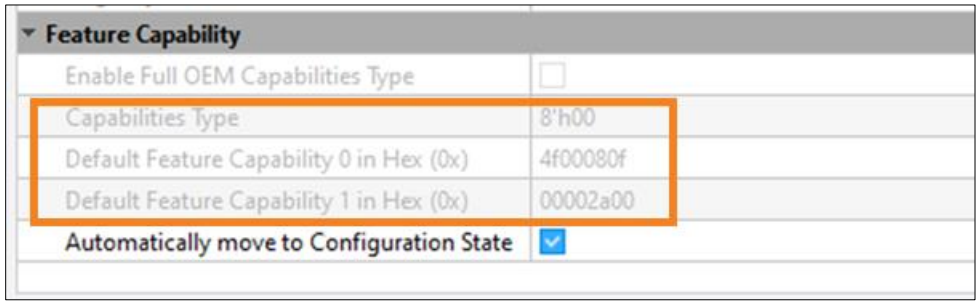


Figure 9.1. Capabilities Match

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

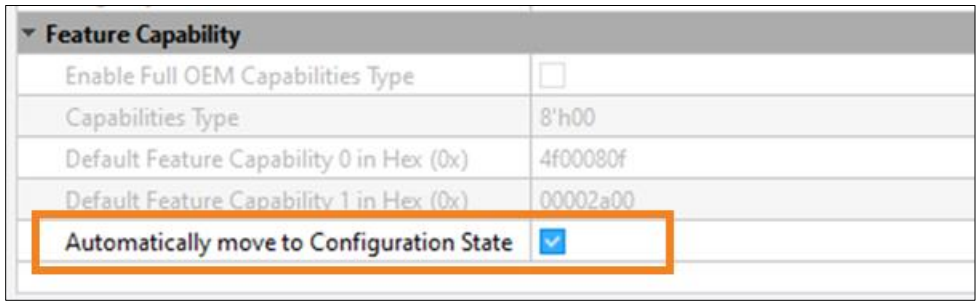


Figure 9.2. Enable Automatically Move to Configuration State

9.1.2. Firmware – Propel C Project

Currently, only the 25 MHz default SDR and 200 MHz DDR options are enabled in the firmware. If the design requires another speed, uncomment on the pll_init function of main.c.

9.2. eSPI Target

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. Hardware

- Using two MachXO5-NX Development Boards and two LTPI connector boards
- Using Promira Serial Platform for the eSPI IP test validation
- Utilizing Sentry 4.0 Board, with MachXO3D device as the M-PESTI targets

10. IP Versions

Table 10.1. IP Versions

IP	Version
AHB-Lite Interconnect	1.3.1
AHB-Lite to APB Bridge	1.1.1
APB Interconnect	1.2.1
eSPI Target	2.0.0.02
GPIO	1.6.2
M-PESTI Initiator	1.1.0
SGPIO Controller	1.1.0
SGPIO Target	1.1.0
Server LTPI	1.0.0
UART	1.3.0
RISC-V SM	1.6.0

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_fw – 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 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 MachXO3D M-PESTI targets design.
- SCM Folder:
  - .metadata – generated by the Lattice Propel Builder software. Do not remove.
  - scm_fw – 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 Diamond project to generate the configuration file.

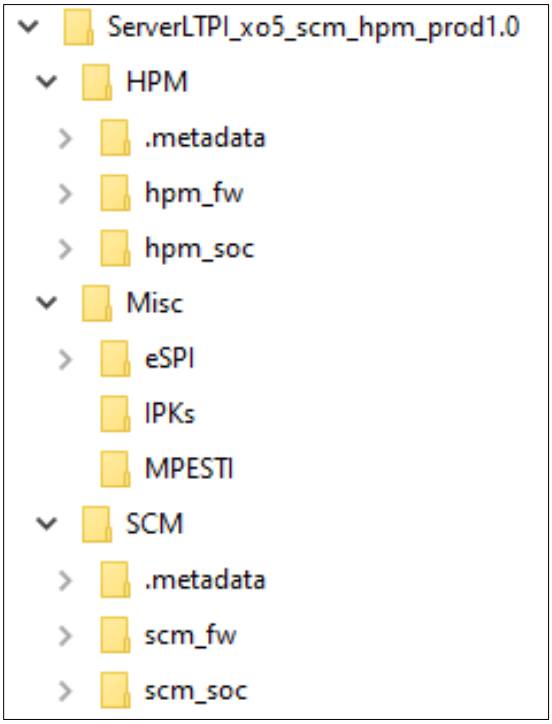


Figure 11.1. Sentry 4.0 Design Package

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 0.80, September 2025

Section	Change Summary
All	Preliminary release.

www.latticesemi.com