MachXO5-NX and L-ASC10 Platform Power Management Using RISC-V Demo

User Guide

FPGA-UG-02194-1.0

October 2023

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 REDUDANCIES, 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.

Contents

Figures

Tables

Acronyms in This Document

A list of acronyms used in this document.

Acronym	Definition
AHBL	Advanced High Performance Bus Light
APB	Advanced Peripheral Bus
API	Application Programming Interface
ASC	Analog Sense and Control
BSP	Board Support Package
CPU	Central Processing Unit
DSP	Digital Signal Processing
EBR	Embedded Block RAM
ECO	Engineering Change Order
EFB	Embedded Function Block
GPIO	General Purpose Input / Output
HDL	Hardware Description Language
HVOUT	ASC10 High Voltage Output (Charge Pump Output)
IMON	ASC10 Current Monitor
IP	Intellectual Property
LED	Light Emitting Diode
LUT	Look Up Table
LUT4	LUT with by-4 Architecture
PFU	Programmable Function Unit
PLL	Phase Locked Loop
PTM	Platform Manager
PTM2	Platform Manager Version 2
RAM	Random Access Memory
RISC-V	Reduced Instruction Set Computer – Version 5
SM0 – SM4	State Machine 0 – State Machine 4 (Logic view in Platform Designer)
SDK	Software Development Kit
SOC	System On a Chip
TMON	ASC10 Temperature Monitor
VMON	ASC10 Voltage Monitor

1. Introduction

This Demo Design and User Guide shows how to combine Lattice's Analog Sense and Control (L-ASC10) devices with Lattice's MachXO5-NX FPGA. The Demo Design is a centralized Platform Management system using three L-ASC10 devices. The Demo Design has three separate power sequencers (one for each L-ASC10) that are cascaded in both powering-up and powering-down. The RISC-V in this Demo Design configures the L-ASC10 devices over an I^2C bus at initial power up; this illustrates how multiple boot and remote updates can be supported. The firmware APIs interface with the centralized Platform Manager to enable both powering-up and powering-down along with fault monitoring.

This User Guide steps thru the several software tools used in building the Demo Design, those tools include; Propel Builder, Propel SDK, Diamond's Platform Designer, Radiant, and ASC_HEX_Reader.exe. Propel Builder creates the RISC-V SOC for the MachXO5-NX device. Propel SDK builds the firmware for the RISC-V CPU. Platform Designer in Diamond builds the power sequences and generates the L-ASC10 HEX file with the analog settings; it also exports the RTL that embodies the three-wire interface for each L-ASC10 and the centralized power sequencer. Radiant combines the RTL generated from Propel Builder and Platform Designer along with the firmware from Propel SDK to generate a single programming file for the MachXO5-NX. The ASC_HEX_Reader.exe is a utility that converts the L-ASC10 HEX programming file into a C=type data array file for inclusion in the firmware.

1.1. Overview

This User Guide describes the MachXO5-NX and (3) L-ASC10s demo project. Figure 1.1 illustrates both the project block diagram and a block diagram of the design flow. The design flow for a project like this one is not a linear flow; rather it can be a bouncing back and forth from several tools to achieve a final system design.

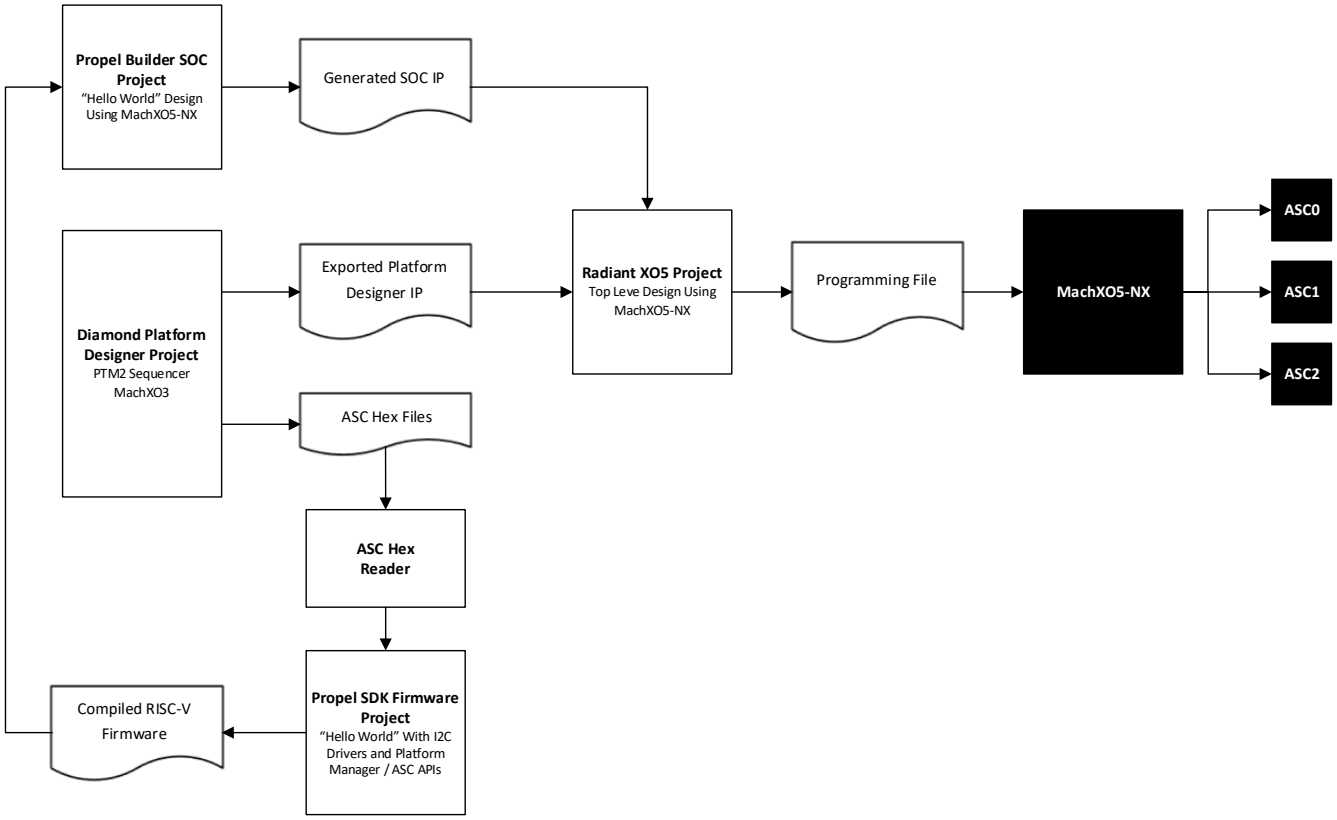

Figure 1.1. MachXO5-NX and (3) L-ASC10s Project Flow Block Diagram

A more detailed diagram of the demo project is provided in Figure 1.2. Here we can see the handshaking connections between the Platform Manager 2 IP and the RISC-V system on a chip (SOC) design. It also shows the I^2C bus connecting all three L-ASC10s to the I^2C Master that is controlled by the RISC-V firmware. Three separate ASC-I/F are shown, one for each L-ASC10. However, the RESETN pins are tied together for a single mandatory reset input to the MachXO5-NX; so that the Platform Manager 2 logic cannot start until all three have powered up. This is a high-level block diagram, the details of which are provided in the sections that follow.

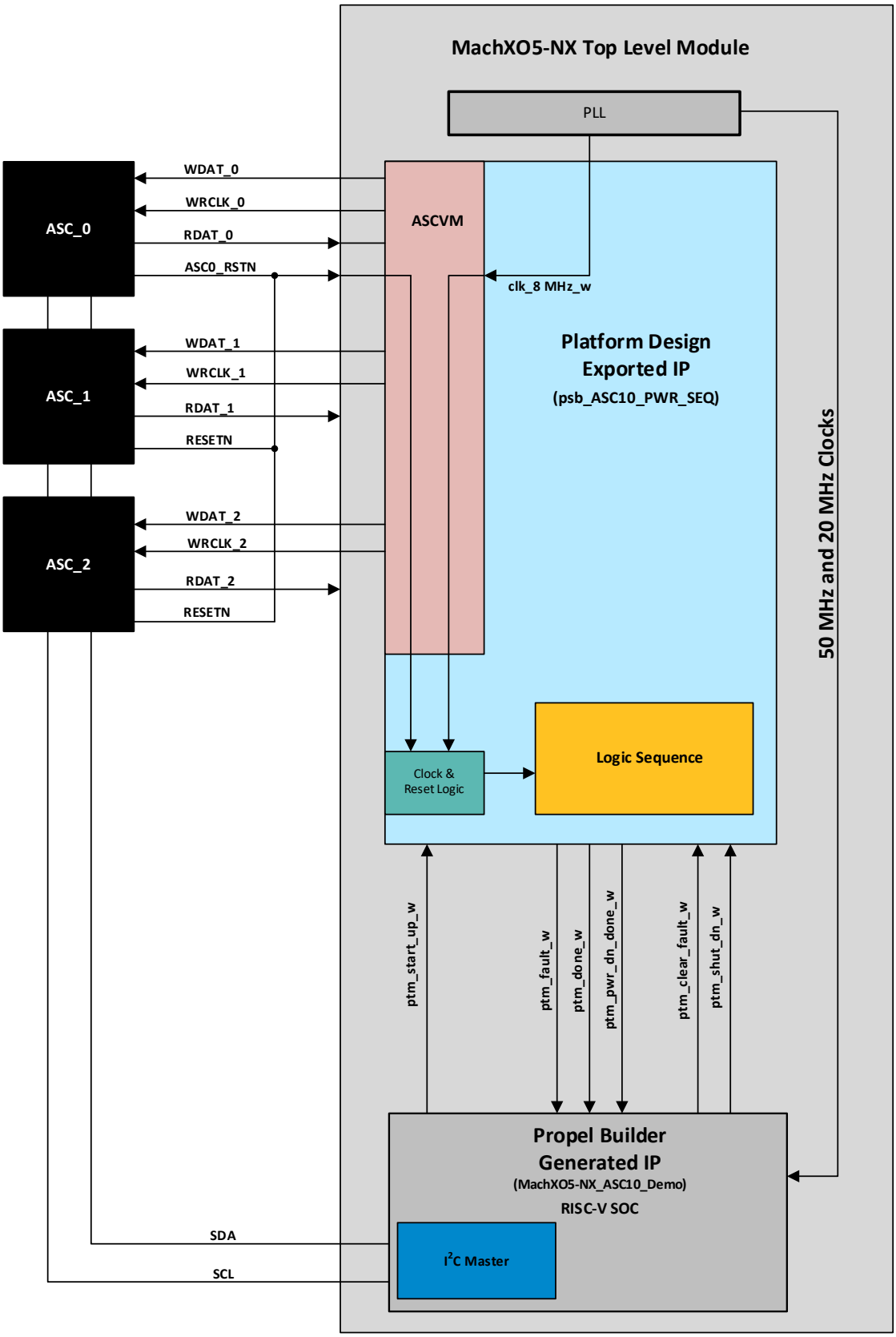

Figure 1.2. MachXO5-NX and (3) L-ASC10s Demo Block Diagram

2. Demo Setup

This section describes the demo setup.

2.1. Software Requirements

- Lattice Propel Builder version 2.2.2207251517
  - Is a graphical interface that is used to connect various IP modules to build a system on a chip (SOC). It generates logic and firmware files for subsequent tool use.
- **Lattice Propel version 2.2.22075251517**
  - Is a software development kit (SDK) that is used to write and compile C-code firmware that will execute on the RISC-V part of the SOC. This tool also contains a terminal view to monitor output from the firmware thru the UART interface.
- **Lattice Diamond version 3.12.1.454 with Patch Version 131140**
  - The Platform Designer tool within Diamond generates the Platform Manager 2 IP to sense and control the system thru the L-ASC10 devices. The Patch Version 131140 enables exporting the RTL and analog settings from the Platform Designer tool.
- **Lattice L-ASC10 HEX File Reader version 2.0**
  - The executable named ASC_Hex_Reader.exe is run in a Command Prompt window to convert the L-ASC10 programming (.hex) file, generated from the Platform Designer tool, into a C-type unsigned char data array file that can be included into the RISC-V firmware.
- **Lattice Radiant version 2.3**
  - This software tool combines all the RTL and firmware to generate a single programming file for the MaxhXO5-NX.
- **Lattice Radiant Programmer version 2.3**
  - This software utility programs the MachXO5-NX evaluation board to verify and debug the demo design.

2.2. Compressed File Contents

When the MachXO5_ASC10_Demo.zip file is unzipped to the root “C:” drive, many folders and files will be available. Here is a short list and description of the more significant folders and files:

The demo project root folder:

```
C:\MachXO5_ASC10_Demo
```

Diamond 3.12 Platform Manager 2 Project File:

```
C:\MachXO5_ASC10_Demo \ASC10_PWR_SEQ.ldf
```

Diamond 3.12 Platform Manager 2 Project Folder:

```
C:\MachXO5_ASC10_Demo \Diamond
```

Exported Platform Manager 2 IP Folder:

```
C:\MachXO5_ASC10_Demo \ASC10_PWR_SEQ_IP
```

ASC HEX Reader Executable file:

```
C:\MachXO5_ASC10_Demo \ASC10_PWR_SEQ_IP\L_ASC10\ASC_HEX_Reader.exe
```

Radiant 3.2 Project File:

```
C:\MachXO5_ASC10_Demo \MachXO5_ASC10_Demo.rdf
```

Radiant 3.2 Project Folder:

```
C:\MachXO5_ASC10_Demo\Radiant
```

Radiant 3.2 Programming File:

```
C:\MachXO5_ASC10_Demo\Radiant\MachXO5_ASC10_Demo_Radiant.bit
```

Propel Builder 2.2 Project Folder:

```
C:\MachXO5_ASC10_Demo\Propel
```

Propel Builder 2.2 Project File:

```
C:\MachXO5_ASC10_Demo\Propel\MachXO5_ASC10_Demo\MachXO5_ASC10_Demo.sbx
```

Propel 2.2 SDK Workspace Folder:

```
C:\MachXO5_ASC10_Demo\PPL_Workspace
```

Platform Manager 2 API Source Folder:

```
C:\MachXO5_ASC10_Demo\PPL_Workspace/MachXO5_ASC10_Demo/src
```

2.3. Hardware Requirements

The following list of hardware is required to run this demo design and Figure 2.1 shows the actual hardware running the demo design. Note the VMON slider POTs in different locations; this will be discussed in more detail in later sections.

- MachXO5-NX Development Board
- ASC Bridge Board
- 3x ASC Breakout Boards
- 2x USB cables
- 12 V Wall wart (power supply)

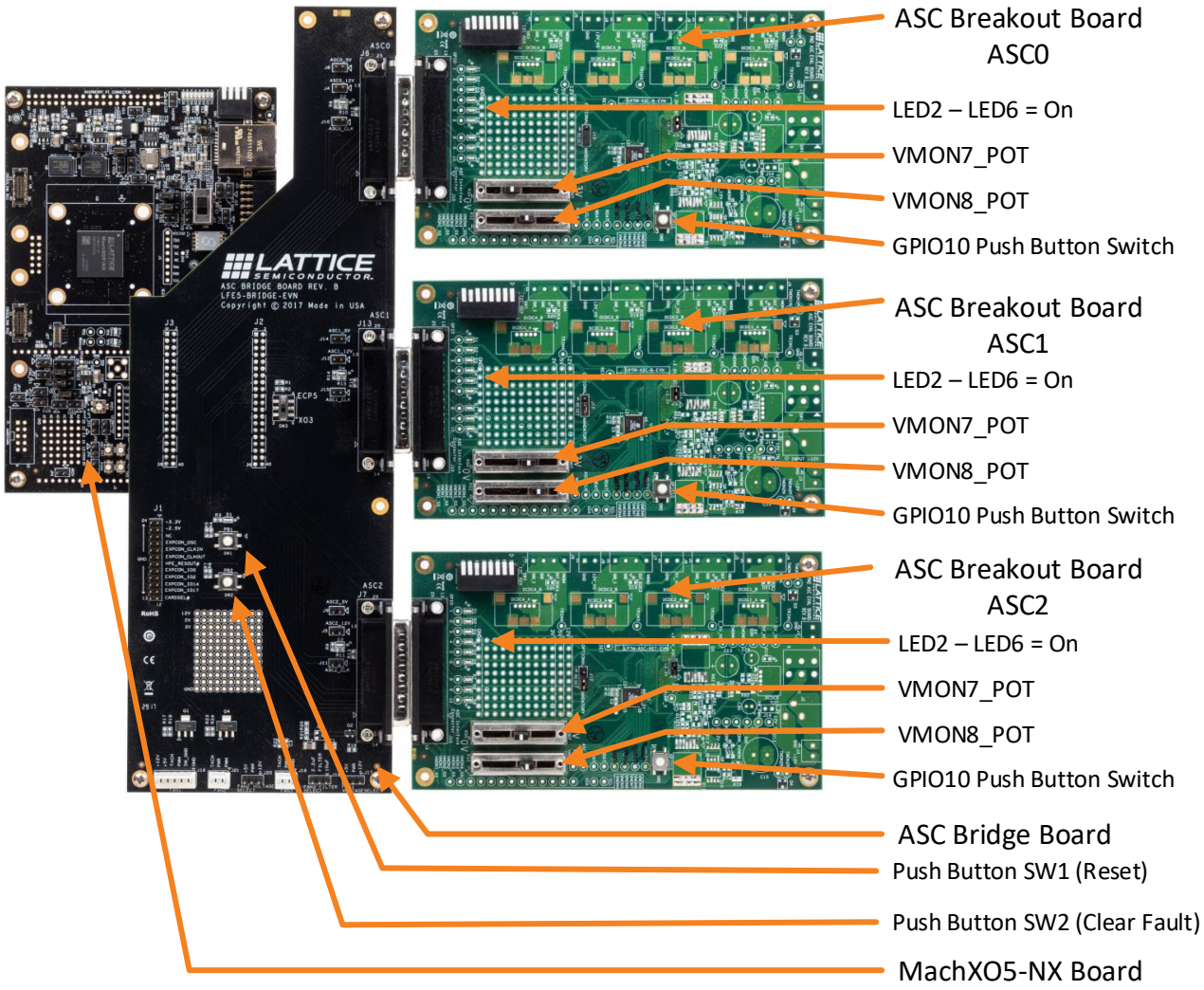

Figure 2.1. MachXO5-NX and (3) L-ASC10s Demo Hardware

3. Initial Step by Step Design Flow

This is just one example of a flow to start this demo design. A project like this could also start from an existing Platform Manager 2 design, RISC-V firmware, or MachXO5-NX SOC design. Also, this step-by-step flow combines several iterations of the build, program, and test process that were part of the demo development. For example, after the whole project was initially built in Radiant, the number of GPIO lines in the Propel Builder component was increased from eight to ten, and later to 11 in separate build cycles.

3.1. Creating a new project in the Propel Builder

1. From the Propel Builder menu, click **File->New Design** the **Design Information** window will appear, as shown in Figure 3.1. Select the project type and fill in the name and path then click **Next**.

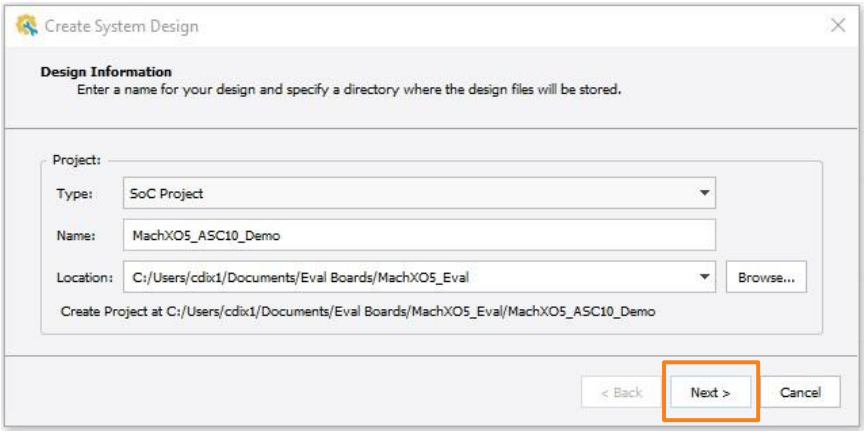

Figure 3.1. Propel Builder – Project Name

2. The **Configure Propel Project** window will be displayed, as shown in Figure 3.2. In this window, select the **Language**, **Device**, **Processor**, and **Template** then click **Next**.

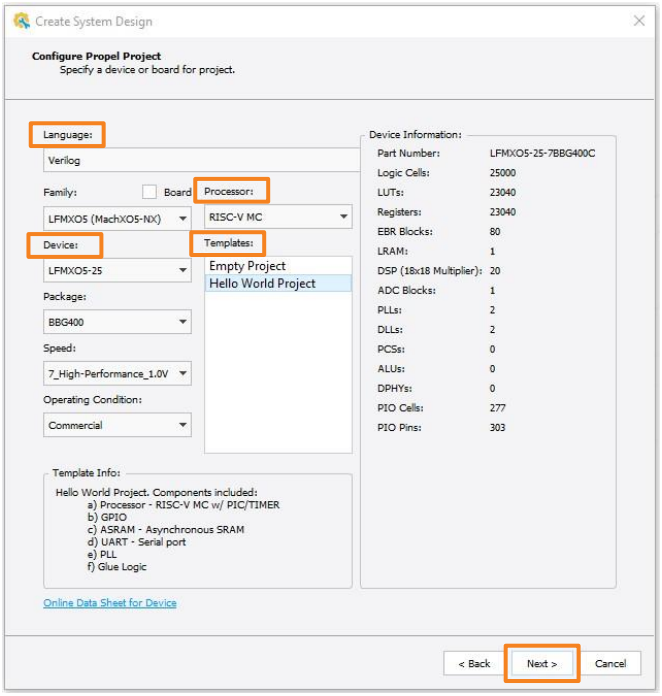

Figure 3.2. Propel Builder – Device and Template

3. The **Project Information** window will appear, as shown in Figure 3.3. To summarize the project click **Finish** to generate the project.

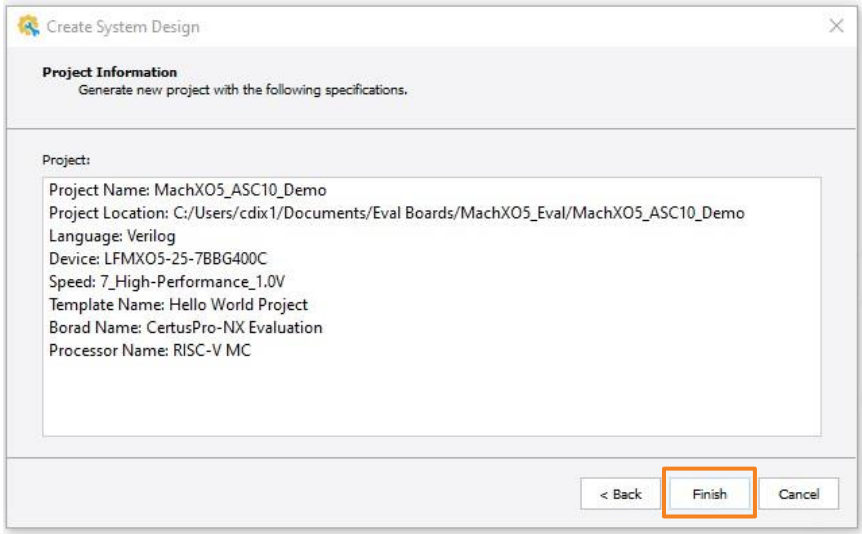

Figure 3.3. Propel Builder – Summary

4. The Propel Builder will now display the SOC high-level schematic, as shown in Figure 3.4.

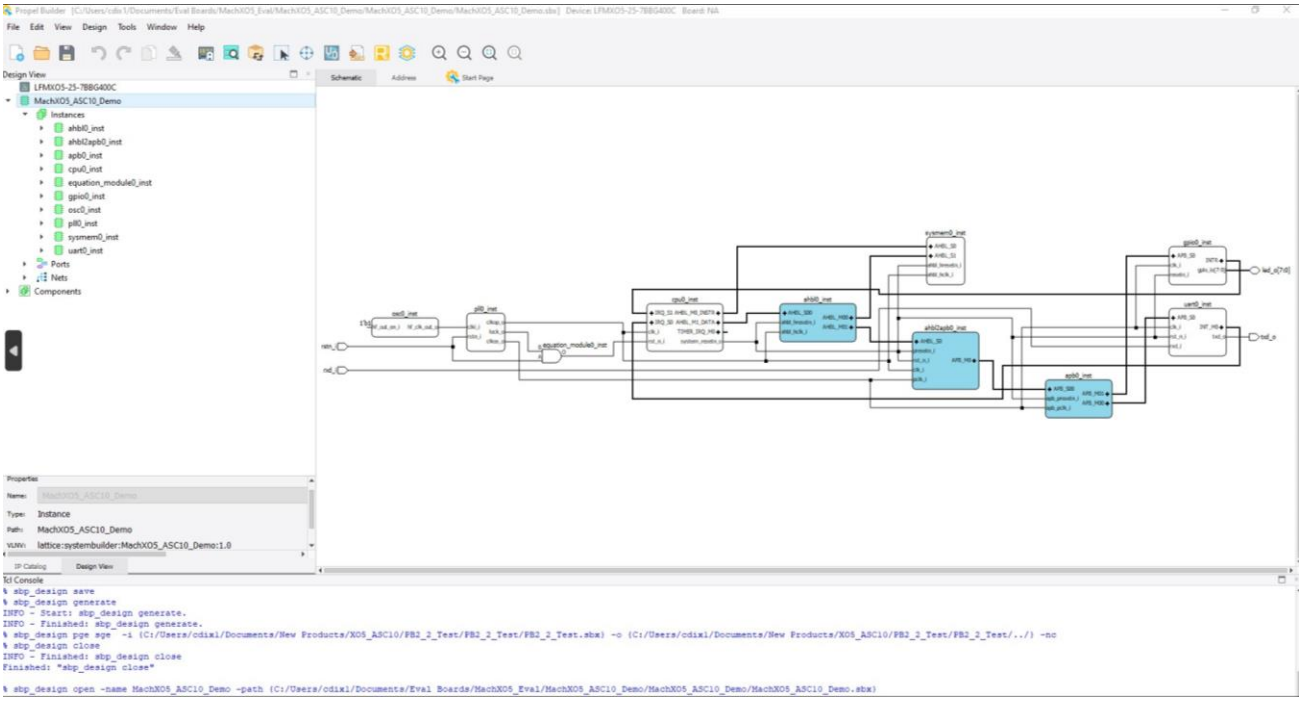

Figure 3.4. Propel Builder – Initial Schematic

3.2. Adding I²C Master IP to the Design

If you don't have the I²C Master IP in the IP on Local tab (under the IP Catalog), do the following steps:

1. Click on the IP on Server tab and find the I²C Master in the list, as shown in Figure 3.5. and click to download it.

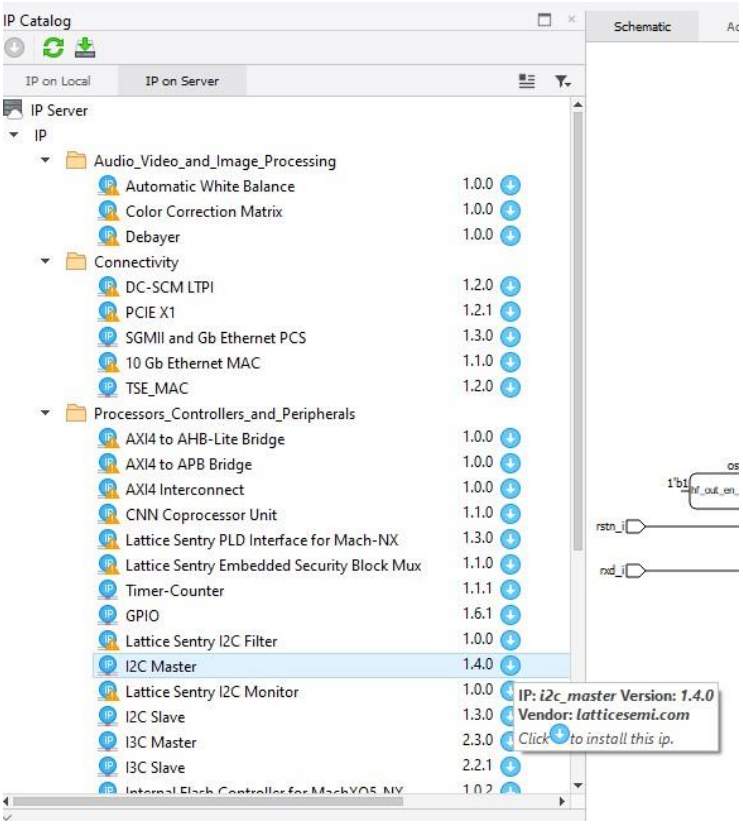

Figure 3.5. Propel Builder – I²C IP on Server

2. A license agreement will be appear as shown in Figure 3.6. Click **Accept** to install the I²C Master IP.

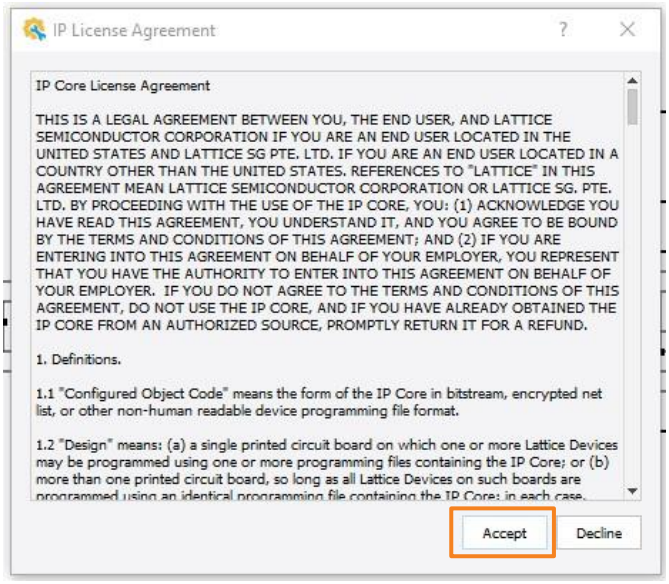

Figure 3.6. Propel Builder – I²C IP License

3. The Propel Builder will ask for a **Component Name** for the I²C Master, as shown in Figure 3.7.

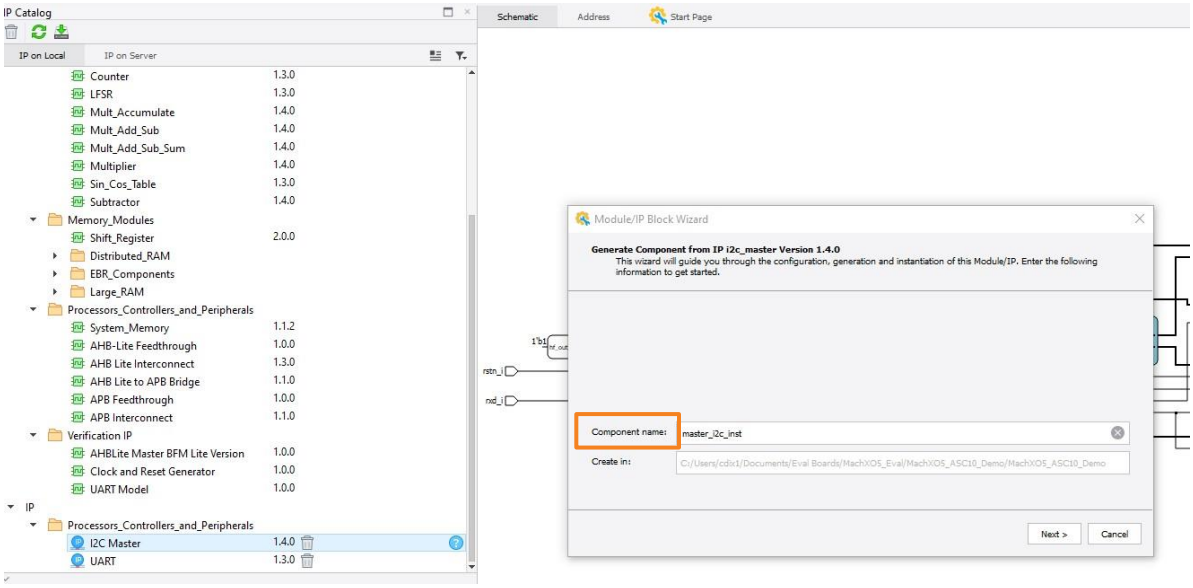

Figure 3.7. Propel Builder – Instantiate I²C Block

4. The IP configuration window will appear as shown Figure 3.8. Edit the settings and click **Generate**.

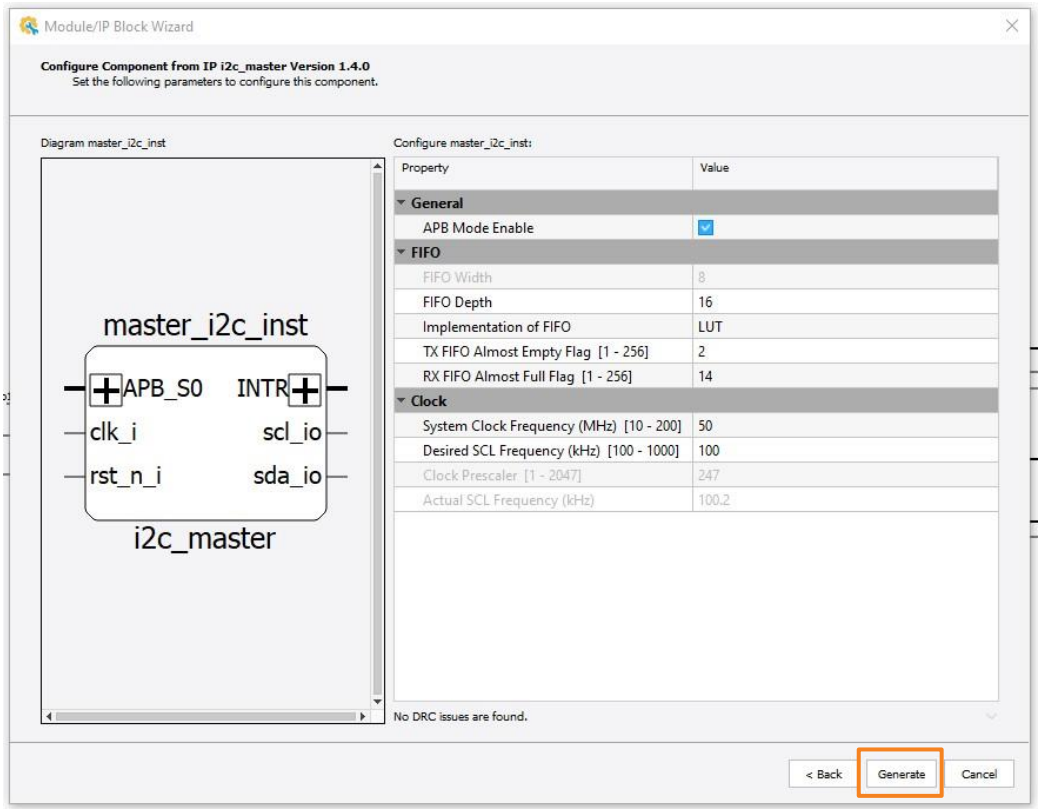

Figure 3.8. Propel Builder – Configure I²C IP

5. Click **Finish** and ensure the **Insert to project** checkbox is ticked, as shown in Figure 3.9.

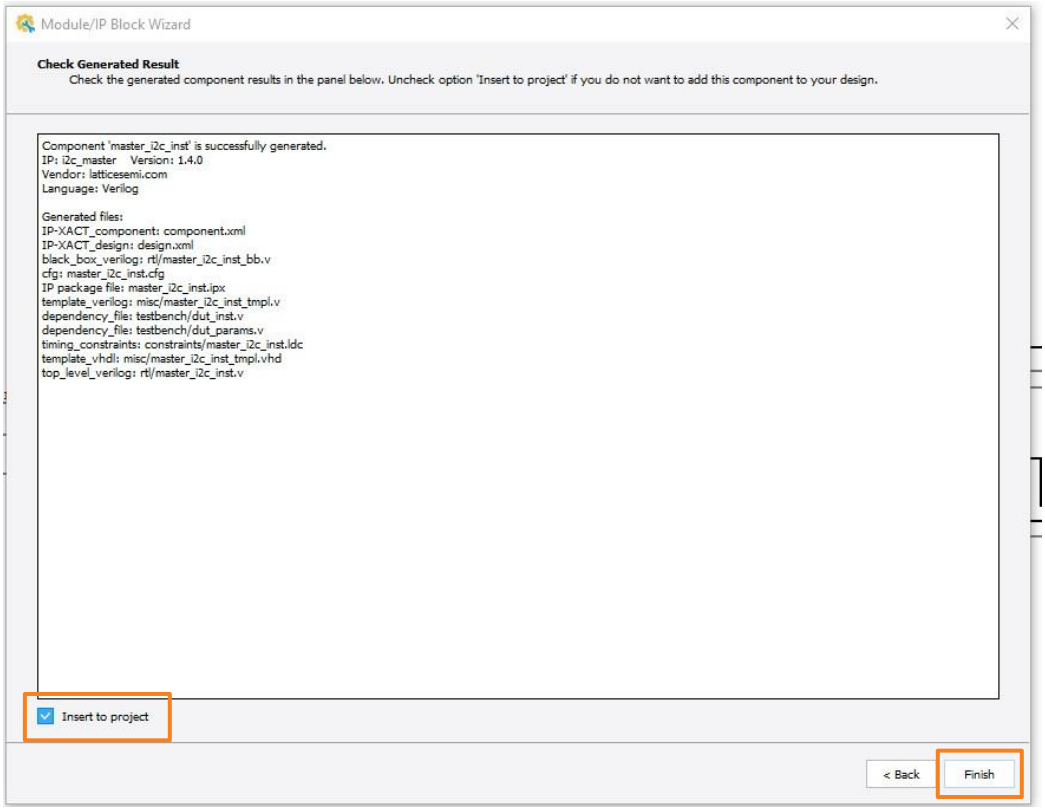

Figure 3.9. Propel Builder – Finish

6. The Propel Builder will request a name for the instance of the I²C Master IP, as shown in Figure 3.10.

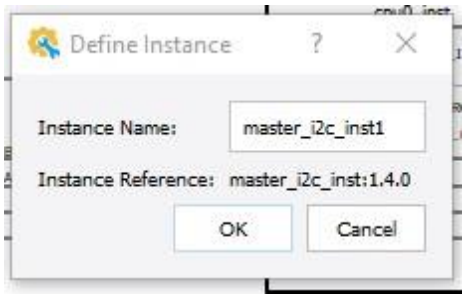

Figure 3.10. Propel Builder – Instance Name

7. The Propel Builder will drop the unconnected I²C Master component on the schematic, as shown in Figure 3.11.

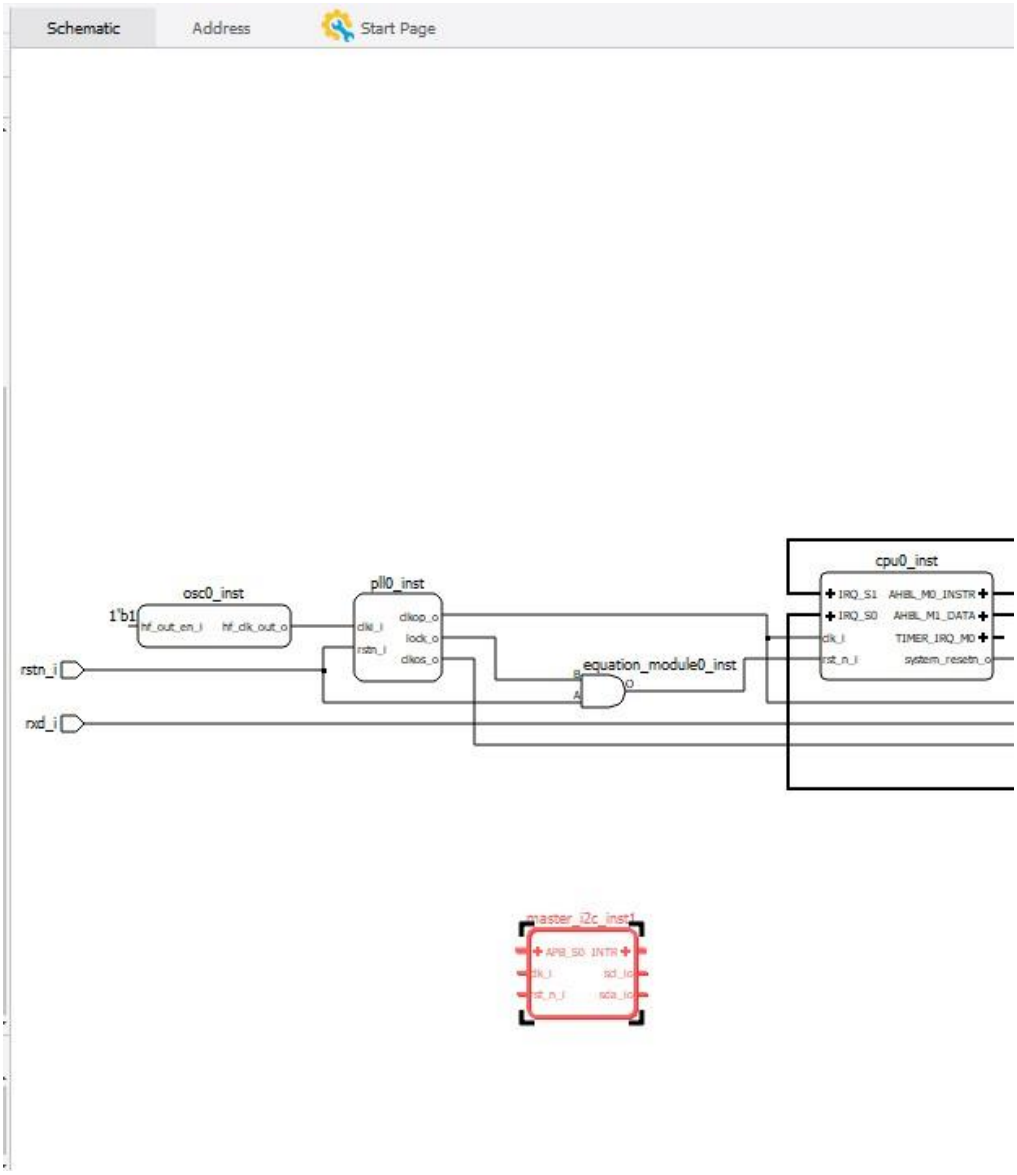

Figure 3.11. Propel Builder – I²C Master added to Schematic

8. To connect the I²C Master to the SOC we need to add a slave connection to the APB Interconnect. Double-click on the APB component to change the Total APB Slaves from 2 to 3, as shown in Figure 3.12 then click **Generate**.

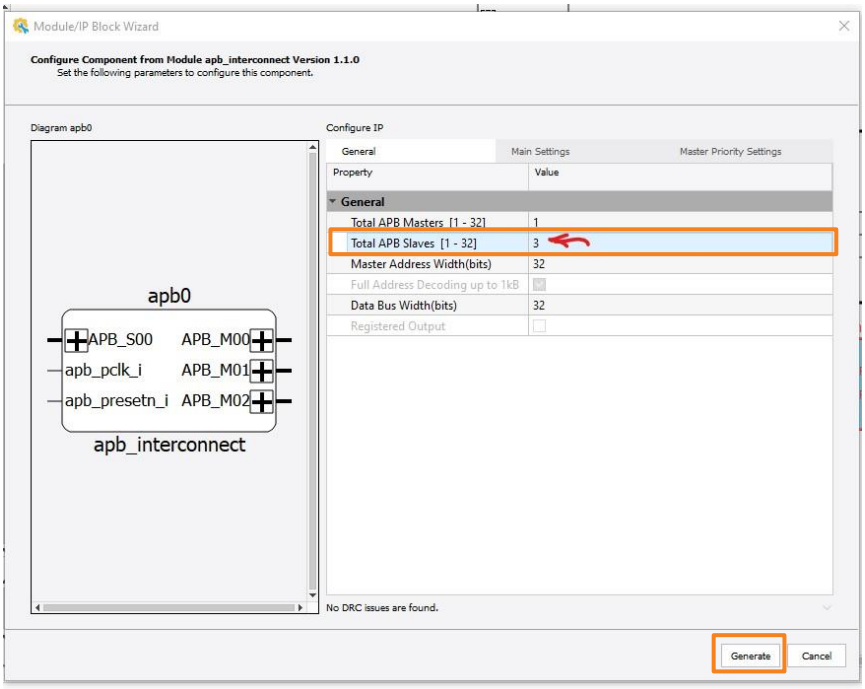

Figure 3.12. Propel Builder – Adding Slave to APB

9. Click and drag the I²C Master component to the right side of the schematic. Then click on the APB_M02 port and drag a connection to the APB_S0 port of the I²C Master component, as shown in Figure 3.13. Note the green check-mark indicates the width and direction of the ports match and is OK to connect the two.

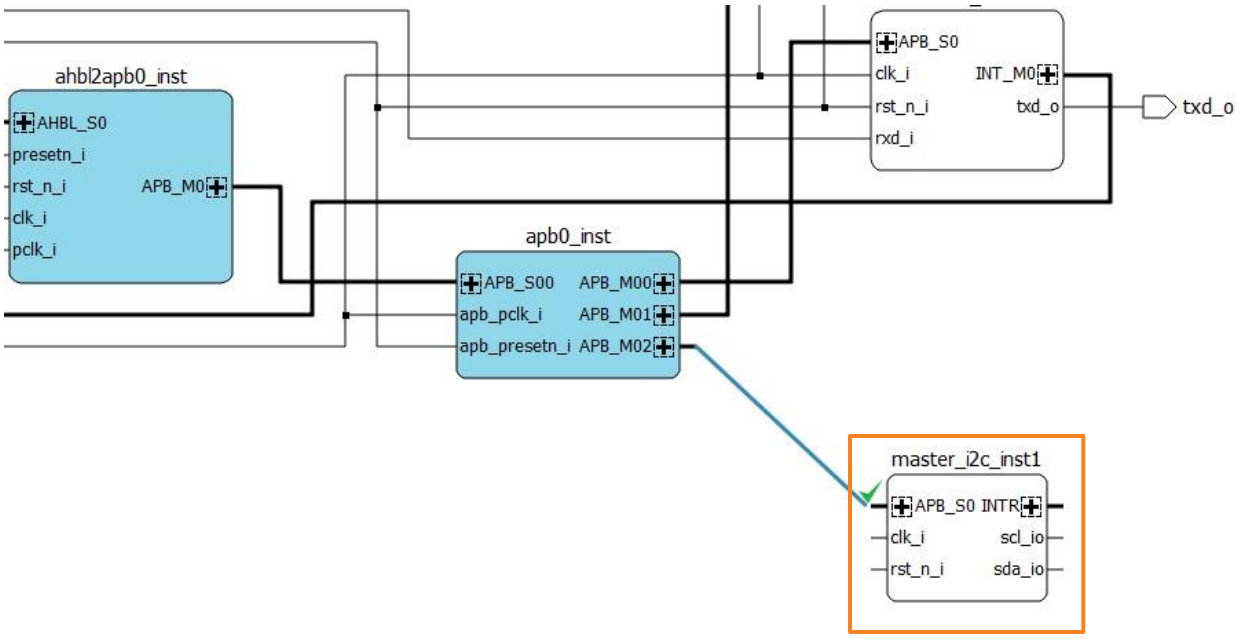

Figure 3.13. Propel Builder – Connecting I²C IP to APB

10. In a similar manner, connect the clk_i and rst_n_i ports of the I^2C Master to the existing nets. Next, right-click in a blank area of the schematic to open the pop-up menu, as shown in Figure 3.14, and select **Create Port.**

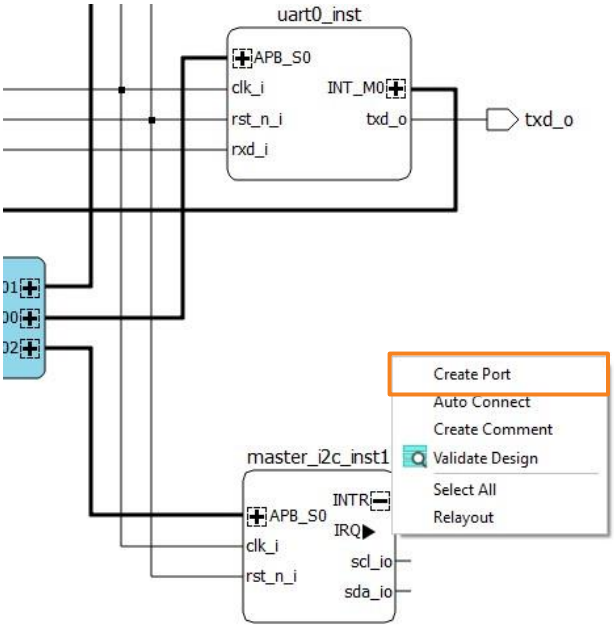

Figure 3.14. Propel Builder – Adding a Port

11. In the **Create Port** window, as shown in Figure 3.15, **Name** the port **ptm_scl**, select the port **Direction** to **Inout**, and click **OK**. Repeat this process to create an **Inout** port named **psb_sda**.

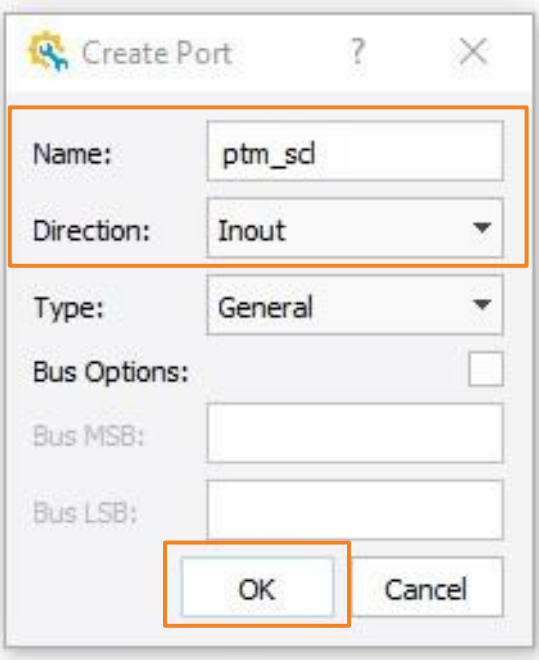

Figure 3.15. Propel Builder – Adding ptm_scl Port

12. Connect the newly created ports to the respective ports of the I²C Master component, as shown in Figure 3.16.

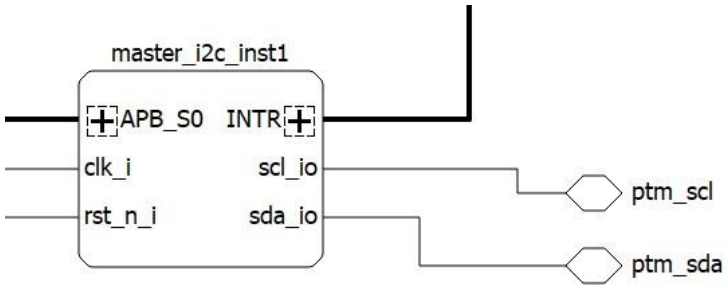

Figure 3.16. Propel Builder – I²C Ports Connected

3.3. Modifying the GPIO component

The GPIO component from the Propel Builder template only supports an eight-bit bi-directional port. In this demo design additional bits are needed to handshake between the RISC-V firmware and the Platform Manager 2 design.

1. Double-click on the **gpio0_inst** component to open the IP configuration window, as shown in Figure 3.17.
2. Tick the **Remove Tri-State Buffer** checkbox to create both an input and output port.
3. Change the **Number of I/O Lines** from 8 to **11**.
4. Set the **I/O Direction** to all outputs **(7FF)**.
5. Click **Generate**.

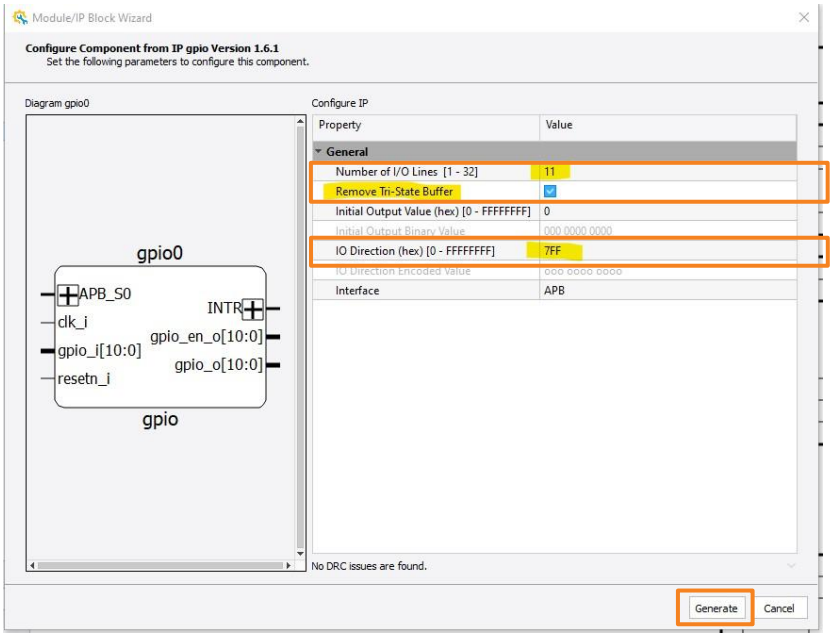

Figure 3.17. Propel Builder – GPIO IP Configuration

6. Right-click on the **gpio0_inst** component to open the pop-up menu and select the **Auto Connect** as shown in Figure 3.18. The Propel Builder will generate and connect the input, output, and output enable ports.

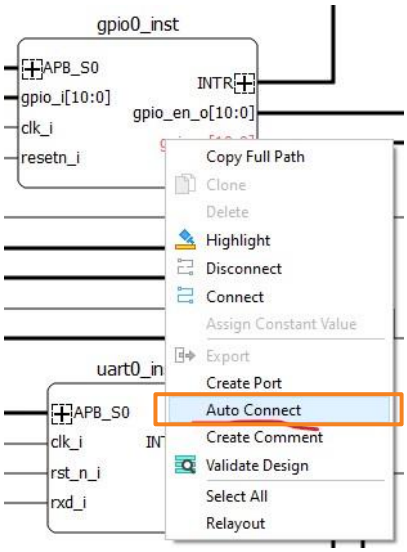

Figure 3.18. Propel Builder – GPIO Auto Connect

7. The auto connect will generate port names that are verbose. Click on the port and edit the **Name** in the **Properties** box in the lower left of the screen as shown in Figure 3.19.

Figure 3.19. Propel Builder – Renaming GPIO Port

3.4. Adding and Modifying the Clocks

The primary clock for this design is the built in oscillator of the MachXO5-NX.

1. Double-click on the **osc0_inst** component to open the IP configuration window as shown in Figure 3.20. Change the **HFCLK Frequency** to **90.0 MHz**. Then click **Generate**.

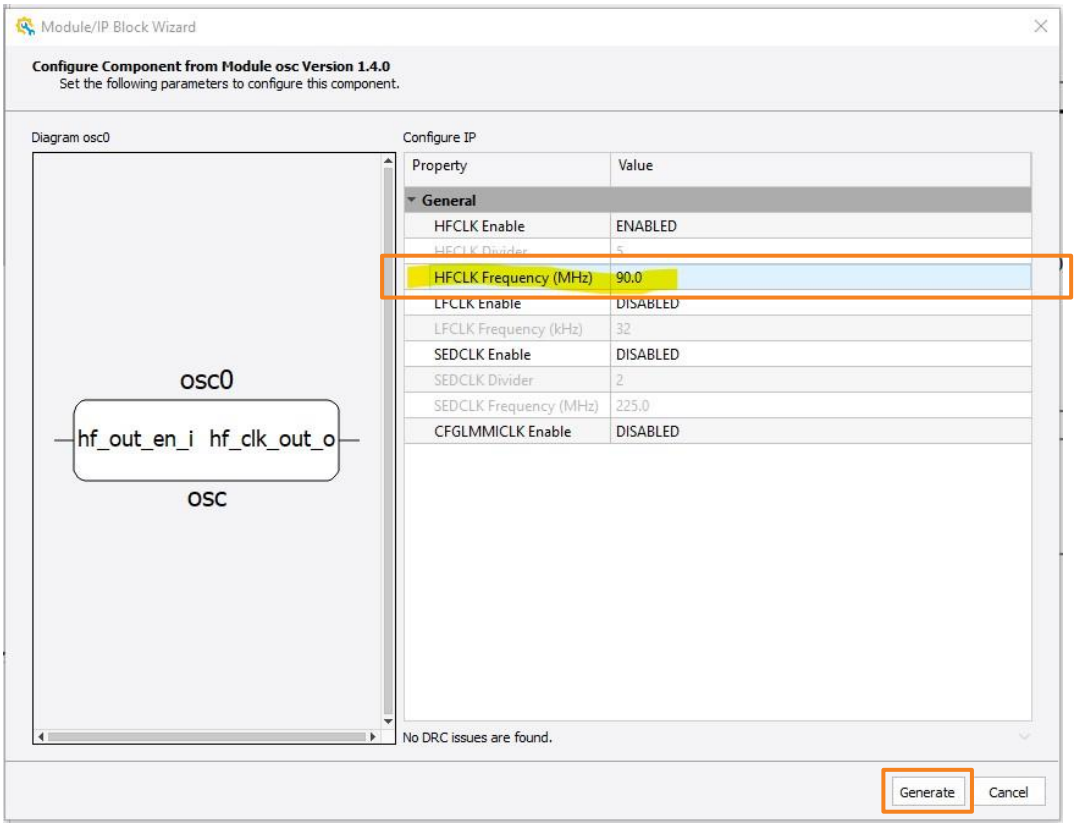

Figure 3.20. Propel Builder – Oscillator IP Configuration

2. Double-click on the **pll0_inst** component to open the IP configuration window as shown in Figure 3.21.

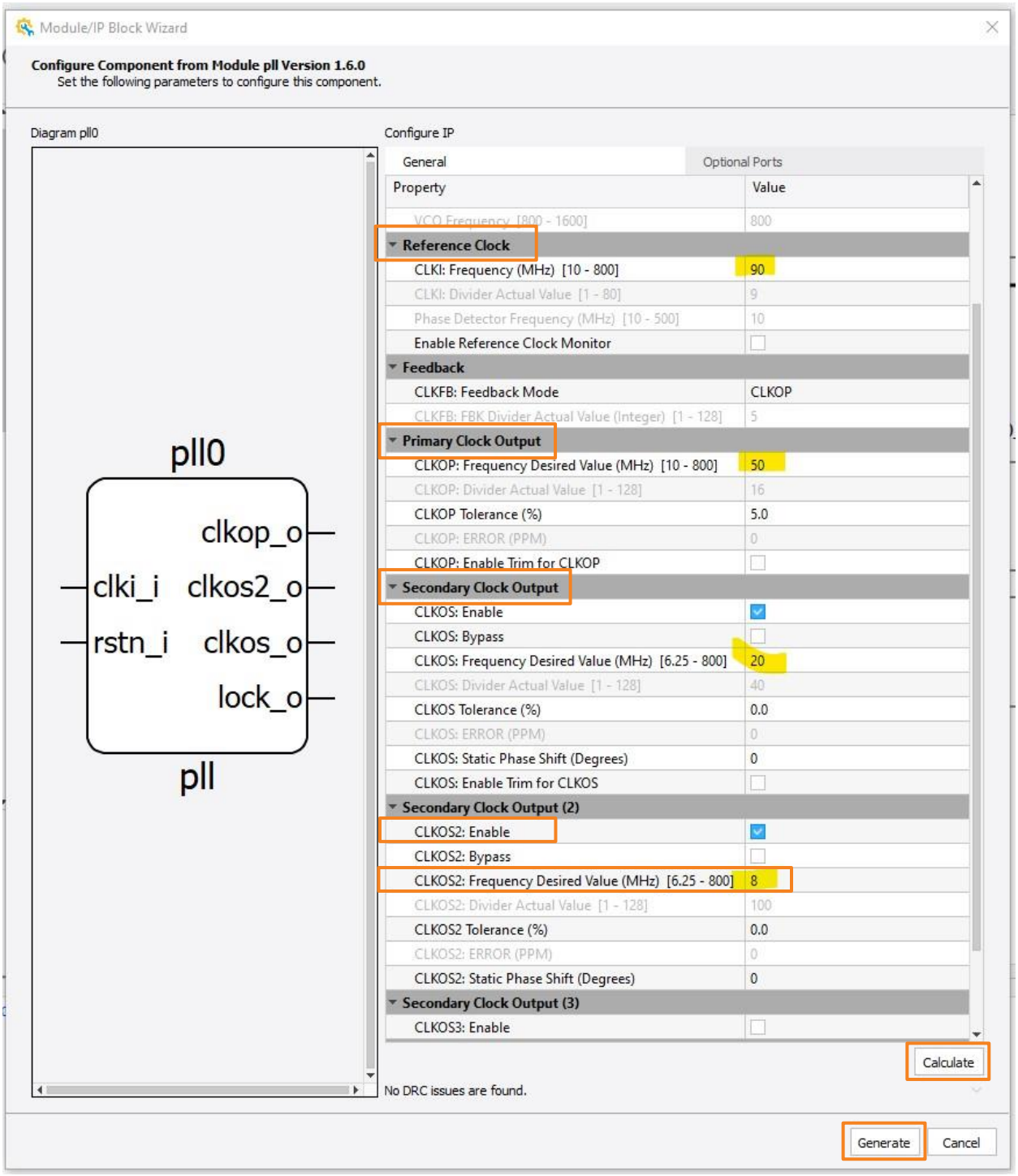

Figure 3.21. Propel Builder – PLL IP Configuration

3. Change the **Reference Clock** to match the oscillator output of **90.0 MHz**.
4. Change the **Primary Clock Output** to **50.0 MHz** with a **5.0 % CLKOP tolerance**; this is the RISC-V clock.
5. Change the **Secondary Clock Output** to **20.0 MHz**; this is the peripheral clock for UART, GPIO, and I^2C components.
6. Tick the **CLKOS2:Enable** checkbox and set the **frequency** to **8.0 MHz**; this is the Platform Manager 2 IP clock.

7. Click the **Calculate** button and then click **Generate** to update the PLL.
8. Add an output port named **Clk_8MHz** and connect it to the **pll0_inst clkos2_o** port to enable connection to the Platform Manager 2 IP with the 8 MHz clock. This may be required because the **ASC0_CLK**, that is normally the clock source for Platform Manager 2 designs, is not connected in the hardware to a MachXO5-NX primary clock pin. The Lattice LSE synthesis tool in Radiant generates an error when clocks are not connected to primary clock pins.
9. Add an output port named **Clk_20MHz** and connect it to the clock bus that is connected to the peripheral **clk_i** ports to drive additional logic in the top level design (described later in this document) as shown in Figure 3.22.

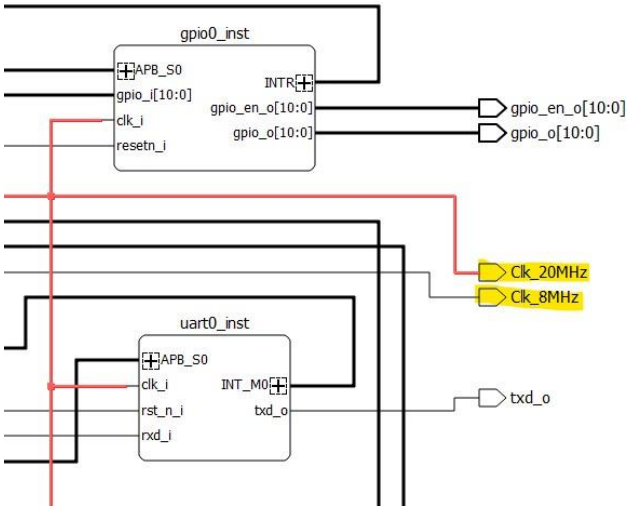

Figure 3.22. Propel Builder – Clock Ports

3.5. Updating the RISC-V IP

1. Double-click on the **cpu0_inst** component and the Unable to find IP message may pop-up.
2. Click **Yes** to update to the later version of the RISC-V IP as shown in Figure 3.23.

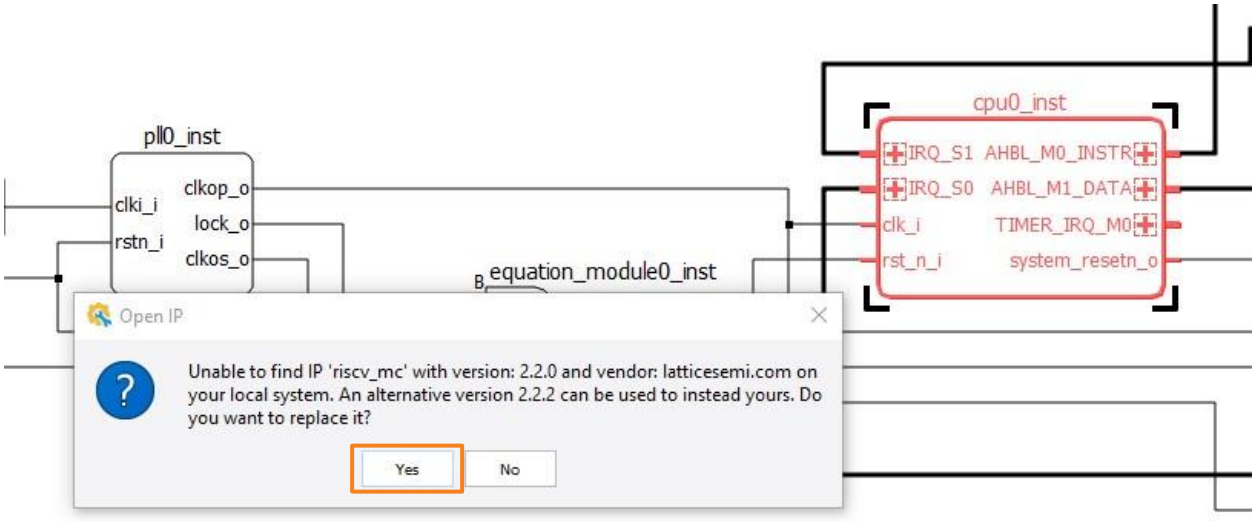

Figure 3.23. Propel Builder – Update RISCV IP

3.6. Checking the Memory Map

1. Click on the **Address** tab to show the system memory map.
2. Click to expand the memory display and to show where the peripherals are mapped as shown in Figure 3.24.
3. Ensure the memory addresses do not overlap and the **Lock** checkboxes are ticked.

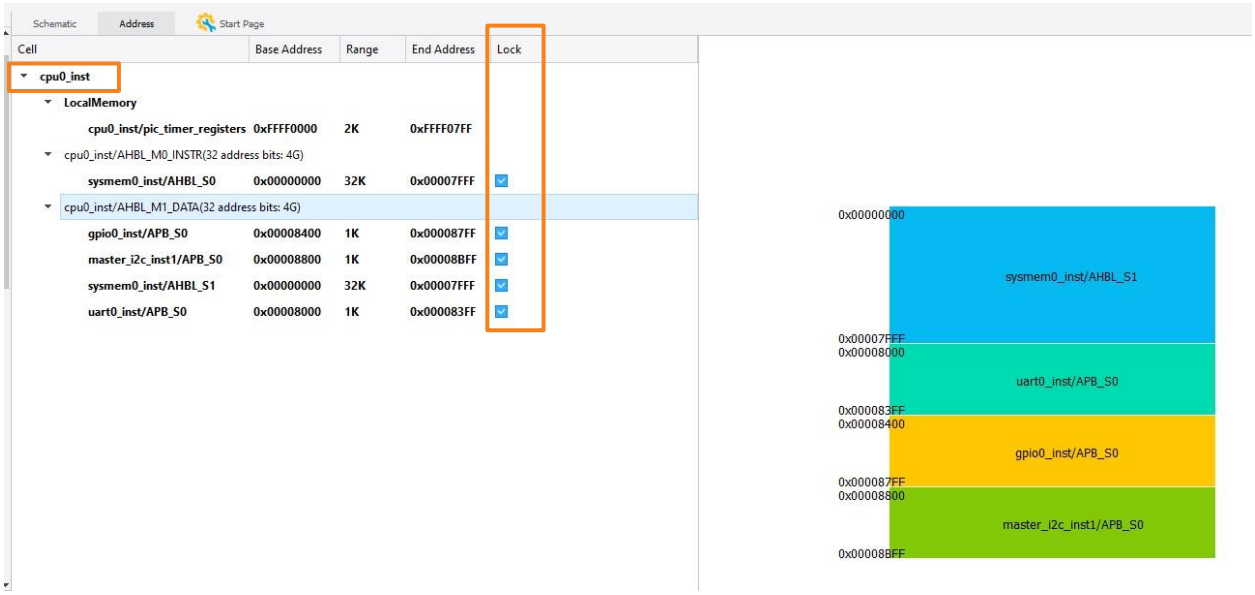

Figure 3.24. Propel Builder – Memory Map

3.7. Generating the SOC RTL

The final step in creating the SOC RTL is to click on the **Generate** icon as shown in Figure 3.25, or click **Design-> Generate**. This will generate a top-level SOC module that can be instantiated in the overall top level module that also includes a top-level module for the Platform Manager 2 IP.

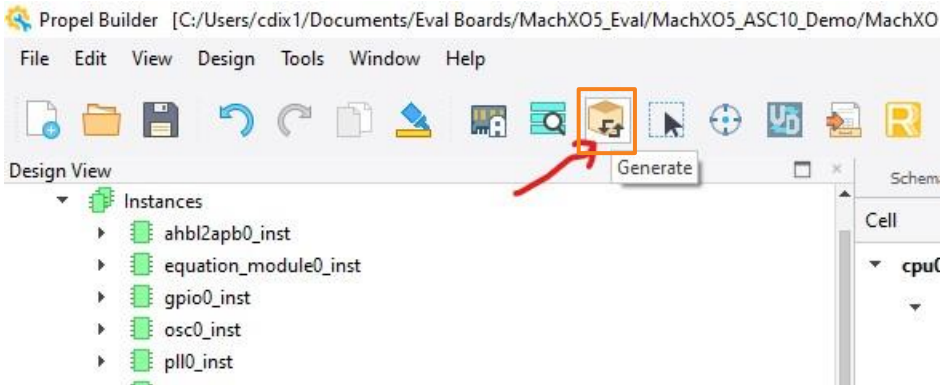

Figure 3.25. Propel Builder – Generating the Design

3.8. Creating the Platform Manager 2 IP

1. In the Lattice Diamond software, click **File**->**New**->**Project** to create a new project named ASC10_PWR_SEQ.
2. Select the **MachXO3LF-4300C** (or similar) device and add three **L-ASC10** devices as shown in Figure 3.26.

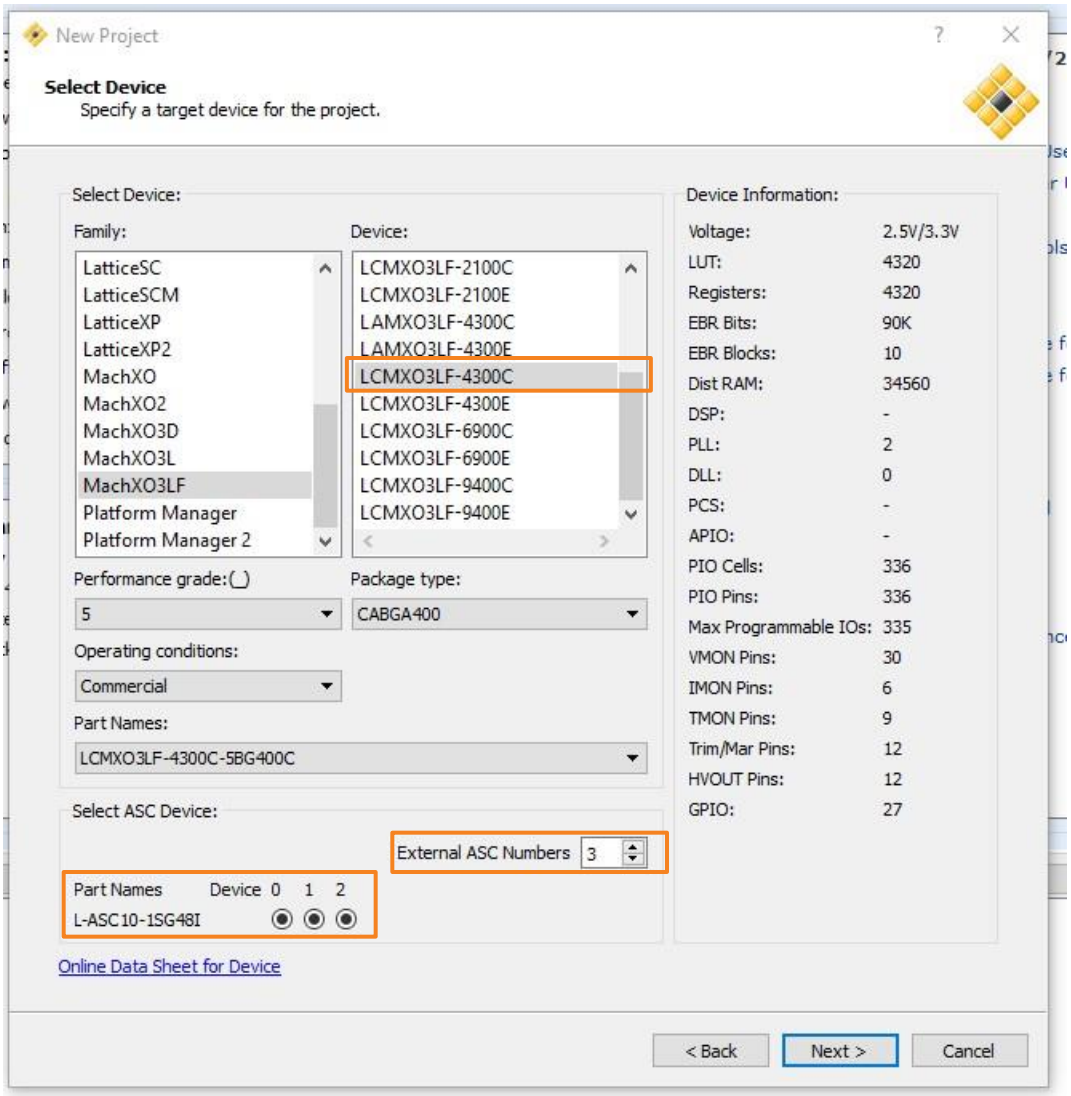

Figure 3.26. Lattice Diamond – New Platform Designer Project

3. Click **Next** in the following windows until the project is generated.
4. After the **Platform Designer** tool opens, select the **Global** view and the **ASC** Options tab.
5. Change the **Reset Type** for ASC1 and ASC2 to **Mandatory**, as shown in Figure 3.27. This will require all the L-ASC10 RESETb pins to be connected together (wire-OR'd), so that the sequence cannot start until all three L-ASC10s have completely powered up. The wire or'ing in hardware is done with jumper J12 on the ASC Breakout boards.

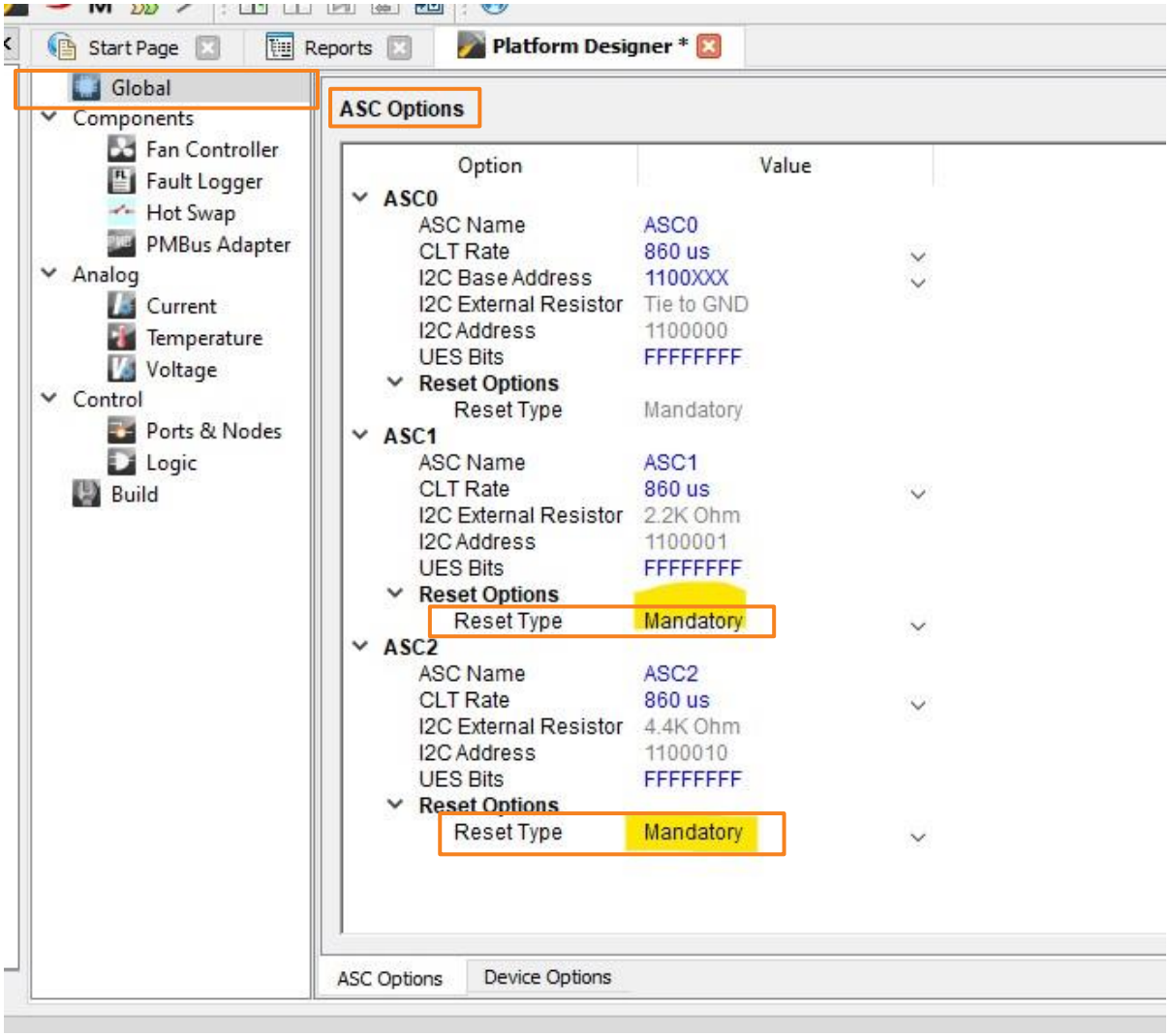

Figure 3.27. Lattice Diamond – ASC Options

6. Click the **Device Options** of the **Global** view, and check the **Export Platform Designer IP** box, as shown in Figure 3.28. Note that the Clock Source is set to Global_Clock to support connection to the PLL 8 Mhz output when using the Lattice LSE synthesis tool.
7. Select **Nexus** as the family to **Export To** and select the **Export IP Folder**. The name of top-level module for the exported IP defaults to a prefix of *psb_* added to the project name. The **Export IP File Name** can be renamed by the user in this view.

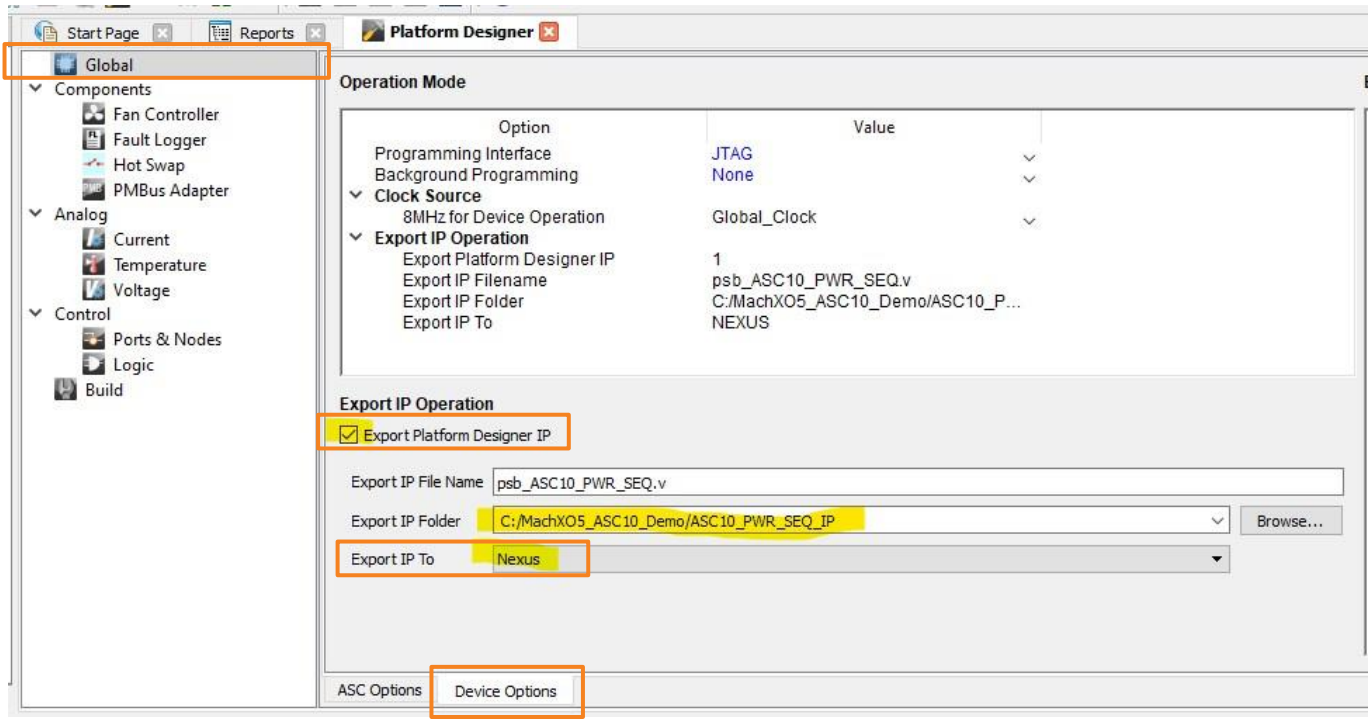

Figure 3.28. Lattice Diamond – Global Device Options

8. Select the **Logic** view and click the **Inserted HDL** tab.
9. Uncheck the **Enable EFB Module** because, the MachXO5-NX does not have an embedded function block (EFB).

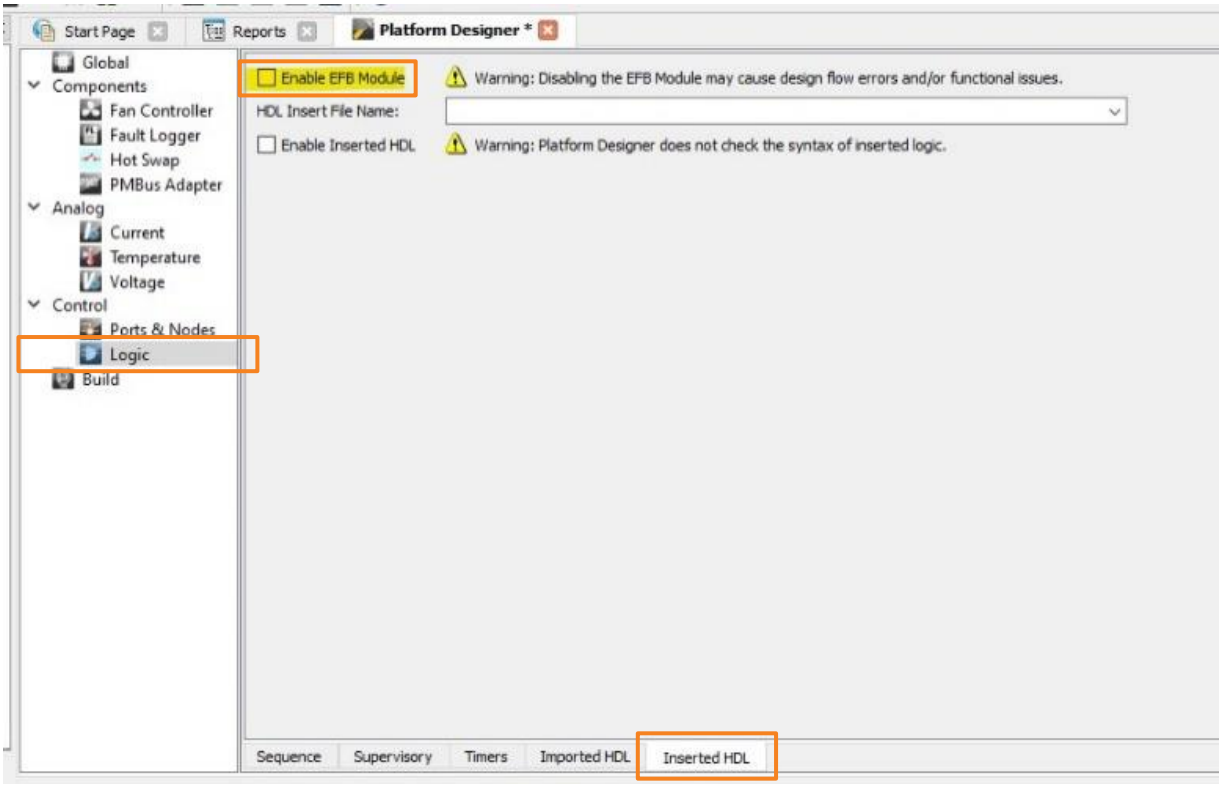

Figure 3.29. Lattice Diamond – Disable EFB Logic

Now, the Platform Manager 2 design process can proceed just like any other Platform Manager 2 design by configuring the analog settings (VMON, IMON, TMON, and HVOUT) and building the sequencer logic.

In this demo design, each ASC Breakout board has its own sequence (SM0 - SM2). There are two additional sequences to deal with Fault Logging (SM3 & SM4) for a total of five separate sequences. There is also three supervisory equations, shown in Figure 3.31, to support fault logging and sequencing.

Each ASC sequence is similar in that they enable pseudo power supplies (GPIO LEDs) and wait for the supply voltage (VMON7 and VMON8 slide POTs) to be within range before completing the sequence. The ASCs are cascaded so that after ASC0 has completed its sequence, then ASC1 starts and finally ASC2 starts its sequence when ASC1 is done. The entire sequence is initiated from the RISC-V firmware by setting a GPIO bit (more about this in the firmware and Radiant discussions).

Similarly, the power down sequence cascades in reverse order from ASC2, to ASC1, and finally ASC0 without waiting for the supply voltage (VMON7 and VMON8 slide POTs) to drop. The power down sequence can be triggered by any of the GPIO10 push-buttons on the ASC Breakout boards or from the RISC-V firmware by setting a GPIO bit (more about this in the firmware and Radiant discussions).

In this demo design the VMON trip points used different values to verify that each ASC was configured with its unique and correct programming pattern. The values used are shown in Table 3.1.

Table 3.1. Platform Designer – VMON Trip Points

ASC	VMON	Logical Name	Lower Trip Point	Upper Trip Point
0	VMON7	A0_VM7_POT_OK	0.667 V	0.964 V
0	VMON8	A0_VM8_POT_OK	1.205 V	1.805 V
1	VMON7	A1_VM7_POT_OK	1.197 V	1.805 V
1	VMON8	A1_VM8_POT_OK	1.592 V	2.401 V
2	VMON7	A2_VM7_POT_OK	1.592 V	2.415 V
2	VMON8	A2_VM8_POT_OK	1.999 V	2.999 V

The three ASC sequences are similar and follow the flow listed in Table 3.2.

Table 3.2. Platform Designer – SM1: ASC1 Power Sequence

Step	Instruction	Outputs	Comment
0	Begin Startup Sequence		
1	Wait for ASC1_AGOOD	ASC1_GPIO1 – ASC1_GPIO9 = 1 ASC1_Seq_Done_node = 0 ASC1_Seq_Pwr_Dn_node = 0	Turn LEDs Off – Wait for AGOOD and reset Power Up and Power Down flags
2	Wait for A1_VM7_POT_OK AND ASC0_Seq_Done_node	ASC1_GPIO2 = 0	Turn LED2 On and wait for VMON7 voltage to be in window and ASC0 sequence done
3	Wait for 250ms using Timer		Sequence Delay
4	ASC1_GPIO3 = 0		Sequence LED3 On
5	Wait for A1_VM8_POT_OK	ASC1_GPIO4 = 0	Turn LED4 On and wait for VMON8 voltage to be in window
6	Wait for 250ms using Timer		Sequence Delay
7	ASC1_GPIO5 = 0		Sequence LED5 On
8	Wait for 250ms using Timer		Sequence Delay
9	ASC1_GPIO6 = 0		Sequence LED6 On
10	Wait for 250ms using Timer		Sequence Delay
11	ASC1_Seq_Done_node = 1		Enable ASC2 to sequence up
12	Begin Shutdown Sequence		
13	Wait for ASC2_Seq_Pwr_Dn_node		Wait for ASC2 to complete sequence down
14	ASC1_GPIO6 = 1, ASC1_Seq_Done_Node = 0		Sequence LED6 Off and reset Power Up Done
15	Wait for 250ms using Timer		Sequence Delay
16	ASC1_GPIO5 = 1		Sequence LED5 Off
17	Wait for 250ms using Timer		Sequence Delay
18	ASC1_GPIO4 = 1		Sequence LED4 Off
19	Wait for 250ms using Timer		Sequence Delay
20	ASC1_GPIO3 = 1		Sequence LED3 Off
21	Wait for 250ms using Timer		Sequence Delay
22	ASC1_GPIO2 = 1		Sequence LED2 Off
23	Wait for 250ms using Timer		Sequence Delay
24	ASC1_Seq_Pwr_Dn_node = 1		Enable ASC0 to sequence down
25	Halt (end-of-program)		

Note:

1. Greyed out Steps 0, 12, and 25 are default steps included in every sequence.

One main difference in the ASC0 sequence is that Step 2 in Table 3.2, does not wait for a previous ASC to complete its sequence. Instead, it only waits for A0_VM7_POT_OK because, ASC0 is the first L-ASC10 in the group.

Similarly, the ASC2 sequence does not wait for the next ASC to complete power down before starting the shutdown sequence. Instead, it waits for the ptm_shutdown_node in Step 12 in Table 3.2 . The ptm_shutdown_node is defined in Supervisory Equation EQ1 (see Figure 3.31); it is the OR of the three GPIO10 push button switches (on the three ASC Breakout boards) OR the input port ptm_shut_down_i that is connected to a RISC-V GPIO bit.

In like fashion, Step 24 in Table 3.2 for ASC0 does not set a node but rather the output port ptm_pwr_dn_done_o that is connected to a RISC-V bit.

In this demo design, the Voltage, Current, and Temperature (V,I,T) fault logging is enabled as shown in Figure 3.30. The user created node ptm_fault_trigger_node is used to trigger the storing of V,I,T and ASCx_GPIO status within the L-ASC10's SRAM. The trigger signal is sent over the three-wire ASC/IF. The fault record is later read back via I^2C.

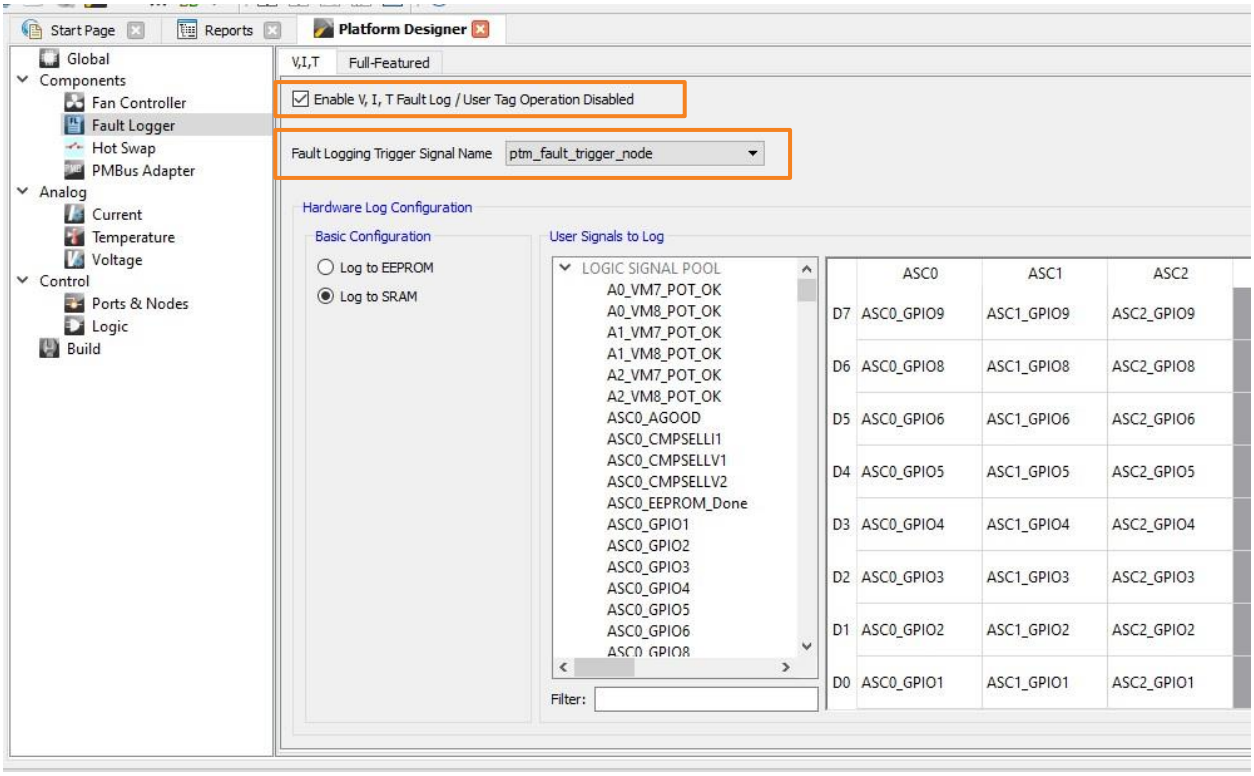

Figure 3.30. Platform Designer – V, I, T Fault Logging Enabled

The logic to detect and trigger the fault log is a combination of a supervisory equation and a sequence. The supervisory equation EQ2 is the OR of any POT not OK as shown in Figure 3.31. Since the fault_node will be true during the power up sequence, a separate Logic sequence is used to enable the ptm_fault_trigger_node as listed in Table 3.3.

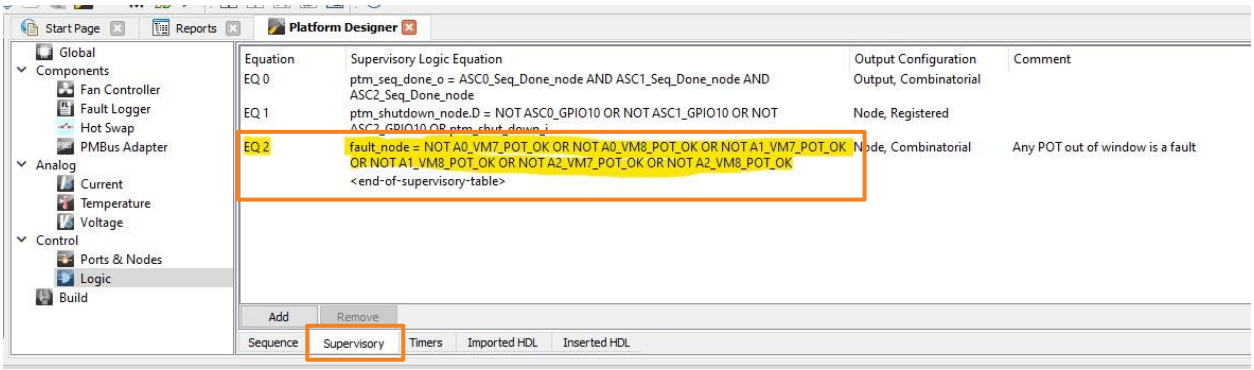

Figure 3.31. Platform Designer – Fault Node Supervisory Equation

Table 3.3. Platform Designer – SM3: Fault Logger Sequence

Step	Instruction	Outputs	Comment
0	Begin Startup Sequence		
1	Wait for ASC0_Seq_Done_node AND ASC1_Seq_Done_node AND ASC2_Seq_Done_node AND NOT ptm_clr_fault_i	ptm_fault_o = 0 ptm_fault_trigger_node = 0	Wait for all sequences to be Done and RISC-V ready to clear the next fault
2	Wait for fault_node		Wait for any POT to be out of its window = FAULT
3	ptm_fault_trigger_node = 1		Trigger the ASCs to capture Fault Status in Volatile Register
4	ptm_fault_o = 1 ptm_fault_trigger_node = 0		Tell RISC-V that fault occurred and clear the trigger signal to the ASCs
5	Wait for NOT fault_node		Wait for faults to go away
6	Wait for 50ms using Timer		
7	Wait for ptm_clr_fault_i		Wait for RISC-V to clear the fault
8	Go to step 1		Get ready for next fault
9	Begin Shutdown Sequence		
10	Halt (end-of-sequence)		

Note:

1. Greyed out Steps 0, 9, and 10 are default steps included in every sequence.

The Fault Logger sequence manages both the trigger to log a fault and the handshaking between the RISC-V firmware and the Platform Manager 2 design. So that the RISC-V firmware has time to read out the faults before re-enabling the fault trigger.

The last sequence is state machine 4 (SM4) and it simply drives the seven-segment LED display on the MachXO5-NX development board. If any of the VMON POTs are outside their window then an "**F**" will be displayed otherwise the display LEDs are off (see Table 3.4). This visual feedback helps when running the demo to restore the slide POT within the lower and upper VMON trip points. This also demonstrates how the Platform Manager 2 design can directly control the MachXO5-NX Top-Level Ports without involving the RISC-V.

Table 3.4. Platform Designer – SM4: Fault Display on 7-segment LED

Step	Instruction	Outputs	Comment
0	Begin Startup Sequence		
1	Wait for ASC0_Seq_Done_node AND ASC1_Seq_Done_node AND ASC2_Seq_Done_node		Wait for all sequences to be Done
2	If fault_node Then Goto 3 Else Goto 5		
3	Segment = 01110001		Display "F" on 7-sement LED
4	Goto step 2		
5	Segment = 11111111		7-segment LEDs Off
6	Go to step 2		Get ready for next fault
7	Begin Shutdown Sequence		
8	Halt (end-of-sequence)		

Note:

1. Greyed out Steps 0, 7, and 8 are default steps included in every sequence.

Throughout this discussion several Ports and Nodes have been mentioned. Table 3.5 provides a summary of the user defined Ports and Nodes in this demo design.

Table 3.5. Platform Designer – User Defined Ports and Nodes

Name	Direction	Description	RISC-V bit-map
segment_0	From PTM2	seg_dp_o	n/a
segment_1	From PTM2	seg_g_o	n/a
segment_2	From PTM2	seg_f_o	n/a
segment_3	From PTM2	seg_e_o	n/a
segment_4	From PTM2	seg_d_o	n/a
segment_5	From PTM2	seg_c_o	n/a
segment_6	From PTM2	seg_b_o	n/a
segment_7	From PTM2	seg_a_o	n/a
ptm_fault_o	From PTM2	Fault Flag	0x400
ptm_seq_done_o	From PTM2	Power Up sequence done	0x200
ptm_pwr_dn_done_o	From PTM2	Power Down sequence done	0x100
ptm_clr_fault_i	From RISC-V	Clear the Fault Flag	0x400
ptm_shut_down_i	From RISC-V	Start Shutdown sequence	0x200
fault_node	n/a	VMON POT = NOT OK	n/a
ptm_fault_trigger_node	n/a	Trigger signal for Fault Log	n/a
ptm_shutdown_node	n/a	OR of RISC-V signal or push buttons on ASC Breakout boards	n/a

The signal in the MachXO5-NX Top-Level Radiant design, **ptm_start_up_w,** has the RISC-V bit map of 0x100. It does not connect to the Platform Manager 2 Ports or Nodes. It connects to the **ptm_enable_seq** port (see Figure 3.36) of the Platform Designer exported module and ultimately to the external reset port of the clock and reset module. The **ptm_enable_seq** port is not a user defined port, it is always included by the export function. Thus, once this signal is set it should not be cleared otherwise, the ASC outputs will immediately assume their safe state and the **Powerdown** Sequence will not be followed.

The final step in generating the Platform Manager 2 IP is to click the **Compile** button in the **Build** view, as shown in Figure 3.32. This will generate the IP and export the files to the folder specified in Figure 3.28.

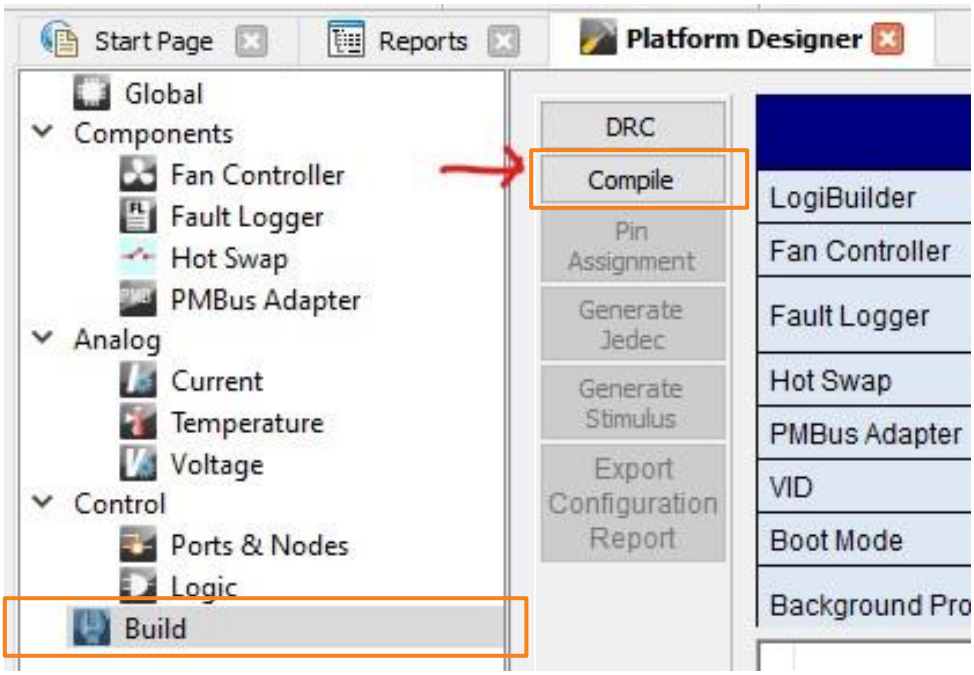

Figure 3.32. Platform Designer – Compiling the IP

3.9. Building the MachXO5-NX Top Level Design

When Step 3.7 Generating the SOC RTL is completed, the Propel Builder placed a top-level RTL module and all the IP RTL library files in a folder for the Radiant software project as shown in Figure 3.33.

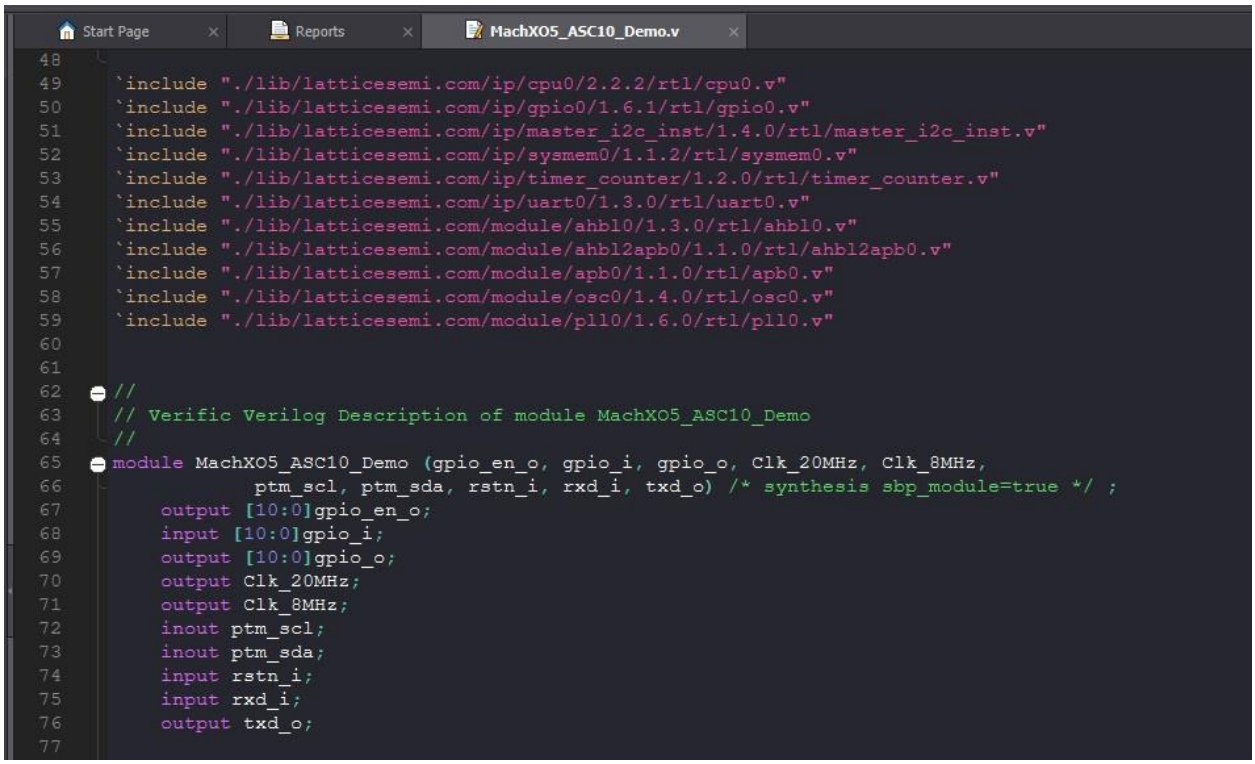

Figure 3.33. Radiant – MachXO5_ASC10_Demo Module

The whole SOC design has ten ports to connect to the outside world and to the Platform Manager 2 IP. To include the Platform Manager 2 IP do the following:

1. Right-click on the **Input Files** from the File List view, as shown in Figure 3.34, and select **Add -> Existing File**.

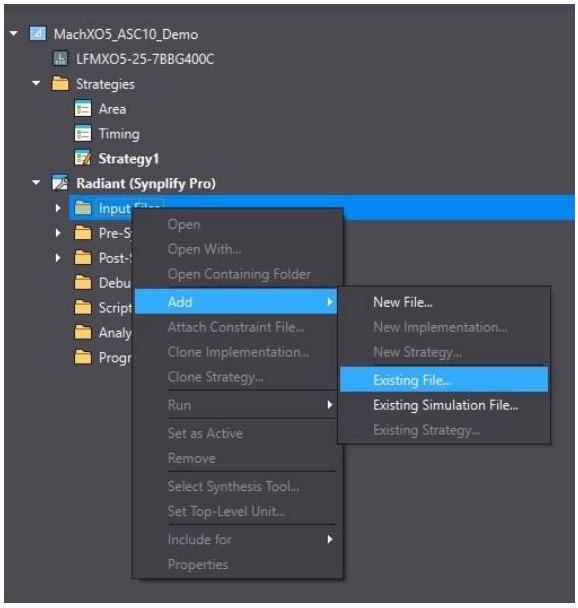

Figure 3.34. Radiant – Input Files, Add Existing File

2. Browse to the folder where the Platform Manager 2 IP was exported to and add the files listed below and shown in Figure 3.35.
    - psb_ASC10_PWR_SEQ_tmpl.v
    - psb_ASC10_PWR_SEQ.v
    - ASC10_PWR_SEQ_lgb.v
    - RDAT.v
    - WDAT.v
    - CLKRST/src/rtl/verilog/clkrst.v
    - CLKRST/src/rtl/verilog/clkrst_core.v
    - ASCVM/src/rtl/verilog/ASCVM.v
    - ASCVM/src/rtl/verilog/ASCVM_DATAPATH.v
    - ASCVM/src/rtl/verilog/ASCVM_NX_RAM_DIST.v
    - ASCVM/src/rtl/verilog/ASCVM_NX_RAM_EBR.v
3. Right-click on the **psb_ASC10_PWR_SEQ_tmpl.v** file and select **Exclude from Implementation** as this is only a template file; the contents of which can be copied and pasted into the Top-Level design file
4. To add the **MachXO5_ASC10_Demo_Top.v** file, right-click on the **Input Files** and select **Add -> New**. This top level module will be used to connect the Platform Manager 2 IP with the Propel Builder SOC IP and both of them to the physical pins of the MachXO5-NX device. This is the only RTL file that the user has to code by hand.

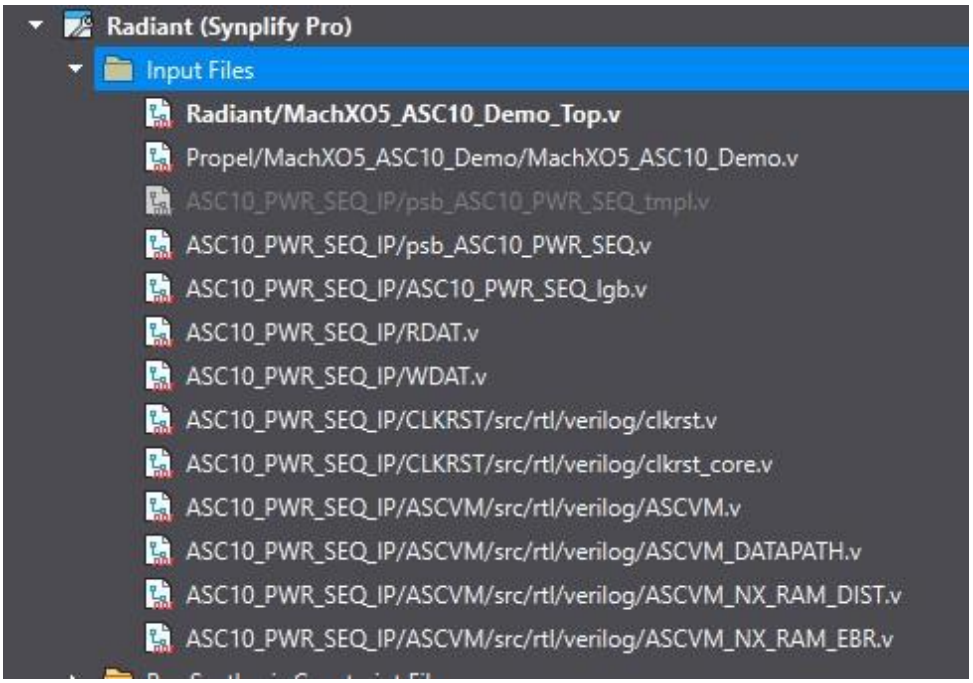

Figure 3.35. Radiant – Input File List

The Platform Manager 2 IP module **psb_ASC10_PWR_SEQ** is instantiated and connected in lines 159 to 196 of the Top-Level design file, as shown in Figure 3.36.

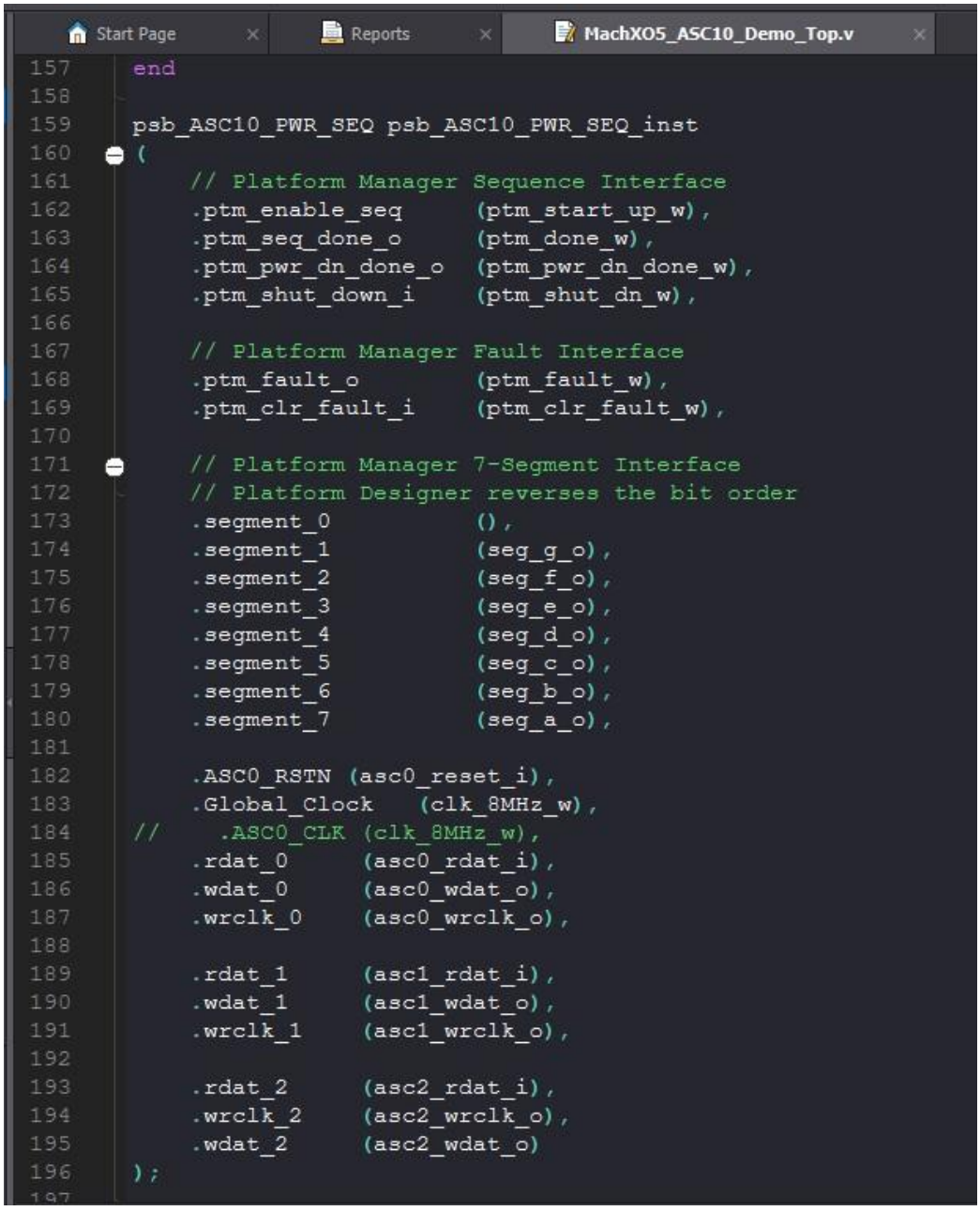

Figure 3.36. Radiant – Platform Manager 2 IP instantiation

The Propel Builder SOC IP module **MachXO5_ASC10_Demo** is instantiated and connected in lines 200 to 210 of the Top-Level design file, as shown in Figure 3.37.

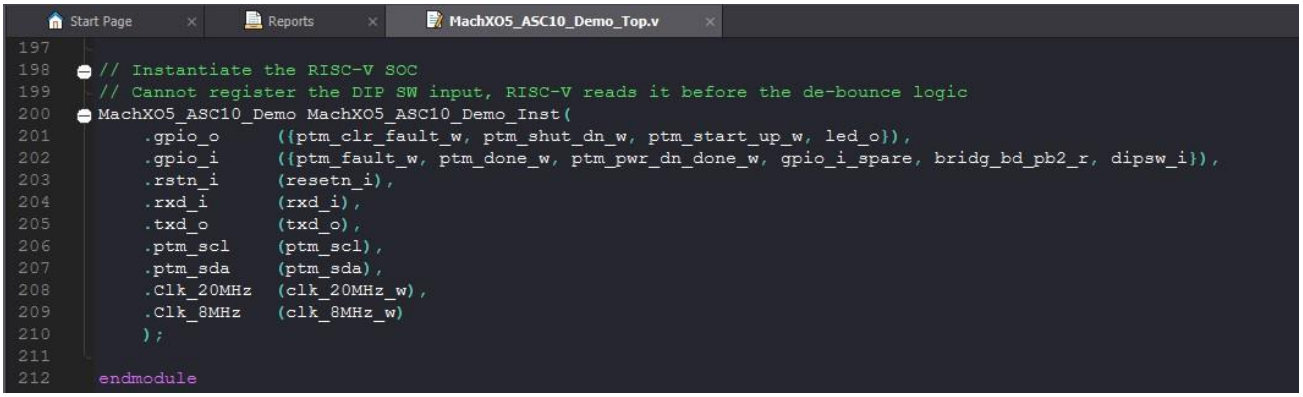

Figure 3.37. Radiant – SOC Instantiation

Between these two Figures, it can be seen how the two IPs are connected with the **ptm_xxx_w** wires. The placement of the wires in the SOC GPIO input and output ports determines the bit map in the RISC-V firmware.

The Top-Level ports (the MachXO5-NX physical connections) are between line 42 and line 87 of the MachXO5_ASC10_Demo_Top module as shown in Figure 3.38.

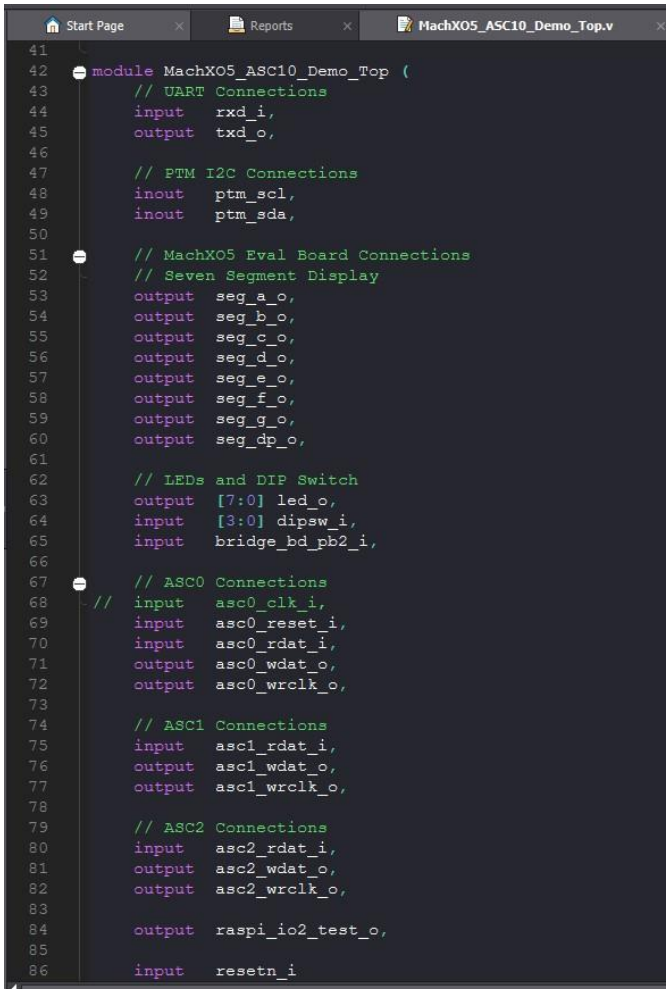

Figure 3.38. Radiant – Top Level Ports

Radiant's device **Constraint Editor** is used to map the Top-Level ports to the MachXO5-NX physical pins based on the schematics of the evaluation boards.

Now, the design is set up for Synthesis, Map, Place & Route, and Export a programming file. If either the SOC design in Propel Builder or the Platform Manager 2 design in Diamond are modified, then Radiant will know the source files have been updated and a re-build is necessary. If changes in the number of ports, names of ports, or the size of the ports changes, then the Top-Level instantiations will need to be modified. However, the RISC-V still needs some firmware before we have a working demo.

3.10. Starting the Propel SDK

In Step 3.7 Generating the SOC RTL, Propel Builder also generates system information files that are used in the Propel software development kit (SDK).

When Propel SDK is first launched for this demo a new workspace is created as shown in Figure 3.39.

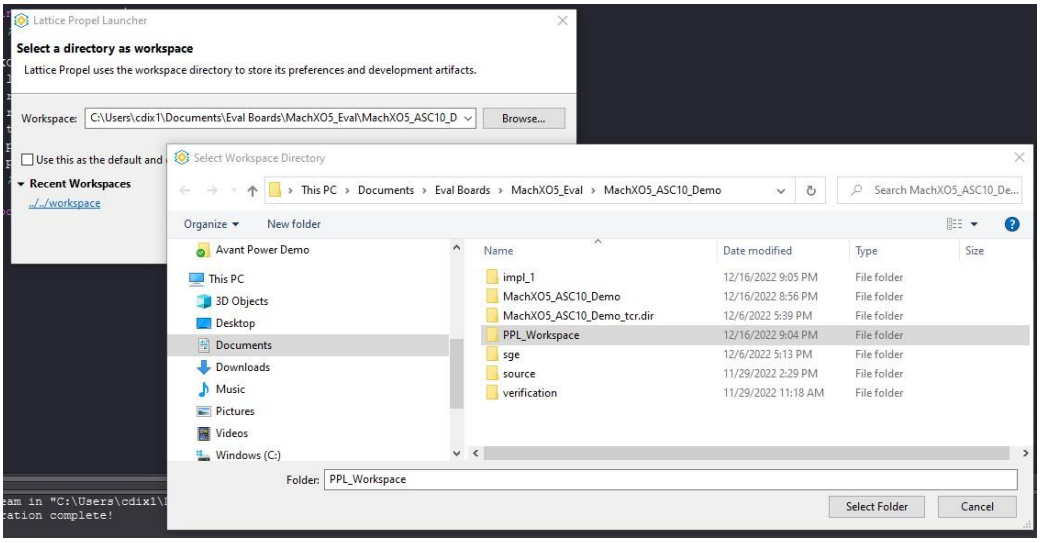

Figure 3.39. Propel SDK – New Project Workspace

1. Browse the Propel Builder generated file **sys_env.xml** as shown in Figure 3.40.

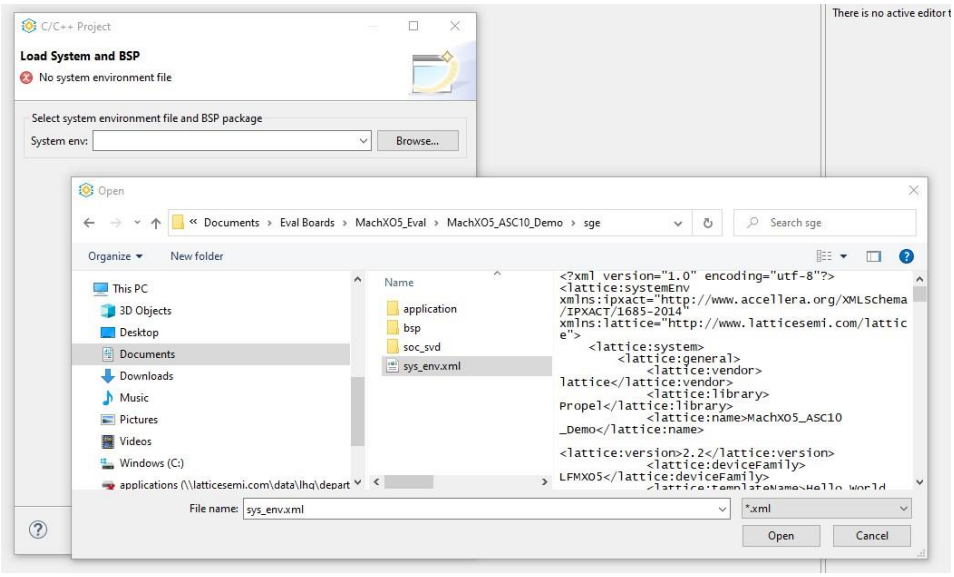

Figure 3.40. Propel SDK – New Project System and BSP

2. Select the **Project type** as **"C++"** as shown in Figure 3.41, and then provide a project name as shown in Figure 3.42.

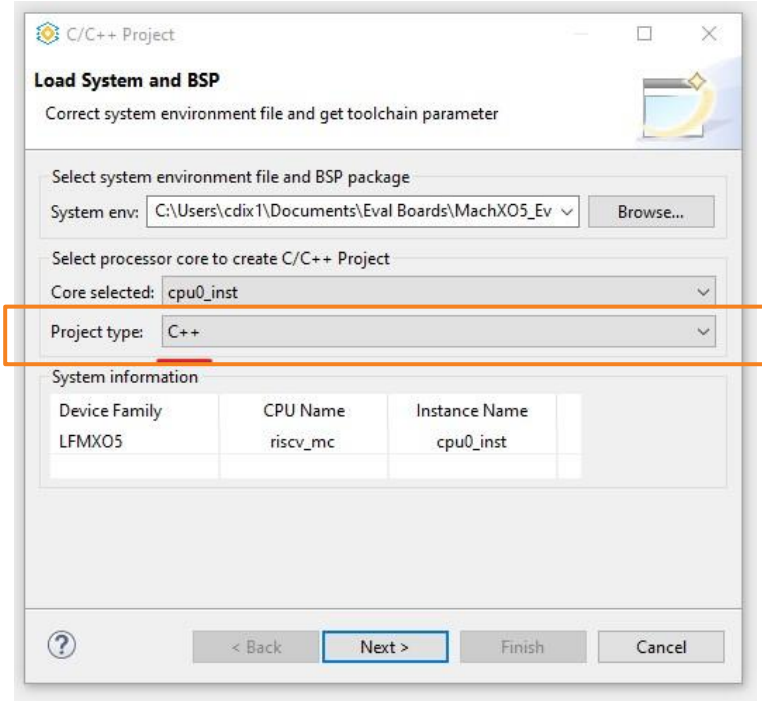

Figure 3.41. Propel SDK – New Project C++

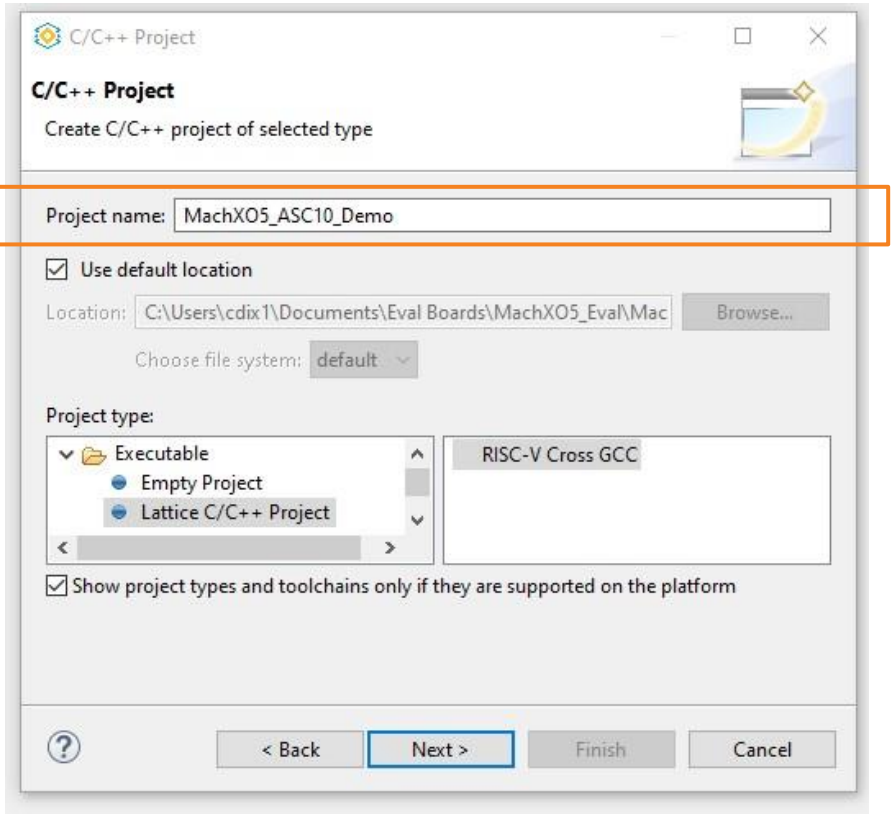

Figure 3.42. Propel SDK – New Project Name

3. Accept the default Toolchain Settings and click the **Finish** button as shown in Figure 3.43.

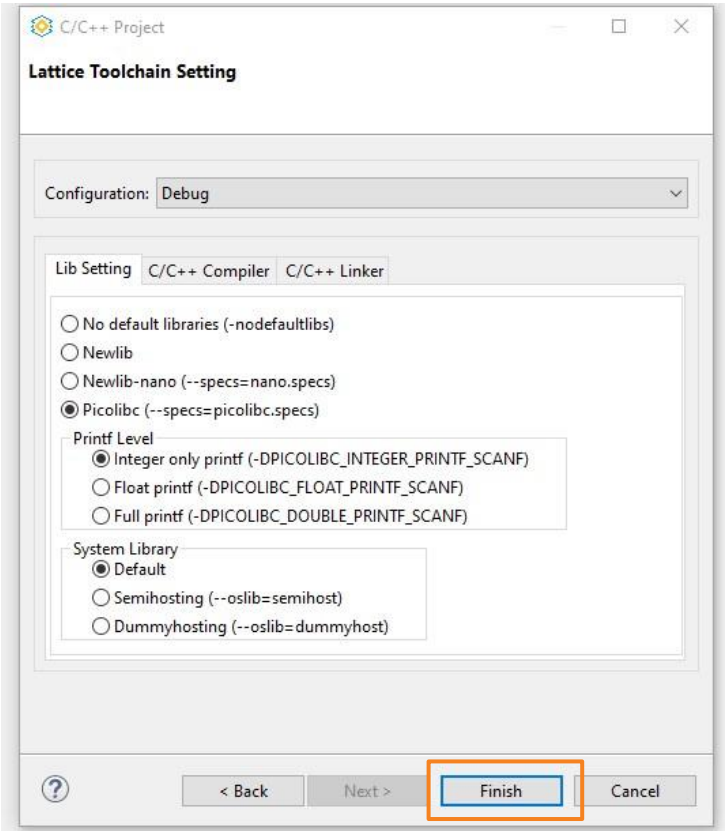

Figure 3.43. Propel SDK – New Project Tool Settings

4. The Propel SDK environment will be displayed with the **main.c** file open as shown in Figure 3.44.

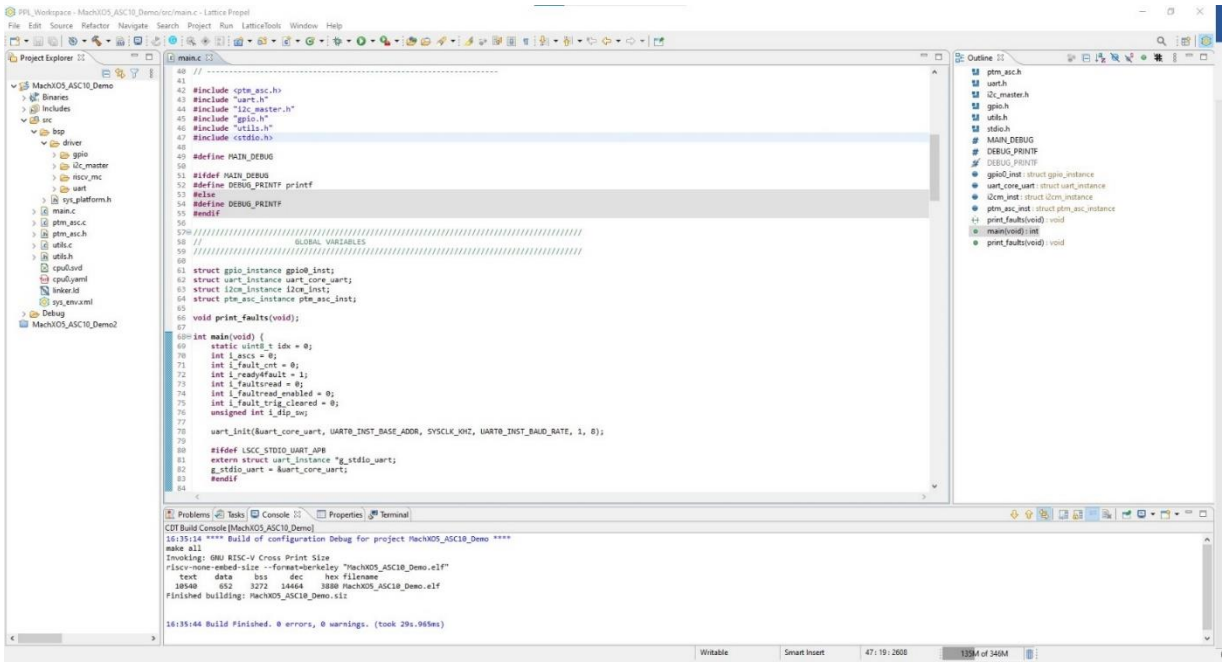

Figure 3.44. Propel SDK – IDE main.c File

3.11. Modifying main.c

Several edits to main.c were made for the demo design and are covered in this section.

1. Add global variable **ptm_asc_inst**:

   This structure is used to hold pointers and data to support interface with the L-ASC10s. Details of the structure are covered in the next step.

2. Add several local variables:

   Basic looping index, integers for number of ASCs and number of faults, and several flags to manage the Platform Manager 2 Fault detection.

3. DIP Switch:
   - **None**: The L-ASC10s use the settings within their EECMOS – no configuration is done from the RISC-V**.**
   - **SW1**: The L-ASC10s will be configured by the erase, program, and verify algorithm (similar to Programmer).
   - **SW2**: The L-ASC10s will be configured by writing to their SRAM and then verifying.

```
109     gpio_input_get(&gpio0_inst, 0, &i_dip_sw);
110     i_dip_sw &= 0x0F;
111     DEBUG_PRINTF("DIP SW = %d\n\r", i_dip_sw);
112
113     if((i_dip_sw & 0x01) == 0) {
114         DEBUG_PRINTF("Programming ASC10s!\r\n");
115         i_ascs = ptm_asc_program_eeprom_config(&ptm_asc_inst);
116         if(i_ascs < PTM_ASC_I2C_ERROR_NAK)
117             DEBUG_PRINTF("Total of %d ASC10s Programmed!\r\n", i_ascs);
118         else
119             switch(i_ascs) {
120             case(PTM_ASC_I2C_ERROR_NAK) :
121                 DEBUG_PRINTF("NAK Errors Ocurred - ASC10 Programming Failed!\n\r");
122             break;
123             case (PTM_ASC_ERROR_PRG_MODE) :
124                 DEBUG_PRINTF("Program Mode Errors Ocurred - ASC10 Programming Failed!\n\r");
125             break;
126             case (PTM_ASC_ERROR_VERIFY):
127                 DEBUG_PRINTF("Verify Errors Ocurred - ASC10 Programming Failed!\n\r");
128             break;
129             case (PTM_ASC_ERROR_ERASE):
130                 DEBUG_PRINTF("Erase Errors Ocurred - ASC10 Programming Failed!\n\r");
131             break;
132             default :
133                 DEBUG_PRINTF("Too many Errors Ocurred - ASC10 Programming Failed!\n\r");
134             break;
135             }
136     }
137
138     if((i_dip_sw & 0x02) == 0){
139         DEBUG_PRINTF("Configuring ASC10s!\r\n");
140         i_ascs = ptm_asc_write_sram_config(&ptm_asc_inst);
141         if(i_ascs < PTM_ASC_I2C_ERROR_NAK)
142             DEBUG_PRINTF("Total of %d ASC10s Configured!\r\n", i_ascs);
143         else
144             switch(i_ascs) {
145             case(PTM_ASC_I2C_ERROR_NAK) :
146                 DEBUG_PRINTF("NAK Errors Ocurred - ASC10 Config Failed!\n\r");
147             break;
148             case (PTM_ASC_ERROR_VERIFY):
149                 DEBUG_PRINTF("Verify Errors Ocurred - ASC10 Config Failed!\n\r");
150             break;
151             default :
152                 DEBUG_PRINTF("Too many Errors Ocurred - ASC10 Config Failed!\n\r");
153             break;
154             }
155     }
156
```

Figure 3.45. Propel SDK – Main DIP Switch Options

4. Platform Manager 2 Interface

   This is the primary goal of this demo design.

   One function call is made after decoding the DIP switch to enable the Platform Manager 2 sequence. Followed by a function to see if the Platform Manager 2 sequence is complete, as shown in Figure 3.46.

```
156
157     DEBUG_PRINTF("Starting ASC10 Power Up Sequence!\r\n");
158     ptm_asc_set_pwr_seq_start(&ptm_asc_inst);
159
160     while (ptm_asc_inst.i_seq_done_flag == 0) {
161         ptm_asc_get_pwr_up_seq_done(&ptm_asc_inst);
162     }
163     DEBUG_PRINTF("ASC10 Power Up Sequence Complete!\r\n");
164
```

Figure 3.46. Propel SDK – Main Platform Manager 2 Start

This is followed by a loop where several functions are called to interface with the Platform Manager 2 using the GPIO and I^{2}C objects as shown in Figure 3.47.

```
167     while (true) {
168         ptm_asc_get_pwr_dn_seq_done(&ptm_asc_inst);
169         if(ptm_asc_inst.i_seq_off_flag) {
170             DEBUG_PRINTF("ASC10 Power Down Sequnce Complete!\r\n");
171             return 0;
172         }
173         if(i_fault_cnt == 5){
174             DEBUG_PRINTF("Too Many Faults - Shutting Down!\r\n");
175             i_fault_cnt = 6;
176             ptm_asc_set_fault_trig(&ptm_asc_inst, PTM_ASC_FAULT_TRIG_ENABLE);
177             ptm_asc_set_pwr_seq_shutdown(&ptm_asc_inst);
178         }
179         ptm_asc_get_pwr_fault(&ptm_asc_inst);
180         if((ptm_asc_inst.i_seq_fault_flag) && (i_ready4fault)){
181             i_ready4fault = 0;
182             i_faultsread = 0;
183             i_fault_cnt++;
184             DEBUG_PRINTF("Total ASC10 Faults = %d\r\n", i_fault_cnt);
185         }
186         if(!(ptm_asc_inst.i_seq_fault_flag) && !(i_ready4fault)){
187             i_ready4fault = 1;
188             i_faultread_enabled = 0;
189             i_faultsread = 0;
190             i_fault_trig_cleared = 0;
191             ptm_asc_set_fault_trig(&ptm_asc_inst, PTM_ASC_FAULT_TRIG_ENABLE);
192             DEBUG_PRINTF("ASC10 Ready for Faults\r\n");
193         }
194         if((ptm_asc_inst.i_seq_fault_flag) && !(i_faultread_enabled)){
195             ptm_asc_get_fault_read_enabled(&ptm_asc_inst);
196 //          DEBUG_PRINTF("Enabling the Reading of ASC10 Faults\r\n");
197             i_faultread_enabled = 1;
198         }
199         if((ptm_asc_inst.i_seq_fault_flag) && !(i_faultsread)){
200             if(ptm_asc_inst.i_read_fault_en_flag){
201                 DEBUG_PRINTF("Reading ASC10 Faults  ");
202                 ptm_asc_get_faults(&ptm_asc_inst);
203                 print_faults();
204                 i_faultsread = 1;
205             }
206         }
207         // Check if PB2 on ASC Bridge Board is pressed
208         gpio_input_get(&gpio0_inst, 0, &i_dip_sw);
209         if((i_dip_sw & 0x10) && !(i_fault_trig_cleared)){
210             DEBUG_PRINTF("Clearing ASC10 Fault Trigger\r\n");
211             ptm_asc_set_fault_trig(&ptm_asc_inst, PTM_ASC_FAULT_TRIG_CLEAR);
212             i_fault_trig_cleared = 1;
213         }
```

Figure 3.47. Propel SDK – Main Loop

5. Platform Manager 2 Fault Decoder:

   This local function **print_faults()**, listed in Figure 3.48, decodes the fault record for the VMONs monitored in this design and sends a message over the UART to indicate which ASC had the fault and if the fault was an over-voltage or under-voltage. When running the demo, these messages correspond to actions taken on the VMON slider POTs.

```
void print_faults(void){
    if((ptm_asc_inst.asc[0].fault[4] & FAULT_MASK_VMON7A) == 0){
        if((ptm_asc_inst.asc[0].fault[4] & FAULT_MASK_VMON7B) == 0){
            DEBUG_PRINTF("ASC0 VMON7 Under Voltage\r\n");
        } else {
            DEBUG_PRINTF("ASC0 VMON7 Over Voltage\r\n");
        }
    }
    if((ptm_asc_inst.asc[0].fault[4] & FAULT_MASK_VMON8A) == 0){
        if((ptm_asc_inst.asc[0].fault[3] & FAULT_MASK_VMON8B) == 0){
            DEBUG_PRINTF("ASC0 VMON8 Under Voltage\r\n");
        } else {
            DEBUG_PRINTF("ASC0 VMON8 Over Voltage\r\n");
        }
    }
    if((ptm_asc_inst.asc[1].fault[4] & FAULT_MASK_VMON7A) == 0){
        if((ptm_asc_inst.asc[1].fault[4] & FAULT_MASK_VMON7B) == 0){
            DEBUG_PRINTF("ASC1 VMON7 Under Voltage\r\n");
        } else {
            DEBUG_PRINTF("ASC1 VMON7 Over Voltage\r\n");
        }
    }
    if((ptm_asc_inst.asc[1].fault[4] & FAULT_MASK_VMON8A) == 0){
        if((ptm_asc_inst.asc[1].fault[3] & FAULT_MASK_VMON8B) == 0){
            DEBUG_PRINTF("ASC1 VMON8 Under Voltage\r\n");
        } else {
            DEBUG_PRINTF("ASC1 VMON8 Over Voltage\r\n");
        }
    }
    if((ptm_asc_inst.asc[2].fault[4] & FAULT_MASK_VMON7A) == 0){
        if((ptm_asc_inst.asc[2].fault[4] & FAULT_MASK_VMON7B) == 0){
            DEBUG_PRINTF("ASC2 VMON7 Under Voltage\r\n");
        } else {
            DEBUG_PRINTF("ASC2 VMON7 Over Voltage\r\n");
        }
    }
    if((ptm_asc_inst.asc[2].fault[4] & FAULT_MASK_VMON8A) == 0){
        if((ptm_asc_inst.asc[2].fault[3] & FAULT_MASK_VMON8B) == 0){
            DEBUG_PRINTF("ASC2 VMON8 Under Voltage\r\n");
        } else {
            DEBUG_PRINTF("ASC2 VMON8 Over Voltage\r\n");
        }
    }
}
```

Figure 3.48. Propel SDK – Main Print Faults

3.12. Writing Code to Interface with Platform Manager 2

In this section we will highlight some of the code written to interface with the Platform Manager 2 IP. All of the code is contained in the header file (ptm_asc.h) and source file (ptm_asc.c). A complete description of all of the functions within the source file (ptm_asc.c) is provided in Appendix A.

The header file (ptm_asc.h) can be divided into three major sections;

- #defines
- Structures
- function prototypes.

Most of the #defines are derived directly from the L-ASC10 Data Sheet. A few of the #defines at the very top of the file, listed in Figure 3.49, represents the connection made in the Radiant design (see Figure 3.37 and Table 3.5).

```
45 // PTM Control Bits
46 #define GPIO_PTM_PWR_CLR_FAULT 10
47 #define GPIO_PTM_PWR_SHUTDOWN 9
48 #define GPIO_PTM_PWR_ENABLE 8
49
50 // PTM Control Masks
51 #define PTM_PWR_CLEAR_FAULT 0x400
52 #define PTM_PWR_SHUTDOWN 0x200
53 #define PTM_PWR_ENABLE 0x100
54
55 // PTM Status Bits
56 #define GPIO_PTM_PWR_FAULT 10
57 #define GPIO_PTM_PWR_DONE 9
58 #define GPIO_PTM_PWR_DN_DONE 8
59
60 // PTM Status Masks
61 #define PTM_PWR_FAULT 0x400
62 #define PTM_PWR_SEQ_DONE 0x200
63 #define PTM_PWR_DN_DONE 0x100
64
```

Figure 3.49. Propel SDK – ptm_asc.h, Defines PPL_IDE

The **ptm_asc_instance** structure, listed in Figure 3.50, holds pointers to both the GPIO and I^2C interfaces, various fault related flags, and an array of eight **asc_device** structures.

```
135 struct asc_device {
136     unsigned char * data;
137     unsigned char fault[7];
138 };
139
140 struct ptm_asc_instance {
141     const char *instance_name;
142     struct i2cm_instance * i2cm;
143     struct gpio_instance * gpio;
144     int i_total_asc;
145     int i_seq_done_flag;
146     int i_seq_fault_flag;
147     int i_seq_off_flag;
148     int i_read_fault_en_flag;
149     struct asc_device asc[8];
150 };
151
152 enum ptm_asc_fault_trigger {
153     PTM_ASC_FAULT_TRIG_CLEAR,
154     PTM_ASC_FAULT_TRIG_ENABLE
155 };
```

Figure 3.50. Propel SDK – ptm_asc.h, Structures

The **asc_device** structure holds a pointer to the **configuration data** for the respective L-ASC10 device and the **fault** array contains the full fault record for the same device. The **data pointer** is set when the **ptm_asc_init** function is called and the fault array is populated by the **ptm_asc_get_faults** function (both are described in the API section).

The source file (ptm_asc.c) contains commands, listed in Figure 3.51, that are based on the L-ASC10 Data Sheet and Programming Specification.

```
// ASC I2C Commands
unsigned char ptm_asc_cmd_read_id[] = { 0x02};
unsigned char ptm_asc_cmd_read_status[] = {0x03};
unsigned char ptm_asc_cmd_write_config[] = { 0x31, 0x00};
unsigned char ptm_asc_cmd_read_config[] = { 0x33, 0x00};
unsigned char ptm_asc_cmd_load_config[] = {0x35};

unsigned char ptm_asc_cmd_read_fault_enable[] = {0x74, 0xAC};
unsigned char ptm_asc_cmd_read_fault_disable[] = {0x74, 0x00};  //
unsigned char ptm_asc_cmd_read_volatile_fault_reg[] = {0x73};
unsigned char ptm_asc_cmd_read_user_tag_cfg_reg[] = {0x33, 0x66};
unsigned char ptm_asc_cmd_enprog[] = {0x04, 0x3D, 0xE5};
unsigned char ptm_asc_cmd_usermode[] = {0x05};
unsigned char ptm_asc_cmd_becfg[] = {0x21};
unsigned char ptm_asc_cmd_writecfg[] = {0x22};
unsigned char ptm_asc_cmd_progcfg[] = {0x24, 0x00};
unsigned char ptm_asc_cmd_vercfg[] = {0x25, 0x00};
unsigned char ptm_asc_cmd_progdone[] = {0x27};
unsigned char ptm_asc_cmd_ldshdw[] = {0x28};
```

Figure 3.51. Propel SDK – ptm_asc.c, I²C Commands

The configuration data arrays are #included in the source file (ptm_asc.c), as listed in Figure 3.52, instead of having to copy-and-paste every time the L-ASC10 configuration changes. Also, the configuration header files for each L-ASC10 is added to the list of Project Includes.

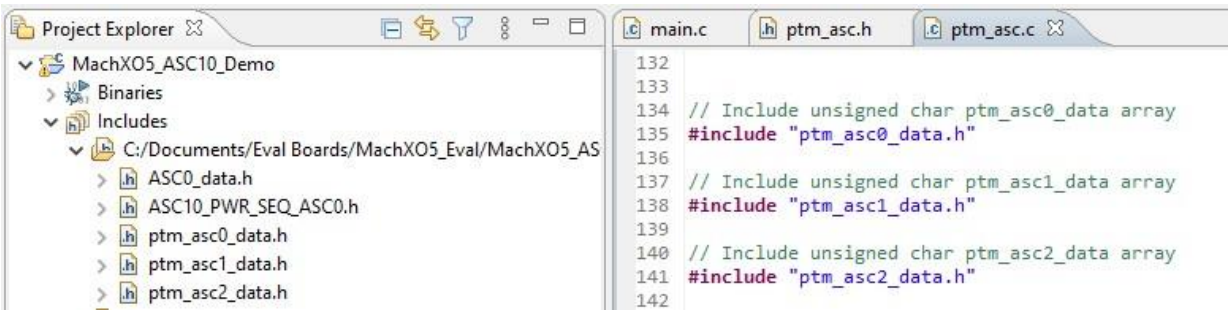

Figure 3.52. Propel SDK – ptm_asc.c, Configuration includes

3.13. Converting the L-ASC10 Programming Files

The L-ASC10 configuration header files (ASC0 is shown in Figure 3.53) are generated from the ASC_Hex_Reader.exe utility that converts the .hex programming file into the unsigned char array that can be #included into the design.

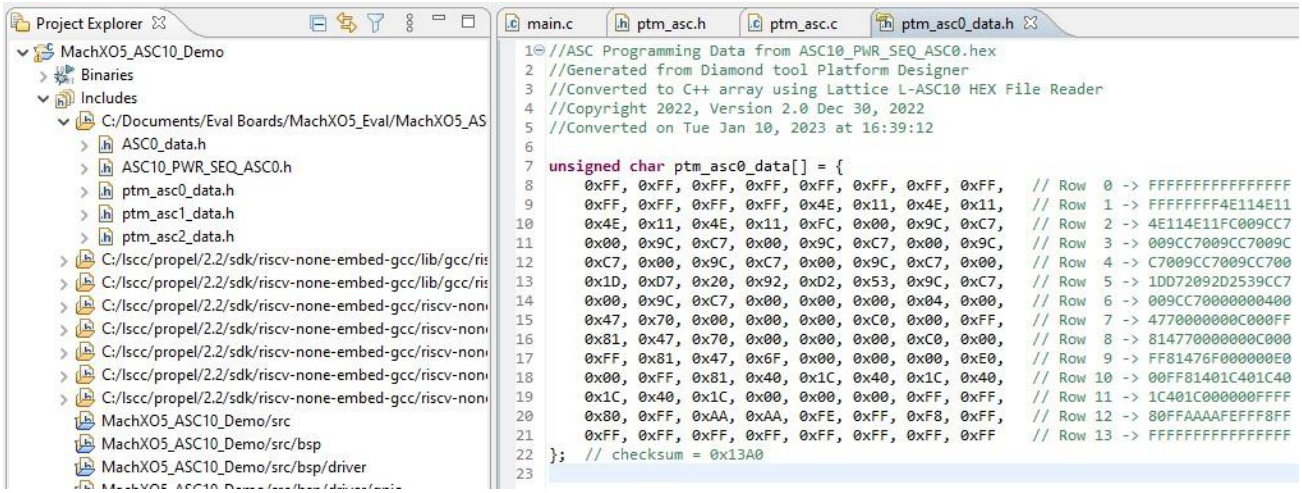

Figure 3.53. Propel SDK – ptm_asc0_data.h

The ASC_Hex_Reader.exe utility is copied into the L_ASC10 subfolder, where Platform Designer exports the ASC .hex files. By running the utility in the same folder, the command line does not have to include the full path, as shown in Figure 3.53. For this demo design, the “up” arrow was used with the “left” arrow to edit the “0” to “1” and later to “2” followed by the “enter” key to generate the header files for ASC1 and ASC2. This reduced the amount of typing. This utility could be ported into a graphical user interface, in the future, to simplify this step.

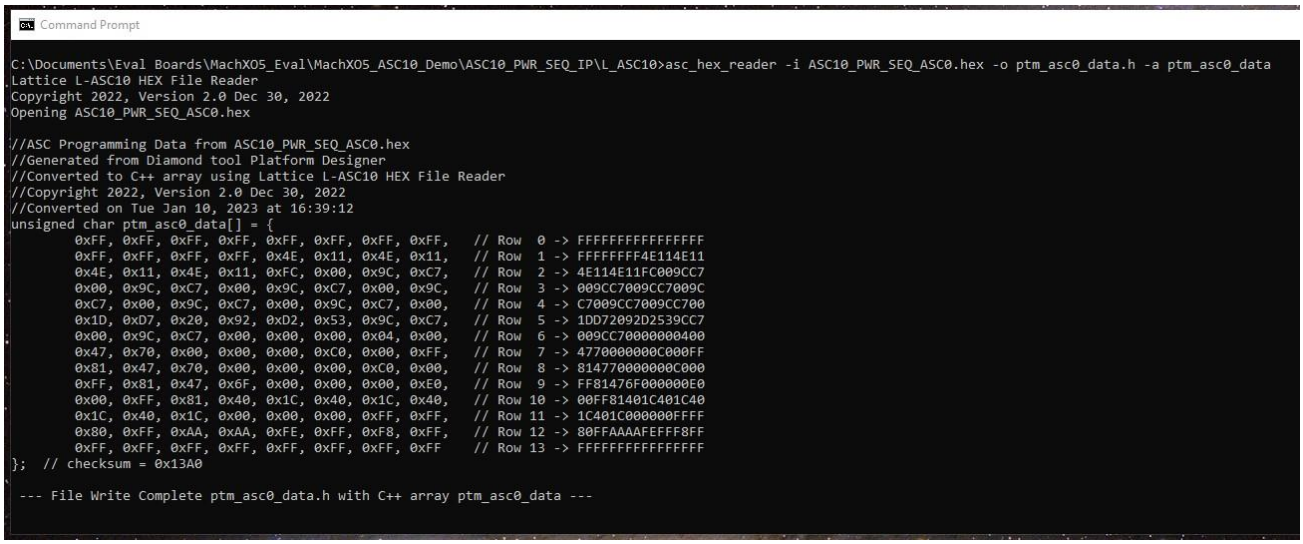

Figure 3.54. Command Prompt – ASC_Hex_Reader

3.14. Loading the Firmware into Memory

After the Propel SDK tool is used to successfully compile the firmware, we need to go back to Propel Builder and initialize the system memory with the firmware.

1. Double-click on the **system0_inst** to open the memory IP configuration dialog, shown in Figure 3.55.
2. Tick the **Initialize Memory** checkbox and browse to the **Initialization File** MachXO5_ASC10_Demo.mem
3. Click the **Generate** button.
4. Click the **Generate** tool to update the SOC IP RTL.
5. Return to Radiant to build the entire project so that it can be programmed into the evaluation board.

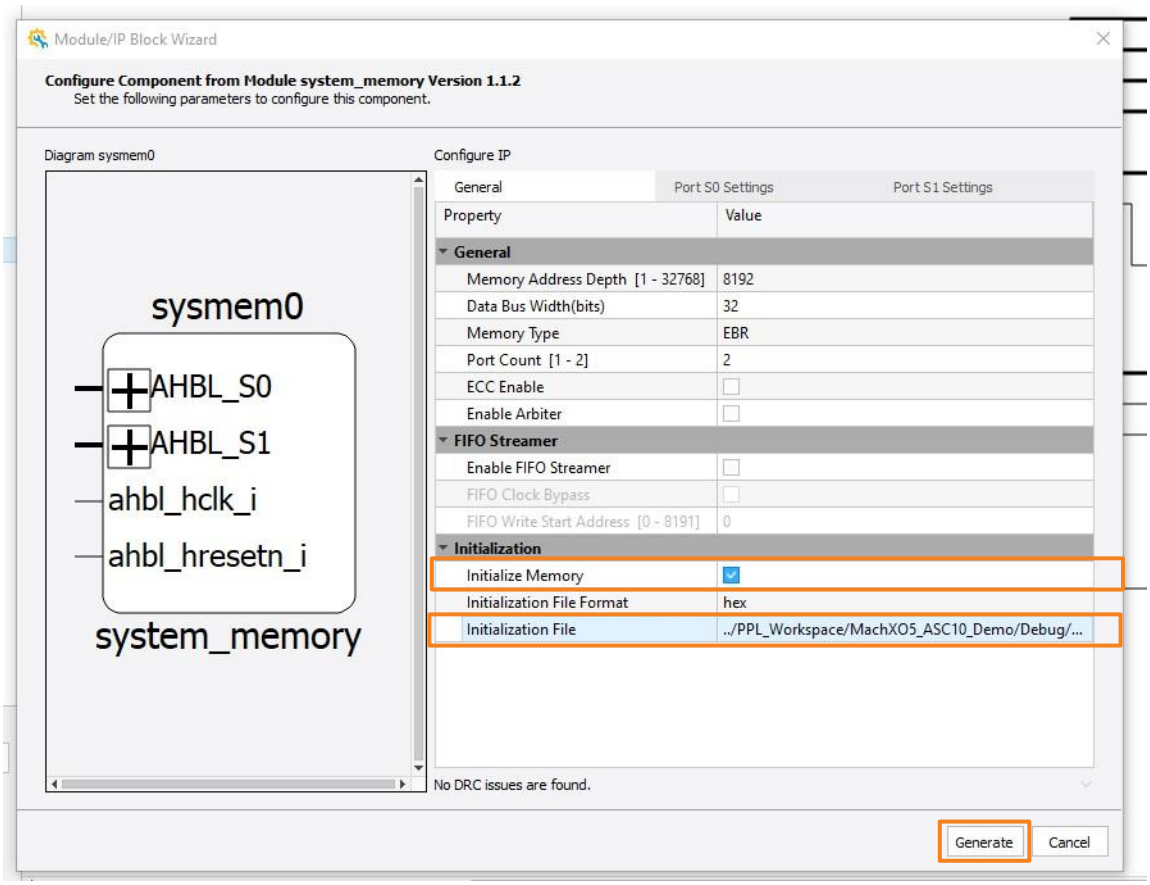

Figure 3.55. Propel Builder – System Memory Initialization

There is a shortcut in Radiant for updating the firmware without rebuilding all the SOC and Platform Manager 2 IP. This is useful when developing or modifying the firmware.

1. Click on the ECO Editor tool icon, shown in Figure 3.56, or click **Tools -> ECO Editor** to open the tool.

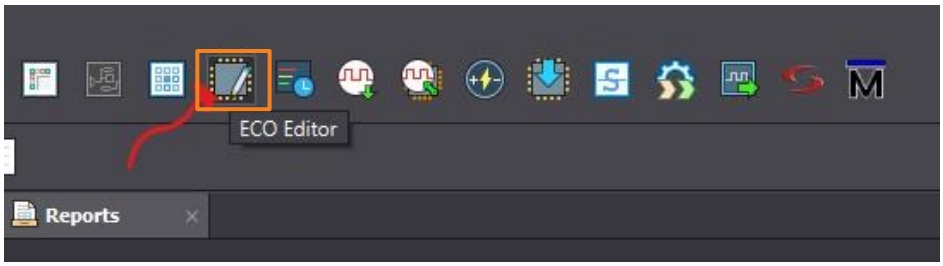

Figure 3.56. Radiant ECO Editor – Icon

2. When the tool opens, click on the **Memory Initialization** tab, shown in Figure 3.57, near the bottom of the screen.

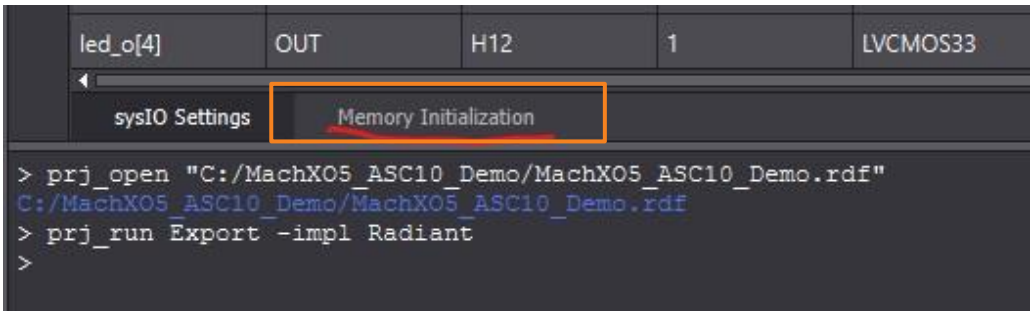

Figure 3.57. Radiant ECO Editor – Memory Initialization Tab

3. Double-click the **Choose File** as shown in Figure 3.58, and the **Memory Initialization Settings** window will open, shown in Figure 3.59.

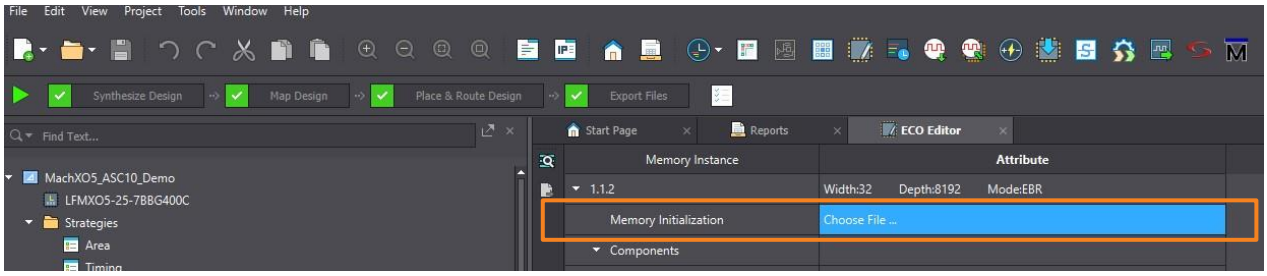

Figure 3.58. Radiant ECO Editor – Choose File

4. In the Settings window, select **Hexadecimal** for the **File Format** and browse to the **MachXO5_ASC10_Demo.mem** file, then click **OK**.

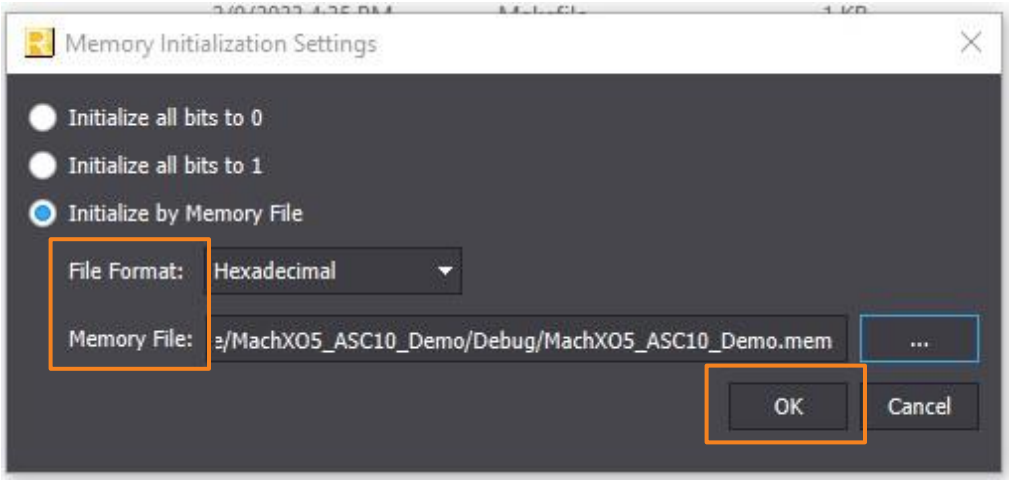

Figure 3.59. Radiant ECO Editor – Settings

5. After clicking the **OK** button, double-click on the **Export Files** and the **Save** dialog box will open as shown in Figure 3.60. Click the **Save** button and Radiant will include the latest firmware into the programming file without having to repeat the synthesis, map, and P&R processes as shown in Figure 3.61.

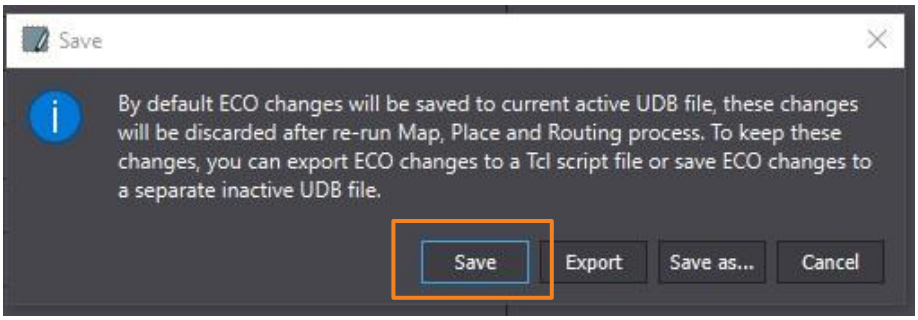

Figure 3.60. Radiant ECO Editor – Save

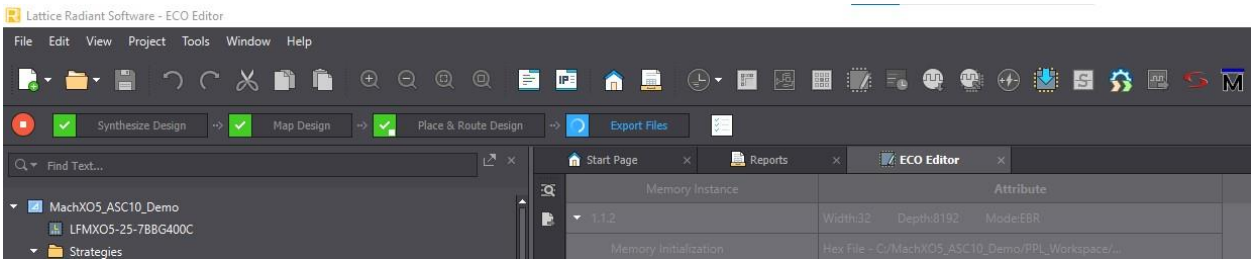

Figure 3.61. Radiant ECO Editor – Exporting Files

6. With the ECO Editor tool left open, the shortcut gets even shorter by using the Tcl Console to rerun the command. Use the "**up**" arrow (on the keyboard) to bring the command to the cursor and press the "**Enter**" key, as shown in Figure 3.62. Repeat the **Save** and **Export Files** operations and a new programming file is ready for testing.

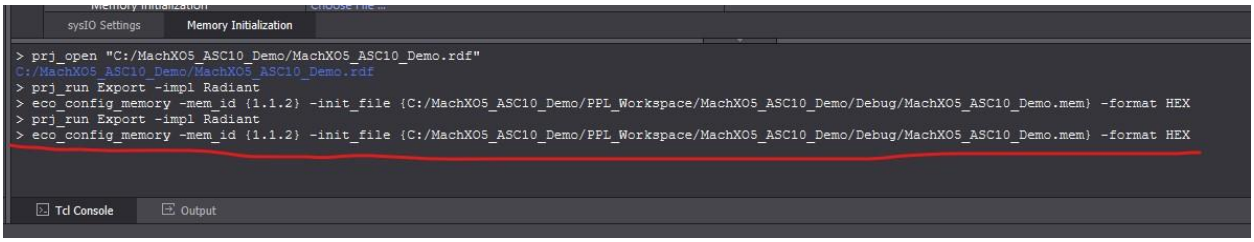

Figure 3.62. Radiant ECO Editor – TCL Command

3.15. Running the Demo

Depending on the setting of the DIP switch (SW1) on the MachXO5-NX Development board, the demo will start by configuring the three L-ASC10s or skip right to enabling the Platform Manager 2 IP.

In any case, the demo should start with the VMON slider POTs on the L-ASC10 Breakout boards all the way down (zero volts nearest the ASC Bridge board).

1. The Platform Manager 2 IP will turn on LED2 on the ASC0 board and wait for the VMON7 slider POT voltage to be in the window (see Table 3.1).
2. Gradually move VMON7 POT up until LED3 and LED4 turn on. Gradually slide VMON8 POT up until LED5 and LED6 turn on. Repeat this process for ASC1 and ASC2.
3. When all the VMON slider POTs are in position and the LEDs on all three ASC Breakout boards are on, then the Terminal will display "ASC10 Power Up Sequence Complete!" as shown in Figure 3.63.

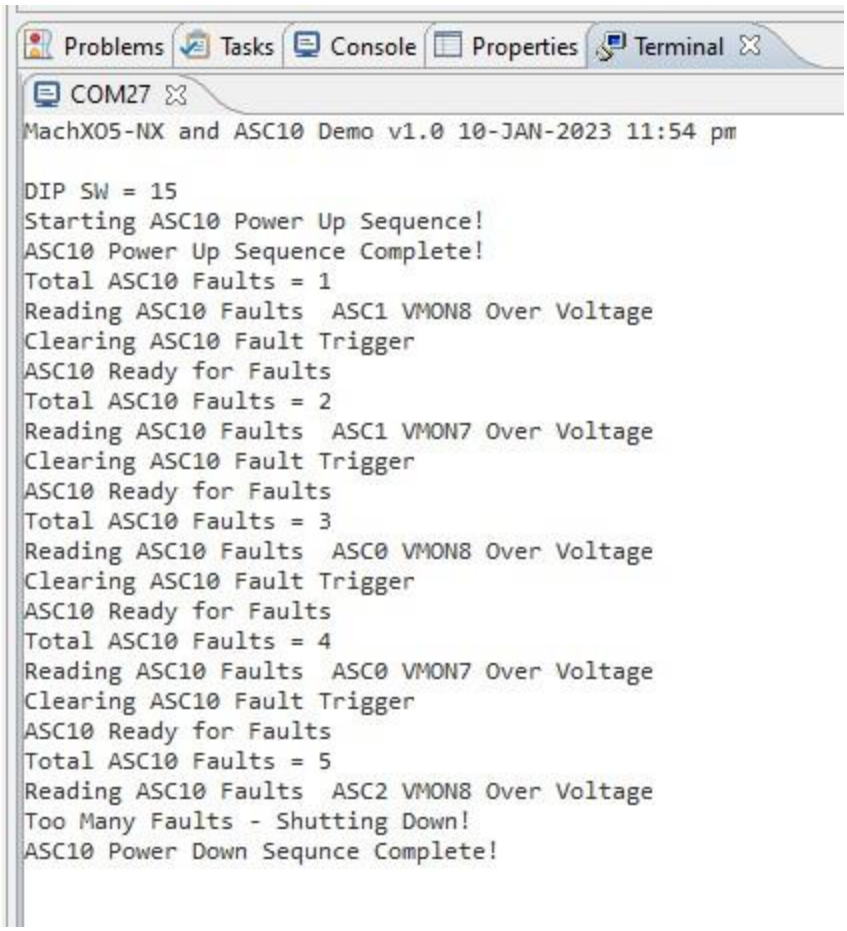

Figure 3.63. Propel SDK Terminal – Demo Running Power Sequence

4. Now the system will wait for either a press of a GPIO10 push button to sequence the power down or an accumulation of five faults to initiate a power down sequence.
5. To generate a fault, any VMON slider POT can be moved above the upper trip point value or below the lower trip point value. The 7-segment display on the MachXO5-NX board will display the letter 'F' to indicate that a VMON input is outside the window values. The terminal will display a fault count followed by a description of the fault, as shown in Figure 3.63.
6. Restore the VMON slider POT to within the lower and upper trip point values and the 7-segment display will turn off. Press push button SW2 on the ASC Bridge board and the terminal will display the "Clearing ASC10 Fault Trigger" and "ASC10 Ready for Faults" messages, as shown in Figure 3.63.
7. When four more faults have been generated (and cleared) the demo will send the message to the terminal that too many faults have occurred and the system is shutting down. At the same time all the LEDs on the ASC Breakout boards will sequence off.

The flexibility of this demo design is that any VMON can be over or under voltage in any order to generate a total of five faults.

Figure 3.64 displays the terminal results when the DIP switch is set to configure the L-ASC10 SRAM before enabling the power sequence and Figure 3.65 displays the terminal results when the DIP switch is set to Erase / Program / Verify the EECMOS of the L-ASC10s before enabling the power sequence.

Figure 3.64. Propel SDK Terminal – Demo Running Configure and Sequence

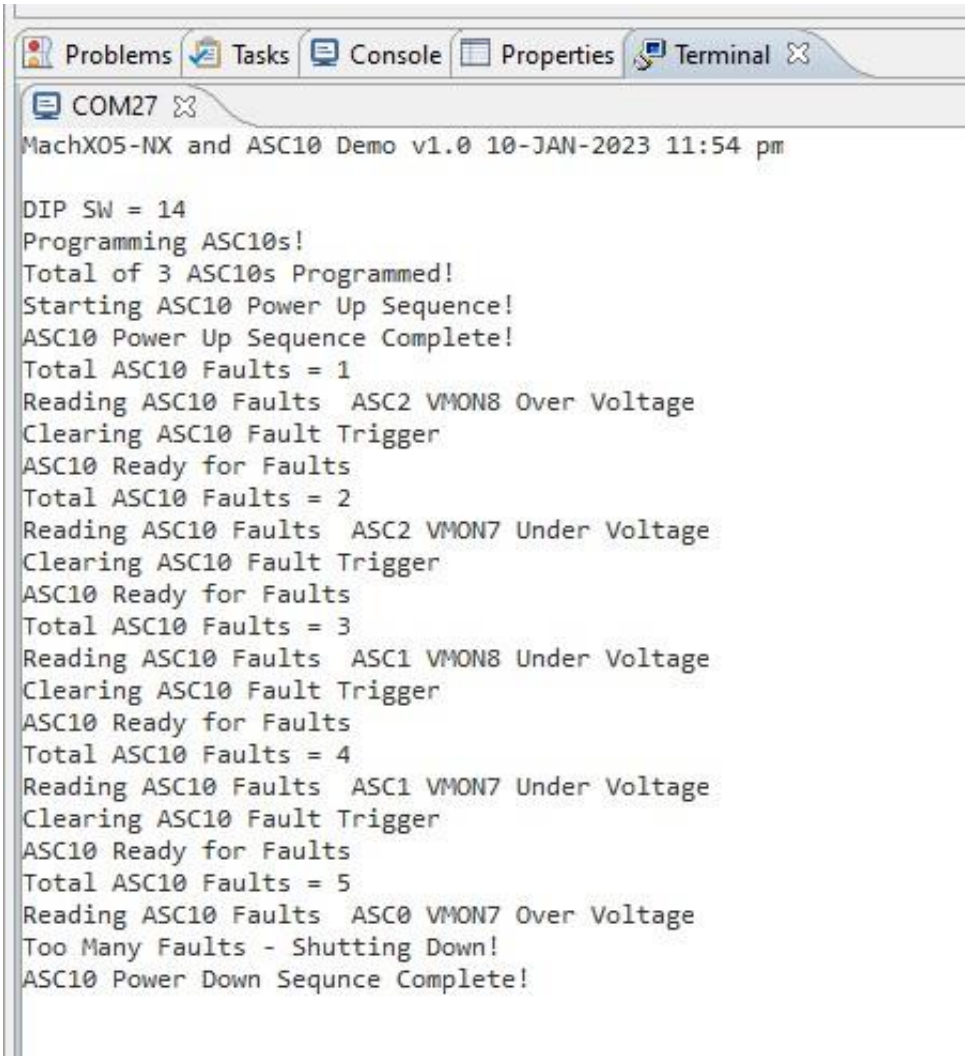

Figure 3.65. Propel SDK Terminal – Demo Running Program and Sequence

4. Summary

This demo design illustrates how L-ASC10 devices can be integrated into a MachXO5-NX design. It shows how the RISC-V SOC can interface with power sequencing and respond to faults. This User Guide provides a reference to the multiple software tools involved in merging the L-ASC10 with the MachXO5-NX, including Diamond's Platform Designer, Radiant, ASC_HEX_Reader.exe, Propel Builder, and Propel SDK.

By following the design steps outlined in this demo, one could easily add power sequencing and fault logging to Lattice's Sentry Solution Stack. Such systems provide a NIST SP8000-193-compliant Platform Firmware Resiliency (PFR) Root of Trust solution for authenticated and secure updates with a centralized platform management.

This demo design only used 27 % of the total logic available in the MachXO5-25 device and the distribution of resources is summarized in Table A.1.

Table A.1. MachXO5-NX and L-ASC10 Demo Resource Utilization

Module	LUT4	PFU Reg	IO Reg	IO Buff	DSP Mult	EBR
Total	**6,298**	**4,111**	**8**	**38**	**6**	**24**
psb_ASC10_PWR_SEQ_inst	585	463	3	0	0	6
uart0_inst	262	146	1	0	0	0
timer_counter_inst	938	728	0	0	0	0
system0_inst	52	29	0	0	0	16
pll0_inst	0	0	0	0	0	0
osc0_inst	0	0	0	0	0	0
master_i2c_inst1	997	604	0	2	0	0
gpio0_inst	147	117	4	0	0	0
cpu0_inst	2,935	1,752	0	0	6	2
apb0_inst	63	5	0	0	0	0
ahbl2apb0_inst	191	236	0	0	0	0
ahbl0_inst	98	6	0	0	0	0

Appendix A

Firmware APIs

All the L-ASC10 supporting APIs are contained in the ptm_asc.c source file. The following documentation is divided between the external functions, those called from main(), and internal functions, those that are shared by several external functions.

External Firmware APIs

These functions can be called from main() to initialize, configure, program, and read the L-ASC10 using an I²C Master. They also support the GPIO interface between the RISC-V firmware and the Platform Manager 2 IP. All the external functions start with the prefix **ptm_asc_**.

ptm_asc_init

```
uint8_t ptm_asc_init(struct ptm_asc_instance * ptr_asc, struct i2cm_instance * i2cm, struct gpio_instance * gpio, int i_total_asc);
```

Description:

This function initializes the ptm_asc_instance structure with pointers to the GPIO and I²C structures to support the other APIs. It also sets the total number of L-ASC10s in the system and sets the pointers to the configuration data for each L-ASC10. This function also resets the following status flags:

- **i_sequence_done_flag** – Set when the Platform Manager 2 IP power up sequence is complete.
- **i_seq_fault_flag** – Set when Platform Manager 2 IP detects a fault.
- **i_seq_off_flag** – Set when the Platform Manager 2 IP power down sequence is complete.
- **i_read_fault_en_flag** – Set when the Platform Manager 2 IP has enabled the reading of faults from the L-ASC10s.

Inputs:

- **ptr_asc** : pointer to the ptm_asc_instance structure that is being initialized.
- **i2cm** : pointer to the i²c_instance structure that has already been initialized.
- **gpio** : pointer to the gpio_instance structure that has already been initialized.
- **i_total_asc** : integer that sets the number of L-ASC10 devices in the system.

Outputs:

- Returns a value of 1 if there was an error and a value of 0 upon success.

ptm_asc_write_sram_config

```
int ptm_asc_write_sram_config(struct ptm_asc_instance * ptr_asc);
```

Description:

This function writes the L-ASC10 configuration data into the respective L-ASC10s in the system. It also reads the data back to verify the write operation.

Input:

- ptr_asc : pointer to the ptm_asc_instance structure that has pointer to the I²C interface.

Outputs:

- Upon success returns the number of L-ASC10s configured, otherwise returns an error code.

ptm_asc_program_eeprom_config

```
int ptm_asc_program_eeprom_config(struct ptm_asc_instance * ptr_asc);
```

Description:

This function performs the erase, program and verify of the L-ASC10's non-volatile memory.

Input:

- ptr_asc : pointer to the ptm_asc_instance structure that has pointer to the I^2C interface.

Outputs:

- Upon success returns the number of L-ASC10s configured, otherwise returns an error code.

ptm_asc_set_pwr_seq_start

```
void ptm_asc_set_pwr_seq_start(struct ptm_asc_instance * ptr_asc);
```

Description:

This function sets the GPIO signal that releases the reset to the Platform Manager 2 IP to start the power up sequence and logic.

Input:

- ptr_asc : pointer to the ptm_asc_instance structure that has pointer to the GPIO interface.

Output:

- None.

ptm_asc_get_pwr_up_seq_done

```
void ptm_asc_get_pwr_up_seq_done(struct ptm_asc_instance * ptr_asc);
```

Description:

This function reads the GPIO signal from the Platform Manager 2 IP to check if the power up sequence is complete or not and sets the i_seq_done_flag accordingly.

Input:

- ptr_asc : pointer to the ptm_asc_instance structure that has pointer to the GPIO interface.

Output:

- The i_seq_done_flag of the ptm_asc _instance structure is set or cleared based on the GPIO signal.

ptm_asc_set_pwr_seq_shutdown

```
void ptm_asc_set_pwr_seq_shutdown(struct ptm_asc_instance * ptr_asc);
```

Description:

This function sets the GPIO signal that triggers the Platform Manager 2 IP to sequence down the power.

Input:

- ptr_asc : pointer to the ptm_asc_instance structure that has pointer to the GPIO interface.

Output:

- None.

ptm_asc_get_pwr_dn_seq_done

```
void ptm_asc_get_pwr_dn_seq_done(struct ptm_asc_instance * ptr_asc);
```

Description:

This function reads the GPIO signal from the Platform Manager 2 IP to check if the power down sequence is complete or not and sets the i_seq_off_flag accordingly.

Input:

- ptr_asc : pointer to the ptm_asc_instance structure that has pointer to the GPIO interface.

Output:

- The i_seq_off_flag of the ptm_asc_instance structure is set or cleared based on the GPIO signal.

ptm_asc_get_pwr_fault

```
void ptm_asc_get_pwr_fault(struct ptm_asc_instance * ptr_asc);
```

Description:

This function reads the GPIO signal from the Platform Manager 2 IP to check if a power fault has occurred or not and sets the i_seq_fault_flag accordingly.

Input:

- ptr_asc : pointer to the ptm_asc_instance structure that has pointer to the GPIO interface.

Output:

- The i_seq_fault_flag of the ptm_asc_instance structure is set or cleared based on the GPIO signal.

ptm_asc_get_fault_read_enabled

```
void ptm_asc_get_fault_read_enabled(struct ptm_asc_instance * ptr_asc);
```

Description:

This function reads the GPIO signal from the Platform Manager 2 IP to check if reading faults from the L-ASC10s has been enabled or not and sets the i_seq_read_fault_en_flag accordingly.

Input:

- ptr_asc : pointer to the ptm_asc_instance structure that has pointer to the GPIO interface.

Output:

- The i_seq_read_fault_en_flag of the ptm_asc_instance structure is set or cleared based on the GPIO signal.

ptm_asc_get_faults

```
void ptm_asc_get_faults(struct ptm_asc_instance * ptr_asc);
```

Description:

Reads the volatile fault registers from all L-ASC10s in the system, using I^2C, and stores the results in the ptm_asc_instance structure.

Input:

- ptr_asc : pointer to the ptm_asc_instance structure that has the pointer to the I^2C interface and the asc_device structure to store the fault data.

Output:

- The seven bytes of the fault data are stored in the asc_device part of the ptm_asc_instance structure.

ptm_asc_set_fault_trig

```
void ptm_asc_set_fault_trig(struct ptm_asc_instance * ptr_asc, enum ptm_asc_fault_trigger
i_trig);
```

Description:

This function is used to handshake between the Platform Manager 2 IP and also enable fault logging within the L-ASC10 devices in the system.

Input:

- ptr_asc : pointer to the ptm_asc_instance structure that has the pointer to the I^2C and GPIO interfaces.
- i_trig : enumeration
  - PTM_ASC_FAULT_TRIG_CLEAR = Sets the GPIO signal that tells the Platform Manager 2 IP that the fault has been read and it can re-arm the fault detector.
  - PTM_ASC_FAULT_TRIG_ENABLE = Clears the GPIO signal so the Platform Manager 2 IP will wait for the RISC-V firmware to read the next fault and clear the trigger before re-arming the fault detector. Also sends the I^2C command to all the L-ASC10 devices in the system to re-enable fault capture.

Output:

- None

Internal Firmware APIs

These are support functions that modularize redundant routines to reduce the size and complexity of the external firmware APIs. All the internal functions start with the prefix ptm_.

ptm_fill_tx_buff

```
void ptm_fill_tx_buff(unsigned char * asc_inst, int i_inst_sz, unsigned char * asc_data,
int i_byt_cnt);
```

Description:

Fills the I^2C transmission buffer with an instruction and a set number of bytes in preparation of an I^2C write.

Inputs:

- asc_inst : pointer to the L-ASC10 command or instruction.
- i_inst_sz : number of bytes in the L-ASC10 command or instruction.
- asc_data : pointer to the data to be copied into the transmission buffer.
- i_byt_cnt : number of data bytes to copy into the transmission buffer.

Outputs:

- i2c_tx_buff: global unsigned char array holds instruction and data.

ptm_verify_asc

```
int ptm_verify_asc(unsigned char * a_buff, unsigned char * b_buff);
```

Description:

Verifies the L-ASC10 SRAM configuration data using a mask for each byte. The Diamond software tool Platform Designer creates .hex files with some of the unused bits set but, readback from the hardware has the unused bits cleared (or visa-versa) – so, the mask is used to prevent false fail indication.

Inputs:

- a_buff : pointer to one set of data.
- b_buff : pointer to other set of data.

Output:

- Returns a value of zero if the two data sets match (using the mask array), otherwise returns a verify error code.

ptm_read_asc_id

```
unsigned char ptm_read_asc_id(struct ptm_asc_instance * ptr_asc, unsigned char i_asc_adx);
```

Description:

Reads the L-ASC10 ID code.

Inputs:

- ptr_asc : pointer to the ptm_asc_instance structure that has pointer to the I^2C interface
- i_asc_adx : I^2C address of the L-ASC10 whose ID is being read.

Output:

- Returns 0x88 for valid L-ASC10s otherwise returns 0x00.

ptm_read_asc_sram_cfg

```
int ptm_read_asc_sram_cfg(struct ptm_asc_instance * ptr_asc, unsigned char i_asc_adx,
unsigned char * asc_data);
```

Description:

Reads the SRAM configuration from the specified L-AS10 (skipping register address 0x61).

Inputs:

- ptr_asc: pointer to the ptm_asc_instance structure that has pointer to the I^2C interface.
- i_asc_adx: I^2C address of the L-ASC10 whose SRAM is being read.
- asc_data: pointer to the array to hold the configuration data read from the specified L-ASC10.

Outputs:

- Returns a value of zero upon success otherwise returns an error code.

ptm_write_asc_sram_cfg

```
int ptm_write_asc_sram_cfg(struct ptm_asc_instance * ptr_asc, unsigned char i_asc_adx,
unsigned char * asc_data);
```

Description:

Writes the SRAM configuration to the specified L-AS10 (skipping register address 0x61).

Inputs:

- ptr_asc: pointer to the ptm_asc_instance structure that has pointer to the I^2C interface.
- i_asc_adx: I^2C address of the target L-ASC10.
- asc_data: pointer to the data to write into the L-ASC10s SRAM

Outputs:

- Returns a value of zero upon success otherwise returns an error code.

ptm_print_asc_cfg_data

```
void ptm_print_asc_cfg_data(unsigned char * asc_data);
```

Description:

A debug function to display the configuration data over the UART on the terminal.

Inputs:

- asc_data: pointer to an array filled with the L-ASC10 configuration data.

Outputs:

- None

References

For more information, refer to:

- Lattice Insights for Lattice Semiconductor training courses and learning plans
- Lattice Sales Office
- MachXO5-NX Family Data Sheet
- L-ASC10 Data Sheet
- MachXO5-NX Development Board User Guide
- ASC Bridge Board User Guide
- ASC Breakout Board User Guide
- Platform Designer User Guide
- L-ASC10 Breakout Board
- MachXO5-NX Web page
- Platform Manager 2 & L-ASC10 Web page
- Development Kits & Boards for MachXO5-NX
- Development Kits & Boards for Platform Manager 2 & L-ASC10
- IP & Reference Designs for MachXO5-NX
- Lattice Radiant FPGA design software
- Lattice Propel FPGA design software
- Lattice Diamond FPGA design software

Technical Support Assistance

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

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

Revision History

Revision 1.0, October 2023

Section	Change Summary
All	Initial release

www.latticesemi.com