Processor-Based Controller for Avant Device Target SPI x8 Configuration

Reference Design

FPGA-RD-02320-1.0

August 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
AHB	Advanced High-Performance Bus
AHB-Lite	Advanced High-Performance Bus-Lite
AMBA	Advanced Microcontroller Bus Architecture
APB	Advanced Peripheral Bus
AS2	Alpha Silicon 2
ASCII	American Standard Code for Information Interchange
AXI4	Advanced eXtensible Interface Version 4
BSE	Bitstream Engine
BSP	Board Support Package
CPU	Central Processing Unit
CRC32	Cyclic Redundancy Check 32-bit
DDR	Double Data Rate
DRAM	Dynamic Random-Access Memory
EBR	Embedded Block Random-Access Memory
FPGA	Field Programmable Gate Array
FTDI	Future Technology Devices International
GPIO	General Purpose Input/Output
GUI	Graphical User Interface
HDL	Hardware Description Language
I2C	Inter-Integrated Circuit
IP	Intellectual Property
LPDDR4	Low Power Double Data Rate 4
LSCC	Lattice Semiconductor Corporation
LUT	Look-Up Table
OSC	Oscillator
OSPI	Octal Serial Peripheral Interface
PLL	Phase-Locked Loop
RISC-V	Reduced Instruction Set Computing V
RTL	Register Transfer Level
SDK	Software Development Kit
SPI	Serial Peripheral Interface
SRAM	Static Random-Access Memory
SSPI	Target Serial Peripheral Interface
TCM	Tightly-Coupled Memory
UART	Universal Asynchronous Receiver/Transmitter
USB	Universal Serial Bus
xSPI	eXpanded Serial Peripheral Interface

1. Introduction

The Lattice Avant™ device features a dedicated target serial peripheral interface (SPI) configuration port that enables access to configuration logic functions. This port supports reconfiguring the configuration SRAM and accessing various status and control registers within the configuration logic block. The Avant device target SPI port supports x1 (serial), x2 (dual), x4 (quad), and xSPI (x8 DDR) modes of operation. To configure the FPGA, a host controller is required to read the configuration bitstream from a storage device and transmit it to the Avant device target SPI port.

This reference design demonstrates a RISC-V-based host controller that configures the Avant FPGA using the target SPI port in x8 DDR mode. During the target SPI x8 configuration process, the bitstream is first read from an SPI flash device and loaded into DDR memory. It is then transferred byte-by-byte through the host controller general purpose input/output (GPIO) pins to the Avant device target SPI port. Additionally, this reference design includes C source code that demonstrates the complete process of configuring the Avant device through the target SPI port.

Note: In this document, target SPI x8 configuration mode or target SPI x8 mode refers to the mode for configuring a device through the target SPI port in xSPI (x8 DDR) mode. Additionally, the C source code provided with the reference design that demonstrates the complete process of configuring the Avant device through the target SPI port is referred to as the octal SPI (OSPI) host source code.

1.1. Quick Facts

Download the reference design files from the Lattice Processor-Based Controller for Avant Device Target SPI x8 Configuration Reference Design web page.

Table 1.1. Summary of the Reference Design

General	Target Devices	Host controller – LFCPNX-100 Target – LAV-AT-X70
	Source code format	Verilog, C
Simulation	Functional simulation	Not supported
	Timing simulation	Not performed
	Test bench	Not available
	Test bench format	Not available
Software Requirements	Software tool and version	Lattice Propel™ Design Environment 2025.1 Lattice Radiant™ Software 2025.1 Python 3.11.x or later
	IP version (if applicable)	OSC v1.4.0 PLL v1.9.1 RISC-V RX CPU v2.6.0 AXI4 Interconnect v2.2.0 AXI4 to APB Bridge v1.4.0 APB Interconnect v1.3.0 AXI4 to AHB-Lite Bridge v1.4.0 LPDDR4 Memory Controller for Nexus Devices v2.6.0 GPIO v1.8.0 Tightly-Coupled Memory v 1.5.2 SPI Flash Memory Controller v2.0.0
Hardware Requirements	Board	Host controller – CertusPro-NX Versa Board Target – Avant-X Versa Board
	Cable	USB to micro-USB cable

1.2. Features

Key features of the Processor-Based Controller for Avant Device Target SPI x8 Configuration reference design include:

- Enables Avant device target SPI x8 configuration mode – The host controller can configure the Avant device in target SPI x8 configuration mode.

1.3. Naming Conventions

1.3.1. Nomenclature

The nomenclature used in this document is based on Verilog HDL.

1.3.2. Signal Names

Signal names that end with:

- _*n* are active low signals (asserted when value is logic 0)
- _*i* are input signals
- _*o* are output signals

2. Directory Structure and Files

Figure 2.1 shows the directory structure.

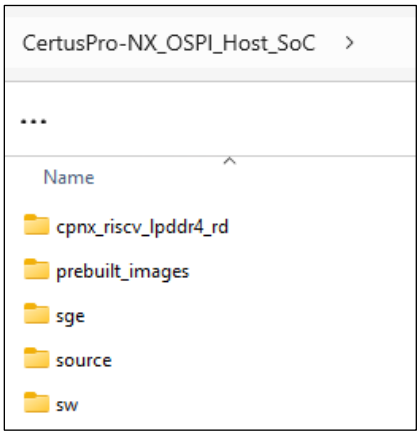

Figure 2.1. Directory Structure

The Processor-Based Controller for Avant Device Target SPI x8 Configuration reference design package includes the hardware design files, software project, and prebuilt images.
Table 2.1 shows the list of directories included in the reference design package.

Table 2.1. Directory List

Directory	Description
cpnx_riscv_lpddr4_rd	Contains the Lattice Propel Builder project and generated HDL files for IPs.
prebuilt_images	Contains the bitstream file, software image file, and Lattice Radiant Programmer scripts.
sge	Contains the Propel Builder project software handoff files.
source	Contains the Radiant hardware design project top wrapper and post-synthesis constraint file.
sw	Contains the software project and source code used in this reference design.

3. Functional Description

3.1. Design Block Diagram

The reference design demonstrates a complete functional RISC-V processor system created using Propel Builder. All components used in the system are available, instantiated, and connected in Propel Builder, which also generates the corresponding RTL design files.

Figure 3.1 shows the top-level block diagram and connections of the reference design.

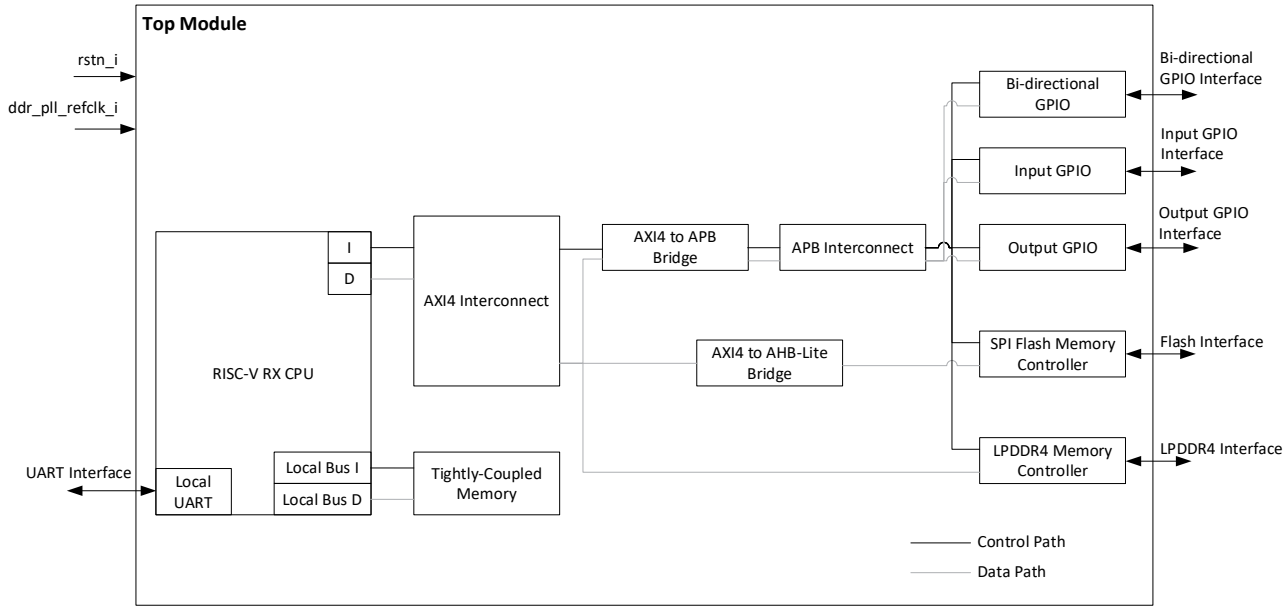

Figure 3.1. Reference Design Block Diagram

3.2. Design Components

The Processor-Based Controller for Avant Device Target SPI x8 Configuration reference design includes the following components:

- RISC-V RX CPU (with local UART)
- LPDDR4 memory controller for Nexus devices
- SPI flash memory controller
- Tightly-coupled memory
- GPIO
  - Bi-directional I/O
  - Input
  - Output
- AXI4 interconnect
- AXI4 to APB bridge
- AXI4 to AHB-Lite bridge
- APB interconnect

Each component in the block diagram is instantiated using its corresponding IP in Propel Builder.

3.2.1. RISC-V RX CPU

The RISC-V RX CPU IP includes four separate ports: local instruction and data ports for accessing the tightly-coupled memory, and AXI4-based instruction and data ports for interfacing with external memory and peripherals. The local ports are used for high-speed access to internal memory containing software code and immediate data. The AXI4 ports enable communication with external resources. A local UART interface is also enabled to support serial communication.

3.2.2. LPDDR4 Memory Controller for Nexus Devices

The LPDDR4 Memory Controller IP enables access to external LPDDR4 memory modules. The memory can be used to store CPU software code and data.

3.2.3. SPI Flash Memory Controller

The SPI Flash Memory Controller IP is used to read and load the bitstream file (converted to .bin) from SPI flash memory into LPDDR4 memory. The bitstream is then accessed from the LPDDR4 memory when performing the target SPI x8 configuration process.

3.2.4. Tightly-Coupled Memory

The Tightly-Coupled Memory IP is used to store the initial software code that the RISC-V RX CPU executes out from reset. Using the Tightly-Coupled Memory IP allows the initial software code to be pre-compiled into the FPGA bitstream file. Once the device configuration process is complete, the tightly-coupled memory contains the valid initial software code.

3.2.5. GPIO

GPIO pins are used to emulate SPI communication through bit-banging, enabling manual control of the SPI signals through software. This method is used to support the target SPI x8 configuration flow in the absence of a dedicated SPI controller.

The following are the GPIO pin connections:

- Bi-directional I/O: Connected to target SPI signals SDQ0 to SDQ7.
- Input: Connected to Avant device target SPI signal SDS and sysCONFIG pins INITN and DONE.
- Output: Connected to target SPI signals SCSN and SCLKP.

3.2.6. AXI4 Interconnect

The AXI4 Interconnect IP is used to connect AXI4 and AXI4-Lite components. It supports data width conversion, clock domain crossing, and allows multiple managers to access multiple memory-mapped subordinates. In this design, the RISC-V RX CPU has a single AXI4 interface, which branches out through the interconnect to multiple peripherals and devices.

3.2.7. AXI4 to APB Bridge

The AXI4 to APB Bridge IP is used to connect the AXI4 manager to APB completer. Read and write transfers on the AXI4 bus are converted into corresponding transfers on the APB.

3.2.8. AXI4 to AHB-Lite Bridge

The AXI4 to AHB-Lite Bridge IP is used for interfacing one AXI4 manager to one AHB-Lite subordinate. The read and write transfers on the AXI4 bus are converted into equivalent transfers on the AHB-Lite.

3.2.9. APB Interconnect

The APB Interconnect IP is used to connect one or more controllers to one or more targets in AMBA 3 APB-based systems. It is optimized for low power and simple interfaces, making it ideal for low-bandwidth peripherals like UART and GPIO. The APB protocol is unpipelined, meaning each transfer must complete before the next begins. All transfers use the full width of the data bus, which must match the processor width, for example, 32 bits.

3.3. Clocking Scheme

3.3.1. Clocking Overview

Figure 3.2 shows the clocking scheme of the reference design. Table 3.1 lists the clock signals.

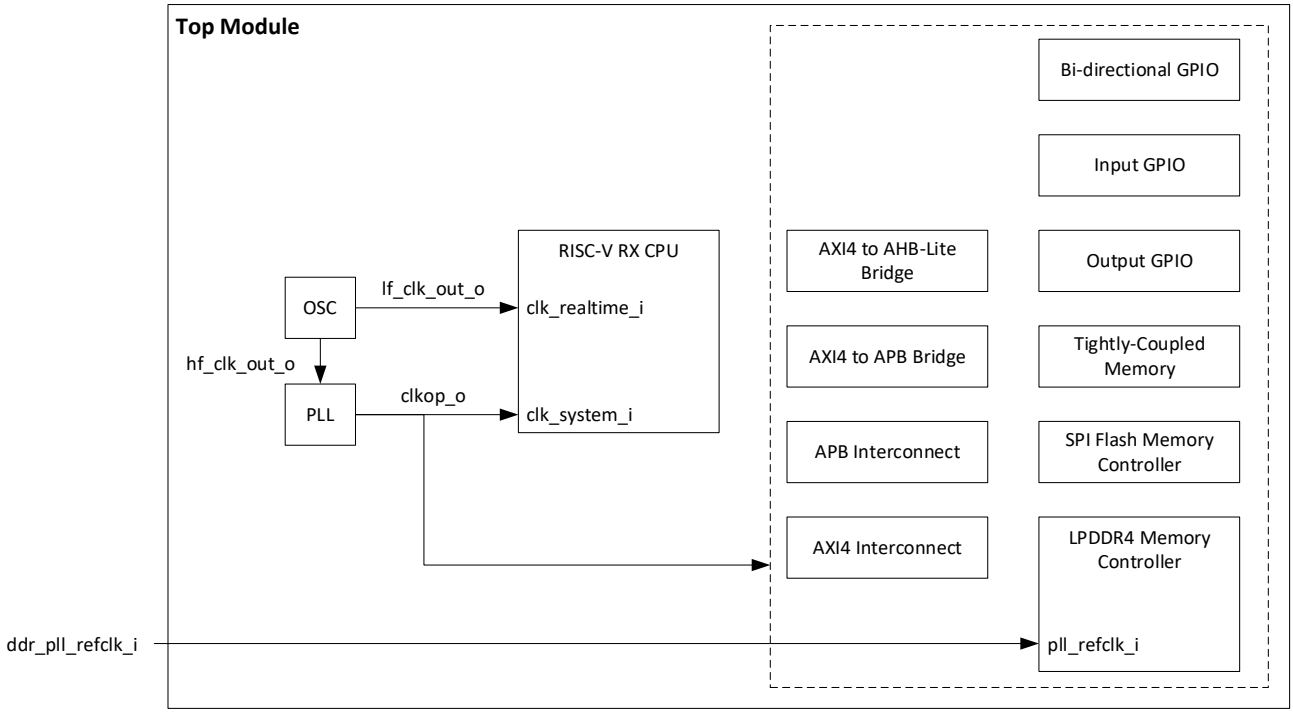

Figure 3.2. Reference Design Clock Domain Block Diagram

Table 3.1. Clock Signals

Name	Source	Destination	Clock Frequency	Description
lf_clk_out_o	FPGA oscillator	RISC-V RX CPU	32 kHz	Low speed real-time clock to RISC-V RX CPU
hf_clk_out_o	FPGA oscillator	PLL	50 MHz	Reference clock input to the PLL
clkop_o	PLL	RISC-V RX CPU, all IPs in the system	50 MHz[1]	PLL primary clock output to clock the RISC-V RX CPU high speed input and all components in the system
ddr_pll_refclk_i	Oscillator on development kit	LPDDR4 memory controller	100 MHz	Reference clock input to the dedicated PLL of the LPDDR4 memory controller

Note:

1. The PLL clkop_o output frequency can be adjusted to values other than 50 MHz, which changes the RISC-V RX CPU and system clock frequencies. In the current reference design, timing closure is achieved at 50 MHz to ensure the stability of the GPIO signals. This is because the GPIO toggling speed is controlled by software and is directly affected by the RISC-V RX CPU and system clock frequencies. Therefore, PLL clkop_o is configured to 50 MHz.

3.4. Reset Scheme

3.4.1. Reset Overview

Figure 3.3 shows the reset scheme of the reference design. Table 3.2 lists the reset signals.

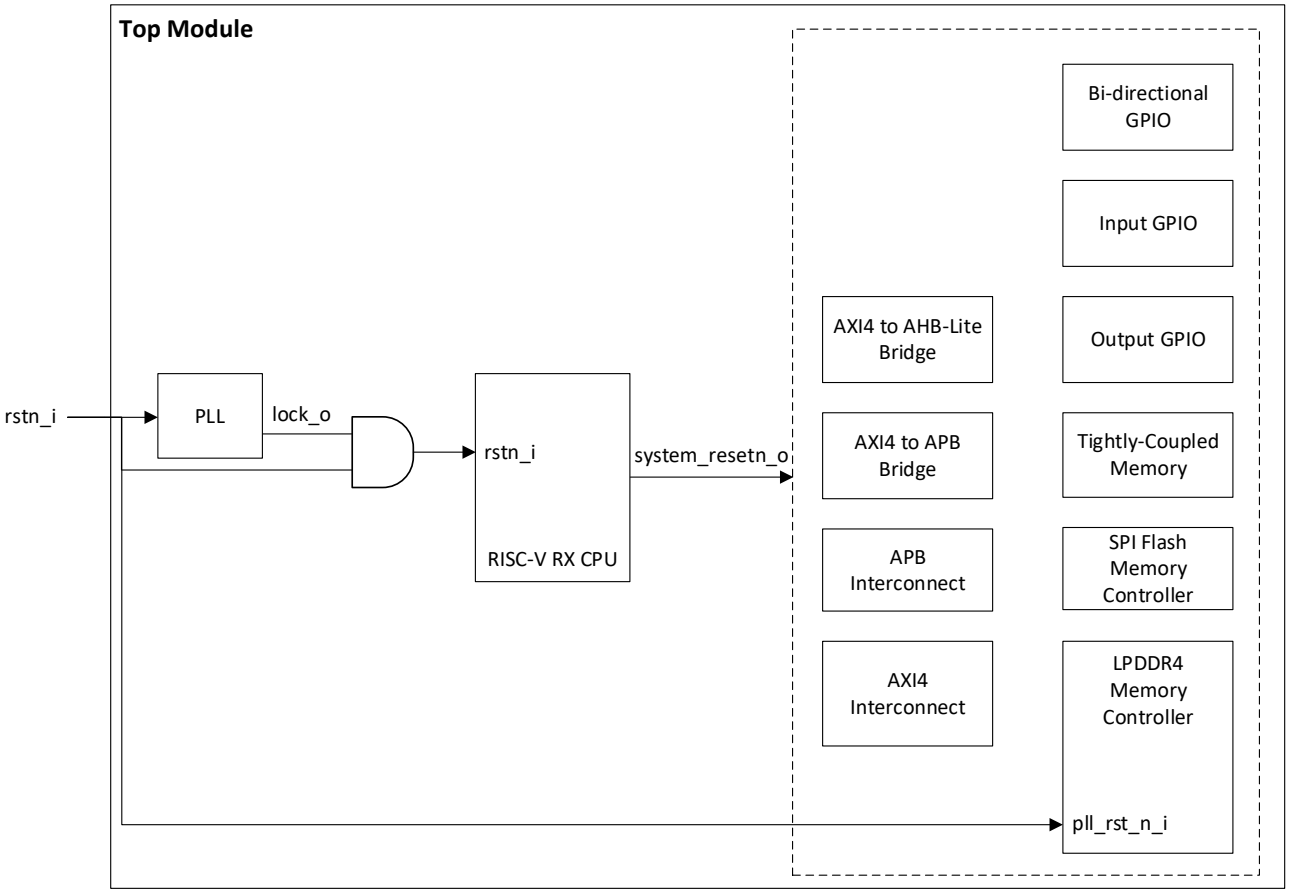

Figure 3.3. Reference Design Reset Scheme Block Diagram

Table 3.2. Reset Signals

Name	Source	Destination	Description
rstn_i	Push button on development kit	PLL reset input	Active-low reset input for the reference design Resets the PLL and LPDDR4 dedicated PLL. Activated by pressing a push button on the board (CertusPro-NX Versa Board SW3).
rstn_i (RISC-V RX CPU)	PLL lock and rstn_i	RISC-V RX CPU reset input	Releases RISC-V RX CPU from reset once rstn_i is not asserted AND PLL is locked.
system_rstn_o	RISC-V RX CPU	All components in system	RISC-V RX output reset is used to reset all components in the system. This trigger resets during CPU debugging mode and is synchronous to the system clock domain.

4. Signal Description

The input/output interface signals for the top-level module are shown in Table 4.1.

Table 4.1. Primary I/O

Port Name	I/O	Width	Description
Clocking			
ddr_pll_refclk_i	In	1	Reference clock input to the dedicated PLL of the LPDDR4 memory controller
Reset			
rstn_i	In	1	Active-low reset input for the reference design Resets the PLL and LPDDR4 dedicated PLL. Activated by pressing a push button on the board (CertusPro-NX Versa Board SW3).
Bi-directional GPIO Interface			
OSPI_DATA_io	In/Out	8	GPIO bi-directional signals used to support data transfer between host GPIO and Avant device target SPI port in x1 and x8 DDR modes
Input GPIO Interface			
QSPI_DS_i	In	1	Data strobe signal of the data bus in xSPI (x8 DDR) mode
initn_i	In	1	INITN signal from the Avant device The Avant device is ready to accept configuration data when INITN is high. The Avant device INITN signal toggles low when power is applied, PROGRAMN pin is pulsed, or REFRESH command is received.
done_i	In	1	Active-high DONE signal from the Avant device Indicates that the Avant device is in user function mode.
Output GPIO Interface			
OSPI_CLK_o	Out	1	Output clock signal for the clock input of the Avant device target SPI port
OSPI_CS_o	Out	1	Active-low output chip select signal for the SCSN input of the Avant device target SPI port.
UART Interface			
rxd_i	In	1	UART receive input Connects to the board FTDI2232 TXD signal.
txd_o	Out	1	UART transmit output Connects to the board FTDI2232 RXD signal.
SPI Flash Interface			
sclk_o	Out	1	Serial clock to SPI flash
ss_n_o	Out	1	SPI flash chip select
mosi_o	Out	1	Serial data from SPI flash controller (FPGA) to SPI flash
miso_i	In	1	Serial data to SPI flash controller (FPGA) from SPI flash
wpj_o	Out	1	SPI flash write protect Not used.
LPDDR4 Memory Interface			
ddr_ca_o	Out	6	LPDDR4 command/address
ddr_ck_o	Out	1	LPDDR4 clock
ddr_cke_o	Out	1	LPDDR4 clock enable
ddr_cs_o	Out	1	LPDDR4 chip select
ddr_dmi_io	In/Out	4	LPDDR4 data mask
ddr_dq_io	In/Out	32	LPDDR4 data
ddr_dqs_io	In/Out	4	LPDDR4 data strobe
ddr_reset_n_o	Out	1	Memory reset signal

5. RISC-V RX Software Flow

This section describes the software flow of the reference design.

5.1. Target SPI x8 Configuration Flow

The C source code provided in the reference design demonstrates the complete process of configuring the Avant device using the target SPI port in x8 mode. Table 5.1 shows the step-by-step sequence of the configuration flow.

Table 5.1. Target SPI x8 Configuration Flow

Operation	Command Code	Description
CFGMODE	—	Set the CFGMODE pin to low after power-up. By default, the device target SPI port operates in x1 mode.
REFRESH	0x80 0x00 0x0a 0x00	Execute command in target SPI x1 mode. Clears SRAM and starts the device in x1 mode.
Wait 300 µs	—	Wait for the specified amount of time. Allows sufficient time for INITN to transition from high to low and back to high.
Wait for INITN HIGH	—	Poll INITN until it goes high. Indicates the device is ready for configuration.
SSPI_MODE	0x80 0x00 0x0F 0x03	Execute command in target SPI x1 mode. Switches device to target SPI x8 mode. Remains in x8 DDR mode until PROGRAMN pulsing, REFRESH event, power cycle, or another SSPI_MODE mode change.
Wait 180 µs	—	Wait for the specified amount of time. Allows sufficient time for the device to enter target SPI x8 mode.
PROG_ENABLE	0x80 0x00 0x0C 0x00	Execute command in target SPI x8 mode. Enables programming mode.
BITSTREAM_BURST	0x05 0x00 0x00 0x00	Execute command in target SPI x8 mode. Enables bitstream mode. A valid bitstream must follow. The device returns to command mode after the bitstream is fully sent.
Clock in Bitstream	—	Clock in a valid bitstream in target SPI x8 mode.
Wait 40 µs	—	Wait for the specified amount of time. Allows sufficient time for the transition back to command mode after bitstream clock in.
READ _STATUS0	0x01 0x07 0x00 0x00	Execute command in target SPI x8 mode to read from Status Register 0. Checks for the following: • Bit[7] (DONE) = 1; PROG_DONE command has been successfully received through the bitstream. • Bit[11] (Busy Flag) = 0; configuration engine is not busy. • Bit[12] (Fail Flag) = 0; previous hardware command is valid. • Bits[24:21] (BSE Primary Boot Error Code) = 0000; no error has occurred.
PROG_DISABLE	0x80 0x00 0x0E 0x00	Execute command in target SPI x8 mode. If DONE bit is set, initiates wake-up and enters user mode. Otherwise, exits programming mode.
Wait for DONE HIGH	—	Poll DONE until it goes high. Indicates the device has entered user function mode.

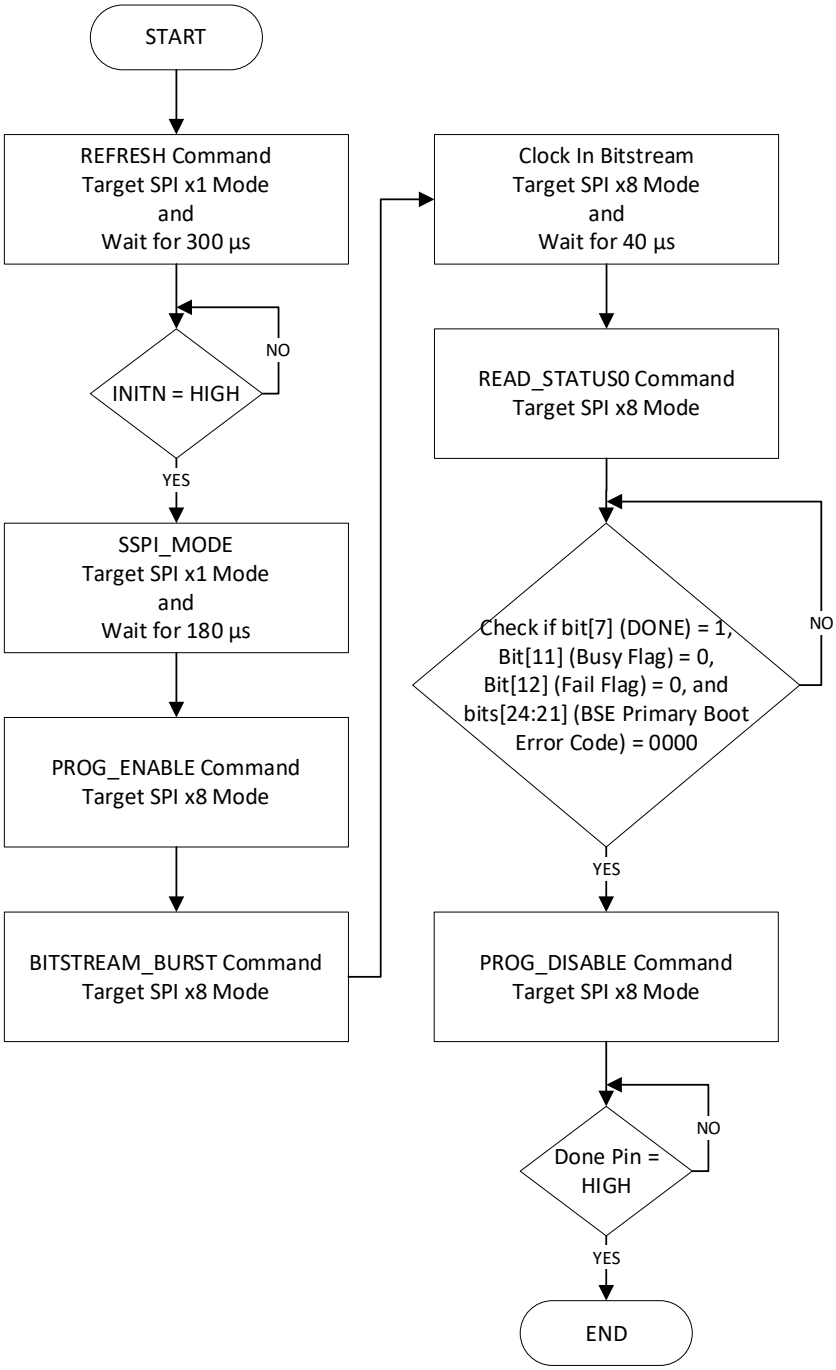

Figure 5.1. Software Flow of Target SPI x8 Configuration

5.2. Application Image Format

The application image contains the application software binary and extra information appended as the header. Figure 5.2 illustrates the application image format. The header information is required for bootloader to perform image validation and integrity check during boot up.

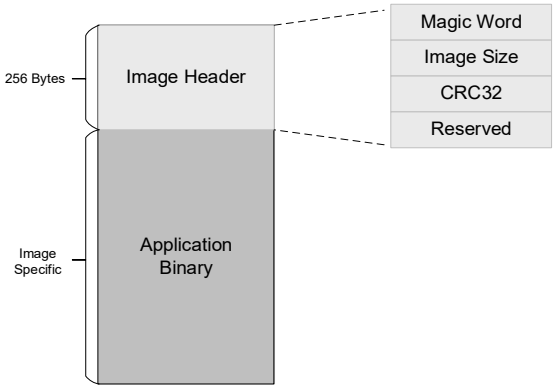

Figure 5.2. Application Image Format

Table 5.2 provides the image header details.

Table 5.2. Application Image Header

Header Property	Value	Description
Magic Word	0x4C534343	Represents *LSCC* in ASCII notation. Magic Word is used to check if a valid image header exists in flash.
Image Size	Variable	Application binary size in number of bytes. This provides the total size to be copied by the bootloader.
CRC32 Checksum	Variable	CRC32 checksum calculated for the application binary.
Reserved	—	Padding to align the image header to flash page size (256 bytes).

5.3. Application Image Generation

Figure 5.3 shows the application image generation process.

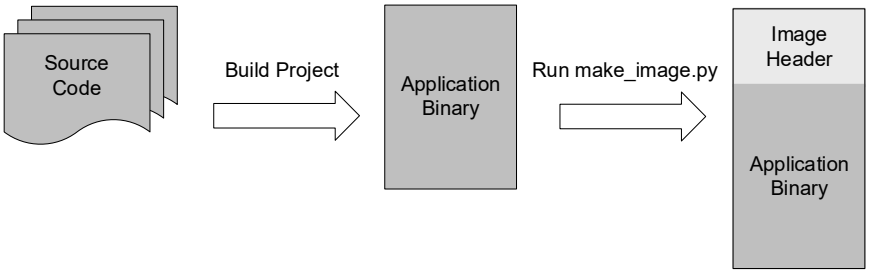

Figure 5.3. Application Image Generation

The application image generation process is as follows:

1. Build the application software project from source code using Lattice Propel SDK. This creates the application binary file (.bin).
2. Run the *make_image* Python script (provided with this reference design) to append the image header to the application binary.

Refer to the Generating the Flash Image section for more information.

6. Running the Reference Design

This section explains how to execute the Processor-Based Controller for Avant Device Target SPI x8 Configuration reference design using the pre-built binary files included in the design package.

The design has been validated on the following Lattice development boards:

- CertusPro-NX Versa Board
- Lattice Avant-X Versa Board

6.1. Extracting the Reference Design Files

Before running the reference design, download the following design package .zip file from the Lattice Processor-Based Controller for Avant Device Target SPI x8 Configuration Reference Design web page:

- CertusPro-NX_OSPI_Host_SoC.zip

Unzip the .zip file to your local directory, for example to C:\workspace\.

The extracted directory is named *CertusPro-NX_OSPI_Host_SoC*. This path is hereinafter referred to as *<my_work_dir>*. For example, ***<**my_work_dir**> = C:\workspace\CertusPro-NX_OSPI_Host_SoC***.

6.2. Implementing the Reference Design

To run the target SPI x8 configuration reference design, two boards are required:

- Host controller board: CertusPro-NX Versa board
- Target board: Lattice Avant-X Versa board (with hardware modifications)

Note: For details on modification of the Lattice Avant-X Versa board, submit a technical support case. Refer to the Technical Support Assistance section for more information.

Two configuration files are required to initialize the design and perform the target SPI x8 configuration:

- BIN file: Contains the bitstream data used to configure the target device (Avant LAV-AT-X70-3LFG1156C device).
- BIT file: Contains the bitstream data used to configure the host controller device (CertusPro-NX LFCPNX-100-9LFG672C device).

6.3. Generating the Flash Image

The application image requires a header, appended using the provided Python script as described in the Application Image Generation section. The *make_image* Python script is provided in the reference design package in the *<my_work_dir>\sw* directory.

Note: This header is required as part of the process of reading and loading the bitstream file (converted to .bin) from SPI flash memory into LPDDR4 memory.

To generate the flash image:

1. Launch the Command Prompt from the Windows Start menu.
2. Ensure you have Python 3 installed. To check, enter *python --version*. The output is as shown in Figure 6.1.
   Note: If you do not have Python 3 installed, go to the Python official website to download and install the latest stable version.

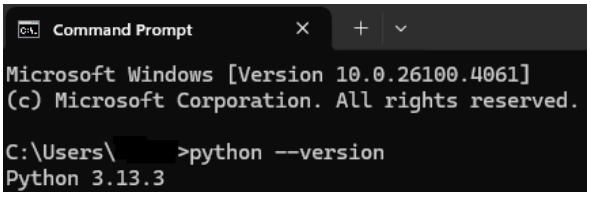

Figure 6.1. Verifying Python Version

3. In the Command Prompt, enter *cd <my_work_dir>\sw* to change the directory.
4. Enter the following to run the *make_image* Python script:

```
python make_image.py <Avant .bit file> <Output .bin file>
```

   For example:

```
python make_image.py avant_x70_soc.bit avant_x70_soc.bin
```

   The output .bin file is generated in the directory.

6.4. Setting Up the UART Terminal

The software code in this reference design displays messages on the terminal through the UART interface.

To set up the UART terminal:

1. Launch Tera Term.
   Note: Serial Terminal in Propel SDK, or other terminal emulator software such as PuTTY, may be used.
2. Select **File > New Connection**.
3. In the New Connection dialog box, select **Serial**.
4. Select the correct COM port from the **Port** drop-down list. This is typically the COM port with the higher number. For example, if you see COM1 and COM2, choose COM2.
5. Click **OK**.
6. Select **Setup > Serial Port**.
7. In the Serial port setup and connection dialog box, select **115200** for **Speed**.
8. Set the remaining options as follows:
   - Data: 8 bit
   - Parity: none
   - Stop bits: 1 bit
   - Flow control: none
   - Transmit delay: 0 msec/char; 0 msec/line
9. Click **New setting** to complete the setup process.

6.5. Programming the Application Image to SPI Flash

The bitstream file for the target device (Avant LAV-AT-X70-3LFG1156C device) is converted into a .bin file and programmed into the SPI flash memory on the CertusPro-NX Versa board before starting the target SPI x8 configuration software. This ensures that there is a valid image to be loaded into the LPDDR4 memory. Refer to the Generating the Flash Image section for the steps to generate an application image.

To program the application image to SPI flash:

1. Launch Radiant Programmer from the Windows Start menu.
2. In the Getting Started dialog box, select **Open an existing programmer project**.

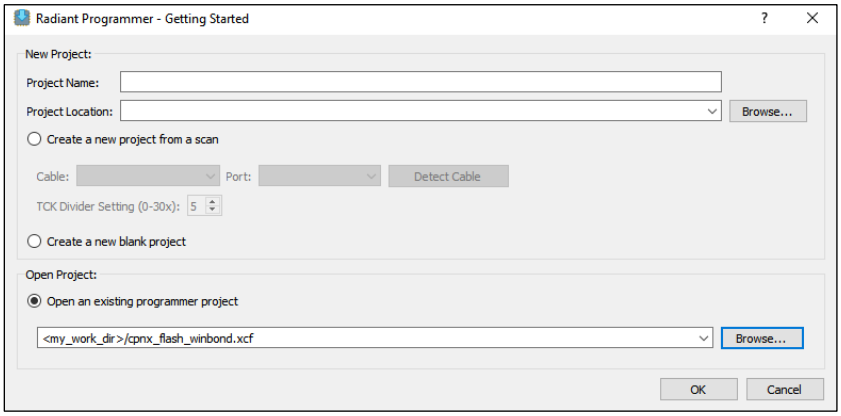

Figure 6.2. Radiant Programmer – Getting Started Dialog Box

3. Click **Browse**.
4. Navigate to the *<my_work_dir>\prebuilt_images* directory and select *cpnx_flash_winbond.xcf*.
5. Click **OK**.
6. In the Radiant Programmer main interface, select **Run > Program Device**.

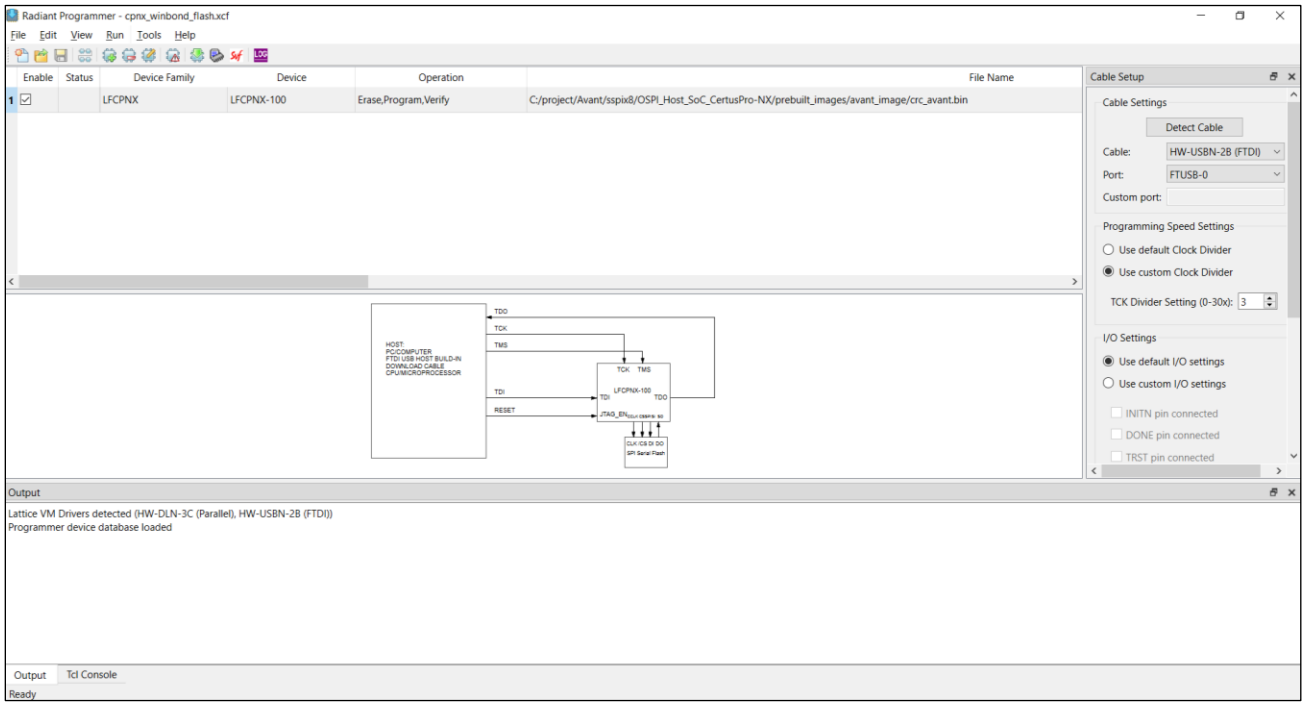

Figure 6.3. Radiant Programmer – SPI Flash Erase, Program, Verify Operation

The **Output** console displays the operation successful message.

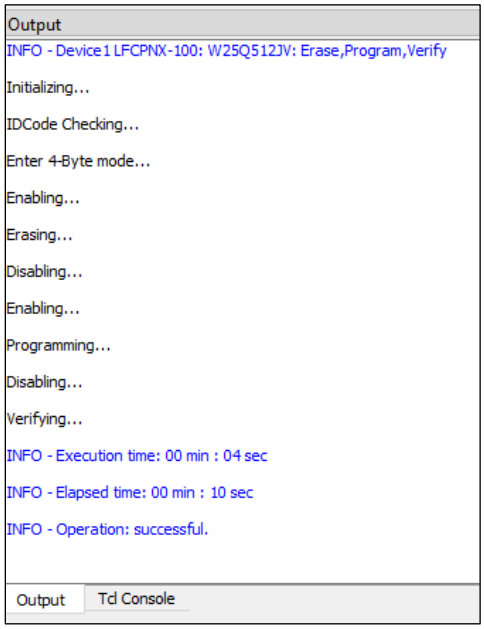

Figure 6.4. Output Console – Programming the Application Image to SPI Flash

6.6. Programming the FPGA Bitstream

The bitstream (.bit) file programmed into the CertusPro-NX Versa board contains the RISC-V-based controller design for Avant device target SPI x8 configuration.
To program the bitstream into the device:

1. In the Radiant Programmer main interface, select **File** > **Open Project**.
2. Click **Browse**, navigate to the *<my_work_dir>\prebuilt_images* directory, and select *cpnx_bitstream_program.xcf*.
3. Click **Open**.
4. In the Radiant Programmer main interface, select **Run > Program Device**.
   The **Output** console displays the operation successful message.

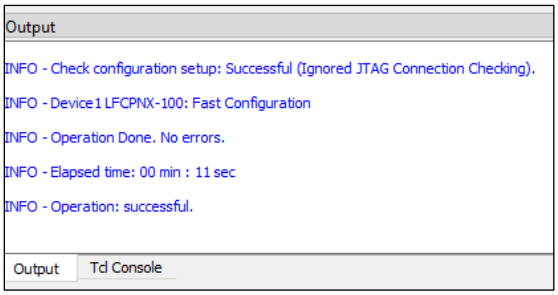

Figure 6.5. Output Console – Programming the FPGA Bitstream

6.7. Executing OSPI Host Source Code Using Lattice Propel Software

To manually run and debug the OSPI host source code for configuring the Avant device using the Lattice Propel software:

1. Launch Lattice Propel.
2. In the Propel Launcher window, navigate to the \workspace\CertusPro-NX_OSPI_Host_SoC\sw directory.
3. Click **Launch**.

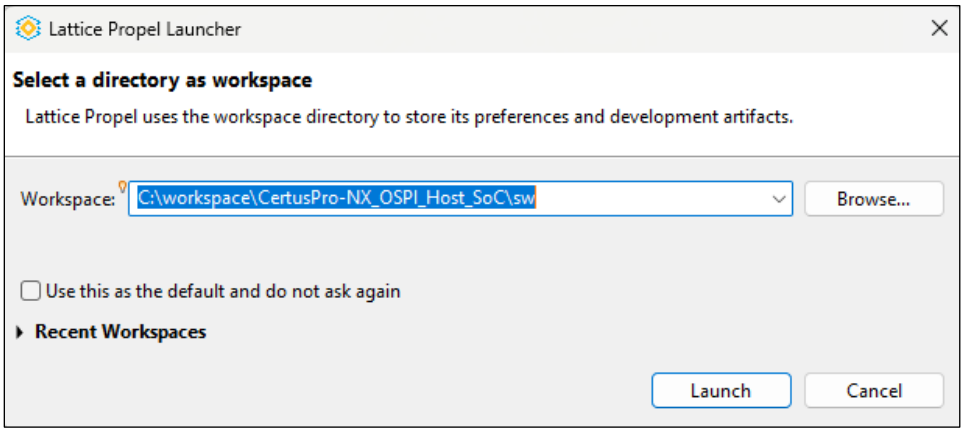

Figure 6.6. Lattice Propel Launcher Window

4. In the Lattice Propel main interface, select **Run > Debug Configurations**. The Debug Configurations window appears.

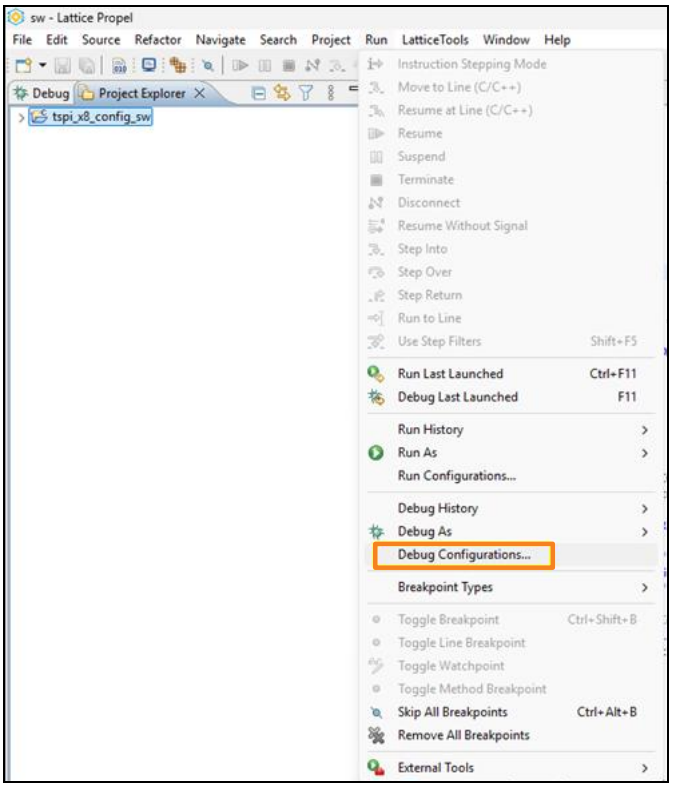

Figure 6.7. Lattice Propel Main Interface – Selecting Debug Configurations

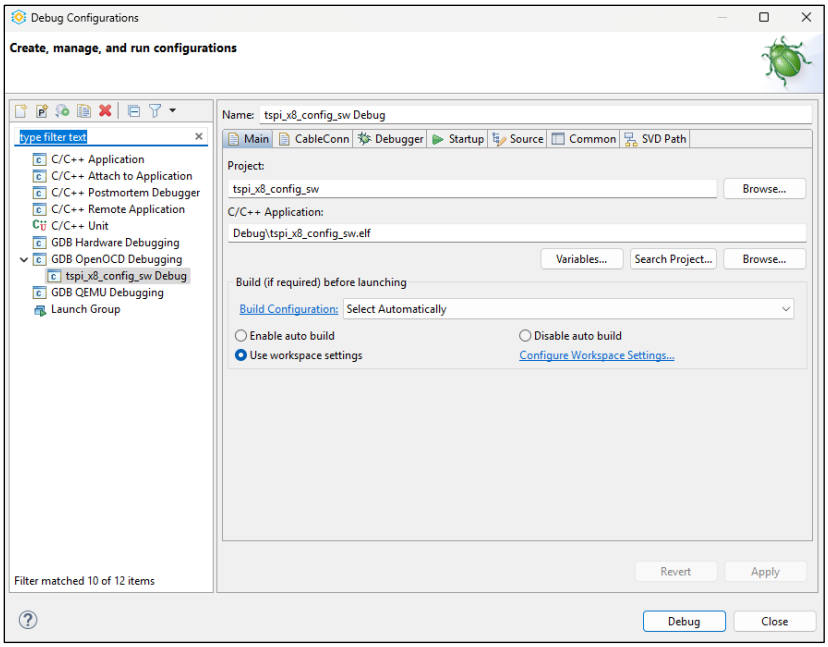

Figure 6.8. Debug Configurations Window

5. In the Debug Configurations window, perform the following to setup the cable connection:
   a. Click on the **CableConn** tab.
   b. Click **Detect Cable** and select **FTUSB0** from the drop-down list.
   c. Click **Scan Device** and select **LFCPNX-100:0x010F4043** from the drop-down list.
   d. Click **Apply** followed by **Debug**.

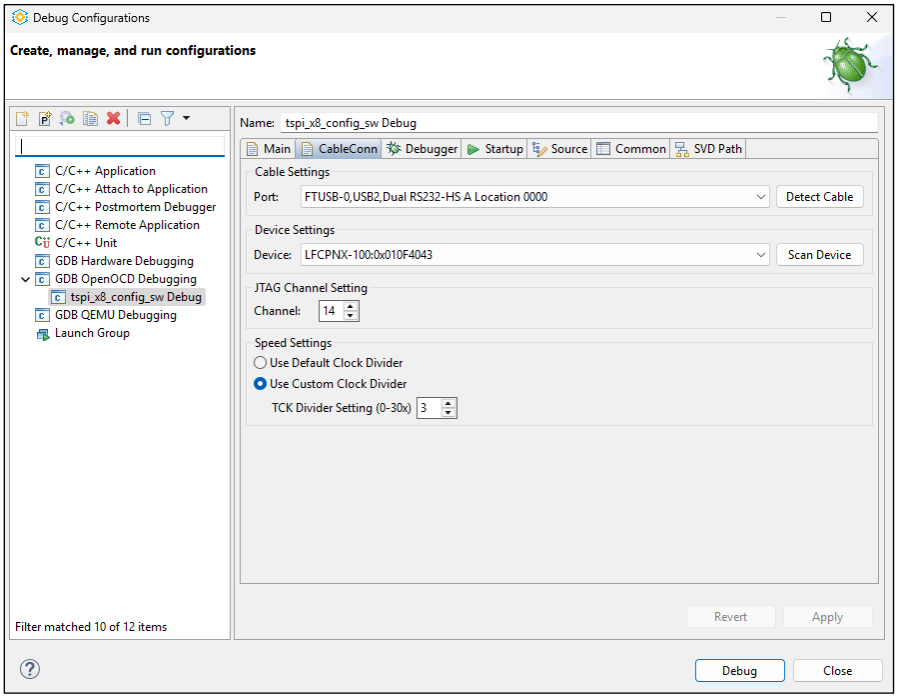

Figure 6.9. Debug Configurations Window - Setting Up Cable Connection

6. Once debug mode starts, the C source code (main.c) begins executing. Click on the **Console** tab at the bottom of the Lattice Propel main interface to monitor system messages.

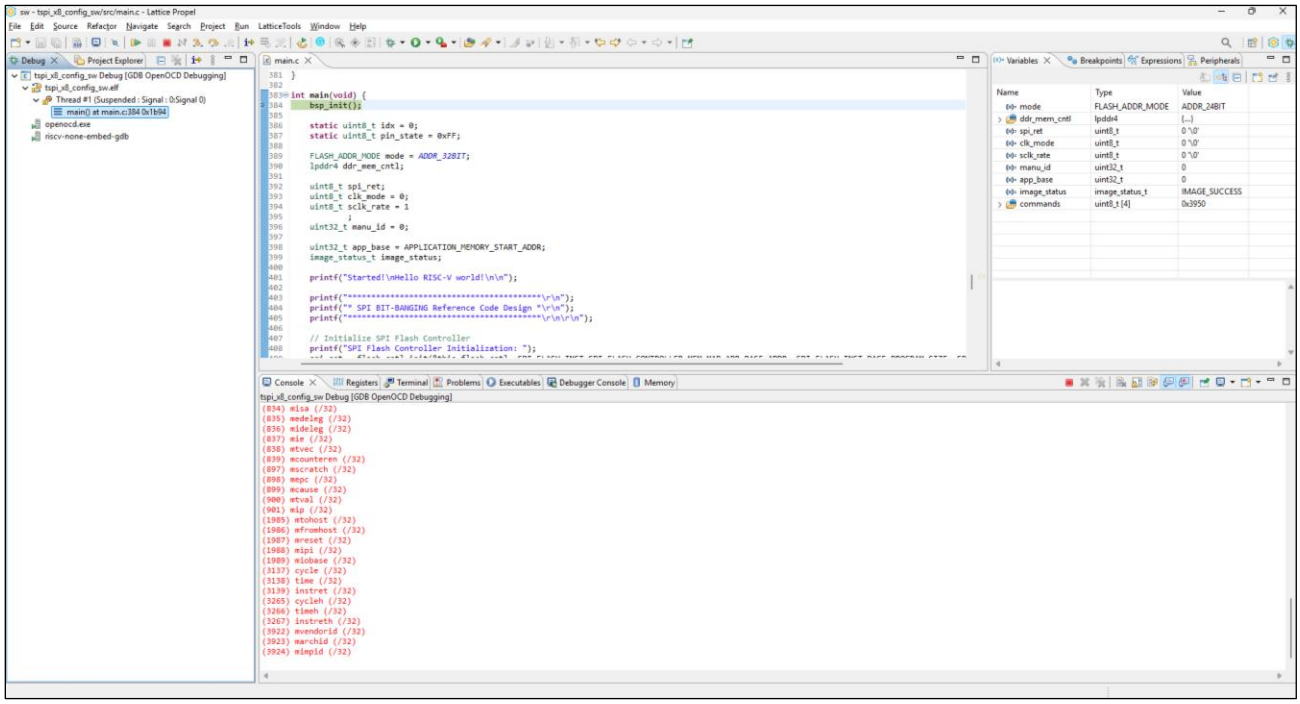

Figure 6.10. Lattice Propel Main Interface – Monitoring Console Output

7. To view printed messages from the OSPI host source code, set up a UART terminal. Refer to the Setting Up the UART Terminal section for more information. After setting up the terminal, click the **Resume** button (or press F8) to continue code execution.

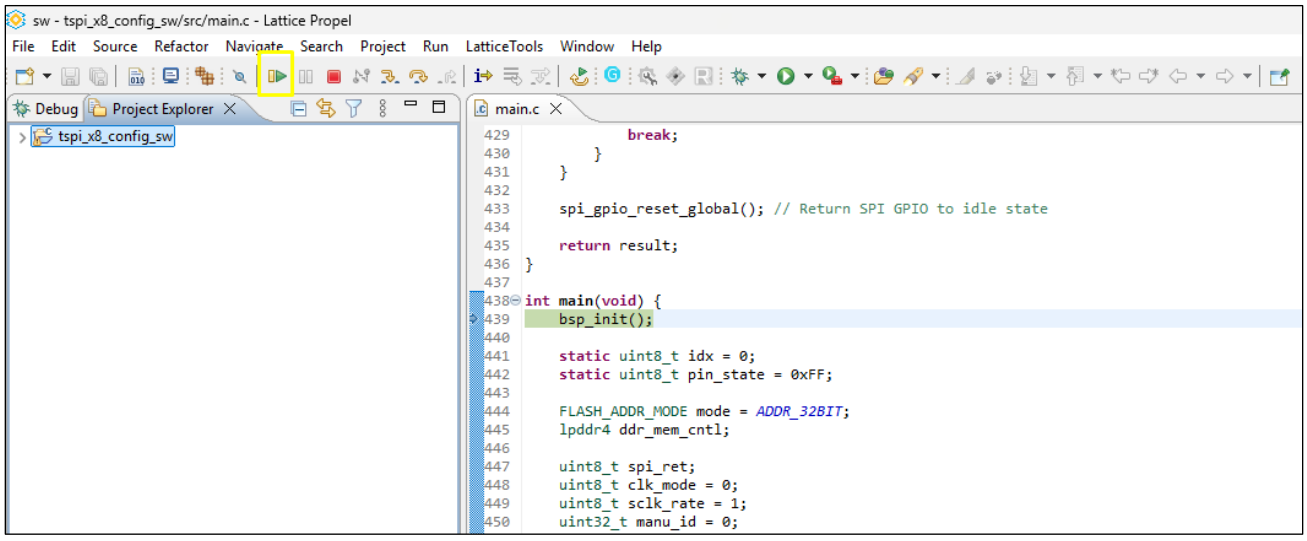

Figure 6.11. Lattice Propel Main Interface – Resuming Code Execution

8. Verify that the printed messages from the OSPI host source code in the UART terminal (for example, Tera Term) confirm successful execution of the configuration process.

```
Started!
Hello RISC-V world!

****************************************
* SPI BIT-BANGING Reference Code Design *
****************************************

SPI Flash Controller Initialization: Passed
SPI Flash Controller Set ADDR mode: Passed
Flash manufacturer ID: ef

Memory Controller Initialization:
INFO: DDR PLL Locked
INFO: LPDDR4 training complete; LPDDR4 training status = b001f
INFO: LPDDR4 initialized successfully

Load and CRC check image
SRC : Flash addr 0
DEST: DDR addr 40000000
Image Check Passed

SPI Configuration Start
Initialization signal received.
spi_x8_write_byte_burst: Start
spi_x8_write_byte_burst: End
Received Data: 00 00 04 F0
DONE bit (7) is HIGH
Busy Flag (bit 11) is LOW
Fail Flag (bit 12) is LOW
BSE Primary Boot Error Code (bits 24:21) is 0000
Done signal received.
SPI Configuration Complete

Main loop completed. System entering idle state.
```

Figure 6.12. Printed Messages in UART Terminal

6.8. Target SPI x8 Configuration Flow Validation from Waveform Analysis

The target SPI x8 configuration flow is validated using a logic analyzer. The captured waveforms provide visual validation of the execution of each command during the configuration process by the OSPI host source code.

- REFRESH command (0x80 0x00 0x0A 0x00)
  - Mode: Target SPI x1
  - Function: Clears the SRAM and prepares the device for configuration.
  - Signal behavior:
    - INITN is de-asserted (low) after the command is sent.
    - INITN is re-asserted (high) once the device is ready.

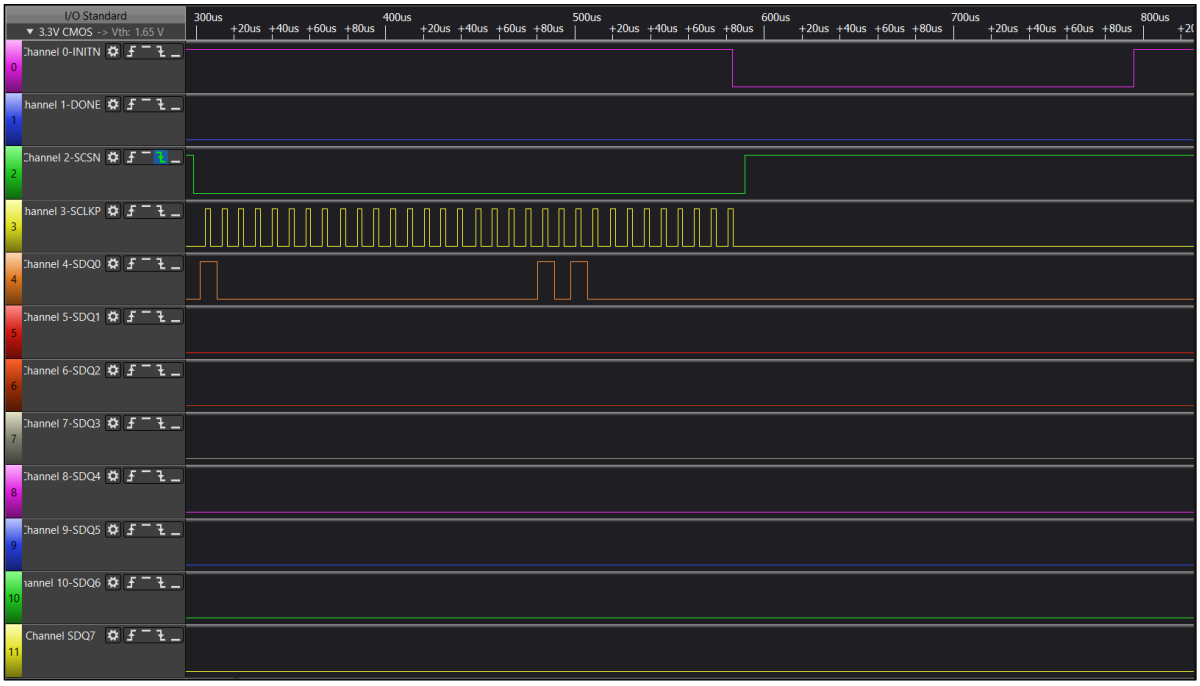

Figure 6.13. REFRESH Command Waveform in Target SPI x1 Mode

- SSPI_MODE command (0x80 0x00 0x0F 0x03)
  - Mode: Target SPI x1
  - Function: Switches the target SPI interface from x1 mode to x8 DDR mode.

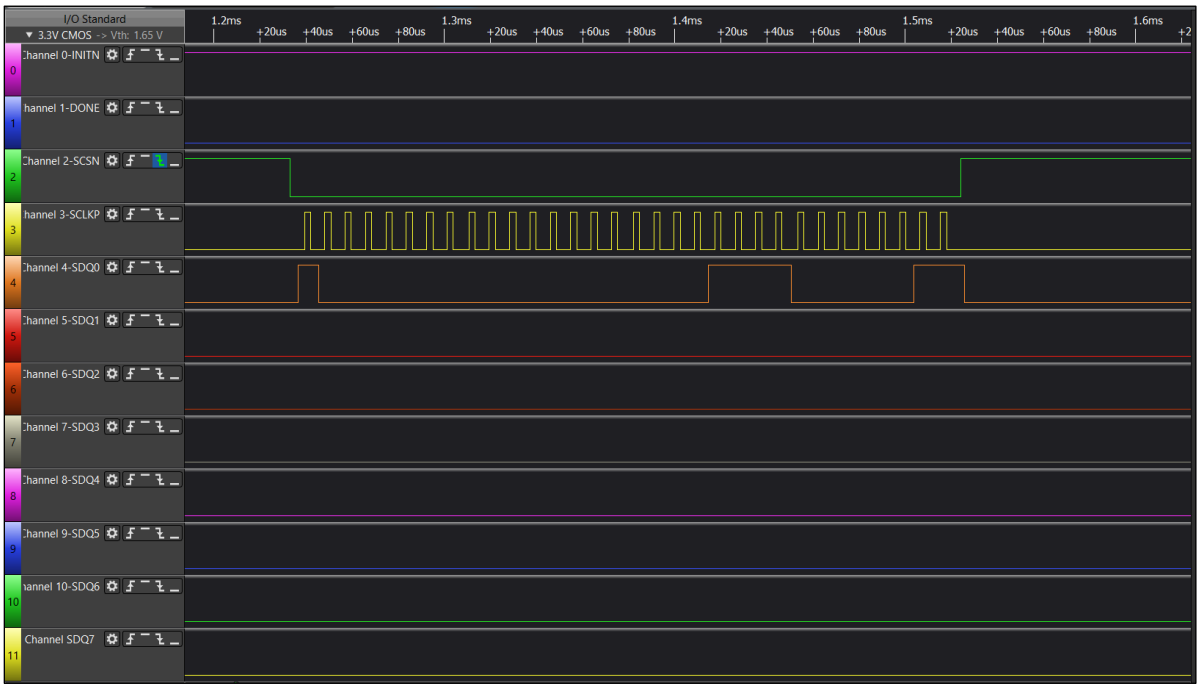

Figure 6.14. SSPI_MODE Command Waveform in Target SPI x1 Mode

- PROG_ENABLE command (0x80 0x00 0x0C 0x00)
  - Mode: Target SPI x8
  - Function: Enables the device for programming.

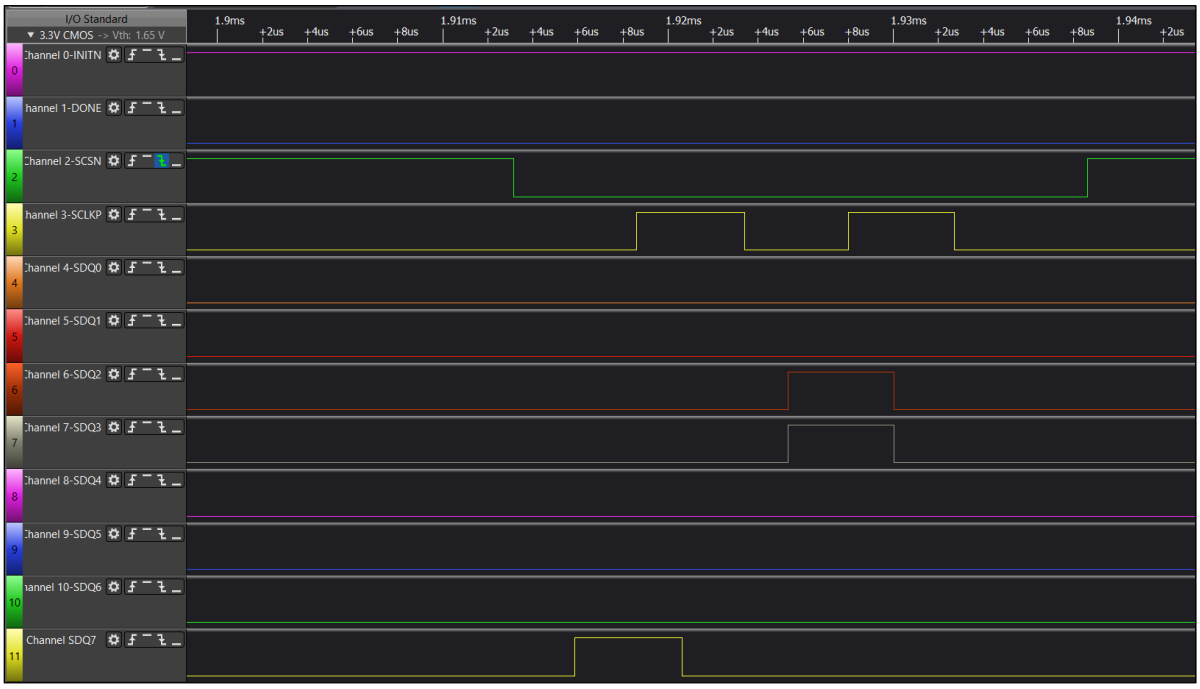

Figure 6.15. PROG_ENABLE Command Waveform in Target SPI x8 Mode

- BITSTREAM_BURST command (0x05 0x00 0x00 0x00)
  - Mode: Target SPI x8
  - Function: Initiates bitstream mode. A valid bitstream must follow this command.

Figure 6.16. BITSTREAM_BURST Command Waveform in Target SPI x8 Mode

Figure 6.17. Start of Bitstream Data Waveform in Target SPI x8 Mode

Figure 6.18. End of Bitstream Data Waveform in Target SPI x8 Mode

- READ_STATUS0 command (0x01 0x07 0x00 0x00)
  - Mode: Target SPI x8
  - Function: Reads Status Register 0 and checks if bit[7] (DONE) = 1, bit[11] (Busy Flag) = 0, bit[12] (Fail Flag) = 0, and bits[24:21] (BSE Primary Boot Error Code) = 0000.

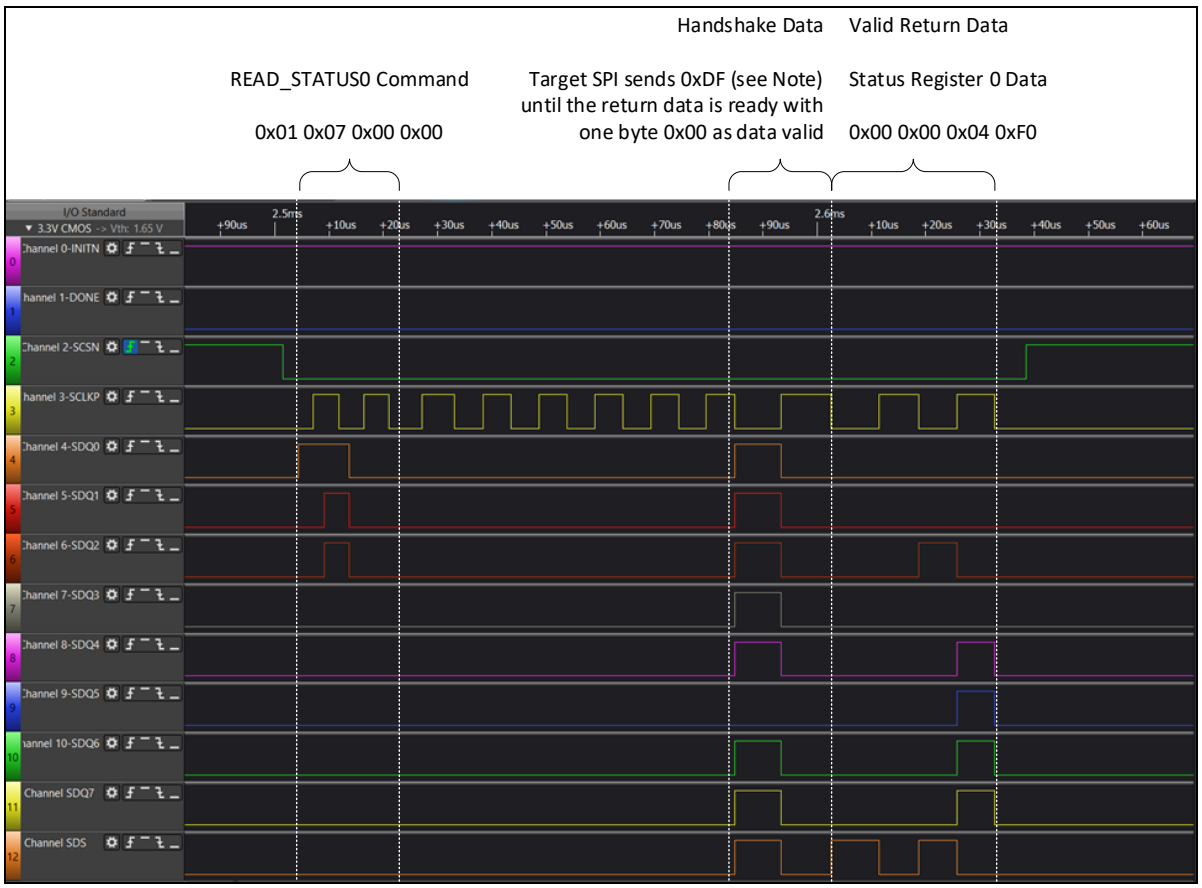

Figure 6.19. READ_STATUS0 Command Waveform in Target SPI x8 Mode

Note: Due to a known bug in the AS2 device, the target SPI sends 0xDF instead of the expected 0xFF during the handshake process. This issue will be resolved in the production version of the device.

- PROG_DISABLE Command (0x80 0x00 0x0E 0x00)
  - Mode: Target SPI x8
  - Function: Initiates the wake-up sequence.
  - Signal behavior: Upon successful configuration, the DONE pin is asserted high, indicating that the device has entered user mode.

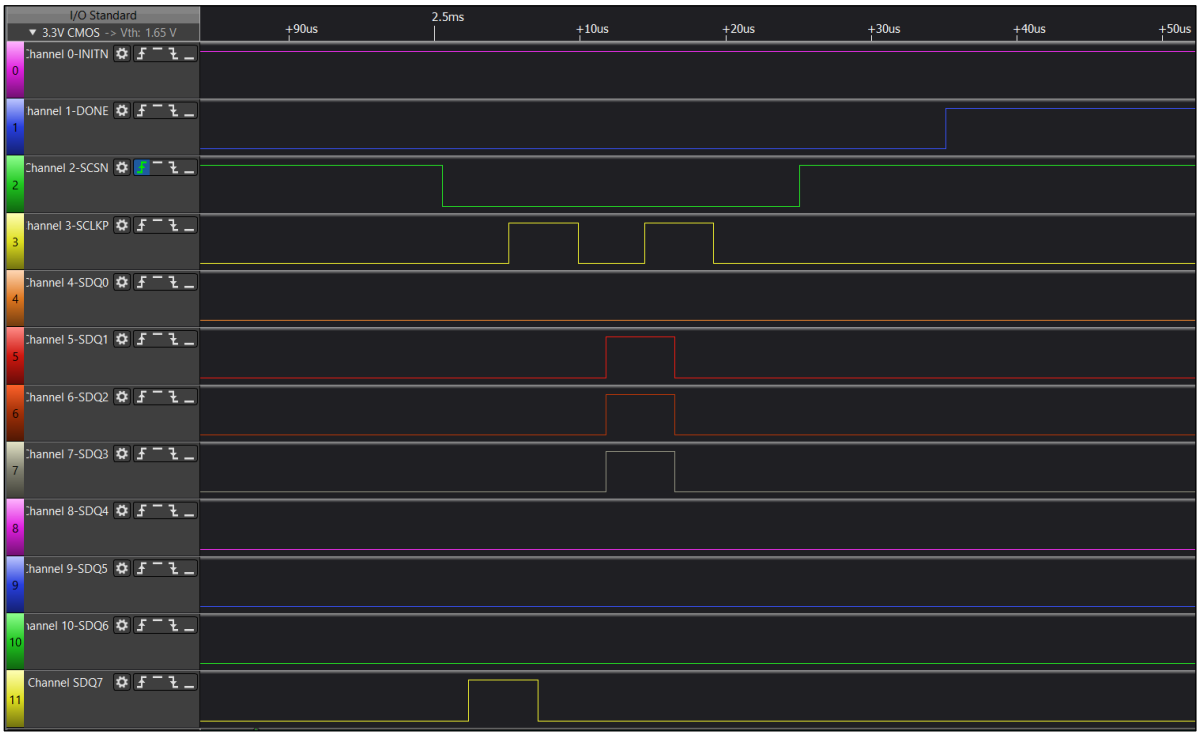

Figure 6.20. PROG_DISABLE Command Waveform in Target SPI x8 Mode

7. Compiling the Reference Design

This section describes the process of compiling the Processor-Based Controller for Avant Device Target SPI x8 Configuration reference design. Compilation is required to generate the binary files from the source files. The compilation process involves the following software tools for generating the FPGA bitstream and the RISC-V processor software executable files:

- Lattice Propel Builder 2025.1
- Lattice Radiant Software 2025.1
- Propel Software Development Kit (SDK) 2025.1

Note: Recompilation is not required. However, you can recompile the design if you want to re-parameterize the IP.

The following sections show the typical design flow for the RISC-V-based design using Lattice software tools:

- Building and Generating the RISC-V Processor Using Lattice Propel Builder
- Synthesizing RTL Files and Generating the Bitstream File Using Lattice Radiant Software
- Building the Software Project Using Propel SDK

7.1. Building and Generating the RISC-V Processor Using Lattice Propel Builder

The RISC-V processor system in this reference design is built and generated using Lattice Propel Builder. The components (IPs) are instantiated and connected using the schematic graphical user interface (GUI) of Propel Builder as shown in Figure 7.1.

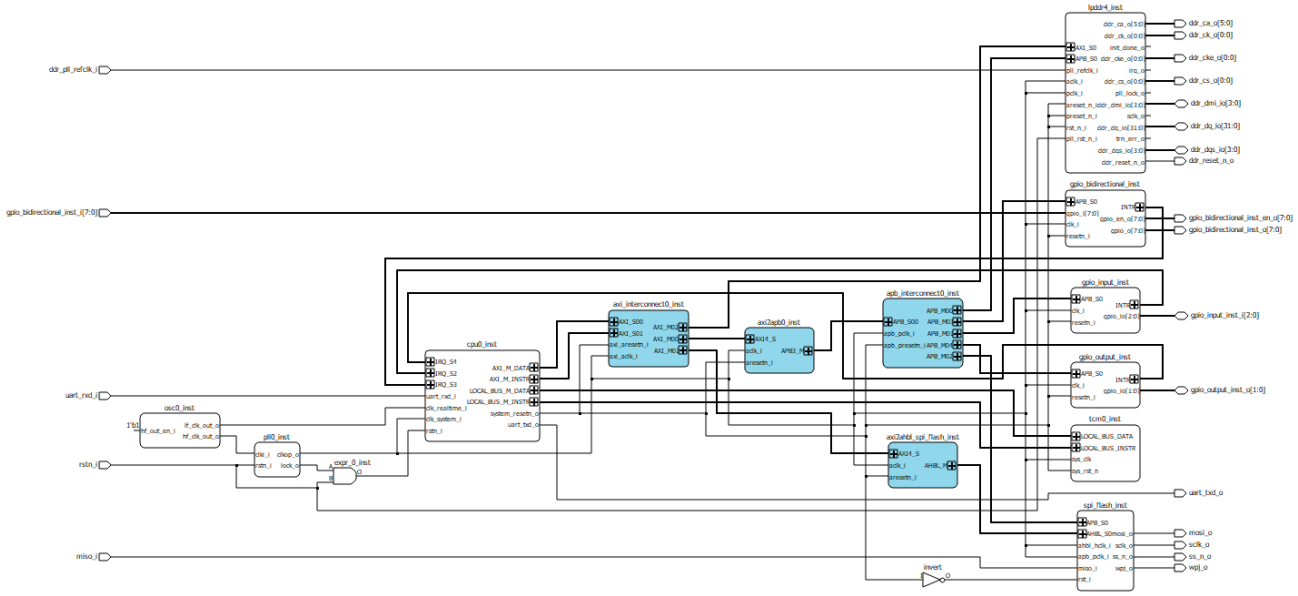

Figure 7.1. Lattice Propel Builder Schematic View

Note: The following procedure is required only after making modifications to the design. Refer to the Customizing the Reference Design section for more information.

To generate the RISC-V processor system:

1. Launch Lattice Propel Builder from the Windows Start menu.
2. In the Propel Builder main interface, select **File > Open Design**. This opens the Open sbx dialog box.
3. Navigate to the *<my_work_dir>\cpnx_riscv_lpddr4_rd* directory.
   Note: Before performing this step, ensure you have already completed the procedure in the Extracting the Reference Design Files section.
4. Select *cpnx_riscv_lpddr4_rd.sbx* and click **Open**.
   Note: *Port Bus width does not match for port* INFO messages can be safely ignored.

5. In Propel Builder, select **Design > Validate Design**. You should not see any errors in the **Tcl Console** window.
   Note: *Dangling input* and *no available driver for localbus_tcm* INFO messages can be safely ignored.

```
Tcl Console
INFO <2359136> - Start: sbp_design drc.
INFO <2357031> - Dangling input Port 'BUSER' is set to default value '0' on bus 'AXI_M02' in component 'axi_interconnect0_inst'.
INFO <2357031> - Dangling input Port 'RUSER' is set to default value '0' on bus 'AXI_M02' in component 'axi_interconnect0_inst'.
INFO <2357031> - Dangling input Port 'ARUSER' is set to default value '0' on bus 'AXI_S00' in component 'axi_interconnect0_inst'.
INFO <2357031> - Dangling input Port 'AWUSER' is set to default value '0' on bus 'AXI_S00' in component 'axi_interconnect0_inst'.
INFO <2357031> - Dangling input Port 'WUSER' is set to default value '0' on bus 'AXI_S00' in component 'axi_interconnect0_inst'.
INFO <2357031> - Dangling input Port 'ARUSER' is set to default value '0' on bus 'AXI_S01' in component 'axi_interconnect0_inst'.
INFO <2357031> - Dangling input Port 'AWUSER' is set to default value '0' on bus 'AXI_S01' in component 'axi_interconnect0_inst'.
INFO <2357031> - Dangling input Port 'WUSER' is set to default value '0' on bus 'AXI_S01' in component 'axi_interconnect0_inst'.
INFO <2357031> - Dangling input Port 'HMASTLOCK' is set to default value '0' on bus 'AHBL_S0' in component 'spi_flash_inst'.
INFO <2359137> - Finished: sbp_design drc.
```

Figure 7.2. Tcl Console – No Error from Validate Design Operation

6. Click **Design > Generate**. You should not see any errors in the **Tcl Console** window.
   Note: *Dangling input* INFO messages can be safely ignored.

```
Tcl Console
INFO <2359189> - Start: sbp_design generate.
INFO <2357031> - Dangling input Port 'BUSER' is set to default value '0' on bus 'AXI_M02' in component 'axi_interconnect0_inst'.
INFO <2357031> - Dangling input Port 'RUSER' is set to default value '0' on bus 'AXI_M02' in component 'axi_interconnect0_inst'.
INFO <2357031> - Dangling input Port 'ARUSER' is set to default value '0' on bus 'AXI_S00' in component 'axi_interconnect0_inst'.
INFO <2357031> - Dangling input Port 'AWUSER' is set to default value '0' on bus 'AXI_S00' in component 'axi_interconnect0_inst'.
INFO <2357031> - Dangling input Port 'WUSER' is set to default value '0' on bus 'AXI_S00' in component 'axi_interconnect0_inst'.
INFO <2357031> - Dangling input Port 'ARUSER' is set to default value '0' on bus 'AXI_S01' in component 'axi_interconnect0_inst'.
INFO <2357031> - Dangling input Port 'AWUSER' is set to default value '0' on bus 'AXI_S01' in component 'axi_interconnect0_inst'.
INFO <2357031> - Dangling input Port 'WUSER' is set to default value '0' on bus 'AXI_S01' in component 'axi_interconnect0_inst'.
INFO <2357031> - Dangling input Port 'HMASTLOCK' is set to default value '0' on bus 'AHBL_S0' in component 'spi_flash_inst'.
% sbp_design save
% sbp_design pge sge  -i {C:/workspace/CertusPro-NX_OSPI_Host_SoC/cpnx_riscv_lpddr4_rd/cpnx_riscv_lpddr4_rd.sbx} -o {C:/workspace/CertusPro-NX_OSPI_Host_SoC/cpnx_riscv_lpddr4_rd/../} -nc
INFO <2359992> - (PGE FileExistence) No available driver for localbus_tcm
```

Figure 7.3. Tcl Console – No error from Generate Operation

7.2. Synthesizing RTL Files and Generating the Bitstream File Using Lattice Radiant Software

The Lattice Radiant software is used to synthesize the RTL design files (from Propel Builder) and generate the FPGA bitstream file.

To create the FPGA bitstream file, perform the following:

1. Launch the Lattice Radiant software from the Windows Start menu.
   Note: Alternatively, you can launch the Lattice Radiant software from Lattice Propel Builder.
2. In the Lattice Radiant software main interface, select **File > Open > Project**.
3. Navigate to *<my_work_dir>\cpnx_riscv_rx_rd.rdf* and click **Open**.
4. Click **Export Files** to start bitstream compilation. The compilation may take some time to complete. The .bit file is generated or updated in the project path (*<my_work_dir>\impl_1*) after compilation is completed successfully.

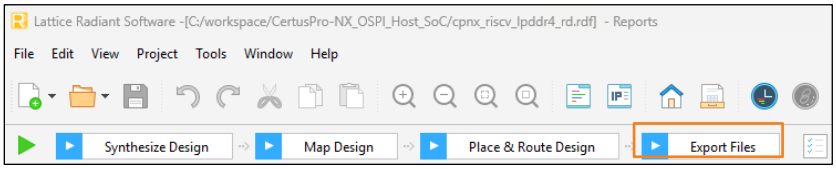

Figure 7.4. Export Files

Note: If you observe timing violations after the Lattice Radiant compilation, this may be due to the **Placement Iteration Start Point** number for place and route not working optimally on your machine. In this case, perform these next steps.

5. In Lattice Radiant, select **Project > Active Strategy > Place & Route Design Settings**.
6. Change **Placement Iteration Start Point** from **1** to **2.**
7. Change **Placement Iterations** from **1** to **5**.

8. Change **Placement Save Best Run** from **1** to **5**.
9. Click **OK**.
10. Click **Export Files**. This re-runs the place and route design with five different incremental start point values. The place and route reports show all five placement results and the best timing result is selected.

7.3. Building the Software Project Using Propel SDK

The Lattice Propel SDK is used to build the RISC-V processor software project.

7.3.1. Setting Up the Project Workspace

This section describes launching the Propel SDK tool and setting up the Propel SDK workspace for software projects.

Note: These steps are required only when you create a Propel SDK workspace for the first time. On subsequent Propel SDK launches, you can reuse the previously created workspace.

To create a new Propel SDK workspace:

1. Launch Lattice Propel from the Windows Start Menu.
2. In the Lattice Propel Launcher dialog box, click **Browse**.
3. Navigate to the *<my_work_dir>* directory.
4. Create a new folder named *propel_workspace* and click **Select Folder**. Figure 7.5 shows the reference design workspace setup.

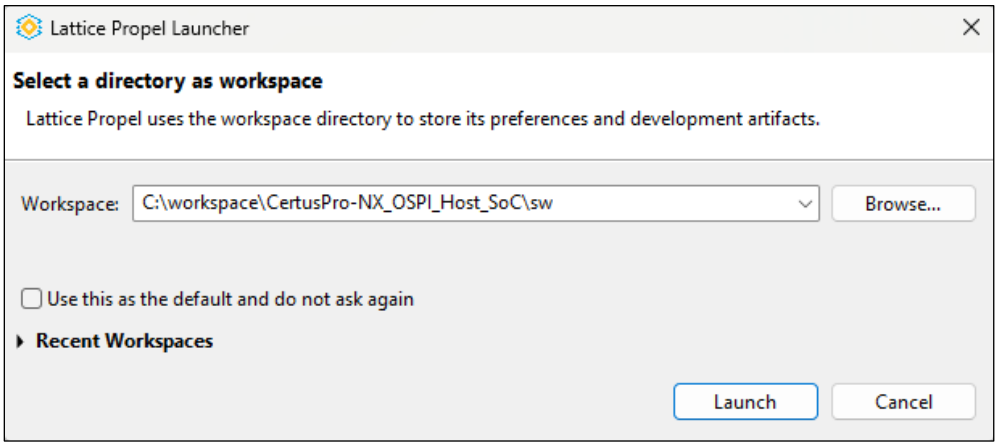

Figure 7.5. Propel SDK Workspace Setup

5. When ready, click **Launch**.

7.3.2. Building Target SPI x8 Configuration Software

The source code for the target SPI x8 configuration software is provided in the reference design. The following file is available in the *<my_work_dir>\sw* directory:

- *tspi_x8_config_sw.zip* – Contains the project and source code for the target SPI x8 configuration software.

To build the software project in Propel SDK:

1. In the Lattice Propel main interface, click **File > Import**.

2. On the Import wizard Select page, choose **Existing Projects into Workspace** under **General**.

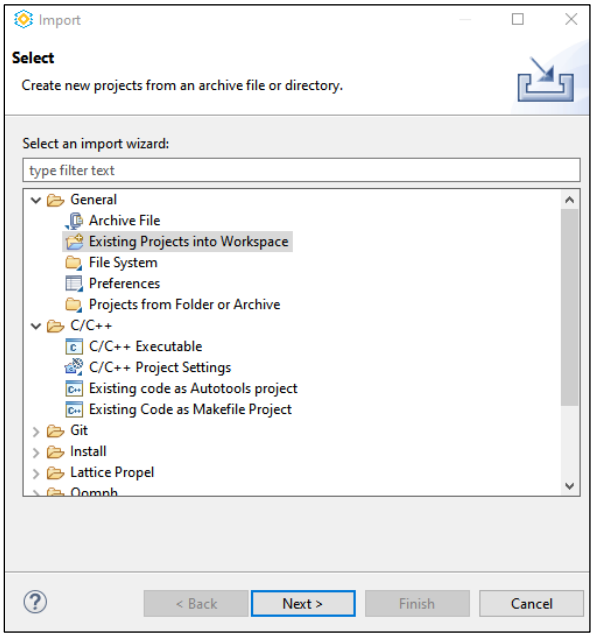

Figure 7.6. Select Existing Projects

3. Click **Next**.
4. On the Import Projects page, choose **Select archive file**.

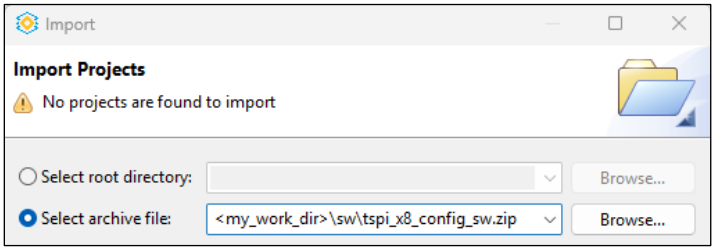

Figure 7.7. Import Projects

5. Click **Browse** and navigate to *<my_work_dir>\sw \tspi_x8_config_sw.zip*.
6. Click **Open**.
7. Click **Finish** to complete the process. The tspi_x8_config_sw project is listed in **Project Explorer**.

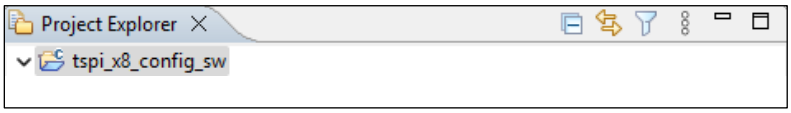

Figure 7.8. Project Explorer

8. In **Project Explorer**, right-click on tspi_x8_config_sw and select **Build Project**.
9. Verify that no error message is shown in the **Console** window.

10. Ensure that the *tspi_x8_config_sw.mem* file is generated in the *<my_work_dir>\propel_workspace\tspi_x8_config_sw\Debug* directory.

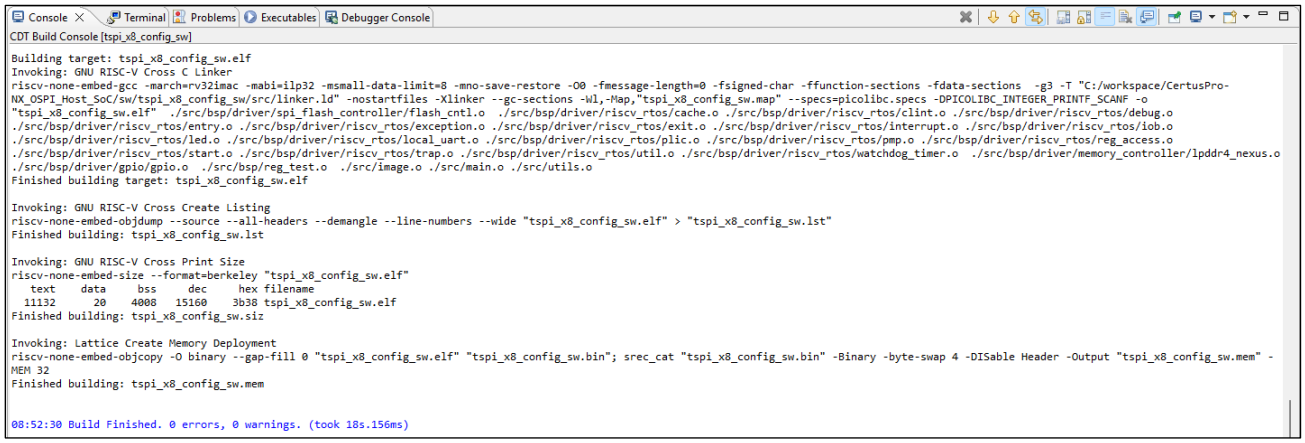

Figure 7.9. Build Project Console Output

7.4. Pre-initializing Tightly Coupled Memory

The tightly coupled memory can be pre-initialized with the target SPI x8 configuration software through FPGA configuration.

Note: By default, this reference design does not include pre-initialization of target SPI x8 configuration software in the tightly-coupled memory. The following steps are required when initializing the system memory with your own software program.

To initialize system memory:

1. In Propel Builder, double-click on the **tcm0_inst** block. This opens the Module/IP Block Wizard.

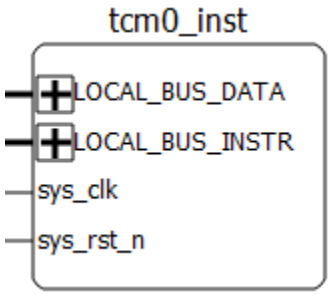

Figure 7.10. tcm0_inst Block

2. In the Module/IP Block Wizard, set **Memory Initialization** to **Memory File**.
3. In the **Memory File** field, click the **…** (browse) button.
4. In the Select File dialog box, navigate to the *<my_work_dir>\propel_workspace\tspi_x8_config_sw\Debug* directory.
5. Select *tspi_x8_config_sw.mem* and click **Open**.

6. Set **Memory File Format** to **hex**. Figure 7.11 shows the system memory IP settings.

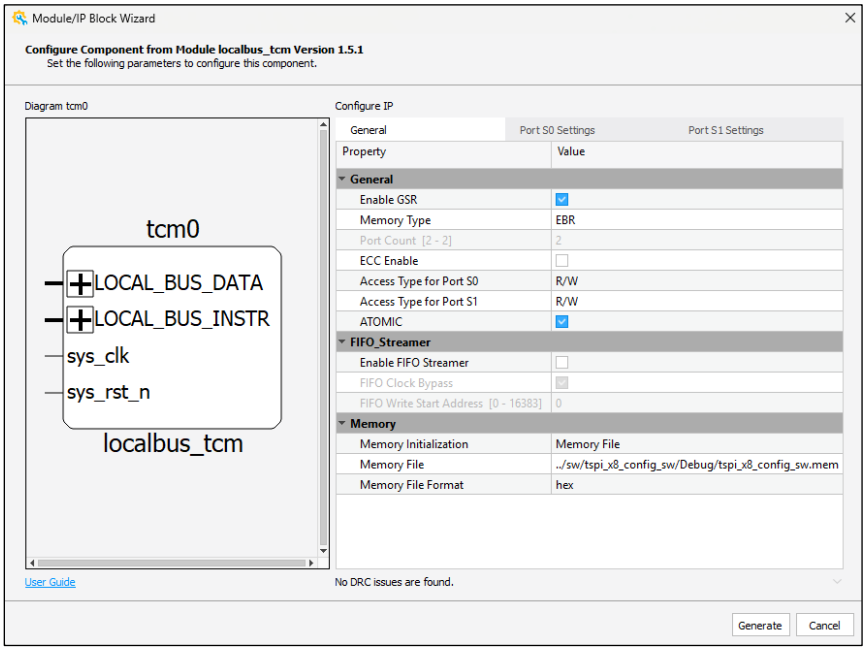

Figure 7.11. System Memory IP Settings to Pre-initialize with Bootloader Software

7. In the Module/IP Block Wizard, click **Generate** followed by **Finish**.
8. In Lattice Propel Builder, click **Design > Generate** to update the output.
9. In Lattice Radiant, click **Export Files** to generate the updated bitstream file.

Note: Refer to the following sections for the Propel Builder and Radiant compilation steps:

- Building and Generating the RISC-V Processor Using Lattice Propel Builder
- Synthesizing RTL Files and Generating the Bitstream File Using Lattice Radiant Software

8. Customizing the Reference Design

This section describes the customization that can be applied to the reference design. Hardware-related modifications are made using Propel Builder since it is the primary design entry tool. Software-related modifications are made using Propel SDK.

8.1. Changing LPDDR4 Memory Controller Parameters

The LPDDR4 memory controller IP parameters are set according to the DRAM chips available in the CertusPro-NX Versa Board. You can change the IP parameters to suit the board and DRAM chip in your project.

To modify the IP parameters:

1. In Propel Builder, double-click on the **lpddr4_inst** block.

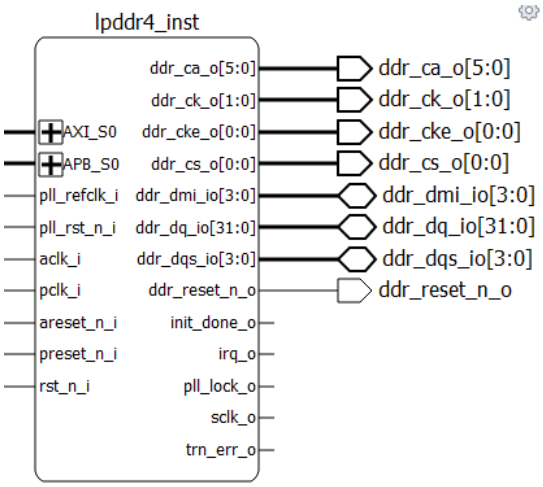

Figure 8.1. lpddr4_inst Block

2. In the Module/IP Block Wizard, change the parameters to suit your board and DRAM chip. Table 8.1 shows the possible customization.

Table 8.1. LPDDR4 Memory Controller IP Parameters Customization

Parameter	Description
DDR Command Frequency (MHz)	Change the frequency to the desired value that matches your design. For CertusPro-NX devices, the value is 533 MHz.
DDR Density	Change the density (in terms of Gb) per channel to match the DRAM chip. This affects the AXI4 address bus width of the IP interface.
DDR Bus Width	Change the bus width of the DDR data bus to match the DRAM chip. This affects the DQ, DQS, and DMI bus widths. This also requires updates to FPGA pin assignments.

3. In the Module/IP Block Wizard, click **Generate** to ensure the RTL is updated per customization.
4. In Propel Builder, regenerate your system.
5. In Lattice Radiant, click **Export Files** to recompile the design and update the bitstream.

8.2. Adding a New Component to the Propel Builder System

This reference design can be used as a base design upon which you can add a new component or IP as required by your project. You can perform this in Propel Builder using the schematic view.
Note: Propel Builder system here refers to the system you built and generated using Propel Builder.

The following sections describe the general design flows.

8.2.1. Hardware Flow

To add a new IP:

1. In Propel Builder, select and add the new IP from the IP Catalog window. Complete the IP configuration using the wizard and generate the IP.
   Note: If you need to create a custom IP, refer to the Lattice IP Packager 2025.1 User Guide (FPGA-UG-02236).
2. Connect the new IP to the RISC-V RX CPU or other IPs in the system. There are three primary interface types:
   - AXI4 or AXI4-Lite – Add a new Manager/Subordinate interface in axi_interconnect0_inst (depending on the interface type on the new IP). Connect the new IP to the newly added interface on the interconnect.
   - APB – Add a new Requestor/Completer interface in apb_interconnect0_inst. Connect the new IP to the newly added interface on the interconnect.
   - AHB-Lite – Add a new Manager/Subordinate interface in axi_interconnect0_inst. Add the AXI4 to AHB-Lite Bridge IP then connect the new IP to the newly added interface on the interconnect through this bridge IP.
3. Connect the clock and reset signals for the new IP.
4. Export the I/O ports from the new IP to the top-level module (if applicable).
5. In Propel Builder, assign the base address for the new IP through the **Address** view.
6. Click **Generate** in Propel Builder.
7. In Lattice Radiant, assign pins for the new IP (if applicable).
8. Click **Export Files** to generate the updated bitstream.

8.2.2. Software Flow

If the new IP contains a software driver, update the Board Support Package (BSP) in the software project.

To update the BSP:

1. In Propel SDK, right-click on the software project that you are working on and select **Update Lattice C/C++ Project**. In the Update System and BSP dialog box, you might observe a *Directory is not correct!* error as shown in Figure 8.2. This occurs when the software project is created in another PC with a different system environment path when the project is imported to SDK. If so, proceed to Step 2. Otherwise, skip to Step 3.

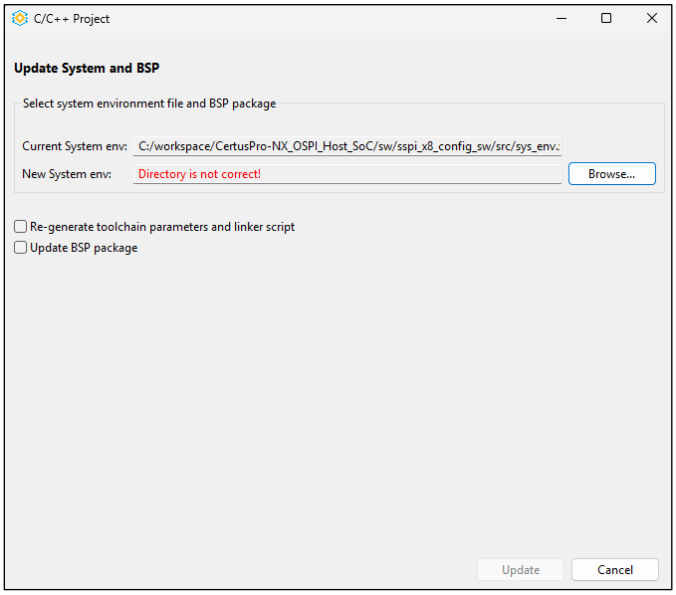

Figure 8.2. New System Error

2. Click **Browse** then navigate to the new system env path in your PC.

   For example: *<my_work_dir>/sge/sys_env.xml*

   Note: This step updates the software project to point to the Propel Builder system env file located in your PC. The correct path is displayed instead of the previous error.
3. Select the **Update BSP package** check box. This shows the new IP driver and version available for update.
   Note: The **Re-generate toolchain parameters and linker script** option is not required.
4. Click **Update** to complete the process.
5. Build the software project to obtain the updated executable files (.mem).

9. Known Issues

9.1. REFRESH Command Fails in SSPI x8 Mode when Device is Configured with Target SPI x8 Port Persisted

When the device is configured with SLAVE_SPI_PORT set to XSPI, sending a REFRESH command in target SPI x8 mode fails.

To reconfigure the device and resolve the issue:

1. Switch to x1 mode.
   Send the SSPI_MODE command 0x80 0x00 0x0F 0x00 in target SPI x8 mode to switch to x1 mode.
2. Reconfigure the device.
   Use the C source code provided in this reference design to perform the target SPI x8 configuration.

10. Resource Utilization

Table 10.1 shows the resource utilization using a Lattice CertusPro-NX device.

Note: The values shown are for reference purposes only. Actual usage may vary.

Table 10.1. Resource Utilization using LFCPNX-100-9LFG672C

Configuration	LUTs[1]	Registers[1]	EBRs[1]	I/O[1]
Processor-Based Controller for Avant Device Target SPI x8 Configuration reference design	34137/79872 (43%)	21206/80769 (26%)	84/208 (40%)	85/299 (28%)

Note:

1. Values indicate Utilization/Total with percentage utilization in parentheses.

References

- Lattice RISC-V Embedded Design Guidelines (FPGA-AN-02072)
- sysCLOCK PLL Design and User Guide for Nexus Platform (FPGA-TN-02095)
- Lattice Avant sysCONFIG User Guide (FPGA-TN-02299)
- Lattice Propel 2025.1 Builder User Guide (FPGA-UG-02235)
- Lattice Propel 2025.1 SDK User Guide (FPGA-UG-02234)
- Lattice IP Packager 2025.1 User Guide (FPGA-UG-02236)
- Revision Control – Lattice Propel Builder 2025.1 User Guide (FPGA-UG-02237)
- OSC Module – Lattice Radiant Software User Guide (FPGA-IPUG-02065)
- Processor-Based Controller for Avant Device Target SPI x8 Configuration Reference Design web page
- CertusPro-NX web page
- Avant-G web page
- Avant-X web page
- RISC-V RX CPU IP Core web page
- AXI4 Interconnect IP Core web page
- AXI4 to APB Bridge Module web page
- APB Interconnect IP Core web page
- AXI4 to AHB-Lite Bridge Module web page
- Memory Controller IP Core web page
- GPIO IP Core web page
- Tightly-Coupled Memory (TCM) IP Core web page
- SPI Flash Memory Controller IP Core web page
- CertusPro-NX Versa Board web page
- Avant-G Versa Board web page
- Avant-X Versa Board web page
- Lattice Radiant Software web page
- Lattice Propel Design Environment web page
- Lattice Solutions IP Cores web page
- Lattice Solutions Reference Designs web page
- Lattice Insights for Lattice Semiconductor training courses and learning plans

Technical Support Assistance

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

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

Revision History

Revision 1.0, August 2025

Section	Change Summary
All	Initial release.

www.latticesemi.com